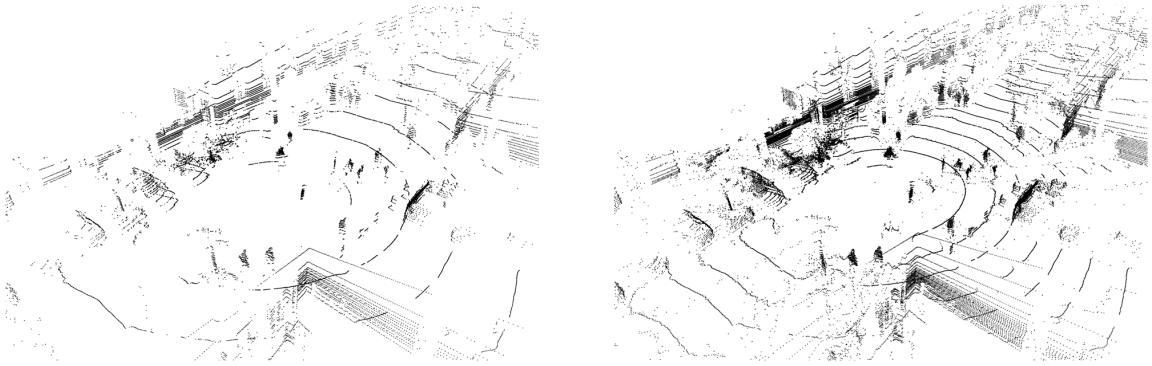
University of Stuttgart
Germany

Institute of Telecommunications
Professor Dr.-Ing. Stephan ten Brink

Master Thesis

Synthesizing realistic high-resolution LiDAR point clouds by upsampling with neural networks

Larissa T. Triess



Date of hand out:	April 02, 2018
Date of hand in:	October 02, 2018
Supervisor:	Prof. Dr.-Ing. Rainer Ott, INUE University of Stuttgart
	Dr. rer. nat. David Peter, Daimler AG

Kurzfassung

Selbstfahrende Autos erfordern ein detailliertes Verständnis ihrer Umgebung. Sensoren, wie Kameras, LiDARe, Radare und GPS stellen umfassende Information über die Umgebung des Fahrzeuges bereit. Der LiDAR Sensor ist der Schlüssel zur Erzeugung von präzisen 3D Umgebungsinformationen. Die generierten Informationen dienen als wichtige Eingangsdaten für eine Vielfalt von Problemen, wie zum Beispiel Objekterkennung. Mit steigender Anzahl an Ebenen im LiDAR Sensor hat sich auch die Performance von Algorithmen zur Umgebungserkennung verbessert. In dieser Arbeit werden mehrere Neuronale Netzwerk Architekturen zum Hochskalieren von Punktwolken erarbeitet und ausgewertet. Die Standart-Architektur erreicht hierbei eine mittlere quadratische Abweichung von 24,9 und einen mittleren absoluten Fehler von 0,79, dies entspricht einer Verbesserung von 72% bzw. 65% im Vergleich zu einem traditionellen Algorithmus mit bilinearer Interpolation.

Abstract

Self-driving cars require a detailed understanding of their environment. Sensors, like cameras, LiDARs, RADARs, and GPS provide extensive data about the vehicles surrounding, complementing each others weaknesses. LiDAR is the key sensor to obtain accurate 3D information and provides crucial data input to accomplish a variety of tasks with neural networks, for example object detection. The performance of perception algorithms increased with the amount of layers in the sensor, equaling the vertical resolution. In this work several system setups based on a deep residual network for LiDAR point cloud up-sampling have been introduced. The regular network architecture was able to decrease the mean squared error to 24.86 and the mean absolute error to 0.79, which corresponds to an improvement of 72% and 65% compared to an traditional approach with bilinear interpolation.

Title page image: Shown are 2D projections of two 3D rendered point clouds of the same scene. A point cloud is a 360 degree scan of the environment, captured with a LiDAR sensor. The left image features 16 vertical layers, the right point cloud is denser, as it is up-sampled to 32 layers. Both scenes show a road crossing with buildings at the side, parked cars and some trees and pedestrians. The ego vehicle is headed to the top right corner of the image.

Contents

1 Introduction

In the present-day world, everything has to be faster, better connected and more efficient. One step towards the future is the development of self-driving cars to explore other means of transportation. Those vehicles have to understand their surrounding in the best way possible to maneuver through the world. Environment perception is one of the essential components of self-driving cars. A variety of sensors are used to generate a detailed understanding of the 3D world. Cameras, LiDARs, radars, and GPS provide extensive data of the vehicles surrounding, complementing each others weaknesses. LiDAR is the key sensor to obtain accurate 3D information by mapping the world with rotating laser scanners in 360 degree view. LiDAR sensor data is a crucial input to accomplish a variety of tasks, such as object detection.

The data obtained by a LiDAR sensor has a very high horizontal resolution, for example 1800 points for a full 360 degree scan, which depends on the rotation speed and the frequency of the sensor. The vertical resolution is considerably lower, for example 32 or 64 and depends on the hardware properties of the sensor, meaning the amount of vertically stacked send and receive modules, i.e. layers. The development in the last years showed, that the amount of layers within the sensors is increasing, however the hardware development of these sensors is quite time intensive. Having such higher resolving LiDAR data, i.e. point clouds, already available prior to the actual sensor can accelerate or even enable the development of downstream perception algorithms. Therefore, this work focuses on up-sampling the amount of vertical layers in LiDAR point clouds in a realistic manner.

Up-sampling of camera images is a well known practice and there exists a variety of traditional and learning based methods for it. Point cloud up-sampling is more difficult in terms of having a sparse data structure. In order to use convolutional neural network with 2D convolutions, it is a common practice to transfer the 3D point cloud into a 2D data representation, a so-called distance image with size 32x1800. Those images serve as the ground truth data during the training process. An input with the size 16x1800 is up-sampled to a size of 32x1800 with the help of a deep residual network. It is a fully convolutional network, therefore it is possible to insert larger data representations later, thus achieve an up-sampling from 64 layers to 128 for example.

Several different network structures are introduced in this work, which all showed major improvement over traditional up-sampling methods in terms of the error value between the prediction and the target. The most commonly used loss function for super-resolution tasks is the L2-Loss. However, experiments with the same network architecture have shown that the network trained with an L1-loss instead of L2-loss, generates less noisy and more smooth

high-resolution point cloud predictions. The network decreased the mean squared error from 87.49 to 24.86 and the mean absolute error from 2.25 to 0.79 compared to the traditional bilinear interpolation. Furthermore, it achieved a better performance when evaluated on semantic segmentation and a higher mean opinion score in the point cloud survey.

This work is structured into four major parts. Chapter 3 explains how LiDAR sensors work and which up-sampling methods exist. Furthermore it gives an introduction to machine learning and convolutional neural networks with a focus on the loss functions used for training. In the subsequent chapter, important building blocks of this work are explained, such as the used dataset and the evaluation metrics for performance assessment. It also introduces the evaluation results of the traditional approaches, which serve as a baseline for this work. Chapter 5 presents the system setups and architectures developed and implemented in this work. The first part gives a general overview on the pipeline and the two core networks. Subsequent sections explain the detailed experiment setups. In the last major chapter, the experimental results are presented and two additional evaluation series are illustrated, a semantic segmentation assessment and a mean opinion score testing on point clouds among humans. Conclusion and outlook can be found in chapter 7.

2 Related Work

The main objective of this work is to enable or accelerate the development of downstream perception algorithms [1], such as object detection, by providing realistic high-resolution point clouds without an available LiDAR sensor with many layers, i.e. high vertical resolution. This work focuses on single-image super-resolution, which means that no additional information of previous or subsequent frames is used to restore high-resolution. In the context of point clouds, an image corresponds to a full 360 degree scan of the LiDAR, projected into a 2D cylindrical depth image. In contrast to other work, only the LiDAR information is used here, no additional camera images.

The aim of super-resolution is to estimate the high-resolution visual output of a corresponding low-resolution input, e.g. a single image. The goals range from providing better content visualization for traditional image processing applications, up to achieving better visual recognition, including computer vision tasks. The high-resolution output can be obtained by either proving sensors with large spatial resolution, at the cost of very high prices for the device or by using software related tools. The advantage of post-processing the captured visual data is that it allows to balance computational and hardware costs.

Super-resolution features two main techniques: image interpolation and image restoration. Interpolation changes the dimension of an image while restoration recovers a degraded input. Thus, image super-resolution is a technique that restores the degraded image and also increases its size. Based on the application, super-resolution approaches can be grouped into three major categories: Interpolation-based, reconstruction-based, and data driven.

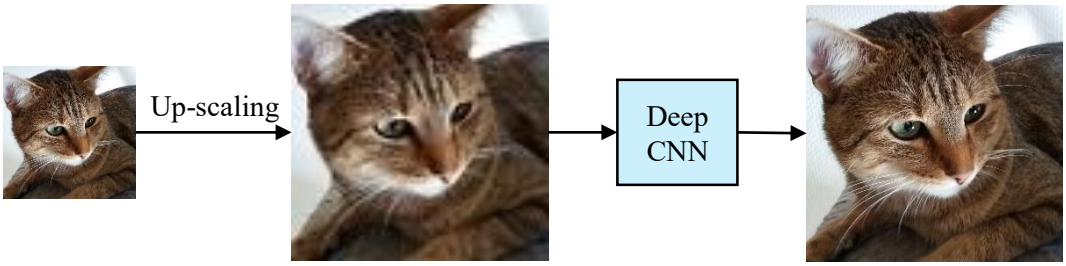
Interpolation based techniques are the simplest way to generate high-resolution images and are based on image re-sampling, i.e. a mathematical technique used to create a new version of the image with a different width and/or height. The technique has a low computational complexity and can therefore also be used for real-time applications. However, it is not possible to obtain the high-frequency information, i.e. fine details in the resized image, due to the low-pass behavior of the interpolation filters, e.g. bilinear, bicubic.

Reconstruction based super-resolution approaches [1, 2, 3] enforce a reconstruction constraint and impose a prior knowledge on the high-resolution image. The constraint requires that the high-resolution image should be as close as possible to the low-resolution image via smoothing and down-sampling. One of the limitations of reconstruction based super-resolution techniques, is that the imposed prior is not valid for arbitrary images and artifacts appear in the high-resolution image [4].

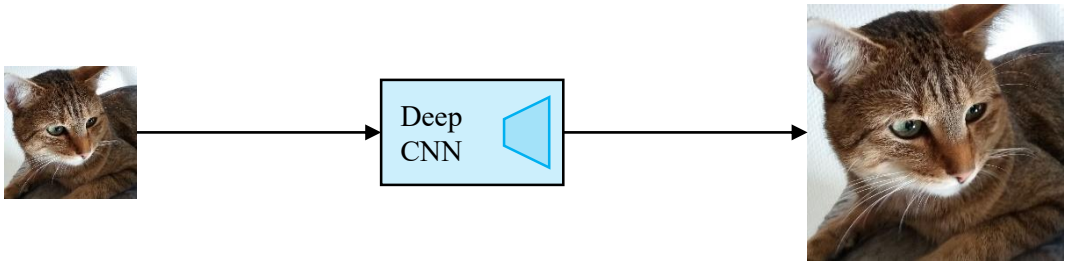
[1] subsequent algorithms using the output of the previous computation unit

Data driven methods are based on machine learning. Many supervised learning approaches use convolutional neural networks (CNNs) which use the power of back-propagation in order to learn hierarchical representations and step by step reduce the error between the prediction and the target [5]. A CNN directly learns an end-to-end mapping between the low and high-resolution images, little or no pre- and post-processing are needed besides optimization. Most work focuses on minimizing the mean squared reconstruction error.

Figure 2.1 shows two variants to restore high-resolution. The upper one is used by most super-resolution methods. It first upscales the input image to the desired size using bicubic interpolation in a pre-processing step and afterwards restores fine textures. SRCNN [6] is the first deep learning based method to outperform traditional algorithms and consists of a network with only 3 convolutional layers: patch extraction and representation, nonlinear mapping, and reconstruction. Patches from the interpolated image are extracted and represented as high-dimensional vectors which comprise a set of feature maps. Each high-dimensional vector is non-linearly mapped to another high-dimensional vector, representing a high-resolution patch, which comprises another set of feature maps. The final high-resolution image is obtained by overlapping the high-resolution patches (reconstruction).



(a) Bicubic up-scaling is applied to resize the image to desired output size, afterwards a deep neural network is applied to restore fine details.



(b) No pre-processing step is needed. The up-scaling happens within the network.

Figure 2.1: Data driven image up-scaling with and without pre-processing of the data.

However, it is also possible to not use any up-scaling as a pre-processing step (fig: 2.1b). Here, the network itself contains a fractionally strided convolution which achieves the desired size of the output image. As most methods rely on an optimization with a simple single pixel level error, the high-resolution images might not appear visually realistic for humans [7]. Therefore, some techniques use a perceptual loss instead, which rather optimizes on the feature maps extracted by a subsequent feature extractor in order to capture high-level information within the image [8].

Nevertheless, resulting high-resolution images are often lacking high-frequency details and are perceptually unsatisfying in the sense that they fail to match the fidelity expected at the higher resolution. Therefore, Ledig et al. proposed SRGAN [9], a generative adversarial network (GAN) for image super-resolution. Their method uses a perceptual loss function [2] which consists of an adversarial loss and a content loss. The adversarial loss pushes the output to the natural image manifold using a discriminator network. This network is trained to differentiate between super-resolved images and the origin photo-realistic images. Additionally, a content loss was used which enforces perceptual similarity instead of similarity in the pixel space.

All the above mentioned methods are applied to camera images. Analyzing 3D point clouds, i.e. a 360 degree scan of the environment, with deep convolutional networks is challenging due to the sparsity and irregularity of the data. Yu et al. recently proposed PU-Net [10], a point cloud up-sampling network which learns multilevel features per point and expands the point set via a multi-branch convolution unit. The expanded feature is then split to a multitude of features, which are then reconstructed to an up-sampled point set. Similar to [6], the network operates at a patch level with a join loss function that encourages the up-sampled points to remain on the underlying surface with a uniform distribution.

This work does not use unordered point clouds, therefore it is possible to transform the data into a 2D space, enabling the usage of classical 2D convolutions. Thus, it is possible to make use of the same or similar techniques as used for image super-resolution. However, the biggest struggle here, which is also the reason why point cloud up-sampling is more complex than image up-sampling, is how to deal with the sparsity of the data. In an image all pixels have a valid value assigned to it, this does not account for the 2D structure of the point cloud. A point cloud is created by sending out laser beams in 360 degrees, their reflection gives information about the distance of an object. Sometimes there is no reflection received, for example when the beam is faced towards the sky. This causes the sparsity in the data. When transforming to a 2D space, dummy values for those missing points have to be inserted. These points need special treatment now, which is one of the focuses in this work.

[2] a function that measures high-level perceptual and semantic differences between images

3 Basics

This chapter gives an introduction to the fundamental knowledge and methods applied in this work. Section 3.1 explains the operation principle of light detection and ranging (LiDAR) sensors and how it generates the desired data. The next section gives an introduction to traditional and learning based up-sampling methods for both image and point cloud data. In section 3.3 an overview on neural networks (NNs) is presented followed by section 3.4 which gives an extended and more detailed explanation on CNNs. The last section introduces several different loss functions suitable for NNs trained for super-resolution tasks.

3.1 LiDAR Sensors

LiDAR is a similar measurement procedure to radio detection and ranging (RADAR). Both methods provide information about the distance, position and reflectance intensity of an object. In contrast to RADAR, LiDAR does not use radio waves, but infrared laser beams and has a higher range and resolution.

Figures 3.1 and 3.2 show the schema of how a LiDAR sensor works and the visualization of the measured data points, respectively. LiDAR sensor measures the time of flight (ToF) of their emitted laser pulses to determine the distance of surrounding objects with high accuracy, resulting in a point cloud like in figure 3.2. For every point also a reflectivity estimate exists.

In this work Velodyne VLP-32 LiDAR scanners are used. It features 32 vertically stacked send and receive modules which rotate around a common vertical axis. While rotating, each module periodically measures the distance and reflectivity to possible surrounding objects, resulting in a 360° scan, called point cloud.

Figure 3.3 shows that the angles of the layers are not equidistantly distributed, but are denser in the middle than at the edges of the vertical field of view (FOV).

Each measurement point of a LiDAR scan belongs to a specific location within the sensors coordinate system, i.e. the respective azimuth and elevation angles. With this knowledge cylindrical depth and reflectivity images can be generated for every 360° scan. At a speed of ten revolutions per second, images of size 32x1800 pixels are generated. This projection allows the usage of 2D convolutional layers, as used in state-of-the-art image-based CNN architectures and reduces inference time compared to the use of full 3D input representations. It is possible to transform the image back to a full 3D point cloud representation without any information loss.

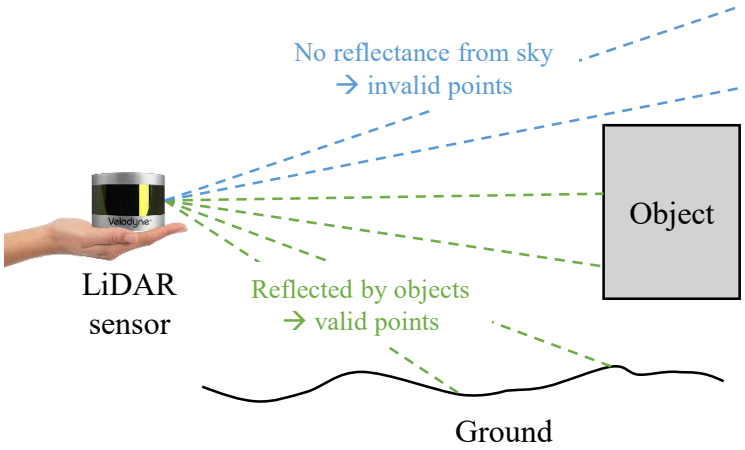


Figure 3.1: Sketch of a LiDAR sensor emitting light beams. The ones reflected by obstacles (green) create the output 3D point cloud. When transforming the data to a 2D structure, the positions where beams did not receive a reflection (blue), are marked as invalid, i.e. if the beam points towards the sky or the window of a car. [11]

3.1.1 Distance Images

Figure 3.4 shows an example of such 2D representation of a point cloud (same scene as fig. 3.2). In the following this structure is referred to as a *distance image*. The brighter the color, the closer the object. The distance image is two dimensional with the height H and the width W. The height corresponds to the amount of layers in the sensor, the width arises from the speed of revolution and always comprises a full 360 degree scan. In the following, the distance of a single point is denoted as $d_{\mathrm{h,w}}$, with $h \in \mathbb{N} : 0 \leq h < H$ counting over the rows of the image and $w \in \mathbb{N} : 0 \leq w < W$ over the columns.

3.1.2 Invalid Points

When the sensor scans the environment, there are cases where no laser reflection is received, e.g. when pointed towards the sky (dotted blue lines in fig. 3.1). This causes a point cloud to be a sparse data representation, as points are missing for some azimuth and elevation angles. When transforming the point cloud to the 2D structure, dummy distance values are assigned to all missing points. Additionally a second channel is saved, which is called 'point valid mask' in the following (fig. 3.4). This mask holds binary values, which states whether a corresponding pixels in the distance image is 'valid' (1) or 'invalid' (0). Invalid points are the ones where no reflection was received and are marked black in figure 3.4. These points need special treatment, thus makes this task more complicated than camera image up-sampling.

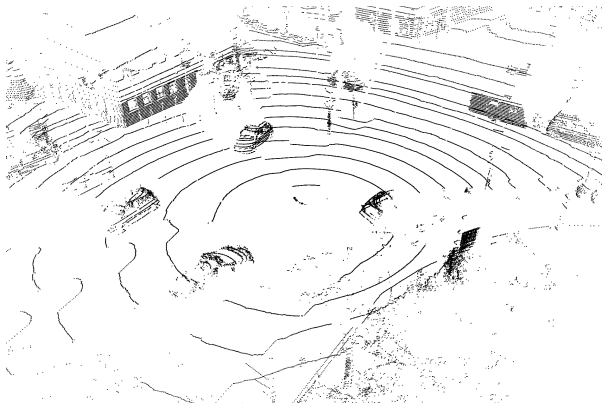


Figure 3.2: Example 3D visualization of a scene captured with a LiDAR sensor. The ego vehicle is located in the center of the image and headed to the top right corner of the image. Shown is a road scene at an intersection with some vehicles, traffic signs and buildings.

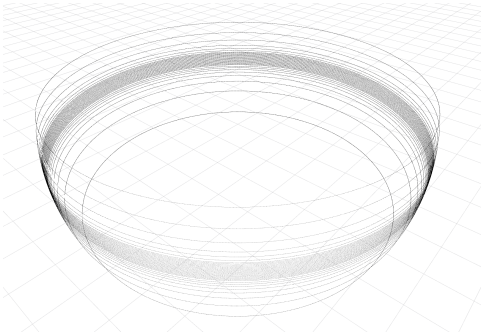


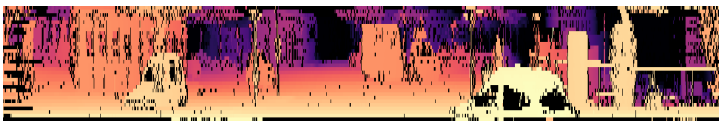
Figure 3.3: Visualization of the layer distribution within the laser scanner (Velodyne VLP-32).

3.2 Up-Sampling Methods

This section explains common techniques for image and point cloud up-sampling. As described in section 3.1, a 2D representation of the 3D point cloud is used in this work, referred to as 'LiDAR image' or 'distance image' in the following. Due to the common structure to camera images, it suggests itself, that it is possible to make use of classical image up-scaling techniques. Therefore, this section is subdivided into up-scaling methods for camera images (section 3.2.1) and for point clouds (section 3.2.2).

3.2.1 Image Up-Sampling

In computer graphics, image scaling refers to the resizing of a digital image, where a new image with a higher or lower number of pixels has to be generated. The process can be interpreted as a form of image re-sampling for which there exists a number of common interpolation methods. An extract of the most important ones are introduced below.



(a) LiDAR distance image: the brigther the color, the closer the object. Black denotes positions where no reflection was received by the sensor, i.e. invalid points.

(b) Invalid point mask: white are valid points, black are positions where no reflection was received by the sensor.

Figure 3.4: Example of a 2D LiDAR depth image (top) and its corresponding invalid point mask.

Bilinear interpolation

Bilinear interpolation (fig. 3.5b) is one of the basic re-sampling techniques in image processing and computer vision. It is an extension of the linear interpolation (fig. 3.5a) for interpolating functions of two variables on a 2D grid. Basically, a linear interpolation is first performed in one direction, and then again in the other.

(a) Linear interpolation (b) Bilinear interpolation

Figure 3.5: Comparison of linear and bilinear interpolation. The black dots corresponds to the interpolated points, the green and yellow ones to the neighboring samples. The height above ground displays their values. (source: [12])

Nearest-neighbor interpolation

Nearest-neighbor interpolation (fig. 3.6) is a simple method of multivariate interpolation in one or more dimensions. It is also known as proximal interpolation. For a non-given point, the algorithm selects the value of the nearest point and does not consider the values of any other neighboring points at all. It is very simple to implement and commonly used in real-time 3D rendering to select color values for a textured surface.

(a) 1D nearest neighbor interpolation

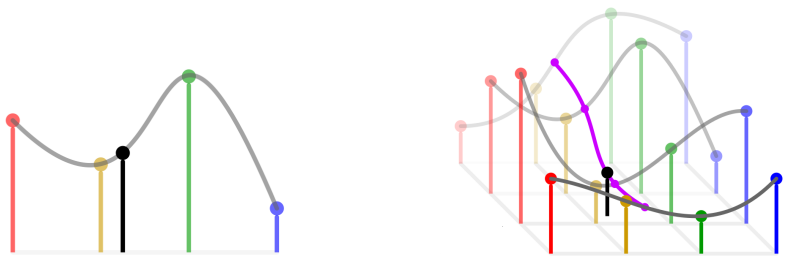
(b) 2D nearest neighbor interpolation

Figure 3.6: Comparison of 1D and 2D nearest neighbor interpolation. The black dots corresponds to the interpolated points, the green and yellow ones to the neighboring samples. The height above ground displays their values. (source: [12])

If the interpolated point lies exactly in the middle of neighboring points, then the value of the point with the smallest index in the grid is used. For example, one point with the value 4 is located at index 2 and another one with value 8 at location 6. If now a point shall be interpolated at position 4, the value of the first point is assigned, because the index of 2 is smaller than 6, resulting in an interpolated point with value 4.

Bicubic interpolation

Bicubic interpolation (fig. 3.7b) preserves fine details better than the bilinear algorithm. Similar to bilinear interpolation, it is an extension of the cubic (fig. 3.7a) interpolation. Bicubic interpolation can be accomplished using either Lagrange polynomials, cubic splines, or a cubic convolution algorithm. The interpolated surface is smoother than corresponding surfaces obtained by bilinear or nearest-neighbor interpolation. Thus is often chosen over those methods in image re-sampling, when speed is not an issue. In contrast to bilinear interpolation, which only takes into account 4 pixels (2x2), bicubic interpolation considers 16 pixels (4x4).



(a) Cubic interpolation

(b) Bicubic interpolation

Figure 3.7: Comparison of cubic and bicubic interpolation. The black dots correspond to the interpolated points, the red, yellow, green and blue ones to the neighboring samples. The height above ground displays their values. (source: [12])

Learning based approaches

Many learning-based approaches for image super-resolution make use of CNNs (explained in section 3.4). Their aim is to learn a mapping between the low-resolution and high-resolution images in the training set. One of the first methods for single image super-resolution describes a three-layer CNN, called SRCNN [6], which is often used as a reference. It is divided into four steps: first, preprocessing which up-scales the low-resolution image to the desired high-resolution size, second, feature extraction, followed by non-linear mapping, and finally the reconstruction step, which produces the high-resolution images from high-resolution patches mapped in step three. The network is trained with a mean squared error (MSE) loss.

Most methods follow the rule of first up-scaling the image to the desired size (preprocessing) and then train a network to de-blurr the image in order to restore high resolution. This means that the way of up-scaling to the higher image size is always fixed and cannot be learned. However, it is beneficial to also include this preprocessing step into the network and let the training find the optimal choice of up-scaling. This can be achieved with a transposed convolution layer for instance (explained in section 3.4.2).

Since 2014, a special version of NNs has emerged, called generative adversarial networks (GANs) [13]. These networks are able to generate very realistic data for human perception. In image up-sampling, the perceptual image quality of super-resolved images is heavily dependend on the choice of loss function which is minimized during training. As mentioned before, recent work is largely based on MSE optimization which improves commonly used image quality metrics. However, those reconstruction metrics, like peak signal-to-noise-ratio (PSNR) may not capture fine details in the image. The adversarial loss used in GANs, on the other hand, pushes the reconstructed image to the natural image manifold using a discriminator model. Quality testing with human opinion showed that this approach generates visually more pleasant images, meaning they are more realistic even though their mathematical difference to the target is larger.

3.2.2 Point Cloud Up-Sampling

In contrast to camera images, point clouds are sparse data structures and usually computationally more complex when processed in the three dimensional space. Therefore, it is a common practice to transfer the point clouds into a 2D space, where potentially similar algorithms like on camera images can be applied. The before mentioned dummy values for invalid points in the 2D structure causes these algorithms, like the above mentioned traditional interpolation methods, to not produce satisfying up-sampling results on point clouds. Additionally, the literature on 3D point cloud processing falls far behind similar tasks on 2D data. Recently, Yu et al. [10] proposed PU-Net, a data-driven up-sampling technique. Their approach encourages points to remain on the underlying surface with a uniform distribution.

The main difference to the task in this work, is that the point cloud is unordered and random points shall be interpolated to generate high-resolution point clouds. The data in this work,

however, possesses additional information encoded in the pixel location of the LiDAR images, i.e. elevation and azimuth angle. Furthermore, the aim here is not to interpolate random points, but rather entire layers of the sensor.

3.3 Training Neural Networks

Artificial neural networks (ANNs) are inspired by biological neural networks in animals and are used in machine learning. They aim to understand the underlying structure of data and fit it into models that can be used and understood by humans. ANNs are able to solve a variety of problems in pattern recognition, prediction, and optimization. This section gives a general overview about NNs, mainly inspired by [14] and [15]

3.3.1 General Structure

The basic computation unit of an ANN is a neuron (figure 3.8). Similar to its biological model, each neuron receives input signals x_i from other neurons, processes them and produces an output signal y. Each input signal is multiplied with a weight w_i. The basic idea here, is that the weights are learnable and control the strength of influence of one neuron on another. The multiplied signals are then all summed up (in many cases a bias b is added) and in many cases passed through a non-linearity, called activation function f. Common activation functions are for example the sigmoid function, which takes a real-valued input and squashed it to a range between 0 and 1, or the rectifier linear unit (ReLU) activation function, which simply thresholds at zero.

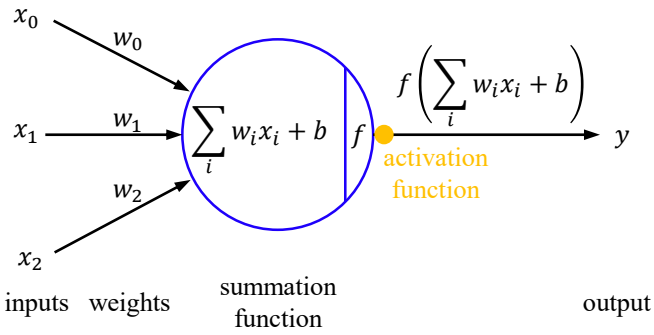


Figure 3.8: Mathematical model of an artificial neuron. (source: [16])

In general, the neurons of a NN are interconnected via directed edges, forming a weighted directed graph. Depending on the arrangement, they can be grouped into feed-forward neural networks (no loops, memoryless) and recurrent neural networks (loops due to feedback connections). Neural network models are often organized into distinct layers of neurons. The most

common layer type is the fully-connected layer in which neurons between two adjacent layers are fully pairwise connected, but neurons within a single layer share no connections. Another type of layer is the convolutional layer used within CNNs (explained in section 3.4.1).

Figure 3.9 shows an example of a 3-layer feed-forward NN with fully-connected layers. The input layer is not counted, when speaking of a N-layer NN. The example possesses three inputs (orange), two hidden layers of four neurons each (blue) and two outputs (green).

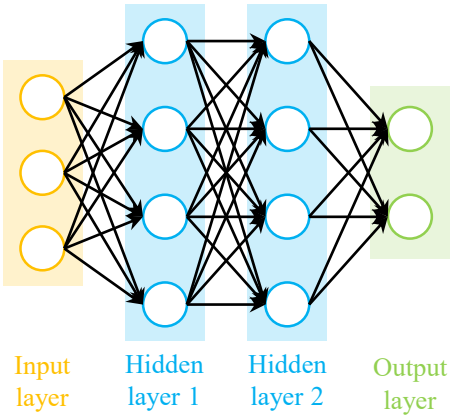


Figure 3.9: A 3-layer neural network with three inputs (orange), two hidden layers of four neurons each (blue) and two outputs (green). There are connections between neurons across layers, but not within a layer

3.3.2 Training Process

Just like their biological models, ANNs have to learn how to accomplish certain tasks. They cannot be programmed to work in a predefined way like traditional algorithms. The ability to learn is the major advantage of NNs over traditional systems. Learning describes the problem of updating the connection weights, that are learned from training samples, in a way that the network can efficiently perform a specific task. Iterative updates of the weights improve performance over time.

Figure 3.10 shows the three major types of learning. For a human, supervised learning is the most intuitive way of learning, i.e. learning with a teacher. The network is provided with a correct answer for each output and determines the weights as close as possible to it. This method can only be used if a large enough set of data with known results, i.e. labels exists. In reinforcement learning, a program performs a particular task for which it is provided with a feedback in terms of rewards or punishment, thus continuously trains itself using trial and error method. Unsupervised learning, on the other hand, does not receive any feedback, nor require the correct answer. It organizes patterns into categories derived from correlations of the underlying structure of the data.

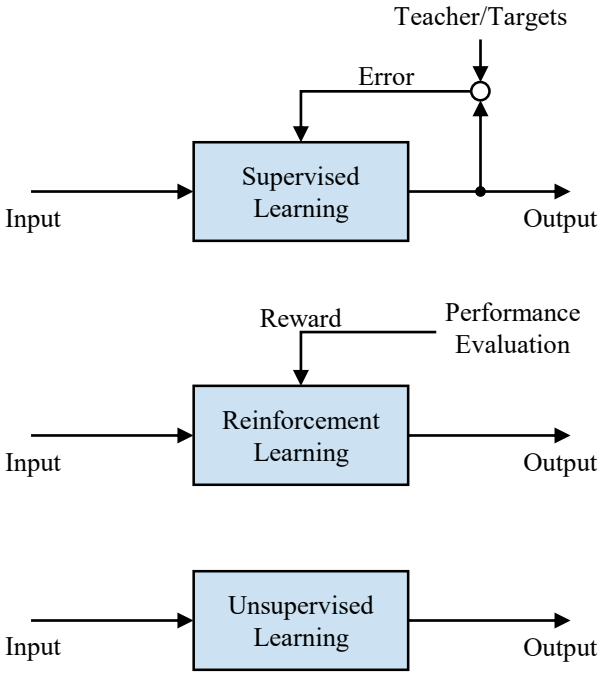


Figure 3.10: The three major learning techniques for neural networks: supervised, reinforcement, and unsupervised learning

3.3.3 Loss function

This work is based on a supervised training process. The network is given a desired output d^{gt} (a high-resolution point cloud), for each input sample (a low-resolution point cloud). During the learning process, it generates an output d^{pred}, which is usually not equal to the desired output. A cost function, also called loss, compares the two outputs and calculates the error signal $\mathscr{L}\left(d^{\text{gt}}, d^{\text{pred}}\right)$. It is then used to update the weights, which results in gradually reducing this error. The process of updating the weights is called backpropagation and was first introduced by LeCun et al. [5].

3.3.4 Optimizer

In NNs the aim of an optimizer is to adjust the parameters of the network in order to minimize the loss function. The most common approach is the gradient descent technique. After retrieving the gradients of a model through backpropagation [5], the weights are updated in the opposite direction of the loss functions gradient to move closer to local minima. An additional parameter, called learning rate, is added to better regulate the amount of adjustment during the training process. Generally, its value is reduced over time to reduce the risk of obtaining a sub-optimal local minimum instead of the desired global one. The higher the parameter update, i.e. the learning rate, the higher the ”jumps” in the loss function. This is useful at

the beginning of a training in order to step over local minima, but later this might lead to a ”ping-pong”-like effect with no convergence if the update rate is too high.

There exist many different optimizer types, which evolved over time. In this work, the Adam optimizer is used for all experiments. Adam stands for adaptive moment estimation and was developed by D. Kingma and J. Ba in 2014 [17].

3.4 Convolutional Neural Networks (CNNs)

For many image processing tasks CNNs are used nowadays. In general, a CNN receives an input vector and multiplies it with a matrix which produces an output. Usually a bias vector is added to the output before passing it through a non-linearity (see section 3.3.1). Many types of data can be used as input, but they all have to possess certain properties. First, they must be represented as multi-dimensional arrays. Second, they feature one or more axes for which ordering matters (e.g. height and width axes for an image). And at last, one axis, called the channel axis, is used to access different views of the data (e.g. RGB channels of a color image). For some tasks, like computer vision or speech recognition, it is advantageous to preserve the implicit structure of the data. A discrete convolution is a linear transformation that preserves this notion of ordering.

Convolutional layers mimic the way a cat’s visual system works by using overlapping areas of sensor signals as input values. In the first layers, it detects local visual features like edges or corners, which are then combined in subsequent layers. The ability to take into account local distortions of the input makes them invaluable for spatial input, like images. Furthermore, they only require a fraction of the weights used by a standard fully-connected layer by using a method called weight sharing. It is assumed, that if a local feature detector was computed on one part of the image, it is most likely useful on the entire image.

Figure 3.11 shows that the neurons (circles) of a convolutional layer are arranged in three dimensions (height, width and depth) which are similar to the layout of an image with dimensions x, y and color channels. In each layer, a 3D input volume of neurons is transformed through a differentiable function to form a new 3D volume. Furthermore, each layer also possesses a set of trainable filters which are arranged in three dimensions and contain the individual weights. However, their width and height are relatively small, while the depth is equal to the corresponding depth of the input volume, i.e. the number of feature maps. Common filter sizes range from 1x1xd to 7x7xd, d being the depth.

A convolutional layer is defined by some additional parameters compared to the mathematical definition of a convolution. In this work, the vocabulary from `TensorFlow` is used. A convolutional layers output shape is determined by its input shape, as well as the kernel shape and the choice of padding and strides. Zero padding denotes the insertion of zeros and a stride refers to the step size if the kernel when moved over the input. It is not trivial to infer the relationship between these properties in contrast to fully-connected layers, where the output size is independent of the input size. The following sections explain the mathematics of such

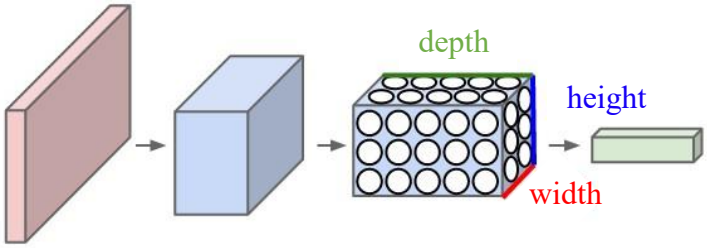


Figure 3.11: Visualization of a CNN with one input layer (red), two hidden layers (blue) and an output layer (green). For instance, the input layer holds an RGB image, so its width and height equals the dimensions of the image, and the depth corresponds to the RGB channels and therefore equals 3. The second hidden layer shows that a CNN arranges its neurons in three dimensions, according to width, height, and depth. (source: [16])

a convolution layer and how the transposed convolution works. More details are given by V. Dumoulin and F. Visin in [18].

3.4.1 Convolution Layer

Figure 3.13a shows an example 2D convolution. When convolving a 3x3 sized kernel (gray shaded) over an 5x5 input (blue) with no padding and stride of 2 (in both dimensions), the output size will result to 2x2 (green). The stride defines the step size with which the kernel is moved (slided) over the input. If the stride was 1, the output size results to 3x3. Mathematically, a convolution over the input d^{in} with the kernel w is defined as

$$d^{\text{out}} = d^{\text{in}} * w \tag{3.1}$$

with d^{out} being the output. In the language of machine learning, additional parameters of padding and stride are added.

Figure 3.12 shows a convolution operation with the same shapes as in fig. 3.13a, including the input (blue) and kernel (gray) values. The value of the output (green) at position (0, 0) calculates with

$$d^{out}_{0,0} = \sum_{h=0}^{2} \sum_{w=0}^{2} d^{\text{in}}_{\text{h,w}} \cdot w_{\text{h,w}} \tag{3.2}$$

with $w_{h,w}$ being the kernel values, also called weights. The other three values of the output are calculated likewise.

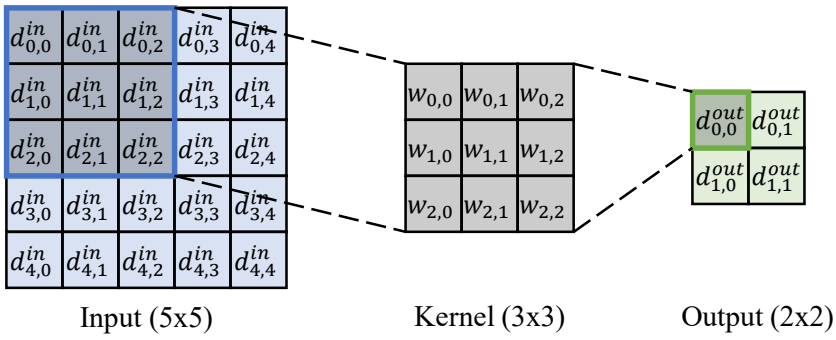


Figure 3.12: Example of 2D convolution calculation

3.4.2 Transposed Convolution Layer

Transposed convolutions are used for transformations going in the opposite direction of a regular convolution. They work by swapping the forward and backward passes of a convolution. For instance, although the kernel w defines a convolution whose forward and backward passes are computed by multiplying with C and C^T respectively (C being the convolution matrix), it also defines a transposed convolution whose forward and backward passes are computed by multiplying with C^T and $(C^T)^T = C$ respectively. Therefore, it is always possible to emulate a transposed convolution with a direct convolution, but it usually involves adding many zeros to the input, resulting in a much less efficient implementation.

Figure 3.13b shows an example for a transposed 2D convolution. Given the output and kernel size from fig. 3.13a, a transposed convolution can restore the original input size from fig. 3.13a with first padding (dashed lines) the input (blue). However, it can only ensure producing the original size, but not the sames values [1]. A transposed convolution is also called fractionally strided convolution and is a popular tool for up-scaling tasks.

In the following, a transposed convolution over the input d^{in} with the kernel w will be written as

$$d^{\text{out}} = d^{\text{in}} *^T w \tag{3.3}$$

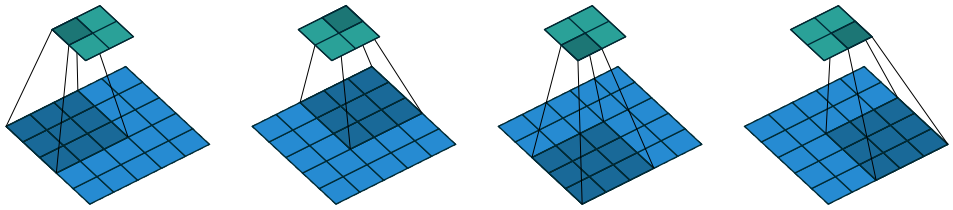
with d^{out} being the output and $*^T$ denoting the transposed convolution operation.

3.4.3 Emulation of Up-Scaling with Convolution Layers

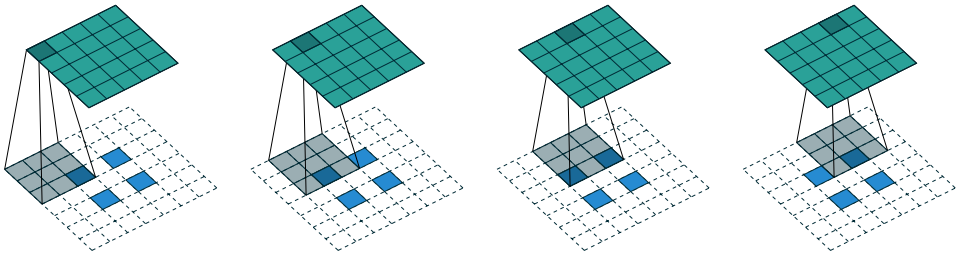
Depending on the desired up-sampling factor f, the kernel size w_{size} of the transposed convolution is

$$w_{size} = 2 \cdot f - f \bmod 2 \tag{3.4}$$

[1] The operation of a transposed convolution is not a deconvolution, as often called in literature. A deconvolution is the exact reverse of a convolution, restoring both shape and values.



(a) Convolving a 3x3 kernel over a 5x5 input (blue) with no padding using a stride of 2 results in a 2x2 output (green).



(b) The transpose of convolving a 3x3 kernel over a 5x5 input using strides of 2 is equivalent to convolving a 3x3 kernel (gray shaded) over a 2x2 input (blue) padded (dashed lines) with a 2x2 border of zeros and one zero inserted between inputs using unit strides. The resulting output (green) equals the input size of 3.13a, but holds other values.

Figure 3.13: Depiction of a transposed convolution as a convolution with zero padding (source: [18])

and the stride s is

$$s = f \tag{3.5}$$

for up-scaling in one dimension.

In this work, we want to up-sample the layers of the point cloud. Transferred to the distance image, this means up-sampling the height dimension, but leaving the width as it is. This means an up-scaling factor of $f_{\text{height}} = 2$ and $f_{\text{width}} = 1$ is required. Therefore, the kernel size and strides are

$$\begin{aligned} w_{size} &= (w_{\text{height}}, w_{\text{width}}) \\ &= (2 \cdot f_{\text{height}} - f_{\text{height}} \bmod 2, \quad 2 \cdot f_{\text{width}} - f_{\text{width}} \bmod 2) \\ &= (2 \cdot 2 - 2 \bmod 2, \quad 2 \cdot 1 - 1 \bmod 2) \\ &= (4, 1). \end{aligned} \tag{3.6}$$

and

$$s = (s_{\text{height}}, s_{\text{width}}) = (f_{\text{height}}, f_{\text{width}}) = (2, 1) \tag{3.7}$$

respectively.

The up-scaling step is then written as

$$d^{\text{out}} = d^{\text{in}} *^{T} w \tag{3.8}$$

with w being the kernel and d^{in}, d^{out} the input and output, respectively. Considering an input of size (3, 2) and a desired output of size (6, 2), suitable kernel values have to be found to achieve up-scaling of factor (2, 1). As depicted in fig. 3.13, a transposed convolution can be understood as a normal convolution with padding.

Figure 3.14 shows this padding step with which equation 3.8 results to

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ d_{00}^{\text{in}} & d_{01}^{\text{in}} \\ 0 & 0 \\ d_{10}^{\text{in}} & d_{11}^{\text{in}} \\ 0 & 0 \\ d_{20}^{\text{in}} & d_{21}^{\text{in}} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} * \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} w_2 \cdot d_{00}^{\text{in}} & w_2 \cdot d_{01}^{\text{in}} \\ w_1 \cdot d_{00}^{\text{in}} + w_3 \cdot d_{10}^{\text{in}} & w_1 \cdot d_{01}^{\text{in}} + w_3 \cdot d_{11}^{\text{in}} \\ w_0 \cdot d_{00}^{\text{in}} + w_2 \cdot d_{10}^{\text{in}} & w_0 \cdot d_{01}^{\text{in}} + w_2 \cdot d_{11}^{\text{in}} \\ w_1 \cdot d_{10}^{\text{in}} + w_3 \cdot d_{20}^{\text{in}} & w_1 \cdot d_{11}^{\text{in}} + w_3 \cdot d_{21}^{\text{in}} \\ w_0 \cdot d_{10}^{\text{in}} + w_2 \cdot d_{20}^{\text{in}} & w_0 \cdot d_{11}^{\text{in}} + w_2 \cdot d_{21}^{\text{in}} \\ w_1 \cdot d_{20}^{\text{in}} & w_1 \cdot d_{21}^{\text{in}} \end{bmatrix} = \begin{bmatrix} d_{00}^{\text{out}} & d_{01}^{\text{out}} \\ d_{10}^{\text{out}} & d_{11}^{\text{out}} \\ d_{20}^{\text{out}} & d_{21}^{\text{out}} \\ d_{30}^{\text{out}} & d_{31}^{\text{out}} \\ d_{40}^{\text{out}} & d_{41}^{\text{out}} \\ d_{50}^{\text{out}} & d_{52}^{\text{out}} \end{bmatrix} \tag{3.9}$$

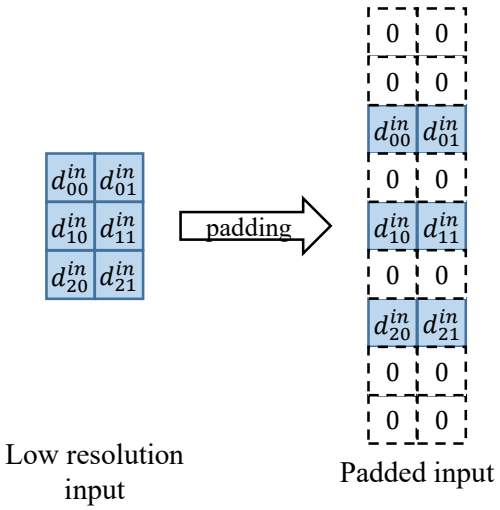


Figure 3.14: Visualization of the internal padding in a transposed convolution

With this knowledge it is possible to reproduce traditional up-scaling methods mentioned in section 3.2.1. For all the following examples, an up-scaling factor of (2, 1) and an input of size (3, 2) is considered. Furthermore, kernels of size (4, 1) and strides of (2, 1) are used, as introduced in eq. 3.6 and 3.7.

A problem where an output shall possess the original input values and have linearly interpo-

lated values in between them, can be formulated as

$$\begin{bmatrix} d_{00}^{\text{in}} & d_{01}^{\text{in}} \\ d_{10}^{\text{in}} & d_{11}^{\text{in}} \\ d_{20}^{\text{in}} & d_{21}^{\text{in}} \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} d_{00}^{\text{in}} & d_{01}^{\text{in}} \\ \frac{1}{2}(d_{00}^{\text{in}} + d_{10}^{\text{in}}) & \frac{1}{2}(d_{01}^{\text{in}} + d_{11}^{\text{in}}) \\ d_{10}^{\text{in}} & d_{11}^{\text{in}} \\ \frac{1}{2}(d_{10}^{\text{in}} + d_{20}^{\text{in}}) & \frac{1}{2}(d_{11}^{\text{in}} + d_{21}^{\text{in}}) \\ d_{20}^{\text{in}} & d_{21}^{\text{in}} \\ ? & ? \end{bmatrix}. \tag{3.10}$$

Therefore, with the help of equation 3.9, the kernel values result to $w = [0, 0.5, 1, 0.5]^T$. This is one version of a classical linear interpolation.

Another interpolation procedure is the nearest neighbor method. It simply takes the low resolution input and reproduces the rows. The following shows the problem that has to be solved

$$\begin{bmatrix} d_{00}^{\text{in}} & d_{01}^{\text{in}} \\ d_{10}^{\text{in}} & d_{11}^{\text{in}} \\ d_{20}^{\text{in}} & d_{21}^{\text{in}} \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} d_{00}^{\text{in}} & d_{01}^{\text{in}} \\ d_{00}^{\text{in}} & d_{01}^{\text{in}} \\ d_{10}^{\text{in}} & d_{11}^{\text{in}} \\ d_{10}^{\text{in}} & d_{11}^{\text{in}} \\ d_{20}^{\text{in}} & d_{21}^{\text{in}} \\ d_{20}^{\text{in}} & d_{21}^{\text{in}} \end{bmatrix}. \tag{3.11}$$

The kernel which is then computed, is defined as $w = [0, 1, 1, 0]^T$ and is denoted as *reproducing kernel* in the following.

A very primitive method of up-scaling would be to simply insert zeros into the low resolution input

$$\begin{bmatrix} d_{00}^{\text{in}} & d_{01}^{\text{in}} \\ d_{10}^{\text{in}} & d_{11}^{\text{in}} \\ d_{20}^{\text{in}} & d_{21}^{\text{in}} \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} d_{00}^{\text{in}} & d_{01}^{\text{in}} \\ 0 & 0 \\ d_{10}^{\text{in}} & d_{11}^{\text{in}} \\ 0 & 0 \\ d_{20}^{\text{in}} & d_{21}^{\text{in}} \\ 0 & 0 \end{bmatrix}. \tag{3.12}$$

This leads to kernel with values $w = [0, 0, 1, 0]^T$, called *zero padding kernel*.

Table 3.1 gives an overview on the above introduced kernel types. All of them are applied and tested in two setups in this work. First, as an additional implementation of the traditional methods (section 4.3) and second as a kernel initializer for a minimal LiDAR super-resolution network (section 4.4).

Table 3.1: Overview on interpolation kernels

kernel name	values	description
linear	$\left[0, \frac{1}{2}, 1, \frac{1}{2}\right]^T$	preserves the original values in the present rows and linearly interpolates the missing values in the rows between them
reproducing	$\left[0, 1, 1, 0\right]^T$	like nearest neighbor, simply replicate each row two times
zero padding	$\left[0, 0, 1, 0\right]^T$	inserts rows with zero values, should have low performance

3.5 Loss Functions for Super-Resolution

Super-resolution is a classical regression task. Most of the models for this task are trained with a pixel-wise loss to minimize the euclidean error between the predicted and the target high-resolution image. In the context of LiDAR point clouds, we speak of point-wise loss which is calculated on 2D distance images. However, there are several possibilities how to exactly design the loss, some of them are explained in the following sections.

3.5.1 Point-wise Loss

A point-wise loss compares the value of every single point in the predicted high-resolution scan with its corresponding counterpart in the ground truth scan, i.e. the points of both scans at the same azimuth and elevation angle. Least absolute deviations (L_1) and least square errors (L_2) are the two most used loss functions for regression tasks and are explained in the following.

L_1 and L_2 Loss

An L_1-loss function minimizes the absolute difference between the predicted values d^{pred} and the target values d^{gt}, according to

$$\mathscr{L}_1 = \sum_{\text{H,W}} \left| d^{\text{gt}}_{\text{h,w}} - d^{\text{pred}}_{\text{h,w}} \right| \tag{3.13}$$

where $d_{h,w}$ is the value at the index (h, w) in a 2D structure and H, W being the number of sample values in both dimensions. It is also called mean absolute error (MAE).

The L_2-loss function minimizes the squared difference between the predicted d^{pred} and the existing target values d^{gt} and is defined as

$$\mathscr{L}_2 = \sum_{\text{H,W}} \left(d^{\text{gt}}_{\text{h,w}} - d^{\text{pred}}_{\text{h,w}} \right)^2 \quad . \tag{3.14}$$

Another name for it is mean squared error (MSE).

Equations 3.13 and 3.14 show, that the L_2 error becomes much larger in the case of outliers compared to L_1. The reason is, that the squared difference between an incorrectly predicted value and its target value is even larger than the normal difference between them. Transferred to the problem in this work, it means that an L_2-loss "punishes" large errors more than an L_1-loss in distance estimation.

As a result, an L_1 error is more robust against outliers, while L_2 tries to adjust the model according to these outlier values, even on the expense of other good samples. Hence, L_2 may result in huge deviations in some samples which results in reduced accuracy. If it is possible to ignore the outliers in the dataset, it is best to use the L_1-loss function. On the other hand, if there should not be any undesired outliers in the dataset, they first have to be removed before applying the L_2-loss function.

When using L_2-loss it is assumed that the data follows a Gaussian distribution (minimizing the L_2-loss equals maximizing the log-likelihood of a Gaussian). This is not the case for real data, it usually has multiple peaks because there exists multiple variations. In other words, the distribution is multimodal. The problem is, in the loss function it is assumed that the particular input comes from a Gaussian which is a unimodal distribution, meaning there is only one single peak.

Figure 3.15 shows what happens if a unimodal distribution is fit to a bimodal one using L_2-loss. During optimization L_2-loss has to minimize the distance of the reconstruction to both peaks. Therefore, the Gaussian of the L_2-loss would be centered in the middle of the two modes. If data is now sampled from the Gaussian distribution, the sample is extracted from the middle of the two modes in image space, even though this region has a very low probability. Therefore, the sample is the average of the samples from the two modes which yields a blurry image.

Figure 3.16 shows a vivid example. Here, a network for image colorization was trained with an L_2-loss. The input is a gray scale image and the output the colored equivalent. Even though the output is colored and looks realistic, it is quite faded compared to the original which shows a higher color depth. To address the problem of L_2-loss yielding blurry outputs, some new methods for loss design have been introduced, some of them are mentioned in the following sections [19].

Masked Loss

In contrast to camera images, there are missing points in the LiDAR scans which makes it more complicated to train a neural network. Those points are called invalid points, i.e. positions

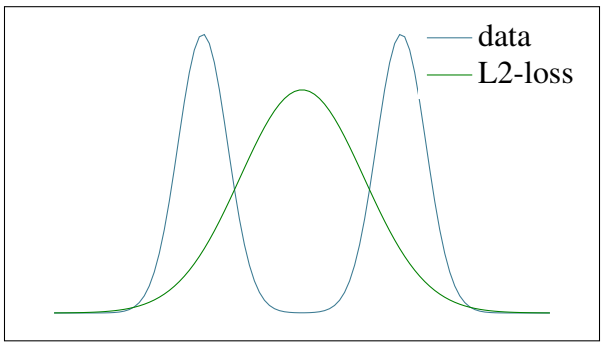


Figure 3.15: Fitting a Gaussian to a bimodal distribution

(a) input (b) output (c) ground truth

Figure 3.16: Example of L_2-loss yielding a faded output image for image colorization.

of azimuth and elevation angles where the LiDAR sensor did not receive any reflection, thus a dummy distance is assigned at this position in the 2D depth image. However, those values must not be considered in the loss function to avoid letting the network learn those random values.

More precisely, a network predicts real values for all points in the high-resolution output. If now, the error between the predicted and the ground truth image is calculated at a position with an invalid point a possibly big error is calculated, though the predicted value might actually be more reasonable than the inserted numeric value of the dummy, e.g. 0m or 500m.

Figure 3.17 shows a block diagram of the modified loss. It is necessary to remove invalid points when calculating the loss. To our advantage, the dataset provides an additional channel, which states whether a position in the 2D structure is valid or not, i.e. invalid mask. This information is directly obtained by the sensor, which recognizes whether it receives a reflection or not. The modified loss not only receives the ground truth and predicted image as an input, but additionally the invalid mask, which operates as a weight function for every single point (0 for invalid, 1 for valid).

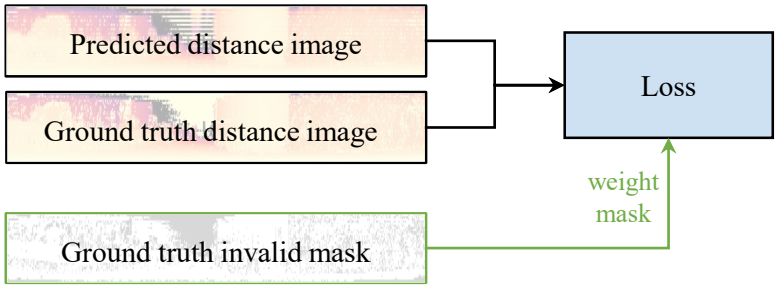


Figure 3.17: Block diagram of the modified loss

Distance weighted Loss

Another extension to the mentioned loss models, is a distance weighted loss. Here, not only weights of value 0 and 1 are given to the loss, but continuous values, depending on the distance value of the ground truth distance image. The reason is, that the same error in meter is more or less severe depending on at which distance this miscalculation happens. For example, if the predicted distance is 2m off the target value at a distance of 70m, this is less fatal, than if the same error of 2m happens at a distance of 5m to the car.

3.5.2 Perceptual Loss

As explained before, the Euclidean distance (L_2-loss) is insufficient for assessing structured outputs such as images, due to its unimodal character. Furthermore, it assumes pixel-wise independence, just like all previously mentioned loss versions. As LiDAR scans possess similar structures and perceptual features as images, this accounts for LiDAR scans as well. Zhang et al. showed that deep features trained on supervised objectives model low-level perceptual similarity and outperform widely-used regression metrics [20].

In their work, Johnson et al. used the VGG-16 network pre-trained on the ImageNet dataset to compute their perceptual loss for image super-resolution. Basically, their network architecture consists of two parts [8, 21, 22]. First, a so-called image transformation network, which does the up-sampling task, and then VGG-16. Both, the predicted high-resolution image from the image transformation network, as well as the high-resolution target image, are propagated through the loss network, i.e. VGG-16. The feature maps of both images are extracted at `relu3_3`. Then the squared normalized Euclidean distance between both feature representations is calculated. This loss is then minimized, which causes up-sampled images to be visually more similar to the original.

Figure 3.18 shows the network architecture of Johnson et al. [8]. In their paper they performed image super-resolution which refers to the blue path with y_c as the content target (high resolution image) and style transfer (not shown), which is not further considered in this work.

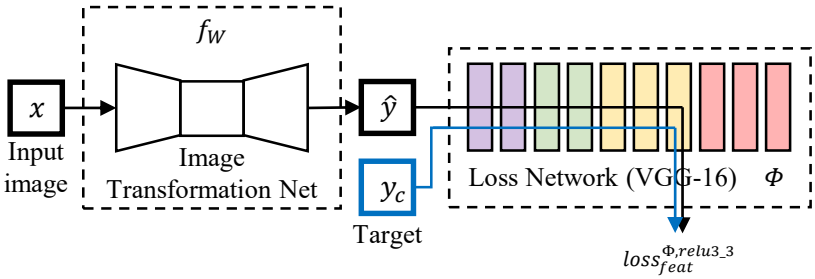


Figure 3.18: System overview Johnson et al. [8]: The image transformation network is trained to transform low-resolution images into a high-resolution output. The loss network is pre-trained for image classification and is used to define the perceptual loss functions that measure perceptual differences in content between the images. The loss network remains fixed during the training process (is not being trained).

For our usecase, it is not possible to use the network with the same weights. The main reason is, that the one Johnson et al. used is pre-trained on an image classification dataset. Thus, the network is not able to extract useful information from LiDAR data. The loss network is used to extract high-level content information from the images, therefore it is assumed it is possible to use a network which is pre-trained for semantic segmentation on LiDAR scans, in particular *LiLaNet* [23]. Such a network is aware of the underlying structure of the data and can generate the desired feature maps.

3.5.3 Adversarial Loss

An adversarial loss is used in generative adversarial networks (GANs) [13]. A GAN is a deep neural network architecture comprised of two networks, pitting against each other (figure 3.19). It can learn to mimic any distribution of data, thus outperform the improvement of networks using a perceptual loss. One of the two networks, called the generator G (petrol blue), generates new data instances, while the other, the discriminator D (salmon pink), evaluates them for authenticity. The discriminator decides whether each instance of data belongs to the actual training dataset or not.

The adversarial loss basically has the form of

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1-D(G(z)))] \tag{3.15}$$

with $p_z(z)$ being the input. $D(x)$ represents the probability that x came from the data rather than p_g, which is the generator's distribution over data x [13]. D is trained to maximize the probability of assigning the correct label to both training examples and sampled from G and simultaneously G is trained to minimize $\log(1-D(G(z)))$.

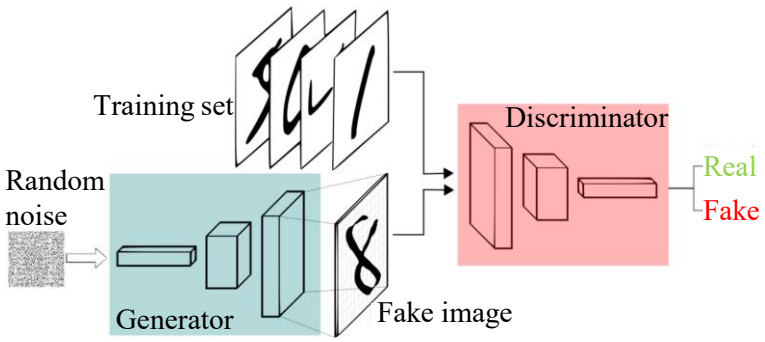


Figure 3.19: Schema of a GAN generating handwritten numbers from random noise. (source: [24])

4 System Requirements

This chapter gives an overview on the requirements to meet in this work. All of the introduced structures and pipelines have been developed during the course of this work and are implemented in Python 3.5 with TensorFlow 1.10 [25]. It is an open source library for high performance numerical computation and support for machine learning and GPU optimization. Here, all the results are obtained by training on a cluster with NVIDIA Volta GPUs.

The most important ingredient to this work is the dataset and the training data generation, which are explained in section 4.1. The subsequent section introduces the evaluation process and its corresponding metrics. In chapter 4.3, traditional up-sampling approaches, e.g. bilinear interpolation are evaluated on the introduced dataset and serve as the baseline of this work. The last chapter gives an overview on some experiment results with a minimal network architecture, which were performed prior to start developing a suitable up-sampling network.

4.1 Training Data

To train the LiDAR super resolution network, a large scale LiDAR dataset of urban and rural road scenes (section 4.1.1) was used. It served as the ground truth data for training. The actual training input to the network was generated by down-sampling the frames in this dataset, explained in section 4.1.2.

4.1.1 LiDAR Dataset

The dataset contains approximately 555,000 frames with different scenes (urban, rural and highway). The recorded frames were split into subsets for training, validation and testing. This is a typical procedure in supervised learning. The network is trained on the training split and the validation split is used as a reference, whether the network generalizes enough and can still perform well on unseen data. After the best performing network was chosen from several experiments, it is again validated on the test set to ensure absolute independence from any previous samples. In order to prevent correlations between the subsets, the dataset split was performed on a sequence basis instead via random frame selection.

Table 4.1 shows the details of the dataset split. One frame refers to a single 360 degree revolution of the LiDAR sensor, resulting in frames with a shape of 32x1800 points (explained in section 3.1). The optimized subset of the dataset are frames at which the difference between

the camera principal axis orientation and the current LiDAR azimuth angle at the camera time stamp does not exceed a threshold of 60 degrees. This is especially needed when using the auto-generated class labels (car, person, bike, ...) of the dataset [23].

Table 4.1: Split of the data into sets for training, validation, and testing.

	training	validation	testing
split factor	0.62	0.13	0.25
original frames	343,926	73,473	137,636
optimized frames	57,324	12,261	22,983

Figure 4.1 shows a 3D scene with class labeled points and the corresponding camera image. The dataset contains 13 semantic classes which are a subset of the Cityscapes dataset [26]: road, sidewalk, person, rider, vehicle, truck , bicycle, construction, pole, traffic sign, vegetation, terrain, and sky.

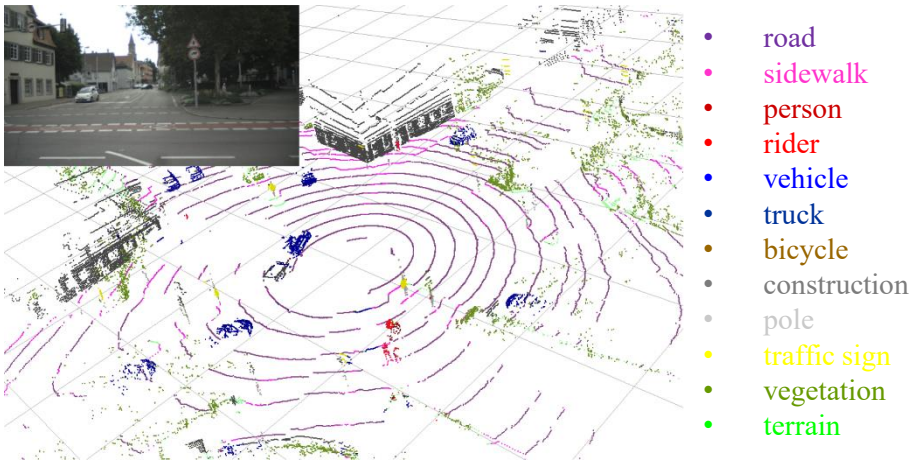


Figure 4.1: 2D projection of a LiDAR point cloud with semantically labeled points. The ego vehicle is located in the center of the image and headed to the top right, the corresponding camera image is displayed in the top left corner. Each point is colored and belongs to a certain semantic class, for example blue for vehicle. (source: [23]

In case of super-resolution no class labels are needed, however sometimes it is useful to compute the error between the up-scaled and the original high-resolution image for each class label separately, in order to investigate which object classes are most challenging for the up-sampling network. The networks in this work are trained on the training split of the original dataset and evaluated on the validation split of the optimized dataset. The reason is, that the higher amount of data during training overweights the accuracy of the labels, enhanced by the fact that the labels are not very important for super-resolution training.

4.1.2 Training Data Pre-Processing

The above mentioned dataset provides the ground truth data for the up-sampling task. In order to let a network learn an up-scaling factor of (f_h, f_w), the ground truth data has to be down-sampled by this factor to provide the desired network input (training data). In particular, for a factor of $(f_h, f_w) = (2, 1)$, the single images in the dataset are reduced by a factor of 2 in the height dimension.

Figure 4.2 shows how the desired low-resolution training data is generated. As opposed to many image super-resolution tasks, here no bicubic down-sampling interpolation is used, rather a primitive deletion of every other row / layer. The reason is simple: in the later application, a real world low resolution point cloud will be the input to the network, which then shall interpolate intermediate layers in between the existing ones. A bicubic down-sampling would generate unrealistic blurry training input data, therefore, simply every other layer of a frame is removed.

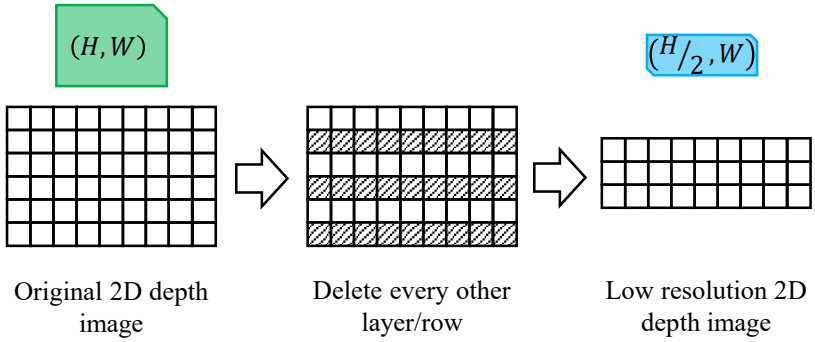


Figure 4.2: Training data generator: in the online training data generator, the high-resolution ground truth frames (green) are loaded and reduces to half the height in order to create the low-resolution training frames (blue). The down-sampling is achieved by a simple extinction of the sensor layers (rows of the frames).

4.2 Evaluation Process and Metrics

Figure 4.3 shows the basic setup of the evaluation pipeline. Here the data from the optimized validation split are used together with a pre-trained network of one of the presented experiments (chapter 6). The predicted high-resolution output is fed to the evaluation unit, together with the corresponding ground truth. This works very similar to the loss module, except that several different metrics are computed on the whole dataset and the output is not further used for training. The evaluation is designed as an online evaluation, meaning that after a certain amount of training iterations, the pipeline switches its mode, freezes the network containing the so-far learned weights, and propagates the evaluation data through it. With this method it is possible to simultaneously generate training curves on both training and evaluation data,

which enables early detection of overfitting or other symptom of bad generalization of the network.

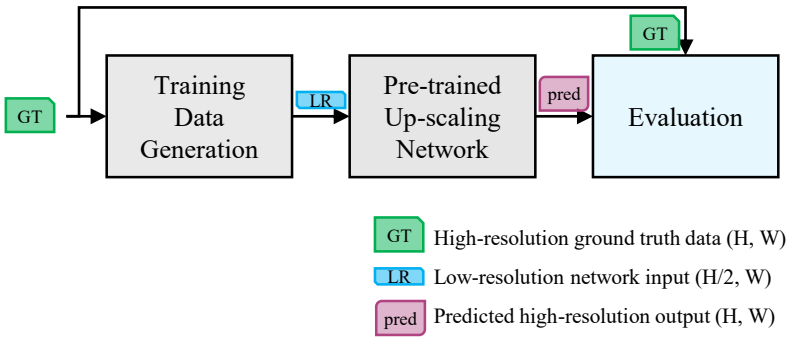


Figure 4.3: System overview on the evaluation pipeline. The leftmost block corresponds to the training data generation block, introduced in the previous section, followed by the up-sampling network. The network output and the ground truth data is fed to the evaluation block.

Within the evaluation module, several metrics are computed to gain information about different properties of the network. All metrics exclude the invalid points of the ground truth data for their calculation. As mentioned in section 3.5.1, MSE is the most used error function for super-resolution tasks and also an important evaluation metric. In contrast to the loss in the training, where the MSE is computed over the current batch, in evaluation it is necessary to compute the MSE over the complete dataset [1].

The point-wise MSE for the valid points of the predicted frame d^{pred} and the ground truth frame d^{gt} is defined as

$$\text{MSE} = \frac{1}{NHW} \sum_{n,h,w} \left(d_{nhw}^{\text{pred}} - d_{nhw}^{\text{gt}} \right)^2 \tag{4.1}$$

with N, H, W being the batch size, image height and width, respectively. n denotes the current batch.

Furthermore, it is useful to compute the mean absolute error (MAE), section 3.5.1, which is defined as

$$\text{MAE} = \frac{1}{NHW} \sum_{n,h,w} \left| d_{nhw}^{\text{pred}} - d_{nhw}^{\text{gt}} \right| \tag{4.2}$$

for all valid points. It is equivalent to the mean deviation (in meter) and makes it easy for humans to understand the extend of the physical error in the predicted frames[2]. Both metrics are not comparable in their values, as they range in different scales.

In some later experiments an additional evaluation of semantic segmentation is conducted. The mean intersection over union (IoU) score is used as a metric. The IoU per class if defined

[1] In training the loss implementation of TensorFlow is used (`tf.losses.mean_squared_error`), in the evaluation the metrics implementation is applied (`tf.metrics.mean_squared_error`).

[2] Similar to the MSE, `tf.metrics.mean_absolute_error` is used here.

as

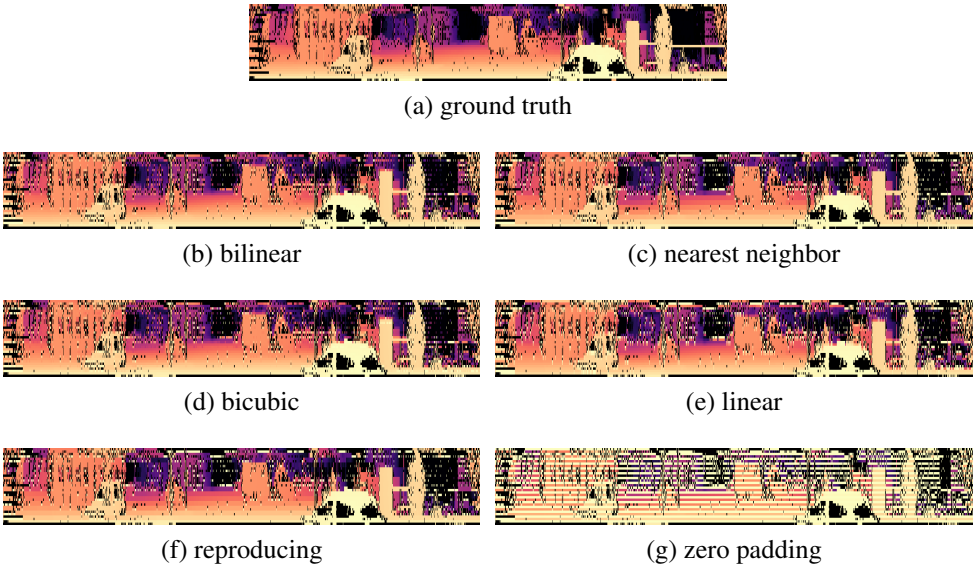
$$\text{IoU} = \frac{\text{true positives}}{\text{true positives} + \text{false positives} + \text{false negatives}} \quad . \tag{4.3}$$

The mean IoU score is simply the average over all classes.

4.3 Traditional Methods as a Baseline

For the evaluation of traditional up-sampling methods, the experimental setup shown in figure 4.3 is used. The pre-trained network, here, is a fixed interpolation, e.g. bilinear. Inserting the traditional approach into the whole pipeline and using the same evaluation block as for the other learning based methods, ensures same conditions and comparable evaluation results.

Figure 4.4 and 4.5 show examples of the six methods evaluated in this test set. Bilinear, nearest neighbor and bicubic interpolation (from section 3.2.1) are implemented with the traditional image resize methods available in TensorFlow[3]. The remaining three methods, linear, reproducing and zero padding, are implemented with a non-trainable transposed convolution. The distance of invalid points is set to 0. Figure 4.4 illustrates the distance images. It is not intuitive for humans to interpret scenes in the distance image, therefore fig. 4.5 shows the same scenes visualized in a 2D projection of a 3D point cloud.



(a) ground truth

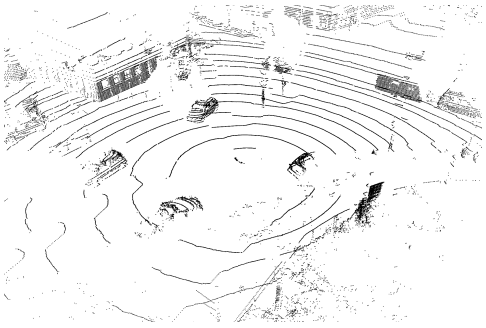
(b) bilinear (c) nearest neighbor

(d) bicubic (e) linear

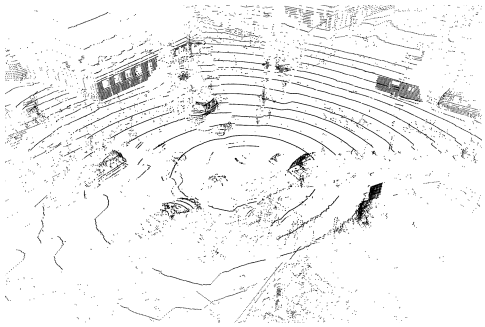
(f) reproducing (g) zero padding

Figure 4.4: Comparison of LiDAR distance images up-sampled with six traditional methods (4.4b to 4.4g) to the high resolution ground truth (4.4a). The brighter, the closer the measured point, invalid points are depicted in black.

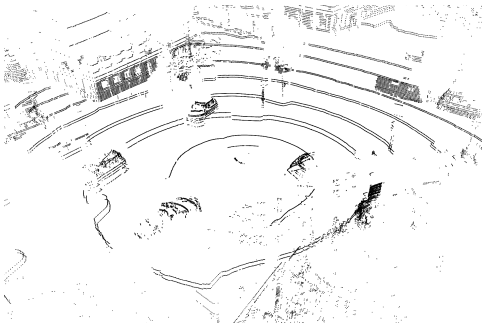
[3] from `tf.image` methods `resize_bilinear`, `resize_nearest_neighbor`, and `resize_bicubic`



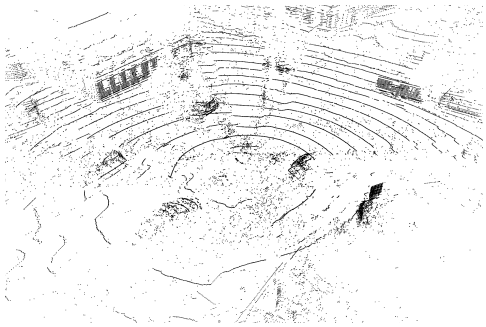
(a) ground truth



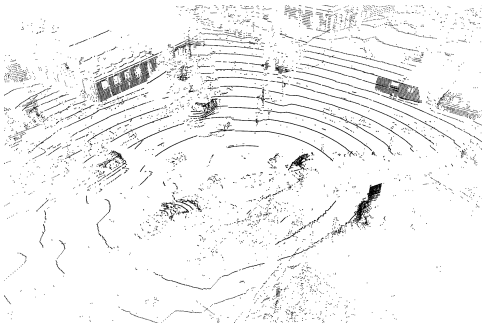
(b) bilinear



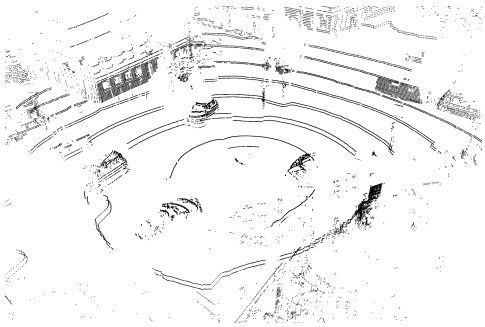
(c) nearest neighbor



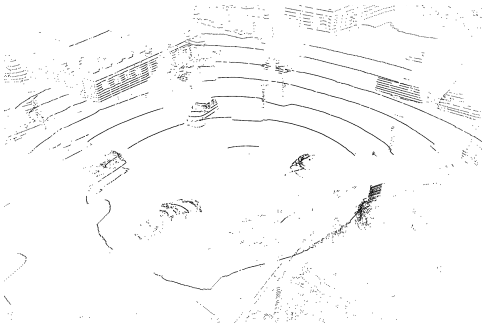
(d) bicubic



(e) linear



(f) reproducing



(g) zero padding

Figure 4.5: Comparison of 3D LiDAR point clouds, up-sampled with six traditional methods (4.5b to 4.5g) as well as the high resolution ground truth (4.5a). Invalid points of the 2D structure are not shown here as they correspond to missing points in 3D.

Table 4.2 shows the experiment results of the described methods. As the reproducing kernel matches the nearest neighbor method, it obtains the same evaluation results. A mean deviation of 2.75 meters is achieved between the interpolated and the true high-resolution distance image. As expected, the method of zero padding (inserting zeros in missing rows) achieves very bad results, as it only restores the desired output size, but no reasonable values.

Table 4.2: Evaluation results when applying traditional up-scaling algorithms to the complete optimized validation dataset.

	MSE	MAE
bilinear	**87.5**	**2.25**
nearest neighbor	145.0	2.75
bicubic	96.8	2.55
linear	193.1	4.75
reproducing	145.0	2.75
zero padding	604.8	12.75

Bilinear and linear interpolation are not the same computation and therefore differ in performance. The bilinear version takes into account horizontally neighboring points additionally to the neighboring layers. Therefore it receives more context information resulting in smaller errors. With the linear kernel, only the point above and below the point of interest is considered. However, this version retains the existing layers and only makes an interpolation of the missing points, in contrast to bilinear, which alters all data points. Therefore, the error in the existing layers must be zero. Nevertheless, this effect is less dominant than the context information from neighboring points, thus bilinear performs better than linear.

With 2.25m of mean deviation, the bilinear method also performs better than the bicubic interpolation. However, in most image up-scaling cases bicubic interpolation achieves better results than bilinear interpolation. In this case, it is assumed that the additional points used for bicubic interpolation are too far away in distance, from the point that is interpolated. Therefore, the outcome is negatively influenced by those values. In particular, the bilinear interpolation only takes into account the distance values of directly neighboring layers. The bicubic interpolation however uses the directly neighboring layer and the one that is three layers apart from it in the high-resolution case to compute the distance of the interpolated point (figure 4.6). In the denser area, layers are $\gamma = 0.33$ degrees apart, going up to $\gamma = 9.36$ degrees in the sparse part of the sensor. Assuming a point at distance $d = 40m$ shall be interpolated, possible points at the same distance, which are included in the interpolation, are $l = 70cm$ away in the dense part of the sensor and many times over in the sparse area. Therefore, it cannot be assumed that the context is still the same that far apart from the point.

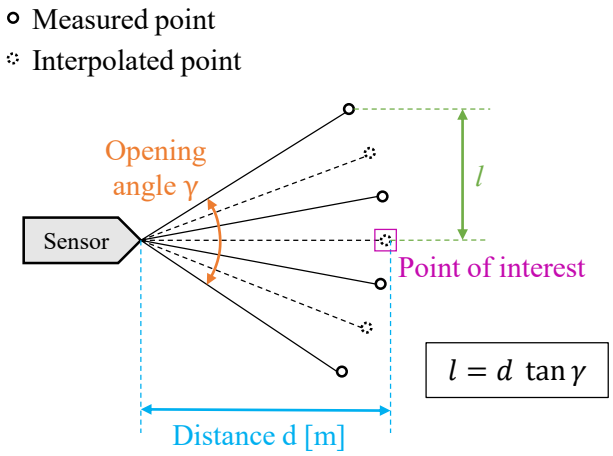


Figure 4.6: Schema of the relationship between opening angle and distance

4.4 Finding useful system settings

Prior to the design phase of an up-sampling network architecture, a minimal experiment setup was created. It is used to explore typical behavior of a CNN fed with 2D LiDAR depth images.

Figure 4.7 shows the pipeline of the minimal architecture, which performs up-scaling with a single transposed convolution with one filter (orange), i.e. a CNN with one network layer. Mathematically this layer is able to represent the before mentioned traditional methods of bilinear and nearest neighbor interpolation. Therefore, the question asked in this setup is, whether this single layer will learn a bilinear interpolation (which achieved the best result in section 4.3) or if it finds something smarter. As the network is very small, it is fast to train and many different experiments, providing useful insights, can be made.

4.4.1 Influence of invalid points

This section is about how to deal with so-called invalid points within the frames of the dataset. There are cases where the LiDAR sensor does not receive a reflection from its emitted beam, e.g. when pointed towards the sky. As explained in section 3.1, a 2D representation of the 3D data is used in this work, which means that the sparse data has to be transferred to a dense 2D structure. The locations where no reflection was received is represented by a dummy value with a consistent distance value over the whole dataset (for example d=0 or d=500). These points are called invalid points. Depending on the choice of the dummy value, the trained network produces different output and differs in performance. However, the network should be independent or at least not be influenced by those points.

Figure 4.8 shows two example scenes in three different versions. In one of those versions, a striped pattern emerged (fig. 4.8b) during training, the dummy value for invalid points was

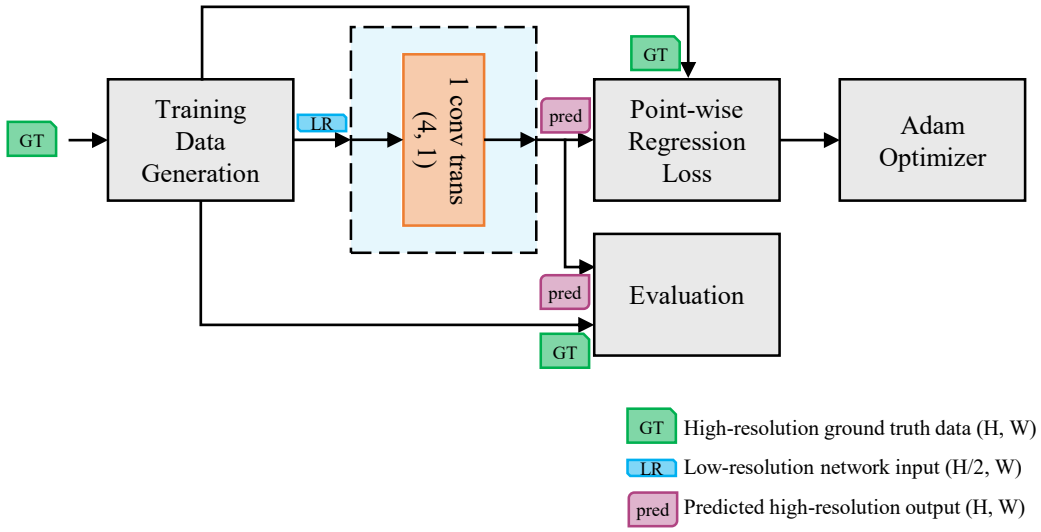
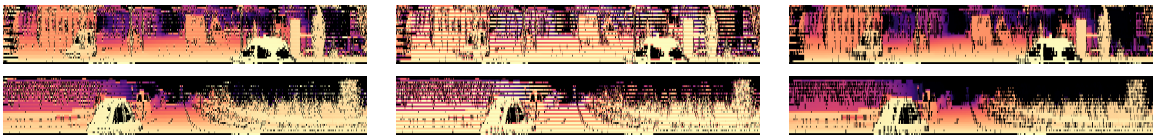


Figure 4.7: Setup of the minimal up-scaling network with one transposed convolution (orange) and a point-wise L_2-loss.

set to a high distance. No pattern occurs (fig. 4.8a) when the invalid points are set to a low dummy distance.



(a) invalid distance at 0m, no loss mask

(b) invalid distance at 500m, pure loss mask

(c) invalid distance at 500m, extended loss mask

Figure 4.8: Striped pattern in distance images

Figure 4.9 shows how this pattern emerges, assuming a high default value (d=500) for invalid points (dark red). Taken a regular linear interpolation, the high distance values of invalid points in the low resolution scan are transferred to the high-resolution prediction (light red), making the error very high at those positions (upper part). In an architecture, where an optimizer tries to find the optimal interpolation by minimizing the error, no linear interpolation (or similar) will be learned. In order to avoid having high errors, the optimizer will rather force the network to learn to interpolate all rows with zero (gray in lower part). This causes the striped pattern.

In order to verify these assumptions, three settings to weight the input of the loss function are introduced. In the first one, all points are equally weighted with 1 (no mask applied), second, the invalid mask is used to weight the invalid and valid points with 0 and 1, respectively. The last option is an extended version of the mask, it excludes additional points above and below an invalid point.

Table 4.3 shows the evaluation result after 15,000 iterations with a batch size of 10 for the same minimal network, but with the three different loss masks and a high and a low default

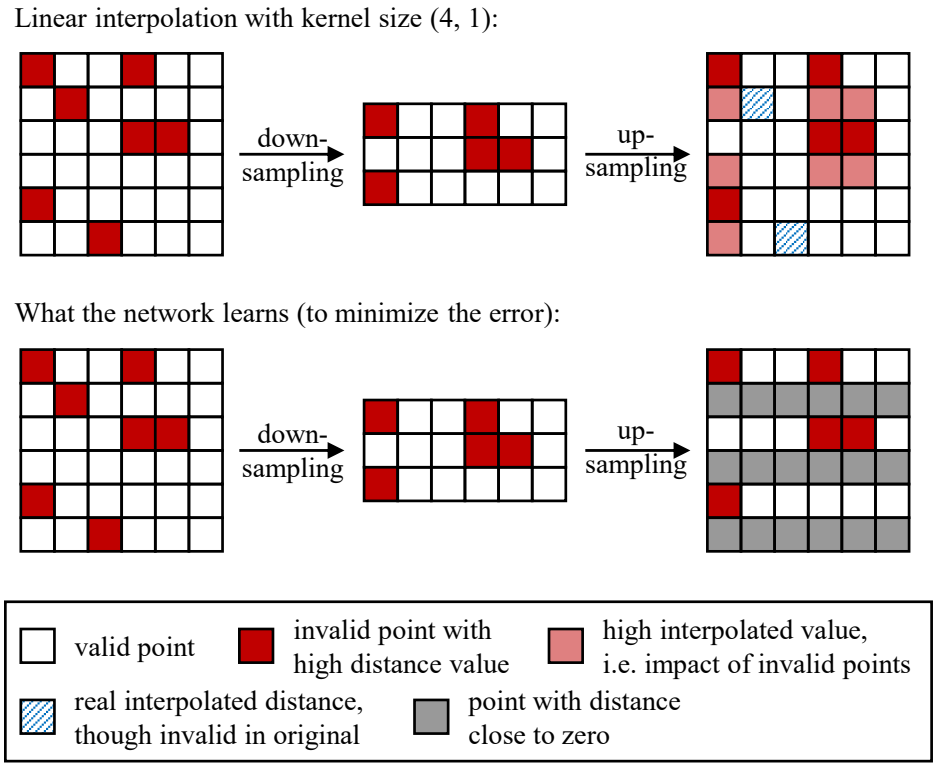


Figure 4.9: Influence of invalid points (red) to the up-sampling process, when the default distance is set to a high value

value for invalid points. For the evaluation on the complete validation split all invalid points are masked out for all settings. In the additional two columns marked with ex, the extended mask was used in the evaluation. The most obvious observation is, that the error is higher in the experiments with the higher distance for invalid points than the ones with a lower value. For the experiment at a distance of 0m, it can be seen that the mask yields some improvement over ignoring special treatment of invalid points at all.

The previously mentioned striped pattern occurs only in the version with distance at 500m and the pure invalid mask. Striped pattern means, all points of every other layer are set to 0. For the other two experiments with this high invalid distance, a ring of points at about 500m and 250m distance can be observed in the point cloud. Meaning the invalid points at 500m and the neighboring points which had to be interpolated between 500m and a closer distance, e.g. 15m, lying in the ring at about 250m. In the case of no mask applied, the resulting point cloud, first features those rings and second, has shifted layers. This means the interpolated layers are not aligned with the existing ones, but shifted towards the sensor origin. This happens because, even though the network has to interpolate the high values of invalid points, it tries to minimize the error by shifting all interpolations closer to zero. With the mask used in the loss, the network ignores the high values of the invalid points. However, assuming a linear interpolation, it would still result in neighboring points having very high values and

thus a very high error, as shown in figure 4.9. To prevent this from happening, the network sets all interpolated points to zero (fig. 4.8b). In contrast to the training without any mask, it does not have to take care of the true high values of the invalid points anymore, therefore the error is considerably lower.

Considering the training with the extended mask now, it can be seen in the distance images that the striped pattern vanished (fig. 4.8c), but the error is again very high. However, it has to be considered, that the evaluation metrics are all computed with the pure mask applied. The second two columns of tab. 4.3 show the result when the extended mask is not only applied to the loss, but also the evaluation. The values of these two versions cannot be compared. It has to be noted that the same error values are now achieved for both distances. This means the network is independent of the dummy values assigned to invalid points. However, this special form of the extended mask only works for this minimal setup, in a bigger network, the influence of the invalid point will extend to the complete receptive field, which is considerably bigger, making it impractical to filter out all effects. Therefore, in all following experiments, the default distance of invalid points is set to zero and the pure invalid point mask is used within the loss function for training.

Table 4.3: Evaluation results of the minimal network with different loss masks and default values for invalid points

invalid distance [m]	loss mask	MSE	MAE	MSE^{ex}	MAE^{ex}
0	none	110.9	3.30	-	-
0	pure	83.5	2.50	-	-
0	extended	86.2	2.24	16.0	0.81
500	none	5330.0	23.64	-	-
500	pure	336.7	8.62	-	-
500	extended	8358.0	28.25	16.0	0.79

4.4.2 Kernel Initializer Types

The previous experiments with the minimal network, are conducted without any initialization of the transposed convolution weights. In the traditional algorithms section, three of the six presented methods are implemented with a fixed non-trainable convolution layer: linear, reproducing, and zero padding. This section uses those kernel types as an initializer for the transposed convolution in the minimal network, in order to analyze the training behavior and investigate potential improvement over an uninitialized network.

Figure 4.10 shows the training curves of all four settings. It can be observed, that the kernel initializers do not result in any advantage in terms of performance, as all trainings converge to approximately the same error value. The only difference can be spotted in the convergence speed and the start value. As expected, the start value (shown here after 200 iterations) with the

zero padding kernel is quite high, as well as the uninitialized version. The other two kernels are closer to the one that is learned by the network.

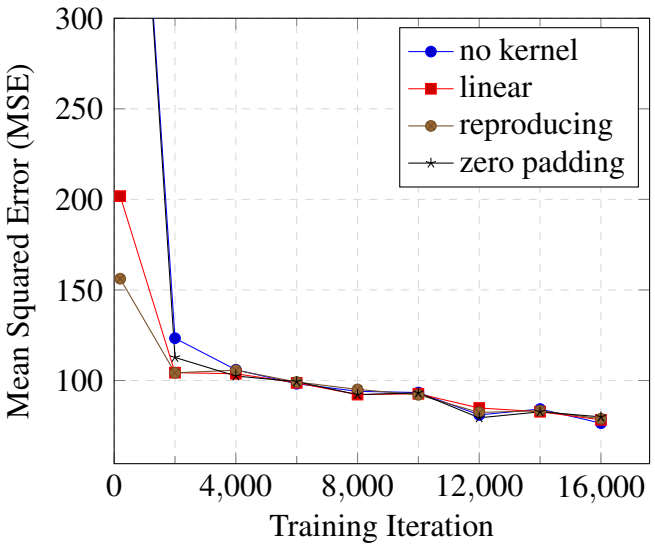


Figure 4.10: Comparison of the MSE on the complete evaluation dataset over training iterations with different kernel initializers. The closer the kernel initializer towards a linear interpolation, the faster the convergence.

Table 4.4 shows the kernel values at the beginning and at the end of the training. All four trainings resulted in the same kernel values to be learned. As expected, the kernel represents a linear interpolation. However, as opposed to the linear interpolation described by the *linear* kernel, here the order of which lines are maintained and which are interpolated are swapped. Furthermore, the layers are not evenly weighted with 0.5, but the upper layer is always weighted a bit stronger than the lower layer, 0.69 vs. 0.4. The MAE of these trainings is around 2.5, which is higher than the traditional bilinear interpolation with 2.25 and the MSE around 83 is a bit lower. One would expect the network is able to learn something smarter, that works better than the bilinear interpolation. However, it is important to mention, that the minimal network only had the information of adjacent layers in order to interpolate, the bicubic method additionally has the information about the neighboring points in the same and the adjacent layers. Therefore it has more information available to generate the interpolated output. However, the network almost achieved the same results, even with less information provided and achieved an improvement over the linear kernel.

Table 4.4: Kernel value development over training

kernel name	initializer	after 16,000 iterations
no kernel	-	$[0.69, 1, 0.40, 0]^T$
linear	$[0, 0.5, 1, 0.5]^T$	$[0.69, 1, 0.40, 0]^T$
reproducing	$[0, 1, 1, 0]$	$[0.69, 1, 0.41, 0]^T$
zero padding	$[0, 0, 1, 0]$	$[0.69, 1, 0.41, 0]^T$

5 Proposed Systems

In the course of this work, a modular pipeline was designed and implemented in `TensorFlow`. All experiments use the same pipeline, which has interchangeable modules.

Figure 5.1 shows a block diagram of the very basic pipeline. Only the up-sampling network (blue) and its corresponding loss function differ in each experiment setup, the other blocks stay the same. The training data generation is an online down-sampling step that loads the high-resolution ground truth data and generates low-resolution training data (explained in section 4.1.2). Afterwards, the up-sampling network receives the low-resolution input and outputs predictions of high-resolution images that have the same shape as the ground truth data. The loss calculates an error between those images and its scalar output is then fed to the Adam optimizer, which updates the weights in a backward pass. An evaluation module, which runs in a parallel branch, computes the metrics presented in section 4.2. With this it is possible to obtain the same metrics for the training and evaluation dataset by a simple mode switch.

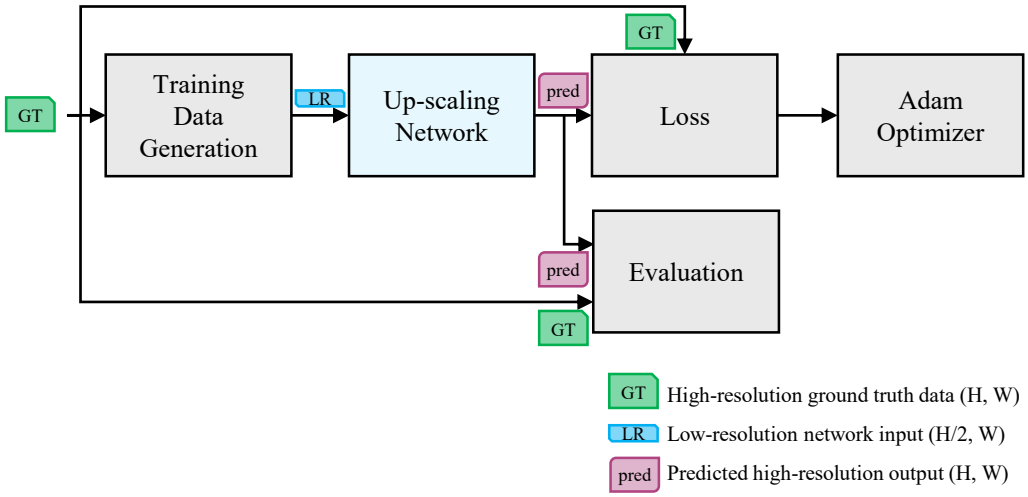


Figure 5.1: Basic structure of the modular pipeline constructed in this work.

In the following, section 5.1 gives a short overview on the two main network architectures used within the proposed systems. The subsequent sections present different experiment setups, their results are presented in chapter 6.

5.1 Network Architectures

5.1.1 Image Transformation Network

The actual up-sampling network within the architecture is referred to as the image transformation network. It is inspired by the one presented by Johnson et al. [8].

Figure 5.2 shows the construction of the model. The network possesses four residual blocks (yellow), each with two convolutions of 64 filters and a kernel size of 3x3, followed by batch normalization. After the residual blocks, the actual up-scaling happens within a transposed convolution. The number of the required layers n can be derived from the up-sampling factor f

$$n = \log_2(f) \quad f \in \{2^i | i \in \mathbb{N}_0\} \tag{5.1}$$

with $f = 2$, n results to 1. As explained in section 3.4.2, a kernel of size 4x1 is applied. The subsequent output layer then generates a one channel output with a 9x9 kernel, equaling the predicted high resolution distance image.

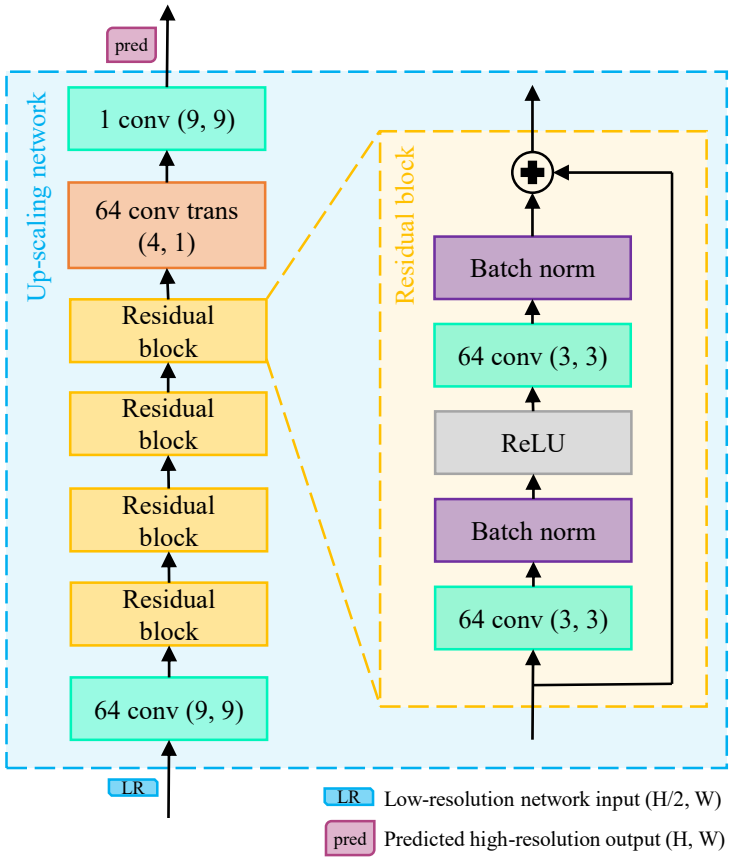


Figure 5.2: The image transformation network is the centerpiece of the architecture and is derived from Johnson et al. [8]

5.1.2 Semantic Segmentation Network

In some experiment setups a helper network is used. The so-called *LiLaNet* is originally used for semantic segmentation of point clouds [23].

Figure 5.3 shows the structure of the network. A detailed explanation can be found in [23]. However, two modifications have been applied. First, the input to the network is only the distance channel, not additionally the intensity channel. Second, the number of filters is reduced by 77%. However, a mean IoU of 56% was obtained, which is only 8% worse than the one in the paper. The reason for the modification is explained in section 5.4.

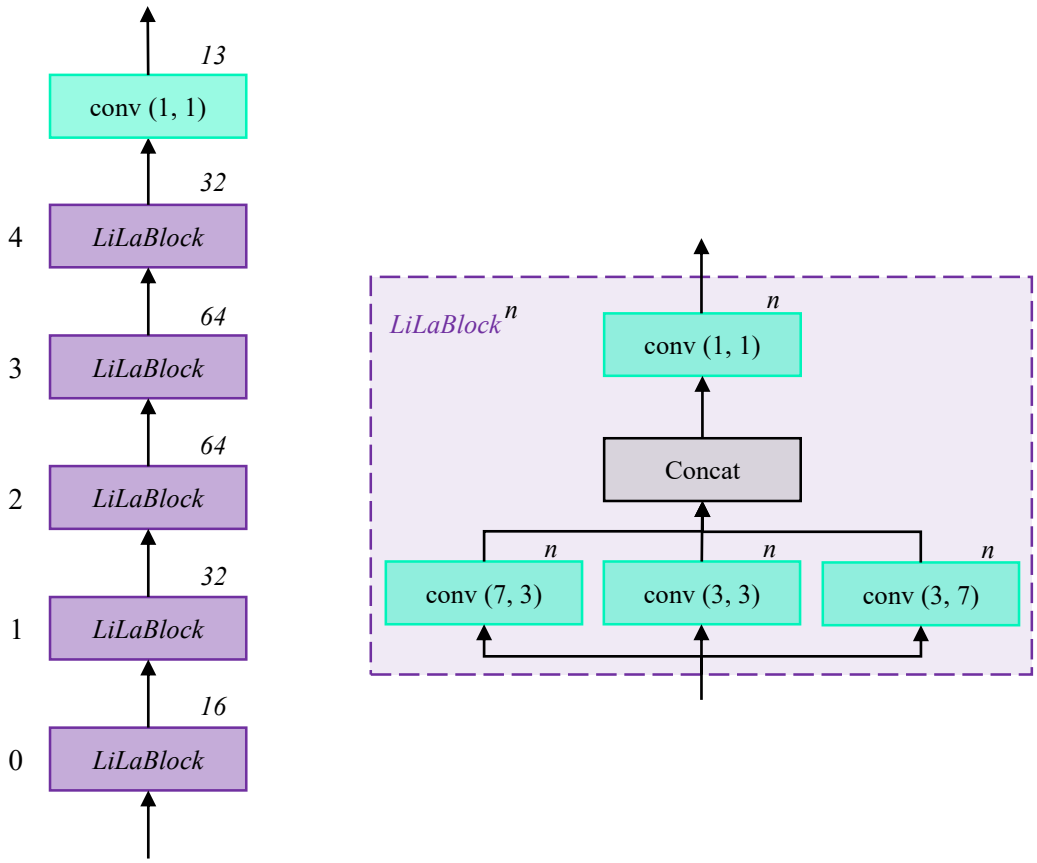


Figure 5.3: The *LiLaNet* [23] is a semantic segmentation network for LiDAR. It consists of a sequence of five consecutive *LiLaBlocks*. The final 1 x 1 convolution reduces the dimensionality to the desired label set (i.e. 13 semantic classes).

5.2 Regular Up-Sampling Network

The regular up-sampling network is the core product of this work. It is further extended in subsequent experiment setups.

Figure 5.4 depicts the pipeline for this system. The image transformation network module (blue) equals exactly the architecture introduced in section 5.1.1. The input layer receives a distance channel and, depending on the setup, also the corresponding invalid mask of the current batch. In the loss, only the valid points are considered, as well as in the evaluation module, which is not depicted in the block diagram. A batch size of 10 was used.

Two major experiments have been conducted with this setup. As explained in section 3.5.1, though L_2-loss is most commonly used for regression tasks, it yields blurry images. Therefore, trainings with both L_1- and L_2-loss are examined in this experiment series.

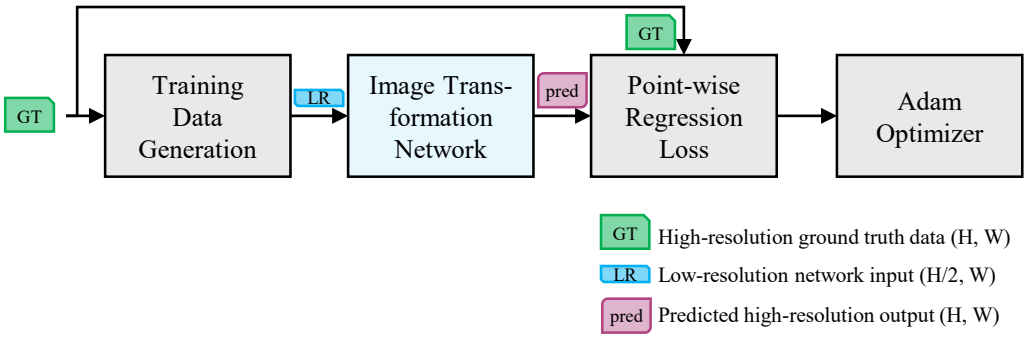


Figure 5.4: System setup with the regular network architecture (evaluation path not shown here)

5.3 Sparsity aware Up-Sampling Network

The sparsity aware up-sampling network not only predicts a high-resolution distance image, but also its corresponding high resolution invalid point mask. It is referred to as sparsity aware network, because point clouds are sparse data structures. Due to the given 2D structure of the input data, the network is not aware of this. However, the dummy values inserted for missing points cause effects which reduce the performance of the network.

Figure 5.5 shows the extension of the regular pipeline to a multi-task problem. As there are two tasks to train on, there are also two losses, which are then combined in the total loss block to generate a common input to the optimizer. The image transformation network is the same as described in section 5.1.1, except for one modification in the output layer.

Figure 5.6 shows this modification in the networks output layer. It is now possible to generate both the prediction for the high-resolution distance image (violet) as well as the logits for the high-resolution invalid point mask (gray). For each output, a separate convolution layer is

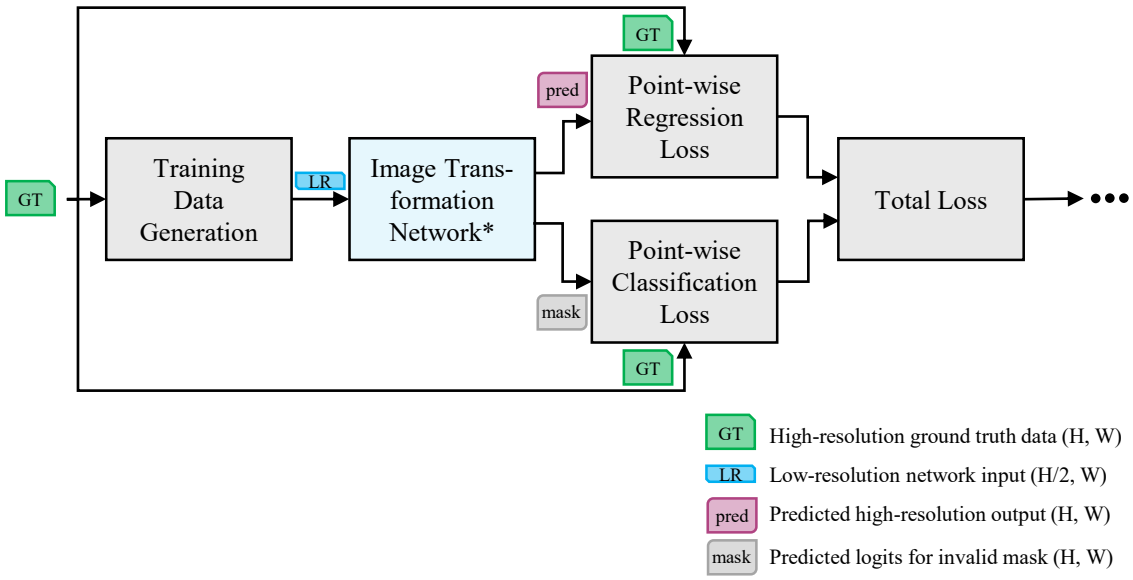


Figure 5.5: System overview on the sparsity aware network architecture (evaluation path not shown)

inserted. The one for the distance channel is exactly the same as before. The one for the invalid mask possesses two filters, which generate an output with two channels. Those channels are the logits for the classification prediction. As there are two classes, valid and invalid, the output possesses two channels[1].

The logits are needed to compute the point-wise classification loss, which in this case is a softmax cross-entropy loss. Therefore, the ground truth labels are one-hot encoded (resulting in two channels) and inserted together with the logits. Afterwards, the scalar output of the regression loss, which is a L_1-loss, and the classification loss are combined in the total loss block. Two different options are examined in this experiment setup and presented below.

5.3.1 Multi-Task Learning with Linear Loss Combination

The easiest and most obvious way to train a multi-task problem, is a linear combination of the losses. Here, the regression loss (MAE) and the classification loss (cross entropy) are combined to the total loss

$$\mathscr{L}_{\text{total}} = (1-p)\cdot\mathscr{L}_r + p\cdot\mathscr{L}_c \tag{5.2}$$

with $0 < p < 1$ being a constant weighting factor, $\mathscr{L}_r$ and $\mathscr{L}_c$ being the regression and classification loss, respectively. Several different weighting factors have been investigated.

[1] in this binary decision one channel would be enough to solve the problem, but it is easier for implementation purposes to include the layer like this into the system

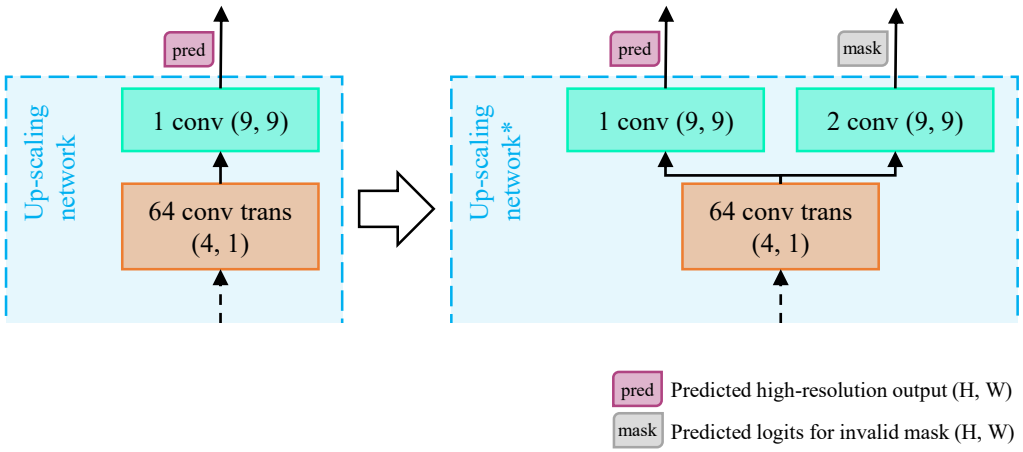


Figure 5.6: Modification of the original image transformation network (left) to generate two outputs (right). The violet rectangle is the prediction for the high-resolution distance image and the gray one represents its corresponding logits for the invalid mask.

5.3.2 Multi-Task Learning with trainable Loss Weights

A fixed linear combination of the two losses might not be the best way, therefore it is possible to learn the weighting factor within the training, too. Kendall et al. proposed a method using uncertainty to weight losses [27]. They state, that a combination of the two losses can be defined as

$$\mathscr{L}_{\text{total}} = \frac{1}{2\sigma_r^2} \cdot \mathscr{L}_r + \log(\sigma_r) + \frac{1}{\sigma_c^2} \cdot \mathscr{L}_c + \log(\sigma_c) \tag{5.3}$$

with σ_r and σ_c being trainable variables for the regression and classification loss, respectively. The log terms prevent the optimizer from pushing σ_r and σ_c towards infinity in order to minimize the loss. Therefore, it is possible to let the network decide which loss of which task has to be weighted more. An additional scaling factor can be introduced to balance differing scales of the losses. Kendall et al. further showed, that multi-task learning can improve single tasks, by introducing a so called helper task. The helper task in this work is the prediction of the invalid points, which shall improve the up-sampling.

5.4 Feature Reconstruction Up-Sampling Network

Another way to train the network for up-sampling, is to use a feature reconstruction network with a perceptual loss. As described in section 3.5.2, a perceptual loss can better reconstruct the high-level information of the data than a low level point-wise loss, as used in the setups before. This setup is similar to the regular setup in section 5.2, except that a feature extraction network is inserted between the image transformation network and the loss.

Figure 5.7 shows a block diagram of the feature reconstruction network. Both the predicted high resolution image and the corresponding ground truth are propagated through a pre-trained *LiLaNet* (properties from section 5.1.2), which is not being updated during training. At a specific location within the network, the feature maps are extracted and an L_1-loss is computed between them. The extraction points for the feature maps are the ReLU activations after the final 1x1 convolution in each *LiLaBlock*, referred to as bottleneck convolution. Extractions at the other layers are also possible. However, in this setup we focused on the bottleneck layer and only varied the block number, i.e. more at the edge or middle of the network.

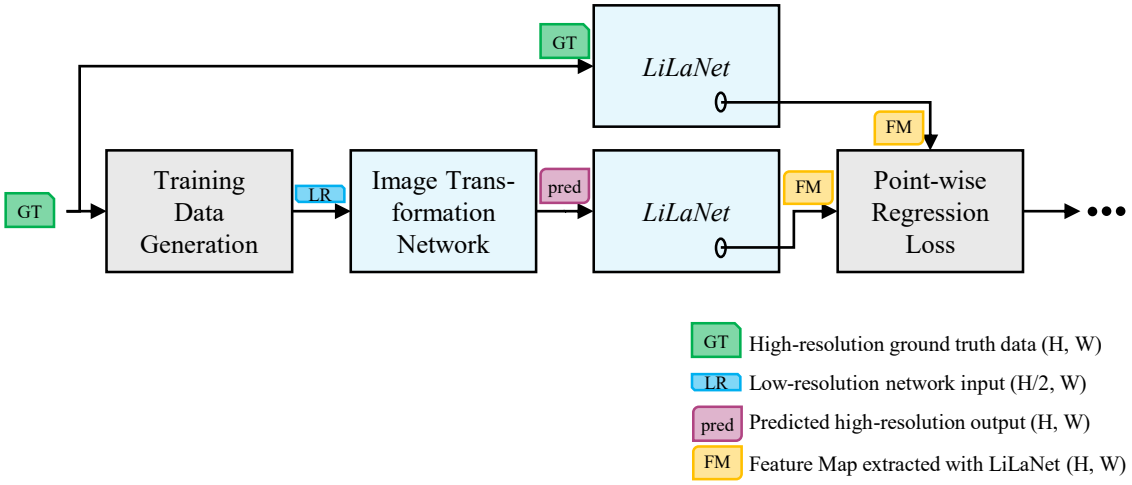


Figure 5.7: Feature reconstruction system using a LiDAR semantic segmentation network *LiLaNet* for feature extraction. The perceptual loss equals a L_1-loss on the extracted feature maps (yellow).

5.5 Semantically guided Up-Sampling Network

This section introduces a system setup which uses additional semantic information to improve the performance of the up-sampling network. Generally speaking, it is a very similar approach to the feature reconstruction network, as it also makes use of the high-level semantic information here. However, this network not only relies on the semantic labels to train the up-sampling, but also on the quality of the up-sampled output itself. This multi-task problem is similar to the one presented in the sparsity aware system setup.

Figure 5.8 illustrates the network architecture. First, the up-sampling network predicts a high-resolution distance image. Subsequently the L_1 distance between the prediction and the ground truth is calculated, i.e. the regression loss. In a parallel path, the network output is fed to a semantic segmentation network, called *LiLaNet*. Other than in section 5.4, it is propagated through the complete network. The 13 channel outputs are the logits for the 13 semantic classes. Afterwards the softmax cross-entropy loss is calculated and combined with the regression loss. The combination factor p is trainable.

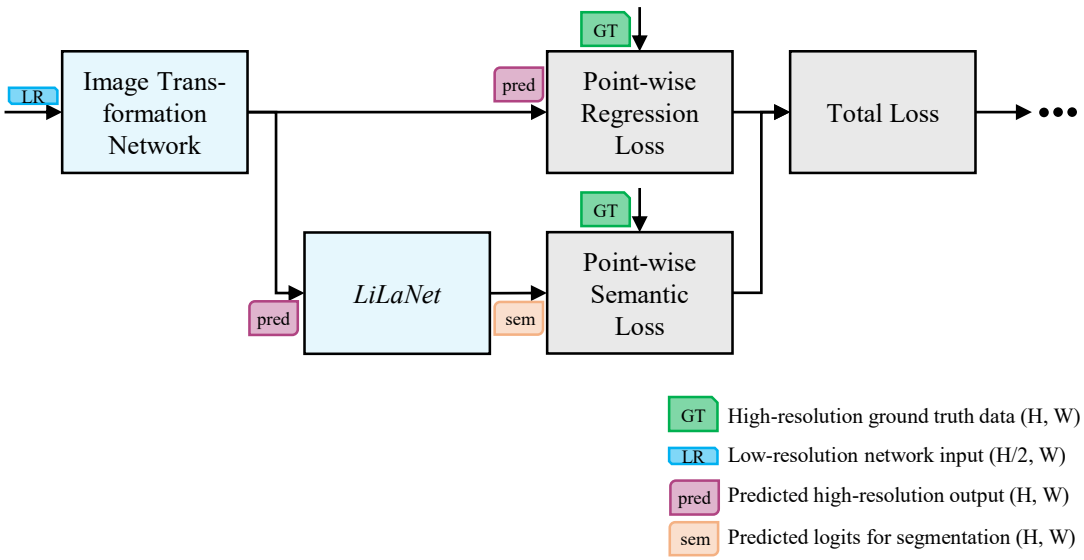


Figure 5.8: System setup of the semantically guided up-sampling network. It is a combination of the sparsity aware and the feature reconstruction network.

6 Experiment Results

This chapter presents the results of the experiments presented in chapter 5 in the same order. Section 6.5 presents two additional network evaluations and the last section gives a short summary over all the presented methods and conducted experiments.

6.1 Regular Up-Sampling Network

The system architecture that produced the following results is presented in section 5.2. Four variants have been tested.

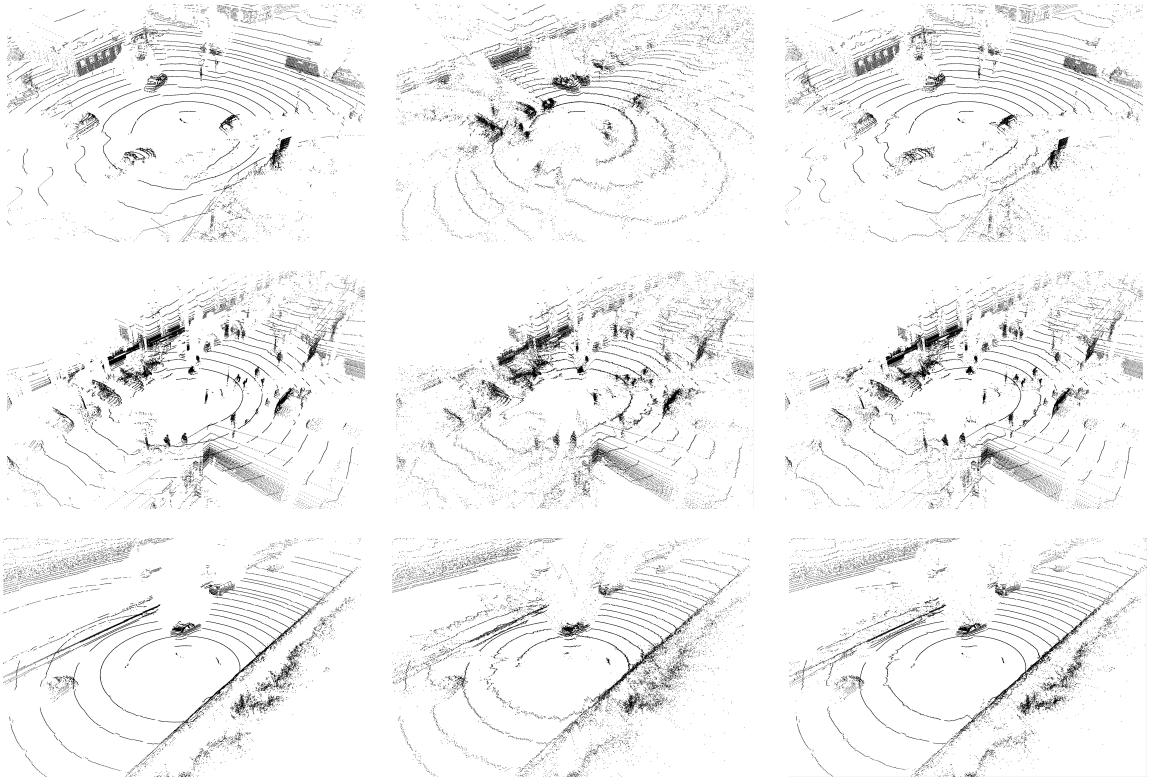
Table 6.1 lists the MSE and MAE for the evaluation set. Trainings with L_1- and L_2-loss have been conducted, as well as trainings with only the distance channel or additional invalid mask as input. There is no difference between the two version of input channels, which leads to the assumption, that the invalid mask does not provide any useful additional information which the network can understand and benefit from. Furthermore, when only regarding the MSE, the network trained with L_2-loss performs better. On the other hand, if only the MAE is considered, the training with L_1-loss achieves a lower error. This is reasonable, as L_1-loss minimizes the MAE and L_2 MSE. Therefore it is not possible to assess the quality of the up-sampling only by the numerical output value, it is rather necessary to reflect upon the qualitative point cloud output.

Table 6.1: Evaluation results regular up-sampling network (d: distance channel, i: invalid mask)

loss function	input channels	MSE	MAE
L_1	d	24.8	0.79
	d, i	24.9	**0.79**
L_2	d	21.0	0.97
	d, i	**20.9**	0.97

Figure 6.1 shows the point clouds of three scenes (top to bottom). The left column shows the ground truth image and the middle and right one, predictions of a network trained with L_2- and L_1-loss, respectively. The input channels are distance and invalid mask. It can be

observed, that the L_2-trained network generates a lot more noise than the one trained with L_1-loss. However, also the —lf1-predicted output features some variance around edges or small objects.



ground truth L2-loss L1-loss

Figure 6.1: Example scenes of the regular up-sampling network. Three different scenes, from left to right: the high-resolution ground truth image, the prediction of a network trained with L_2-loss and L_1-loss, respectively.

Figure 6.2 depicts the distribution of the MSE and MAE for 12 of the 13 semantic classes (sky is not shown here). It can be observed that road and sidewalk have a very little error in both metrics. For the network it should be quite easy to interpolate on such flat surfaces, as there are barely any edges. The MSE of middle sized objects ranges from 15 to 50. Very thin objects like traffic signs and poles are harder to predict, because sometimes only a single point per layer hits the pole. Thus the context information is very little here. Point clouds of vegetation, like trees and bushes, are very scattered, therefore it is not surprising, that the network struggles to obtain good up-sampling results here.

Figure 6.3 shows the distance distribution of the data points for all classes. A surprising result from fig. 6.2 showed that even difficult classes, like bicycle and person, only have a very low error. However, it is important to notice that some smaller objects are only detected by the LiDAR sensor, in the very near surrounding. As for bicycle, only 0.33% of the total amount of points belong to a bicycle and those points occur in a maximum range of 10 meters. The

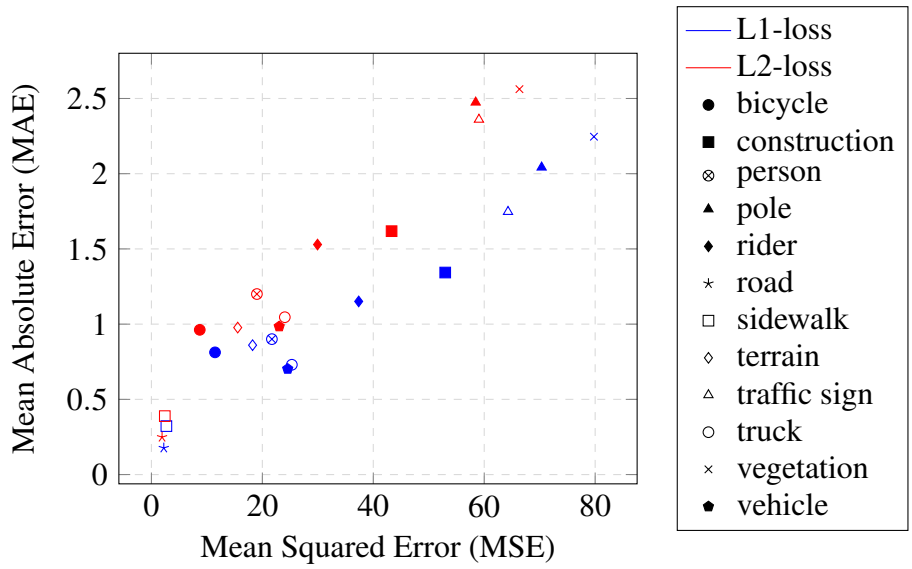


Figure 6.2: Evaluation results per semantic class. All blue markers belong to a network trained with L1 loss (therefore the MAE value is better) and all red markers result from a training with L2 loss (therefore the MSE is lower). The different markers show 12 of the 13 semantic classes. For better visualization sky is not shown here, its MSE and MAE are around 320 and 6.5, respectively. Naturally, all sky points are invalid.

farer away the object, the bigger the error. Therefore, those complicated classes have this low error compared to easier classes, like truck, which appear up to a distance of 30 meters.

The obtained results show that for point cloud up-sampling L_1-loss is the better choice to generate sharper edges in the data output. The MSE was reduced to 24.9, which equals a factor of 3.5 in comparison to the bilinear up-sampling with 87.5. A reduction factor of 2.8 was achieved for the MAE (from 2.25 to 0.79). Therefore, all the following experiments use L_1-loss instead of L_2.

6.2 Sparsity aware Up-Sampling Network

The sparsity aware network additionally predicts the invalid mask of the up-sampled image. Its structure is presented in section 5.3. In the first setup several different factors for the linear loss combination have been examined in order to evaluate the influence of the invalid mask on the up-sampling prediction. Afterwards, a learning loss combination was tested. A batch size of 10 was used for all experiments.

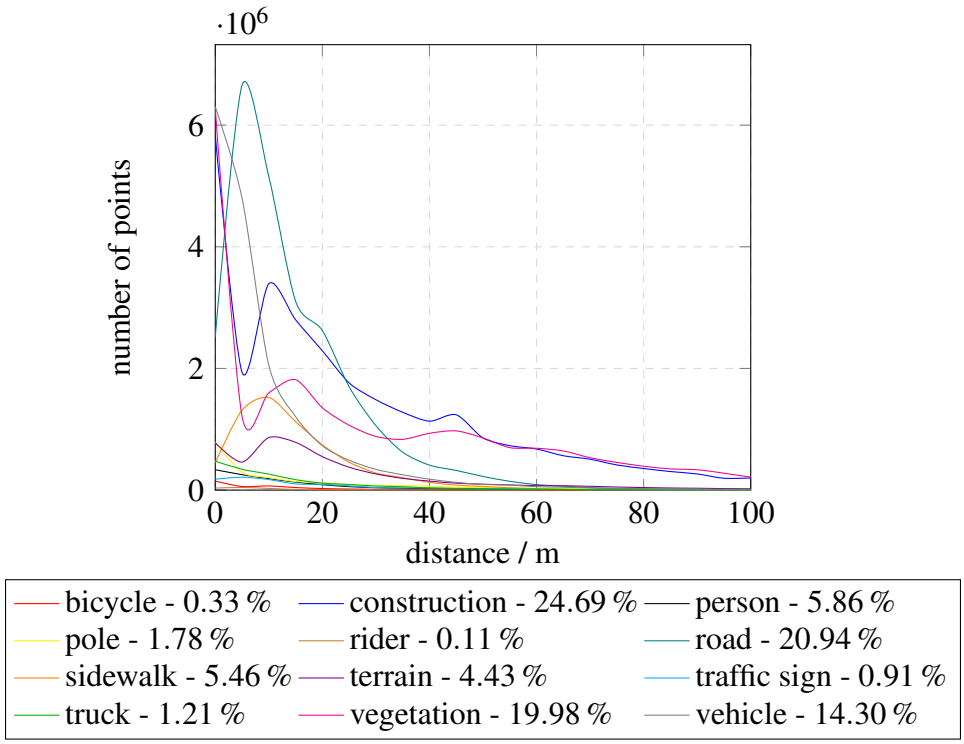


Figure 6.3: Distance distribution per semantic class

6.2.1 Linear Loss Combination

The total loss combines the regression loss of the up-sampling problem with the classification loss for the valid mask prediction in a linear manner (eq. 5.2). With a higher factor, the classification loss is weighted more than the regression loss and vice versa. It can be observed that when scaling the factor p between 0 and 1 in 10^{-1} almost no change in the performance occurs. Therefore, more factors in higher and lower magnitudes have been tested in order to see if the factor has any influence on the evaluation outcome at all. It has to be kept in mind, that the regression and classification loss themselves lie in different scales, regression around 10^{1} and classification around 10^{-1}.

Table 6.2 shows the mean IoU for the classification of valid points and the MSE and MAE for the up-sampling task. Only if the factor is very high, i.e. regression approximately weighted with zero, a difference can be spotted. In this case the error for the up-sampling rises, however the classification performance barely increases. Even if the classification loss is barely weighted (very low factor), there is no change in either the up-sampling, nor the classification task. This means, that the additional prediction of the invalid mask does not have a very big influence on the performance. However, it still helps to slightly improve the mathematical metrics. The MSE improves from 24.9 to approximately 21.1 in comparison to the regular network trained with L_1-loss (tab. 6.1). The MAE, on the other hand, increases from 0.8 to 1.0. This causes the point clouds to be a bit more noisy. However, the big advantage of this

setup is, that we obtain additional information about which points in the predicted output are valid and which are not. The mean IoU of the prediction is at about 80%. The valid points are more in number and a bit easier to predict thus obtain an IoU of about 90%, while the invalid points lie around 69%.

Table 6.2: Evaluation results at different loss combination factors

factor p (eq. 5.2)	mean IoU	MSE	MAE
	$\mathscr{L}_{\text{total}} \approx \mathscr{L}_r$		
0.000001	0.79	21.1	1.03
0.00001	0.79	21.1	1.00
0.0001	0.79	21.2	1.00
0.001	0.79	21.1	**0.99**
0.01	0.79	21.1	1.00
0.1	0.80	**21.0**	1.01
0.5	0.81	21.2	1.06
0.9	0.82	21.1	1.01
0.99	0.82	21.3	1.02
0.999	**0.83**	22.1	1.22
0.9999	**0.83**	23.6	1.33
0.99999	**0.83**	26.9	1.85
0.999999	**0.83**	33.5	2.60
	$\mathscr{L}_{\text{total}} \approx \mathscr{L}_c$		

6.2.2 Trainable Loss Weights

This section exhibits the same setup as the one before, except that here the factor is not chosen by hand, but rather learned by the network itself. The two losses are combined according to equation 5.3. The log terms are introduced to prevent the network from weighting each loss with zero. $\frac{1}{2\sigma_r}$ and $\frac{1}{\sigma_c}$ are the weighting factors for regression and classification loss, respectively.

Figure 6.4 shows how the two factors develop over the course of the training. Both are initialized with 1, in order to prevent a random initialization with negative values. σ_c slightly decreases to 0.83 and σ_r first linearly increases and then converge to around 6.5. With equation 5.3 the total loss results in

$$\mathscr{L}_{\text{total}} = 0.01 \cdot \mathscr{L}_r + 1.45 \cdot \mathscr{L}_c + \text{const} \quad . \tag{6.1}$$

After 500k training iterations, the up-sampling task reaches a MSE of 21.1 and a MAE of 0.99. The classification task achieves a mean IoU of 82%. This equals a combination of the linear

trainings with factors $p = 0.001$ and $p = 0.9$, which means the trainable loss combination is capable of learning a more intelligent loss weighting than a pure linear combination with a chosen factor.

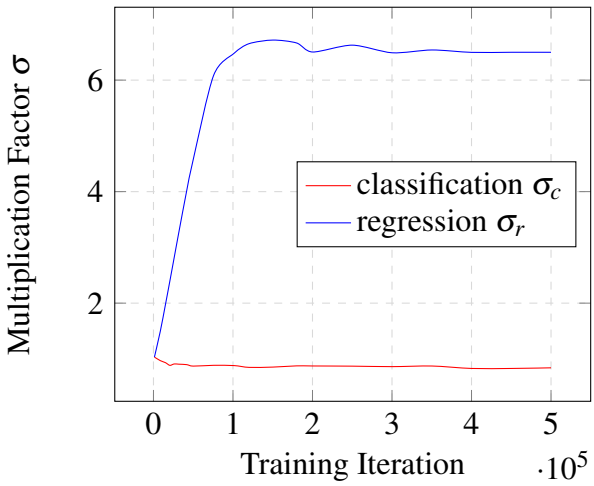


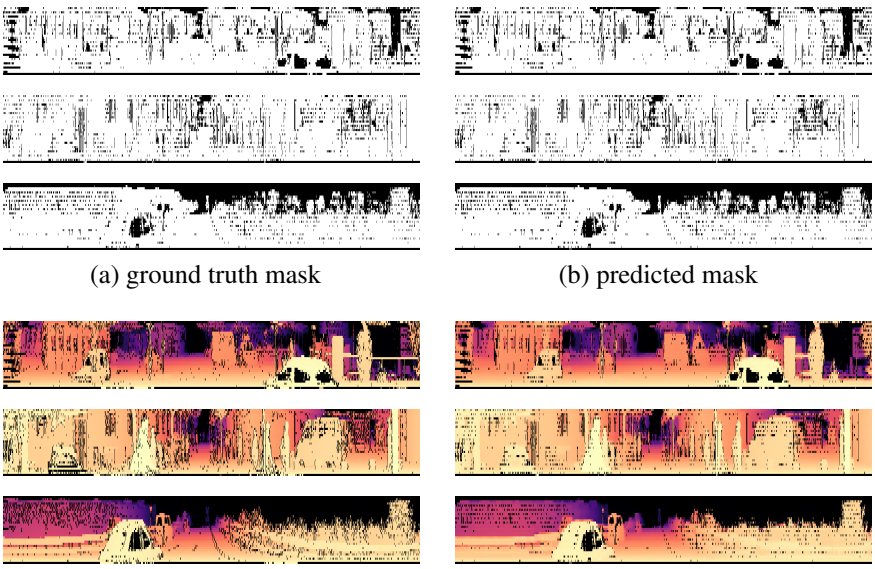
Figure 6.4: Development of the loss multiplication factors σ_r and σ_c

Figure 6.5 shows the predicted and ground truth masks and distance images of three different scenes obtained by the sparsity network. In the distance images the corresponding mask was used to draw invalid points in black. This means, opposed to previous illustrations of distance images, not the ground truth invalid mask was applied, but the predicted mask is used to mark the invalid points in black for illustration purposes. In this 2D structure with the chosen color code, the ground truth and predicted image look very much alike. The differences can only be spotted in the 3D view.

Figure 6.6 shows the corresponding 3D point clouds of the above scenes. Additionally the output of the regular network is depicted. In contrast to the 2D image, it is easy to spot, that the sparsity network yields more noise in the point clouds than the regular network. That was to be expected after observing a higher MAE. As discovered before, MAE is the more important metric when talking about those little errors than MSE which rather considers big errors. However, this lets the point cloud appear more unrealistic.

6.3 Feature Reconstruction Up-Sampling Network

The feature reconstruction network tries to fill the gap between a mathematically good solution in terms of the metrics introduced and a realistic output. However, it is necessary to have a good feature extractor in the loss network. Here, *LiLaNet* was trained and used as a feature extractor [23]. It achieved a mean IoU of 55.6% which falls behind state of the art networks. In a first setup, the original version of the network was used, which has blocks with filters



(a) ground truth mask (b) predicted mask

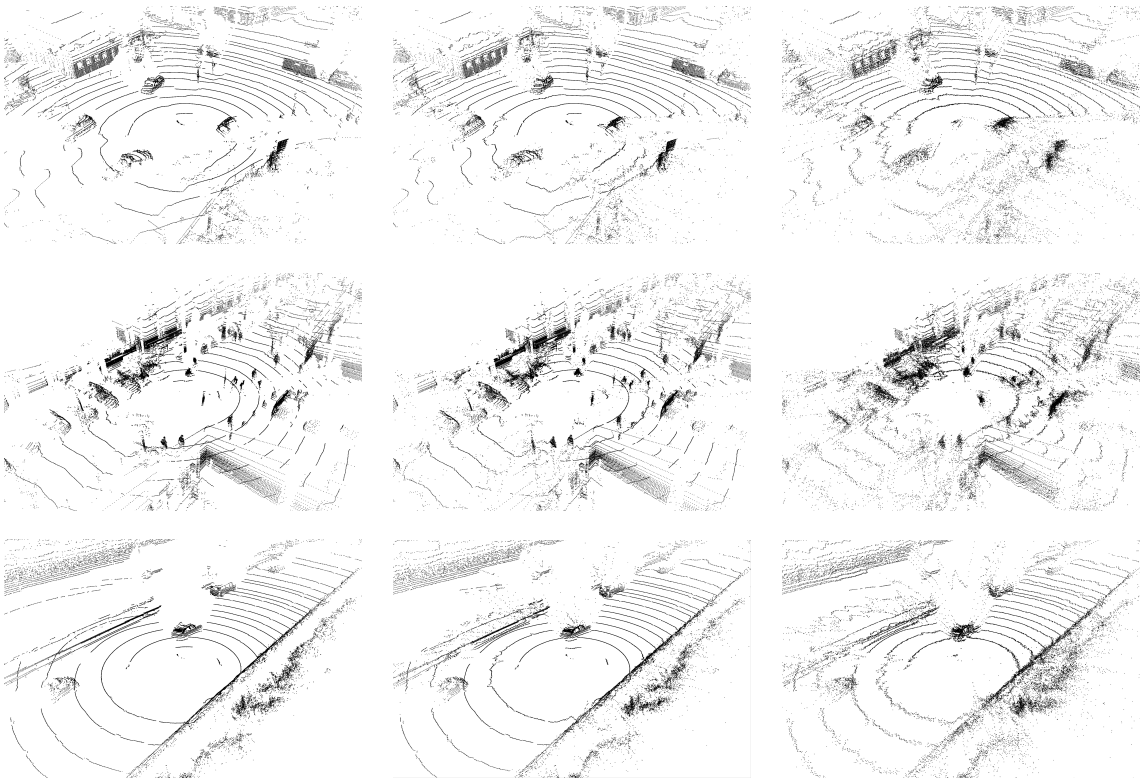
(c) ground truth distance (d) predicted distance

Figure 6.5: Comparison of the predicted point valid mask and distance image to the ground truth. White pixels in the mask mark a valid point in the point cloud, black equals an invalid point. The darker the color in the distance image, the farer away the object, the brighter, the closer. Invalid points are marked in black.

of 96, 128, 256, 256, 128, 13. The training did not result in any reasonable output, therefore one conclusion was that the feature maps are too big when 128 or 256 channels are used. It is very likely that a big amount of those channels does not represent any reasonable features. Therefore, a much smaller version of the network was used, with filters of 16, 32, 64, 64, 32, 13, as presented in section 5.1.2.

In summary ten experiments have been conducted. Half of them use the setup as it is presented in section 5.4, the other half use a pre-trained regular network from section 5.2 as an initialization for the up-sampling network. Thus this training serves as a fine-tuning step. For each of the two settings different feature maps of the *LiLaNet* have been used. They were extracted at the bottleneck layer conv(1,1) of each *LiLaBlock* 0, 1, ..., 4. It is not fair to compare the evaluation results of MSE and MAE of this training to the ones of previous trainings, as this setup optimized on the perceptual similarity between ground truth and prediction. However, it is still possible to compare the point clouds.

Figure 6.7 shows the training behavior of the feature reconstruction network over 250k iterations. The red lines illustrates the L_1-loss of the extracted feature maps of the predicted and the ground truth scan over training. The blue lines are the evaluation results of the MAE of the predicted and the ground truth distance images in the validation dataset. Solid lines represent the regular training and dashed lines the finetuning training, where a pre-trained up-sampling network is used for initialization. In all figures, it can be observed, that the uninitialized networks not only start at a higher error or loss, but also converge to a higher MAE. Especially



ground truth regular network sparsity network

Figure 6.6: Comparison of the resulting point clouds between the ground truth, the regular network and the sparsity network, which additionally predicts the invalid mask. The predicted mask is applied for the sparsity network to not show the invalid points, for the ground truth and the regular network, both the ground truth invalid mask was used.

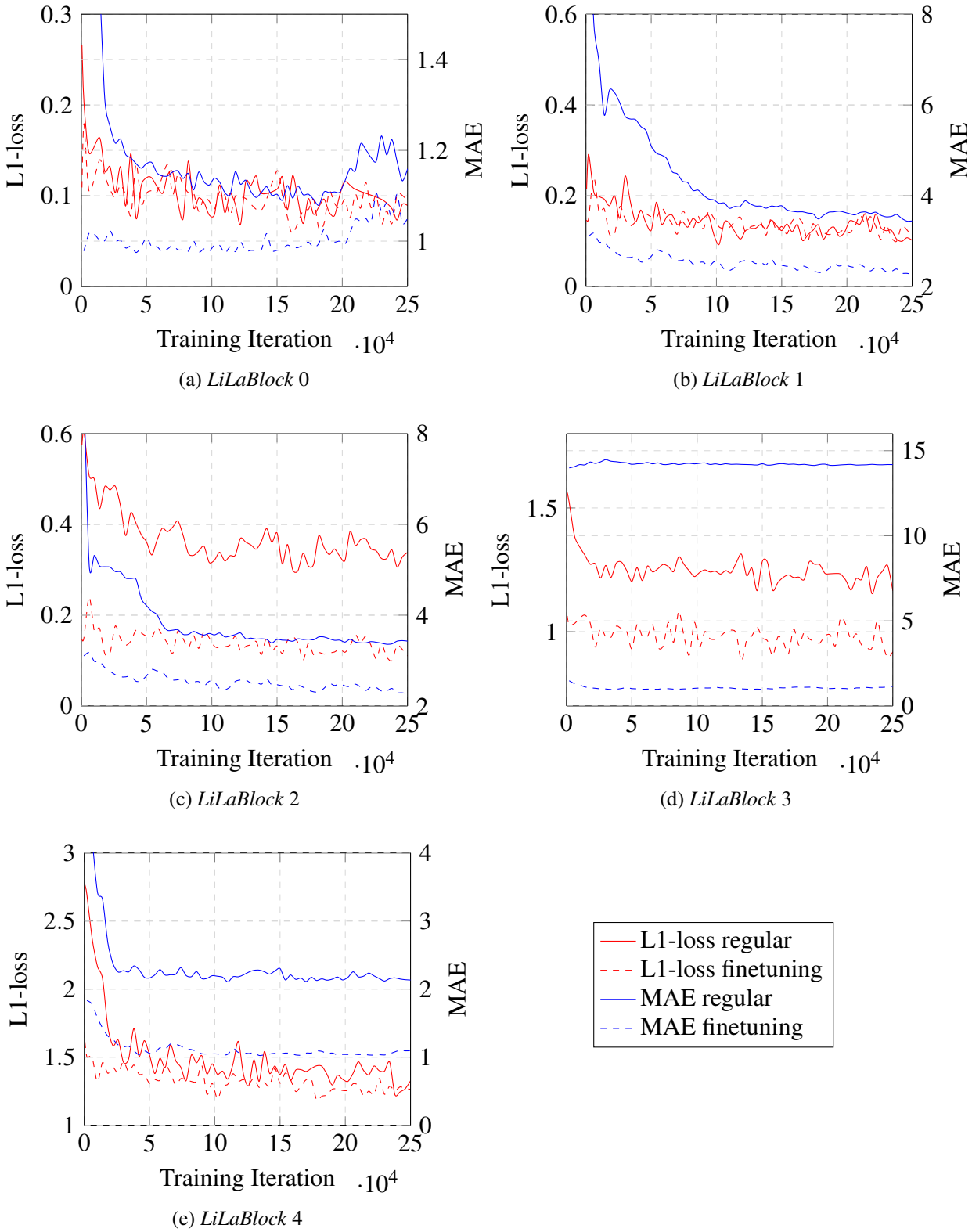
at *LiLaBlock* 3 the difference is very high, 14.17 vs. 1.17, which can also be seen in the illustration of the predicted point cloud in figure 6.8 (column two and three).

Another thing to mention, for the trainings with the feature maps extracted at *LiLaBlock* 0, the MAE of the distance images increases after 200k training iterations, however the loss is still decreasing or staying approximately the same. This means that referred to the feature maps of the loss network, it is still possible to reduce the error. This can lead to an increase of the point wise error of the distance images. However, the aim of this network is to produce perceptual similarity between the prediction and the ground truth and not to reduce the error between them, therefore this is reasonable behavior.

Figure 6.8 shows the predicted high resolution point cloud output for three perceptual networks. The leftmost column shows the output when the original version of *LiLaNet* is used. As mentioned before, it has a large number of filters which cause the feature maps to partially not represent useful information. Therefore, the predicted outputs of the intermediate layers are not reasonable. Block 0 is closer to the original structure, therefore a more realistic output

is produced. At block 4, also an output close to a point cloud is created, which is mainly because it has a much smaller amount of layers than intermediate blocks, furthermore it is close to the semantic representation of the distance image, thus can produce surfaces. However the output is still very noisy and cannot restore fine details. This experiment series is used as a reference for the other two columns in the figure. Both of them use the reduced version of the *LiLaNet*, which has less than one fourth of the filters. The middle column refers to the regular training and the right to the finetuning where the up-sampling network was initialized with a pre-trained version from section 6.1. Especially for the point clouds in the middle column, a clutter of random points around the sensor origin is formed. Furthermore, with the feature maps at *LiLaBlock* 3 it is still not possible to generate a point cloud scene. In general, the images obtained from the finetuning training show less outliers and noise compared to the uninitialized version, however it was not possible to remove them completely at object edges. The finetuning training achieves producing real point cloud scenes for all five versions.

It is not possible to say which of the five versions produces the best output. The finetuning training with feature map extracted from *LiLaBlock* 3 produces an output which has the most visual similarity to the output produced by the regular network trained with L_1-loss, illustrated in figure 6.1. It is further possible to compare the error metrics between the different versions, but as stated before, a lower error does not necessarily mean a better point cloud in terms of being realistic. For completeness, the mathematical evaluation results are included in table 6.3.



(a) *LiLaBlock* 0

(b) *LiLaBlock* 1

(c) *LiLaBlock* 2

(d) *LiLaBlock* 3

(e) *LiLaBlock* 4

Figure 6.7: Trainings of the perceptual network over 250k iterations with a batch size of 5. The red lines show the L_1-loss of the feature maps extracted from *LiLaNet* at the location specified by the corresponding caption, the blue curves depict the MAE of the up-sampled and target distance image over the complete evaluation dataset. Solid lines represent trainings where the up-sampling network is not initialized, dashed lines are trainings with a pre-trained up-sampling network, therefore called finetuning.

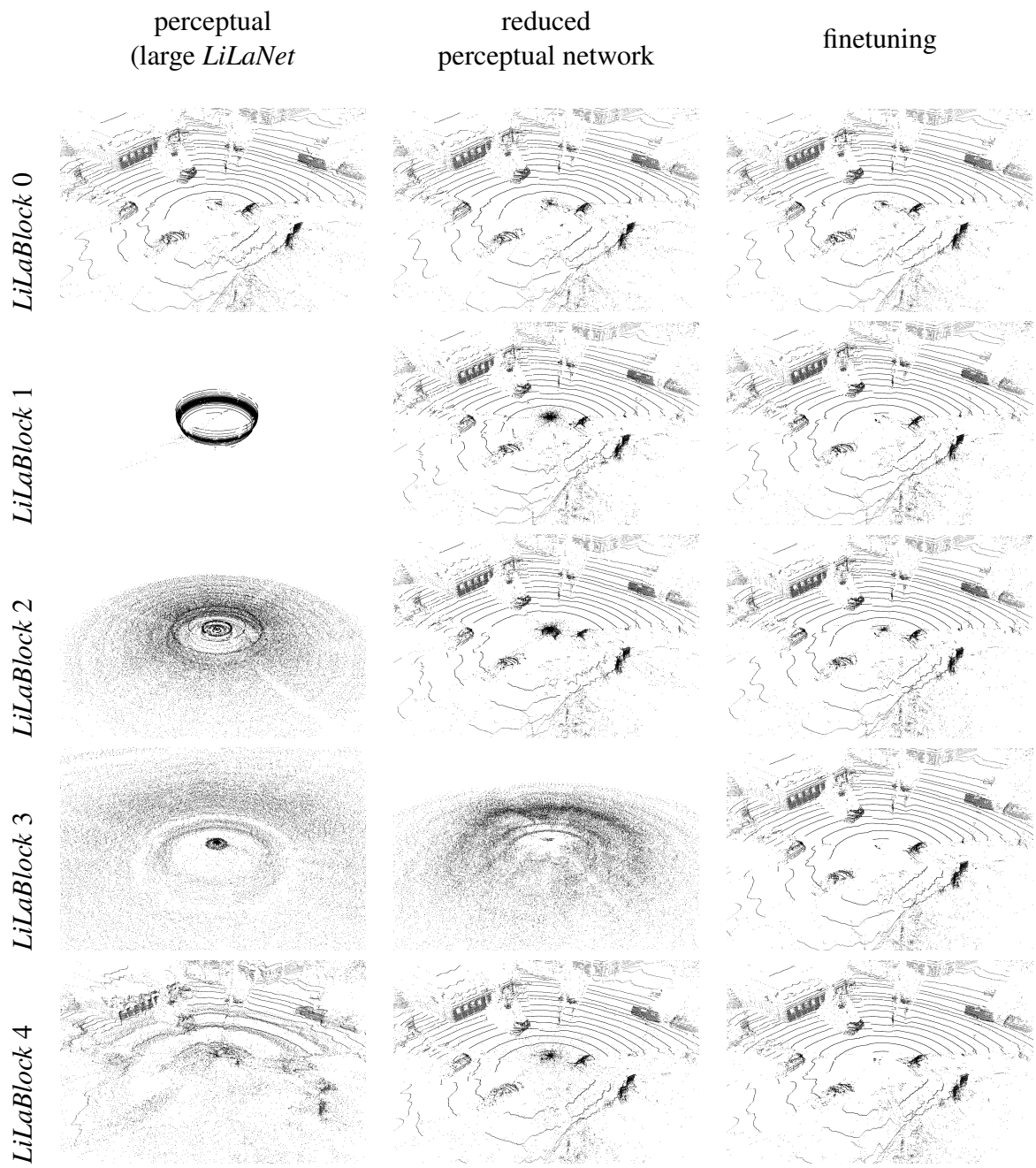


Figure 6.8: Shown are different version of one 3D LiDAR point cloud scene. The columns correspond to the three different versions of the network and the rows at which location of the loss network the feature maps were extracted. The columns represent the following settings (from left to right): the large version of *LiLaNet* as a feature extractor, the reduced version of the network trained with a non initialized up-sampling network and a finetuning training where the up-sampling network use pre-trained weights for initialization. The scene shows a road crossing with the ego vehicle headed to the top right corner of the image.

Table 6.3: Evaluation results of the three versions of the perceptual network

LiLaBlock	MSE			MAE		
	large	reduced	finetuning	large	reduced	finetuning
0	53.3	55.0	**49.6**	1.15	1.16	**1.05**
1	758.0	139.3	**50.1**	15.79	3.45	**2.27**
2	635.6	143.4	**67.5**	14.20	3.43	**2.11**
3	509.1	659.2	**30.6**	13.89	14.18	**1.15**
4	81.5	105.4	**32.4**	3.94	2.13	**1.09**

6.4 Semantically guided Up-Sampling Network

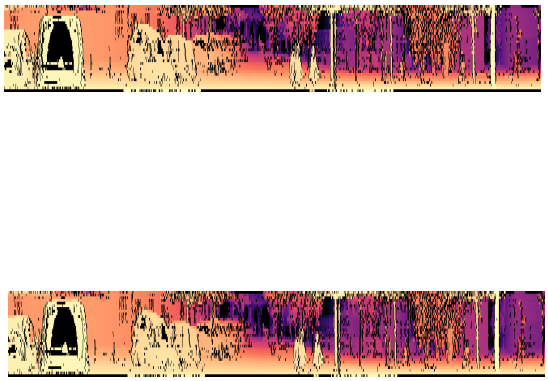
One of the applications of downstream perception algorithms is the semantic segmentation of the up-sampled point cloud. As semantic labeling is based on high-level abstractions of the single points, a semantically guided point cloud up-sampling could improve the structural correctness of the output, similar to the feature reconstruction network. Therefore an architecture, presented in section 5.5, was introduced, which consists of an up-sampling network and a downstream semantic segmentation network. On each network output a loss is computed and combined in a total loss block, which learns the weighting of the losses during training. The up-sampling network is initialized with weights obtained from the regular network trained with L1-loss from section 6.1.

Table 6.4 shows how the MSE, MAE, and mean IoU evolved from the initialization state with the regular network to the semantically guided network after 300k iterations at a batch size of 5. The additional semantic input decreased the MSE to 21.9 and improved the semantic labeling performance to 41.9%. However, the resulting point clouds are more noisy, resulting in the MAE to increase to 1.00.

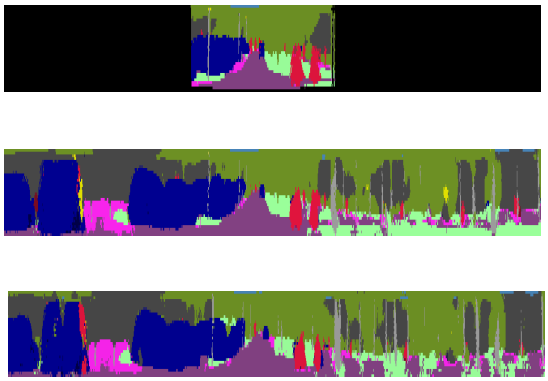
Table 6.4: Evaluation results of the semantically guided up-sampling network

iteration	MSE	MAE	mean IoU [%]
0	24.9	**0.79**	37.6
300k	**21.9**	1.00	**41.9**

Figure 6.9 shows an example of the distance and label image output of this network. Figure 6.10 shows the corresponding 3D representation of the same scene. Both the semantic prediction on the ground truth and on the up-sampled distance image are shown. It can be seen that the semantic prediction does not differ too much between those two versions, however the up-sampled point cloud is noisier than its ground truth and also than the prediction obtained with the regular network trained with L_1-loss.



(a) Distance images. Top: ground truth, bottom: high-resolution prediction.



(b) Semantic labels masks. From top to bottom: ground truth annotated labels (black is unlabeled), predicted labels on the ground truth distance image, predicted labels on the up-sampled distance image.

Figure 6.9: Output distances of the semantically guided up-sampling network (left) and semantic label images obtained by *LiLaNet*. Color coding of the distance images: the closer points are brighter, invalid points are marked in black. Color coding of the label images: each object class has its own color according to fig. 4.1, the black points in the top picture are unlabeled (no annotations obtained from the auto-labeling process, because not in the cameras field of view)

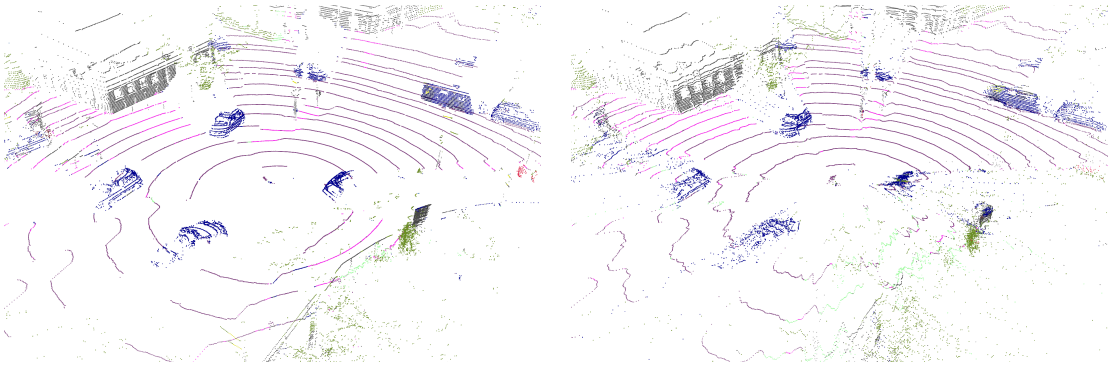
6.5 Structural Network Comparisons

This section compares the two regular trainings with L_1- and L_2-loss with one traditional up-scaling approach, namely bilinear interpolation, and the ground truth data. Two different kinds of evaluation procedures have been conducted, both aiming to asses the structural quality of the up-sampled output instead of the mathematical error. The first approach evaluates semantic segmentation on the up-sampled distance images and the second is a survey among LiDAR experts to evaluate the general quality of the resulting point clouds.

6.5.1 Semantic Segmentation Assessment

Semantic segmentation is the point wise classification of the LiDAR points into object classes, i.e. car or person. Networks trained for this usecase have to understand the structural properties of the data. Therefore, the evaluation metric for segmentation, mean IoU, allows to evaluate the up-sampled point cloud from another perspective.

Table 6.5 shows the mean IoU values of the three up-sampling versions and the ground truth. It is very reasonable that none of the up-sampling methods can achieve the same performance as the ground truth distance images. The L_2-trained regular network has the lowest performance



(a) Label prediction on the ground truth point cloud

(b) Label prediction on the up-sampled point cloud

Figure 6.10: Point clouds with semantically annotated points obtained by *LiLaNet* after generating high-resolution point clouds with the semantically guided up-sampling network.

by far. In previous example images, the point clouds generated with this L_2-loss trained network are very noisy, therefore it is not possible to extract good semantic information from the scans. The L_1-loss trained network and bilinear interpolation lie very close together. The main difference spotted in their output point clouds, was that bilinear interpolation caused single outlier points between layers and the L_1 version had difficulties when interpolating at object edges, both methods resulted in smooth surfaces. Compared to the semantically guided network, the networks perform worse in terms of mean IoU, but generate more realistic point clouds.

Table 6.5: Semantic segmentation evaluation of different up-sampling methods.

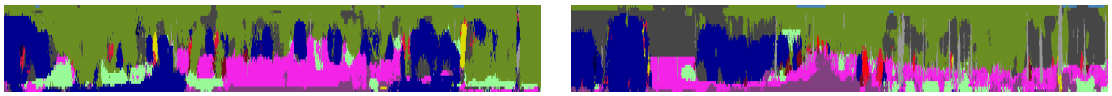
	ground truth	bilinear interpolation	regular network	
			L2-loss	L1-loss
mean IoU	55.5%	36.3%	14.7%	**37.6%**

Figure 6.11 shows the ground truth annotations and the predicted labels of all the four distance image types from table 6.5. The color code is the same as in fig. 4.1, each color represents one semantic class. It is important to compare the images to the label prediction of the ground truth distance image, the ground truth annotations are only shown as a reference and to illustrate that even on the original scan the network is not possible to label the points correctly (mean IoU of 55.5%). It is easy to spot how the bad mean IoU result of the regular network trained with L_2-loss emerged. Most of the points are labeled as vegetation and terrain. In combination with figure 6.2 and the observation that this network yields noisy point clouds, it can be said that the semantic segmentation network learned to classify all noisy structures as vegetation and therefore it is almost not able to recognize anything else than this class. One exception are vehicles, which seem to have a very explicit appearance that is not destroyed enough by

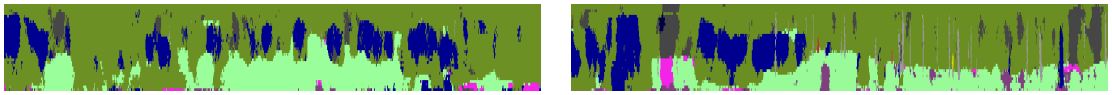
the noise. When comparing the semantic output of the bilinear interpolation and the regular network trained with L_1-loss, it can be said that both have their assets and drawbacks. For example, with the L_1-trained output, the network is better in distinguishing between road and sidewalk.

(a) Ground Truth annotations obtained in the autolabeling process [23]

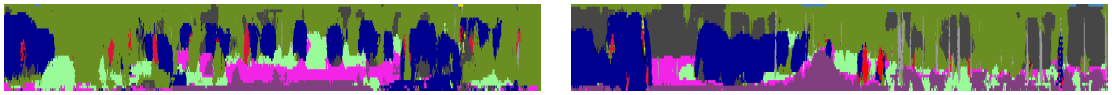
(b) Predicted labels on the ground truth distance image



(c) Predicted labels on the bilinear interpolated distance image



(d) Predicted labels on the distance image up-sampled with the regular L2-loss trained network



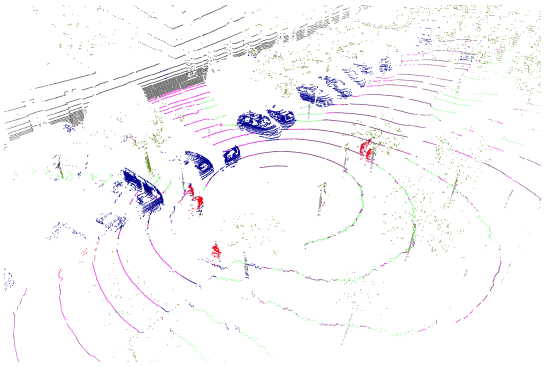
(e) Predicted labels on the distance image up-sampled with the regular L1-loss trained network

Figure 6.11: Semantic segmentation on the output of different up-sampling networks. Each color represents one object class, e.g. purple - road, blue - vehicle, green - vegetation. The color code is the same as in fig. 4.1. The ground truth annotations are only available for the part of the scan which was seen by the camera, the other points are marked in black and are excluded from the evaluation.

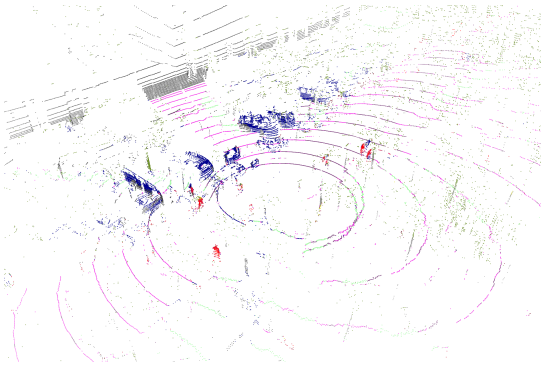
Figure 6.12 shows the semantically labeled point clouds of the scene on the right in figure 6.11.

6.5.2 Mean Opinion Score Testing

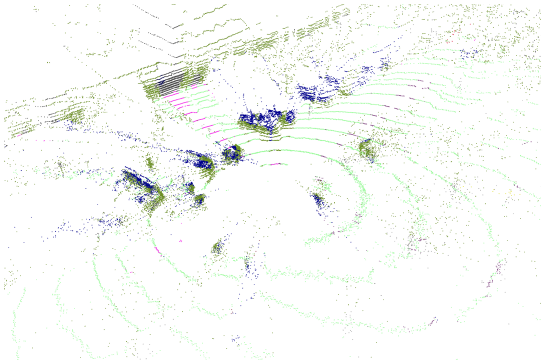
Additionally to the statistical evaluations, a point cloud survey has been conducted. 19 human raters have been asked to assign a score from 1 (bad quality) to 5 (excellent quality) to the up-sampled point clouds. The persons rated 4 versions of 9 different scenes, all illustrated as a 3D point cloud projection to a 2D image. The samples were presented in a randomized fashion without any indication of the applied up-sampling approach.



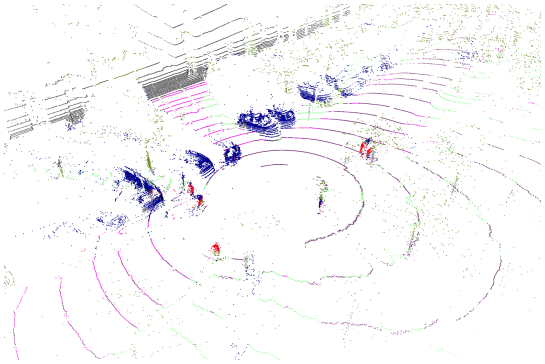
(a) Predicted semantic labels on the ground truth point cloud



(b) Predicted semantic labels on the bilinear interpolated point cloud



(c) Predicted semantic labels on the point cloud up-sampled with the regular L_2-loss trained network



(d) Predicted semantic labels on the point cloud up-sampled with the regular L_1-loss trained network

Figure 6.12: Example scene of a 3D point cloud obtained by different up-sampling methods with semantic labels

Figure 6.13 shows the distribution of the opinion score ratings for each approach. The darker the color, the more raters assigned the corresponding score to the samples of the belonging approach. In summary 162 samples per method have been assessed (9 images times 18 raters), the darkest color refers to 55% of those samples, i.e. 89 votes. The red markers show the mean. It is not surprising that the ground truth samples achieved the best results, however not all of its samples have been rated with 5 points. Some scenes are easier or harder to evaluate for humans due to the chosen perspective of the projected camera and the contents of the scene (highway scene vs. rural).

The regular network trained with L_1-loss achieved the best score of the up-sampling approaches with 3.26. The network trained with L_2-loss obtained worse ratings than the traditional approach of bilinear interpolation. Those results are similar to the ones obtained by the semantic evaluation. In both investigations, the regular network trained with L_2-loss performed worst. For both human and machine single outliers seem to have a less negative impact than a noise over the complete point cloud.

Figure 6.13: Color coded distribution of the point cloud survey opinion scores. For each category or method (left) 162 samples (9 images times 18 raters) have been assessed. The color coding from 0% to 55% is referred to the amount of samples (darkest color at 55% of 162 samples equals 89 votes for the corresponding score). The mean is illustrated by the red marker.

6.6 Summary

Table 6.6 summarizes the performances of the most important networks of all presented system setups. In this work five categories of up-sampling networks with several versions have been designed, tested and evaluated. The bilinear interpolation of the traditional methods served as a baseline. The regular residual network showed major improvement over this approach, the one trained with L_2-loss decreased the MAE from 2.25 to 0.97 and the network trained with L_1-loss even achieved a MAE of 0.79, which equals an improvement of almost 65%. Furthermore, the L_1-loss trained network achieved a better mean opinion score and a slightly better mean IoU for semantic segmentation. It also features less single outliers in the point cloud than the traditional approach.

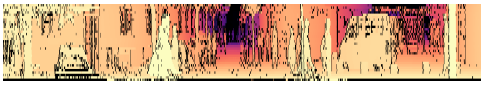
The other three approaches did not perform as well as the regular network in terms of MSE and MAE, which was also not the aim of these trainings. The sparsity network is able to additionally predict the invalid point mask of the up-sampled point cloud and still performs considerably better than the traditional algorithm. It is not fair to compare the metric values of the feature reconstruction network to the previous ones, as this networks focus is on restoring high-level features and not the exact point value. Therefore, it is more complicated to assess the quality of the output point clouds and compare it to other methods. In general it has to be said, that there is still improvement for this kind of architecture, especially concerning the feature extractor. The semantically guided network behaves very similar. Furthermore, its focus was on optimizing the up-sampling for subsequent semantic segmentation, which is an aspect not considered in any of the previous networks. After all this causes the network to obtain the highest mean IoU score of all networks in this work, but have substantial loss in the quality of the output point cloud.

Figure 6.14 and 6.15 show one example scene for each of the methods in table 6.6 and the ground truth. It is easier to see the differences in the 3D point cloud projection than in the distance image. The regular network trained with L_2-loss and the sparsity network are more

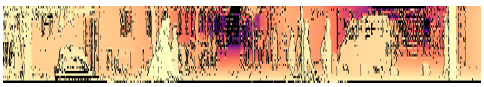
Table 6.6: Overview on the evaluation results of some presented networks

	MSE (smaller is better)	MAE (smaller is better)	mean IoU [%] (GT: 55.47%)	opinion score (1 - bad, 5 - excellent)
Bilinear Interpolation	87.5	2.25	36.25	2.49
Regular Network (L2-Loss)	**20.9**	0.97	14.70	1.63
Regular Network (L1-Loss)	24.9	**0.79**	37.59	**3.26**
Sparsity Network (trainable weights)	21.1	0.99	-	-
Feature Reconstruction Network (*LiLaBlock* 3 finetuning)	30.6	1.15	-	-
Semantically guided Network	21.9	1.00	**41.92**	-

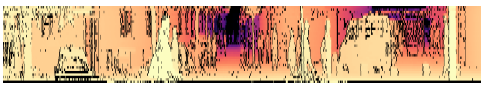
noisy than the other approaches. The regular network trained with L_1-loss and the feature reconstruction network yield smoother surfaces and reduce the number of single outliers, which are common in the method of bilinear interpolation. Furthermore, the feature reconstruction network generates more realistic point clouds than the semantically guided network.



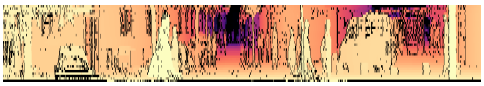
(a) Ground Truth



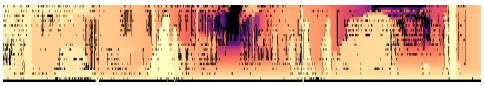
(b) Bilinear Interpolation



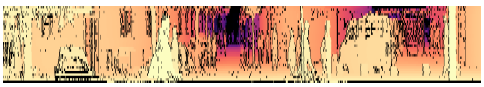
(c) Regular Network trained with L_2-loss



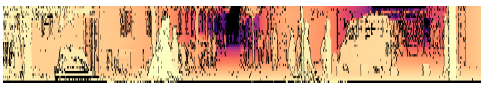
(d) Regular Network trained with L_1-loss



(e) Sparsity aware Network with trainable loss weights

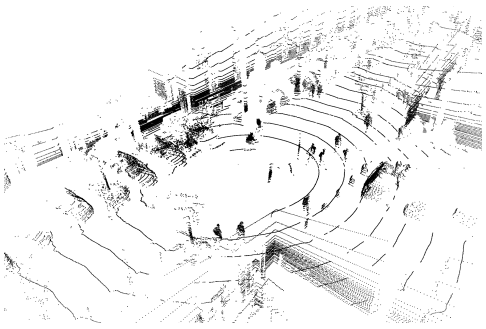


(f) Feature Reconstruction Network with feature maps extracted at *LiLaBlock* 3 (fine-tuning version)

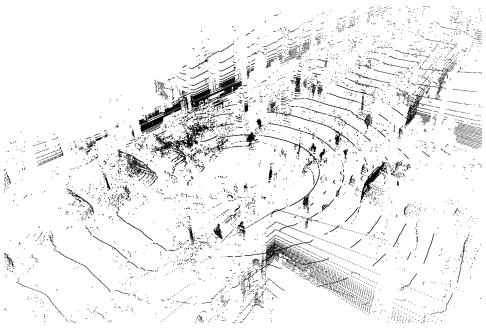


(g) Semantically guided Network

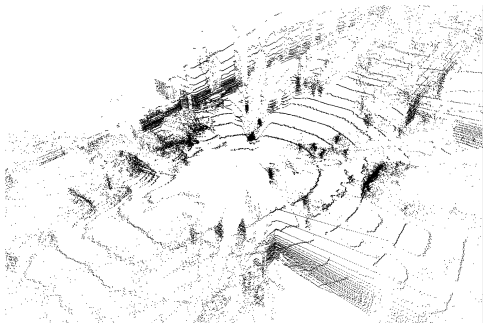
Figure 6.14: Examples of distance images obtained with several up-sampling approaches. Bright means close, dark is far. Invalid points illustrated as black.



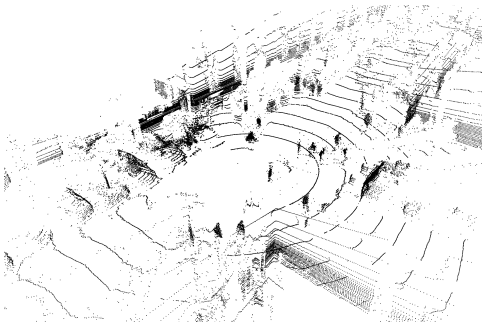
(a) Ground Truth



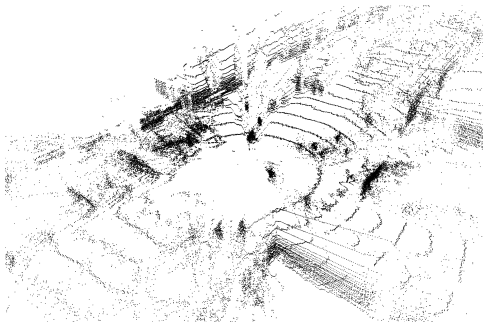
(b) Bilinear Interpolation



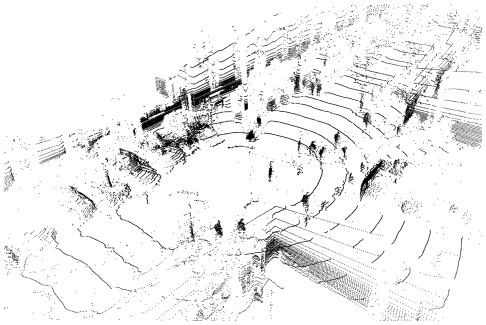
(c) Regular Network trained with L_2-loss



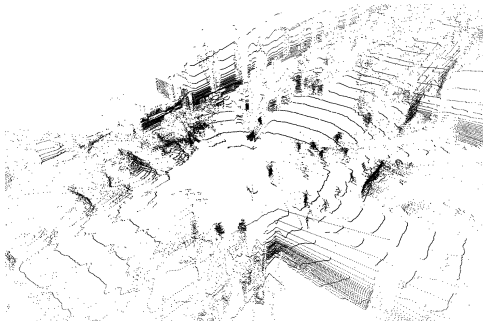
(d) Regular Network trained with L_1-loss



(e) Sparsity aware Network with trainable loss weights



(f) Feature Reconstruction Network with feature maps extracted at *LiLaBlock* 3 (fine-tuning version)



(g) Semantically guided Network

Figure 6.15: Examples of a 3D point cloud scene obtained with several up-sampling methods. The ego vehicle is headed to the top right corner of the images. Shown is a complex scene at a road crossing with trees, vehicles, persons and bicycles.

7 Conclusion

This work introduced four main network architectures for point cloud up-sampling and applied three independent evaluation procedures. The main challenges of up-sampling the distance images of the point clouds are the effects caused by the invalid points and the generation of realistic outputs. It showed, that a network trained with L_1-loss is able to generate smoother and more realistic point clouds than the same network trained with L_2-loss. The regular L1-trained network achieved an improvement of the mean squared error by 72% and the mean absolute error by 65% compared to the traditional approach with bilinear interpolation. In order to estimate the location of the invalid points in the up-sampled output, an extended network architecture was developed which outputs a mask stating whether a point is valid or not. The network achieved a slightly better mean squared error compared to the regular network, but resulted in a worse mean absolute error and a noisier point cloud prediction.

It is a common phenomenon that outputs from networks trained with a hand-designed loss function are good and valid in terms of their statistical properties, but lack the characteristic of being realistic enough. This can also be observed in the outcome of the opinion score testing, where none of the networks was able to achieve a value as high as the ground truth point clouds. The regular network trained with L_1-loss achieved a better value than the traditional approach, but the L_2-trained version reached a very low value and also performed poorly on the semantic segmentation assessment. Another network architecture was introduced to optimize the structural properties of the point cloud instead of minimizing the mathematical error. This perceptual network is able to generate point clouds which appear very similar to the ones generated by the regular network, even though having a much larger mean squared error and mean absolute error. With a more powerful feature extractor it can be possible to generate more realistic outputs.

All introduced networks are able to output a reasonable point cloud. One major challenge is that the network is not aware of the layer distribution within the sensor. This means, for the network it appears as if all rows in the 2D structure are uniformly distributed, however this is not the case as layer angles vary. If the additional information of the layer distribution or the 3D coordinate of the point is inserted into the network, this could improve the performance. Another aspect is the choice of the loss function. A distance weighted or object class weighted loss function could also help to decrease the error. For example in order to punish large errors in a far away distance less than the same error in the near field and to decrease the influence of the class vegetation, which is very noisy and increase the weighting of important objects, such as persons.

Recently, generative models have proven to be able to generate realistic data by transforming from one domain to the other. Applied to this work, it means to transfer from a low resolution point cloud to a high resolution point cloud. It is possible to apply a GAN to solve the LiDAR super-resolution task and thus address the challenge of generating truly realistic high-resolution point clouds. Nevertheless, the networks presented in this work have proven major improvement over traditional approaches and applicability to downstream perception algorithms, like semantic segmentation, thus can be used as a pre-processing step in the offline training of neural networks.

List of Figures

List of Tables

Bibliography

[1] B. S. Morse and D. Schwartzwald, "Image magnification using level-set reconstruction," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, Dec 2001, pp. I–I. DOI: 10.1109/CVPR.2001.990494 (p. 3)

[2] R. Fattal, "Upsampling via imposed edges statistics," *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, vol. 26, no. 3, p. to appear, 2007. (p. 3)

[3] Q. Shan, Z. Li, J. Jia, and C.-K. Tang, "Fast image/video upsampling," *ACM Transactions on Graphics (SIGGRAPH ASIA)*, 2008. (p. 3)

[4] Q. Zhou, S. Chen, J. Liu, and X. Tang, "Edge-preserving single image super-resolution," in *Proceedings of the 19th ACM International Conference on Multimedia*, ser. MM '11. New York, NY, USA: ACM, 2011, pp. 1037–1040. DOI: 10.1145/2072298.2071932 (p. 3)

[5] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller, *Efficient BackProp*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 9–50. (p. 4), (p. 14)

[6] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 2, pp. 295–307, Feb 2016. DOI: 10.1109/TPAMI.2015.2439281 (p. 4), (p. 5), (p. 11)

[7] R. Dahl, M. Norouzi, and J. Shlens, "Pixel recursive super resolution," in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017, pp. 5449–5458. DOI: 10.1109/ICCV.2017.581 (p. 4)

[8] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *ECCV*, 2016. (p. 4), (p. 24), (p. 25), (p. 41)

[9] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4681–4690. (p. 5)

[10] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng, "Pu-net: Point cloud upsampling network," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. (p. 5), (p. 11)

[11] Velodyne LiDAR, "PUCK VLP-16," [Online; accessed May 28, 2018]. [Online]. Available: http://velodynelidar.com/vlp-16.html (p. 7)

[12] Wikipedia, the free encyclopedia, "Comparison of nearest-neighbour, linear, cubic, bilinear and bicubic interpolation methods," 2016, [Online; accessed June 11, 2018]. [Online]. Available: https://commons.wikimedia.org/wiki/File:Comparison_of_1D_and_2D_interpolation.svg (p. 9), (p. 10)

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680. [Online]. Available: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf (p. 11), (p. 25)

[14] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: a tutorial," *Computer*, vol. 29, no. 3, pp. 31–44, Mar 1996. DOI: 10.1109/2.485891 (p. 12)

[15] D. Kriesel, *A Brief Introduction to Neural Networks*, 2007. [Online]. Available: http://www.dkriesel.com (p. 12)

[16] CS231n, "Convolutional neural networks for visual recognition," [Online; accessed June, 04 2018]. [Online]. Available: http://cs231n.github.io/convolutional-networks/ (p. 12), (p. 16)

[17] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," 12 2014. (p. 15)

[18] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *ArXiv e-prints*, mar 2016. (p. 16), (p. 18)

[19] M. Mathieu, C. Couprie, and Y. Lecun, "Deep multi-scale video prediction beyond mean square error," 11 2015. (p. 22)

[20] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. (p. 24)

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 09 2014. (p. 24)

[22] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, Dec 2015. DOI: 10.1007/s11263-015-0816-y (p. 24)

[23] F. Piewak, P. Pinggera, M. Schäfer, D. Peter, B. Schwarz, N. Schneider, D. Pfeiffer, M. Enzweiler, and M. Zöllner, "Boosting lidar-based semantic labeling by cross-modal training data generation," 04 2018. (p. 25), (p. 28), (p. 42), (p. 53), (p. 62)

[24] skymind, "A beginner's guide to generative adversarial networks (gans)," [Online; accessed September, 24 2018]. [Online]. Available: https://skymind.ai/wiki/generative-adversarial-network-gan (p. 26)

[25] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/ (p. 27)

[26] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. (p. 28)

[27] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. (p. 45)