Bachelor Thesis

DSM Refinement
with Deep Encoder-Decoder Networks

Nando Metzger

31st of May 2019

Chair of Photogrammetry and Remote Sensing
Institute of Geodesy and Photogrammetry
ETH Zurich

Professorship
Prof. Dr. Konrad Schindler

Mentoring
Corinne Stucker

Abstract

Digital surface models (DSM) can be generated from aerial images. However, the calculated DSM suffer from noise, artefacts, and missing values that have to be post-processed in a time-consuming, manual process. This thesis presents an approach that automatically refines such DSMs. The key idea of the methodology is to teach a neural network the characteristics of urban area from reference data with a deep learning approach. In order to approximate the human perception of geometric structures, a feature loss function is used to train the neural network. These features are constructed using a pre-trained image classification network. To learn to update the height maps, the network architecture is set up based on the concept of deep residual learning and an encoder-decoder structure. The experiments prove that this combination can be used to conserves the relevant geometric structures while also cleaning the undesired artefacts and noise. The thesis also includes an analysis of the complexity of the architecture such that an optimal number of convolutions and dimension reductions is concluded. Furthermore, the paper delivers evidence that the necessary amount of context area is in the magnitude of 25x25 m.

Acknowledgement

I would like to thank the following people:

- Prof. Dr. Konrad Schindler for making this thesis possible.
- Corinne Stucker for supervising this thesis and always having time to answer my questions.
- Sophie Trösch and Miriam Anderson for proof reading.
- Andrés Rodríguez and Nico Lang for the support in the field of neural networks.

Contents

List of Figures III

1 Introduction 1

2 Theoretical Principles 3

2.1 Neural Networks . . . 3

2.1.1 Neurons/Units . . . 3

2.1.2 Layers . . . 4

2.1.3 Activation Function . . . 5

2.1.4 Universal Approximation Theorem . . . 6

2.2 Convolutional Neural Networks . . . 7

2.2.1 Convolutional Layers . . . 7

2.2.2 Pooling & Sampling Layers . . . 8

2.2.3 Autoencoder Model . . . 9

2.2.4 Encoder-Decoder Model . . . 10

2.2.5 Residual Neural Networks . . . 10

2.3 Training a Neural Network . . . 11

2.3.1 Forward Propagation . . . 11

2.3.2 Loss Function . . . 11

2.3.3 Backpropagation . . . 12

2.3.4 Gradient Descent & Learning Rate . . . 13

2.3.5 Stochastic Gradient Descent . . . 14

2.3.6 Optimisation Algorithms . . . 15

2.3.7 Regularisation . . . 16

2.3.8 Hyperparameter Tuning . . . 18

2.3.9 Training Strategy . . . 18

3 Methodology 19

3.1 General Idea . . . 19

3.2 Raw Data . . . 20

3.2.1 Photogrammetric DSM . . . 20

3.2.2 Ground Truth Data . . . 21

3.3 Data Processing . . . 22

3.3.1 Preprocessing . . . 22

3.3.2 Data Allocation . . . 22
3.3.3 Data Augmentation . . . 24
3.3.4 Normalisation . . . 26
3.4 Network Architecture . . . 27
3.5 Loss Function . . . 29
3.6 Implementation and Training . . . 33

4 Results and Discussion **34**
4.1 Patch Size . . . 34
4.2 Model Depth . . . 36
4.3 Skip Connections . . . 38
4.4 Pixel-wise Loss Metric . . . 40
4.5 Perceptual Loss Metric . . . 42

5 Conclusion **45**

Bibliography **50**

A Prediction Performance **51**

B Error Distribution **55**

C Declaration of Originality **56**

List of Figures

1 Automatically generated DSM and manually cleaned 3D city model. 1
2 Flowchart of the basic concept of the approach. 2
3 Biological neuron vs. mathematical neuron/unit. 3
4 A simple network with two hidden layers. 4
5 Activation functions. 6
6 Convolutional layer. 7
7 MAX-pooling. 9
8 Concept of an autoencoder. 10
9 Residual neural network. 11
10 Behaviour of a perceptual metric. 12
11 Behaviours of the gradient descent algorithm. 14
12 Visualisation of underfitting and overfitting. 16
13 Flowchart of residual learning network and feedback loop. 19
14 A grey-scaled visualisation of a raw DSM. 20
15 Visualisation of LOD 2. 21
16 Data allocation of the complete area. 23
17 Stratified random sampling 25
18 Architecture of the neural network. 27
19 Behaviour of the L1 norm and L2 norm as loss functions. 30
20 Construction of a perceptual loss function 31
21 Visualisation of different patch sizes. 34
22 Prediction performance with different levels of depth 37
23 Prediction performance of models with and without skip connections 39
24 Prediction performance comparison of L1 and L2 norm as loss functions . . 41
25 Prediction Performance of models trained with perceptual metrics 43
26 Strengths and weaknesses of the model. 46
27 Prediction Performance on the test area 1/4 51
28 Prediction Performance on the test area 2/4 52
29 Prediction Performance on the test area 3/4 53
30 Prediction Performance on the test area 4/4 54
31 Error distribution 55

1 Introduction

3D city models are an integral part of online map services and have a wide range of applications such as navigation, urban planning, insurances, facility management, entertainment industry, virtual reality, routing, 3D cadastral systems, noise simulations, and many more (Biljecki et al. 2015). A favoured way to create a *digital surface model* (DSM) is to use dense photogrammetric reconstruction. Matching points in aerial images and estimating and fusing depth maps are fully automated processes. However, the resulting DSM contain artefacts as the comparison of Figure 1 shows. In the automatically generated DSM (Figure 1a) the surfaces are noisier than in the reference DSM (Figure 1b). Another issue is that the image matching algorithm can have problems in resolving in the images. The building in the lower part of Figure 1 has a lot of similar-looking roof dormers. The dense matching algorithm has difficulties matching the right dormer in every aerial image which leads to systematic outliers like the holes in the building. Furthermore, to reconstruct the coordinates of an object point, the point has to be visible and detectable in at least two aerial images. Due to shadowing effects from buildings and trees, some points cannot be reconstructed which results in lack of data in some areas. Note that Figure 1 does not include this issue.

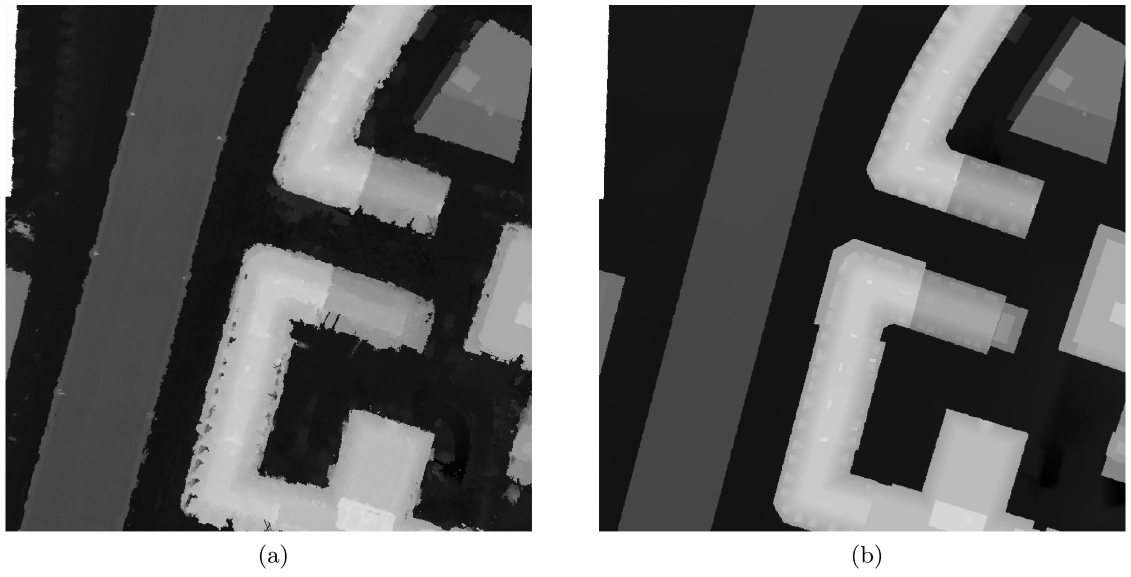


Figure 1: Raw DSM generated by automatic image matching of aerial imagery (a) and a manually cleaned DSM (b).

The problems of image-based 3D modelling can be summed up as: The algorithms used are naive. In this context naive means that the algorithm does not incorporate any domain knowledge. Buildings are likely to have parallel walls, many surfaces are piece-wise planar, dormers have characteristic dimensions, building facades are perpendicular to the street and so on. However, encoding this domain knowledge into a mathematical constraints is difficult. The alternative is to learn the characteristics of urban areas from data. State-of-the-art image processing approaches for learning such complex patterns are based on deep learning neural networks.

The goal of this thesis is to enhance DSMs generated by dense photogrammetric reconstruction by learning an *a priori* model of urban structures from data. The idea is to train a deep convolutional neural network that refines an initial DSM by completing missing geometry, removing noise and enhance missing details. This steps are currently performed in a manual manner. The automation of these steps makes the generation of 3D city models a more efficient process.
An important thing to elucidate is that the model does not have to master how to construct a DSM as the input already is a DSM (but with artefacts). The approach is to start with an initial DSM X and learn to predict residuals f(X) to update the DSM input as shown in Figure 2.

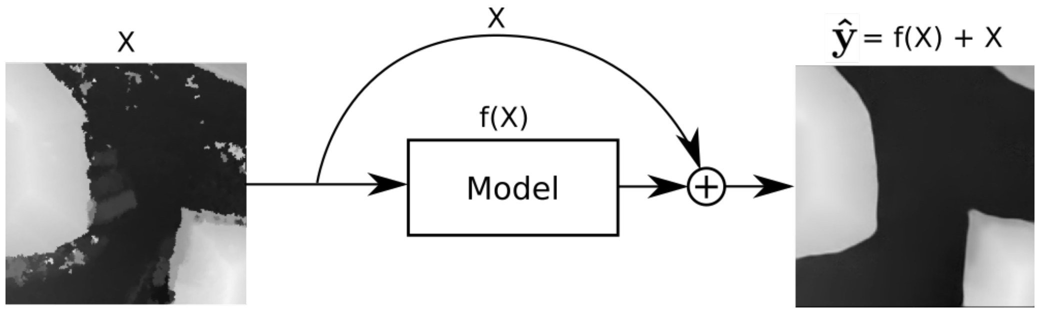


Figure 2: Flowchart of the basic concept of the approach.

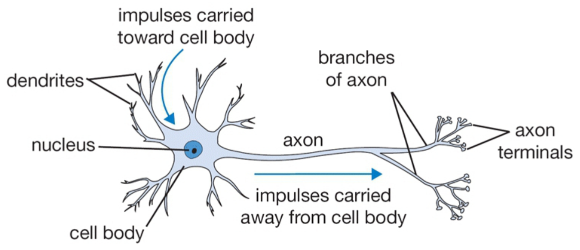
2 Theoretical Principles

This chapter summarises the basic concepts of deep learning. Furthermore, the principle of convolutional neural networks is introduced and some important architectures of convolutional neural networks are explained. The last part of this chapter will focus on the process of training a network. The source of this Chapter is Karpathy (2019), if not stated explicitly.

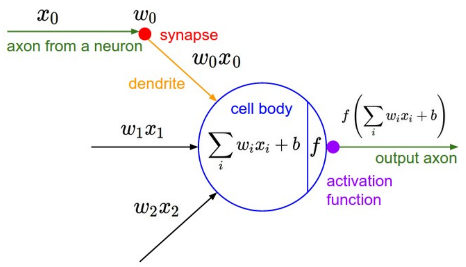
2.1 Neural Networks

Neural networks are an approach to learn from data and are designed to accomplish tasks that are too complex to formalise them mathematically. This technique is used for self-driving cars, language translation, stock market prediction, image compression, and many more applications which require dealing with a huge amount of data in complicated problems.

2.1.1 Neurons/Units


impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

(a) (Karpathy 2019)



(b) (Karpathy 2019)

Figure 3: Comparison of a biological neuron (a) and a mathematical neuron/unit (b).

The concept of neural networks is inspired by the biological brain. The basic components are neurons as shown in Figure 3 (a). Biological neurons take in impulses of other neurons, evaluate the signals in the body cell, and carry away impulses via axons. Mathematical neurons (also called *units*) as shown in Figure 3 (b) work very similarly. They perform a linear combination of all inputs signals x_i and add a bias b:

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

The variable w_i denotes the weight for the corresponding signal x_i and n is the number of input signals. The unit then carries away an output signal:

$$f\left(\sum_{i=1}^{n} w_i x_i + b\right). \tag{2}$$

The function $f()$ is the called the activation function, which is discussed in Section 2.1.3. It is worth mentioning that the mathematical unit is only inspired by the basic concept of the biological neuron. In fact, biological neurons are far more complex.

2.1.2 Layers

Neural networks are organised in *layers*, each containing several units. Three types of layers are distinguished: input layer, hidden layer and output layer as shown in Figure 4. The number and size of hidden layers vary depending on the task. The more numerous and bigger the layers, the larger the representational power of the neural network is and the fewer and smaller the layers, the better its generalisation performance is. Depending on the task, either representational power or generalisation performance is more desired.

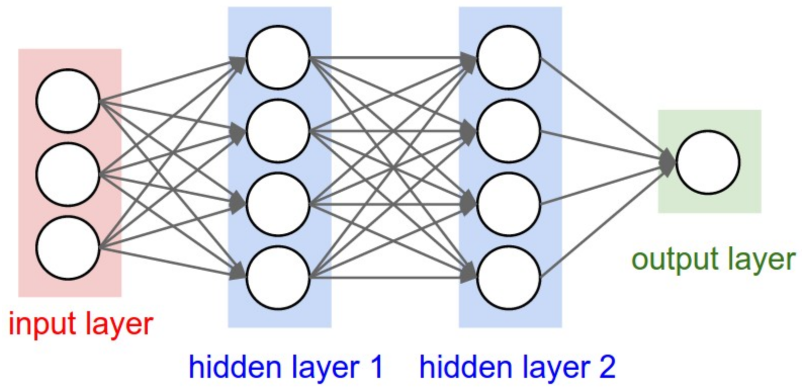


Figure 4: A simple network with two hidden layers. (Karpathy 2019)

In the case shown in Figure 4, the network consists of an input layer with dimensions 3, two hidden layers with each four units (dimension 4) and a scalar output. Layers, in which every previous unit has a connection to every unit in the layer, are called *dense layers* or *fully-connected layers*. Note that the size of each layer can be any arbitrary number and the sizes of the two hidden layers do not have to be the same.
As already explained in Section 2.1.1, every unit evaluates its input signals by $\sum_{i=1}^{n} w_i x_i + b$.

This means that the ith unit of the first hidden layer $\mathbf{h}^{(1)}$ can be represented as

$$\mathbf{h}_j^{(1)} = \sum_{i=1}^{3} w_{ji} x_i + b. \tag{3}$$

However, nesting several linear evaluation functions will lead to an output that is a linear function itself, and hence, the whole neural network could be represented with a single layer. That is where the activation function comes in.

2.1.3 Activation Function

Activation functions are used to represent non-linear functions. After the (linear) evaluation of a unit, the activation function is applied in order to receive the output. Such functions come in various forms each with different properties suitable for different purposes. The choice depends on the characteristics of the function, mainly:

- range of the output
- monotony of the function and its derivative
- differentiability
- continuity
- derivative function (see Section 2.3.3)
- approximation of the identity function near zero
- zero-centring

In the following, some of the most widely applied functions are explained. Figure 5 shows the graphs of these functions.
When it comes to expressing values between 0 and 1, the *sigmoid function* is likely to be the best choice. It is often used for probabilities in classification tasks and has a simple derivative. The *hyperbolic tangent (tanh)* is bounded like the sigmoid function. However, the tanh is symmetrical around the origin. As a consequence, if the input of the function is zero-centred, the output will be zero-centred as well. This leads to better convergence of the network. The *recified linear unit (ReLU)* function is a thresholded activation at zero. Its big advantage is the simplicity of the derivative. According to Krizhevsky et al. (2012), ReLUs can accelerate the learning by a factor of 6. The downside is that ReLUs can become inactive if their input value becomes negative. The *leaky rectified linear unit*

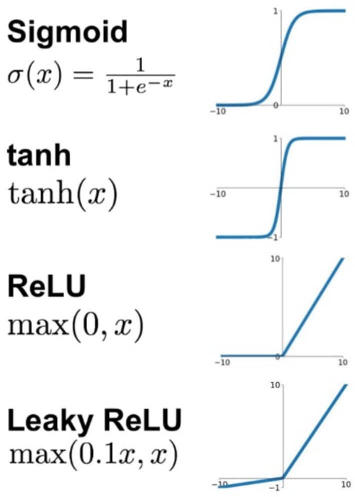


Figure 5: A selection of activation functions. (Jadon 2018)

(LReLU) removes this behaviour by setting a small slope to the negative part of the function. In Figure 5 this slope is set to 0.1. However, it could be any arbitrary number except for 1 (which corresponds to the identity function). He et al. (2015) even proposed to turn the slope value of each unit into a trainable parameter by introducing the *parametric rectified linear unit (PReLU)*.

2.1.4 Universal Approximation Theorem

In Section 2.1.2, it was already discussed that more layers will improve the representational power of a neural network. Hornik et al. (1989) established the universal approximation theorem for neural networks. It states that every standard multilayer feedforward network with more than one layer and ”squashing” functions (e.g sigmoid function or similar) can approximate any function, provided that there are enough hidden units in the neural network. This implies that for every task a neural network leads to a realistic solution, given enough and correct training examples and the right architecture of the neural net. In practice however, it is difficult to find a data set that contains sufficient data with the required accuracy. Furthermore, finding the right architecture is not always simple, either.

2.2 Convolutional Neural Networks

Up to this point, this paper only treated dense layers. However, regarding image interpretation and image generation, dense layers are often not the right choice. A pixel is often correlated with other pixels in the neighbourhood. It is not reasonable to connect the very top-left pixel to the very bottom-right pixel. *Convolutional neural networks* are neural networks which make use of the correlation between neighbouring pixels using special kinds of hidden layers. In the first half of this section, some of these layers are presented and in the second half, some implementations of convolutional neural networks for generative tasks will be explained.

2.2.1 Convolutional Layers

Unlike dense layers, convolution layers only use connections within a specified neighbourhood. Considering a neighbourhood of 3x3 as in Figure 6, each pixel only has to handle nine input signals. The weights of the input signals are stored in a kernel matrix, where the size of the kernel is the size of the neighbourhood. The same kernel is then applied to every pixel in the image and performs a linear combination of the input pixels. Analogously to dense layers, a bias parameter per kernel can be used as well.

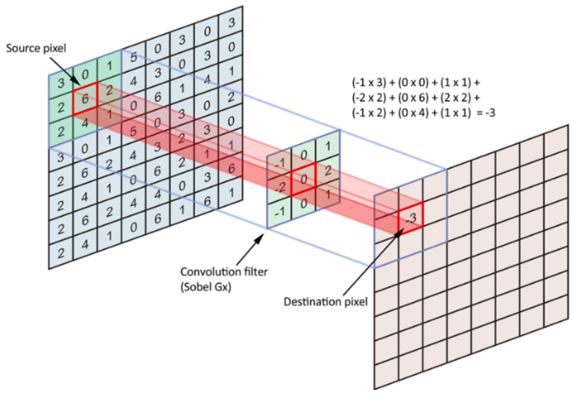

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Figure 6: Convolutional layer. (Cornelisse 2018)

The same kernel iterates over all pixels. This results in local connectivity of the input and output of the layers, such that the relative positions of pixels are preserved (e.g. top-left pixel remains top-left and so on). The movement of the kernel is specified with strides. A stride of 1 means that the kernel moves 1 pixel per iteration step. The output of one pixel is very similar to the output of surrounding pixels. Therefore, strides of two or more are often

used. The stride can be specified for each dimension independently. The stride is also the factor of the dimension reduction. For example, if a convolutional layer with stride 2 (for the x- and y-dimension) is applied to a input with shape 128x128, it results in a dimension reduction by 2 in each axis (e.g. the output will be of size 64x64). Dimension reduction plays an important role in avoiding overfitting, which will be explained in Section 2.3.7.
In a convolutional layer, multiple kernels are applied to the input simultaneously. The outputs are then stacked in channels, which correspond to the third dimension of the layer. One huge advantage of convolutional layers lies in the sparsity of the parameters due to the parameter sharing using the concept of kernels. As an example, a simple neural network considered. It has an input size 256x256x1, two hidden layers, each with dimension 256x256x32 and an output of dimension 256x256x1. Using dense layers the number of parameters will be $[256 * 256 * 1] * [256 * 256 * 32] + [256 * 256 * 32]^2 + [256 * 256 * 32] * [256 * 256 * 1]$ which evaluates to approximately 10^{12} parameters without considering the biases. Whereas using a convolutional layers with biases, stride 1 and kernels of size 4x4 will have $([4 * 4 * 32] + 32) + ([4 * 4 * 32] + 32) + ([4 * 4] + 1) = 1105$ parameters. This means that there are tremendously fewer parameters to be determined. This parameter sharing allows the neural network to be more controllable. Furthermore, the model is also more robust to shift in the image, since the same kernels are applied in every part of the image.
A Question arises: What happens to the border pixel when some parts of the kernel exceed the input dimensions? The solution is padding. This means that the input image is being extended with filling-values in order to be able to calculate outputs for border pixels. Extending the image with zeros (zero-padding) is a commonly used method. However, alternatives such as mirroring or repeating the border elements of the inputs are possible, as well.

2.2.2 Pooling & Sampling Layers

As shown in the section before, strides in convolutional layers can be used for dimension reduction. Another subtle way of doing so are *pooling layers* which are kernel functions with non-trainable parameters. The most commonly used is the *MAX-pooling* layer, which applies a MAX operation on its inputs as shown in the Figure 7. Specialised kernels of convolutional layers detect pattern in the input image. The output is a strong signal if the pattern is clear. Placing a MAX-pooling behind a convolutional layer can be very effective. It will conserve the information about the pattern, while deleting the unimportant signals. MAX-pooling kernel sizes larger than 2x2 are rarely used, because the larger the kernel area, the more information will be lost. Generally, many other pooling operations exist such as average pooling and L2-norm pooling, which will not be discussed here.

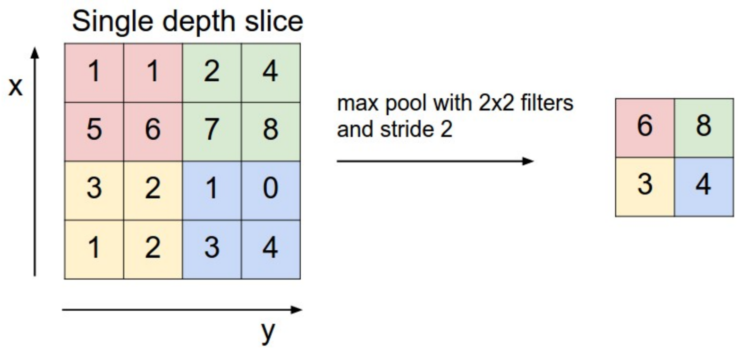


Figure 7: A MAX-pooling operation with a 2x2 filter and stride 2 applied on a 4x4 input. It result in an output of dimension 2x2. (Karpathy 2019)

For generative networks, dimension expansion is as important as dimension reduction. Therefore *upsampling layers* exist, which repeat pixels by a specified amount. Alternatively, *upconvolutional layers* (also called *transposed convolutional layers*) can be used. These layers perform the inverse function of the convolutional layers and enhance the dimensions of an image with a specified inverted stride. These upconvolutional layers will not be discussed further in this paper.

2.2.3 Autoencoder Model

There are two different tasks in deep learning: classification (outputs are discrete class labels) and regression (outputs are continuous values). This section as well as the following two focus on architectures that are used for image-to-image tasks, which belong to the group of regression tasks.

First, it is important to note that an image is always just a representation of an object. The image describes the object in terms of pixel values. Usually, a lot of redundant information is stored within the whole image. An *autoencoder* is a neural network with two parts: an encoder and a decoder (as shown in Figure 8). The task of the encoder is to get rid of the redundant information in the image and store the information in a compressed representation (the latent space). The decoder's job is to reconstruct the original input using only the compressed representation.

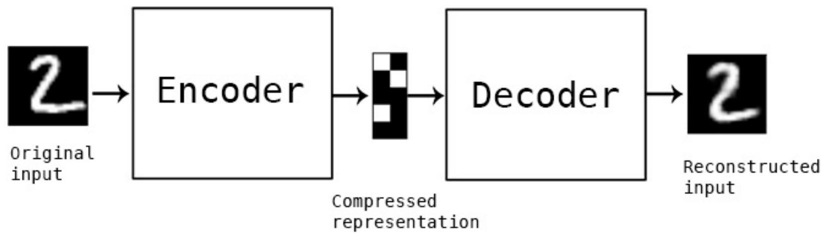


Figure 8: An autoencoder with a latent space of ten dimensions. (Chollet 2017)

2.2.4 Encoder-Decoder Model

The general approach to autoencoders are *encoder-decoder models* where the input is not encoded in the same representation as the output. In this way, representations can be changed. For example, the input data could be an image containing noise (this can be viewed as an own representation of an object), whereas the output is free from noise. The network is forced to learn how to denoise the image. This specific type of an encoder-decoder model is called a denoising autoencoder. (Goodfellow et al. 2016, p. 510-515)
Encoder-decoder models have an even wider application than just image processing. The same concept can also be used for language translation task, where a language can be viewed as a representation of a sentence or a word.

2.2.5 Residual Neural Networks

For some image denoising tasks, a regular encoder-decoder network can be inefficient. The set up of the model forces the network to reconstruct a full image from a compressed dimension. However, the task should rather be removing noise from images, since this is much simpler than a reconstruction task. Therefore, He et al. (2016) proposed the *residual neural network*. In the general case of a residual neural network (shown in Figure 9), the model is forced to learn a residual function $\mathcal{F}(x)$ instead of $\mathcal{H}(x) := \mathcal{F}(x) + x$ as with a regular encoder-decoder model. Copying the values of a layer and adding it at a later point is called a skip connection. It is important that the dimensions match again at the end of a skip connection. Residual learning is easier to optimise and promotes better convergence. As a result, more layers (e.g. parameters) can be used, which can lead to better performance. Combining the concept of residual learning with the encoder-decoder model, it means that the first weight layer is the encoder and the second weight layer is the decoder. (He et al. 2016)

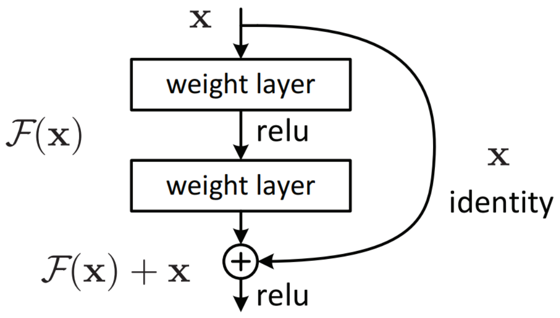


Figure 9: A building block of a residual neural network. (He et al. 2016)

2.3 Training a Neural Network

Until this point, this paper only covered how to set up an architecture of a neural network. This section will first cover how to make predictions with neural networks (the so called forward pass) and grade those predictions. Later the process of how to train a model is explained. Finally, the regularisation of a neural network will be introduced as well as training strategies and hyperparameter handling.

2.3.1 Forward Propagation

Evaluating a neural network for a given input $\mathbf{x}$ with its current weights $\mathbf{w}$ is called a *forward propagation*. The procedure systematically calculates the outputs of every hidden layer according to their connections and activation functions. Finally, the output of the last layer will be the prediction $\hat{\mathbf{y}}$ for the input $\mathbf{x}$.

2.3.2 Loss Function

The *loss function* $\mathcal{L}$ quantifies the "goodness of fit" of a prediction $\hat{\mathbf{y}}$ by comparing it to the ground truth $\mathbf{y}$. The function returns a scalar value $\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$. Stressing its conditionality on the weights $\mathbf{w}$ and the dependency on the input, it can be written as $\mathcal{L}(\mathbf{x}, \mathbf{y}|\mathbf{w})$.
It is important to use a loss function that is appropriate to the task. For regression problems, the L2 norm is a usually an adequate choice. However, the L2 norm is making the assumption of Gaussian distributed noise around the true value (without any bias). Therefore the L1 is sometimes favoured as a loss function
Although, implementing a pixel-wise L1 or L2 norm as a loss function can lead to working convolutional neural networks, these functions do not represent human perception at all.

For example, considering the middle patch of Figure 10 as the reference patch and compare the similarity of the blurred (left) and the distorted (right) patch to the reference. From the point of the human perception, the distorted patch looks more similar than the blurred, because it respects contours and patterns. However, the L1 and L2 norm would be more impacted by the distorted patch, since the loss is calculated pixel-wise, without considering any neighbourhood. The loss function threads many pixels as completely wrong, although they are only shifted by some pixels. For the L1 and L2 norm, the burred patches are considered to be more similar, because the pixels are still at the correct position except that the pixel value is a bit wrong. A perceptual metric should behave like human vision. It should treat blurred images as less similar (higher loss) and distorted images with the same pattern as more similar (smaller loss). To do so, the loss function needs to consider neighbourhoods of pixels instead of just a pixel-wise comparison. (Zhang et al. 2018)

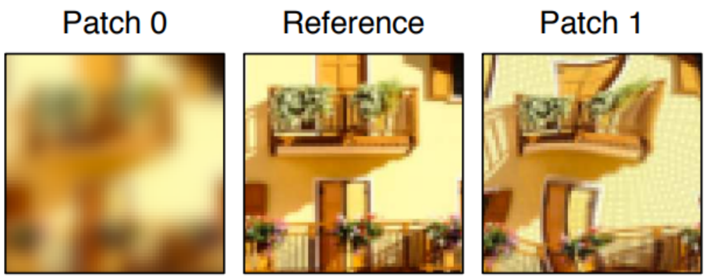


Figure 10: Which patch (left or right) is closer to the middle patch in these examples? (Zhang et al. 2018)

One methodology to benefit the right shape of objects is called *feature loss functions*. The features are calculated using the output of the network. The loss in the feature space is scaled by a hyperparameter λ and added to the loss in the image space as in Equation 4. (Dosovitskiy and Brox 2016; Zhang et al. 2018)

$$\mathcal{L}_{total} = \mathcal{L} + \lambda \mathcal{L}_{feat} \tag{4}$$

2.3.3 Backpropagation

Neural networks consist of thousands or millions of weights. Finding a closed form solution for a global minimum in the loss function is generally impossible. Therefore, the numerical method called *backpropagation* is used. The approach is to use the loss function as an indication of how good the weights of the network are set and to update them until the loss function is minimal. To find the right direction of the update, the function is approximated with a first-order polynomial. Taking the first derivative of the loss function with respect to the weights yields the gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \nabla \mathcal{L}$. It points in the direction of the largest increase of the function. Since the approximation is linear, $-\nabla \mathcal{L}$ shows the direction of maximal decrease.

To compute this gradient the chain rule is applied. It means that for every layer and its activation the derivative has to be calculated. Calculating the derivatives of the units (a linear function as in Equation 1) is easy, whereas activation functions can become more complex. As already mentioned in Section 2.1.3, the characteristics of the derivative of the activation functions matter as well as the function itself. For example, the derivative of the ReLU function is computationally easy compared to the derivative of an arctan function:

$$ReLU'(x) = \begin{cases} 0, & x < 0 \\ 1, & x > 0 \end{cases} \tag{5}$$

$$arctan'(x) = \frac{1}{1 + x^2} \tag{6}$$

Considering that activation functions have to be evaluated millions of times in order to train a neural network, it is important to use activation function with simple derivatives. The sigmoid and the tanh function both have elegant solutions to avoid computational effort. The calculated values during forward propagation can be stored and recycled by the backpropagation step:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{7}$$

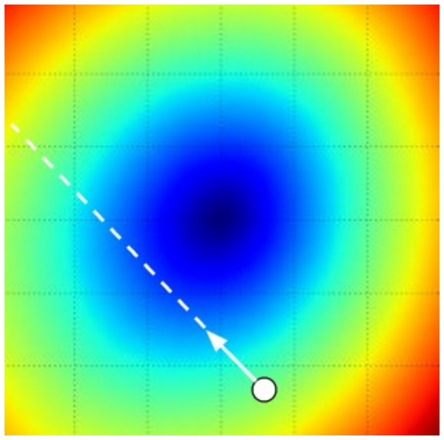
$$tanh'(x) = 1 - tanh(x)^2 \tag{8}$$

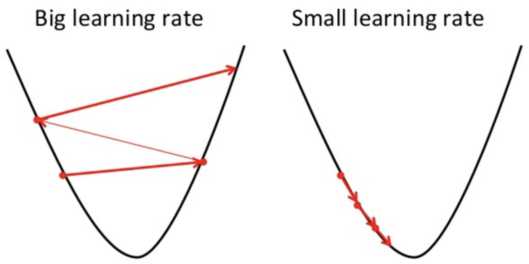
2.3.4 Gradient Descent & Learning Rate

Performing one update to the weights is called a step. Iteratively following the gradient (by updating the parameters) and calculating the new gradient again is called *gradient descent*. Mathematically the parameters $\mathbf{w}_i$ of the i^{th} iteration are calculated as:

$$\mathbf{w}_i = \mathbf{w}_{i-1} - \eta \nabla \mathcal{L}(\mathbf{w}_{i-1}) \tag{9}$$

The parameter η denotes the learning rate. It is a factor that scales the size of the gradient step. Figure 11 (a) shows how the gradient in one point could look like. The topology of the loss function is only known to a linear approximation at the current position of the weights. By setting the learning rate to a large number, there is a risk of overshooting the optimal update. The gradient descent algorithm will start oscillating as shown in Figure 11 (b) (left) or even diverge. However, if the learning rate is too small the training process will take longer as shown on the right side of Figure 11 (b). Furthermore, the parameters can get stuck in local minimum points. The goal is an adequate learning rate which avoids these issues.


(a) Gradient of a two-dimensional parameter space. (Karpathy 2019)



(b) Effect of a learning rate that is too large (left) or to small (right). (Donges 2018)

Figure 11: Behaviours of the gradient descent algorithm.

2.3.5 Stochastic Gradient Descent

Calculating the gradient of the entire training set often takes a long time. The *stochastic gradient descent* algorithm instead proposes to randomly pick one sample and compute the gradient update. The algorithm assumes that the loss topography of a one single training sample is comparable to the loss topography of the whole data set. Therefore the parameter update will be approximately the same. In this way, more steps of gradient descent can be executed with less computational effort. Looping over the whole training data set once is called an epoch.

The *mini-batch gradient descent* offers a compromise between the regular gradient descent and stochastic gradient descent. The algorithm picks a mini-batch of n randomly selected examples and calculates the gradient to execute an update. The hyperparameter n is called batch size. Small batch sizes can lead to unstable gradients, since the different samples produce different gradients. Large batch sizes are more stable, but demand more computational effort and higher memory consumption. The term batch should not be confused with the term patch, that is also used here in this paper: A patch is a snippet of a larger image. Several patches can form a batch.

2.3.6 Optimisation Algorithms

Note that if the parameter vector $\mathbf{w}_i$ could be randomly initialised far away from the optimal solution, large updates are desirable. As the parameters approach the optimal solution, smaller updates are desired. Therefore, common techniques exist for *learning rate decay*. Possible approaches are to reduce the learning rate every epoch by a fixed factor, exponential decay or $\frac{1}{t}$-decay.

In flat regions, the gradients will become close to zero and the algorithm will converge slower. In order to deal with this problem, the concept of momentum is implemented. The analogy is to a ball with its inertia that will keep rolling in flat areas. Mathematically, the cache $\mathbf{v}_i$ is a linear combination between the currently calculated gradient and the previous cache.

$$\mathbf{v}_i = \mu \mathbf{v}_{i-1} - \eta \nabla \mathcal{L}(\mathbf{w}_{i-1}) \tag{10}$$

$$\mathbf{w}_i = \mathbf{w}_{i-1} + \mathbf{v}_i \tag{11}$$

The Hyperparameter μ is introduced as *momentum*. The higher the value of μ, the more robust is the update of the stochastic gradient descent. The reason for this is that frequent directions will accumulate over the steps. (Sutskever et al. 2013)

Sutskever et al. (2013) introduced the *Nesterov Momentum*, which "looks ahead" by using the gradient $\nabla \mathcal{L}(\mathbf{w}_{i-1} + \mu \mathbf{v}_{i-1})$ instead of the gradient at $\nabla \mathcal{L}(\mathbf{w}_{i-1})$. Nesterov Momentum is proven to converge faster than standard momentum algorithms.

The *adaptive gradient algorithm* (short *AdaGrad*) normalises all the steps, while also summing up the gradient updates like the momentum methodology does. In this way, large gradients have the same influence as small or rare gradients. This encourages the finding of rare features (Duchi et al. 2011). However, for neural networks the AdaGrad provides too aggressive updates and tends to stop too early, because the updates get monotonically smaller.

The Adam optimiser deals with these downsides of the AdaGrad algorithm by using an average of quadratic gradients (See Equation 13). This also allows the gradients to increase (Tieleman and Hinton 2012). Furthermore, the algorithm provides smoothed versions of the gradient $\mathbf{m}_i$ (calculated as in Equation 12) to perform the update as in Equation 14

(Kingma and Ba 2014).

$$\mathbf{m}_i = \beta_1 \mathbf{m}_{i-1} + (1-\beta_1)\nabla\mathcal{L}(\mathbf{w}_{i-1}) \tag{12}$$

$$\mathbf{v}_i = \beta_1 \mathbf{v}_{i-1} + (1-\beta_2)(\nabla\mathcal{L}(\mathbf{w}_{i-1}))^2 \tag{13}$$

$$\mathbf{w}_i = \mathbf{w}_{i-1} - \eta\frac{\mathbf{m}}{\sqrt{\mathbf{v}_i}+\epsilon} \tag{14}$$

β_1 and β_2 are hyperparmeters that are usually set to 0.9 and 0.999 respectively. The ϵ is only a small constant to prevent division by zero.

2.3.7 Regularisation

Since neural networks usually have millions of weights, the mathematical system behind it has many degrees of freedom and allows infinitely many solutions to the problem. Many of those solutions do not generalise the problem well, such that the model is only accurate in predicting on the training set. In the case of Figure 2.3.7 (right) the model performs poorly on unseen data. This phenomenon is called *overfitting*. However, reducing the number of parameters could lead to *underfitting*. In the case of Figure 2.3.7 (left) the model would perform poorly, because the network has insufficient capacity to accomplish the given task. In machine learning, the expected error on unseen data (which has to be minimised) is also called generalisation error. (Goodfellow et al. 2016, p. 110-120)

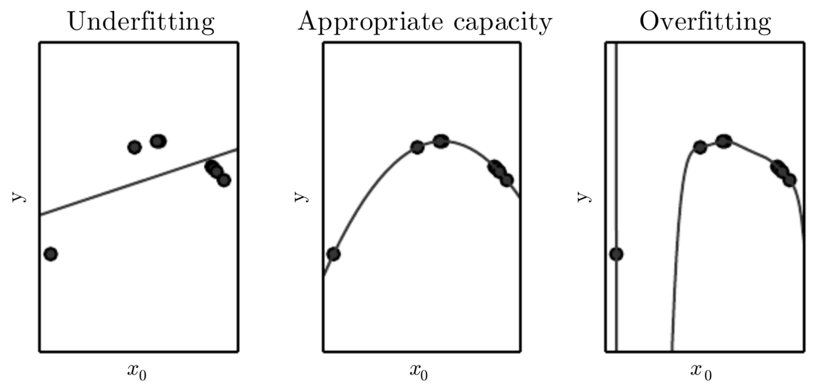


Figure 12: Visualisation of underfitting and overfitting. (Goodfellow et al. 2016)

Instead of using less parameters, an appropriate *regularisation* can be applied to the network in order to prevent overfitting. The goal is to find a regularisation such that the generalisation error will be minimal. Note that the network is still minimising the error on

the training data.(Goodfellow et al. 2016, p. 110-120)
In order to prevent overfitting several regularisation techniques exist:

- *Penalty Regularisation*: Polynomials with large weight are likely to oscillate as shown as in Figure 12 (right). By introducing a penalty term for large weights to the loss function oscillations can be mitigated.

  $$\mathcal{L}_{total} = \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) + \lambda_{weights}\mathcal{L}_{weights}(\mathbf{w}) \tag{15}$$

  $\mathcal{L}_{weights}(\mathbf{w})$ is often chosen to be the L2- or L1-norm of the parameters and $\lambda_{weights}$ is a hyperparameter to harmonise the penalty with the error loss. It is also possible to add a penalty term for activity regularisation in the same manner:

  $$\mathcal{L}_{total} = \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) + \lambda_{activity}\mathcal{L}_{activity}(\mathbf{a}) \tag{16}$$

  The vector **a** denotes selected output activities.

- *Dropout*: Another effective method is to cancel out a fraction of randomly selected units including their connections for the weights update. This prevents the network from adapting to the training set and thus generalising the problem better. To make a prediction, all of the units are active. (Srivastava et al. 2014)

- *Early Stopping*: The intuitively easiest approach is to stop training after the right number of epochs. If the neural network is training for a long time on the same data it is likely that it just remembers the example. For this approach, the model performance on a validation set (unseen data) is monitored during the training process. At the point where the generalisation error is not improving anymore, but the training loss is still dropping, the network starts to overfit. The goal is to find exactly this point. (Prechelt 1998)

- *Data Augmentation*: For convolutional neural networks, it makes sense to apply transformations to the raw images before feeding them to the network. In this way, more examples can be used to train the network and features can be leaned on a more general level. The network can be trained for more steps with less risk of overfitting the training data. With this technique, the network can also get more robust to the applied transformation. (Ding et al. 2016)

2.3.8 Hyperparameter Tuning

Hyperparameters are non-trainable variables of a model such as the number of layers and units, learning rate, dropout ratio, and penalty regularisation parameters. For large models training is time-consuming, hence, the number of trials is limited. One approach is by performing a grid search by systematically iterating through possible optimal solutions and picking the one with the best evaluation score on unseen data. A faster alternative is to use a random search strategy, such that fewer models have to be tested. Unfortunately, this strategy is not guaranteed to find the optimal solution (Bergstra et al. 2011). There are several other algorithms for hyperparameter optimisation which will not be discussed in this paper.

2.3.9 Training Strategy

As already mentioned in section 2.3.7, it is important to avoid overfitting on the training set by keeping an eye on the score of unseen data. However, for larger projects where many different models and hyperparameters are tested, there is also a risk of overfitting the unseen data set. Therefore, the samples are split into three sets: A *training set*, a *validation set* which is used to have an evaluation while tuning hyperparameters, and a *test set*, which is used to compute an unbiased final fitness of the model (Brownlee 2017). A common way is to use 80% of the available data for training and 10% each for validation and testing.

The unfortunate part is that only a fraction of the data can be used to check the performance of a model. The final score is biased based on how the data is split. Therefore, the concept of *k-fold-cross-validation* is applied, where the data set is split into k folds (k is usually chosen to be 5 or 10). Then, k models are trained and each is validated on a different fold while using the remaining folds for training. In the end, the validation results of all the folds are combined to one final score. K-fold cross-validation is very effective for hyperparameter tuning since it provides a less biased result. However, it comes with higher computational costs, since k models have to be trained. (Stone 1974)

3 Methodology

3.1 General Idea

As already mentioned in the introduction, the goal of this thesis is to automate the process of refining DSMs that are generated by dense photogrammetric reconstruction. The approach is to use a residual neural network to learn to predict the residuals f(X) for updating the initial DSM X (as shown in Figure 13). The model's output $\hat{\mathbf{y}}$ can be compared to the reference data $\mathbf{y}$ using a loss function. The loss function measures the discrepancy between the output $\hat{\mathbf{y}}$ of the model and the reference data $\mathbf{y}$. The measured discrepancy is then fed back to the model in order to update its weights and eventually to improve the prediction of the model.

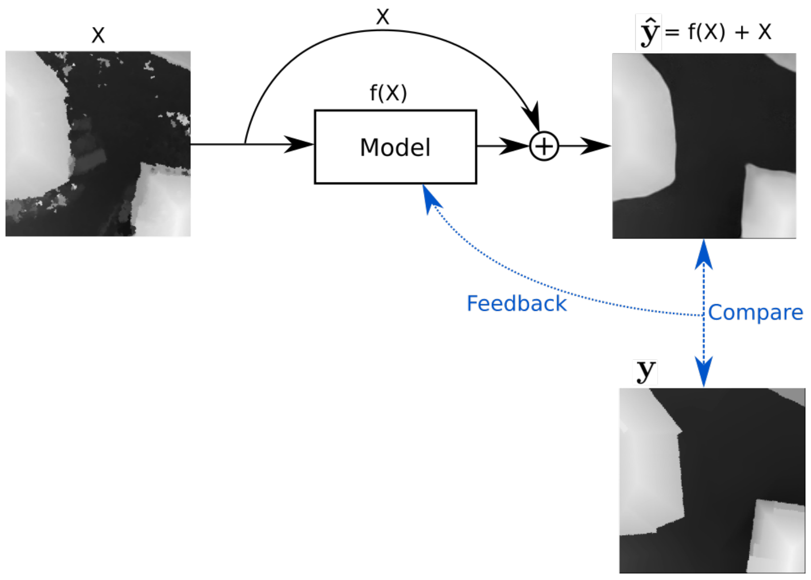


Figure 13: Flowchart of residual learning network and feedback loop.

3.2 Raw Data

3.2.1 Photogrammetric DSM

The input data of the model is a photogrammetric digital surface model (DSM) of the city of Zurich in Switzerland. The raw image data set is provided by the ISPRS/EuroSDR (2014) *Benchmark on High Density Image Matching.* The ground sampling distance of the aerial images is specified as $6 - 13$ cm. The covered area is approximately 1x1 km. The images are processed using the photogrammetric surface reconstruction software SURE (nFrames 2019), where a resolution of 10 cm is defined for computing the DSM.
The process only works with overlapping images. For the reconstruction of an object point, the point has to appear in at least two images. The points can then be automatically matched using dense matching algorithms. Near the border of the project area, there are fewer overlapping images. The risk that points cannot be reconstructed is much higher at the border then in the centre, when also taking into account the shadowing effects due to buildings and vegetation. Therefore, holes in the DSM are very likely to appear in these areas. The white areas in Figure 14 are pixels with no data values.

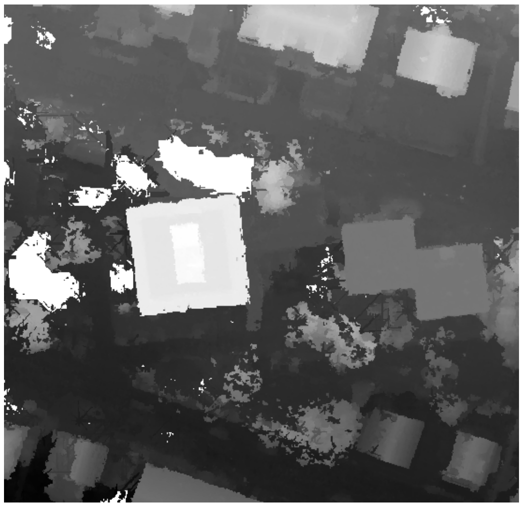


Figure 14: A grey-scaled visualisation of a raw DSM. Missing values are displayed in white.

Vegetation poses challenges to the image matching and depth map estimation procedure due to the repetitive texture of the tree crowns and grass. Furthermore, if the same tree is photographed from a slightly different angle, the pattern of the leaves changes due to coverages and shadows. This creates noise and outliers in the DSM that can be seen as the grey "clouds" in Figure 14.

3.2.2 Ground Truth Data

As ground truth data the 3D city model provided by (Open Data 2018b) is chosen. The data was last updated in 2015. Hence, the possible temporal differences between the city model of Zurich and the photogrammetric DSM (2014) is are expected to be small.
There are three different levels of detail (LOD) according to the CityGML 2.0 standard provided:

- *LOD 0*: This is a digital terrain model (DTM) of the city. The data does not include any objects such as buildings or trees. The model is gained from interpolation of LiDAR data.
- *LOD 1*: At this level of detail buildings are represented as blocks, but they do not have any detailed contours. The data is gained from the cadastral system (building footprints) and automatic photogrammetric measurements (building heights).
- *LOD 2*: The data contains buildings with modelled roof structures as shown in Figure 15. The data is gained from photogrammetric measurements and is processed in a semi-automatic manner. According to the technical information of Open Data (2018b), the data set has a mean position error of 10 – 15 cm and a mean height error of 20 cm.

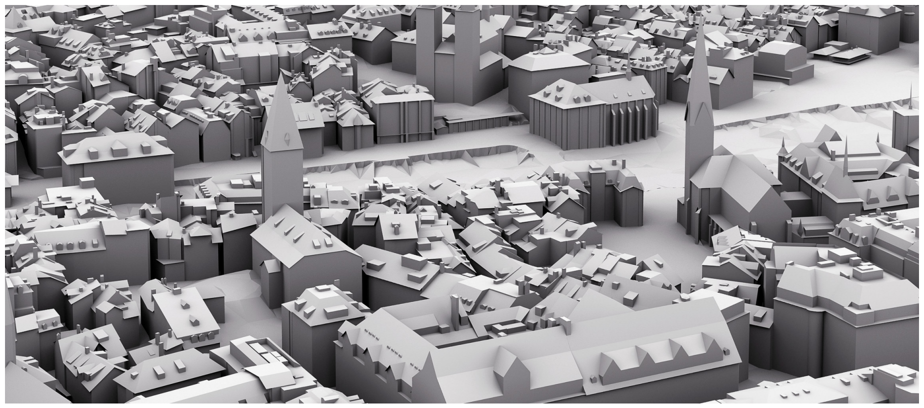


Figure 15: Visualisation of LOD 2. Note: The terrain in this image does not belong to the LOD 2 data set. (Open Data 2018a)

In this thesis, the DTM of the LOD 0 data set is combined with the LOD 2 data set, since the data set of LOD 2 contains only buildings and bridges but no terrain. The next section describe the methodology for merging these two data sets.

3.3 Data Processing

3.3.1 Preprocessing

The purpose of the preprocessing step is to bring the photogrammetric DSM and the LOD 0 and LOD 2 into the same raster format. This step is done using *ArcMap*.

- *photogrammetric DSM*: As already described in Section 3.2.1, photogrammetric measurements can lack data in some regions. Therefore, the holes are filled using the *focal statistic*-function in the *raster calculator*-tool. A mean kernel with radius size of 15 px is gradually applied to the pixels with no height values until all holes are filled. The kernel is intentionally set to be small because kernels that are too large are prone to introduce smoothing artefacts.
- *ground truth data*: The LOD 0 data set can be obtained as a *.shp* file which contains polygons with height references at their corners. First, a triangulated irregular network (TIN) is created using the *create TIN*-tool. Subsequently, the TIN is converted into a raster with cell size of 10 cm (the same resolution as the photogrammetric DSM). The LOD 2, was downloaded in the *multipatch* format. The function *multipatch to raster* is employed to convert the data into a raster representation. Lastly, the data sets are merged using the *raster calculator* by only keeping the higher value of both sets per raster cell.

After transforming both data sets to the same format, the photogrammetric data is first projected from the UTM system into the LV95 system. Furthermore, the undulation of the geoid (47.38 m) is also corrected, in order to obtain the same height system (LN02). The data is cut using the *clip raster*-tool. The clipping mask is a rectangle of size 10953x9933 px (= 1095.3x993.3 m in the real world) rotated by 20°.

3.3.2 Data Allocation

To train the neural network, the whole area has to be cut into smaller patches. Since correlations between pixels only appear in smaller neighbourhoods, it is not reasonable to have patches that are too large. Furthermore, large patch sizes lead to high memory consumption, therefore the batch size for mini-batch gradient descent has to be reduced.

However, setting the size of the patches too small will result in less context for the network. This means that the neural network is possibly unable to learn the characteristics of the data. Therefore, quadratic patches of size 1024 px, 512 px, 256 px, 128 px, 64 px, and 32 px (102.4 m, 51.6 m, 25.6 m, 12.8 m, 6.4 m, and 3.2 m respectively) are cut from the original set, creating six complete data sets. The experiment of Section 4.1 the patch size examines the most suitable patch size for the task.

The whole project area set is split between training, validation, and testing area. The test and validation areas each have a share of 9% and the training area covers 82% of the total area. Figure 16 illustrates the three areas.

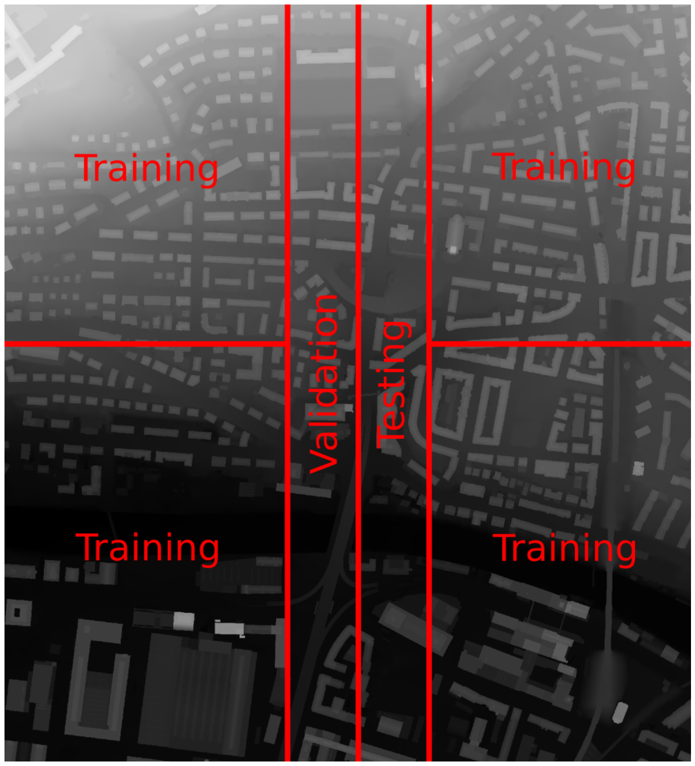


Figure 16: Data allocation of the complete area.

The validation and testing sets are cropped within vertical stripes across the whole area. The patches of the validation set are cropped in the shape of a grid. The test set is also cropped in a grid, but with an overlap of 128 px between adjacent patches (only for patch sizes 256 or higher) in order to visualise the final results without any boundary artefacts. In this data set, stripe-shapes are more representative for the whole data set than compact-shaped areas. A random selection of validation and test patches is not suitable either as such patches would be highly correlated to the training patches. The diversity in the validation set and the test set is necessary to check for biases in the performance of the network. All of the tree data areas are diverse in several aspects:

- The highest points of the region are in the top part and the lowest points in the bottom part. Therefore the set contains samples in the whole range of the height.
- Inside the stripes there are single-family homes in the top part, block buildings from some densely built-up regions in the centre, and industrial buildings at the bottom. The stripes also contain a part of a river.
- The stripes contain border regions with more artefacts, but also centre regions with cleaner data.
- There are areas in the stripes with a lot of vegetation but also areas with less vegetation.

The remaining area is assigned to the training set. It is split into four different tiles (as shown in Figure 16) such that every tile has a shape that is almost a square. The quadratic shape is more beneficial for data augmentation (see Section 3.3.3). The training samples are picked as stratified random examples. This means that the samples are picked randomly from a given sample area. For this project, the sample area is 16 times as large as the target size. The sample areas are cut as a grid. For each sample area, 22 patches are picked at random positions. Figure 17 shows an example for patch size of 256 px. The corresponding sample area has the size of 1024 px (the whole Figure 17). For visualisation reasons, only four of the 22 samples are marked. With this method, more different samples can be picked from the same area. It means that different contexts are created for the same overlapping regions which makes the training set more diverse. Using a grid to cut the training patches, there would be a small risk that the streets and houses are placed in the same spacing as the sampling distance of the grid. This would result in biased patches, that all have a similar content. With this random sampling, the risk of having biased patches is mitigated.

3.3.3 Data Augmentation

Data augmentation is applied to generate more training patches and to make the model more robust to unseen areas. Each of the four training sets shown in Figure 16 is rotated by 90°, 180°, and 270°. However, many walls are aligned to the cardinal points or to the inclination of the hillside. Therefore, the whole angle of 360° is split into eight equally

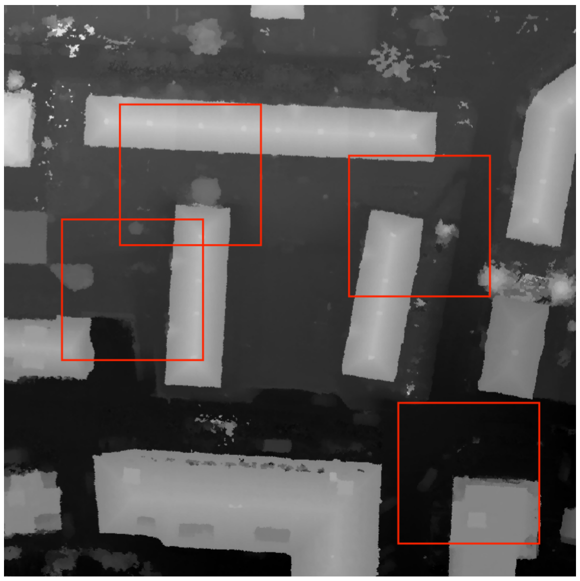


Figure 17: Stratified random sampling. The two patches at the left side share a small area. However, the context of the two patches is different.

spaced bins of each 45°(e.g. [0°, 45°], [45°, 90°], ... , [315°, 360°]). From each bin, one random rotation angle is picked. This results in eight more rotation angles for the four training regions. For every rotated area, the maximal inscribed, axis-aligned rectangle is calculated and resampled nearest-neighbour interpolation. The patches were then again sampled as stratified random examples. The patches of the rotated training sets are cropped without any scaling. The almost quadratic shape of the data set is beneficial because in this way the inscribed rectangle covers more percentage of the original area than with a slim rectangle. In this way, the data can be used without loosing too much of it. Furthermore, for each rotation angle, a vertically mirrored version is produced to double the number of total patches.

Zooming for data augmentation is not done since many houses have dimensions of Swiss construction standard. By scaling the patches, the model could learn the wrong standards. For the same reason scaling the height values is not done either. Applying a shift to the height values to produce more samples is omitted since the values are normalised anyway before the training. This normalisation procedure of the patches is explained in the next section.

3.3.4 Normalisation

As a last step of the data processing, the data is normalised by subtracting the mean height (computed by patch) and a global deviation of the heights derived from the training set of the photogrammetric DSM data. In this way, the training of the network is more stable and converges faster. The scaling had to be the same for every patch. A patch-wise normalisation would result in the patches with tall buildings being scaled to the same height range as patches showing low buildings. The network could face problems in distinguishing a sidewalk from a building. Furthermore, if the loss is calculated from patches which are scaled differently, it means that errors in flatter patches are scaled up. Hence, the network would treat those patches as more important than they truly are.
Mathematically the normalisation of the patch H_i can be expressed as follows:

$$H_i^{Norm} = \frac{H_i - \bar{z}_i}{\sigma_{global}} \tag{17}$$

The global standard deviation of the training area is $\sigma_{global} = 25.936m$. It is calculated with respect to the global mean $\bar{z}_{global} = 435.389m$. It should be noted, that for training, validation and testing the same σ_{global} has to be used. The variable $\bar{z}_i$ indicates the mean height of the patch H_i.

3.4 Network Architecture

The architecture is an encoder-decoder network with skip connections at each level of depth. The visualisation of the model for an input patch with size 256 px can be seen in Figure 18. The network consists of 43,265,345 trainable weights. The model is fully-convolutional which means that the number of parameters is independent of the input/output patch size. In this way, the model can be trained on smaller patches, while predictions can be made with larger sized patches. The trainable parameters are mainly the kernels of the convolutional layer and a few parameters of the PReLU activations.

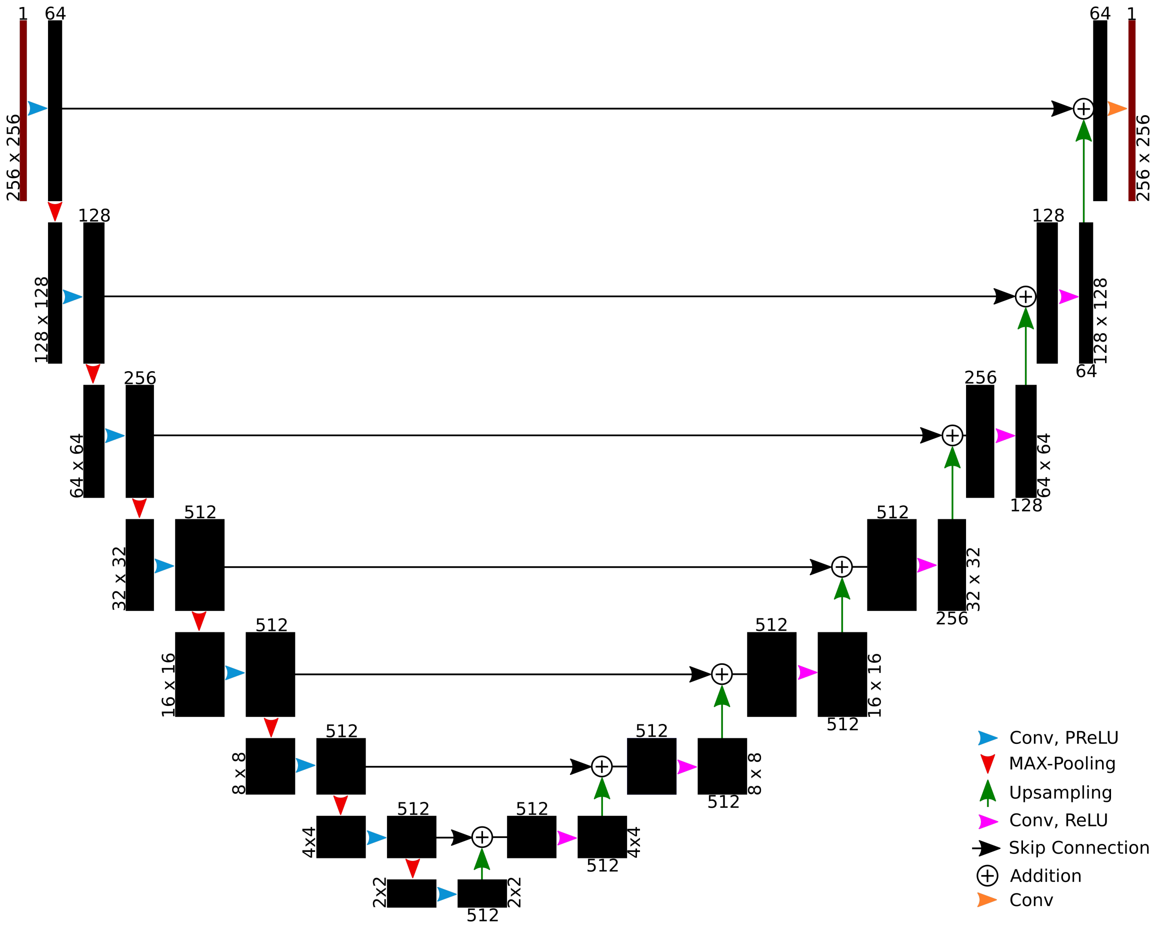


Figure 18: Architecture of the neural network.

Each black box in Figure 18 repesents a set of feature maps. The number of channels is specified at the top or bottom of the box and the image size is specified at the side of the boxes.

The left part of Figure 18 is called the encoder. It is built from a convolutional layer with a PReLU as its activation function (light blue arrow) and a MAX pooling operation (red arrow) to reduce the spatial dimensions. In the rest of this thesis, the sequence of these two operations is called *downsampling block.* The convolutional layers have a stride of 1, a 4x4 kernel, and operate with zero-padding. ReLU functions have the risk of becoming inactive for negative values, hence, the PReLU function is used. The same PReLU parameter is shared for each learned filter. The MAX pooling operation has a kernel and stride equal to 2. The idea is that the convolutional layer can specialise on pattern recognition and the MAX-pooling layer downsamples its input to the next depth level while keeping the important information.

The decoder part (right half of Figure 18) consists of a convolutional layer with ReLU activation (pink arrows) and simply upsampling operation by repetition of pixels (green arrow). The combination of these operations is called an *upsampling block* for the remaining part of this thesis. Here, the inactivity of the ReLU can be beneficial, since the outputs of the layers are residuals. An output of zeros means that there is no update to be added to the original input.

The last operation of the network is a convolutional layer with a linear activation (orange arrow). It is important to ensure that the final output of the model is not bounded such that every real number could be possible. A ReLU function would restrict the range of possible output values values to positive numbers.

The skip connections (black arrows) copy the outputs of the convolutional layers in the encoder block. This output is then added to the corresponding output of the upsampling layers in the decoder such that the dimensions match again (as shown with the addition symbol in Figure 18). Skip connections help to preserve high frequency details that are generally lost during the downsampling process of the encoder.

3.5 Loss Function

As a loss function, the mean absolute error (L1 norm) is used. For many regression problems, the L2 norm is a suitable loss function because it fits best for Gaussian distributions. Hence, the L1 norm is less robust to outliers than the L2 norm (Bektas and Sisman 2010). The data contains edges and corners contaminated by noise of an unknown distribution. Furthermore, outliers due to ambiguities in the point matching algorithm exist. It is important to note that minimising the L1 norm corresponds to minimising the mean absolute error (see Equation 18) since the only difference in the equation is a constant factor (the number of components N). Hence, the (global/local) maximum and minimum points of the loss function of the weights are still at the same positions. The same is true for the L2 norm and the mean squared error (see Equation 19).

$$\underset{\mathbf{w}_1^*}{\operatorname{argmin}} \|\hat{\mathbf{y}} - \mathbf{y}\|_1 = \underset{\mathbf{w}_1^*}{\operatorname{argmin}} \sum_{i=1}^{N} |\hat{y}_i - y_i| = \underset{\mathbf{w}_1^*}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} |\hat{y}_i - y_i| \tag{18}$$

$$\underset{\mathbf{w}_2^*}{\operatorname{argmin}} \|\hat{\mathbf{y}} - \mathbf{y}\|_2 = \underset{\mathbf{w}_2^*}{\operatorname{argmin}} \sqrt[2]{\sum_{i=1}^{N} (\hat{y}_i - y_i)^2} = \underset{\mathbf{w}_2^*}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 \tag{19}$$

In the these equations $\mathbf{w}_1^*$ and $\mathbf{w}_2^*$ indicate the weight configurations which minimise the corresponding loss functions.

For the illustration of the L1 and L2 norm, a simple example is considered. The goal of the model $C(x) = a$ is to estimate the correct height of the points, where a is a trainable parameter. The black points in Figure 19 represent the height attributes of a one-dimensional pixel array along the horizontal direction. The height attributes of the pixels exhibit a sudden discontinuity as it would also be possible for the edge of a building.

On the one hand, a L2 loss function would yield a unique solution $a = \frac{h}{2}$ (red line in Figure 19) which places the constant value exactly in the middle of the roof data points and the ground data points. The mean squared error of this solution amounts to $\frac{1}{8}\sum_{i=1}^{8}(\frac{h}{2})^2 = \frac{h^2}{4}$. On the other hand, a L1 norm does not have a unique solution to this problem setting. The mean absolute loss of the red solution corresponds to $\frac{1}{8}\sum_{i=1}^{8}\frac{h}{2} = \frac{h}{2}$. However, the L1 loss of the green solution is $\frac{1}{8}(\sum_{i=1}^{4}|h-s|+\sum_{i=1}^{4}|s|) = \frac{h}{2}$, for any s that fullfills $0 < s < h$. This implies that the L2 loss will always tend to favour solutions that go through the middle of roof and ground. This is highly unsuitable for this task, because it leads to smoothed edges, although the ground truth points are neither on the roof nor on the ground. The L1 norm has no such bias towards the middle. Theoretically, quasi-norm functions of degrees smaller than one would be even better. However, they lack numerical stability when

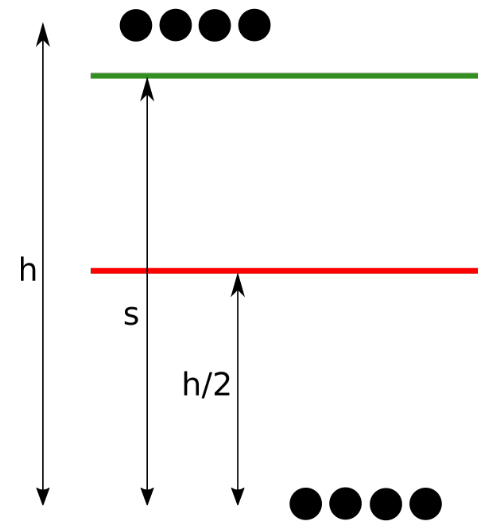


Figure 19: Behaviour of the L1 norm and L2 norm as loss functions.

optimising a network with millions of weights.

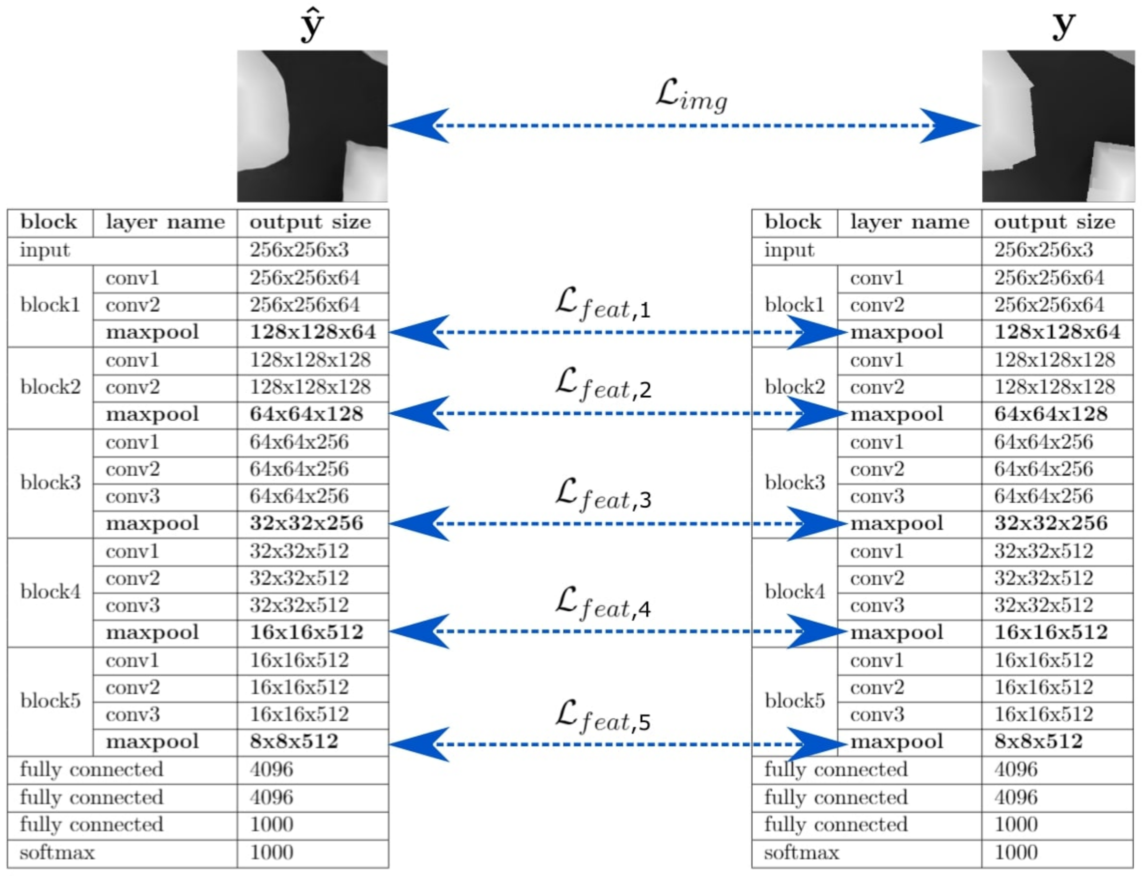
In addition to the pixel-wise L1 loss, a penalty regularisation is used as shown in Equation 20. A L1 weight regularisation $\mathcal{L}_{weights}$ with $\lambda_{weight} = 10^{-8}$ is applied according to the kernels of the convolutional layers. This regularisation encourages kernels with sparse weights (Friedman et al. 2008). Furthermore, an L1 activity regularisation $\mathcal{L}_{activity}$ with $\lambda_{activity} = 10^{-8}$ is applied to the outputs of the convolutional layers in the decoder. This means that residuals are penalised such that it only perform updates to pixels where it is really necessary. The loss function that is to be minimised is the following:

$$\begin{aligned}\mathcal{L}_{total} &= \mathcal{L}_{img} + \lambda_{weights}\mathcal{L}_{weights} + \lambda_{activity}\mathcal{L}_{activity} \\ &= \frac{1}{N_{img}}\|\hat{\mathbf{y}} - \mathbf{y}\|_1 + \lambda_{weights}\|\mathbf{w}\|_1 + \lambda_{activity}\|\mathbf{a}\|_1\end{aligned} \tag{20}$$

The variable **w** represent the weight vector of all kernels and **a** the vector of the activations in the decoder. N_{img} is the number of pixels in the image.

In a further experiment of Section 4.5, the loss function is extended by a perceptual loss term. The perceptual metric is obtained by comparing features from the VGG16 C network. VGG16 is a pre-trained encoder network designed for large-scale image recognition proposed by Simonyan and Zisserman (2014). The network is fully-convolutional and there-

fore usable for any input sizes that are dividable by 32. In Figure 20, the VGG is shown for an input size of 256x256x3 px. The network is designed to classify RGB images. Hence, the number of input channels must be three. Therefore, the output of the encoder-decoder model (Figure 18) is repeated three times and concatenated to obtain the required number of channels. The VGG16 is used with weights that are pre-trained on ImageNet (a large image classification data set from Stanford Vision Lab (2016)). ImageNet is a data set of over 14 million images which are assigned to 1000 labels by humans.
To calculate the perceptual loss term, both the prediction of the encoder-decoder network and the reference patch are fed into the VGG16 network. The calculated features of five intermediate layers (each at the end of a block, as shown in Figure 20) are picked and compared to each other. The layers are distributed over the whole network to find out in which depth the layers with the most useful feature lie.



block	layer name	output size
input		256x256x3
block1	conv1	256x256x64
	conv2	256x256x64
	maxpool	**128x128x64**
block2	conv1	128x128x128
	conv2	128x128x128
	maxpool	**64x64x128**
block3	conv1	64x64x256
	conv2	64x64x256
	conv3	64x64x256
	maxpool	**32x32x256**
block4	conv1	32x32x512
	conv2	32x32x512
	conv3	32x32x512
	maxpool	**16x16x512**
block5	conv1	16x16x512
	conv2	16x16x512
	conv3	16x16x512
	maxpool	**8x8x512**
fully connected		4096
fully connected		4096
fully connected		1000
softmax		1000

block	layer name	output size
input		256x256x3
block1	conv1	256x256x64
	conv2	256x256x64
	maxpool	**128x128x64**
block2	conv1	128x128x128
	conv2	128x128x128
	maxpool	**64x64x128**
block3	conv1	64x64x256
	conv2	64x64x256
	conv3	64x64x256
	maxpool	**32x32x256**
block4	conv1	32x32x512
	conv2	32x32x512
	conv3	32x32x512
	maxpool	**16x16x512**
block5	conv1	16x16x512
	conv2	16x16x512
	conv3	16x16x512
	maxpool	**8x8x512**
fully connected		4096
fully connected		4096
fully connected		1000
softmax		1000

Figure 20: Construction of a perceptual loss function by using features of the VGG16 network.

The features ($\hat{\mathbf{y}}_{feat,i}$ and $\mathbf{y}_{feat,i}$) are compared via a mean absolute error function and added to the loss function as the term $\mathcal{L}_{feat}$. The extended loss function can be expressed as:

$$\begin{aligned}\mathcal{L}_{total} &= \lambda_{img}\mathcal{L}_{img} + \lambda_{weights}\mathcal{L}_{weights} + \lambda_{activity}\mathcal{L}_{activity} + \lambda_{feat}\mathcal{L}_{feat} \\ &= \frac{1}{N_{img}}\lambda_{img}\|\hat{\mathbf{y}} - \mathbf{y}\|_1 + \lambda_{weights}\|\mathbf{w}\|_1 + \lambda_{activity}\|\mathbf{a}\|_1 + \lambda_{feat}\mathcal{L}_{feat} \end{aligned} \tag{21}$$

where the feature loss is calculated as:

$$\mathcal{L}_{feat} = \sum_{i\in\mathcal{F}} \lambda_{feat,i}\mathcal{L}_{feat,i} = \sum_{i\in\mathcal{F}} \frac{1}{N_i}\lambda_{feat,i}\|\hat{\mathbf{y}}_{feat,i} - \mathbf{y}_{feat,i}\|_1 \tag{22}$$

$\mathcal{F}$ is the set of the selected layers (according to Figure 20) and N_i is the number of elements in the i^{th} feature layer.

In the experiment of Section 4.5, the scaling factors (λ_{img} and λ_{feat}) are set such that the order of magnitude of $\lambda_{img}\mathcal{L}_{img}$ is the same as $\lambda_{feat}\mathcal{L}_{feat}$ (see Equation 23). Furthermore, if two or more feature loss terms are used for an experiment, the scaling factors $\lambda_{feat,i}$ (for all the layers $i \in \mathcal{F}$) are all set such that the order of magnitude of each term $\frac{1}{N_i}\lambda_{feat,i}\mathcal{L}_{feat,i}$ is the same (see Equation 24). To not influence the regularisation terms, the sum of λ_{img} and all the terms $\lambda_{feat}\lambda_{feat,i}$ (this product results if $\mathcal{L}_{feat}$ of Equation 22 is inserted into the loss function of Equation 21) is constrained to be 1 (see Equation 25). In this way, all the non-regularisation terms (image loss and feature loss) will be of the same order of magnitude across every experiment.

The constraints on the lambda values can be summarised as:

$$\log\left(\lambda_{img}\mathcal{L}_{img}\right) \approx \log\left(\lambda_{feat}\mathcal{L}_{feat}\right) \tag{23}$$

$$\log((\lambda_{feat,i}\mathcal{L}_{feat,i})) \approx \log((\lambda_{feat,j}\mathcal{L}_{feat,j})), \quad \text{for} \quad i,j \in \mathcal{F} \quad \text{and} \quad i \neq j \tag{24}$$

$$\lambda_{img} + \sum_{i\in\mathcal{F}} \lambda_{feat,i}\lambda_{feat} = 1 \tag{25}$$

3.6 Implementation and Training

The model is implemented using Keras version 2.2.4 with TensorFlow as a backend (Chollet et al. 2015; Martín Abadi et al. 2015). The network is trained with an Adam optimiser with a learning rate of 10^{-5}. The momentum parameters are set to the default values of $\beta_1 = 0.9$ and $\beta_2 = 0.999$ (Kingma and Ba 2014). For the patch size of 256x256 px, the training data set contains 25'000 samples, including data augmentation. Both the validation set and testing set each consists of 160 patches, where no data augmentation is applied. The model is trained for 256 epochs with mini-batch gradient descent of size 12 samples. To pick the samples, a generator function is used. In this way, not all the training samples (roughly 225 GB) have to be loaded in the RAM. The training samples are stored in *.npy*-files in batches of size 200. This is done to reduce the number of accesses and thus save time in the training process of the neural network. Furthermore, TensorBoard from the TensorFlow library is used (Martín Abadi et al. 2015). It is important to note, that the validation loss (mean absolute error and mean squared error) in this setup is not a reliable indicator for the model performance. It can even be misleading as shown in Section 2.3.2 with Figure 10. Also the observation of the training/validation feature loss is not suitable, since *a priori* the characteristics and meaning of the features is unknown. In contrary, this thesis should examine which of the features of the VGG16 have the desired characteristics to build a perceptual metric. Therefore, the predictions of the models are mainly inspected qualitatively.

4 Results and Discussion

This chapter describes the conducted experiments with their outcomes. It examines the behaviour of the model performance depending on the model architecture and hyperparameters. The first four experiment use L1 loss function. The fifth experiment investigates, whether enhancing the loss function with a perceptual loss metric can be used to improve the results.

4.1 Patch Size

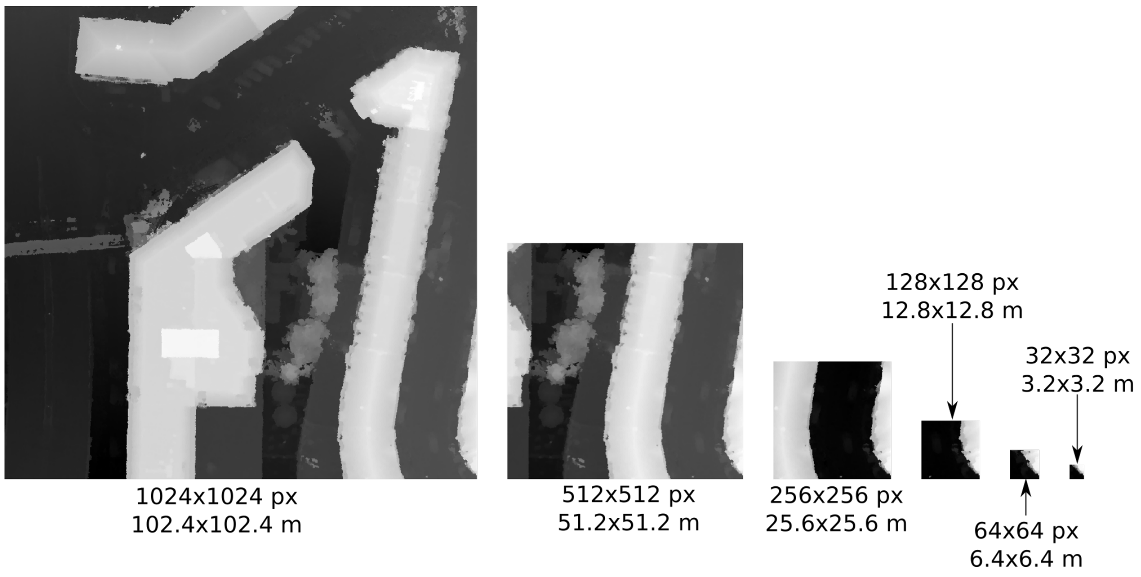


Figure 21: Visualisation of different patch sizes. Note that the colour mapping is unique for every patch in this figure and therefore not comparable.

The first experiment setup investigates the influence of the patch size on the prediction performance. The network is trained in patches because it breaks down the whole problem into smaller sized areas. Quadratic patches of size 1024 px, 512 px, 256 px, 128 px, 64 px, and 32 px (ground sampling distance: 102.4 m, 51.2 m, 25.6 m, 12.8 m, 6.4 m, and 3.2 m respectively) are tested. Figure 21 shows a comparison of the different sizes.

On the one hand, when using patches of size 32 px, 64 px and 128 px, the training loss diverges towards infinity. This means that the learning process fails. It even diverges when lower learning rates are used.

On the other hand, using patches of size 512 px or 1024 px results in unstable learning, and therefore, to inconsistent results. These larger patches result in increased memory consumption during training. As a consequence, the size of the mini-batch for the stochastic gradient descent has to be reduced to ensure that the resources will not be exhausted. For the patch size of 1024 px, the mini-batch has to be reduced to only one training sample per batch. For the size of 512 px the maximum number of patches is 3 and for the size of 256 px 12 patches can fit in a batch.
The input size of 256 px yields reproducible results. This section does not include visualisations of the predicted height maps, since the important learning of this experiment is the behaviour of the learning process.

The smaller the patches, the less context they include. In terms of convolutional neural networks, context is important to detect patterns and objects. Considering the smaller patches of Figure 21: even for a human it is hard to tell if the patch includes a tree or a building. Furthermore, many of the smaller patches do not even contain an edge as in the shown example. A significant part of them contain almost flat surfaces with noise in them. This makes it difficult to find any patterns in the patches. For this setup the learning process even fails due to the lack of context area.
It should be mentioned that in this setup, a smaller batch size does not mean that less training area is used per step. Reducing the batch size from 12 to 3 while increasing the (quadratic) patch size from 256 px to 512 px results in the same size of the area per batch. However, with smaller patch sizes, the samples can be randomly distributed over the whole training area. This is better for the stability of the learning process, since in this way the area is more representative for the whole training area.

The patch size 256 px is the most suitable, because it allows the largest batch size and still converges. The following experiments are all conducted with a patch size of 256 px and a batch size of 12.

4.2 Model Depth

As already mentioned in the Section 3.4, the chosen architecture of the network is modular. Figure 18 has seven downsampling blocks and seven upsampling blocks. This experiment analyses how the depth of the network influences the performance of the model. In this setup, the number of downsampling block and upsampling blocks from 1 through 8 is tested.

Figure 22 illustrates the outcome of the experiment. Both examples show that the first four levels of depth are required to remove vegetation. Furthermore, the comparison shows that the level of complexity of 4 through 7 are necessary to obtain crisp borders of buildings. The model with seven down- and upsampling operations performs significantly better than its predecessors. It seems that eight down- and upsampling operations do not contribute to the performance. Instead, it introduces more degrees of freedom to the network which enlargers the risk of overfitting.

The results demonstrate that seven down and seven upsampling operations is the optimal choice to obtain sharp edges in the prediction.

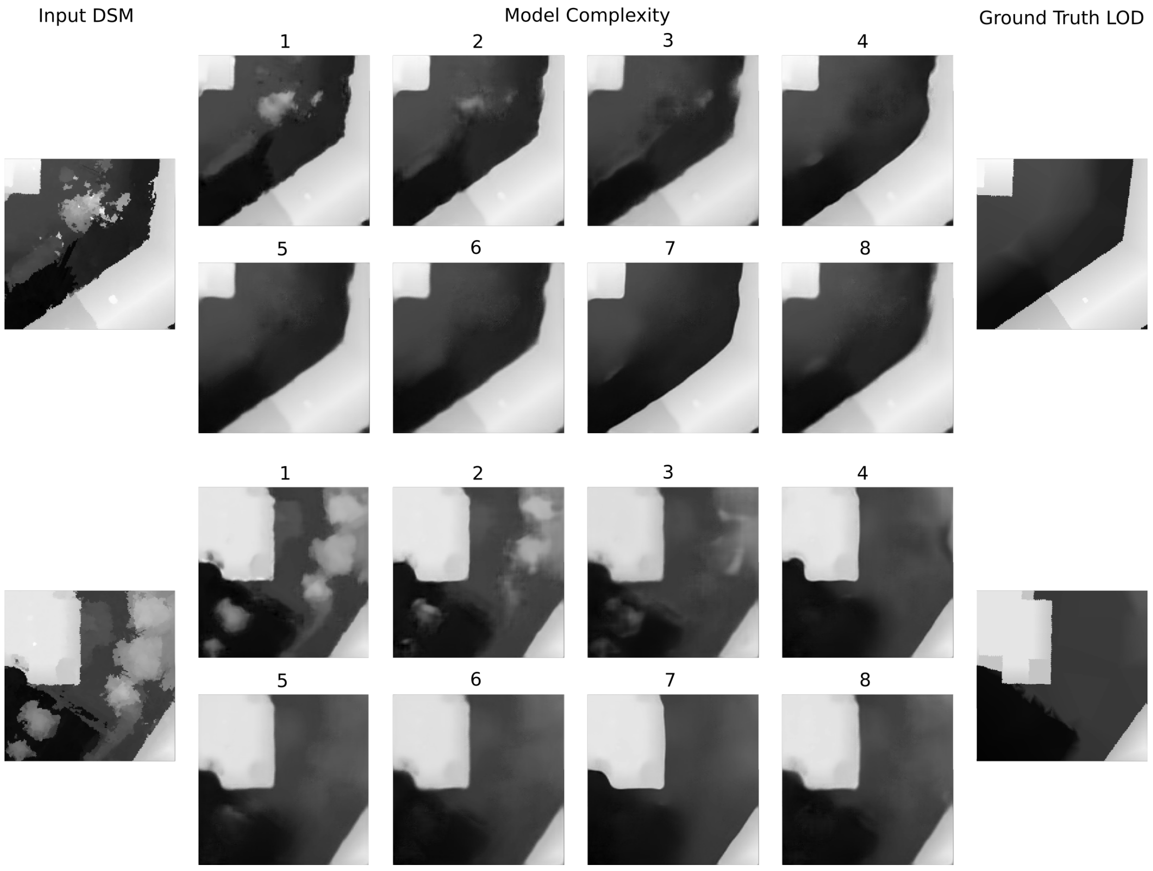


Figure 22: Result of the experiment to analyse the model's depth. The illustration visualises two representative patches of the validation set. The input DSM is arranged on the very left column. The four columns in the middle are the predictions of the trained models. The digit above the images indicates the numbers of down- and upsampling operations performed by the model. The very right column shows the corresponding ground truth patches.

4.3 Skip Connections

This experiment is set up to investigate the assumed advantages of residual learning that are previously described in Section 2.2.5. Two models are trained in the same way. In the first model, the skip connections (black arrows of Figure 18) which connect the encoder to the decoder are omitted. The second model operates as in Figure 18.

Figure 23 illustrates the outcome of the experiment with two example patches. The most striking aspect of the results is that the geometries are not preserved if no skip connection (the middle left column in Figure 23) are incorporated into the model. With this architecture, the model has to reconstruct the geometries of the buildings from a sparse encoding. The network is underfitting. Furthermore, the reconstructed edges and corners are very fuzzy. In contrast, the second model is capable of preserving the geometries of the objects in the patches. Consequently, the patches have a much lower error caused by incorrect geometries. The network can focus its capacity on denoising the input. The DSMs of Figure 23 show that edges drawn from the model with skip connections are much sharper. The profile lines illustrate the same argument as the grey-scaled images. Especially in the bottom example, it can be seen that the model (orange lines) erroneously approximates the bulge in the terrain as a plane, and therefore, underfits the data.

The experiment shows that skip connections are required for solving the task of DSM refinement.

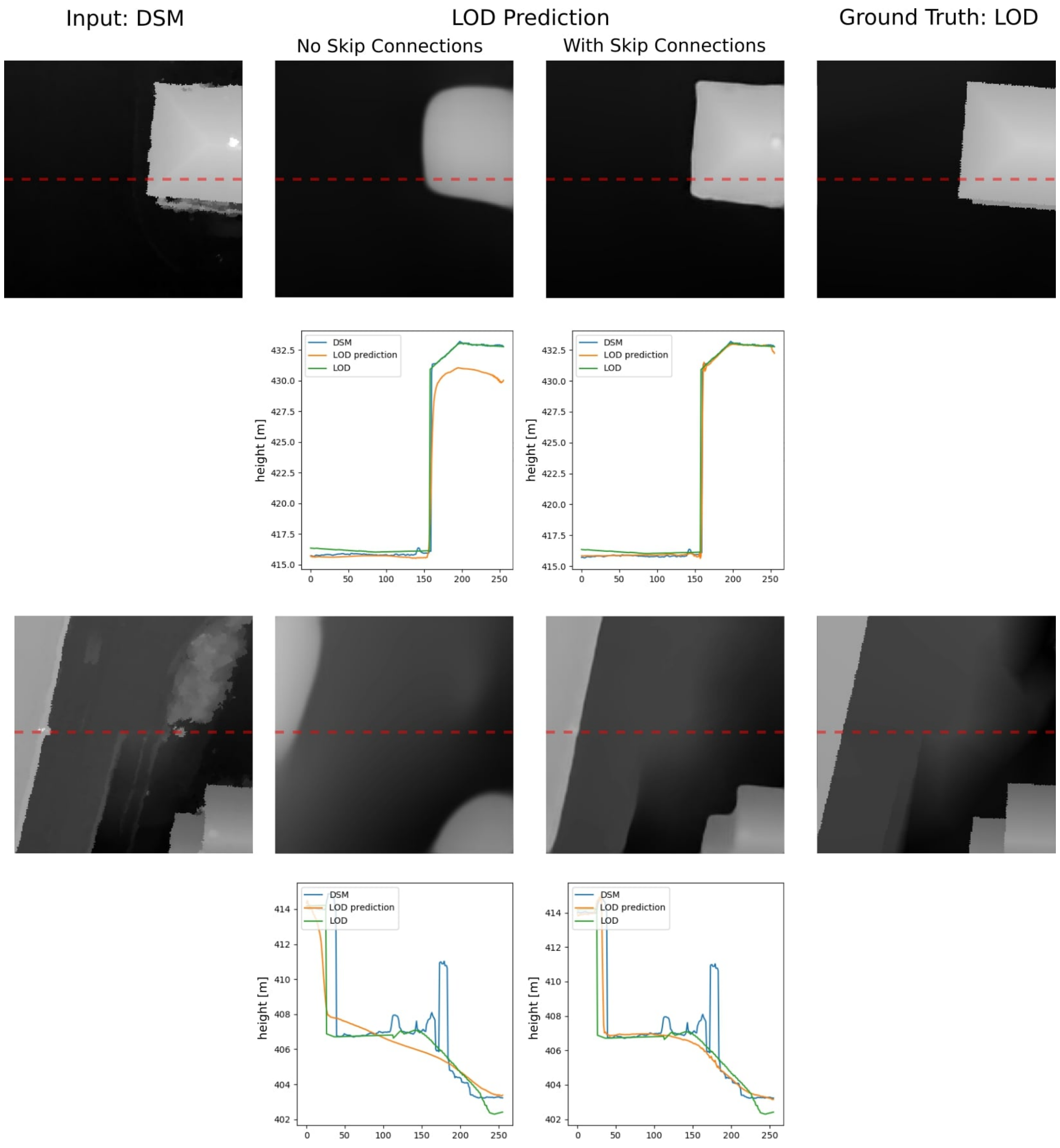


Figure 23: Results of the experiment to examine the influence of skip connections. The figure visualises two independent patches of the validation set. The left column shows the given input DSM. The middle left column visualises the prediction of the model without skip connections. The subsequent column contains the prediction of the model with skip connections. The last column shows the reference patches. The profile lines are obtained by cutting the DSMs along the dashed lines. Each profile plot contains the input DSM (blue), the predicted LOD of the corresponding model (orange), and the ground truth LOD (green).

4.4 Pixel-wise Loss Metric

In another setup, the influence of the loss function is analysed. The same model is trained with two different loss functions. In the first trial, the model is trained to minimise the mean squared error (e.g. L2 norm of the pixel errors), whereas the second one is trained to minimise the mean absolute error (e.g. L1 norm of the pixel errors).

Figure 24 summarises the outcome of the experiment. The results illustrate the differences in the behaviour of the two loss functions. The model trained with the mean square error loss function (L2 norm) performs inferiorly. It yields predictions with blurry edges. Furthermore, the visualisations of the predictions are significantly greyer than the reference patches. In other words, the height values have a bias in the direction of the mean height as explained in Section 3.5 and Figure 19. The predictions of the model trained with the mean absolute error function (L1) are considerably more accurate as they are not affected by the aforementioned bias. Hence, the model can focus on optimising the correctness of the edges.

The comparison of these two loss functions shows that the L1 loss function is more suitable for this task.

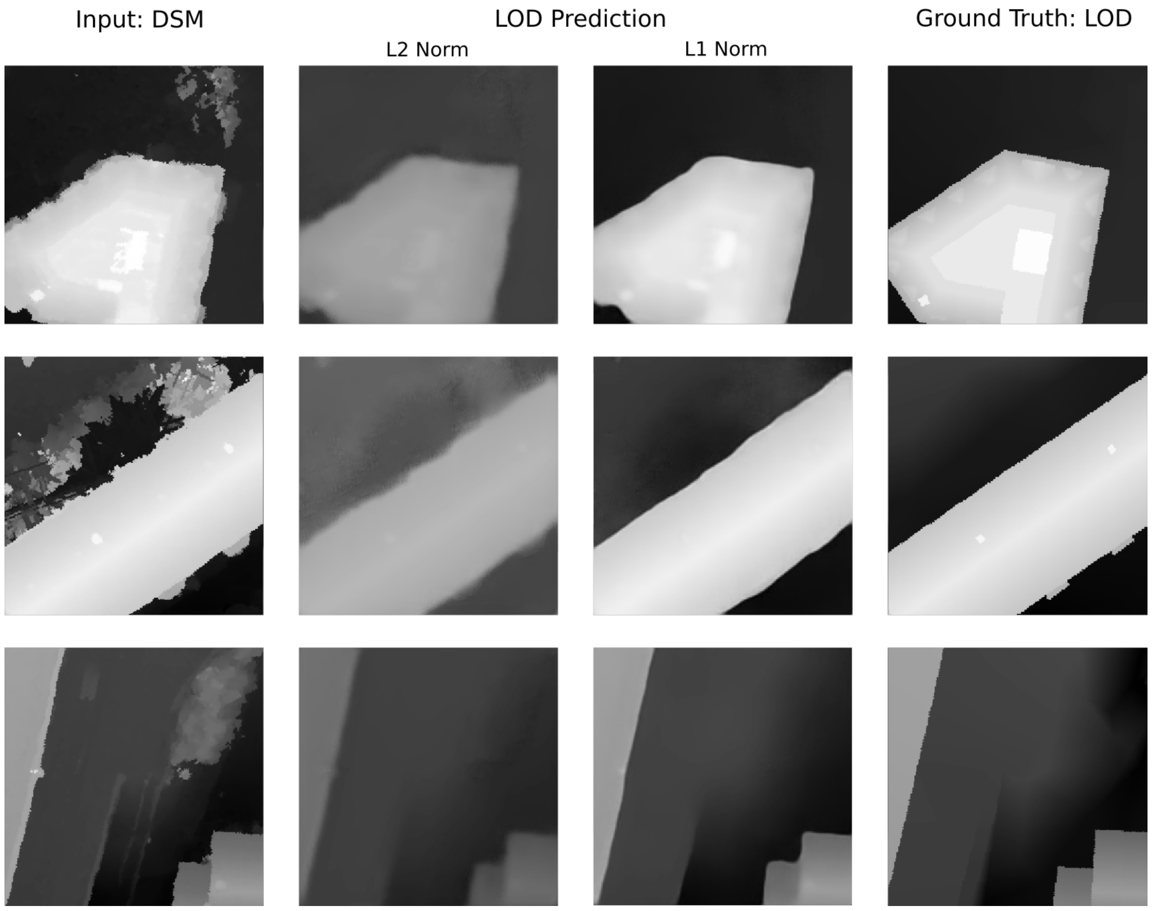


Figure 24: Visualisation of three predictions examples of a model trained with a mean squared error loss function (e.g. L2, middle left column) and a model trained with a mean absolute error loss function (e.g. L1, middle right column). The left most column illustrates the input data patches, whereas the right most column shows the corresponding ground truth LOD patches.

4.5 Perceptual Loss Metric

This experiment extends the previous one by adding a feature loss term to the L1 loss function as suggested in Equation 21 and 22. Each of the five selected feature layers that are shown in Figure 20 are tested separately by adding them to the loss function. Furthermore, several kinds of combinations between the layers are tested. As it turned out, the layers 3 through 5 contain features that make the prediction even worse. Therefore, these results are not discussed. Figure 25 illustrates how the model behaves when the loss function is extended by the loss terms of layers 1, 2, 1+2, and compares them to the result of the previously used L1 pixel loss of Section 4.4.

One important criterion for this task is the sharpness of the edges of buildings. The model trained with the feature layer 2 produces significantly more blurred edges compared to the others. Therefore, this model is not useful for this task. The qualities of the edges predicted by the other three models are comparable.

The model which is trained on the regular pixel-loss function produces small artefacts (red circles in Figure 25) due to imperfect removal of vegetation. In contrast, the models trained with a feature loss are all able to remove the vegetation without any artefacts.

The input DSM of the fourth patch in Figure 25 is distorted because the point matching algorithm does not perform well on the pedestrian ramp. The desired behaviour of the model would reconstruct the ramp from the noisy input. If the provided input information about the ramp is not sufficient, the desired behaviour would be to completely remove the ramp, such that the landscape looks realistic. However, the model trained on the pixel-loss function does not act in any of these ways. It produces unrealistic artefacts as indicated by the blue circle. The three patches at the right of this patch show that a perceptual metric is able to penalise the geometry of objects such that unrealistic shapes are mitigated.

To take the thought a bit further, the loss function should favour geometries with straight contours as typical for man-made objects. The yellow arrows in Figure 25 indicate examples of such failure cases. For example, the prediction of the third patch with the pixel-loss-model yields a corner of a house that is rounded. The model trained with the first feature layer predicts the corner to be too pointy. A trade-off between the two variants is provided by the model trained on the loss function containing both feature layers. The predicted corner of this model is the closest to a perpendicular corner.

In the last two examples, man-made structures are pushed down (green circles) when they are predicted with the model trained on the pixel loss. This behaviour also appears in some of the models trained with a perceptual loss. However, the model trained with the feature layers 1+2 seems not to produce these kinds of distortions.

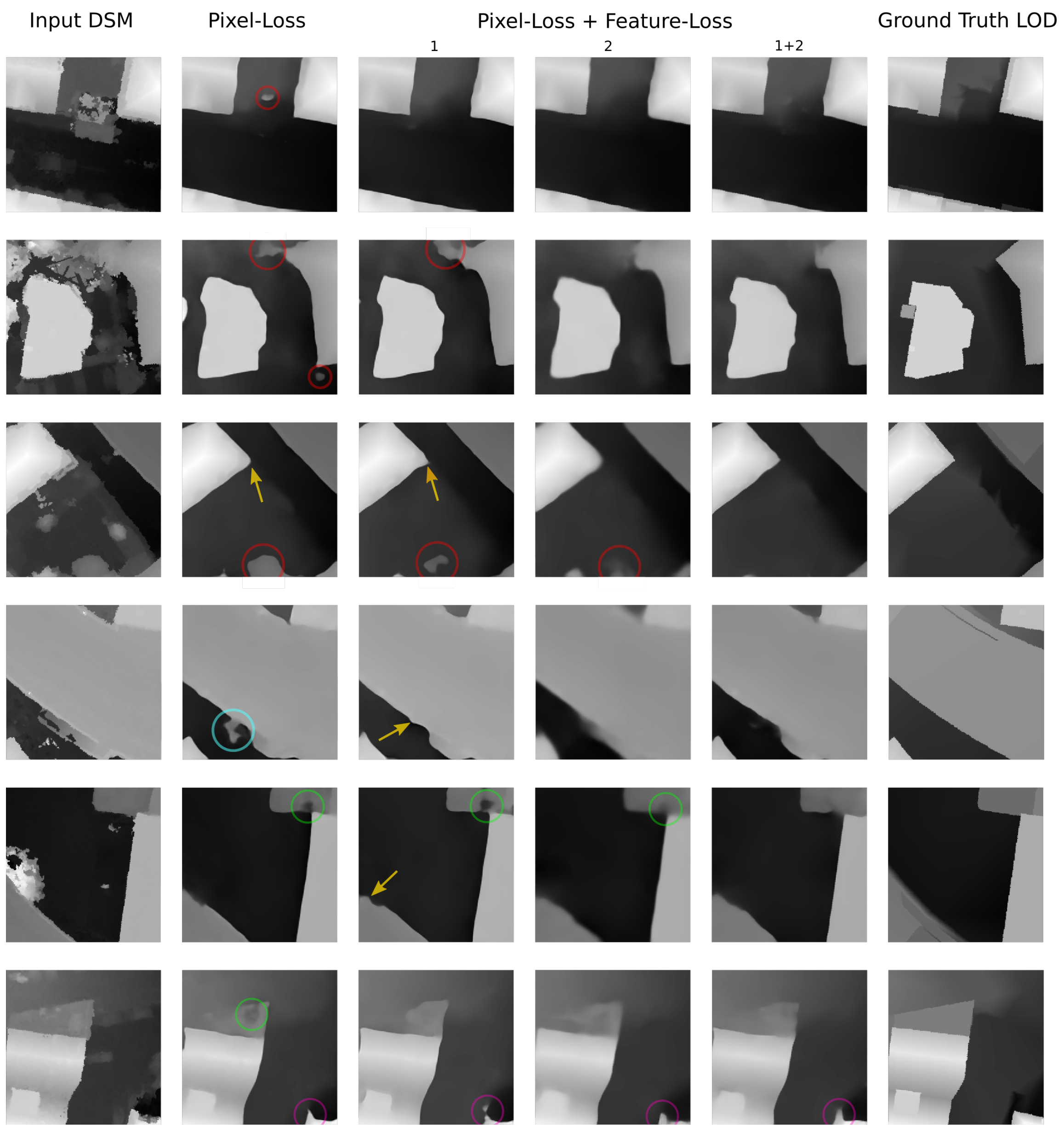


Figure 25: Illustration of the model's predictions on 6 example patches (one for each row). The most left column shows the input DSMs. The prediction performance of a model trained with a mean absolute error (e.g. L1 norm) is shown in the second left column. Columns 3 to 5 are the visualisation of the results generated with a model trained for the enhanced loss function. The loss function is extended by layer 1, 2 and 1+2 respectively, according to the definitions of Figure 20. The right column contains the reference patches.

The last class of artefacts (pink circles) represents mistakes that are caused by a lack of context. In the last example, the input information about the house in the bottom left corner is very sparse. None of the trained models is able to perform a reasonable update for this building. However, it should be mentioned that the border pixels can be ignored if the patches are predicted with an overlap. Furthermore, for predictions on the test set, patches of larger size can be used due to the fully-convolutional property of the network. Therefore, mistakes at the border of the patches are not rated as crucial.

To sum up this experiment, training the model with a pixel-loss function and the feature loss of the layers 1+2 yields the best performance. Even though the predictions are not perfect, this configuration can handle the challenges of this task appropriately.

5 Conclusion

This thesis addresses the problem of DSM refinement for urban areas with a convolutional neural network-based approach. The key idea is to learn typical urban structures from existing reference data instead of expressing these structures mathematically. On the one hand, the architecture of the neural network is decisive whether generated outputs contain realistic and correct objects. This thesis proposes an encoder-decoder architecture based on the concept of residual learning, which is superior over standard encoder-decoder models regarding the the quality of the geometries. On the other hand, the quality of the prediction strongly depends on the choice of the loss function. The experiments also show that the L1 norm is suitable as a loss function for problem set. Furthermore, it can be inferred that the introduction of perceptual loss terms to the loss function increase the performance of the model. The conducted experiments also show that a patch size of 256x256 px ($\approx$ 25x25 m) is an adequate choice. It provides a compromise between having sufficient context while also being able to diversify the mini-batch with a large batch-size.

It can be concluded that the objective of this thesis has been achieved. It was possible to implement a working neural network, which cleans the artefacts of the photogrammetric generated DSM. Figure 26 shows the prediction performance of the final model. The model is able to remove vegetation and outliers from DSMs. Furthermore, the model has learned that buildings are built according to simple geometric standards such as straight walls with planar roof structures. However, the predictions on the test area show that the model is not yet perfect. It would be desirable that smaller structures like balconies are learned too. Furthermore, the model still yields results with walls that are not exactly straight and vegetation is not always removed completely.

The model achieves 74% of the test pixels correctly predicted. A pixel is treated as correct if the prediction is within 0.5 m of the reference height. Figure 31 in the Appendix B illustrates the distribution of errors as a histogram. The mean absolute error is 0.65 m, while the median absolute error is 0.23 m. Note that the input DSM and the ground truth data also have a temporal difference, such as new buildings in Figure 29 of the Appendix A.

The trained model can be applied to unseen data sets with only a few limitations. It learned the architectural style of the training area. It is likely that the model can perform well in cities that are built with similar architectural styles Zurich. However, predictions of this quality cannot be guaranteed for cities that have completely different urban structures. In this case, it is required to train the model on a corresponding training area, before making

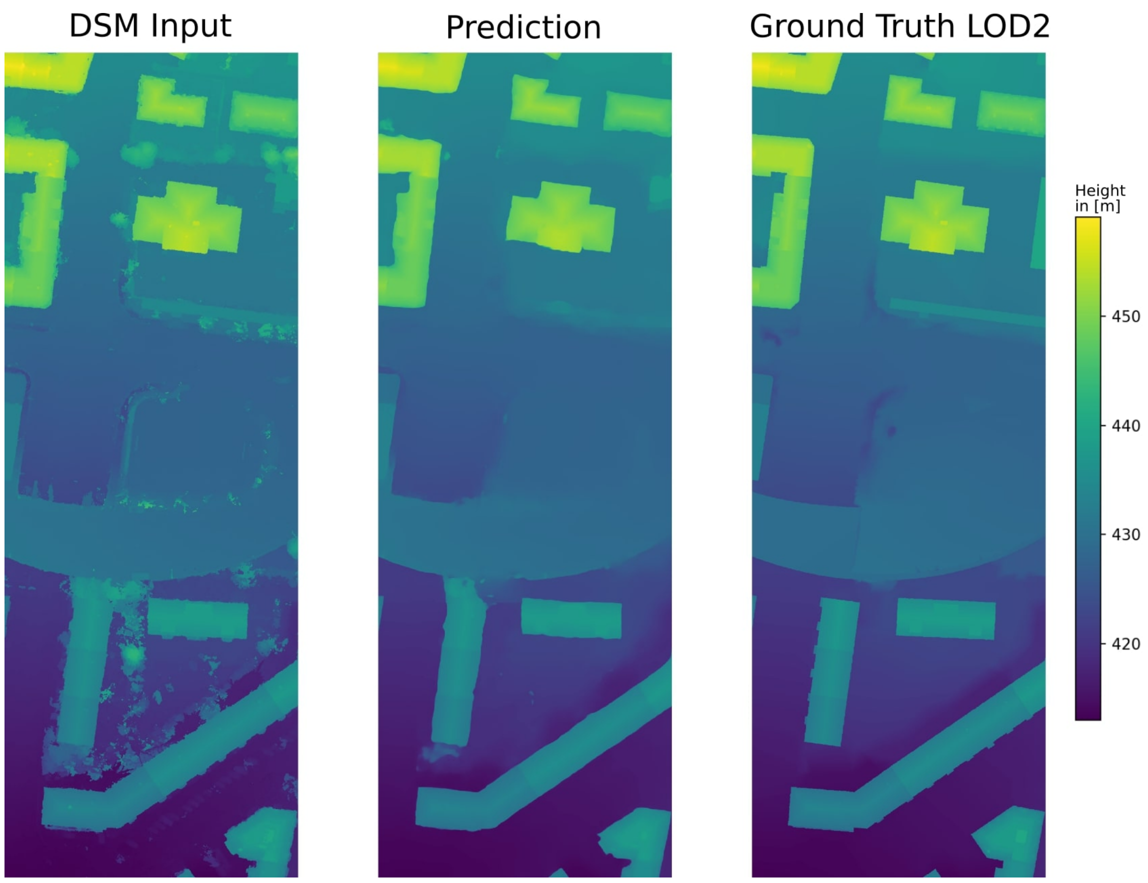


Figure 26: Strengths and weaknesses of the model. The illustration shows a snippet of the test area. From left to right: the given input to the model, prediction of the model, corresponding ground truth patch. The model is trained on patches of 256x256 px, whereas this prediction is calculated using patches of size 1024x1024 px. In this way, the number of pixels with fewer context can be reduced. The single tiles have an overlapping area of 128 px an the whole patch consists of three separate predictions. Appendix A contains the prediction of the final model on the remaining test area.

predictions on such an area. Furthermore, the model is trained with a pixel size of 10 cm. The model is tied to this specification. It learned to measure structures and standards in those pixel units. Therefore, it makes sense to convert any photogrammetric data to a pixel size of 10 cm before applying the model. It is important to recall that the model is not able to deal with missing data. Therefore, the unknown values in the input data have to be interpolated in a preprocessing step.

Several approaches are conceivable to increase the model performance:

- Using a larger training area can improve the result. In this way, the model is able to learn more structures of urban landscapes and potentially generalises better.
- Another approach is to feed in more input information such as the orthophoto generated from the raw aerial images. However, the problem is that for the generation of an orthophoto a height model of the terrain is needed.
- In this thesis, only five layers of the VGG16 network are investigated in order to construct a perceptual loss metric. The layers at the beginning of the network showed promising features. It is likely that the other layers at the top also contain useful features that can be used. Furthermore, it can be considered to use features of another pre-trained network.
- There are already promising results in image segmentation that apply the same encoder-decoder model multiple times in order to archive better results (Mosinska et al. 2018). It is possible that the methodology could also work for this task.

Bibliography

Bektas, Sebahattin and Yasemin Sisman (2010). “The comparison of L1 and L2-norm minimization methods”. In: *International Journal of the Physical Sciences* 5, pp. 1721–1727.

Bergstra, James S et al. (2011). “Algorithms for hyper-parameter optimization”. In: *Advances in neural information processing systems*, pp. 2546–2554.

Biljecki, Filip et al. (2015). “Applications of 3D city models: State of the art review”. In: *ISPRS International Journal of Geo-Information* 4.4, pp. 2842–2889.

Brownlee, Jason (2017). *What is the Difference Between Test and Validation Datasets.* URL: `https://machinelearningmastery.com/difference-test-validation-datasets/` (visited on 05/01/2019).

Chollet, François et al. (2015). *Keras Documentation.* URL: `https://keras.io` (visited on 04/25/2019).

Chollet, François (2017). *Deep learning with Python.* Manning Publications Company.

Cornelisse, Daphne (2018). *An intuitive guide to Convolutional Neural Networks.* freeCodeCamp. URL: `https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050` (visited on 05/17/2019).

Ding, Jun et al. (2016). “Convolutional neural network with data augmentation for SAR target recognition”. In: *IEEE Geoscience and remote sensing letters* 13.3, pp. 364–368.

Donges, Niklas (2018). *Gradient Descent in a Nutshell.* Towards Data Science. URL: `https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0` (visited on 04/25/2019).

Dosovitskiy, Alexey and Thomas Brox (2016). “Generating images with perceptual similarity metrics based on deep networks”. In: *Advances in neural information processing systems*, pp. 658–666.

Duchi, John, Elad Hazan, and Yoram Singer (2011). “Adaptive subgradient methods for online learning and stochastic optimization”. In: *Journal of Machine Learning Research* 12.Jul, pp. 2121–2159.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani (2008). "Sparse inverse covariance estimation with the graphical lasso". In: *Biostatistics* 9.3, pp. 432–441.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep learning*. MIT press.

He, Kaiming et al. (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

– (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

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

ISPRS/EuroSDR (2014). *Benchmark on High Density Image Matching*. URL: `http://www.ifp.uni-stuttgart.de/ISPRS-EuroSDR/ImageMatching/index.en.html` (visited on 05/03/2019).

Jadon, Shruti (2018). *Introduction to Different Activation Functions for Deep Learning*. Medium. URL: `https://medium.com/@shrutijadon10104776/survey-on-activation-functions-for-deep-learning-9689331ba092` (visited on 04/29/2019).

Karpathy, Andrej (2019). *CS231n: Convolutional Neural Networks for Visual Recognition*. Stanford CS. URL: `http://cs231n.github.io/` (visited on 04/25/2019).

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*, pp. 1097–1105.

Martín Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: `https://www.tensorflow.org/`.

Mosinska, Agata et al. (2018). "Beyond the pixel-wise loss for topology-aware delineation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3136–3145.

nFrames (2019). *SURE aerial*. URL: `https://www.nframes.com/` (visited on 05/30/2019).

Open Data, Stadt Zürich (2018a). *3D-Dachmodell (LOD2)*. URL: `https://data.stadt-zuerich.ch/dataset/geo_3d_dachmodell_lod2` (visited on 04/29/2019).

– (2018b). *3D-Stadtmodell*. URL: `https://www.stadt-zuerich.ch/ted/de/index/geoz/geodaten_u_plaene/3d_stadtmodell.html` (visited on 04/29/2019).

Prechelt, Lutz (1998). "Early stopping-but when?" In: *Neural Networks: Tricks of the trade*. Springer, pp. 55–69.

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556*.

Srivastava, Nitish et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Stanford Vision Lab Stanford University, Princeton University (2016). *ImageNet*. URL: `http://www.image-net.org/`.

Stone, Mervyn (1974). "Cross-validatory choice and assessment of statistical predictions". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 36.2, pp. 111–133.

Sutskever, Ilya et al. (2013). "On the importance of initialization and momentum in deep learning". In: *International conference on machine learning*, pp. 1139–1147.

Tieleman, Tijmen and Geoffrey Hinton (2012). "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural networks for machine learning* 4.2, pp. 26–31.

Zhang, Richard et al. (2018). "The unreasonable effectiveness of deep features as a perceptual metric". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 586–595.

A Prediction Performance

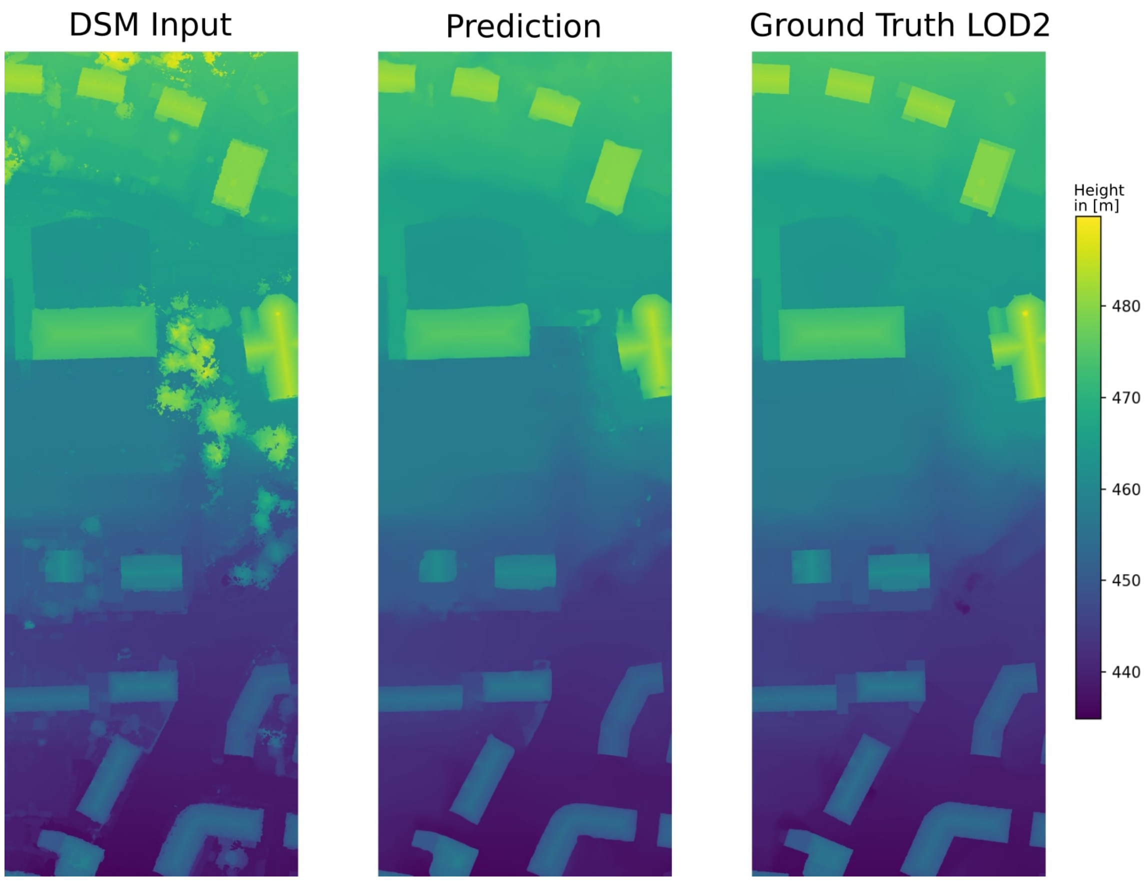


Figure 27: Prediction Performance on the test area 1/4

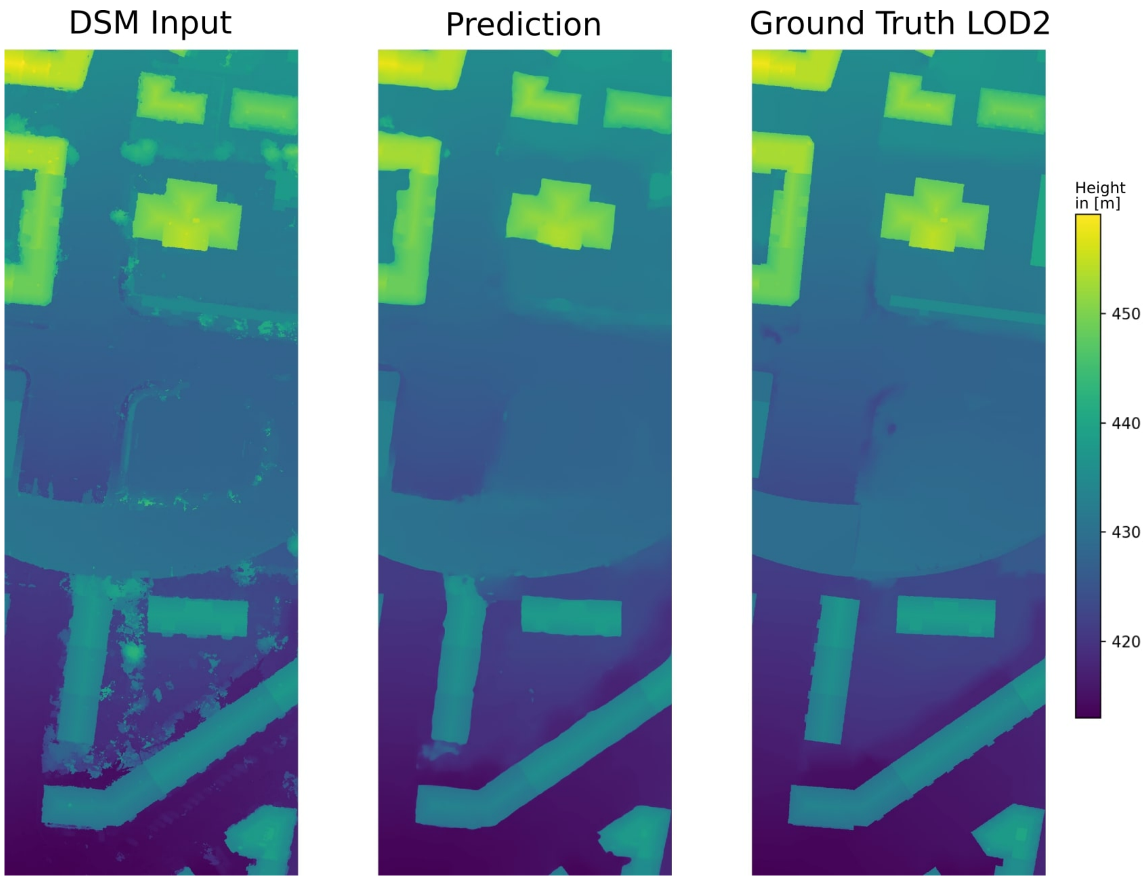


Figure 28: Prediction Performance on the test area 2/4

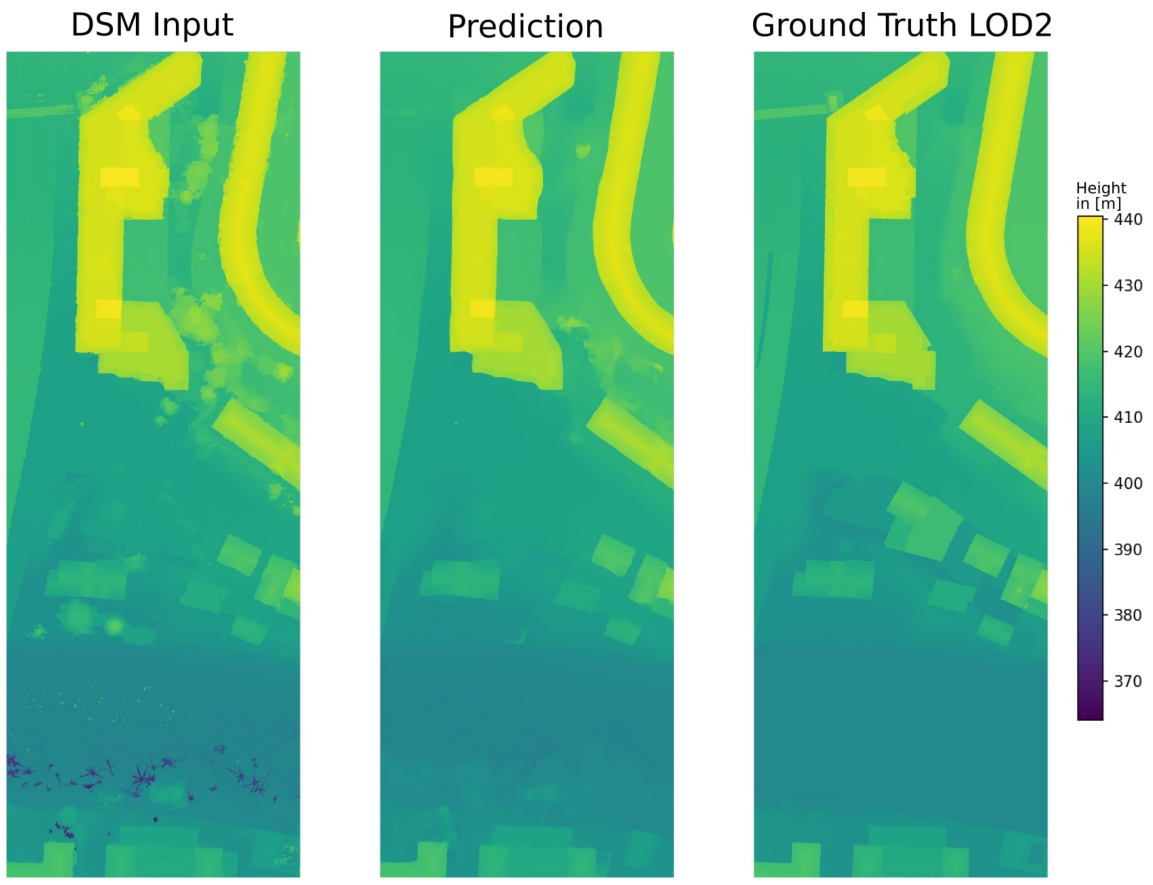


Figure 29: Prediction Performance on the test area 3/4

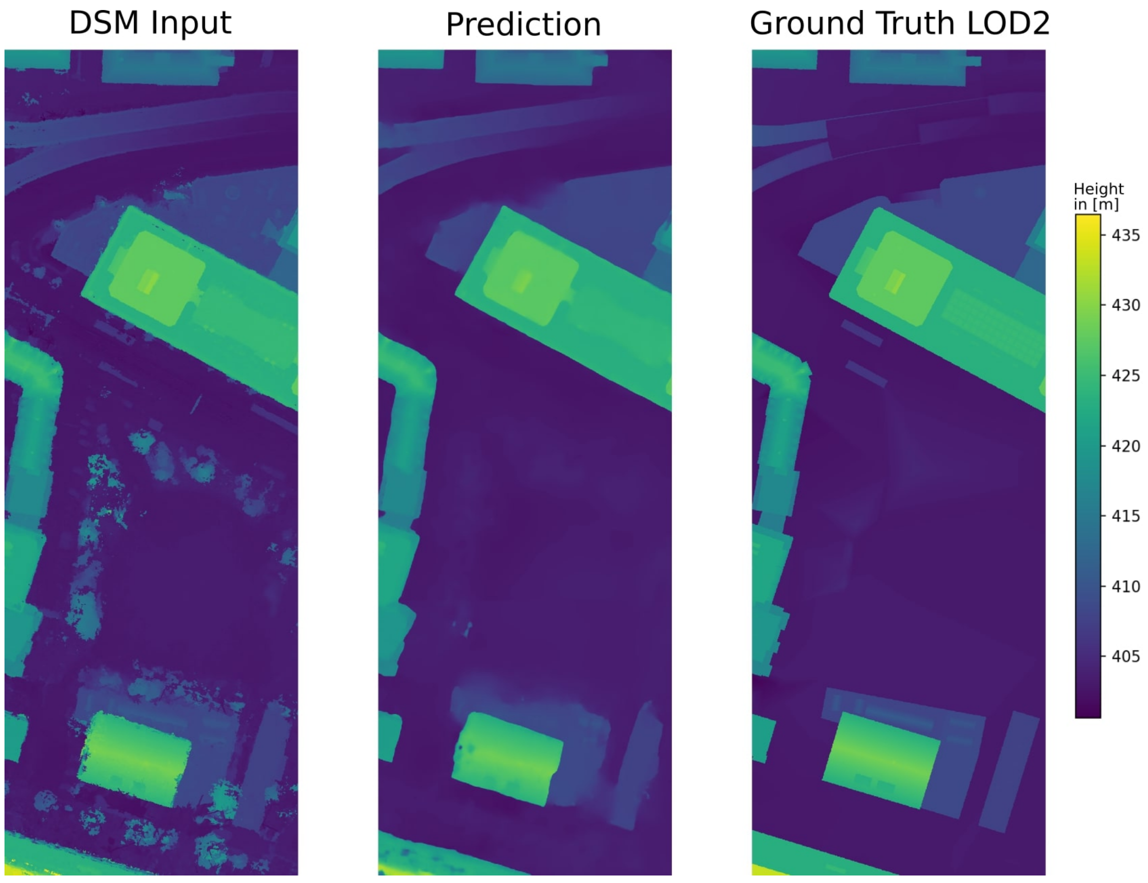


Figure 30: Prediction Performance on the test area 4/4

B Error Distribution

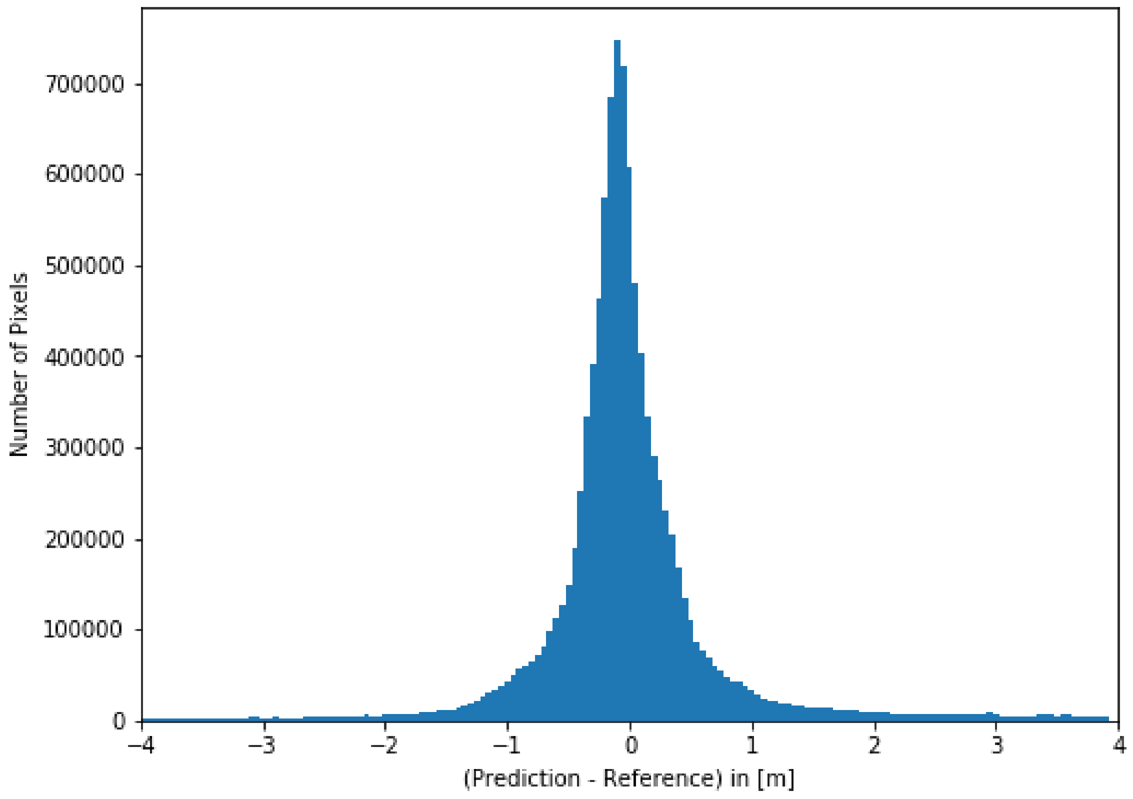
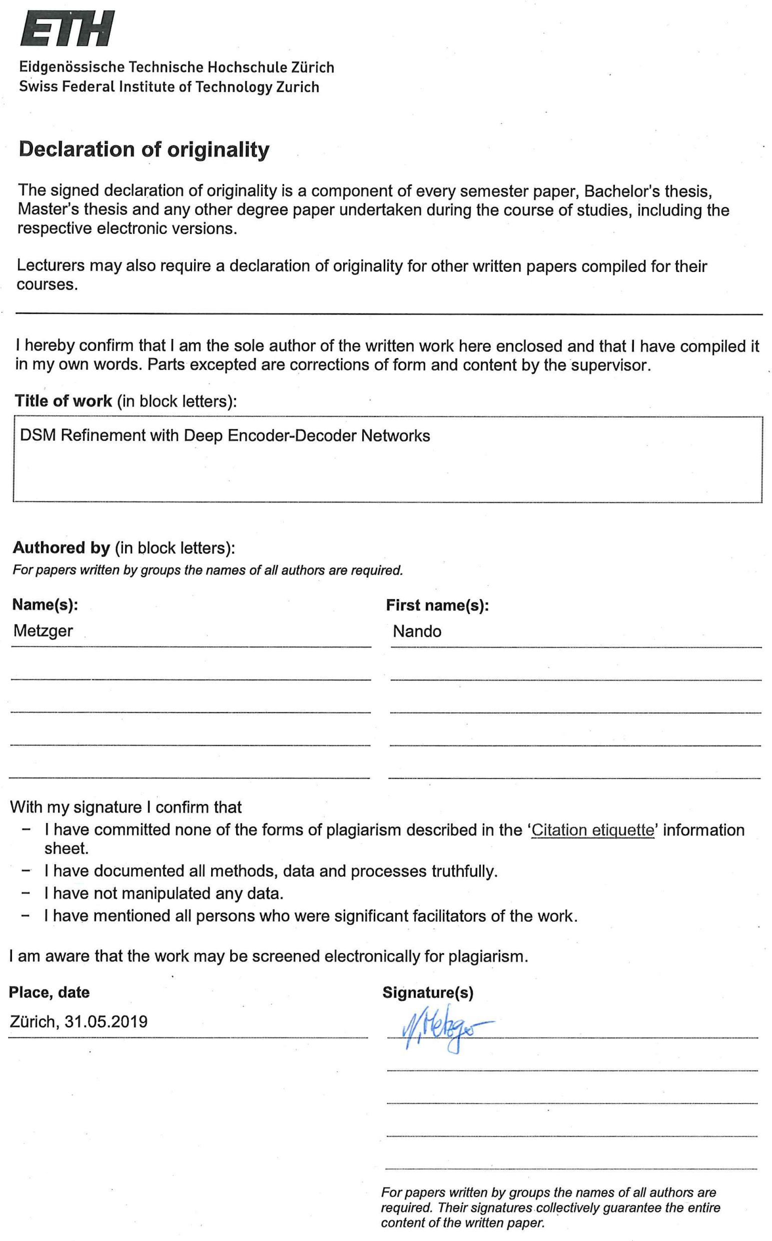


Figure 31: Histogram of the prediction errors in meter between the prediction on the test set and the reference data.

C Declaration of Originality



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

DSM Refinement with Deep Encoder-Decoder Networks

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Metzger	Nando

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 31.05.2019	[illegible]

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.