Comparing CNN- and Transformer-Based Deep Learning Models for Semantic Segmentation of Remote Sensing Images

Master's thesis
in partial fulfillment of the requirements for the degree of
Master of Science

University of Münster
Institute for Geoinformatics

Supervisor:
Dr. Christian Knoth

Second supervisor:
Prof. Dr. Benjamin Risse

Submitted by:
Jan Stenkamp

Münster, January 2024

Comparing CNN- and Transformer-Based Deep Learning Models for Semantic Segmentation of Remote Sensing Images

---

Vergleich von CNN- und Transformer-basierten Deep Learning Modellen zur semantischen Segmentierung von Fernerkundungsbildern

Abstract

Semantic segmentation of remote sensing images has many useful areas of application like post-disaster management. Thus it is crucial to create meaningful segmentation results where deep learning architectures provide useful means. Mostly Convolutional Neural Networks (CNNs) are still state-of-the-art in computer vision, especially in semantic segmentation and in remote sensing although Transformers with their self-attention mechanism rapidly enter the field . To date, little research has been done on the application of pure Transformers segmenting remote sensing images, not to mention the comparison of Transformers and CNNs in this domain. In this work, U-Net and SegFormer as representatives of CNNs and Transformers are evaluated and compared to each other by training them on different data configurations. These include training data of varying patch sizes, a slim dataset and different spectral information. With all tested configurations, both architectures show good results for the segmentation of the ISPRS Potsdam dataset, with all models achieving accuracies above 75%. U-Net models have quantitative advantages, achieving higher IoU values, especially for classes covering smaller objects like trees. Predictions of the SegFormer models have qualitative advantages, creating more homogeneous and smooth results, whereas U-Net models tend to over-segmentation, also apparent from averagely higher values in the VoI metric. Moreover, the results improve with increasing patch sizes, in particular for SegFormer. The validity of the findings for other data was proven by successfully training and applying both architectures on the FloodNet dataset. The results indicate that the attention mechanism is advantageous for the SegFormer and other Transformers over CNNs. However further investigations are required to advance Transformers for semantic segmentation to the point where they outperform U-Net in all aspects.

Contents

List of Figures

List of Tables

1 Introduction

This chapter introduces the topic of CNN- and Transformer-based deep learning architectures and why it is of interest to compare them for semantic segmentation of remote sensing images. Moreover, the contribution, approach and outline of this work are presented.

1.1 Background and Motivation

Semantic segmentation is the task of assigning a class to each pixel within an image and plays a crucial role in remote sensing, for example in determining different land cover types or detecting damages after a natural disaster. In these tasks, deep learning led to significant increases in the accuracy of segmentations, with Convolutional Neural Networks (CNNs) as a leading technique. Here U-Net [1], actually designed for biomedical image segmentation, is one example of many different architectures that have been applied very effectively to segment remote sensing images [2].

As a new method in deep learning, Vaswani *et al.* [3] introduced the Transformer, a network architecture only using attention mechanisms for natural language processing. Attention mechanisms were also used to support other architectures in computer vision tasks until Dosovitskiy *et al.* [4] were the first to use a pure Transformer for image classification, the Vision Transformer (ViT). With some effort, the ViT can also be utilized for semantic segmentation. Similarly, other Transformers that have been designed as a general purpose backbone for different vision tasks like Swin Transformer [5] can be used as well as Transformers specifically developed for the task of semantic segmentation, with Segmenter [6] as one example.

The attention mechanism of Transformers is also what makes it very interesting for remote sensing tasks, as it is capable of capturing long-range dependencies within images, other than CNNs, which only look at local interactions [7].

1.2 Research Gap

So far in remote sensing, the attention mechanism was mostly used within hybrid architectures instead of applying pure Transformers [7]. Often a CNN extracting the local features is supplemented by attention layers incorporating global context (e.g. WiCoNet [8]). Research is also mostly focused on Transformers for classification tasks and less for semantic segmentation. Most research for semantic segmentation is done in the medical imaging domain, which is also the case for the comparison between different deep learning architectures. While working on this thesis, Sourget *et al.* [9] published an interesting paper asking "Can SegFormer be a True Competitor for U-Net for Medical Image Segmentation?". This question is similarly chosen for this work but transferred to the remote sensing domain. Moreover, the evaluation of deep learning architectures is often concentrated on the performance in terms of accuracy measures but often other aspects are relevant too. In remote sensing, for example, images are often too large to be processed as a whole, so the selection of proper patch sizes is crucial.

1.3 Contribution

To shed light on the mentioned gaps in the current research, the following guiding question is addressed in this thesis:

- What are the differences between CNN- and Transformer-based models for semantic segmentation of remote sensing images?

To answer this question different aspects that influence the predictions are evaluated. The architectures are compared regarding the impact of different patch sizes and the amount of training data. Furthermore, the generalizability of the results is investigated. Based on the given literature, my hypotheses include:

1. Pure Transformers are suitable for semantic segmentation of remote sensing imagery.

2. Transformers perform better, especially when applied on larger patches and segmenting larger areas.

3. Transformers need more data to be trained sufficiently.

1.4 Approach

First and foremost representative architectures for CNN and Transformer are implemented and compared for the application on remote sensing data. For each architecture, models with different configurations in terms of training data are realized to evaluate the reasons for differences in the performances. The models are compared qualitatively along exemplary predictions and quantitatively by measures for the similarity between prediction and ground truth. Besides the evaluation of the models altogether, special attention is paid to differences between semantic classes. Another aspect includes the generalizability of the results. This is examined by evaluating the performance of a trained model when applied to a dataset with other properties, and by training models on yet another dataset with different semantic classes.

1.5 Outline

The thesis is structured into five further chapters. First, fundamentals like deep learning basics, CNN, Transformer and semantic segmentation are described. Secondly, the applied methodology is explained, including descriptions of the implemented architectures and the used datasets. In the subsequent chapter, the qualitative and quantitative results are illustrated. The penultimate chapter covers the discussion, addressing the comparison of the models to answer the guiding question and bringing up the limitations of the work. The last chapter concludes the thesis by summarizing the findings and giving an outlook for future work topics. Multiple appendix chapters comprise further qualitative and quantitative results without being evaluated.

2 Fundamentals

In this chapter basic concepts of deep learning are presented together with details about CNNs and Transformers. Afterwards, semantic segmentation and its applicability are described. Moreover, short insights into the training of deep learning models and the evaluation of semantic segmentation results are given.

2.1 Deep Learning

Deep Learning (DL) is a subdomain of Machine Learning (ML) that in turn is a subdomain of Artificial Intelligence (AI). Whereas AI also includes for example knowledge-based systems, ML is the approach to let computers develop solutions for a task without every step of the calculation being programmed by a human. It is a technique that allows computer systems to improve with experience and data [10]. DL as a type of ML uses multiple (hidden) layers to deliver solutions to more difficult and complex tasks by learning to represent the world as a nested hierarchy of concepts.

2.1.1 Neural Network Basics

Artificial Neural Networks (ANNs) are inspired by biological systems. The perceptron introduced by Rosenblatt [11] for example, should mimic a neuron in the brain and is still the basis for modern ANNs. ANNs consist of units that are connected via links, with a link from unit i to unit j propagating the activation a_i from i to j. Each link has a weight $w_{i,j}$, determining the strength of the connection [12]. Each unit j computes a weighted sum of its inputs and then it applies an activation function g to this sum to derive the output:

$$a_j = g(\sum_{i=0}^{n} w_{i,j} a_i) \tag{2.1}$$

The activation function g is of a nonlinear type, ensuring the important property of the whole network to represent a nonlinear function.

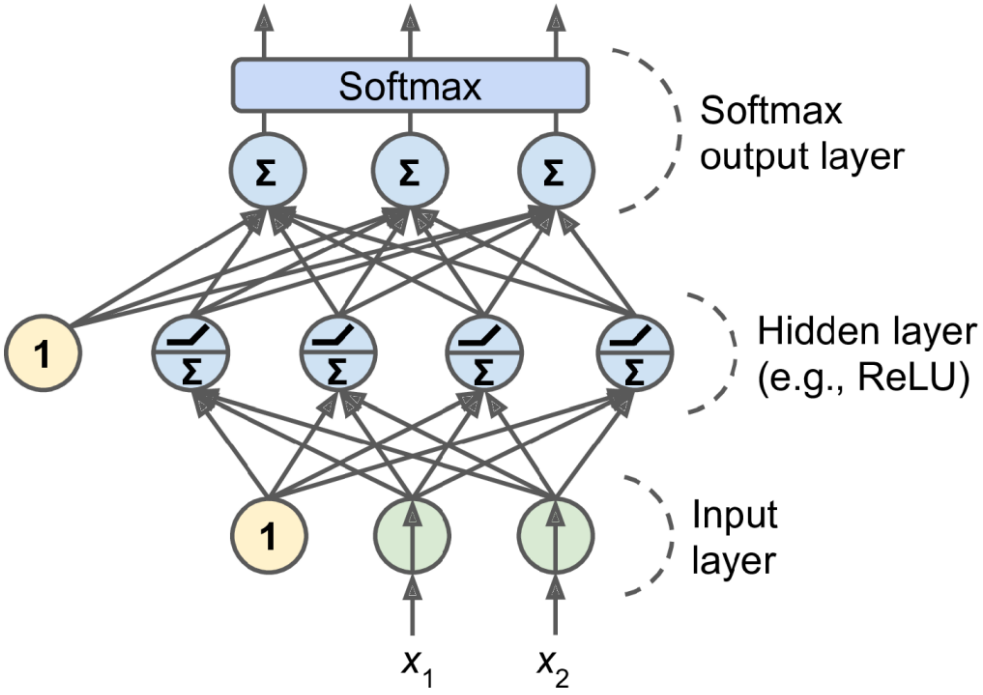


Figure 2.1: Example of a MLP with one hidden layer that uses ReLU for activation and softmax for the output layer [13].

Multilayer Perceptron

Multilayer Perceptrons (MLP), also called feedforward neural networks [14] have one or more layers of hidden units between their input and output layers. A feedforward network has only connections in one direction thus each unit only receives inputs from previous layers. Information flows from the input to the output without feedback connections which would feed the outputs of the model back into itself. The layers can also be seen as functions that are connected in a chain with each other, for example with a function $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$ where f describes the whole network, $f^{(1)}$ the first layer, $f^{(2)}$ the second layer and so on. When the units in a layer are connected to every unit in the previous layer, the layer is called fully connected. A network is fully connected when all its layers are fully connected. Figure 2.1 depicts a fully connected MLP with two input values, one hidden layer consisting of four hidden units and an output layer with three values. The hidden units can be understood as a single perceptron (equation 2.1) where the arrows are the incoming activations a that are processed by a weighted sum and then ReLU (rectified linear unit) as activation function g is applied.

2.1.2 Convolutional Neural Networks

A form of MLP that is particularly well suited to process 2D signals like images, is the Convolutional Neural Network (CNN) [15]. When processing images with neural networks the pixels of the images are usually used as input (cf. x_i in figure 2.1) and the output depends on the task of the network (cf. section 2.2).

Using fully connected feedforward networks for image recognition would connect all pixels to one another, producing huge networks by applying filters that span the whole image size and connecting each input unit with each output unit through all layers [10]. Alternatively, a CNN uses structured sparsity by applying filters with a kernel size smaller than the input image, allowing fewer parameters in an even deeper network, making it more efficient [16]. Moreover, a CNN can detect a local pattern more easily as the fully connected network would evaluate it in the context of the whole image and thus might oversee it. Although the filters of a CNN only look at smaller areas of the image at a time, deep CNNs might nevertheless interact with larger areas of the input indirectly, detecting different motifs and objects at different stages of a network [17].

Usually, a CNN has one or multiple convolutional layers, each consisting of three stages:

1. Convolution: Applying filters to create feature maps.

2. Activation: Nonlinear activation function.

3. Pooling: Downsampling by summarizing neighboring features.

After the input is processed by the convolutional layers the respective results are passed to a fully connected layer to get the actual classification results or they are upsampled for dense prediction tasks. In figure 2.2 all filters and nodes of a convolutional network for the classification of digits are visualized. The mentioned steps will be described in more detail in the following.

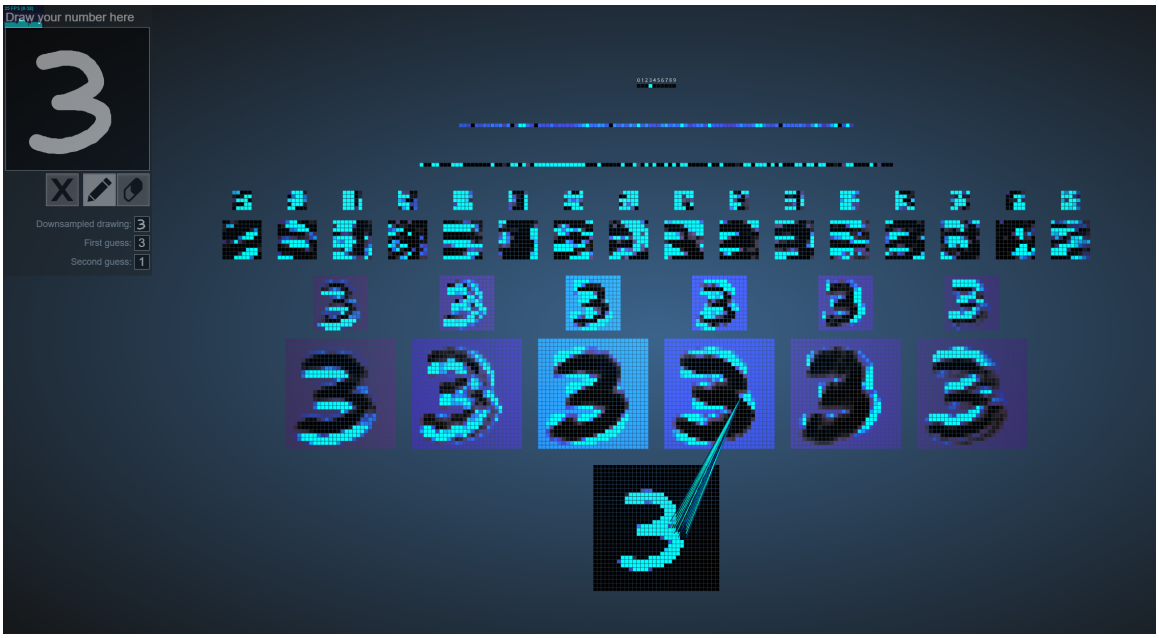


Figure 2.2: Visualization of a convolutional network for classification of digits. The network has two hidden layers, each applying convolution (kernel size 5, stride 1) and max pooling (kernel size 2) and then three fully connected layers [18].

Convolution Operation

In the convolution stage filters are applied to the input. Each filter creates a feature map as output and sliding over the input image, the convolution (cf. equation 2.2) between the filter values, i.e. the weight matrix, and the input values is calculated at each location [19]. Hyperparameters that control the convolution are:

- Depth k: The number of filters to be applied, where each should learn something different in the input.
- Kernel size f: The spatial extent of the filter.
- Stride s: How many pixels is the filter moved each time; a stride larger than 1 will reduce the output size.
- Zero-padding p: A padding of zeros around the input can help to keep the output size equal to the input size; the parameter gives the size of the padding.

Figure 2.3 gives an example of a filter with the hyperparameters $f = 3$, $s = 1$ and $p = 1$, preserving the dimensions $[5 \times 7]$ of the input. The formula to get the dimensions of the output per axis is $(i - f + 2p)/s + 1$, where i is the size of the input axis. Thus setting zero-padding to $p = (f - 1)/2$ when the stride is $s = 1$ makes sure that the input and output have the same size. Also, it must be

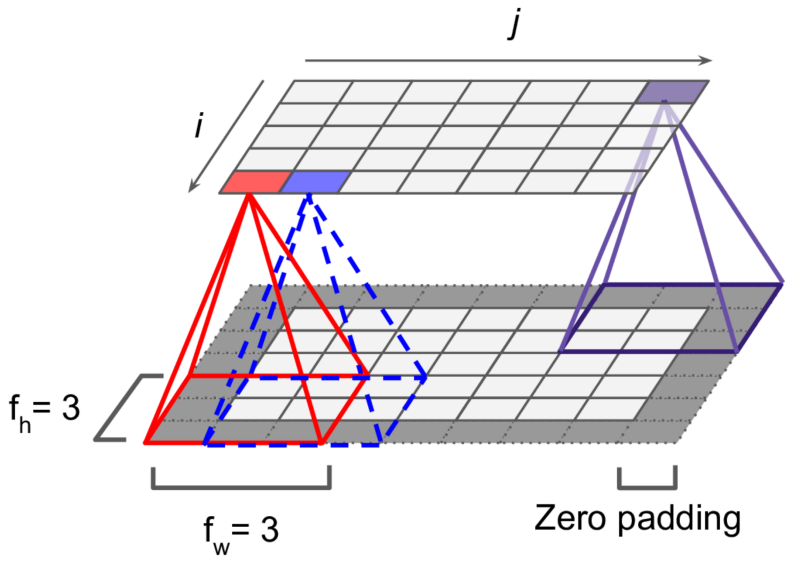


Figure 2.3: Convolution operation with $p = 1$ and $s = 1$, preserving the dimensions of the input $[i \times j]$ in the resulting feature map [13].

taken into account that the result of the mentioned formula needs to be an integer by choosing appropriate parameters.

An example of a convolution layer with $f = 11$, $s = 4$ and $p = 0$ and a depth of $k = 96$ applied to input images of size $[227 \times 227 \times 3]$ (e.g. RGB image with three channels) results in an output of size $[55 \times 55 \times 96]$, as $(227 - 11)/4 + 1 = 55$. Thus each of the $55 \times 55 \times 96$ neurons of the output is connected to a $[11 \times 11 \times 3]$ region in the input [20].

Mathematically the discrete convolution between an image I and a kernel K is described as a summation of finite numbers of array elements [10]. The result of the operation at image position (i, j) with kernel dimension (f_h, f_w) is:

$$S(i,j) = (I * K)(i,j) = \sum_{m=0}^{f_h} \sum_{n=0}^{f_w} I(i-m, j-n)K(m,n) \tag{2.2}$$

Within a feature map, all values are created by the same weights applied to the input, which is called weight sharing. At the same time, all k filters within a layer use different weights to let every filter learn different things. These characteristics are useful for detecting the spatially invariant local motifs in images, as the same pattern can appear in different parts of an image [17].

Nonlinear Activation

Since the convolution is a linear operation, with element-wise matrix multiplication and addition, but most data cannot be described linearly, a nonlinear activation function is applied after the convolution step. The most used function in deep neural networks is ReLU, which is computationally efficient and simple [16]. It is applied to each element in the feature map and is described as $f(x) = max(x, 0)$, thus all positive values are kept, and all negative values are set to 0.

Pooling Operation

The pooling stage reduces the dimensionality of the feature maps and is also called downsampling. The hyperparameters of the pooling operation are:

- Kernel size f: Spatial extent the pooling is applied to.
- Stride s: Downsampling factor.

Usually, the hyperparameters are set to $s = 2$ and $f = 2$ or $f = 3$. Figure 2.4 shows max pooling with $f = 2$, halving the dimensions of the input. Setting $f = 3$ would apply an overlapping pooling, as the filter would overlap by one row and/or column at each step.

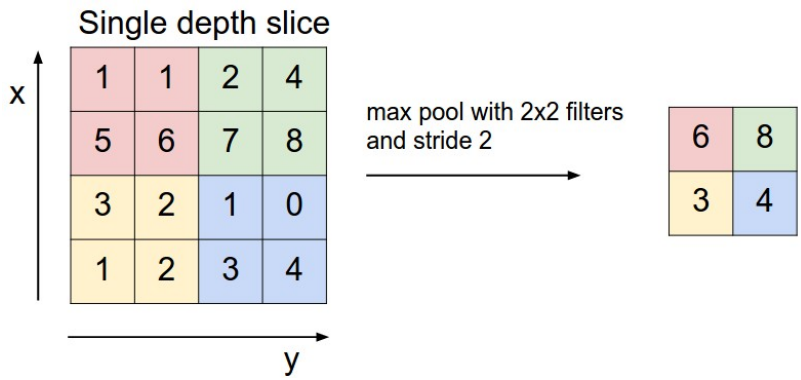


Figure 2.4: Max pool operation with $s = 2$ and $f = 2$ for 4×4 matrix [20].

In most cases, max pooling is used, where the maximum of the region covered by the filter is computed. Alternatively, the average or sum could be calculated but the maximum has proven to work best [16]. Applying one of these functions, pooling merges semantically similar features into one, making the network more resilient against variances in the input [17].

Decoding

In image recognition tasks first one or multiple convolutional layers, each consisting of convolution, activation and pooling stages, are applied to the input. Afterwards, a fully connected layer takes the feature maps of the last convolutional layer as input to calculate scores for all classes. As part of this last step, usually softmax is applied, to calculate the probabilities for the classes as the final result.

To conduct a segmentation and not a classification, instead of translating the coarse feature maps resulting from the convolutional layers into classes, an upsampling needs to be conducted. Hence each pixel of the input can be assigned to a class instead of the whole image. A common way to do this is by using one or more transposed convolutional layers. It is also called deconvolution (although it does not correspond to the mathematical term of a deconvolution) or fractionally strided convolution, as a transposed convolution with stride s can be thought of as a usual convolution with stride $1/s$ [21]. So the stride in a transposed convolution describes how much the input is stretched instead of how large the steps of the kernel are.

Such convolutional architectures can be separated into encoder and decoder. The encoder maps the input to a feature space like the coarse feature maps and the decoder takes these feature maps as its input and processes them to produce an output, for example into finer feature maps that can describe a segmentation.

To recover more of the spatial resolution lost in the pooling stages in the upsampling, skip connections are useful [13]. These can be used to skip information of feature maps with higher resolutions from a lower layer to a higher layer without the need to be processed by every intermediate layer. In this way, it can also pass information from intermediate encoder stages to corresponding decoder stages.

2.1.3 Transformers

Transformers are a type of neural networks that entirely depend on the self-attention mechanism. At first, they were developed by Vaswani *et al.* [3] for natural language processing tasks. In the last years, Transformers also became established in vision tasks, with the first one developed by Dosovitskiy *et al.* [4] in 2020 for image recognition, followed by architectures adapted for tasks like object detection, segmentation and others [22].

In this section, the specifics of using Transformers in vision will first be outlined along with the example of the basic Vision Transformer (ViT) [4] to show the fundamental idea of Transformers. Secondly, the attention layer, underlying all Transformers, will be explained in general and with a focus on multihead self-attention.

Vision Transformer

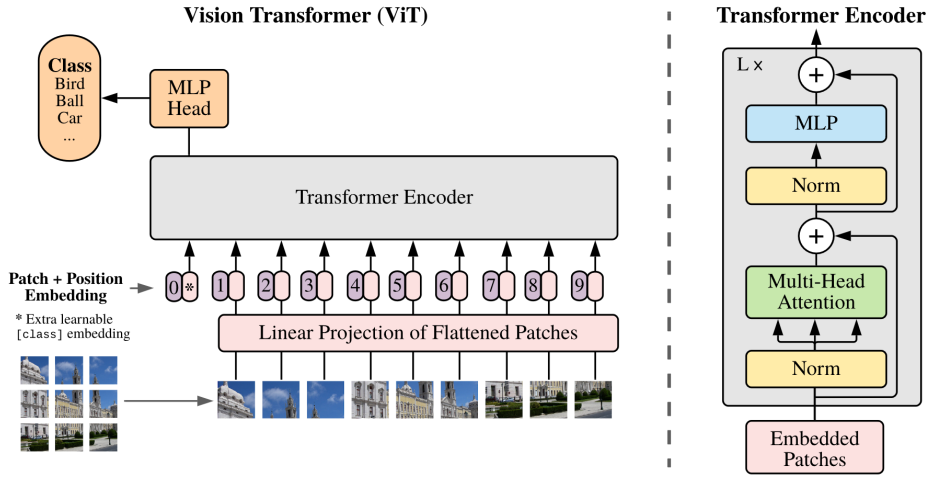


Figure 2.5: Model overview of the Vision Transformer by Dosovitskiy *et al.* [4].

Detecting long-range dependencies between elements of the input sequence is one of the strengths of the attention mechanism, that can be very useful in vision tasks. Applying Transformers in vision tasks, the attention scores between every pixel within an image could be calculated but as this would be computationally very expensive, instead ViT calculates the attention between 16×16 patches. An overview of the ViT structure is depicted in figure 2.5.

ViT splits an input image into patches of the same size, flattens the channels and uses a trainable linear projection to create the patch embeddings. An additional learnable embedding is added to the patch embeddings containing the image representation, i.e. the class of the image. Moreover, a 1D position embedding is added to each patch embedding, since self-attention itself does not incorporate

the position of the inputs but the order of the patches is an essential information in an image. The resulting embeddings are used as the input for the encoder, which is a standard Transformer encoder, as introduced by Vaswani *et al.* [3] and also used in natural language processing tasks.

Two sublayers are part of the encoder, the multi-head attention layer and the MLP layer. The MLP as a fully connected feedforward network is applied after the attention layer and serves to activate the outputs in a non-linear way. Both layers are preceded by the application of a layer normalization [23] that can help to reduce the training time and stabilize the network. Also, both layers are followed by a residual connection [24] for identity mapping, passing the unprocessed embeddings to the next layer to not distort relevant information.

To get the class probabilities as a final result, the output of the Transformer block is passed to an MLP head as a decoder. This enables an easy fine-tuning of the head for different classification tasks but as a downside, the encoder needs to be trained on a very large dataset to achieve competitive results.

Self-Attention

The aim of the self-attention mechanism is to apprehend the interaction between all the input elements In the case of ViT, these are the image patches. To be processed further, the image patches are converted into a sequence of patch embeddings. For each embedding in the sequence, three representations are created: query, key and value. First, the dot product of the query of an embedding with the keys of all other embeddings is computed. This is divided by the square root of the dimension of the keys $\sqrt{d_k}$ to counter vanishing gradients. The result is normalized by softmax to obtain the weights on the values that express the interaction between the elements.

For an easier computation of all the attentions simultaneously, each query Q, key K and value V for all embeddings are put together into matrices. This is achieved by projecting the sequence of embeddings onto the learnable weight matrices W^Q, W^K and W^V. The scaled dot-product attention [3] is depicted in figure 2.6 and the matrix of outputs is calculated as:

$$Attention(Q, K, V) = softmax(\frac{Q * K^T}{\sqrt{d_k}})V \tag{2.3}$$

Multi-Head Attention

To let a model learn more than one attention representation, oftentimes multi-head attention is applied. Mutli-head attention means that instead of one attention layer multiple layers are used

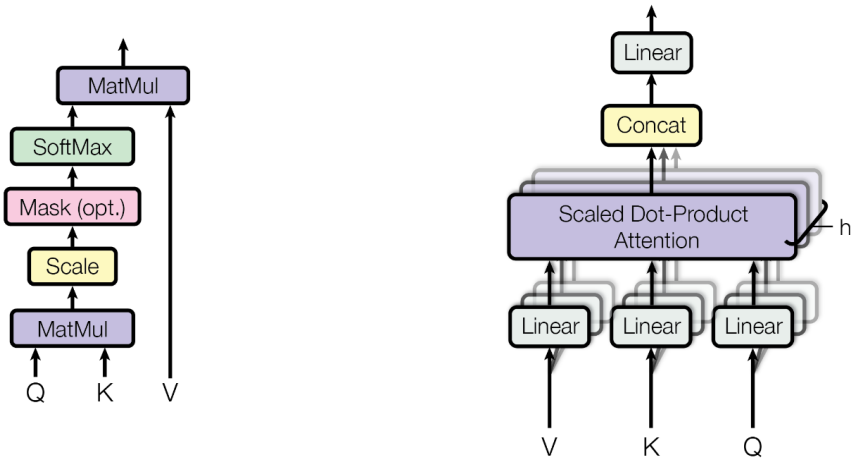


Figure 2.6: (left) Scaled dot-product attention. (right) Multi-head attention [3].

in parallel, each with its own randomly initialized query, key and value matrices. With multiple heads the model can learn different representations of the input vectors and focus on different aspects, increasing the expressivity of the model, comparable to multiple filters that are learned by a CNN. The results of the multiple heads need to be condensed into one result to be processed by subsequent steps. To achieve this the attention heads are concatenated and then multiplied with a weight matrix that is trained together with the model. Figure 2.6 shows how multi-head attention utilizes h scaled dot-product attention heads. The box "Linear" in the figure corresponds to the learnable linear projections of the inputs that are equivalent to the weight matrices W^Q, W^K and W^V mentioned above.

Differences to CNNs

Given the properties of the deep learning architectures described in the previous chapters, there are some considerable differences between CNNs and Transformers, that will be described in this section.

The main difference between CNN and Transformer architectures is the convolutional operation compared to self-attention. Self-attention allows a Transformer to capture long-range and global dependencies as it learns the relationships between all elements of a sequence, i.e. applied to images the relation between their patches, whereas the limited kernel size of convolutional filters only allows capturing local interactions. After multiple layers a CNN can also capture dependencies within larger areas, as pooling operations increase the receptive field and thus lower the distances between features, but self-attention already allows this in lower layers with a higher resolution [25]. Also, convolutional networks do not encode the relative positions of extracted features, which Transformers can do, as long as a positional encoding is implemented within the architecture.

These disadvantages of convolutional operations could be tackled by largely increasing the kernel size of the filter, but this in turn would drop the efficiency of the local convolutions. Moreover, due to the weight sharing, which is one of the properties making the convolution effective in the first place, the CNNs learn static filters. Transformers, on the other hand, can dynamically calculate filters in self-attention, making them more flexible to occlusions and perturbations [26].

Cordonnier *et al.* [27] examined the relationship between self-attention and convolution layers and found that - applied to images - a multi-head self-attention layer with a sufficient number of heads can express any convolutional layer, and in practice, they often learn to do so. But to do so, more data is needed to train a vanilla ViT from scratch, as CNN operations make use of image-specific inductive biases like translational equivariance that Transformers (depending on the architecture) need to learn first, what they need more data for.

2.1.4 Training DL Models

The aim of training a deep learning model is to optimize its weights in such a way that it describes the data it is trained on in the best possible way. This is achieved by minimizing a chosen loss function that describes the fit between the ground truth of the training data and the corresponding prediction of the model.

Loss Function

The loss function evaluates the model during supervised training by calculating the model error. This model error is used to adjust the weights through backpropagation. Thus it helps the model in reaching the ideal combination of weights to map the input data to the output, i.e. the images to corresponding labels in image classification or a pixel to a semantic class in segmentation.

A simple and widely used function is the cross-entropy loss measuring the difference between two probability distributions. In segmentation, it evaluates the class predictions independently for each pixel with

$$CE_{loss}(r, q) = -\sum_{i=1}^{n} r_i log(q_i) \tag{2.4}$$

where r_i is the true probability for the i^{th} class and q_i is the predicted probability of the same class [28]. The resulting value per pixel is averaged over all pixels to receive the actual loss. Since cross-entropy loss focuses on pixel error, it favors the model to focus on classes that are represented

by more entities in the dataset. With modifications like weighted cross-entropy there are options to tackle this issue [29].

Another loss function well suited for semantic segmentation is the Jaccard loss. It is derived from the IoU which is also called the Jaccard index. It is calculated for two sets X and Y as the ratio between the number of instances in the intersection of both sets to the number of instances in the union of both sets:

$$IoU(X, Y) = \frac{|X \cap Y|}{|X \cup Y|} = \frac{|X \cap Y|}{|X| + |Y| - |X \cap Y|} \tag{2.5}$$

Applied to predictions of a model, X would be the ground truth and Y the prediction, thus in the value range of $[0, 1]$, a value of 1 for the IoU would imply that all predictions exactly match the ground truth. Translating IoU to a loss function the formula for Jaccard loss is:

$$Jaccard_{loss}(g, p) = 1 - \frac{\sum_{i=1}^{n} g_i p_i + 1}{\sum_{i=1}^{n} g_i + \sum_{i=1}^{n} p_i - \sum_{i=1}^{n} g_i p_i + 1} \tag{2.6}$$

where g describes the ground truth and p the prediction segmentations calculated over the number of pixels n. 1 is added as a smooth parameter to the numerator and denominator to avoid division by zero. As the weights of the model are optimized by decreasing the value of the loss function during training, the loss value is calculated by subtraction from 1, i.e. the model would be ideal on the training data with $Jaccard_{loss} = 0$.

Optimization

During model training, the goal is to minimize the value of the loss function by adjusting the model weights accordingly. Gradient descent is the basic method applied to change the weights. Variations of the algorithm include the widely used Adam (Adaptive Moment Estimation) [30]. The general concept of gradient descent is to iteratively find the local minimum of a differentiable function, in the case of model training the minimum of the loss function. This is done by calculating the gradient of the loss function with the current weights and going in the opposite direction of the gradient, i.e. along the steepest descent and thus in the direction of the next minimum.

To calculate the gradients in a network, backpropagation [31] is used. In a feedforward neural network, the input is passed through the layers by a forward pass. Then from the scores at the output stage, the model error is calculated by the loss function. These errors are propagated backward through the network to calculate the gradient of the loss function concerning the parameters [10], i.e. the contribution of each weight to the error. To make a step towards the minimum of the loss

function along the negative gradient, a fraction of the gradients is subtracted from the previous weights. The value that determines the step size that is taken, i.e. the mentioned fraction, is also called the learning rate, as it determines how fast the model adapts the weights. A large learning rate lets the model train faster but these large steps might skip the optimal set of weights which might result in an oscillating performance of the model.

To summarize, backpropagation is the method to compute the gradient, and gradient descent (or another algorithm) makes use of the gradient to perform the learning, i.e. it optimizes by adjusting the weights [10].

2.2 Semantic Segmentation

Different computer vision tasks, from coarser to finer object recognition, include image classification, object detection/localization, semantic segmentation and instance segmentation [32]. Image classification assigns a class from a set of given labels to an image as a whole. Object detection not only evaluates the class of the image but also finds the spatial location of multiple classes in the image usually by assigning bounding boxes or determining the centroids of the objects. Semantic segmentation labels each pixel in an image with the category it most likely belongs to but does not make a difference between different instances of the same class. This is done by instance segmentation, where each pixel is assigned to a class and instances are differentiated, i.e. different objects of the same category get different labels whereas they would get the same label with semantic segmentation.

There are multiple useful applications for semantic segmentation in computer vision. In medical imaging, it is applied to different kinds of images like X-rays or MRI images, for example to detect tumors, whereas some applications also require the evaluation of 3D data. Autonomous driving is another area where segmentation plays a crucial role in understanding complex street scenarios and thus can increase the precision and safety of road traffic. In remote sensing also multiple applications exist for semantic segmentation, which will be explained in more detail in the next section.

Semantic segmentation has a history before the deep learning era with one of the simplest methods using a threshold on grayscale images like X-ray CT scans to label pixels [33]. The segmentation process is not only the classification of single pixels but as neighboring pixels are strongly correlated, they can be considered together. Traditional segmentation algorithms are based on clustering. For example, pixels with similar spectral properties that are located in a spatial proximity are clustered together. Then a label can be assigned to these groups.

In deep learning, semantic segmentation architectures are often built upon networks designed for classification tasks replacing their fully connected layer that only produces a score for the class of the whole image with a segmentation layer. One of the first approaches to do so was the Fully Convolutional Network (FCN) [21], replacing the fully connected layers of established recognition networks with 1×1 convolutional layers. Additionally, they add a deconvolution layer for upsampling the coarse output of the 1×1 convolution layers to pixel-dense outputs. Since the final layer needs to be upsampled by a factor of 32 to get to the original resolution the result is still fairly coarse, thus skip connections to lower layers with finer strides are included to add more detail to the final segmentation. Using a deconvolution layer in the decoder for upsampling also allows for learning the upsampling layer, which would not be the case for a simple interpolation. Other segmentation architectures even use stacked deconvolutions combined with activation functions, enabling the decoder to learn a nonlinear upsampling. An example of such an architecture is U-Net [1], which also adds skip-connections between encoder and decoder (cf. section 3.1).

The limited receptive field, as one of the drawbacks of a convolutional network, also applies to semantic segmentation. To tackle this and capture long-range dependencies in the images, attention was incorporated into various architectures and pure Transformer architectures for semantic segmentation were developed [34]. The aforementioned ViT [4] was designed for image classification but as the first pure Transformer for vision, it inspired the design of architectures for other tasks. One of these examples for semantic segmentation is Segmenter [6] which uses the ViT backbone and a mask Transformer as a decoder. Thus it is able to use a backbone trained on a large classification dataset like ImageNet and can fine-tune the decoder on a medium-sized dataset for good results. Another Transformer for semantic segmentation is SegFormer [35] (cf. section 3.2).

2.2.1 Segmentation in Remote Sensing

Remote sensing data contains complex semantic information which allows it to be used for multiple application areas. Often semantic segmentation is used to detect land use classes like agricultural, forest, urban [36] and/or objects like cars, trees, and buildings [37] within aerial images. With more specific use cases and more detailed classes, it is also applied for instance to detect which crop is cultivated on a field [38] or which tree species grow in a forest [39]. Such segmentations can then be compared for different time steps to perform change detections like deforestation, sea level rise or post-disaster damage assessment.

To approach these applications the deep learning and segmentation architectures mentioned in previous sections can often be used out of the box for images with three channels as they are mostly designed for such data [2]. But in many remote sensing domains, it is also useful to incorporate other bands than the visible red, green and blue (RGB). To analyze the vitality of vegetation, for example,

the near-infrared (NIR) band is very meaningful, often replacing the blue band in 3-channel images. Such images are often depicted as false-color images with infrared shown as red band resulting in IRRG-images. But beyond that, more than three channels can be used, requiring adaptations to the established architectures [40]. Hyperspectral images containing tens or hundreds of channels with different spectral information can not be incorporated into most deep learning architectures by default as they are designed for RGB patches and not massive data cubes. Moreover processing data gathered by active sensors like SAR and Lidar leads to further challenges. Especially Lidar data which are point clouds and not images are not researched that well [41]. Even within the passive collection of remote sensing data like RGB images, the fact that these images are taken by varying sensors and from varying distances already extends their diversity. They highly differ in ground sampling distance and objects on the ground have different distortions, depending on the height and angle the sensor has in relation to them. Other specifics to remote sensing images include clouds and shadows which are two distinct topics on their own but are also interrelated as clouds produce shadows, that need to be handled differently than shadows from objects on the ground [42]. Also, remote sensing data instances can be very large, with images or point clouds containing millions of data points. In many cases, these need to be subsetted to be processed, which especially in the case of semantic segmentation can remove crucial information to recognize certain classes.

As it is costly to create datasets for semantic segmentation, often there are no labeled datasets available for the diverse applications and data formats in remote sensing. Examples of openly available labeled datasets are the Inria dataset for building detection based on RGB data [43], ISPRS Urban Modelling and Semantic Labelling Benchmark datasets for 3D building reconstruction including Lidar data [44] and ISPRS 2D semantic labeling with four bands (cf. section 3.3.1). Another interesting source for labeled segmentation data is SpaceNet[1], providing satellite images for challenges of feature extraction and characterization, mainly of roads and buildings. For example in the SpaceNet 6 challenge, a multi-sensor all-weather mapping dataset was provided, including optical and SAR data for building extraction [45].

2.2.2 Metrics

Different metrics can be used to evaluate the performance of semantic segmentation models. In the following equations, k is the number of semantic classes, and p_{ij} is the number of pixels of class i inferred to belong to class j. Thus p_{ii} represents the true positives, p_{ij} is interpreted as the number of false positives and p_{ji} as the number of false negatives.

[1] https://spacenet.ai/

Accuracy

The simplest metric is pixel accuracy, measuring the proportion of correct pixels to the total number of pixels:

$$PA = \frac{\sum_{i=1}^{k} p_{ii}}{\sum_{i=1}^{k} \sum_{j=1}^{k} p_{ij}} \tag{2.7}$$

The downside of global accuracy is that it does not consider imbalanced datasets that have classes that are much more frequent than others. In such scenarios, a model that predicts the frequent classes well but does not detect smaller classes at all might be better regarding accuracy than a model that can predict all classes equally well but predicts fewer of the frequent classes. Mostly this is not desired in semantic segmentation as all classes that are labeled in the data are also of interest for the result.

Mean Intersection over Union

Mean Intersection over Union (mIoU) is a metric widely used in semantic segmentation evaluation because of its representativeness and simplicity [32]. It computes the mean of the IoU (cf. equation 2.5) and thus weights all classes equally in the evaluation of the overall model. It is calculated as follows:

$$mIoU = \frac{1}{k} \sum_{i=1}^{k} \frac{p_{ii}}{\sum_{j=1}^{k} p_{ij} + \sum_{j=1}^{k} p_{ij} - p_{ii}} \tag{2.8}$$

Variation of Information

The Variation of Information (VoI) [46] describes the difference between two clusterings. In terms of semantic segmentation a cluster consists of connected pixels belonging to the same class and a clustering is the set of clusters describing the whole image space. Thus a clustering C consists of clusters $C_1, C_2, ..., C_K$ and the probability of a pixel being in cluster C_k is described by $P(k) = \frac{n_k}{n}$ where n_k is the number of pixels in cluster C_k and n the number of pixels in all clusters together which is equal to the number of all pixels in the image. Now the entropy associated with the clustering is defined as:

$$H(C) = -\sum_{k=1}^{K} P(k) log P(k) \tag{2.9}$$

The entropy can be interpreted as a measure of uncertainty and in this context is 0 when there is only one cluster, so each pixel is certainly within this cluster. Thus for more than one cluster, the value is always greater than 0. Besides the entropy, the mutual information $I(C, C')$ between two clusterings C and C' is important for the VoI and described as:

$$I(C, C') = \sum_{k=1}^{K} \sum_{k'=1}^{K'} P(k, k') log \frac{P(k, k')}{P(k)P'(k')} \tag{2.10}$$

with $P(k, k')$ representing the probability a point belongs to the clusters C_k in C and to $C'_{k'}$ in C'. Entropy and mutual information together describe the VoI between two clusterings:

$$VoI(C, C') = H(C) + H(C') - 2I(C, C') \tag{2.11}$$

This can be converted into:

$$VoI(C, C') = [H(C) - I(C, C')] + [H(C') - I(C, C')] = H(C|C') + H(C'|C) \tag{2.12}$$

where $H(C|C')$ and $H(C'|C)$ represent the conditional entropies. In the context of image segmentation, they can also be interpreted as false merges and false splits respectively, or in other words as under-segmentation and over-segmentation. Thus as they together build the VoI, the metric gives an impression of the correct number of clusters within the predictions compared to the ground truth.

3 Methods

In this chapter, the two deep learning architectures for semantic segmentation that will be compared in this work, U-Net as an established CNN and SegFormer as a promising Transformer, are introduced in more detail. Moreover, the characteristics of the dataset that the models are applied to will be explained as well as the specific training procedures and the aspects that the models are compared along.

3.1 U-Net

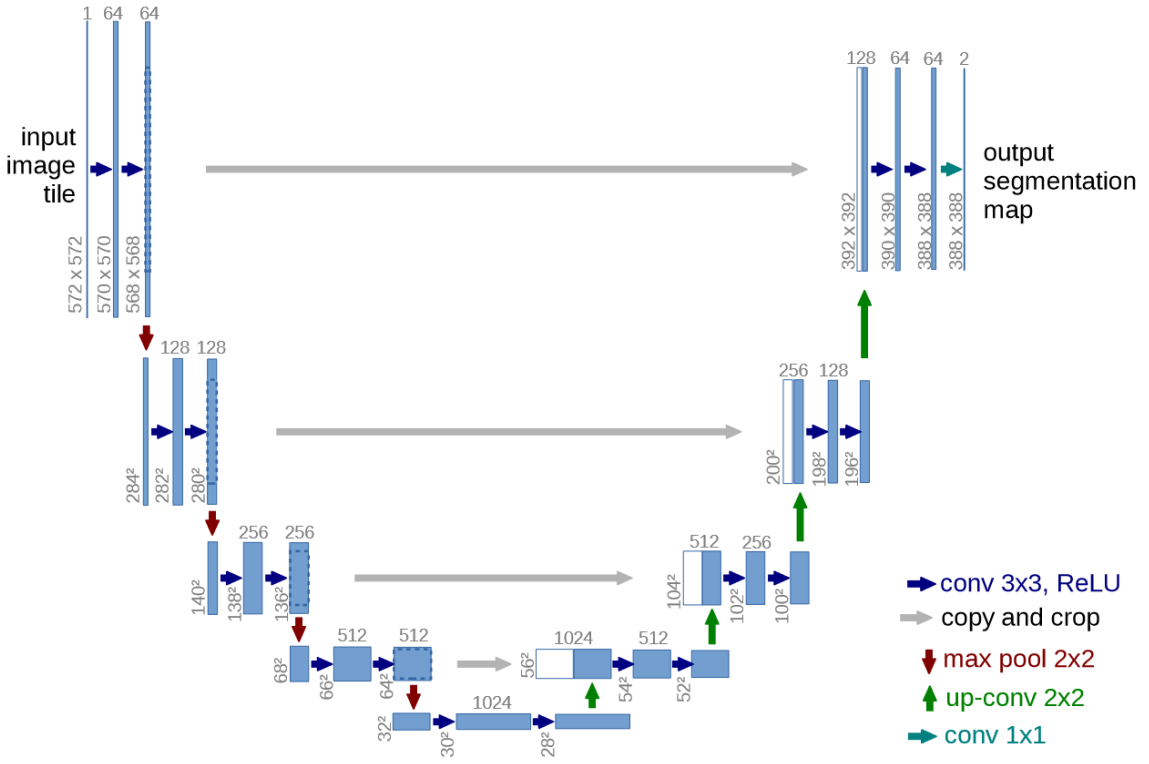


Figure 3.1: Original U-Net architecture from Ronneberger *et al.* [1].

U-Net is a CNN architecture for semantic segmentation. It was developed by Ronneberger *et al.* [1] for the segmentation of medical microscopy images and has a focus on being trained with few images. The term U-Net is derived from the shape of the architecture that consists of a contracting path and a symmetric expanding path that together form a U-shape, as can be seen in figure 3.1.

The contracting path, the encoder of the network, is a typical CNN architecture, with multiple convolutional layers. Each layer includes ReLU, followed by a 2×2 max pooling with stride 2. After each pooling, the number of filters is doubled in the following convolution step.

The expanding path, the decoder of the network, reverses the convolution from the contracting path by replacing pooling operations with transposed convolutions for upsampling. Again usual convolutions are applied, additionally incorporating features from the contracting path for localization purposes using skip connections to the encoder stages. The advantage of the several layers in the decoder is that context information can be propagated to higher-resolution layers.

In the original implementation as shown in figure 3.1, a valid convolution was used, i.e. no padding was applied. This leads to segmentation maps that have a side length of 388 compared to an input size of 572, thus just segmenting less than 50 percent of the area of the input image. The authors argue that in this way only pixels are part of the segmentation map for which the whole context is available. This was not a problematic issue for the authors as they applied the model to very large images and thus were able to apply an overlap-tile strategy and mirroring to segment all parts of the images.

In contrast to the implementation from the original paper, the customized U-Net implemented in this work makes use of zero-padding to preserve the image dimensions. In addition, batch normalization is applied, which is usually part of state-of-the-art architectures and is also implemented in SegFormer. Thus the comparability of both networks regarding current trends and new findings in deep learning research is ensured.

3.2 SegFormer

As the title of the respective paper states, SegFormer is a "Simple and Efficient Design for Semantic Segmentation with Transformers" [35]. This is achieved by:

- Using efficient self-attention, reducing the complexity of the matrix multiplications.
- Aggregating information from multiple layers with different resolutions, enabling the use of a simple decoder.

- Dropping positional encoding to avoid interpolation for testing on another resolution than training.

The SegFormer architecture is depicted in 3.2 and its details are described in the following sections.

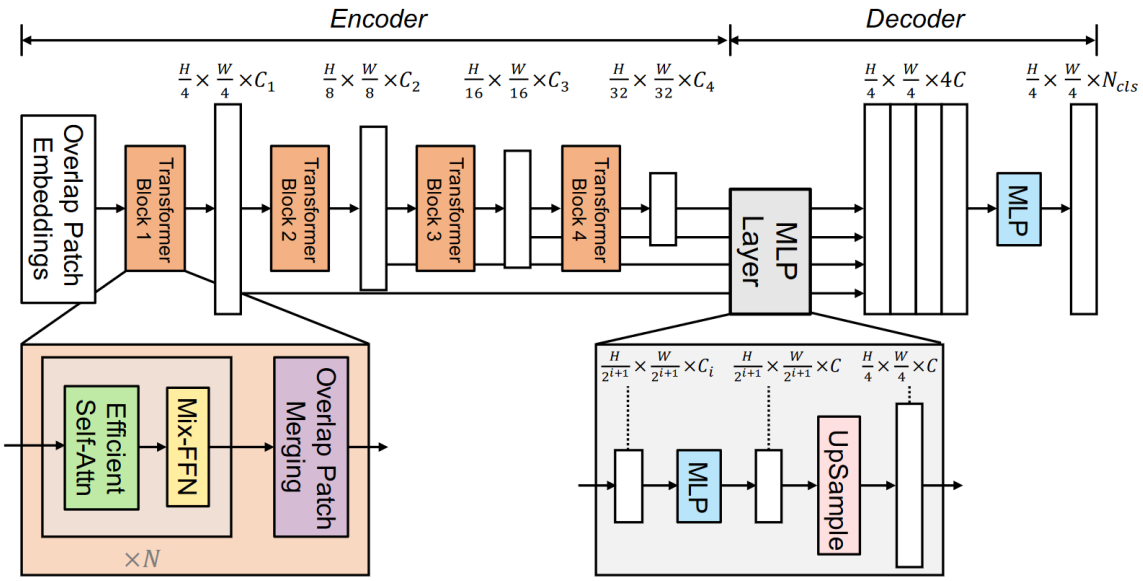


Figure 3.2: Original SegFormer architecture from Xie *et al.* [35].

3.2.1 Encoder

The encoder produces feature maps of four different scales that have sizes $(\frac{H}{k_i} \times \frac{W}{k_i} \times C_i)$ where H and W are the height and width of the input image respectively, C_i is the number of channels and k_i is the factor by which the resolution is reduced at each stage i. The values for k_i and C_i are $k = [4, 8, 16, 32]$ and $C = [64, 128, 320, 512]$ for the stages $i = [1, 2, 3, 4]$ respectively. In earlier stages, local features are learned. More global features and thus coarser feature maps are learned in later stages. The results of previous stages are used as input for the next stage and the resulting feature maps of all four stages are utilized as input for the decoder. Each feature map is created by a Transformer block, consisting of the same three elements applied one after another. These are efficient self-attention, Mix-FFN and overlap patch merging.

Efficient Self-Attention

The usual self-attention (equation 2.3) has a complexity of $O(N^2)$, where $N = H \times W$ is the length of the sequence, that is reduced in the efficient self-attention by a reduction ratio R to get a complexity

of $O(\frac{N^2}{R})$. To achieve this, the sequence reduction process introduced with the Pyramid Vision Transformer [47] is used, which reduces the spatial scale of the key and value sequences before the attention operation. The reduction is done by reshaping and linearly projecting the sequences, which in practice can just be implemented as a convolutional layer, where $\sqrt{R} = stride = kernelsize$. Thus the complexity for matrix multiplications is reduced by R, as each side of the image is reduced by $\sqrt{R}$ and hence the whole image and the corresponding sequence by R. The number of heads for the multihead self-attention is given with $N = [1, 2, 5, 8]$ for the respective stages where each head consists of the efficient self-attention and a Mix-FFN block.

Mix-FFN

Mix-FFN is used to remove positional encoding (PE) as implemented by ViT while still providing positional information. PE only works well when the test images have the same size as the training images, otherwise, the accuracy decreases when interpolating the results. In the Mix-FFN, SegFormer adapts the conditional positional encoding from CPVT [48], by leaving out PE and just using a 3×3 convolutional layer with zero-padding and $stride = 1$ to preserve location information [49], arguing that PE is not necessary for segmentation tasks. It is called Mix-FFN as it mixes a 3×3 convolution into the feed-forward network (FFN). In Mix-FFN, first MLP is applied to the input, followed by the $Conv_{3\times3}$, GELU for activation and MLP again. With a skip connection, the input is added again, resulting in the following formula for these layers:

$$x_{out} = MLP(GELU(Conv_{3\times3}(MLP(x_{in})))) + x_{in} \tag{3.1}$$

where x_{in} is the result of the preceding self-attention block. Thus it allows the positional information to flow through the Transformer block without the explicit positional encoding.

Overlapped Patch Embedding

Usually, Transformer blocks use non-overlapping patches for their input embeddings, but SegFormer makes use of overlapping patches to keep information at the edges of the patches, which is more important for segmentation tasks than for classification. To do so, the patches are generated using a convolution operation with $stride < kernel$. For the first stage, SegFormer implements a convolution with $K = 7, S = 4, P = 3$ and for the later stages the parameters are $K = 3, S = 2, P = 1$, where K is the kernel size, S the stride and P the padding size. With these settings, in the first stage, the image is basically split into 4×4 patches, which is significantly smaller than the 16×16 patches used in ViT, favoring the finer resolutions that are important for a dense prediction task. After each of the following stages, the patch sizes are doubled with the given convolution settings.

3.2.2 Decoder

After creating the four feature maps in the different resolutions, a lightweight All-MLP decoder is applied by SegFormer, to avoid costly computations when decoding. In the decoder, first, the channels of all feature maps need to be set to the same dimension, achieved by using an MLP. To concatenate the features in the next step, the features from stages 2-4 are upsampled to the same size as those from stage 1, i.e. one-fourth of the original resolution in width and height. Thus the feature maps now have the same resolution and number of channels. To unify the concatenated features another MLP layer is applied that is not depicted in figure 3.2. As the final step, a last MLP layer takes the unified features to calculate the segmentation mask, with a shape of $\frac{W}{4} \times \frac{H}{4} \times N_{cls}$, where N_{cls} is the number of categories. The segmentation maps with one-fourth of the original resolution are usually simply interpolated to get a result at the full resolution of the input.

Such a lightweight decoder only consisting of MLP layers is enabled by the large effective receptive field of the encoder, that utilizes the feature maps of the different scales.

3.3 Dataset

In the following, the datasets used within the thesis are explained. They are used for different purposes which are mentioned in section 3.5 and in the results chapter 4.

3.3.1 ISPRS Datasets

The International Society for Photogrammetry and Remote Sensing (ISPRS) provides benchmark datasets for semantic segmentation of remote sensing images. They are used as benchmarks in many publications about semantic segmentation (cf. [2], [7]). One of the datasets covers the city of Potsdam[1] and another one the city of Vaihingen[2]. The data is freely available for download via the corresponding website[3]. Both datasets consist of very high-resolution true orthophoto (TOP) tiles. This means the images are geometrically corrected to correspond to a given map correction which has the advantage of allowing valid measurements in the images but has the disadvantage of some

[1] https://www.isprs.org/education/benchmarks/UrbanSemLab/2d-sem-label-potsdam.aspx
[2] https://www.isprs.org/education/benchmarks/UrbanSemLab/2d-sem-label-vaihingen.aspx
[3] https://www.isprs.org/education/benchmarks/UrbanSemLab/default.aspx

fragments, especially at edges of buildings. For ground truth data the images of both datasets are labeled manually with six land cover classes common for urban scenes, which are:

1. Impervious surfaces
2. Building
3. Low vegetation
4. Tree
5. Car
6. Clutter/background

While the first five are mostly self-explanatory, the class clutter/background includes different objects like water bodies as they are not that frequent in the covered areas, thus they do not build a class on their own. Larger objects that can not be put into one of the other classes, like containers or tennis courts, are also part of this class.

Potsdam Dataset

As the name says the dataset covers the German city of Potsdam showing a typical historic city with large building blocks, narrow streets and dense settlement structures. The images in the dataset were taken in a phase of the year with little vegetation, i.e. bald trees. Three different types of images are provided, each with a ground sampling distance of 5cm. There are images available with 4 channels (R-G-B-IR) and with different combinations of 3 channels (R-G-B and IR-R-G). In addition, digital surface models (DSM) are provided which are also available in a normalized format, i.e. the ground height is removed thus giving the height above the terrain for each pixel. Overall the dataset consists of 38 patches (cf. figure 3.3), all with the same dimension of 6000×6000 pixels for TOP, ground truth and DSM.

Vaihingen Dataset

The dataset covers the small German village of Vaihingen with 33 patches of different dimensions. TOP, DSM and ground truth are provided with a ground sampling distance of 9cm. The TOP is only available with the band combination IR-R-G. The images were taken in a phase of the year with full-blown vegetation, i.e. leaved trees.

3.3.2 FloodNet

The FloodNet dataset [50] consists of images with about 1.5cm spatial resolution captured by small unmanned aerial vehicle (UAV) platforms after Hurricane Harvey which made landfall in Texas and Louisiana, USA, in August 2017. The images were taken by emergency responders thus it can be expected that they reflect data that would usually be available after flood events. The dataset contains 2,343 images, each image labeled as flooded / non-flooded, with an image classified as flooded when over 30% of the area covered in the image is flooded. Moreover, each pixel is annotated with one of ten classes, which are building-flooded, building-non-flooded, road-flooded, road-non-flooded, water, tree, vehicle, pool, grass and background. Thus natural water bodies like rivers and lakes are distinguished from areas flooded as a consequence of the hurricane by the respective classes. A building and all its pixels are labeled as flooded if at least one side of a building is touching the flood water. The dataset is openly available via GitHub[4].

3.3.3 Preparation

To use the images of the mentioned datasets as input for the models they need to be preprocessed by mutliple steps. One step is the normalization of each channel of the input images by subtracting the mean and dividing by the standard deviation of the whole dataset. Moreover, the large tiles need to be split into smaller patches and the data has to be split into training, validation and test subsets.

Patchifying

One main step is to resize the images to reasonable image sizes, as the patches of the mentioned datasets are too large to be fed into the models. The models can take inputs of different sizes, which is one of the advantages of fully convolutional networks compared to networks with fully connected layers [21], In order to perform convolutions, the number of pixels per row and column must be divisible by 2. To enable this, it makes sense to use images with a square size and with the number of pixels per side to the base of 2, i.e. 2^x. Thus deeper layers of the networks can easily convolute the resulting feature maps without the need for interpolation. Image sizes used within this thesis are 128×128, 256×256, 512×512 and 1024×1024 pixels.

To get images with reasonable sizes different approaches could be employed. First, it would be possible to just shrink the images by resizing, i.e. squeeze all pixels of the original image into one image of the desired size. This would mean the loss of many valuable pixels containing useful

[4]FloodNet on GitHub

information and would blur the image a lot. A more useful option is to subset the original image into smaller patches by cropping them. Again, different possibilities emerge in how to subset the image to make the best use of the data. The differences lie in the overlap and resize settings used for the subsets. As most work was done on the Potsdam dataset, consisting of images with size 6000×6000 pixels each, the details will be explained along this size as an example. For instance, to get subsets of size 512×512, the image could easily be split into 500×500 pixel patches without any overlap and these could be upscaled to 512 pixels per side. The interpolation from 500 to 512 pixels would mean new pixels were invented that were not present before which should be avoided if possible. Thus alternatively, the image could be split directly into 512×512 pixel patches without overlap but this would lead to remaining pixels with stripes of size 368 at the edges, meaning a lot of the data could not be used. The method used eventually, incorporates all available pixels without any resizing. For example with 512×512 pixels as the target size for the patches, it uses an overlap of 12 pixels with neighboring patches. Only the last patches in a row and/or column have an overlap of 24 pixels to the previous patches. Thus, resizing of any kind is avoided, i.e. no pixels have to be newly constructed or summarized. The drawback of the overlap is that the overlapping areas are used more than once in the dataset but as the overlap is quite small, the influence is assumed to be negligible in this work. Actually, the overlap of the patches can even be beneficial to the segmentation of objects at the edges of patches that might lose some of their context due to the subsetting.

Data Split

To train and evaluate a model on a specific dataset, the dataset is split into training, validation and test subsets. To enable an independent evaluation of the model, none of the data that is part of the training subset should also be included in the validation or test subset, as the model would have an advantage for a correct prediction on the data it was already trained on.

The patches of the Potsdam dataset are arranged in rows and columns (cf. figure 3.3). With the help of these, the test subset is split up from the rest of the data by using the patches from the two columns at the eastward boundary of the dataset, namely columns 14 and 15. In doing so, the test data is spatially split from the training data, avoiding a high spatial autocorrelation. Hence the test samples cover neighborhoods of the city the model is not trained on and thus the test results might give a first impression of the stability and transferability of the model on new data. This split also ensures that all classes covered by the dataset are represented by a similar amount of pixels in training and test data, as illustrated in table 3.1. As a consequence, all models trained on the Potsdam dataset are trained on patches generated from columns 07 to 13 and tested on columns 14 and 15. Thus the quantitative results (cf. section 4.3) depict the performance of the models applied

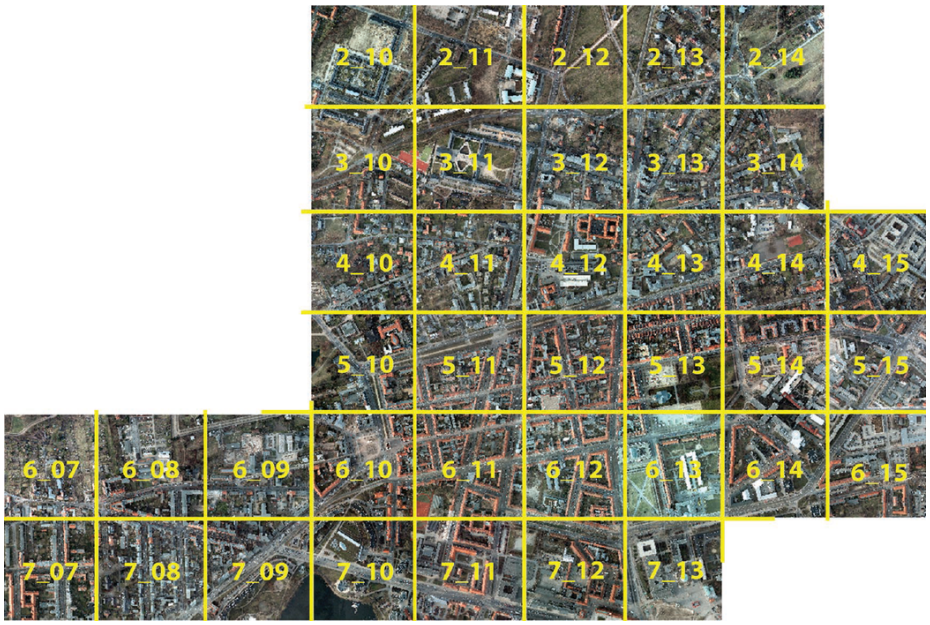


Figure 3.3: The Potsdam dataset contains 38 patches that are arranged in nine columns (07-15) and six rows (2-7).

to the test data. The slim dataset used for one of the evaluated model variants refers to a training subset that only contains images from column 10.

The splitting of training data into a subset for training and validation is done randomly for all models, with 75% of the data used for training and 25% for validation. To ensure the comparability of the models, a random seed is used that makes sure models of the same configuration are trained on the same data split.

Table 3.1: Distribution of pixels belonging to the different classes for the given train and test subsets. Pixels are given in million (M).

	Train/Validation		Test	
Class	Pixel (M)	%	Pixel (M)	%
Impervious	359.0	31.7	90.5	30.0
Building	286.7	25.3	69.3	22.9
Vegetation	251.1	22.2	65.3	21.6
Tree	164.1	14.5	55.6	18.4
Car	18.0	1.6	6.5	2.2
Clutter	53.6	4.7	14.7	4.9

The FloodNet dataset is provided with data that is already split into training, validation and test subsets. Since the test subset is not accompanied by ground truth labels and thus can not be used for the evaluation of the models, the provided validation dataset is used for testing. As a consequence,

for training purposes, the training dataset is used for training and validation by splitting the images randomly as described before for the other dataset.

3.4 Hardware and Software

All relevant calculations were performed on facilities from the University of Münster. The model training was conducted on the PALMA-II high-performance computing cluster that provides different GPUs. GPUs utilized for the training include Nvidia Titan RTX and Nvidia A100 SXM 80GB. Everything was implemented with Python as the programming language and the machine learning framework PyTorch was used to create the deep learning models with version 1.9.0 installed within the PALMA-II fosscuda toolchain.

The evaluation of the trained models and visualization of the results was carried out with the help of JupyterHub also provided by the University of Münster. A Nvidia A40 GPU with 12GB RAM is integrated into JupyterHub and the installed PyTorch version was 1.11.0 together with Python version 3.10.8.

The source code for this thesis is available at GitHub[5] and preserved via Zenodo [51].

3.4.1 Hyperparameters

As part of the development of the models, different hyperparameters are tuned manually. Adam is used as an optimizer and a fixed learning rate is set that is evaluated during several model training runs with different settings. A learning rate is found to be good when the validation loss and other metrics applied during validation are stable, i.e. they are not oscillating a lot For example, a model is found to be stable when the accuracy of the model does not jump from 0.8 to 0.3 and then back to 0.7 within three epochs but rather increases constantly or stays roughly at a value once it is reached. To find such a stable learning rate, first fewer training data and epochs are used. After finding a good fit for the learning rate of a model it is trained on all desired training data and the number of epochs is increased until the model does not improve for multiple epochs.

Learning rates used for most of the final models are between 3×10^{-5} and 5×10^{-5}. Most models were trained for 300 or 400 epochs.

[5] https://github.com/jsten07/CNNvsTransformer

3.5 Model Comparison and Evaluation

The goal of this thesis is to compare the U-Net and SegFormer architecture to each other, which is mainly done by evaluating their performance on the semantic segmentation of the Potsdam dataset as an example for remote sensing data. The images from the Potsdam dataset are prepared as described in section 3.3.3 to train the models with various configurations and thus compare them in different aspects. The following list shows the different dataset compositions along with the annotation that will be used for the models in the next sections, with *Model* in the list being a placeholder for the respective architecture, i.e. U-Net or SegFormer. Correspondingly, model notations and the respective data used for training are:

***Model*-128 to *Model*-1024** Potsdam dataset RGB-images of the annotated patch size from columns 07 to 13 (cf. figure 3.3). Patch sizes used are 128×128, 256×256, 512×512, 1024×1024.

***Model*-IR** Potsdam dataset IRRG-images from columns 07 to 13 with patch size 512×512.

***Model*-S** Potsdam dataset RGB-images from column 10 with patch size 512×512.

***Model*-floodnet-512 and *Model*-floodnet-1024** FloodNet training dataset RGB-images of the annotated patch size.

All resulting models are evaluated qualitatively and quantitatively. The qualitative evaluation is conducted by looking at multiple examples and presenting interesting samples, especially samples where a distinct difference between the results of the different models can be observed. The quantitative evaluation considers the mIoU and accuracy of the models as well as VoI as a measure for the over-segmentation of the models, also referred to as homogeneity in the following. Details about further specific evaluations and how the comparability of the models is ensured are explained in the following subsections.

3.5.1 Comparability

Both architectures can be used with different settings that lead to different numbers of parameters per model. Table 3.2 shows some of the possible variations and their corresponding number of parameters. The settings used in this work are the ones marked with bold text. They were chosen as they provide a comparable number of parameters. For the SegFormer model, the medium configuration was chosen over the configuration with fewer parameters because it provides significantly better results [35] without having a much higher difference in the number of parameters to the respective U-Net configuration. Other means to ensure the comparability as good as possible include the use of comparable settings for the training, for example comparable hyperparameters (cf.

chapter 3.4.1), and the same data split (cf. chapter 3.3.3) for the models that are directly compared to each other.

Table 3.2: Parameter of different selected model configurations for U-Net and SegFormer. Chosen settings are highlighted in bold. Layer settings correspond to the number of feature channels per layer for U-Net and to the number of encoder layers per stage for SegFormer. For SegFormer the number of output channels per layer is the same for all configurations with (64, 128, 320, 512) for the respective stages. Parameters are given in million (M).

U-Net		SegFormer	
Layer	Parameters (M)	Layer	Parameters (M)
(32, 64, 128, 256)	7.8	(3, 4, 6, 3)	24.7
(64, 128, 256, 512)	**31.0**	**(3, 4, 18, 3)**	**44.6**
(128, 256, 512, 1024)	124.1	(3, 8, 27, 3)	61.4

3.5.2 Performance Measures

The quantitative performance of the models is measured by calculating several metrics when applying them to a test dataset that was not included in the training process. Different metrics are calculated for the overall model and for each class separately. Per class, the IoU and the VoI are calculated. For the overall models the accuracy, mIoU and VoI are calculated. It is important to note that the mIoU is simply the mean of the IoU of the individual classes but the VoI for the overall model is calculated independently from the values per class.

3.5.3 Transferability and Generalizability

The transferability evaluated in this work stands for the potential of the model to be transferred to other data in two senses:

1. How well do the models perform applied on unfamiliar data, i.e. data of another shape and/or from another region than the training data?

2. Are the insights gained on the models universal or are they specific to the dataset?

To evaluate the first topic, the models trained on the IRRG data are applied to the Vaihingen dataset. The images of the dataset are also available with IRRG bands and the ground truth data is labeled with the same categories as the Potsdam dataset, allowing a straightforward evaluation of the model performance on unknown data. The datasets differ in the structure of the covered city, the ground sampling distance and the time of the year the images were taken. Since the IRRG models are just trained on 512×512 pixel patches and because of the difference in the ground sampling

distances between the datasets, the transferability of the model is tested on 512×512 pixel patches from the Vaihingen dataset as well as 256×256 patches that are resized to a side length of 512. With upsampling the data, the ground sampling distance is artificially decreased to 4.5cm which is more similar to the training data than the original data and might influence the results of the evaluation.

The second topic is evaluated by training both architectures on the FloodNet dataset and calculating the same metrics as for the usual dataset. For both architectures two models are trained, one for patches of size 512×512 and one for patches of size 1024×1024. Although the training is carried out in a less structured way than for the other dataset, this could give an initial impression about the validity of the other results.

4 Results

In this section, the results are presented in four different sections. First, some of the insights gained during the training are presented like differences in the learning rates and training time. Secondly, a selection of the most interesting qualitative results is shown. Third a quantitative evaluation of the results is done followed by a section about the transferability of the models. The results focus on the differences between the models. Several more than the presented learnings were achieved but presenting and discussing all of them would be out of scope for this work.

4.1 Model Training

During the training of the models, some differences in the training process were observed. SegFormer reacted more sensitively to changes in the learning rate and needed a lower learning rate to achieve satisfying results, i.e. results with stable validation metrics. The average learning rate leading to good results for SegFormer was at 5×10^{-5} whereas U-Net also had good results with a learning rate of 5×10^{-4}. Training SegFormer models with a higher learning rate often led to oscillating metrics, but training U-Net with a lower learning rate still yielded good results. Thus to increase the comparability of the models, both architectures were trained with the same learning rate for the same training data.

With the learning rate set to the same value for both architectures, for most model types U-Net reached its best epoch faster than SegFormer. For example, the best epoch for U-Net-1024 was 159 of 400 and for SegFormer-1024 it was epoch 284. However for some configurations also SegFormer reached its optimum faster, hence no significant difference is identifiable.

Although not systematically measured, another observation was that the training time per epoch was longer for SegFormer than for U-Net models, when trained on the same hardware and with the same learning rate.

4.2 Qualitative Results

In this section, qualitative results are shown by displaying an input image next to the ground truth labels together with the segmentations of the respective U-Net and SegFormer models. In addition to that, the wrong segmentations of the models, i.e. pixels where the prediction does not match the ground truth label, are marked in a supplementary image. Analyses are conducted for differences in the results within the model configurations as well as for particularly interesting study cases. The colors for the illustration of classes in ground truth and prediction are chosen based on the colors from the original dataset representations.

4.2.1 Model Configurations

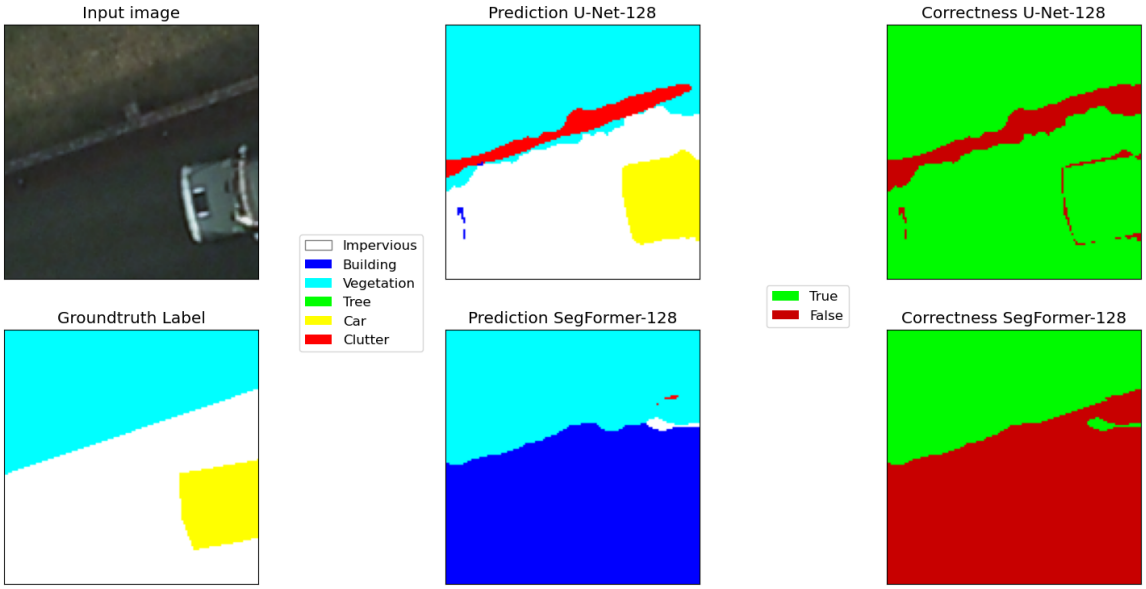
In the following, at least two examples per model configuration are shown, where mostly one example is favourable for the U-Net model and one for the SegFormer model. In appendix 6.2, additionally for each model multiple examples are presented, to give a further impression of the qualitative performance of the models. Overall both architectures perform well across all configurations a nd the examples that are shown illustrate some of the infrequent visible differences.

128×128 Pixel Patches

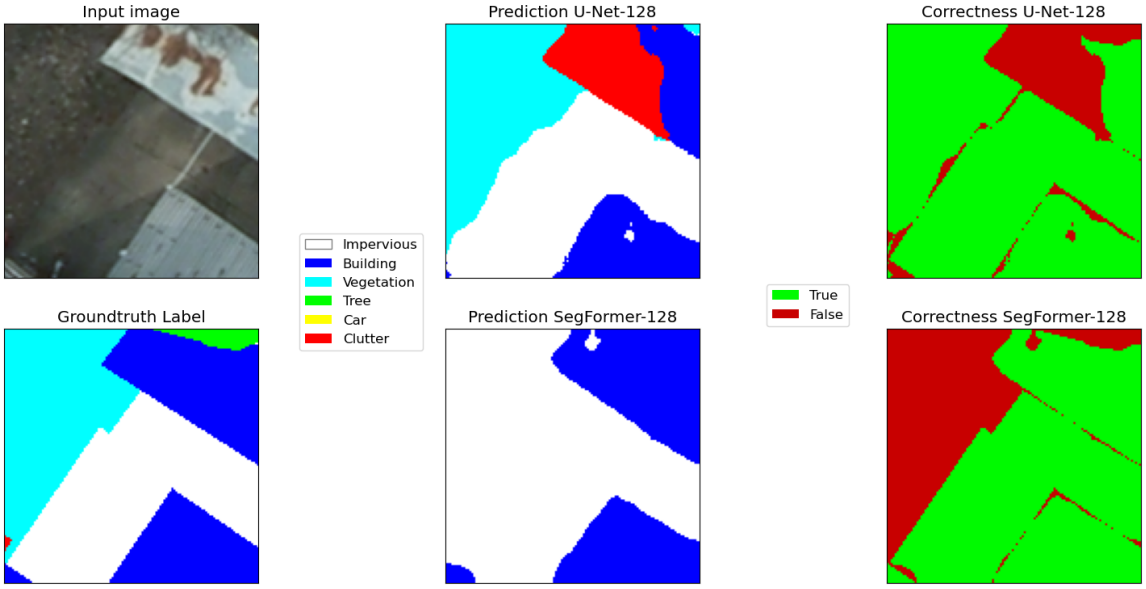
Figure 4.1 shows that the models both segment the examples reasonably. SegFormer-128 has difficulties in predicting larger areas like the impervious surface and the car in figure 4.1a, and it confuses the vegetation area with an impervious surface in figure 4.1b. U-Net-128 performs better but has some difficulties with parts of larger areas. In both examples, it detects parts of vegetation and a building wrongly as clutter. In the second example, SegFormer detects the small part of the building in the bottom left contrary to U-Net and both models miss the part of the tree in the top right.

256×256 Pixel Patches

In figure 4.2a the image mostly shows a paved impervious surface. U-Net-256 produces a noisy result with multiple building clusters within the impervious area. SegFormer-256 segments the whole image as a building, thus also missing the part of the car at the bottom. Both models detect the cars in figure 4.2b well and SegFormer-256 also distinguishes the other areas mostly accurately. U-Net-256 confuses the shadow of the car on the vegetation area partly with an impervious surface and adds some clutter and building clusters on the impervious surface.

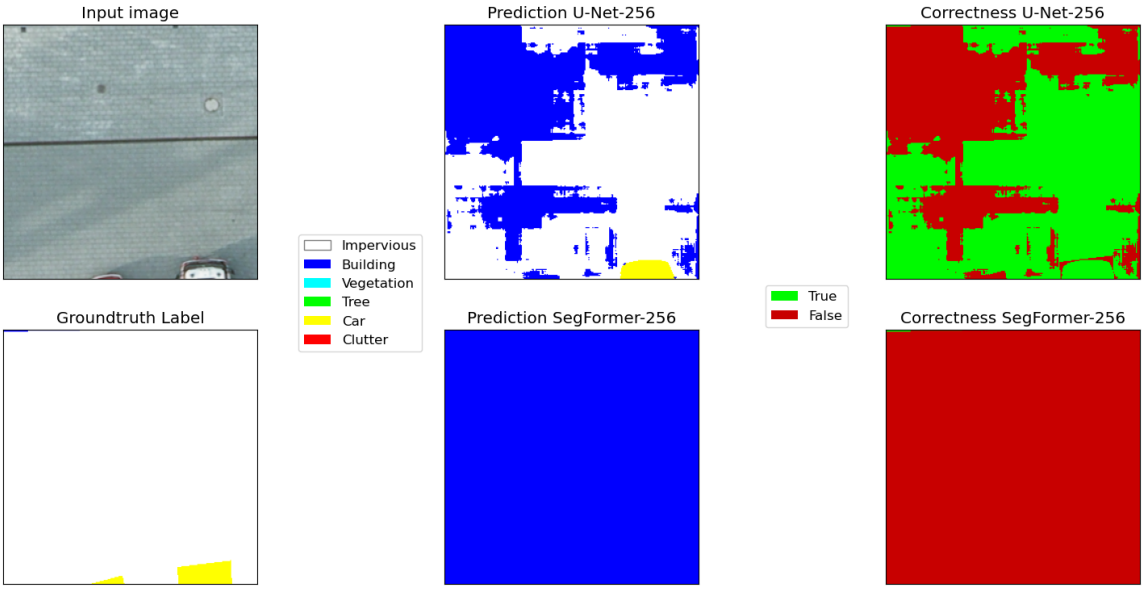


(a) U-Net-128 and SegFormer-128 example predictions (1) on image patch from tile 4_15.

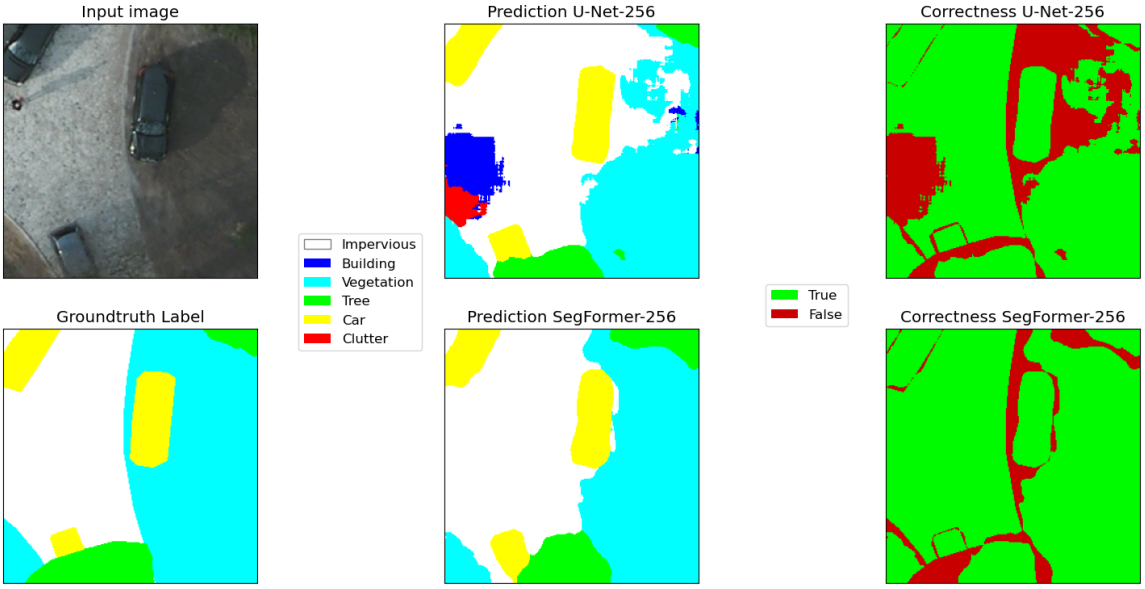


(b) U-Net-128 and SegFormer-128 example predictions (2) on image patch from tile 4_15.

Figure 4.1: Examples for segmentation of models U-Net-128 and SegFormer-128.



(a) U-Net-256 and SegFormer-256 example predictions on image patch from tile 6_15.



(b) U-Net-256 and SegFormer-256 example predictions on image patch from tile 4_15.

Figure 4.2: Examples for segmentation of models U-Net-256 and SegFormer-256.

512×512 Pixel Patches

The results on the 512×512 pixel patches are particularly interesting to look at since they show an extent large enough to cover multiple objects and thus different classes while still not being too overloaded. For example figure 4.3a is a good illustration of how well both models generally perform. This is not only the case for U-Net-512 and SegFormer-512 but also for the other models discussed in the results. In this specific example, both models create good results with their biggest issue missing the building at the bottom right below the tree. The part of the tree at the top left is missed by both models and SegFormer-512 also misses the tree clusters at the bottom that U-Net-512 detects.

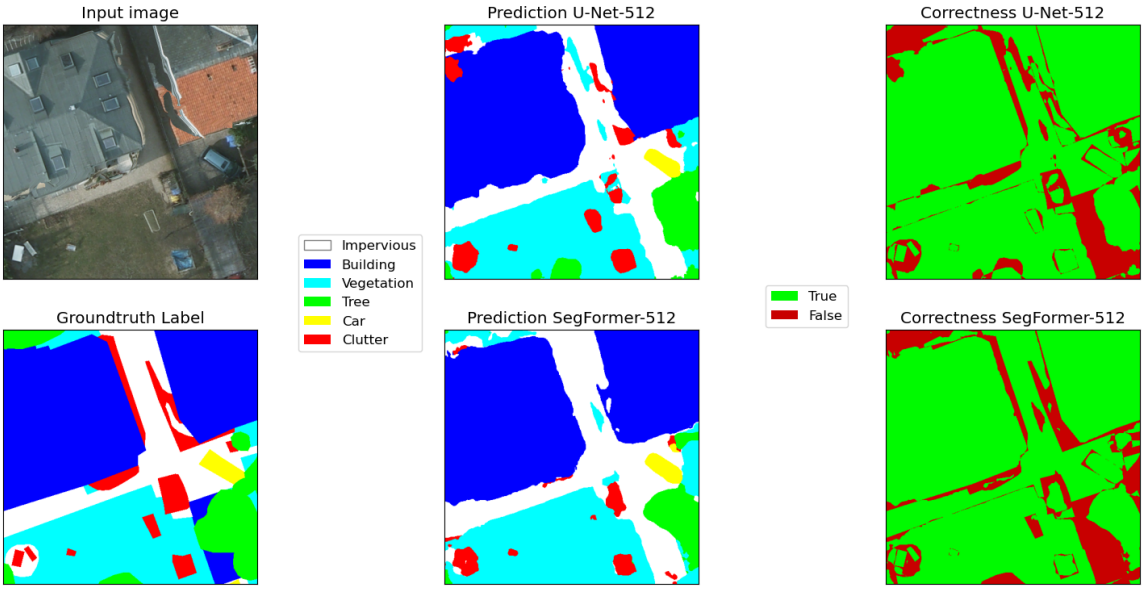
Figure 4.3b is an example of how well both models segment cars. Moreover, both models detect the small part of the vegetation area at the top left. U-Net-512 also segments the other areas of the input well whereas SegFormer-512 confuses the building with an impervious surface. Similarly, SegFormer also segments the whole image in figure 4.4b as an impervious surface although it mostly shows a flat planted roof. However, it detects the building with a flat roof in 4.4a well. Both examples show the inhomogeneity and thus over-segmentation often resulting from U-Net-512 predictions, especially as part of larger areas like buildings.

1024×1024 Pixel Patches

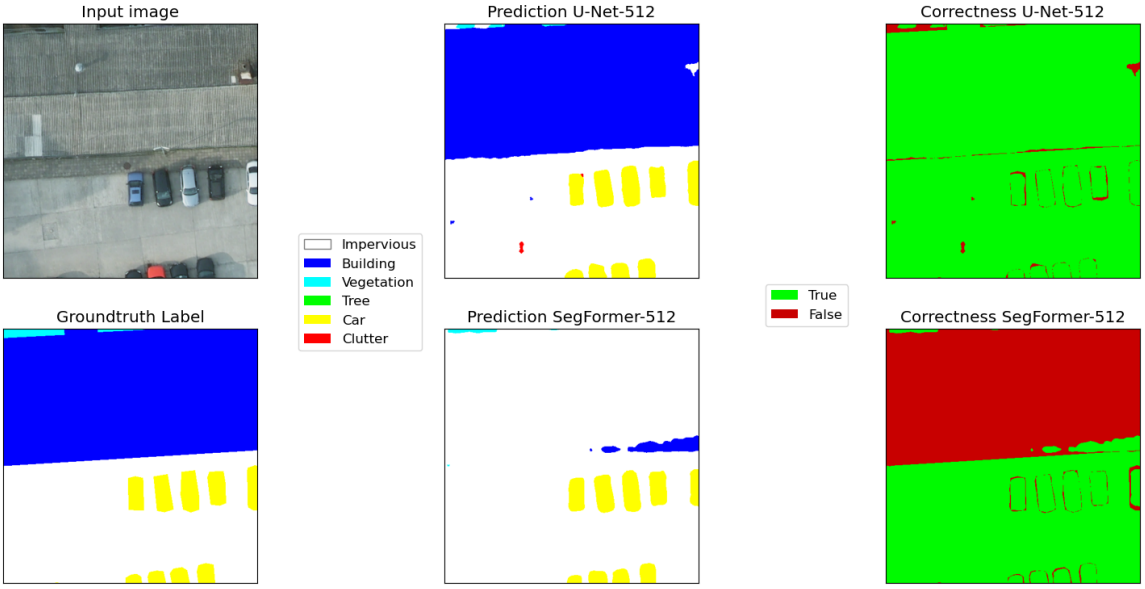
At first glance, the differences between the models get less obvious for larger images and the results of the different models look more similar. Both models perform well detecting larger relevant objects but still, U-Net results are affected by over-segmentation (e.g. figure 4.5a) whereas SegFormer misses some finer structures or confuses vegetation and impervious surfaces more frequently (e.g. figure 4.5b). Both examples in figure 4.5 show that the models have similar problems like the differentiation of vegetation from trees and impervious surfaces. An interesting sample for their difference is the vegetation area within the trees at the top of figure 4.5b where U-Net-1024 over-segments the vegetation and SegFormer-1024 under-segments it.

Slim Dataset

The examples in figure 4.6 show that the models trained on less data still perform very well qualitatively although the lack of training data is visible at some points. Figure 4.6a for instance shows how SegFormer-S misses parts of the large building and segments them as impervious in combination with the windows in the roof that are in parts classified as car. U-Net-S on the other hand segments multiple building clusters in figure 4.6b although there is no building at all in the

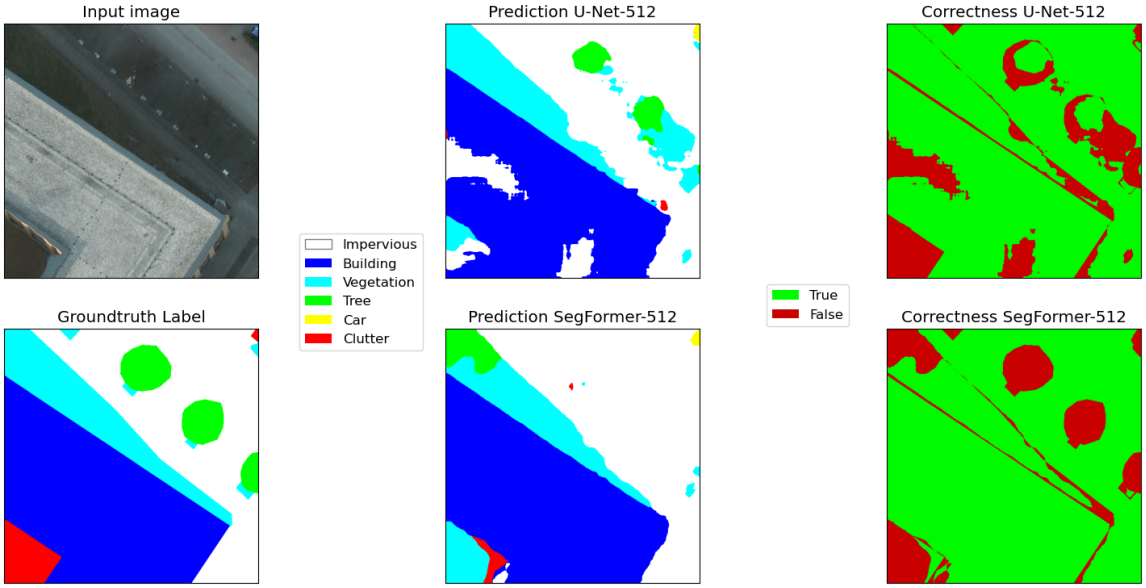


(a) U-Net-512 and SegFormer-512 example predictions on image patch from tile 3_14.

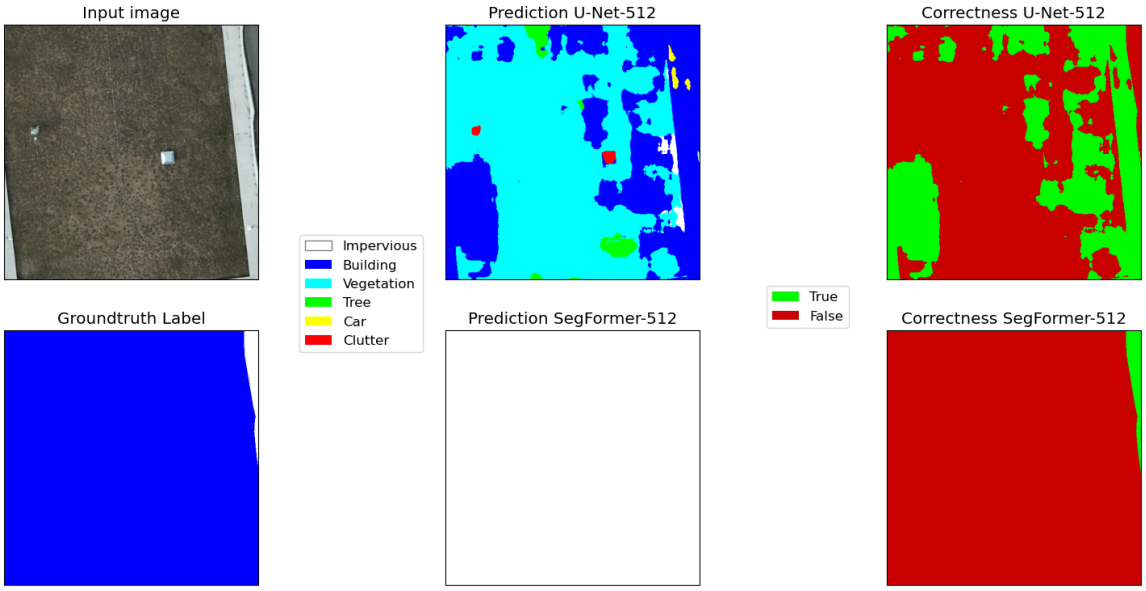


(b) U-Net-512 and SegFormer-512 example predictions on image patch from tile 4_14.

Figure 4.3: Examples for segmentation of models U-Net-512 and SegFormer-512.

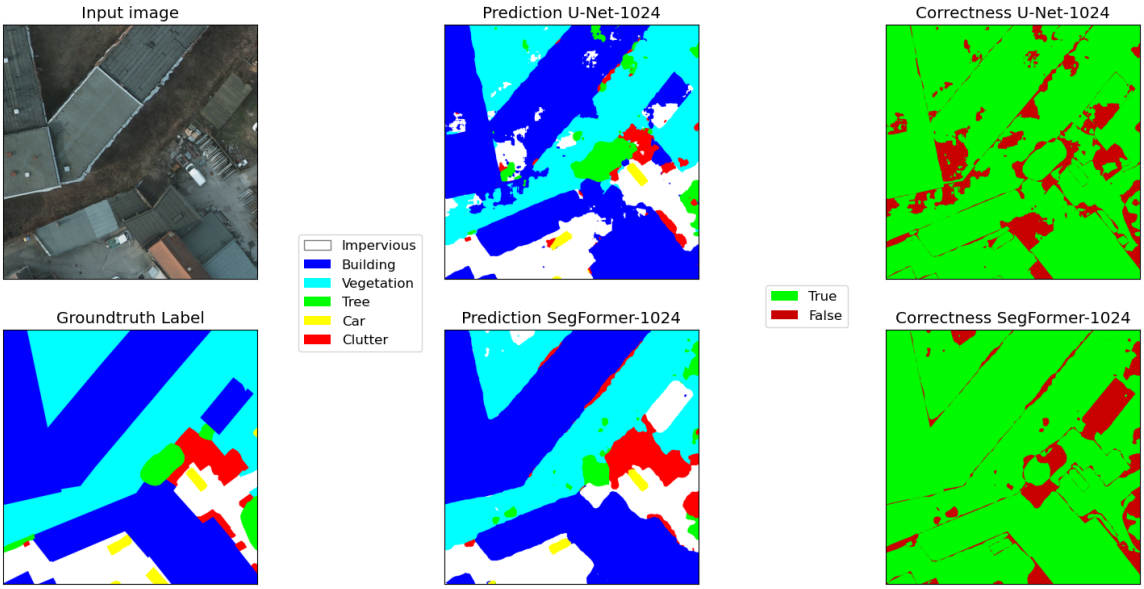


(a) U-Net-512 and SegFormer-512 example predictions on image patch from tile 4_15.

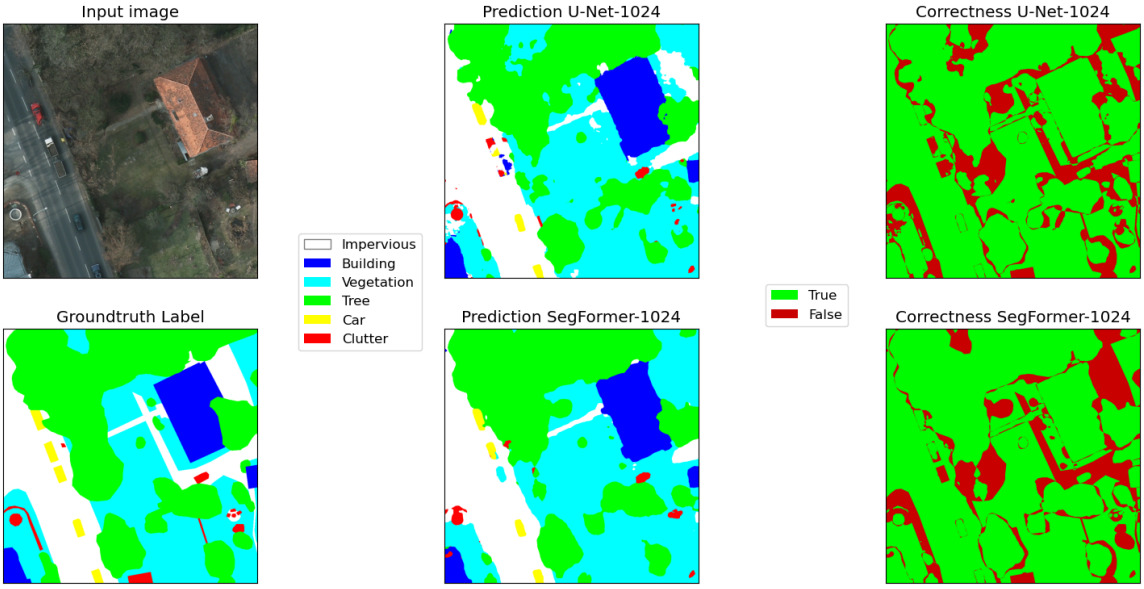


(b) U-Net-512 and SegFormer-512 example predictions on image patch from tile 6_15.

Figure 4.4: Examples for segmentation of models U-Net-512 and SegFormer-512 focusing on over-segmentation.



(a) U-Net-1024 and SegFormer-1024 example predictions on image patch from tile 4_15.



(b) U-Net-1024 and SegFormer-1024 example predictions on image patch from tile 2_14.

Figure 4.5: Examples for segmentation of models U-Net-1024 and SegFormer-1024.

input. Both models miss the building below the tree in the upper example in a similar way as the models trained on more data (cf. figure 4.3a). Altogether, from the examples it is apparent for both models that the quality of the results worsens with missing training data.

IRRG Images

The models trained and applied on the IRRG-images perform similarly well as the corresponding RGB models. Figure 4.7b shows both models segment most objects correctly with a difference in the contours of the trees. These are segmented with fewer bulges and thus closer to the ground truth by SegFormer-IR. The object in the class clutter on the left is classified wrong by both models, from SegFormer-IR as building and from U-Net-IR as impervious surface. The small trees next to this object are detected by U-Net-IR whereas SegFormer-IR only segments a fraction of one of the trees. Also noticeable is the extent containing the two bottommost cars. They are parked below a tree and are detected by both models, but SegFormer-IR segments parts of them as clutter and U-Net-IR classifies the area between them as impervious surface instead of tree and/or vegetation.

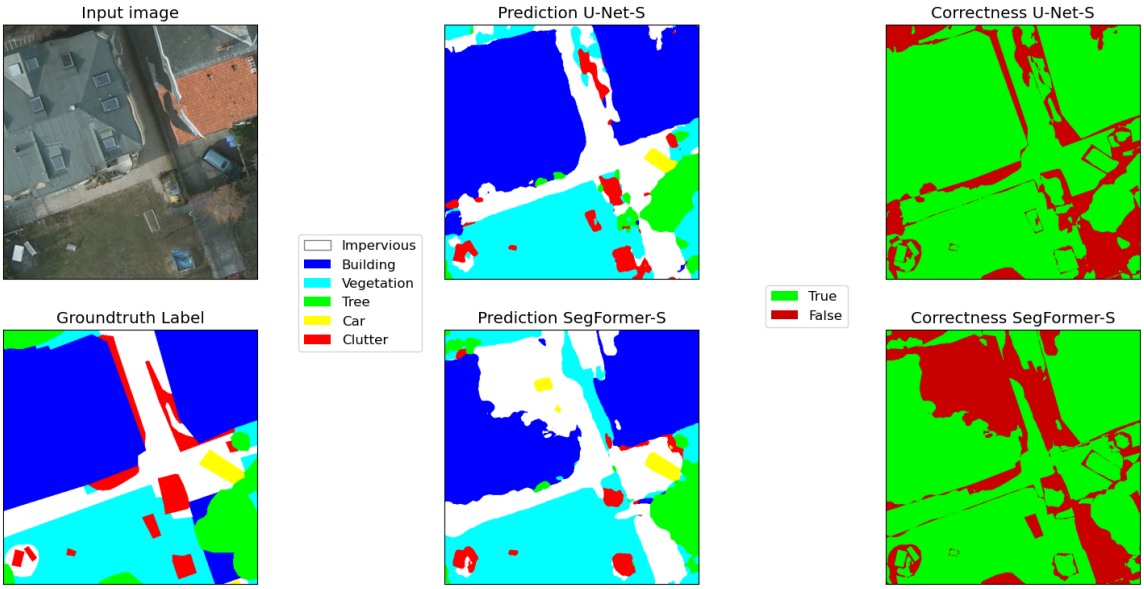
Both models detect the basic structure of the building and impervious surface correctly in figure 4.7a but the parts of the planted area of the roof are classified as vegetation by both models. Moreover, U-Net-IR segments parts of the windows on the roof as clutter and also segments the edges of the building very generous and coarse. Interestingly both models do not detect any of the trees, just some of them as vegetation, although from a human perspective, the IR band gives a clear indication of the structures in the image being trees.

4.2.2 Interesting Study Cases

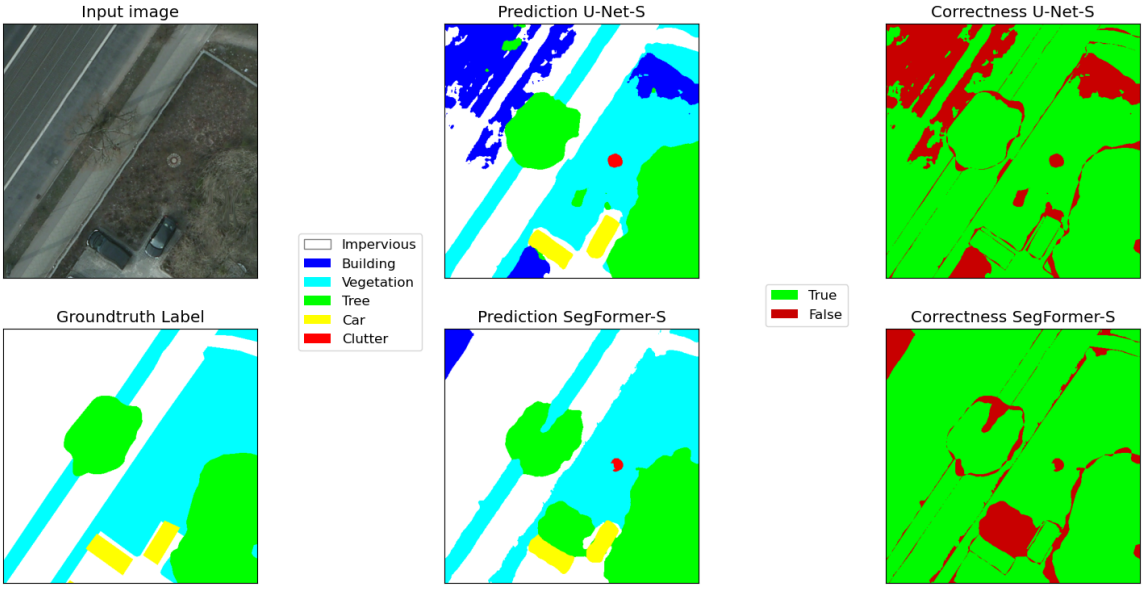
During the review of the qualitative results, some interesting study cases came up where one or both of the architectures consistently had difficulties in the segmentation process.

Missing Context

For the previously shown examples of the models with different patch sizes, it seems that both architectures improve by increasing the sizes of the images. For instance, the examples in figure 4.1 show that for smaller patches, larger areas are detected wrong, especially from the SegFormer model, which is also holding true for example 4.2a. These errors are also visible for larger patches with SegFormer-512, as seen in examples 4.3b and 4.4b. This could be reasoned by the missing context despite the larger patches since three sides of the buildings are cut off in the examples.

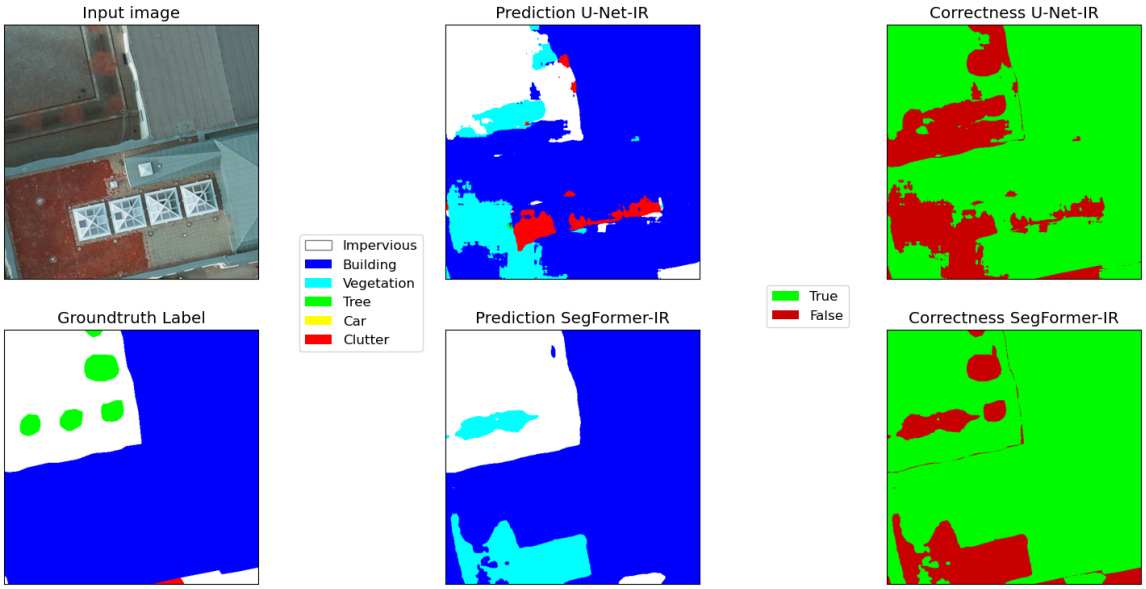


(a) U-Net-S and SegFormer-S example predictions on image patch from tile 3_14.

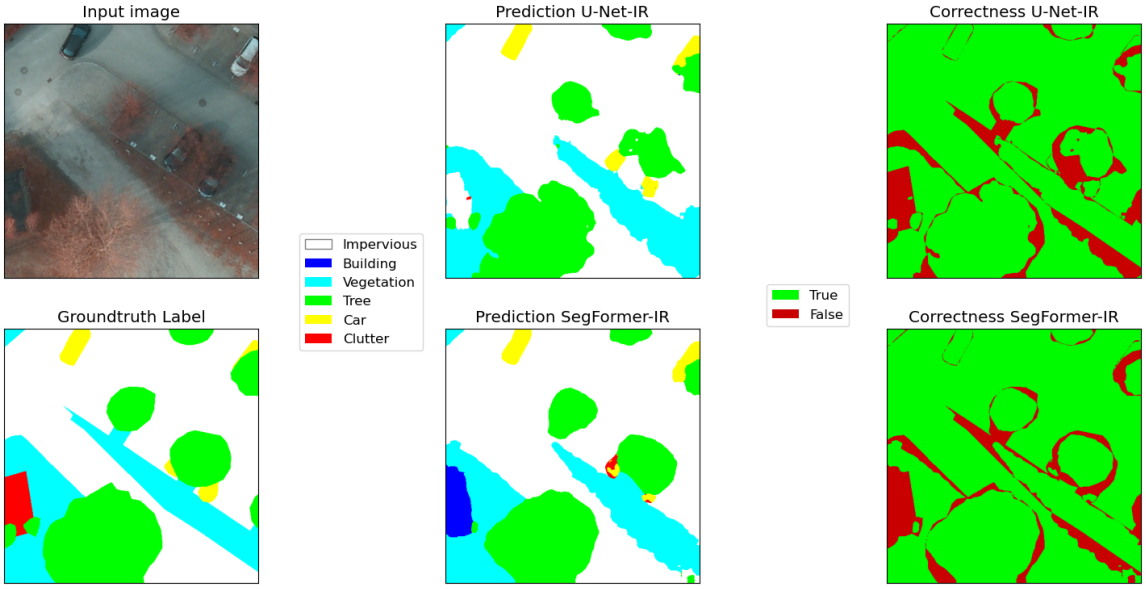


(b) U-Net-S and SegFormer-S example predictions on image patch from tile 6_15.

Figure 4.6: Examples for segmentation of models U-Net-S and SegFormer-S.



(a) U-Net-IR and SegFormer-IR example predictions on image patch from tile 6_15.



(b) U-Net-IR and SegFormer-IR example predictions on image patch from tile 4_15.

Figure 4.7: Examples for segmentation of models U-Net-IR and SegFormer-IR.

For smaller cut-off structures at the edges of images, the issues are not that significant. Only when there is just a very small portion of the object left in the image, it is sometimes missed. For example, parts of trees are missed by both models as depicted in figures 4.3a and 4.1b. In sum, U-Net seems to outperform SegFormer in segmenting cropped objects. Another example is the detection of smaller fractions of cars at image edges as shown in figures 4.2a or 4.5a. Thus the U-Net models have advantages in the prediction of smaller image patches, as smaller images are more likely to contain cropped structures.

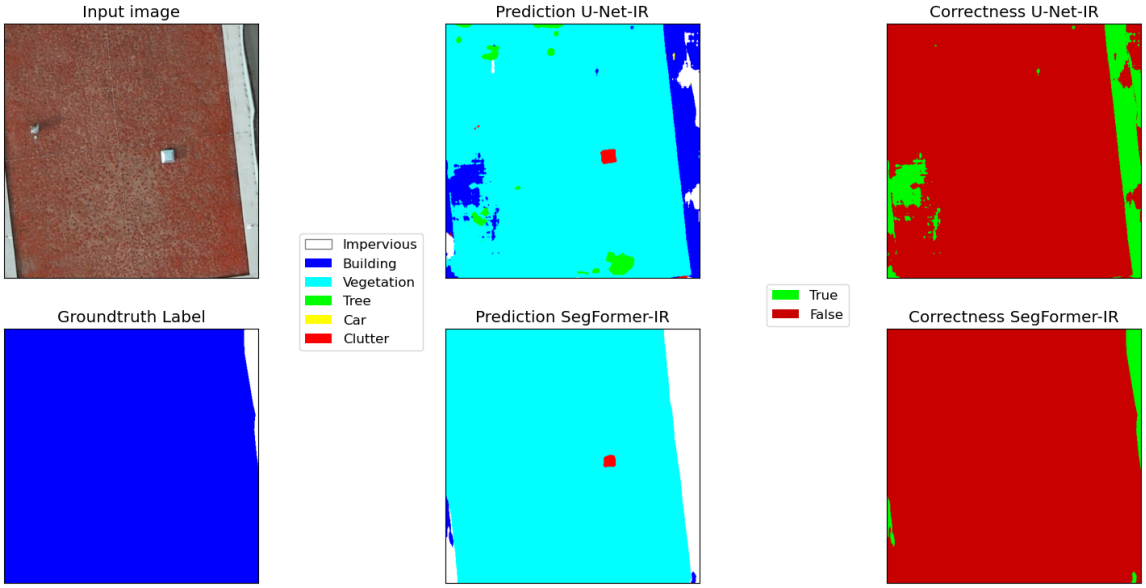
Flat and/or Greened Roofs

Greened roofs are labeled as buildings in the dataset but also have vegetation on top of them. Both architectures have difficulties in correctly segmenting the roofs as buildings, instead often detecting low vegetation or impervious surfaces (cf. figure 4.4b). Including the infrared channel seems to boost this effect as in figure 4.7a where both models segment the part of the building that is greened as vegetation. Example 4.8a also shows that both IRRG models determine the greened roof as vegetation. This is also the case for the prediction from U-Net-S on the same image extent, but SegFormer-S segments large parts of the roof correctly as a building (cf. figure 4.8).

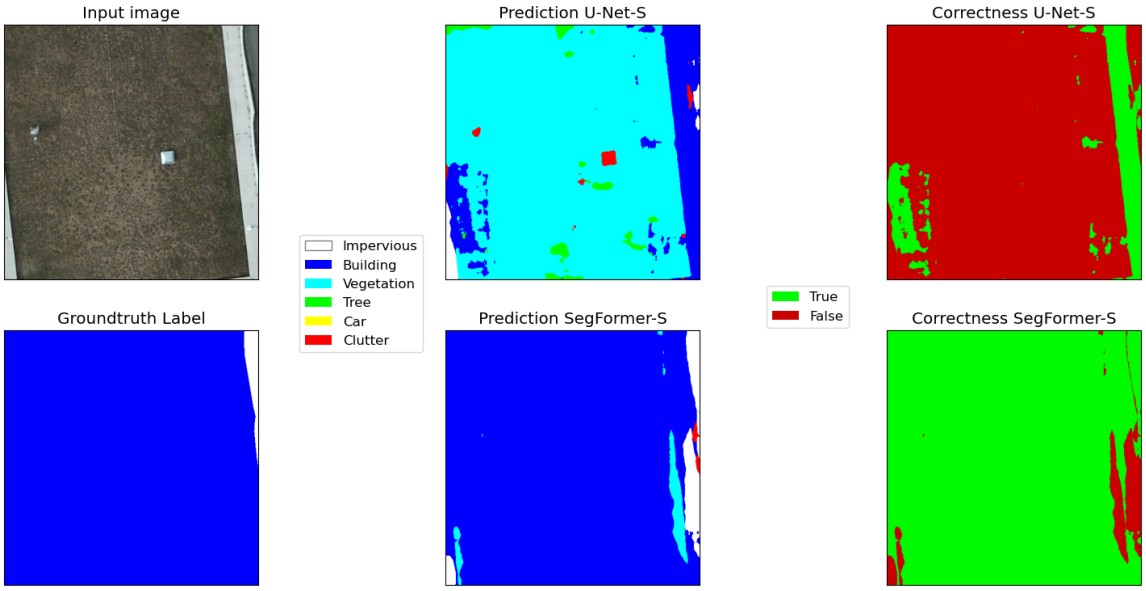
Classifying buildings also seems to be a difficult task with other types of roofs without greening. U-Net has weaknesses in detecting large buildings as a whole, especially buildings with flat roofs. It often adds some clusters of impervious or clutter to them. Examples are depicted in figures 4.7a, 4.5a and 4.4a. SegFormer on the other hand more likely segments larger parts of a building or a whole building wrongly. Mostly buildings are confused with impervious surfaces as can be seen in figures 4.6a, 4.5a (small building on the right) and 4.3b. For both architectures there are also examples of mixing up the classes the other way round, i.e. segmenting impervious as building, again with tendencies of U-Net segmenting parts of the objects wrongly (cf. figures 4.2a, 4.6b) and SegFormer detecting the whole object wrongly (cf. figures 4.1a, 4.2a).

Playing Fields and Water Bodies

It is difficult to evaluate the performance of the models on segmenting water bodies as they are labeled as clutter and not as a class on their own, as they are not that frequent in the data. Still, there are water bodies, specifically lakes, in the training as well as in the test data and there is a clear difference in the qualitative segmentation between the architectures. As depicted in figure 4.9a SegFormer segments the water in combination with the tree clearly as clutter. U-Net's prediction is more noisy and segments parts of the water as impervious or building. The tree above the water is



(a) U-Net-IR and SegFormer-IR example predictions on 512 × 512 image patch from tile 6_15.



(b) U-Net-S and SegFormer-S example predictions on 512 × 512 image patch from tile 6_15.

Figure 4.8: Examples for segmentation by different models on image patches depicting greened roofs.

segmented correctly by the SegFormer models but parts of it are said to be vegetation by the U-Net models. Similar results can be seen for other than the depicted model configurations.

Besides water bodies, non-turf courts are also labeled as clutter and give interesting study cases when segmented by the models. Both architectures interpret different types of court surfaces similarly. As depicted in figure 4.9b, Tartan-like red courts are mostly segmented correctly as clutter whereas the grey court is interpreted as impervious and what seems to be wet on the court is predicted to be vegetation. For the large grey area U-Net-1024 again over-segments, by adding clusters of other classes like building. SegFormer on the other hand adds clusters of buildings to the red court on the top right. Models with other configurations create similar predictions of courts in different image extents.

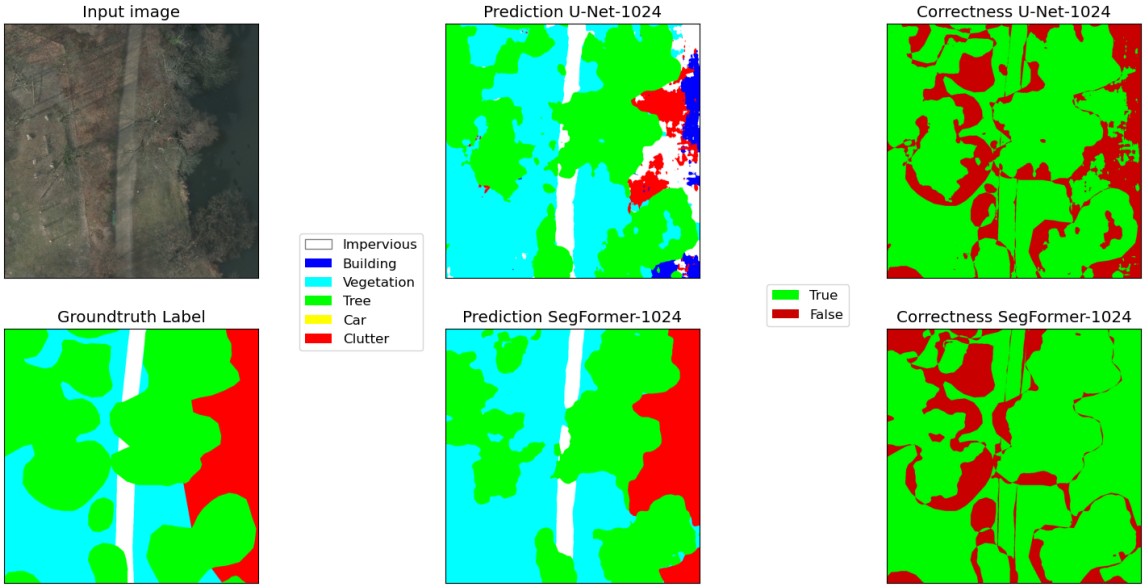
Homogeneity

In the many examples shown above one of the most noticeable differences between the architectures is that U-Net segmentations over all model settings tend to be more inhomogeneous and coarse. The U-Net models often over-segment, especially larger areas like buildings also include clusters of impervious surfaces (cf. figures 4.4a and 4.5a) or reversely (cf. figures 4.6b and 4.2a). Probably this is reasoned by the differences in the architectures, with U-Net only using convolutions that mostly extract local features whereas the self-attention integrated into SegFormer also encodes global interactions. Thus U-Net determines the class of a pixel largely based on the surrounding pixels whereas SegFormer is able to also incorporate information further apart.

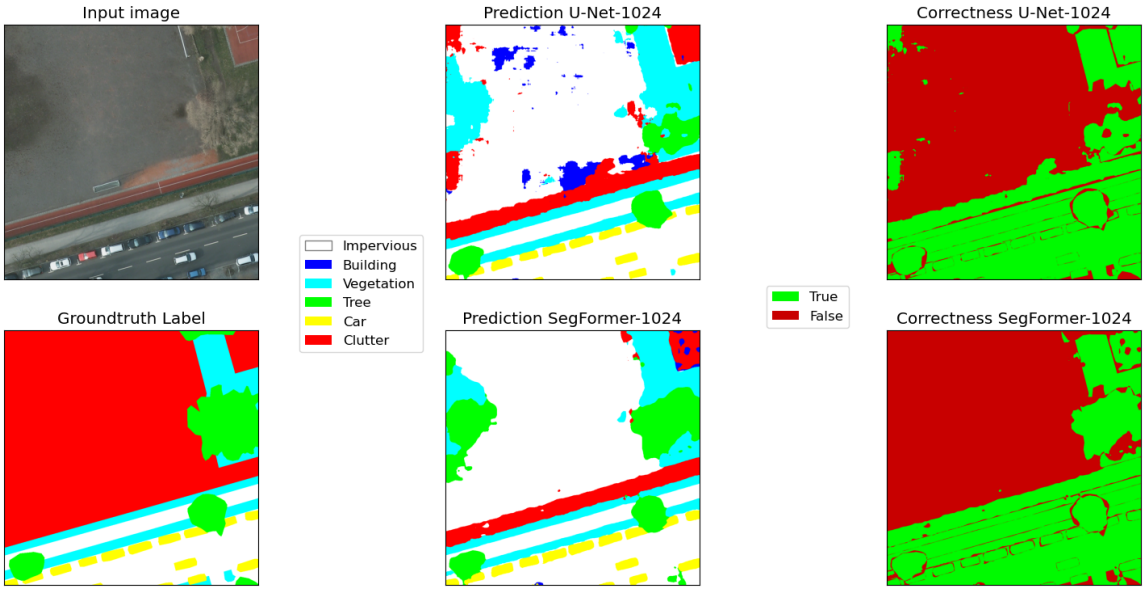
Furhter Findings

The previously mentioned results cover mainly the classes building, impervious surfaces and clutter as the most differences between the models can be seen with them. For the other classes, there is not such a clear difference.

Trees generally are detected well by both architectures and from models with different settings. Differences again can be seen in the homogeneity of the segmentations where SegFormer models create smoother segmentations of the trees than U-Net models. Since the trees are leafless both models have difficulties segmenting the trees as exactly as the labels and especially when the trees are over low vegetation areas both models often assign the vegetation class at the edges of the trees. This is also confirmed by the fact that assigning the class vegetation to tree is the most common misclassification relatively, except for the class clutter. As examples for clarification, confusion matrices for U-Net-512 (cf. table A.2) and SegFormer-512 (cf. table A.1) are added to the appendix. Values in the confusion matrices of the other models are similarly distributed, thus not all of them



(a) U-Net-1024 and SegFormer-1024 example predictions image patch from tile 3_14 depicting a lake.



(b) U-Net-1024 and SegFormer-1024 example predictions image patch from tile 4_14 depicting playing fields.

Figure 4.9: Examples for segmentations of interesting study cases (water bodies and playing fields) by models U-Net-1024 and SegFormer-1024.

are included. Both models also correctly segment trees above cars, although the cars are clearly distinguishable (cf. figures 4.9b, 4.7b, 4.2b). However, they have difficulties detecting buildings below trees, where they might segment the building as an impervious surface (cf. figure 4.3a).

Overall cars are segmented well without problems. As some of the examples indicate, the models often segment finer objects like cars and trees in some cases even more precisely than the labels are. Moreover, some trees that the labels missed were detected by the models.

Objects smaller than cars are also labeled and included in the class clutter, with outdoor furniture as one example that is displayed on the bottom right in figure 4.5b. Both models in this example assign clutter as the correct class to the objects but U-Net-1024 correclty segments distinct objects and SegFormer-1024 segments one connected area. This is an example of under-segmentation that can sometimes be observed in the SegFormer predictions. Altogether the class clutter is difficult to segment for both architectures, probably reasoned by the definition of the diverse class, covering everything that can not be assigned to the other five classes, from furniture to playing fields and water bodies.

4.3 Quantitative Results

The quantitative results are split into (mean) Intersection over Union and Variation of Information. The mean values for both metrics per model are visualized as bar charts and the values per class are depicted as radar charts. Table A.3 shows the accurate values and the accuracies of the models.

4.3.1 Intersection over Union

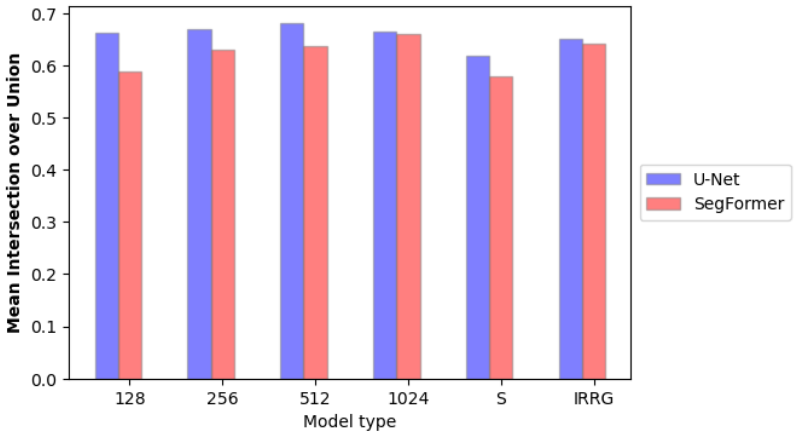
All models, including the models trained on the slim dataset, achieve a mIoU above 0.57 and accuracies between 0.765 and 0.836 when applied to the test dataset. The mIoU is higher for all U-Net models compared to the SegFormer models (cf. figure 4.10a). When comparing the models trained on the same data, U-Net-1024 and SegFormer-1024 have the smallest difference with 0.664 and 0.660. This is followed by another negligible difference between U-Net-IR and SegFormer-IR with 0.651 and 0.642 respectively. The difference is the largest between U-Net-128 and SegFormer-128 with mIoU values of 0.662 and 0.589. Thus the difference in the mIoU between the models of the same type applied to the test data gets smaller for larger patch sizes and SegFormer compared to U-Net seems to benefit more from the infrared band being included instead of the blue band.

Accordingly, also the IoU values per class get closer to each other for larger patch sizes (cf. figure 4.10b). Whereas U-Net-128 has distinctly better values than SegFormer-128 for each class, U-Net-512 is just clearly superior to SegFormer-512 in the classes clutter and tree. SegFormer-1024 even outperforms U-Net-1024 in the classes clutter, building and impervious surface. For all model configurations, the respective models for both architectures have a similar IoU for the class car whereas the class tree is the most disparate class where the U-Net models are superior consistently.

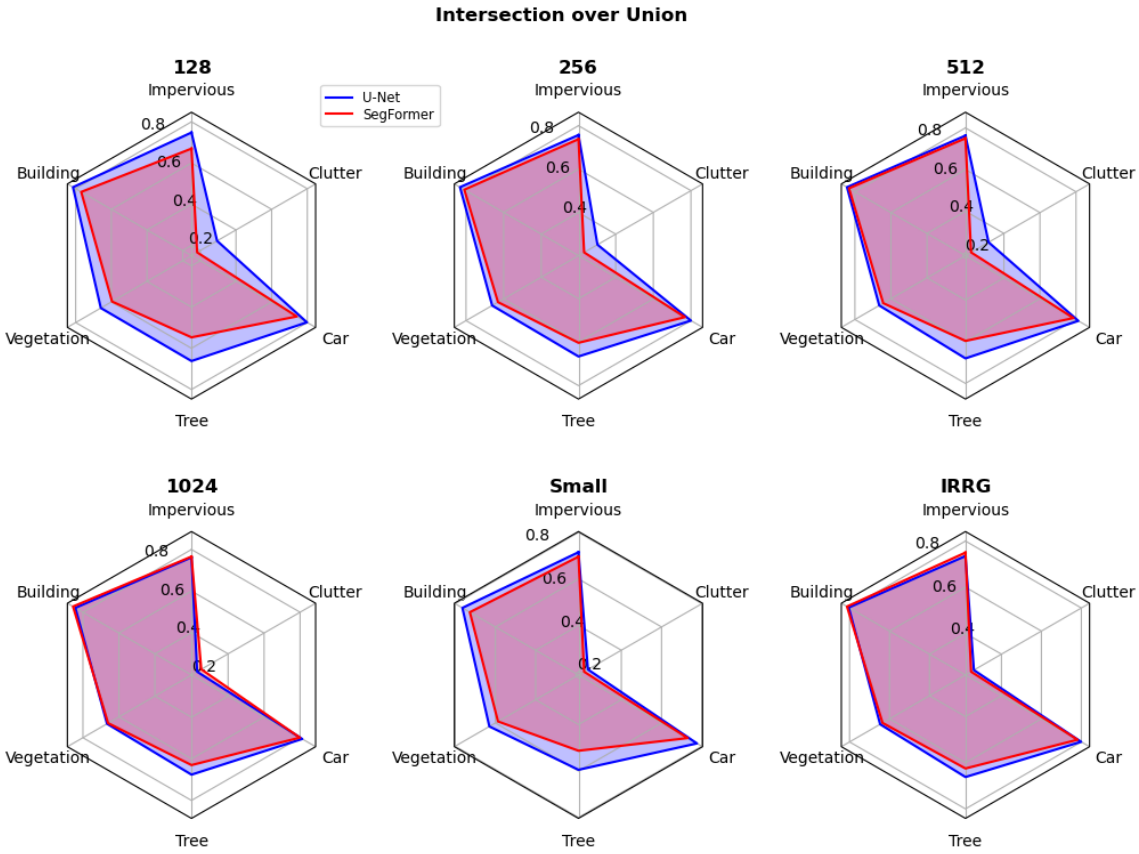
Almost all models across the configurations and architectures perform best on the class building with values around 0.8, followed by car and imperious surface. For all models, the performance on the class clutter is the worst, with 0.315 for U-Net-512 being the best of all clutter values.

4.3.2 Variation of Information

A model performs better in terms of the Variation of Information metric if it has a lower value. Thus for all tested configurations, the SegFormer models have a better VoI than the U-Net models, except for the slim dataset configuration where U-Net-S is better than SegFormer-S (cf. figure 4.11a). The VoI increases for larger image sizes which makes sense as the depicted values are the average VoI per prediction and larger images offer more room for variation between ground truth

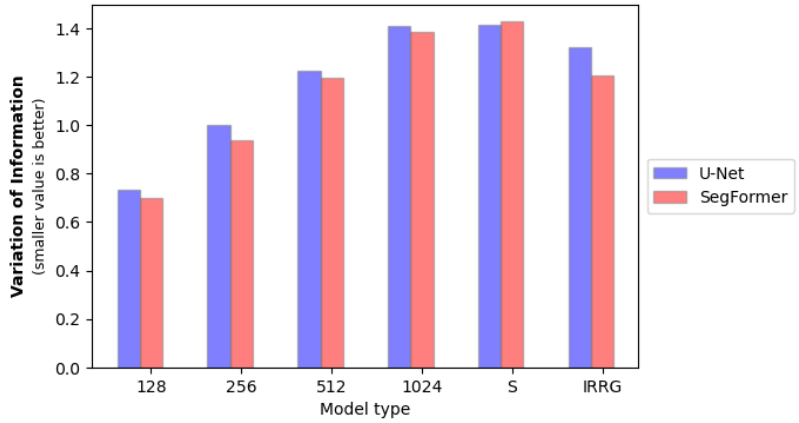


(a) Mean Intersection over Union for the different model settings.

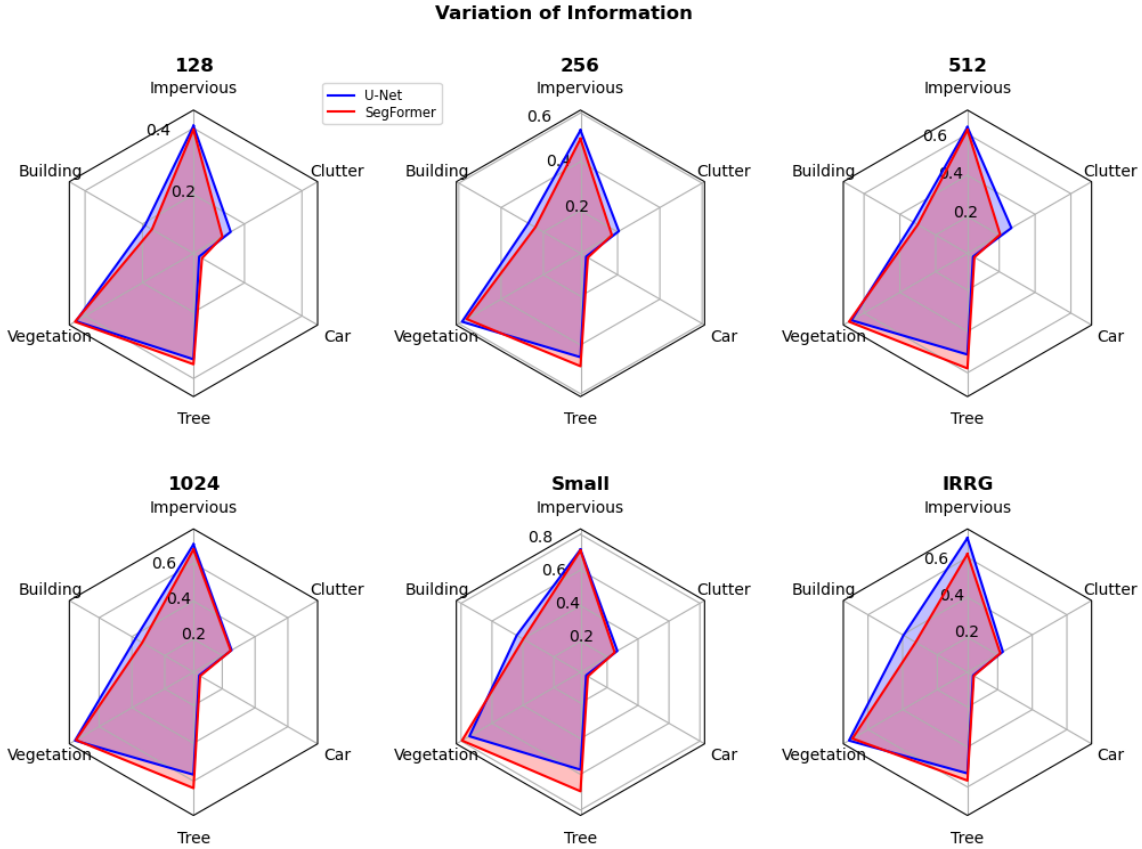


(b) Intersection over Union per class and type of model.

Figure 4.10: (Mean) Intersection over Union for the different model settings.



(a) Variation of Information for the different model settings; smaller values are better.



(b) Variation of Information per class and type of model; smaller values are better.

Figure 4.11: Variation of Information for the different model settings.

and prediction. Still, the models trained on the slim datasets have the highest overall VoI values, although they are trained on image patches with size 512×512.

The VoI per class varies a lot among the classes but not so much between the models. For the prediction of cars, there is a very low VoI with 0.075 for SegFormer-S being the highest value of all models. Vegetation has the highest VoI for all models, followed by impervious surfaces and trees. These values are comprehensible since cars are small distinct objects that the models also detected well regarding the IoU values, thus they do not offer much room for over- or under-segmentation. On the other hand, low vegetation and impervious surfaces cover large areas and are easy to confuse with each other. The VoI values per class that differ the most between models of the same type are tree and building with building having less variation for the SegFormer models and tree having less variation for U-Net models.

4.4 Transferability

In this section, the results to evaluate the transferability of the models are presented as well as the generalizability of the previous results, as described in the methods section 3.5.3.

4.4.1 Application to another Dataset

As the qualitative results (cf. figure 4.12) and also the quantitative results in table 4.1 show, applying a model trained on the Potsdam IRRG data to data with the same bands from the Vaihingen dataset is not very reasonable. Infrequently areas of buildings, vegetation, impervious surfaces and trees are segmented correctly as shown in figure 4.12. However, the depicted examples are chosen as two of the best results that were reviewed, as on average most of the images are segmented wrong (cf. figures B.19, fig:segformer-IR-vaihingen256-appendix, fig:unet-IR-vaihingen512-appendix, and fig:segformer-IR-vaihingen512-appendix). In many examples large areas are segmented as clutter and cars are only recognized occasionally.

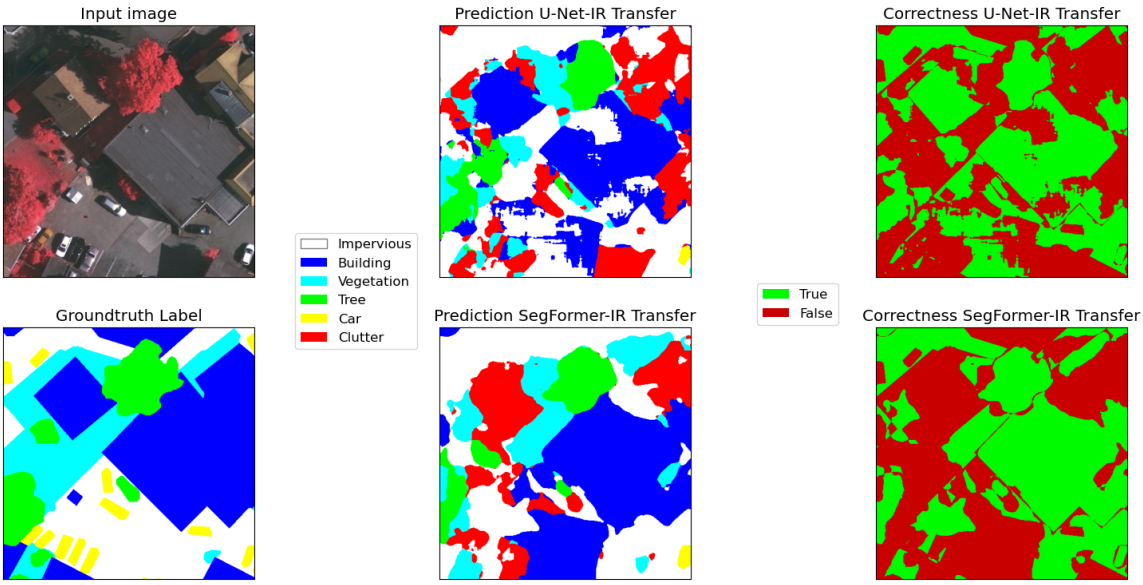
Reasons for this are probably mainly due to the differences in the datasets. The Potsdam images were taken with little vegetation, i.e. in winter with no leaves on the trees, and in Vaihingen in a phase with full-blown vegetation. This would already be a problem with RGB images but even more so with IRRG images since the IR band focuses on the vegetation. Moreover, the datasets have different ground sampling distances, with 9cm for the Vaihingen data and 5cm for Potsdam. Thus an object depicted by one pixel in the Vaihingen data is covered by almost four pixels in the Potsdam images, allowing the model trained on the Potsdam data to focus on details that are not available in the Vaihingen data. Correspondingly, the models were applied to different patch sizes but no significant differences can be observed between the application to patches with the original ground sampling distance of 9cm (figure 4.12a) and to the upsampled patches (figure 4.12b).

Table 4.1: mIoU of models trained on 512×512 Potsdam IRRG images when tested on the Vaihingen dataset with different input image sizes. Input images with patch size 256 are resized.

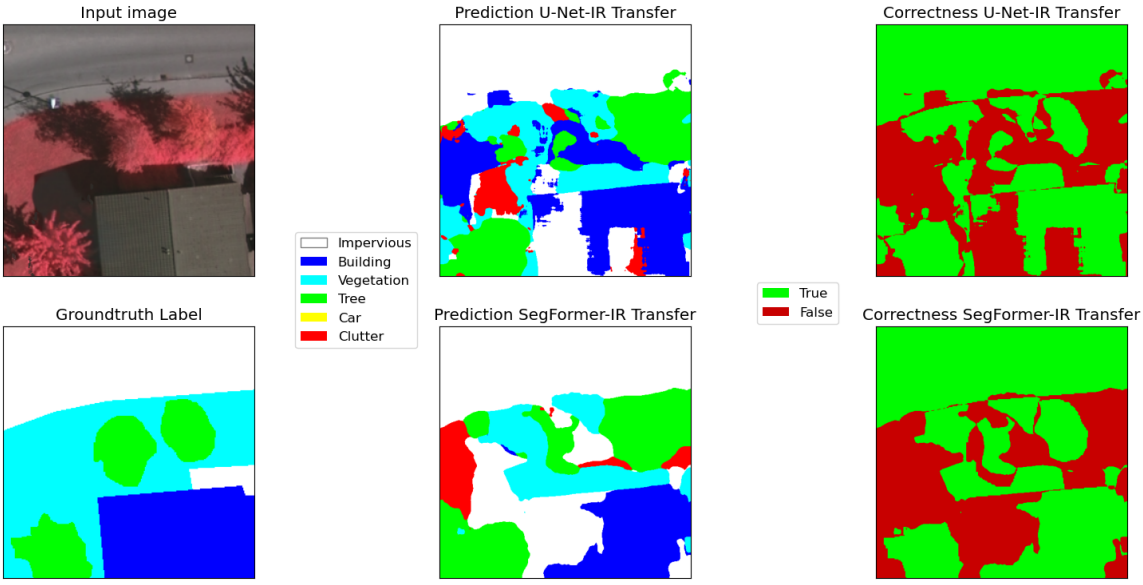
Patch size	mIoU U-Net-IR	mIoU SegFormer-IR
512	0.211	0.178
256	0.206	0.189

4.4.2 Training on another Dataset

In this section the results achieved by training SegFormer and U-Net models on the FloodNet dataset are presented. The results only depict a proof of principle and are not created as systematically as the results on the Potsdam dataset.



(a) Example of applying IRRG models to an 512×512 image patch from the Vaihingen dataset.



(b) Example of applying IRRG models to 256×256 image patch from the Vaihingen dataset resized to a side length of 512 pixels.

Figure 4.12: Examples applying U-Net-IR and SegFormer-Ir to image patches of different sizes from the Vaihingen dataset. Chosen examples are some of the best results achieved compared to other extents.

Training Process

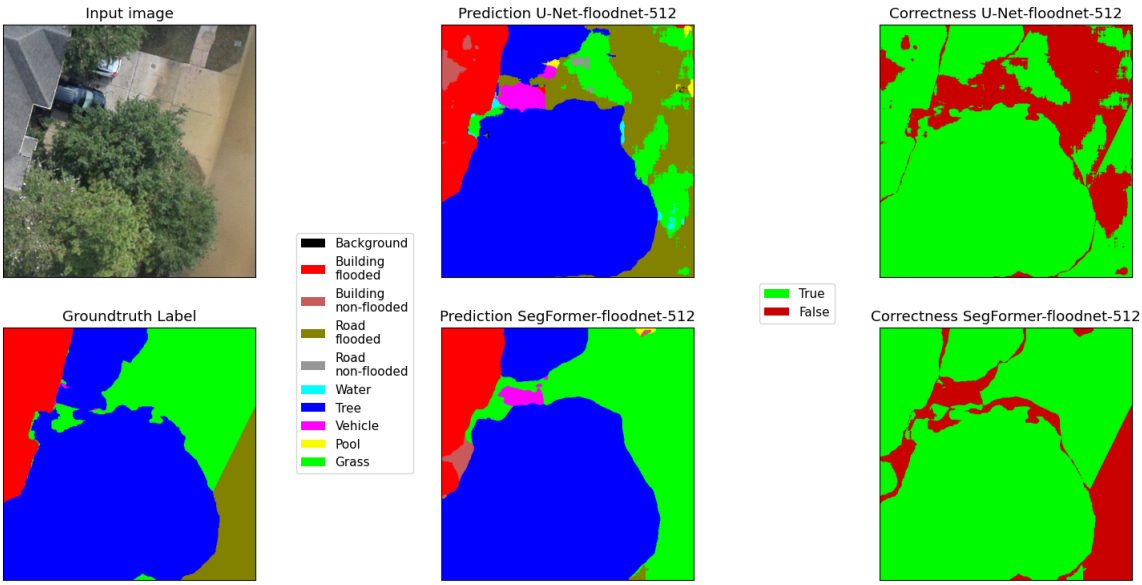
Two model configurations per architecture are trained, one for 512×512 pixel patches and one for 1024×1024 pixel patches. Using Jaccard loss to train the models on the FloodNet dataset led to an overfitting of the models during the training, with two or more classes not being detected by the models in the validation process. After a few epochs, the models did not detect the classes pool and background anymore and after a few more epochs, some of the models just segmented everything as grass with Jaccard loss. This is reasoned by the very imbalanced dataset, where more than 50% of the pixels are assigned to the class grass. Because of this, weighted cross-entropy was used instead to train the FloodNet models.

Qualitative Results

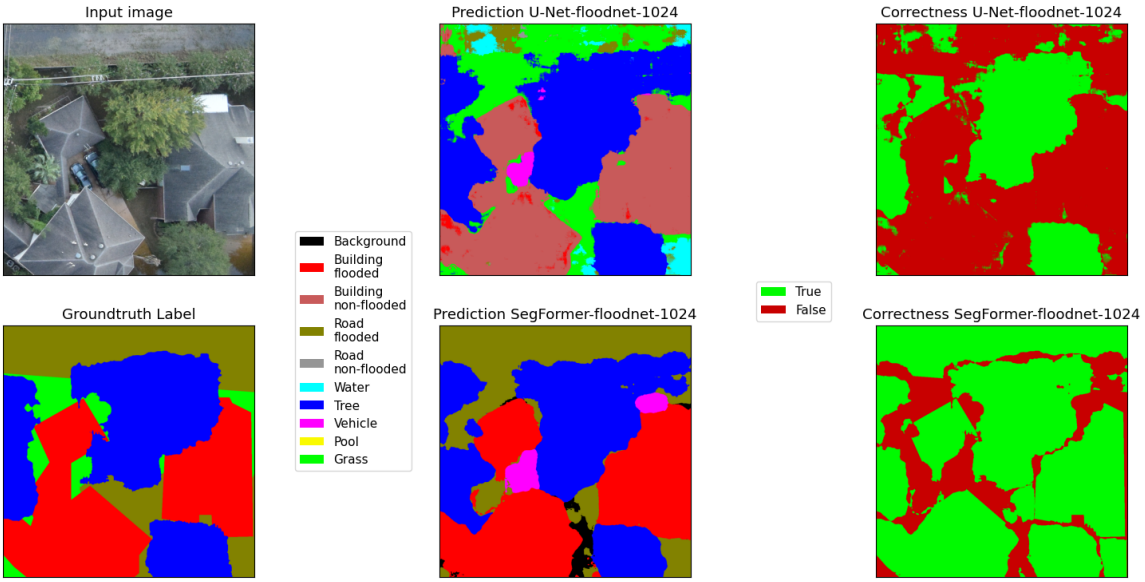
One example per model configuration is shown in figure 4.13, each showing a flooded extent. Further examples are added to the appendix, with figure B.13 for examples of U-Net-floodnet-512, figure B.14 for SegFormer-floodnet-512, figure B.15 for U-Net-floodnet-1024, and figure B.16 for SegFormer-floodnet-1024.

In the light of FloodNet being the more difficult dataset in terms of the number of classes and their semantic meanings, the results look very promising. In large parts, the predictions are correct. Also, the flooded buildings and streets are segmented correctly oftentimes, with figure 4.13b being one counterexample. U-Net-floodnet-1024 segmented flooded buildings and streets mostly as non-flooded buildings and grass respectively, as was also the case for this model in other examples. Some other examples show correct predictions of flooded buildings although no water is visible in the rest of the image (cf. last image in figures B.14 and B.13). This indicates that the models did not (only) learn the context of a building being flooded when it is surrounded by water but rather how buildings in flooded neighborhoods in the training data looked like. Thus the models are capable of detecting buildings or surroundings that look similar to flooded ones, as the dataset was probably not split spatially.

In the examples, it is again noticeable that U-Net predictions are coarser than the SegFormer predictions which look smoother and thus closer to the ground truth and real-world settings. Except for the coarser edges, the U-Net predictions also consist of more clusters, often wrongly added to larger areas.



(a) U-Net-floodnet-512 and SegFormer-floodent-512 example predictions.



(b) U-Net-floodnet-1024 and SegFormer-floodent-1024 example predictions.

Figure 4.13: Selected examples of predictions by the FloodNet models showing flooded areas. More common examples are added in the appendix.

Quantitative Results

Except U-Net-floodnet-1024 all models achieve very good results with accuaracies above 0.8 and mIoU values bigger than 0.55 as depicted in table 4.2. Still U-Net-floodnet-1024 has good results but as apparent from figure 4.14 it has difficulties in detecting flooded buildings and roads, considerably decreasing the overall metrics. Since all the FloodNet models were not trained as accurately as the Potsdam models, the reason for the significantly worse values of the one model might lie in the training process and can probably be erased by further improving the training settings. As a consequence, the following evaluations are mostly focused on the models trained on the 512-pixel patches.

Quantitatively U-Net-floodnet-512 and SegFormer-floodnet-512 have similar values, overall and per class. A noteworthy difference in the IoU is only present for three of the ten classes, with better values for SegFormer in the class water, and better values for U-Net in the classes vehicle and flooded road. The peak for the grass class in the VoI values for both models is most likely reasoned by the imbalanced dataset where more than 50% of the pixels are labeled as grass. Thus the class has more opportunities of false splits and merges in the predictions.

Table 4.2: mIoU and overall VoI of the models trained on the FloodNet dataset.

	FloodNet 512px		FloodNet 1024px	
	U-Net	SegFormer	U-Net	SegFormer
Accuracy	0.853	0.857	0.789	0.837
mIoU	0.599	0.586	0.413	0.579
VoI	0.538	0.447	0.802	0.630

Figure 4.14: IoU and VoI per class for the models trained on the FloodNet dataset.

5 Discussion

The discussion consists mainly of two parts. First the comparison of the architectures based on the previously presented results and their interpretation regarding the guiding question and hypotheses (cf. section 1.3). The second part evaluates the methodology and states some of the limitations of this work.

5.1 Model Comparison

After presenting the qualitative and quantitative results independently in the previous chapter, in this section, their connection with each other is discussed. Moreover advantages of each of the architectures are highlighted with further attempts of finding reasons for the differences between them.

5.1.1 Similarities

Overall both architectures perform well across all model configurations, datasets and semantic classes. Each model has an accuracy on the test dataset above 75% and the lowest mIoU value among them is 0.578 which are decent values. The values could probably even be improved for example by means of a more extensive hyperparameter tuning, data augmentation or transfer learning. All models have their difficulties in detecting the class clutter, qualitatively as well as quantitatively, but this is most likely reasoned by the diversity of the class that covers everything that can not be assigned to any of the other classes. The classes vegetation and tree have the second and third worst values for all models which is reasoned by the models segmenting about 20% of trees as vegetation (cf. tables A.2 and A.1). Otherwise, no significant systematic errors in the prediction of any class or type of object can be observed except for some model specifics discussed in the next sections.

5.1.2 Advantages of SegFormer

The qualitative results on average look better for SegFormer as they are more homogenous than the U-Net results. There are many examples where U-Net segments larger areas mostly correct but adds clusters of other classes to them, from a human perspective often without any visible reason. This is also reflected in the VoI where all SegFormer models except SegFormer-S have smaller and thus better values. Especially the false splits from the U-Net models affect the VoI strongly in favor of the SegFormer models. Moreover, the edges of clusters from U-Net models are coarser than in SegFormer predictions. Thus SegFormer predictions on average are more consistent and better represent real-world scenarios than U-Net predictions.

Although the U-Net models outperform the SegFormer models in accuracy and mIoU in all configurations, it is interesting to see that the metrics for the SegFormer models improve with larger patches, thus approaching the same values of the U-Net models. This could be an indicator of SegFormer benefitting from a larger effective receptive field on large images, enabling the inclusion of a more global context by use of self-attention compared to the plain convolutions of U-Net.

Moreover, SegFormer also benefits from the use of the IR band instead of the blue band. This is apparent from SegFormer-IR, trained on 512-pixel IRRG-patches, having a slightly better mIoU than SegFormer-512, whereas U-Net-IR has worse mIoU and VoI values compared to U-Net-512.

5.1.3 Advantages of U-Net

As mentioned in the previous section, all U-Net models outperform the corresponding SegFormer models regarding accuracy and mIoU and this advantage is more distinct for smaller image patches. For the calculation of the metrics, the U-Net models benefit from their inhomogeneous segmentations. When U-Net detects large parts of an object wrong it often still detects some of it correctly with smaller clusters, whereas SegFormer models often segment the object as a whole wrong. This improves the IoU metrics for U-Net by assigning multiple pixels to the correct class although the model misinterprets large parts of an area. Moreover, the focus on more local information also gives U-Net models an advantage in segmenting cropped objects at the edges better while SegFormer sometimes seems to struggle because of lacking context. Similarly, small distinct objects, for example trees or clutter, are separated better by U-Net models as SegFormer models put some of them together into one class.

Moreover, it is interesting to see, that the IoU advantage of U-Net is bigger for classes covering averagely smaller objects, like clutter and tree. The IoU values for the classes covering larger areas like impervious surfaces, buildings and vegetation, are closer to the ones of the SegFormer models.

Training on a slim dataset seems to be easier for U-Net. Overall the results for both respective models are good with both models having a similar decrease in their IoU. The models are the only ones where the VoI is bigger for SegFormer than for U-Net. That the values of the metrics get worse for fewer training data makes sense as the models do not know as many different examples they could learn from. Further investigations regarding the size of the training dataset that is sufficient for satisfactory results would be interesting.

Independently from the actual results, U-Net also has the advantage of being easier to train. It yields good results with a wider range of learning rates and reaches its optimum after fewer epochs than SegFormer on average.

5.1.4 Transferability

As a case study, the IRRG models were applied to the Vaihingen dataset since it was the only dataset available with the same labels as the training data. As the results show the models were not very useful for this application, probably reasoned by the differences in the datasets. To make the models more useful for application to other areas the simplest solution is to apply it only to images that are more similar to the training data, i.e. data with a similar ground sampling distance and taken at the same season. Another idea is to make the models more robust against variations in the data by adding augmented data within the training process or using data from multiple datasets for training.

The generalizability of the results is evaluated by training both architectures on the FloodNet dataset with two different configurations. Especially the models trained on the 512×512 pixel data show similar results to the models trained on the Potsdam data regarding their accuracy and mIoU. The qualitative results also look very promising, where the models are even able to differentiate flooded and non-flooded buildings as well as flooded streets and usual water areas and pools. It is questionable which information causes the correct segmentation of flooded buildings, as sometimes buildings are segmented correctly as flooded without any context of flood visible in the image. Presumably, the data is not split spatially, thus images with very similar structures, for example from the neighborhoods, are part of the training data.

Another issue with the results is the quality of the dataset. Although it covers a very interesting topic, the dataset often contains similar images that were taken by the UAV only a few meters apart. Moreover, it was found during the review of the dataset and results, that the assignment of the labels was not as neat as for the Potsdam dataset. Sometimes larger areas were entirely assigned to the wrong class. One example is the second to last image in figures B.15 and B.16 where the grass area at the top is assigned to the class water. Since SegFormer-floodnet-1024 segments the area corresponding to the label correctly as water, probably in the training dataset similar examples

with wrong labels are included. In the scope of the superficial proof of principle, it was out of scope to review the whole dataset and filter such images.

Still, the results achieved by training the models on the FloodNet dataset show that similar principles apply to them as to the Potsdam dataset. For the 512-pixel models, that were trained more systematically on the FloodNet data, U-Net performs slightly better quantitatively but tends to over-segmentation compared to SegFormer.

5.2 Limitations and Method Evaluation

Working upon such a broad topic as defined by the title of this thesis some limitations have to be accepted as not every aspect can be covered. Some of the limitations are discussed in the following and the dataset and approach are evaluated.

5.2.1 Dataset

Limitations regarding the dataset are only discussed regarding the Potsdam dataset which was used for most evaluations. Thus the first limitation is that mainly just one dataset was involved and the outcomes are only valid for it. The dataset only included six classes of urban environments with a ground sampling distance of 5cm. To get an impression of the generalizability of the results for other data, some basic tests were also done on the FloodNet dataset. But also the FloodNet dataset contains high-resolution imagery and the validity of the results for lower resolutions, like most satellite images provide, has to be proven.

Further limitations of the dataset arise from the true orthophotos that have the advantage of being geometrically corrected but this leads to side effects for some objects, especially larger buildings. There are multiple examples where fragments of buildings are left at their original position instead of being removed when corrected. These fragments are neither valid buildings nor part of the surrounding class so they need to be assigned to clutter (cf. figure 4.6a), which might lead to difficulties for the models during training.

Moreover, the semantic meaning of the class labels themselves can be problematic, for example, flat roofs which can also be interpreted as being impervious surfaces or greened roofs also being low vegetation. On the other hand, with these definitions, interesting study cases emerge.

5.2.2 Approach

The applied approach to compare CNNs and Transformers is limited to the comparison of the two specific architectures U-Net and SegFormer. There are various other CNNs and Transformers suitable for semantic segmentation, bringing other characteristics that could be evaluated. Even within the comparison of U-Net and SegFormer many more possibilities arise on the aspects that could be compared. Examples to evaluate are the use of other layer configurations, evaluation of further patch sizes or resized patches, applicability to other data, transfer learning and calculation of additional metrics.

Another limitation of the approach is the biased evaluation of the qualitative results. Reviewing the results I noticed that I rated the wrong segmentation of buildings stronger than other misclassifications. This might be reasoned by the human perspective from which a building is more standing out and less mutable than objects of other classes.

Some of the many other topics that could have been considered within the work are related to a more structured hyperparameter tuning, testing of other optimizers and loss functions, and merging of cropped image patches after the prediction.

Further limitations of the approach refer to extensions of the current work and are formulated as an outlook in the respective section.

5.3 Contribution

In summary and referring back to the guiding question and hypotheses (cf. section 1.3) U-Net and SegFormer as representatives of CNN and Transformer architectures respectively, both perform well for semantic segmentation of remote sensing images. U-Net outperforms SegFormer regarding accuracy measures but SegFormer produces smoother results that are qualitatively closer to real-world settings, especially for larger input images and classes covering larger areas. Both models are able to learn satisfactorily from a slim dataset and perform similarly on a dataset with other properties.

Overall I think the applied methods worked well for their purpose of evaluating differences between CNN- and Transformer-based deep learning models. Especially considering my barely existing prior knowledge of deep learning architectures many interesting results were created.

6 Conclusion and Outlook

In this final chapter, the work will be concluded by summarizing the most important findings and giving an outlook on aspects worth further investigation in the future.

6.1 Conclusion

By training U-Net and SegFormer models on remote sensing data with different configurations, this research aimed to identify differences between CNNs and Transformers for semantic segmentation. Based on the presented qualitative and quantitative results it can be concluded that both architectures are very well suited for this task with the main differences of U-Net predictions having a higher Intersection over Union to the ground truth but SegFormer results being more homogenous, which is also reflected in the Variation of Information metric.

The SegFormer advantage of more homogenous segmentation is particularly apparent for larger areas and buildings where false segmentations are more outstanding from a human perspective. Moreover, SegFormer models benefit more from increasing the image patch sizes for training and application compared to U-Net models. With larger patch sizes, for SegFormer especially the IoU of classes covering larger areas like impervious surfaces, buildings and vegetation increase. These findings indicate that SegFormer as well as Transformers generally take advantage of the self-attention mechanism, incorporating more global context into the data processing compared to the local features that the plain convolutions of CNNs are extracting.

Further investigations showed that both architecture's performances suffered from being trained on a slim dataset as expected, yet both still worked well with the investigated data. Both architectures were not suitable for simply being applied to a dataset with other properties. Otherwise, the results have shown to be generalizable beyond the Potsdam dataset, by successfully training and applying both of the architectures to the FloodNet dataset, which led to comparable results.

To sum up, I can say that I learned a lot during the work on this topic and am satisfied with the presented results showing that it is difficult to overcome CNNs and especially the U-Net in the

domain of semantic segmentation of remote sensing images, but that SegFormer and Transformers in general are a worthy competitor meriting further research.

6.2 Outlook

The presented results and the methods applied leave some room for further investigations and future work.

First, the methods applied could be adapted to further improve the model predictions which could be achieved by adjusting the training procedure. The settings for the training could be changed for example by using a learning rate scheduler or other means to find more useful learning rates. Other optimizers could be tested, as well as other methods for normalization and weight initialization. Furthermore, the data preparation could be complemented by data augmentation or the addition of data from other datasets. Fine-tuning or transfer learning of existing models that are already trained on another dataset provides another interesting opportunity for further improvements of the models. This could especially be interesting for the comparison of CNNs and Transformers since Transformers are said to benefit a lot from pre-training on a large dataset. Instead of adding more data to the training, it could be interesting to train the models on even less data than the tested slim dataset, to evaluate which architecture is better suited for training on very small datasets.

A second topic to extend is the evaluation of the models. Additional metrics could be calculated and evaluated as well as the training and inference time. Another interesting research direction for evaluation could be to focus on the explainability of the models. First, it could be evaluated if the decisions of the models can be explained at all, for example by visualizing individual layers or the effective receptive fields. If this is possible and rewarding, secondly, the insights could be utilized to explain the current findings in more detail.

The third topic worth further exploration is the comparison of other model architectures. This could include entirely other architectures than U-Net and SegFormer but it could also be interesting to adapt the investigated models to the application on multispectral data, i.e. more than three input channels.

In the fast-evolving field of deep learning architectures and the current trend of Transformers in computer vision, regularly new architectures worth evaluating will emerge.

Bibliography

[1] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015, Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.1505.04597 (cit. on pp. 1, 18, 23, 24).

[2] X. Yuan, J. Shi, and L. Gu, "A review of deep learning methods for semantic segmentation of remote sensing imagery," *Expert Systems with Applications*, vol. 169, p. 114 417, May 2021, ISSN: 09574174. DOI: 10.1016/j.eswa.2020.114417 (cit. on pp. 1, 18, 27).

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017, Publisher: arXiv Version Number: 6. DOI: 10.48550/ARXIV.1706.03762 (cit. on pp. 1, 12–14).

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2020, Publisher: arXiv Version Number: 2. DOI: 10.48550/ARXIV.2010.11929 (cit. on pp. 1, 12, 18).

[5] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, Montreal, QC, Canada: IEEE, Oct. 2021, pp. 9992–10 002, ISBN: 978-1-66542-812-5. DOI: 10.1109/ICCV48922.2021.00986 (cit. on p. 1).

[6] R. Strudel, R. Garcia, I. Laptev, and C. Schmid, "Segmenter: Transformer for semantic segmentation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2021, pp. 7262–7272 (cit. on pp. 1, 18).

[7] A. A. Aleissaee, A. Kumar, R. M. Anwer, S. Khan, H. Cholakkal, G.-S. Xia, and F. S. Khan, "Transformers in remote sensing: A survey," *Remote Sensing*, vol. 15, no. 7, p. 1860, Mar. 30, 2023, ISSN: 2072-4292. DOI: 10.3390/rs15071860 (cit. on pp. 1, 2, 27).

[8] L. Ding, D. Lin, S. Lin, J. Zhang, X. Cui, Y. Wang, H. Tang, and L. Bruzzone, "Looking outside the window: Wide-context transformer for the semantic segmentation of high-resolution remote sensing images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–13, 2022, ISSN: 0196-2892, 1558-0644. DOI: 10.1109/TGRS.2022.3168697 (cit. on p. 2).

[9] T. Sourget, S. N. Hasany, F. Mériaudeau, and C. Petitjean, "Can SegFormer be a true competitor to u-net for medical image segmentation?" In *Medical Image Understanding and Analysis*, G. Waiter, T. Lambrou, G. Leontidis, N. Oren, T. Morris, and S. Gordon, Eds., vol. 14122, Series Title: Lecture Notes in Computer Science, Cham: Springer Nature Switzerland, 2024, pp. 111–118, ISBN: 978-3-031-48592-3 978-3-031-48593-0. DOI: 10.1007/978-3-031-48593-0_8 (cit. on p. 2).

[10] I. Goodfellow and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org (cit. on pp. 5, 7, 9, 16, 17).

[11] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958, ISSN: 1939-1471, 0033-295X. DOI: 10.1037/h0042519 (cit. on p. 5).

[12] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach* (Prentice Hall series in artificial intelligence), Third edition, Global edition, in collab. with E. Davis and D. Edwards. Pearson, 2016, 1132 pp., ISBN: 978-0-13-604259-4 978-1-292-15396-4 (cit. on p. 5).

[13] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*, Second edition. Beijing [China] ; Sebastopol, CA: O'Reilly Media, Inc, 2019, 819 pp., ISBN: 978-1-4920-3264-9 (cit. on pp. 6, 9, 11).

[14] C. Bishop, *Pattern Recognition And Machine Learning*, 1st ed. 2006, ISBN: 978-1-4939-3843-8 (cit. on p. 6).

[15] K. P. Murphy, *Machine learning: a probabilistic perspective* (Adaptive computation and machine learning series). Cambridge, MA: MIT Press, 2012, 1067 pp., ISBN: 978-0-262-01802-9 (cit. on p. 7).

[16] H. Habibi Aghdam and E. Jahani Heravi, *Guide to Convolutional Neural Networks*. Cham: Springer International Publishing, 2017, ISBN: 978-3-319-57549-0. DOI: 10.1007/978-3-319-57550-6 (cit. on pp. 7, 10).

[17] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 28, 2015, ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature14539 (cit. on pp. 7, 9, 10).

[18] A. W. Harley, "An interactive node-link visualization of convolutional neural networks," in *ISVC*, 2015, pp. 867–877. [Online]. Available: https://adamharley.com/nn_vis/ (visited on 09/04/2023) (cit. on p. 8).

[19] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," 2016, Publisher: arXiv Version Number: 2. DOI: 10.48550/ARXIV.1603.07285 (cit. on p. 8).

[20] A. Karpathy. "Stanford CS class CS231n: Convolutional neural networks for visual recognition." (2015), [Online]. Available: https://cs231n.github.io/convolutional-networks/ (visited on 09/01/2023) (cit. on pp. 9, 10).

[21] 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 (CVPR)*, Jun. 2015 (cit. on pp. 11, 18, 29).

[22] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM Computing Surveys*, vol. 54, no. 10, pp. 1–41, Jan. 31, 2022, ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3505244 (cit. on p. 12).

[23] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016, Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.1607.06450 (cit. on p. 13).

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015, Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.1512.03385 (cit. on p. 13).

[25] M. Raghu, T. Unterthiner, S. Kornblith, C. Zhang, and A. Dosovitskiy, "Do vision transformers see like convolutional neural networks?," 2021, Publisher: arXiv Version Number: 2. DOI: 10.48550/ARXIV.2108.08810 (cit. on p. 14).

[26] M. Naseer, K. Ranasinghe, S. Khan, M. Hayat, F. S. Khan, and M.-H. Yang, "Intriguing properties of vision transformers," 2021, Publisher: arXiv Version Number: 3. DOI: 10.48550/ARXIV.2105.10497 (cit. on p. 15).

[27] J.-B. Cordonnier, A. Loukas, and M. Jaggi, "On the relationship between self-attention and convolutional layers," 2019, Publisher: arXiv Version Number: 2. DOI: 10.48550/ARXIV.1911.03584 (cit. on p. 15).

[28] H. Thisanke, C. Deshan, K. Chamith, S. Seneviratne, R. Vidanaarachchi, and D. Herath, "Semantic segmentation using vision transformers: A survey," 2023, Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.2305.03273 (cit. on p. 15).

[29] M. Yeung, E. Sala, C.-B. Schönlieb, and L. Rundo, "Unified focal loss: Generalising dice and cross entropy-based losses to handle class imbalanced medical image segmentation," *Computerized Medical Imaging and Graphics*, vol. 95, p. 102 026, Jan. 2022, ISSN: 08956111. DOI: 10.1016/j.compmedimag.2021.102026 (cit. on p. 16).

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, Publisher: arXiv Version Number: 9. DOI: 10.48550/ARXIV.1412.6980 (cit. on p. 16).

[31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986, ISSN: 0028-0836, 1476-4687. DOI: 10.1038/323533a0 (cit. on p. 16).

[32] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez, "A review on deep learning techniques applied to semantic segmentation," 2017, Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.1704.06857 (cit. on pp. 17, 20).

[33] S. Hu, E. Hoffman, and J. Reinhardt, "Automatic lung segmentation for accurate quantitation of volumetric x-ray CT images," *IEEE Transactions on Medical Imaging*, vol. 20, no. 6, pp. 490–498, Jun. 2001, ISSN: 02780062. DOI: 10.1109/42.929615. [Online]. Available: http://ieeexplore.ieee.org/document/929615/ (visited on 11/15/2023) (cit. on p. 17).

[34] G. Csurka, R. Volpi, and B. Chidlovskii, "Semantic image segmentation: Two decades of research," *Foundations and Trends® in Computer Graphics and Vision*, vol. 14, no. 1, pp. 1–162, 2022, ISSN: 1572-2740, 1572-2759. DOI: 10.1561/0600000095 (cit. on p. 18).

[35] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "SegFormer: Simple and efficient design for semantic segmentation with transformers," 2021, Publisher: arXiv Version Number: 3. DOI: 10.48550/ARXIV.2105.15203 (cit. on pp. 18, 24, 25, 33).

[36] J. Wang, Z. Zheng, A. Ma, X. Lu, and Y. Zhong, "LoveDA: A remote sensing land-cover dataset for domain adaptive semantic segmentation," in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, J. Vanschoren and S. Yeung, Eds., vol. 1, 2021. [Online]. Available: https://datasets-benchmarks%20proceedings.neurips.cc/paper/2021/file/4e732ced3463d06de0ca9a15b6153677-Paper-round2.pdf (cit. on p. 18).

[37] Y. Sun, X. Zhang, Q. Xin, and J. Huang, "Developing a multi-filter convolutional neural network for semantic segmentation using high-resolution aerial imagery and LiDAR data," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 143, pp. 3–14, Sep. 2018, ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2018.06.005 (cit. on p. 18).

[38] Z. Luo, W. Yang, Y. Yuan, R. Gou, and X. Li, "Semantic segmentation of agricultural images: A survey," *Information Processing in Agriculture*, S2214317323000112, Feb. 2023, ISSN: 22143173. DOI: 10.1016/j.inpa.2023.02.001 (cit. on p. 18).

[39] H. Qin, W. Zhou, Y. Yao, and W. Wang, "Individual tree segmentation and tree species classification in subtropical broadleaf forests using UAV-based LiDAR, hyperspectral, and ultrahigh-resolution RGB data," *Remote Sensing of Environment*, vol. 280, p. 113 143, Oct. 2022, ISSN: 00344257. DOI: 10.1016/j.rse.2022.113143 (cit. on p. 18).

[40] G. Camps-Valls, D. Tuia, X. X. Zhu, and M. Reichstein, Eds., *Deep Learning for the Earth Sciences: A Comprehensive Approach to Remote Sensing, Climate Science, and Geosciences*, 1st ed., Wiley, Sep. 27, 2021, ISBN: 978-1-119-64614-3 978-1-119-64618-1. DOI: 10.1002/9781119646181 (cit. on p. 19).

[41] J. E. Ball, D. T. Anderson, and C. S. Chan, "Comprehensive survey of deep learning in remote sensing: Theories, tools, and challenges for the community," *Journal of Applied Remote Sensing*, vol. 11, no. 4, p. 1, Sep. 23, 2017, ISSN: 1931-3195. DOI: 10.1117/1.JRS.11.042609 (cit. on p. 19).

[42] A. Shahtahmassebi, N. Yang, K. Wang, N. Moore, and Z. Shen, "Review of shadow detection and de-shadowing methods in remote sensing," *Chinese Geographical Science*, vol. 23, no. 4, pp. 403–420, Aug. 2013, ISSN: 1002-0063, 1993-064X. DOI: 10.1007/s11769-013-0613-x (cit. on p. 19).

[43] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez, "Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark," in *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, IEEE, 2017. [Online]. Available: https://project.inria.fr/aerialimagelabeling/ (visited on 11/23/2023) (cit. on p. 19).

[44] F. Rottensteiner, G. Sohn, J. Jung, M. Gerke, C. Baillard, S. Benitez, and U. Breitkopf, "The ISPRS benchmark on urban object classification and 3d building reconstruction," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. I-3, pp. 293–298, Jul. 20, 2012, ISSN: 2194-9050. DOI: 10.5194/isprsannals-I-3-293-2012 (cit. on p. 19).

[45] J. Shermeyer, D. Hogan, J. Brown, A. Van Etten, N. Weir, F. Pacifici, R. Hansch, A. Bastidas, S. Soenen, T. Bacastow, and R. Lewis, "SpaceNet 6: Multi-sensor all weather mapping dataset," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2020 (cit. on p. 19).

[46] M. Meilă, "Comparing clusterings—an information based distance," *Journal of Multivariate Analysis*, vol. 98, no. 5, pp. 873–895, May 2007, ISSN: 0047259X. DOI: 10.1016/j.jmva.2006.11.013 (cit. on p. 20).

[47] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao, "Pyramid vision transformer: A versatile backbone for dense prediction without convolutions," 2021, Publisher: arXiv Version Number: 2. DOI: 10.48550/ARXIV.2102.12122 (cit. on p. 26).

[48] X. Chu, Z. Tian, B. Zhang, X. Wang, and C. Shen, "Conditional positional encodings for vision transformers," 2021, Publisher: arXiv Version Number: 3. DOI: 10.48550/ARXIV.2102.10882 (cit. on p. 26).

[49] M. A. Islam, S. Jia, and N. D. B. Bruce, "How much position information do convolutional neural networks encode?," 2020, Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.2001.08248 (cit. on p. 26).

[50] M. Rahnemoonfar, T. Chowdhury, A. Sarkar, D. Varshney, M. Yari, and R. R. Murphy, "FloodNet: A high resolution aerial imagery dataset for post flood scene understanding," *IEEE Access*, vol. 9, pp. 89 644–89 654, 2021, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3090981 (cit. on p. 29).

[51] J. Stenkamp, *Jsten07/CNNvsTransformer: V1.0.0*, version v1.0.0, Jan. 4, 2024. DOI: 10.5281/ZENODO.10460007 (cit. on p. 32).

Appendix A: Quantitative Results

Table A.1: Confusion matrix for SegFormer-512. The values represent the percentage of ground truth pixels of the class on the left segmented as pixels of the class at the top. Thus a value must be read as: "*Value*100 %* of pixels of class *left* are segmented as class *top*." Bold values represent the class's recall.

	Prediction Impervious	Prediction building	Prediction vegetation	Prediction tree	Prediction car	Prediction clutter
Gt impervious	0.903	0.027	0.042	0.016	0.003	0.010
Gt building	0.066	0.910	0.014	0.005	0.000	0.005
Gt vegetation	0.060	0.015	0.840	0.074	0.000	0.011
Gt tree	0.073	0.009	0.233	0.676	0.004	0.006
Gt car	0.091	0.013	0.006	0.013	0.863	0.014
Gt clutter	0.370	0.164	0.158	0.050	0.006	0.252

Table A.2: Confusion matrix for U-Net-512. The values represent the amount of ground truth pixels of the class on the left segmented as pixels of the class at the top. Thus a value must be read as: "*Value*100 %* of pixels of class *left* are segmented as class *top*." Bold values represent the class's recall.

	Prediction Impervious	Prediction building	Prediction vegetation	Prediction tree	Prediction car	Prediction clutter
Gt impervious	0.879	0.029	0.050	0.014	0.003	0.026
Gt building	0.045	0.918	0.019	0.004	0.001	0.014
Gt vegetation	0.055	0.015	0.848	0.056	0.000	0.026
Gt tree	0.053	0.006	0.180	0.748	0.004	0.009
Gt car	0.071	0.012	0.005	0.012	0.887	0.014
Gt clutter	0.247	0.140	0.147	0.022	0.006	0.437

Table A.3: IoU and VoI per semantic class and mIoU, mean VoI and accuracy for models trained and applied to different image sizes and band selections.

	128px		256px		512px		1024px		Slim dataset 512px		IRRG, 512px	
Metrics	U-Net	Seg-Former	U-Net	Seg-Former	U-Net	Seg-Former	U-Net	Seg-Former	U-Net	Seg-Former	U-Net	Seg-Former
Accuracy	0.825	0.779	0.831	0.808	0.836	0.817	0.835	0.831	0.796	0.765	0.820	0.816
mIoU	0.662	0.589	0.670	0.630	0.680	0.638	0.664	0.660	0.619	0.578	0.651	0.642
VoI	0.732	0.701	1.000	0.938	1.224	1.194	1.408	1.388	1.416	1.428	1.321	1.206
IoU impervious	0.748	0.670	0.757	0.738	0.763	0.750	0.762	0.766	0.715	0.696	0.735	0.750
IoU building	0.815	0.769	0.832	0.809	0.844	0.833	0.842	0.855	0.764	0.727	0.804	0.813
IoU vegetation	0.659	0.596	0.661	0.630	0.665	0.643	0.666	0.660	0.630	0.587	0.647	0.635
IoU tree	0.662	0.547	0.666	0.603	0.680	0.596	0.676	0.630	0.595	0.513	0.662	0.625
IoU car	0.798	0.741	0.800	0.768	0.813	0.786	0.812	0.799	0.776	0.730	0.799	0.781
IoU clutter	0.294	0.182	0.303	0.232	0.315	0.218	0.228	0.251	0.234	0.215	0.262	0.248
VoI impervious	0.407	0.396	0.529	0.494	0.633	0.619	0.700	0.673	0.715	0.709	0.697	0.617
VoI building	0.191	0.167	0.271	0.238	0.324	0.300	0.379	0.337	0.440	0.398	0.395	0.325
VoI vegetation	0.432	0.435	0.585	0.563	0.670	0.684	0.747	0.741	0.746	0.794	0.710	0.690
VoI tree	0.342	0.357	0.450	0.487	0.512	0.579	0.566	0.637	0.574	0.695	0.532	0.568
VoI car	0.042	0.052	0.051	0.059	0.054	0.061	0.059	0.065	0.064	0.075	0.058	0.062
VoI clutter	0.152	0.123	0.206	0.173	0.268	0.205	0.257	0.252	0.266	0.249	0.230	0.215

Table A.4: IoU and VoI per semantic class and mIoU, overall VoI and accuracy for models trained on the FloodNet dataset with different patch sizes.

Metrics	512px		1024	
	U-Net	Seg-Former	U-Net	Seg-Former
mIoU	0.599	0.586	0.413	0.579
VoI all classes	0.538	0.447	0.802	0.630
IoU Background	0.015	0.004	0.008	0.303
IoU Building flooded	0.755	0.731	0.048	0.678
IoU Building non-flooded	0.740	0.711	0.364	0.677
IoU Road flooded	0.436	0.333	0.149	0.498
IoU Road non-flooded	0.770	0.784	0.598	0.720
IoU Water	0.513	0.646	0.518	0.602
IoU Tree	0.801	0.791	0.744	0.790
IoU Vehicle	0.582	0.479	0.427	0.366
IoU Pool	0.540	0.565	0.483	0.350
IoU Grass	0.838	0.818	0.792	0.802
VoI Background	0.020	0.016	0.046	0.037
VoI Building flooded	0.044	0.038	0.084	0.079
VoI Building non-flooded	0.045	0.043	0.164	0.090
VoI Road flooded	0.097	0.065	0.102	0.088
VoI Road non-flooded	0.095	0.079	0.178	0.106
VoI Water	0.129	0.104	0.242	0.143
VoI Tree	0.245	0.228	0.363	0.286
VoI Vehicle	0.010	0.008	0.020	0.019
VoI Pool	0.011	0.010	0.015	0.020
VoI Grass	0.422	0.380	0.581	0.470

Appendix B: Qualitative Results

In the following figures, multiple predictions per model are visualized without being evaluated and commented on. Unlike in the results section, here not the comparison between models is depicted but several examples per model. The figures per model are:

- Figure B.1: U-Net-128
- Figure B.2: SegFormer-128
- Figure B.3: U-Net-256
- Figure B.4: SegFormer-256
- Figure B.5: U-Net-512
- Figure B.6: SegFormer-512
- Figure B.7: U-Net-1024
- Figure B.8: SegFormer-1024
- Figure B.9: U -Net-S
- Figure B.10: SegFormer-S
- Figure B.11: U-Net-IR
- Figure B.12: SegFormer-IR
- Figure B.13: U-Net-floodnet-512
- Figure B.14: SegFormer-floodnet-512
- Figure B.15: U-Net-floodnet-1024
- Figure B.16: SegFormer-floodnet-1024
- Figure B.17: U-Net-IR applied to 512×512 image patches from the Vaihingen dataset
- Figure B.18: U-Net-IR applied to 512×512 image patches from the Vaihingen dataset

- Figure B.19: U-Net-IR applied to 256 × 256 image patches from the Vaihingen dataset
- Figure B.20: SegFormer-IR applied to 256 × 256 image patches from the Vaihingen dataset

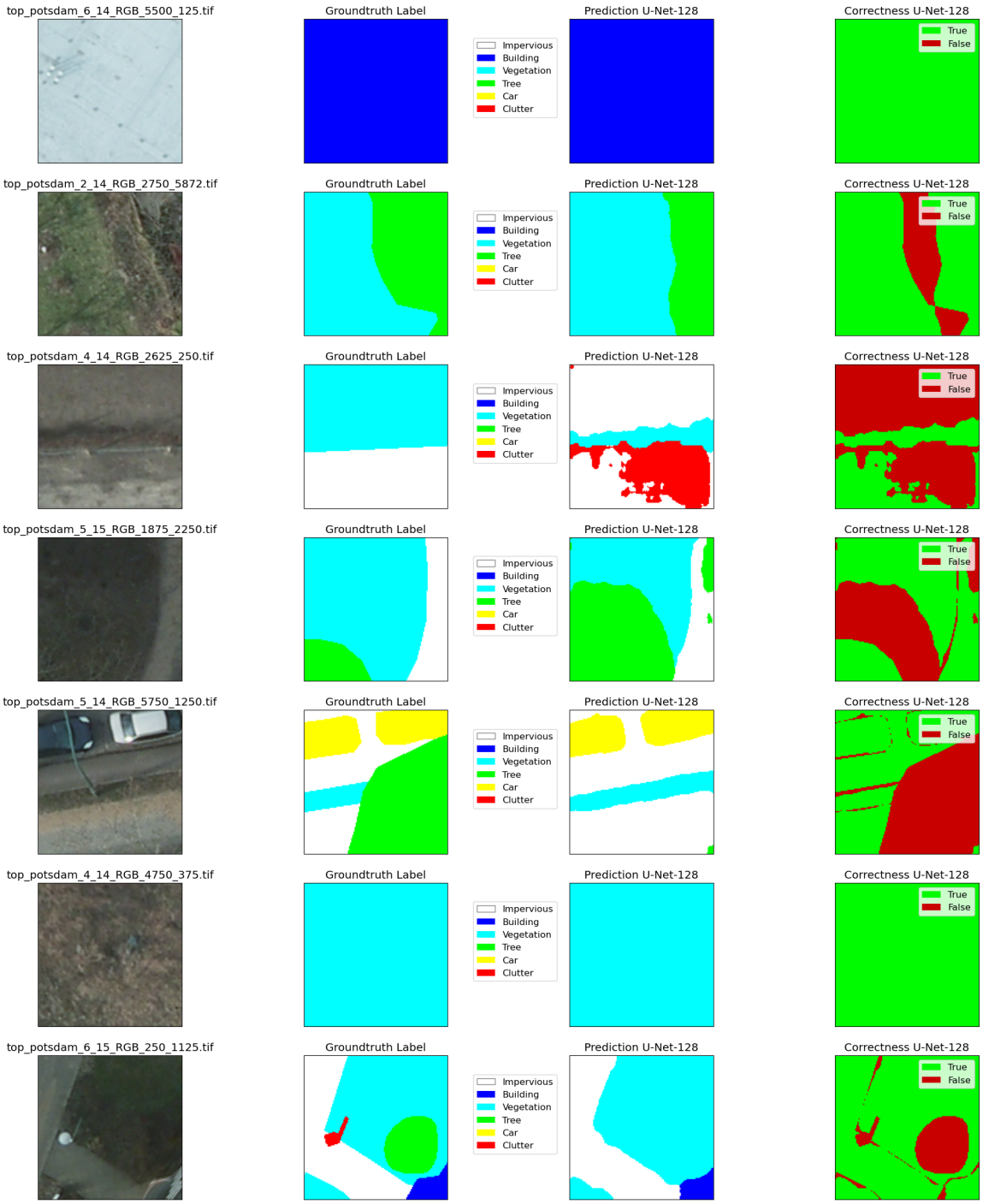


Figure B.1: Examples for model U-Net-128.

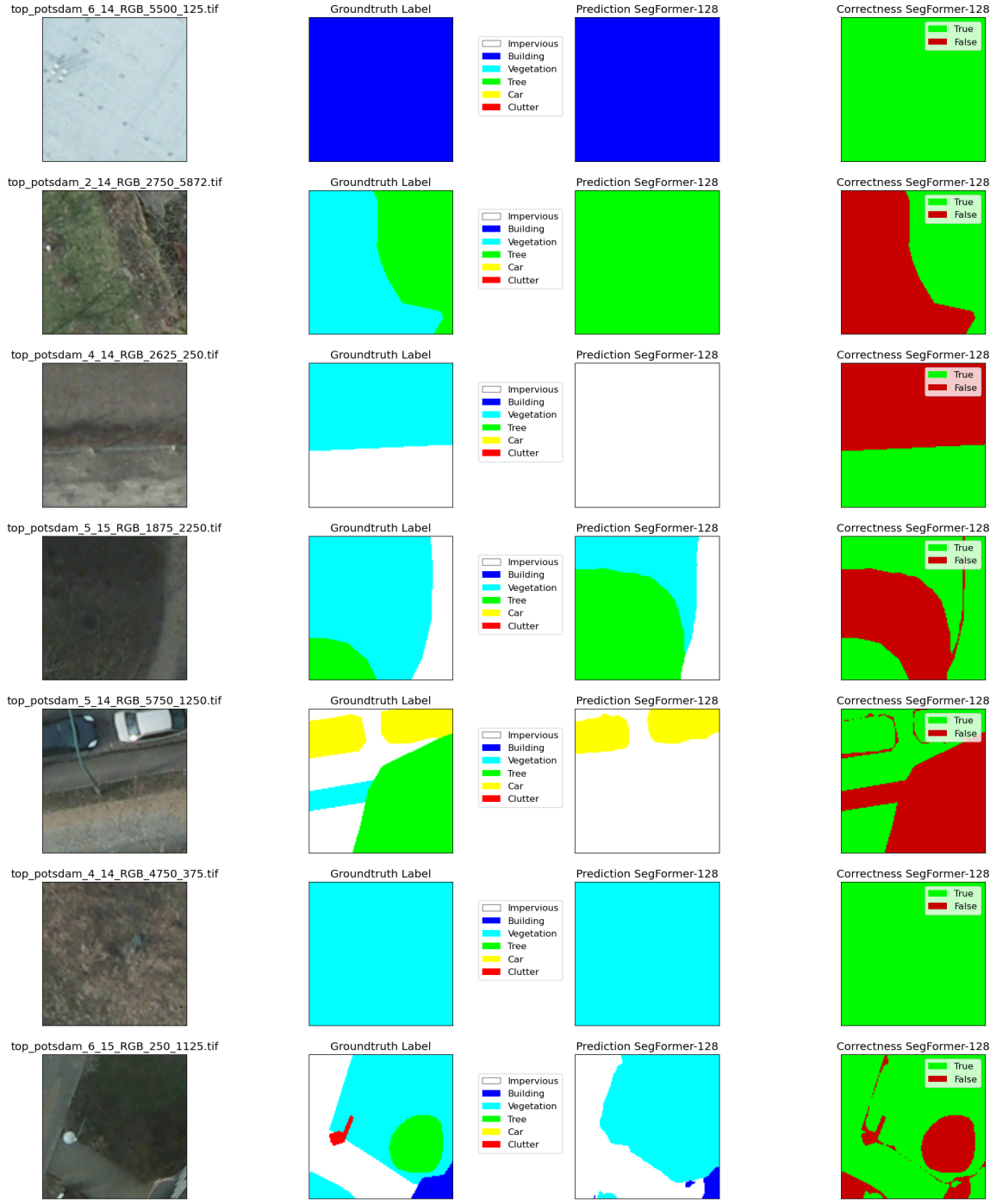


Figure B.2: Examples for model SegFormer-128.

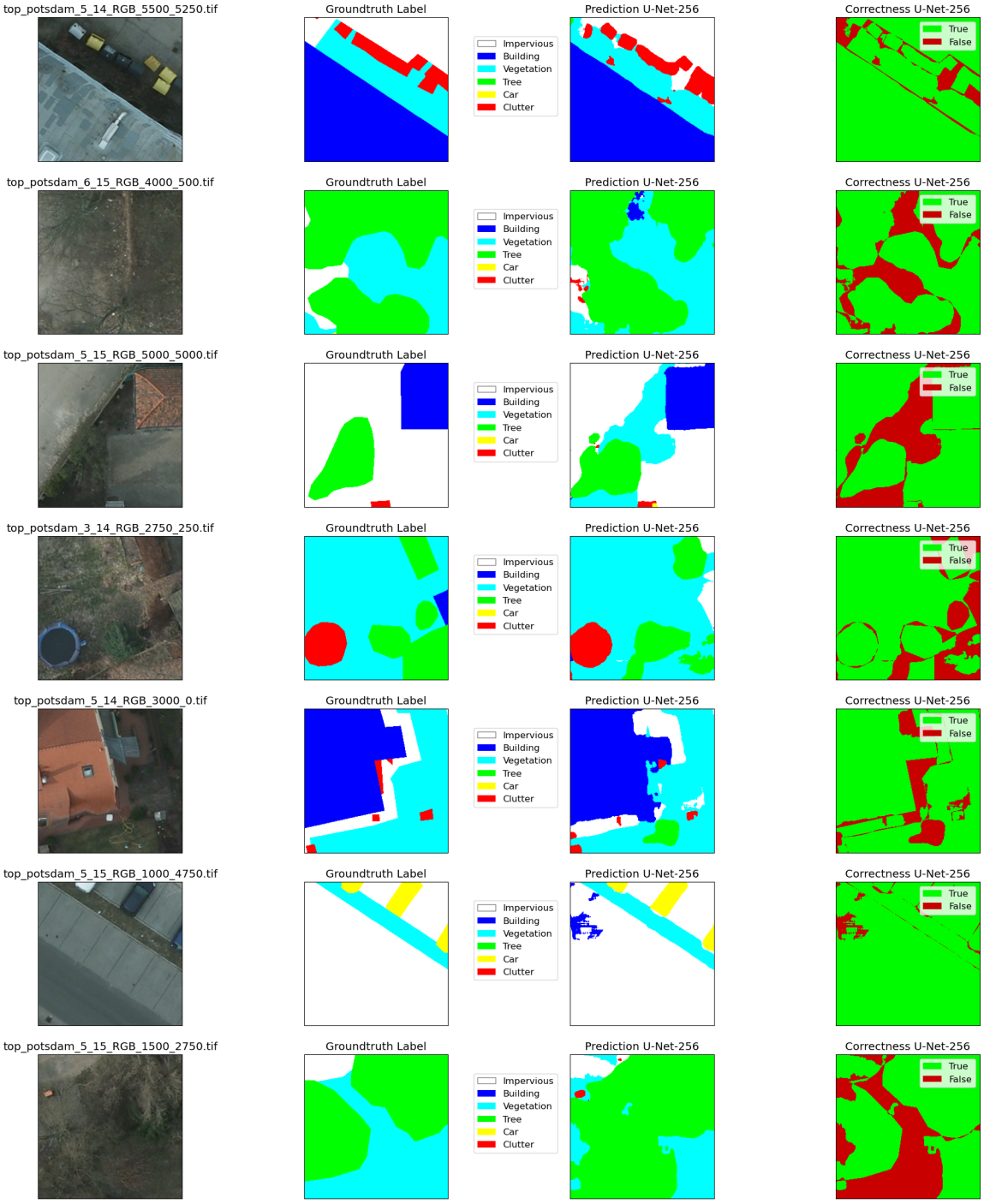


Figure B.3: Examples for model U-Net-256.

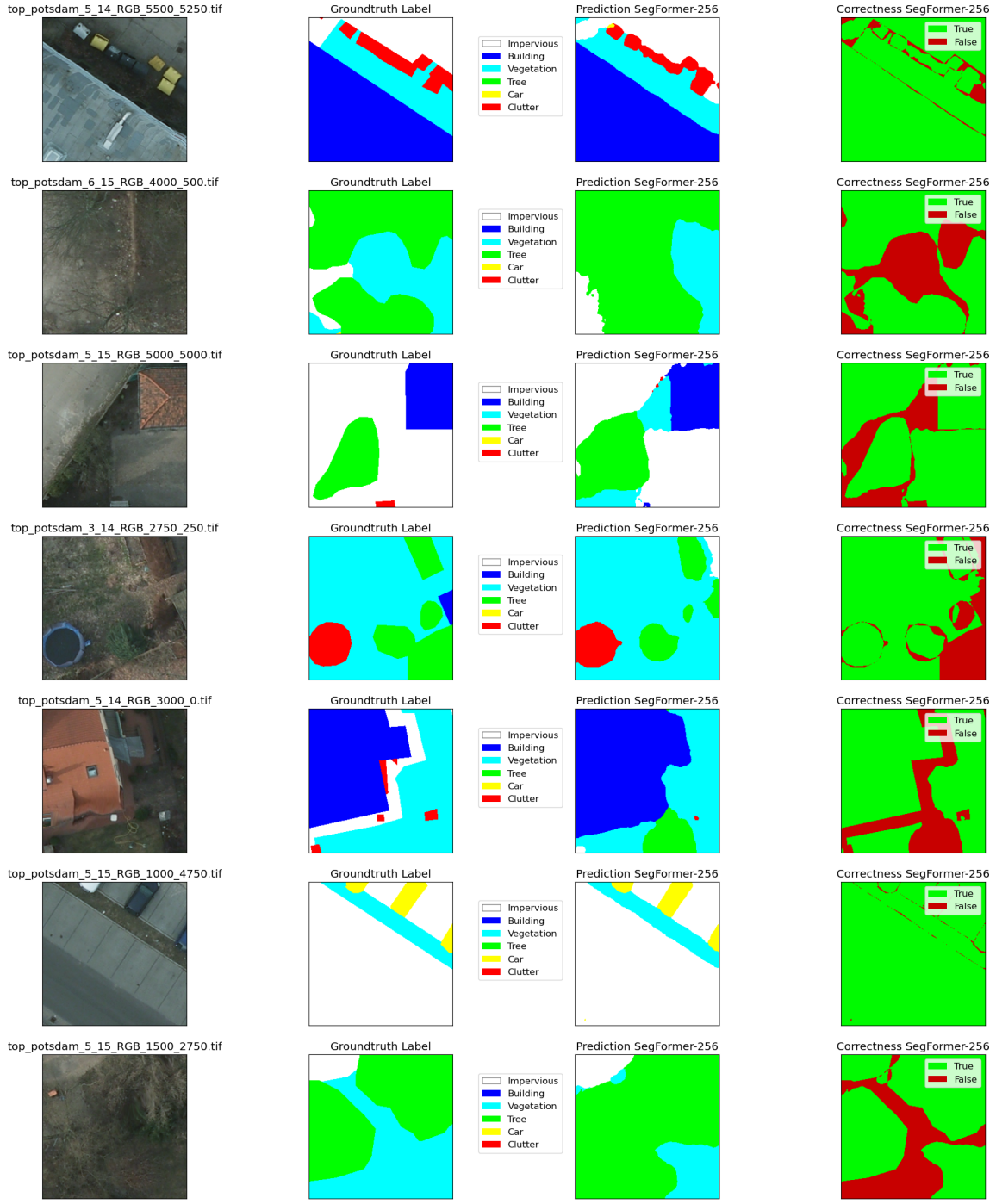


Figure B.4: Examples for model SegFormer-256.

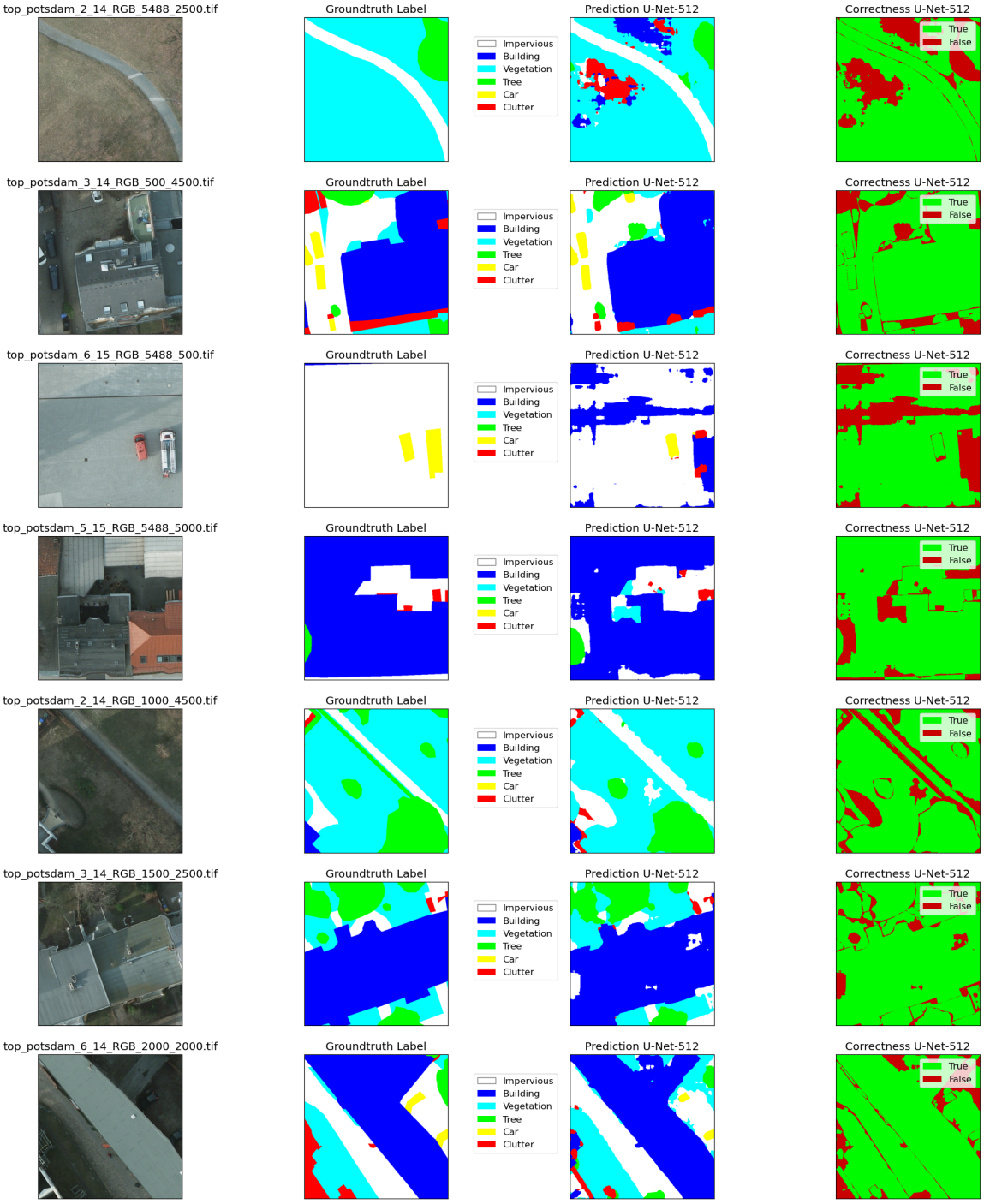


Figure B.5: Examples for model U-Net-512.

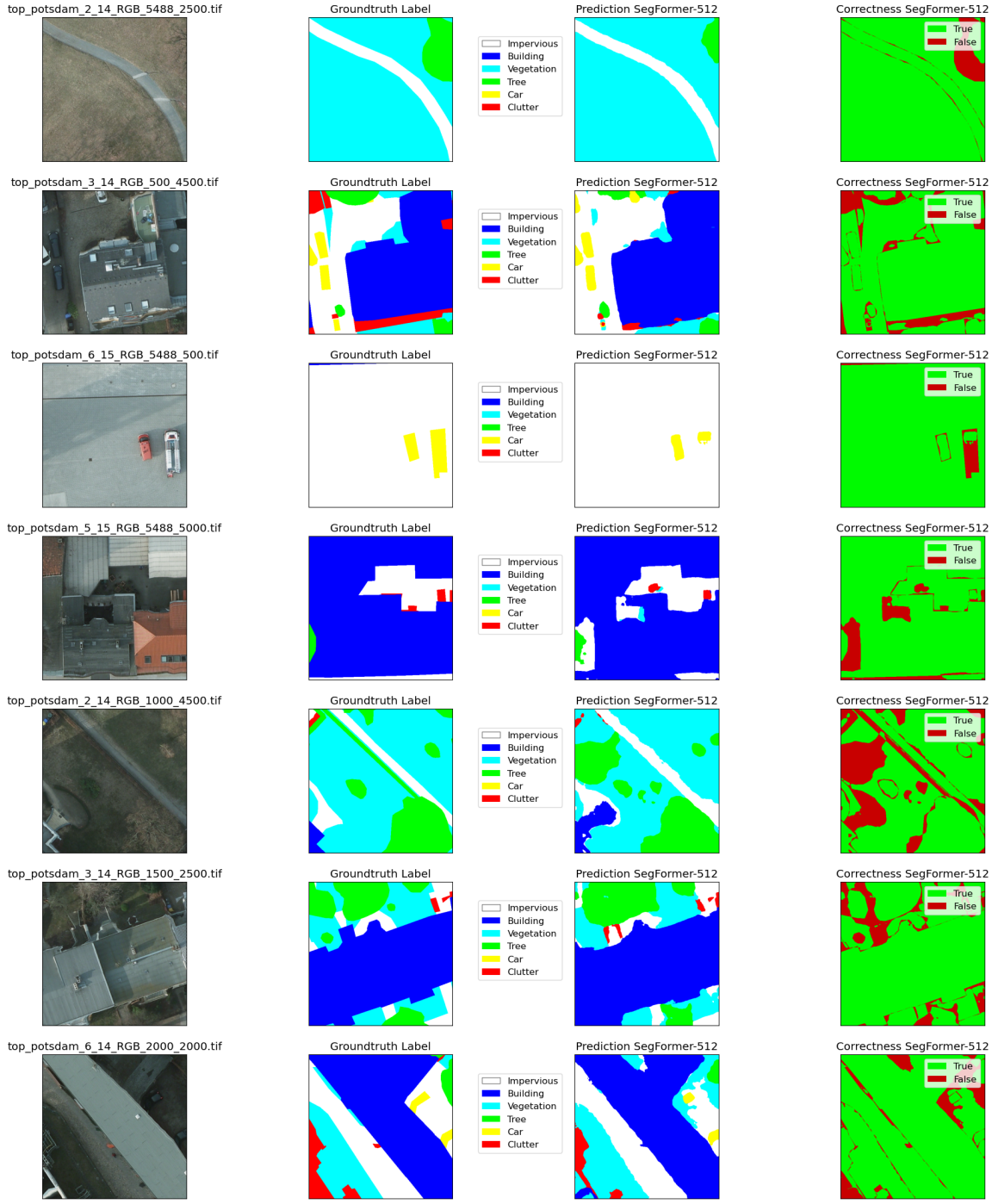


Figure B.6: Examples for model SegFormer-512.

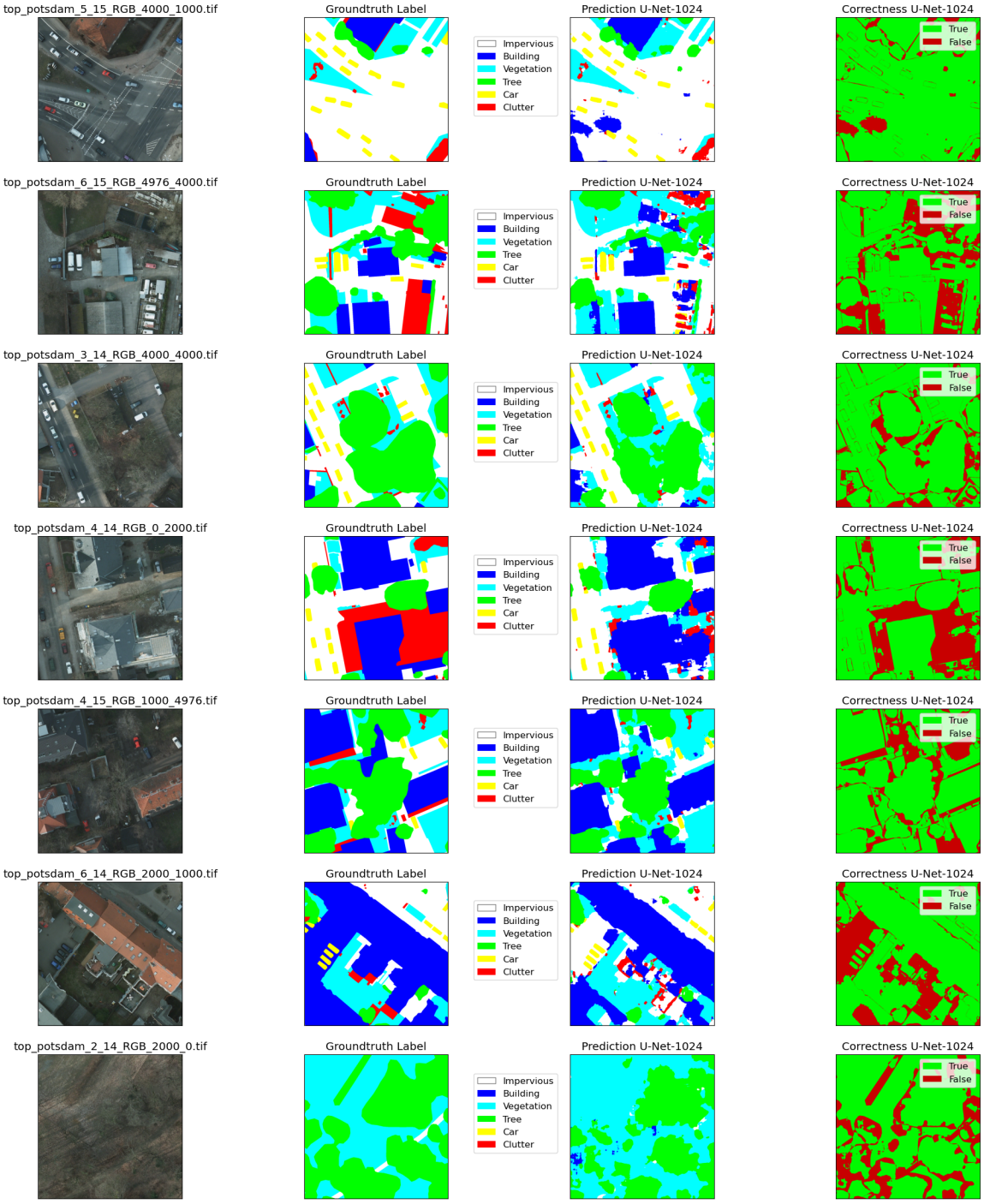


Figure B.7: Examples for model U-Net-1024.

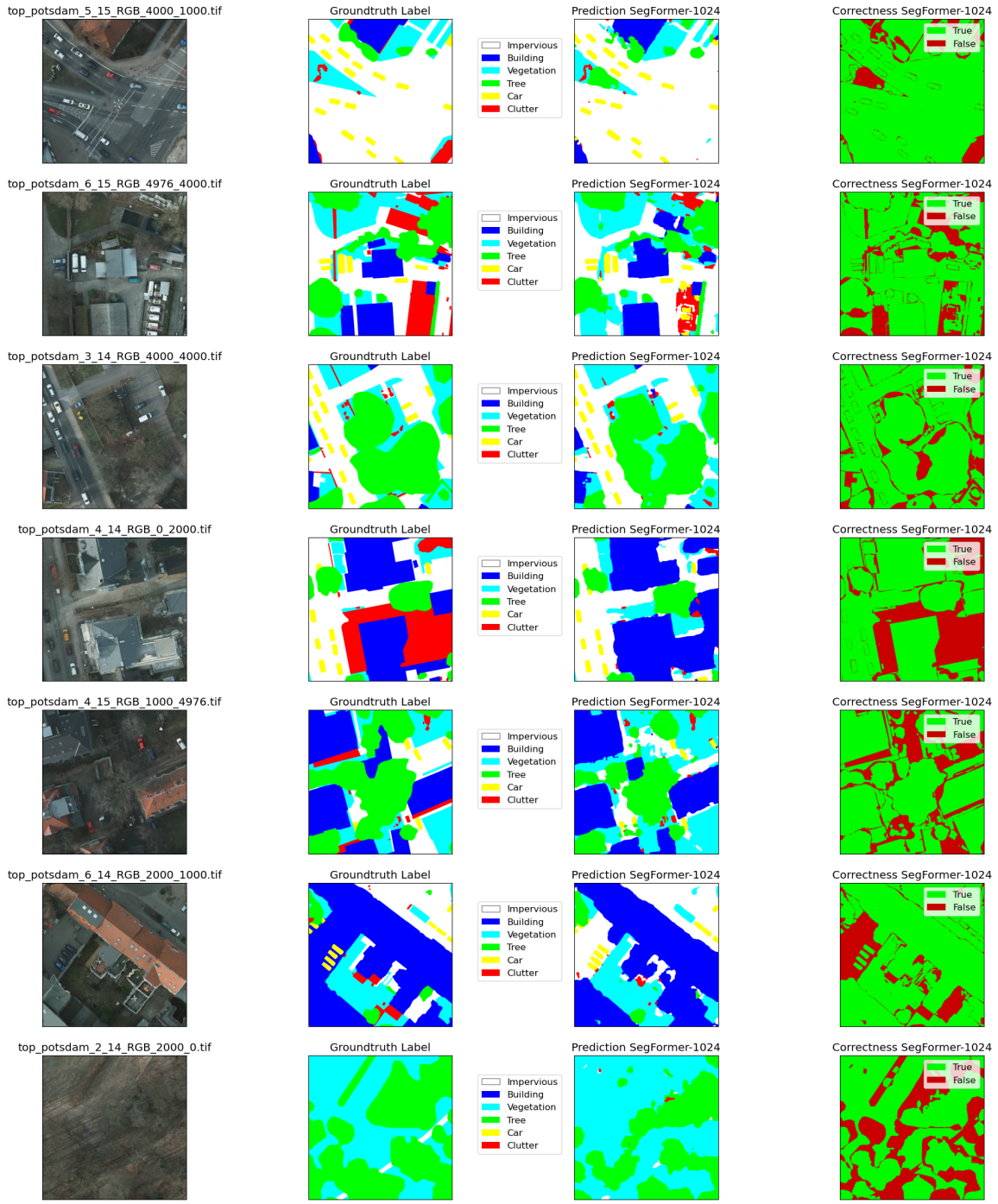


Figure B.8: Examples for model SegFormer-1024.

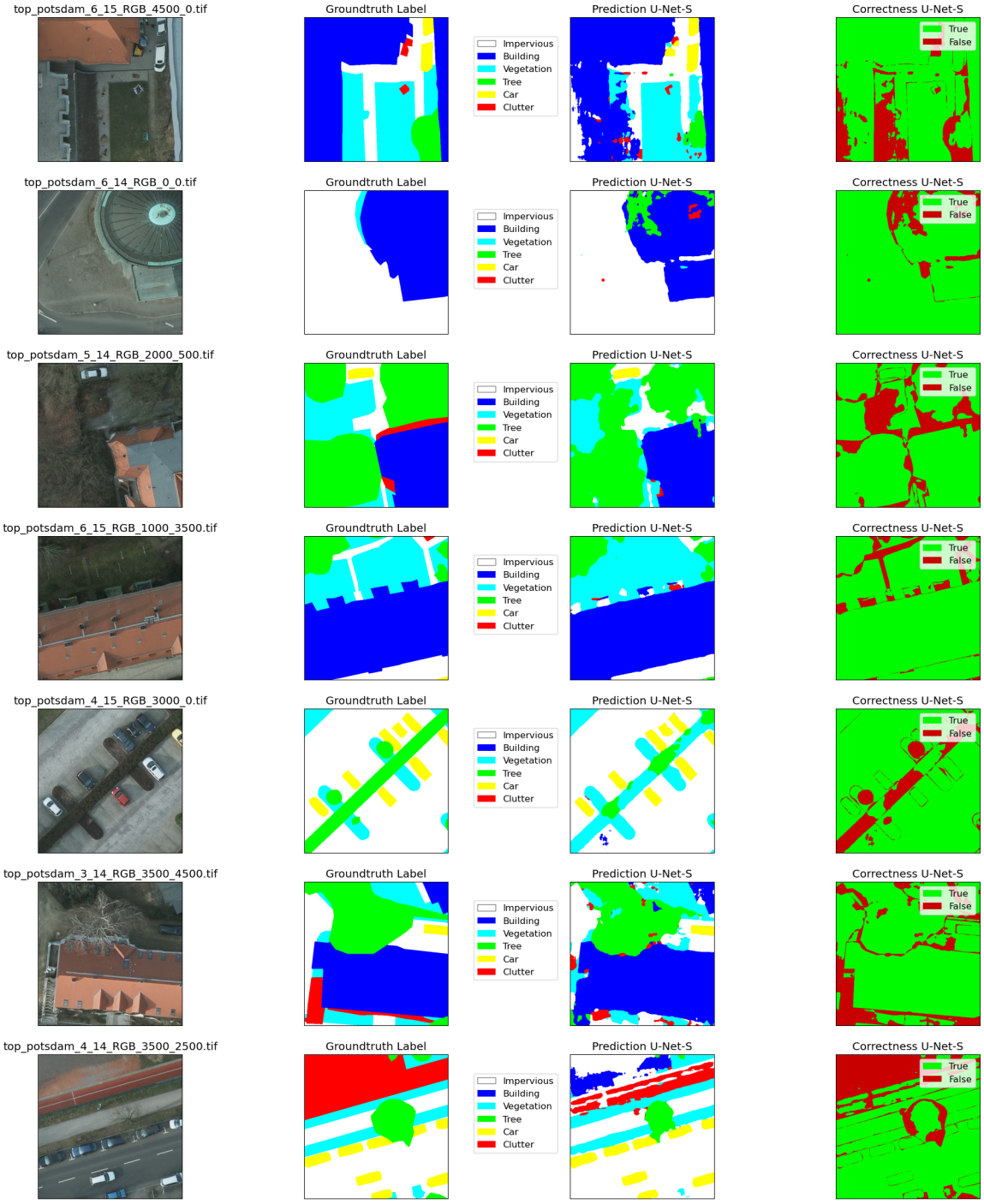


Figure B.9: Examples for model U-Net-S.

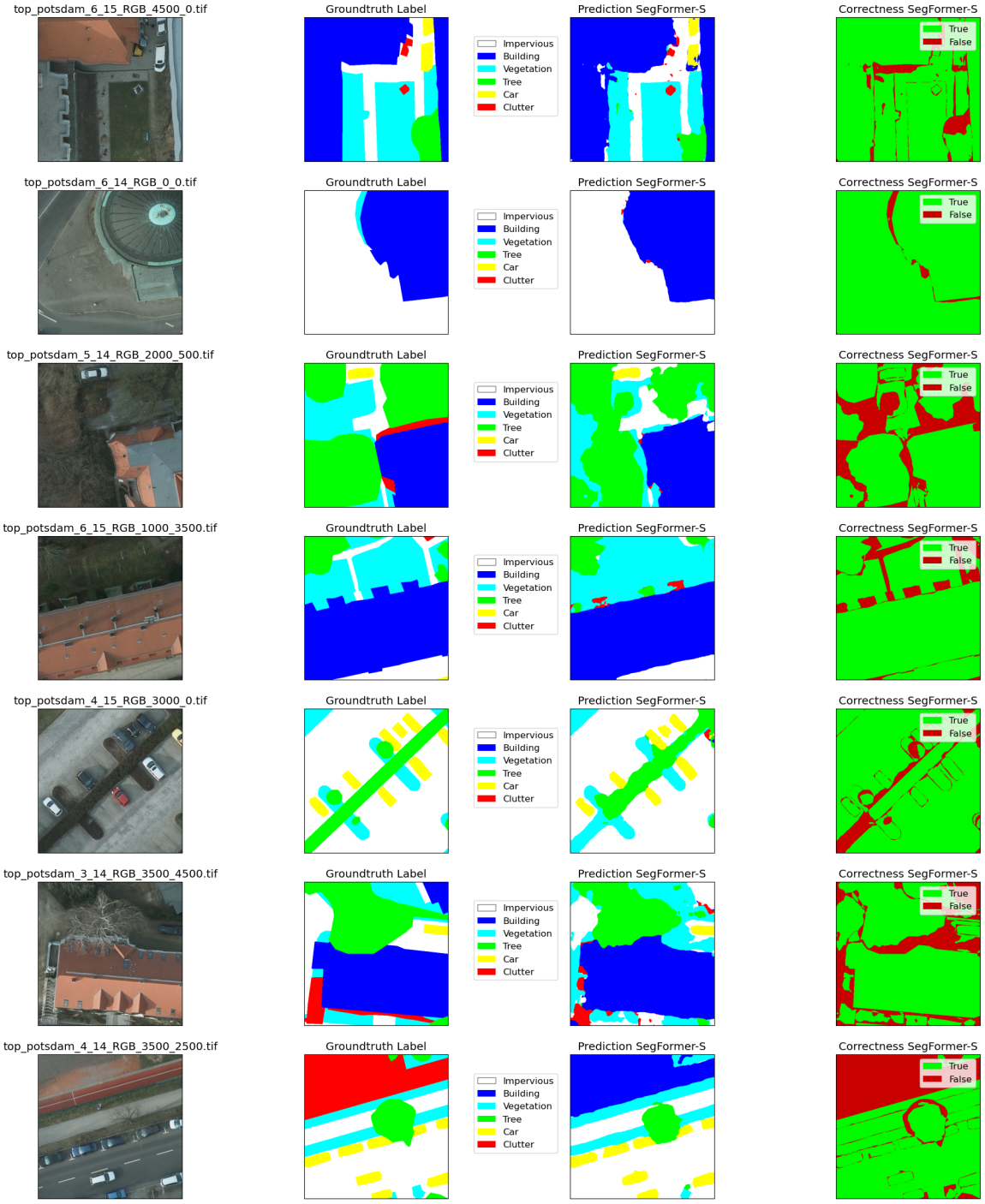


Figure B.10: Examples for model SegFormer-S.

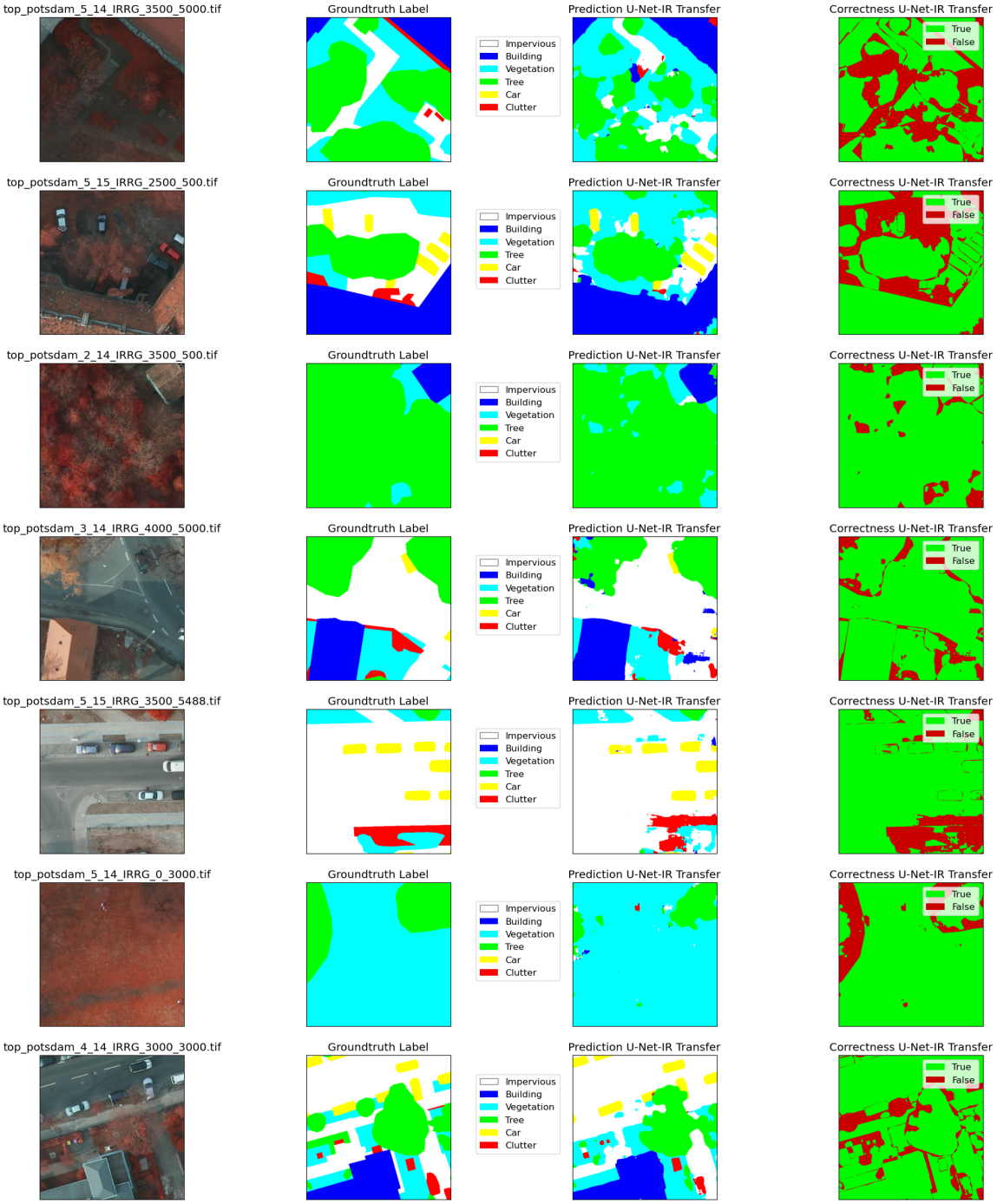


Figure B.11: Examples for model U-Net-IR.

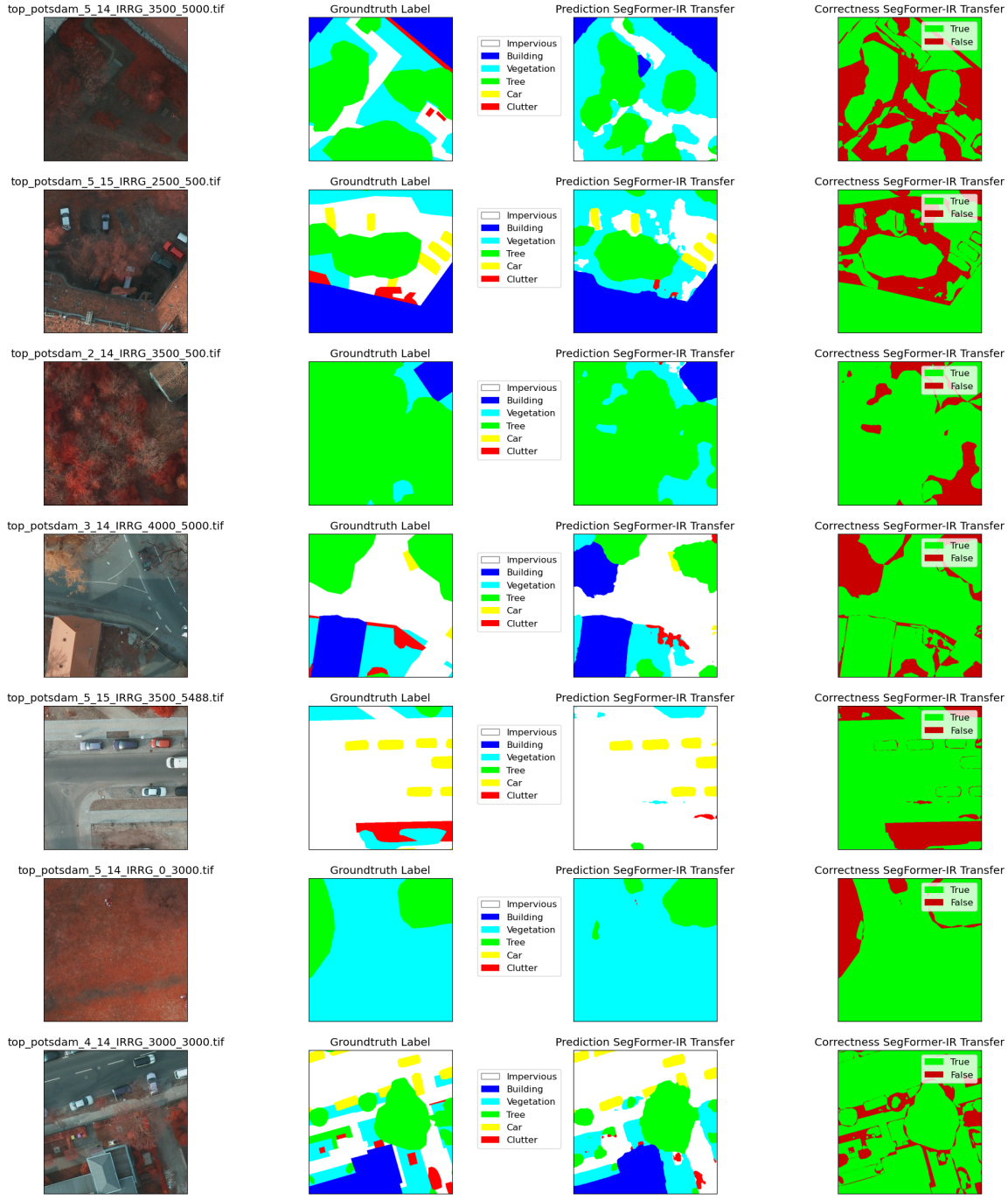


Figure B.12: Examples for model SegFormer-IR.

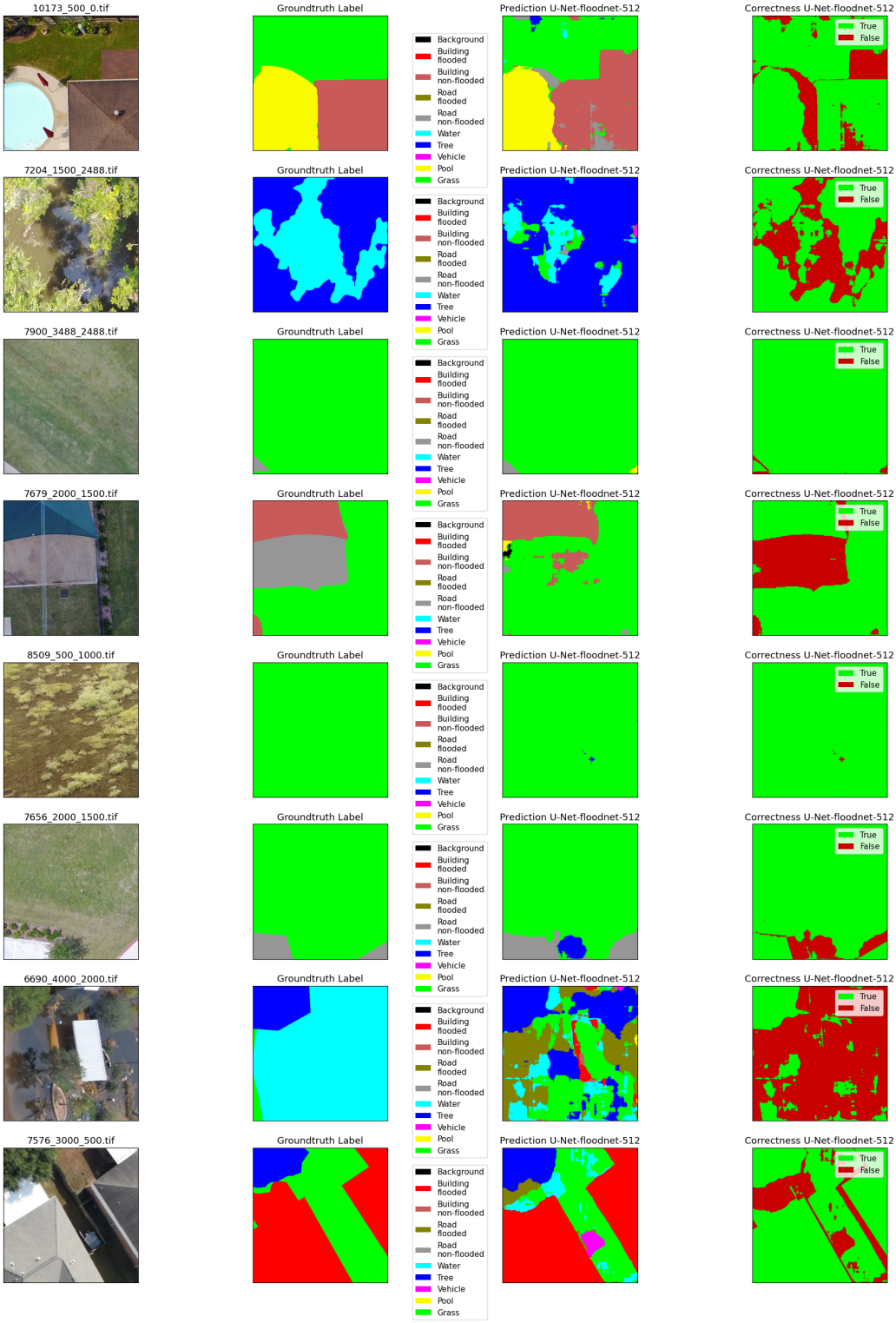


Figure B.13: Examples for model U-Net-floodnet-512.

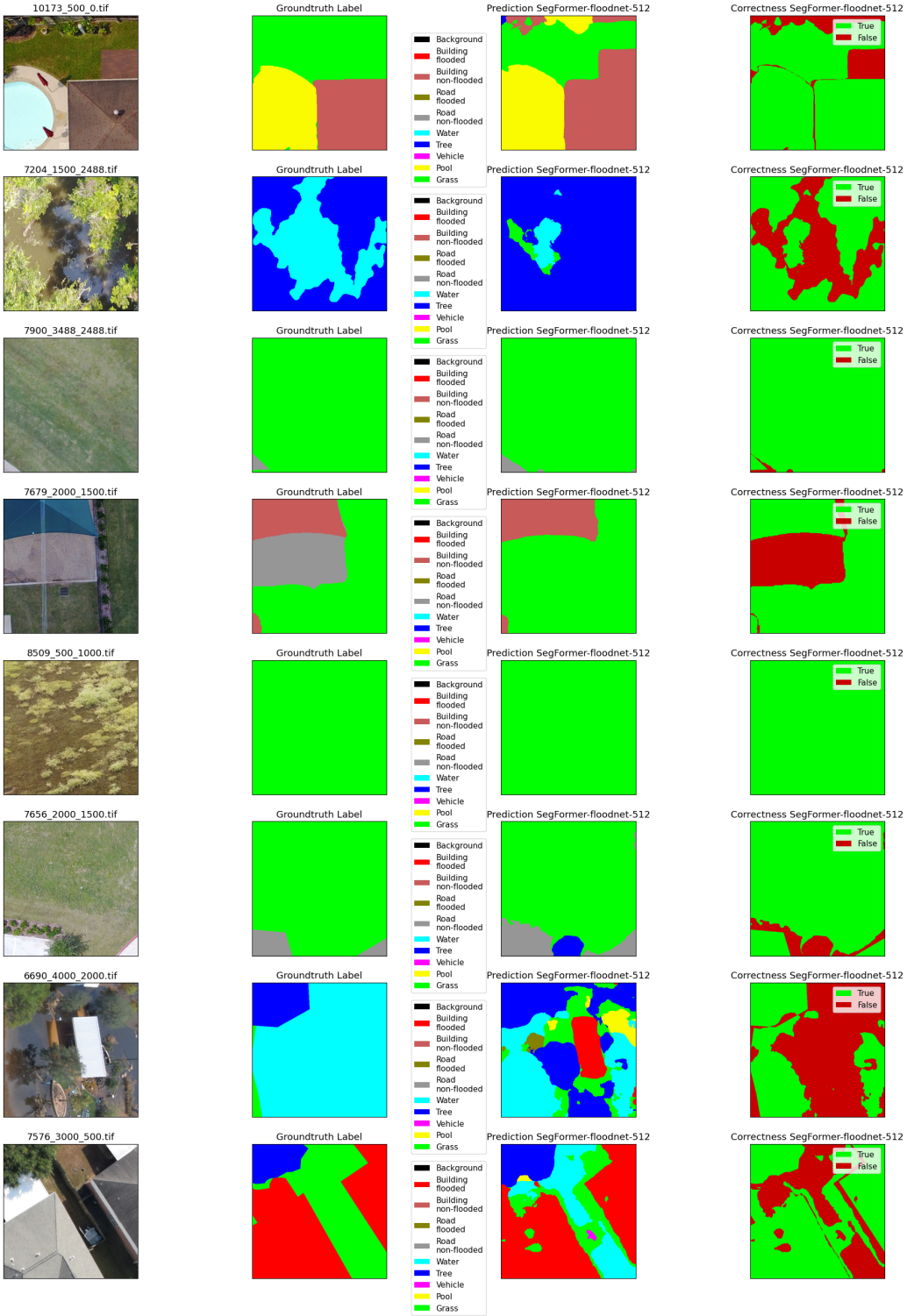


Figure B.14: Examples for model SegFormer-floodnet-512.

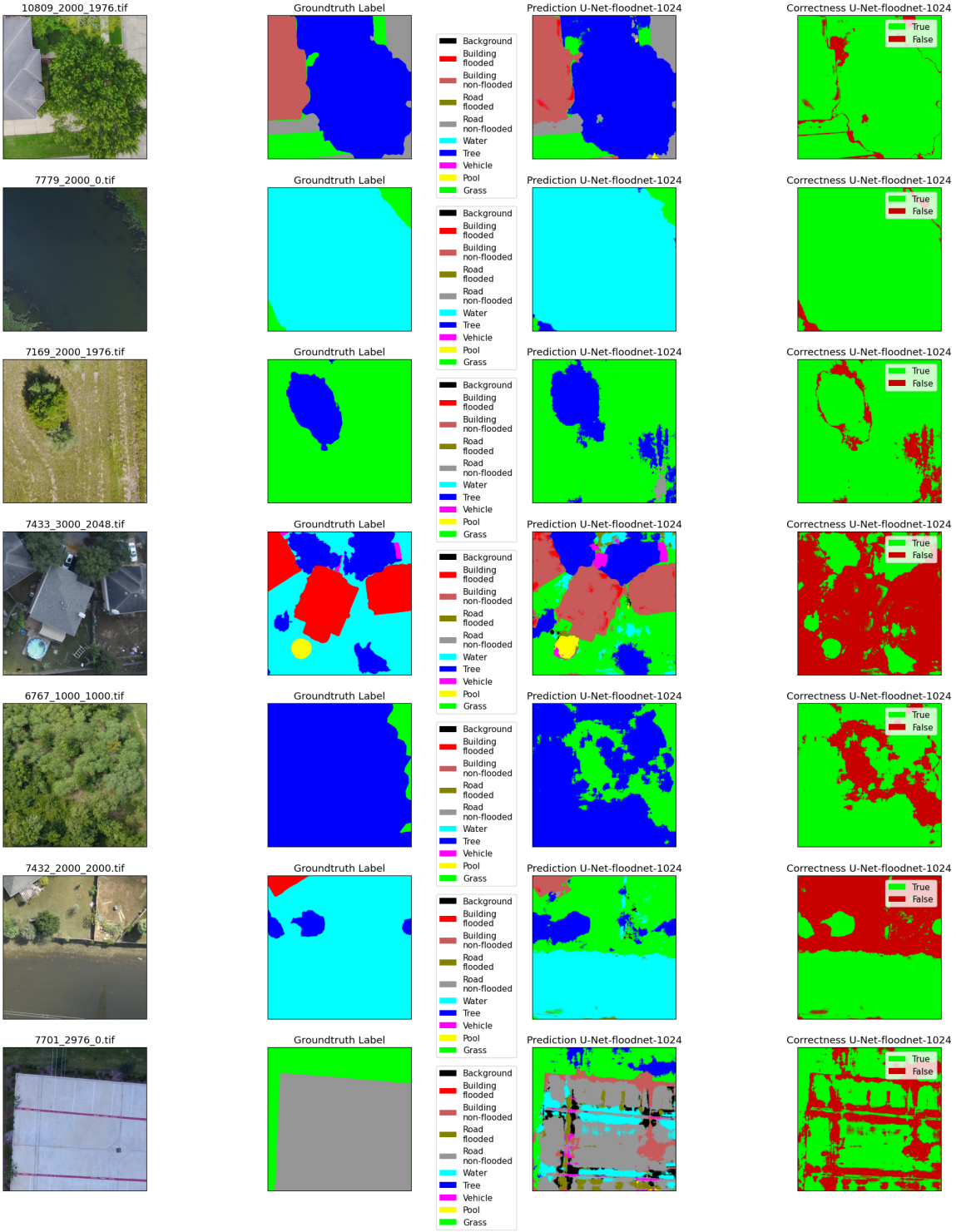


Figure B.15: Examples for model U-Net-floodnet-1024.

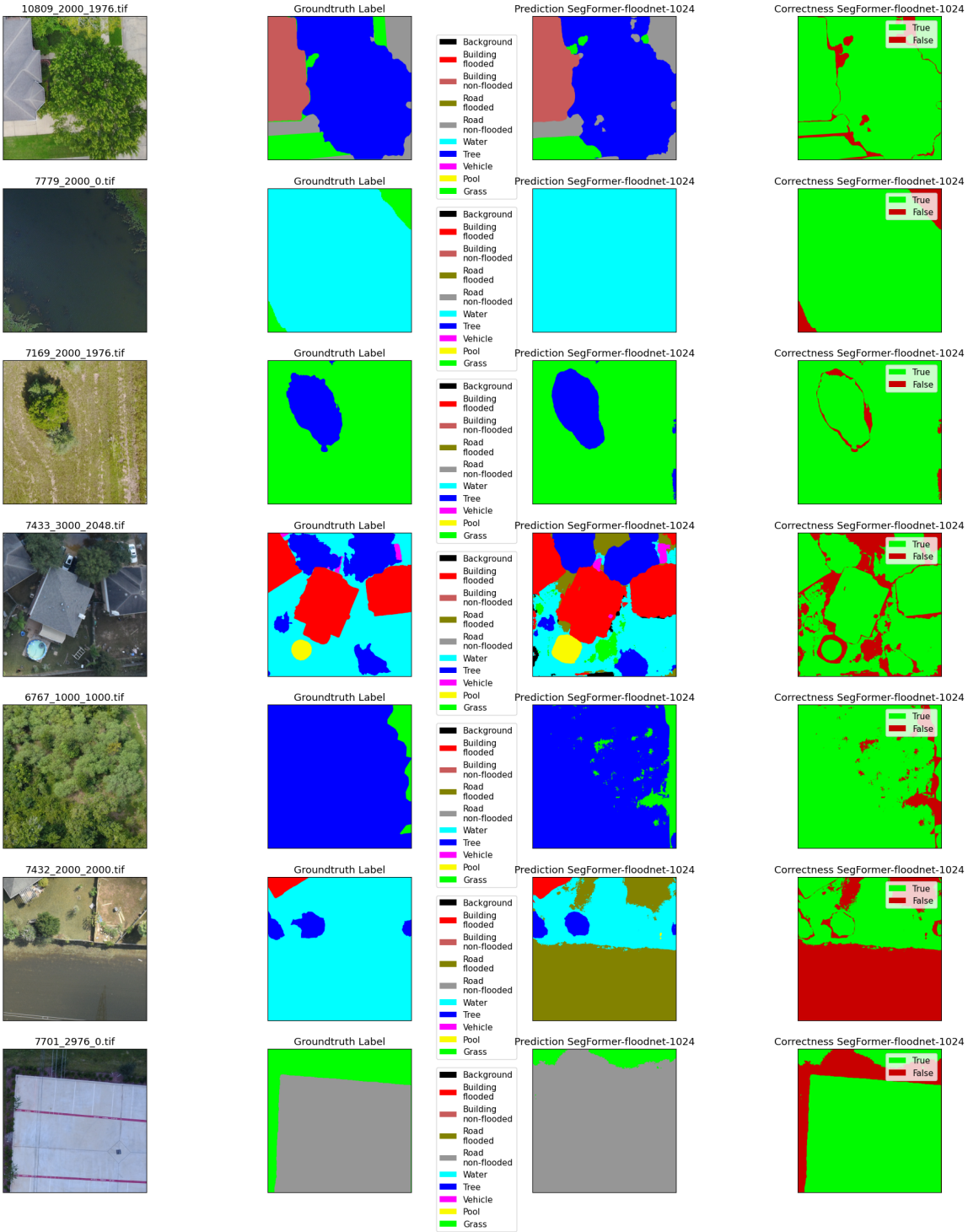


Figure B.16: Examples for model SegFormer-floodnet-1024.

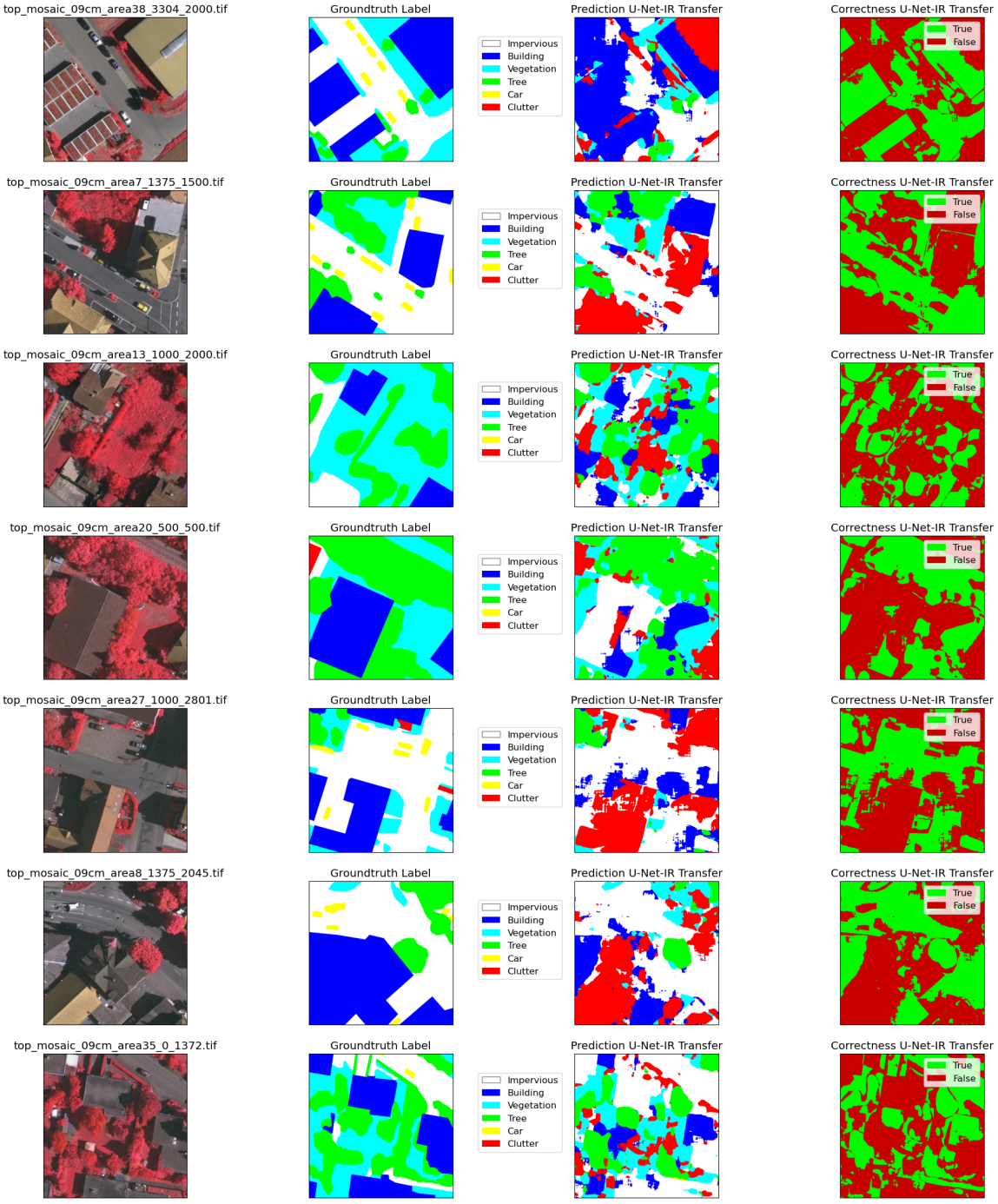


Figure B.17: Examples for model U-Net-IR applied to 512 × 512 image patches from the Vaihingen dataset.

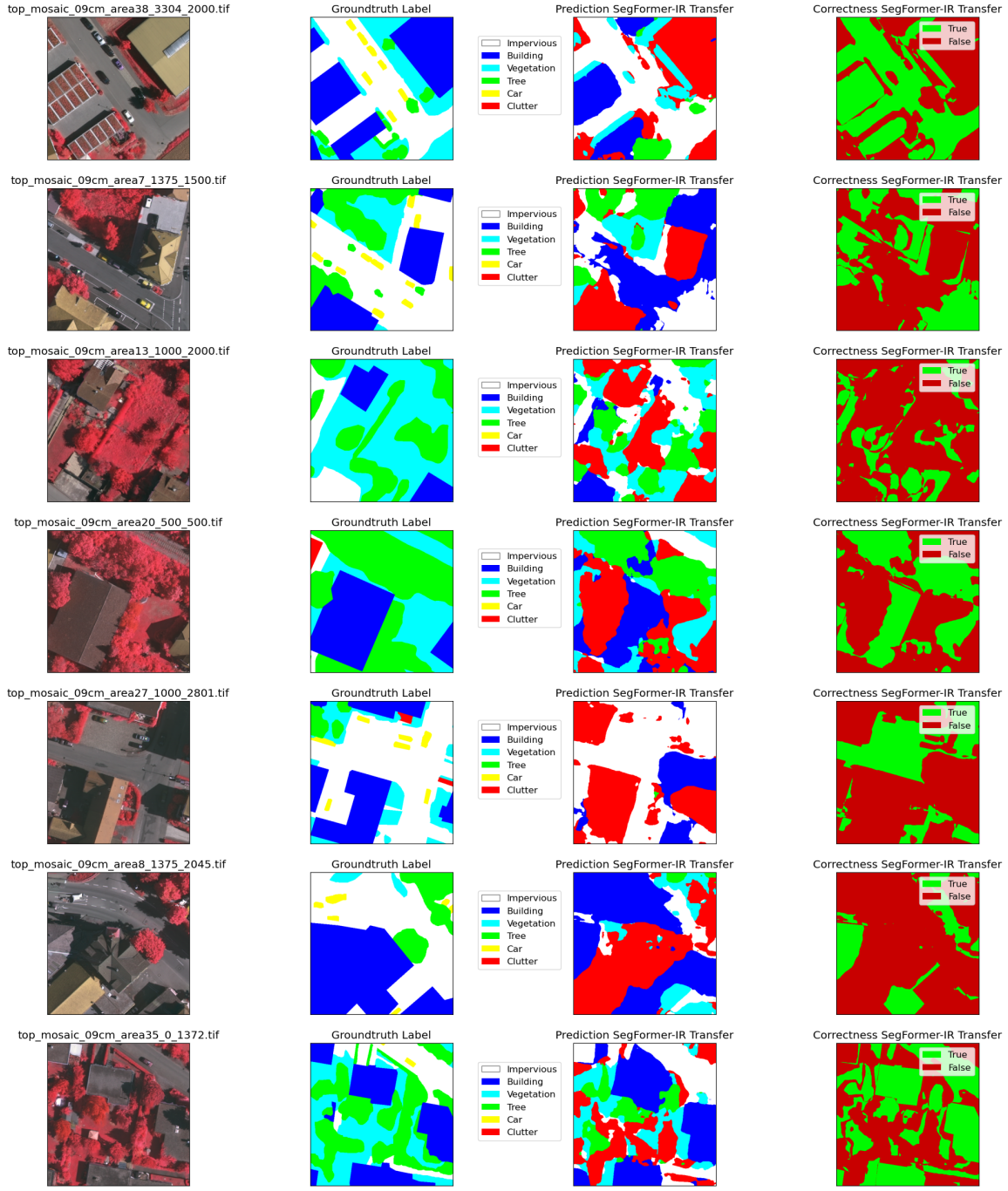


Figure B.18: Examples for model SegFormer-IR applied to 512×512 image patches from the Vaihingen dataset.

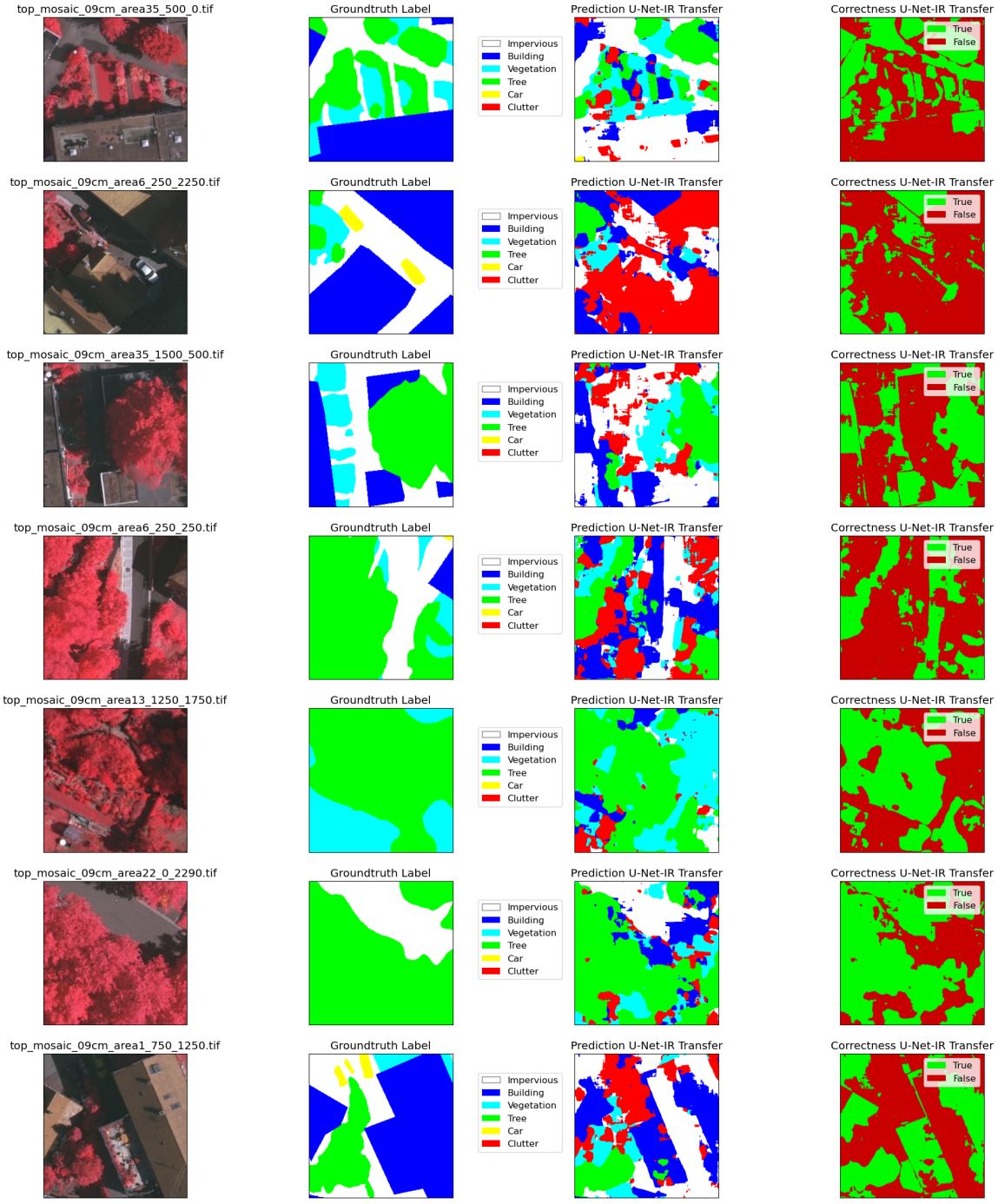


Figure B.19: Examples for model U-Net-IR applied to 256 × 256 image patches from the Vaihingen dataset resized to 512-pixel side length.

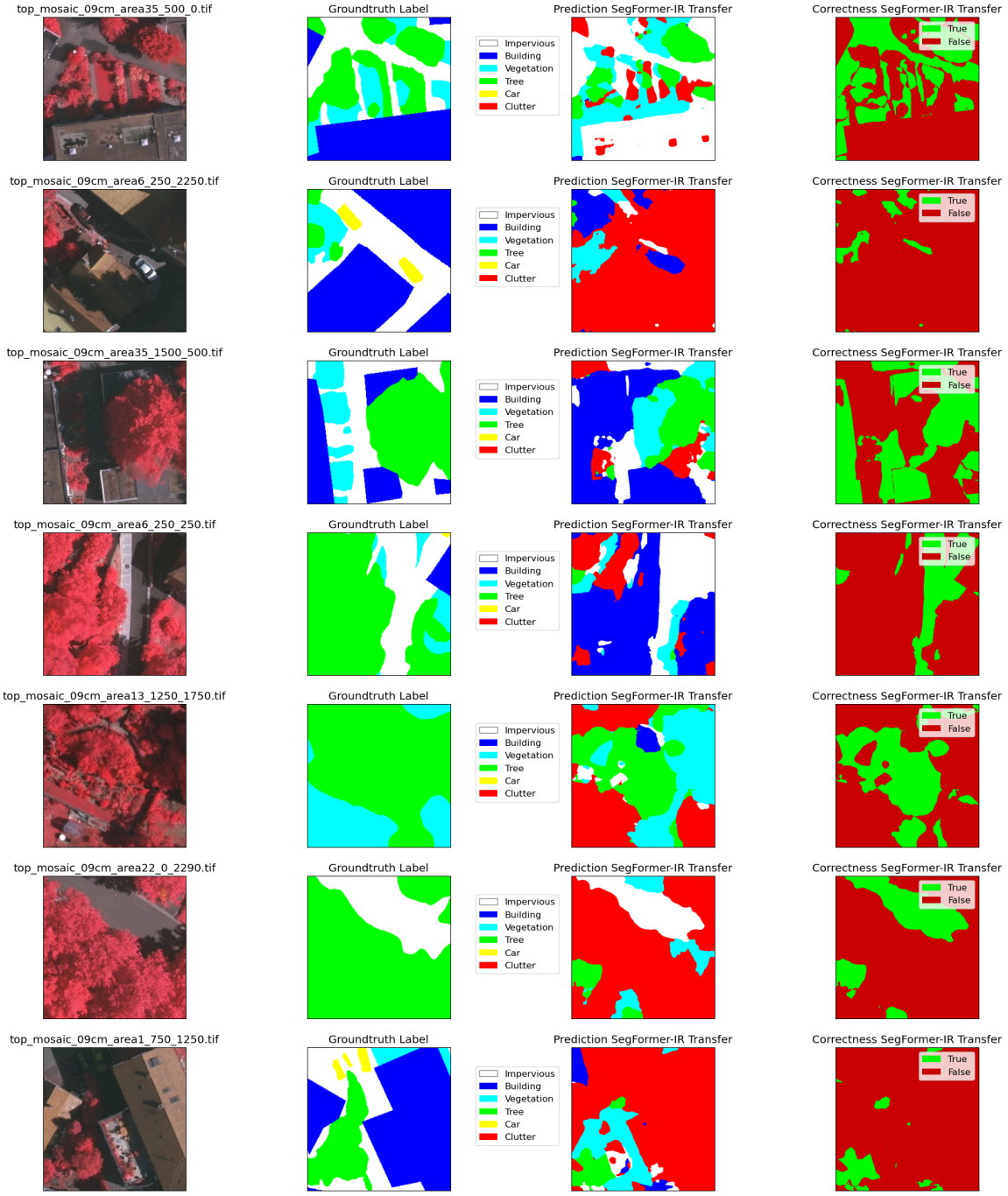


Figure B.20: Examples for model SegFormer-IR applied to 256×256 image patches from the Vaihingen dataset resized to 512-pixel side length.