THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

Recurrent sparsity-invariant convolution for depth completion

by

Etienne GAUTIER

School of Information Technology and Electrical Engineering,

The University of Queensland.

Submitted for the degree of

Master of Engineering Science

in the division of Software Engineering

June 2019

to my family

Etienne GAUTIER
e.gautier@uqconnect.edu.au

June 10, 2019
Prof Michael Brünig
Head of School
School of Information Technology and Electrical Engineering
The University of Queensland
St Lucia, QLD 4072

Dear Professor Brünig,
In accordance with the requirements of the Degree of Master of Software Engineering in the School of Information Technology and Electrical Engineering, I submit the following thesis entitled

"Recurrent sparsity-invariant convolution for depth completion"

The thesis was performed under the supervision of Dr Anthony Ngoc Nguyen. I declare that the work submitted in the thesis is my own, except as acknowledged in the text and footnotes, and that it has not previously been submitted for a degree at the University of Queensland or any other institution.

Yours sincerely,

Etienne GAUTIER

Abstract

In this paper we consider neural networks operating on a sequence of sparse data with the application of depth completion from sparse laser depth maps produced by LIDARs. Classic convolutional Networks have been proven to perform poorly on sparse data but a new variant of convolution, sparse convolution has given state of the art results. We propose a new recurrent network architecture using sparse convolution to perform depth completion on a sequence of sparse LIDAR data. In this approach, we make no assumption regarding the types of movement in the scene. We demonstrate the proposed approach performs well on the SYNTHIA dataset. We believe this technique can improve the resolution of LIDARs and can also be used for better compression of depth maps for diagnostics purposes on autonomous cars.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

According to the World Health Organisation, road traffic injuries caused an estimated 1.35M death worldwide in 2016 [1]. In over 90% of road traffic accidents, human factors (improper lookout, excessive speed, inattention) cause the accident according to Treat et al. [2].

Hence, mass-market adoption of reliable autonomous cars can potentially reduce the number of road traffic fatalities by an order of magnitude and save an estimated 1.2M lives every year.

The DARPA Grand challenge, one of the first major autonomous driving competition kickstarted research in the area. Although no team finished the course, the promising results of Red Team's LIDAR guided vehicle demonstrated the usefulness of precise depth sensing.

More advanced 360° field of view sensors have now been developed but the vertical resolution - determined by the number of lasers - represents a major cost.

Therefore, accurate depth completion algorithms can significantly reduce the cost of autonomous vehicles and speed up mass market adoption.

Furthermore, data collection is an important part of Machine Learning for Autonomous driving. However, rare events such as road works, accidents occurring around or involving the autonomous vehicle are hard to capture. Fleets of autonomous cars need to constantly record the environment around them, generating large amounts of data that is then used for training and diagnostics. Efficient compression of LIDAR data would reduce the cost

associated with storage and transfer of the collected data.

1.2 Problem description

LIDARs are an active sensor that uses laser beams time of flight to measure the distance of an obstacle in a given direction. In the context of autonomous driving, LIDARs are often rotating around the vertical axis, they have a wide horizontal field of view (up to 360°) but a narrow vertical field of view (20° to 40°).
At each frame, the LIDAR outputs the intensity of the light received and distance of the

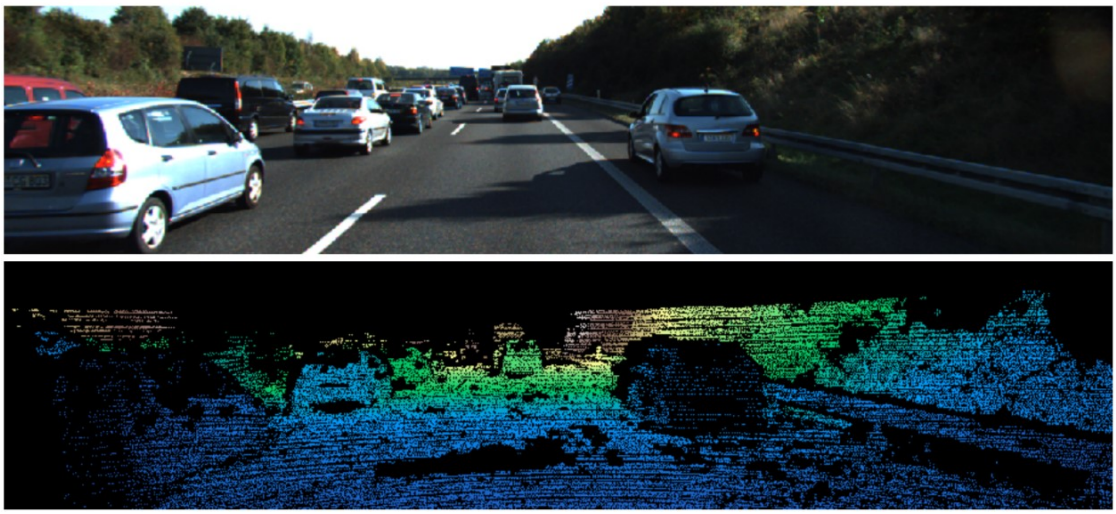


Figure 1.1: RGB image and the associated LIDAR scan of a scene

object in all of the directions measured. However, the lack of obstacle (sky) and some reflective surfaces (glass) will not return enough light to validate some measurements. Low signal intensity is a sign of non-measurable (sky) or unreliable (reflective surface) measurement. Therefore, when the measured signal is below a given intensity threshold, the distance measurement needs to be "dropped".
Once unreliable measurements have been dropped, the distance measured by a LIDAR is very precise, typically to the centimeter. The resulting data is sparse because of:

- The sampling strategy
- The bearings that have been sampled but whose measurement have been "dropped" due to the low signal intensity returned.

Our goal is to perform a type of infilling called depth completion; that is to estimate the range of the obstacle for a bearing that has not been measured. When performing depth

completion for all bearings represented in a pixel grid, we produce a dense depth map. This technique can also be used to recover only some of the missing pixels, for example, the output of a sensor with 32 lasers can be infilled to estimate the output of a more expensive LIDAR with 64 lasers.

1.3 Aims and objectives

This thesis investigates the use of deep learning for sequence-based depth completion. We will draw from progress made on Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) to design a Neural Network architecture that can exploit spatial information of neighbouring pixels from a sparse input as well as temporal information from neighbouring frames.
Our goal is to show that a recurrent approach to depth completion works well and can outperform single-frame techniques. We will study two use cases:

- Depth completion to augment sensor resolution.
- Depth completion for decompression of a LIDAR depth map.

1.4 Thesis organization

This paper is organised as follows:

- Chapter 2 presents the previous publications in the fields of RGB image super-resolution as well as depth completion
- Chapter 3 gives the theoretical background in Convolutional Neural Networks and Recurrent Neural Networks and then introduces new concepts to create a recurrent sparse convolution.
- Chapter 4 presents our experimental setup including dataset, error metric and pre-processing.
- Chapter 5 details our experimental results and presents an analysis of performance.
- Chapter 6 concludes this thesis by summarizing the work achieved in this paper and listing possible improvements to this technique.

Chapter 2

Previous work

2.1 Intensity image super-resolution

Depth completion is closely related to the image processing problem of super-resolution. Super-resolution is the process of transforming a low-resolution input image (or images) to an image with a higher spatial resolution. There has been a large amount of research done on super-resolution for grayscale and color images so it is interesting to study this problem and bring new ideas to depth completion.

2.1.1 Early techniques

The most basic approach is to perform interpolation between pixels to fill in the missing values. The basic principle of interpolation is to find a function that closely matches the data and then compute the value of that function for non-measured locations. In our case, the function maps from a location on a 2D plane to an intensity value. Different interpolation techniques have been applied to solve this problem, mostly polynomial and piece-wise polynomial. The most basic is to use bi-linear interpolation, but more complex interpolation techniques such as bi-cubic and bi-spline interpolation have shown better results. Interpolation and especially bi-cubic interpolation is often used as a baseline.
One of the first specialized approaches was developed by Tsai et al. [3], they showed that by combining grayscale in the frequency domain using Fourier transform they could produce a higher resolution image. Their approach required motion compensation using a projective warp or dense optical flow fields to align the input frames.

2.1.2 Model-based super-resolution

Irani et al. [4] formalized super-resolution by describing it through the reconstruction constraint: the super-resolved image must - when sub-sampled - produce the low-resolution inputs. There are many solutions to this inverse problem, in order to choose one a prior is needed to rate the quality of each possible solution. [5] et al. used a smoothness prior which reduced noise but led to overly-smooth super-resolved images.
Baker and Kanade [6] showed that such a prior could not restore high-frequency detail and proposed a new type of prior. Their approach is to create a specialized prior through a dictionary of similar images stored at different scales. Each image of their dataset is turned into a pyramid of multi-scale derivative features. In practice, it means that for each scale, they do not store the full copy of the image but instead the derivative or texture between low-resolution image and the resolution layer above it. To re-create the high-resolution image from the pyramid, the derivatives in each layer are added successively. In order to super-resolve an image, they process it by patch and find in the dictionary similar patches. Then they add the derivative of each patch from the dictionary weighted by the similarity of the patch to the input image to copy the texture. They were able to obtain good results with regard to restoring high-frequency detail but their method had two main drawbacks:

- It is specialized in one type of image, for example, faces. If a database of faces is used to learn the prior but a different input image is given, a pattern of human face will appear in the super-resolved output. They demonstrated this phenomenon called hallucination by choosing a uniform grey image as input and the output showed the outline of a human face.
- It is very specific to one type of image. In order to apply this method to different types of images, a completely new database is needed.

Both of those concerns are addressed by Yang et al. [7]. They randomly sampled a small number of patches from their dataset and build two dictionaries from it, one for low-resolution derivatives, the other for high-resolution derivatives. They then represent the input patch as a sparse linear combination of low-resolution derivatives from the dictionary, this is a sparse representation of the input. And apply a linear combination of high-resolution derivatives from the dictionary using the same coefficients as for the

Figure 2.1: Output of the "hallucination" algorithm by Baker and Kanade [6] when trained on a face database and used on Gaussian noise

sparse representation of the input. Finally, they modify their solution to satisfy the reconstruction constraint. These techniques lead to great progress towards a general solution applicable to most pictures. Indeed, they trained their algorithm on a dataset of flower pictures which is quite diverse in terms of lighting condition, shot angle and zoom level; and applied it to pictures of a statue. Their algorithm is however not perfectly general, they found that high-frequency patterns such as animal fur must be learned separately as they need a different low dimension basis and dictionary. Further improvements

Figure 2.2: Comparison of super-resolution algorithm. From left to right: Original image, bicubic interpolation, Yang et al [7], Zeyde et al. [8]

were developed by Zeyde et al. [8] who used Principal Component Analysis to reduce the dimension of the patches. They created their dictionary in low dimensional space

and used K-SVD to choose the patches to include in the dictionary. They were able to greatly reduce the size of the dictionary. Those two improvements led to much faster predictions and higher accuracy. Their technique is more general and does not require to train separately on different types of images.

2.1.3 Convolutional Neural Networks

Dong et al. [9] showed that the sparse coding technique of Zeyde et al. [8] was equivalent to a convolution. They performed single-image SR by framing the problem as a neural network and learned the weights through gradient descent. In their algorithm named SR-CNN, the input image was first super-resolved using bi-cubic interpolation they improved the quality of the output with the Neural Network. They used a simple architecture with 3 convolutional layers and RMSE as their loss function. Their technique outperformed the sparse-coding techniques in terms of accuracy and speed due to the small number of filters. They also showed that their algorithm can benefit from long training over large datasets.

Kappeler et al. [10] showed that CNN-based approaches also work for video super-resolution. Their recurrent network uses a concatenation layer to combine inputs (or intermediate representations in hidden layers) at different time frames. They also show how to pre-train the network on images and transfer weights to the recurrent architecture. Their approach is dedicated to a posteriori processing because it uses past frames as well as future frames to super-resolve the current frame.

They benchmarked their algorithm with top image and video super-resolution and outperformed all previous methods. It is also interesting to note that SRCNN performed 2nd best in terms of PSNR although it is not using neighboring frames but simply predicting each frame independently.

2.2 Depth map processing

2.2.1 Dense depth map super-resolution

Time of Flight cameras such as flash LIDAR send a signal in all directions at once and then capture the light returned using a lens and an array of sensors. They can produce images

at a high frame rate but images are noisy and typically have a low spatial resolution.
The first depth map super-resolution algorithms were directly applying techniques from 2D intensity image super-resolution. Schuon et al. [11] successfully used a reconstruction based technique similar to Irani et al. [4] but used a different regularization prior - bilateral filtering - to preserve edges and smooth flat surfaces. Their algorithms produced good results and more importantly showed that the reconstruction based approach from intensity image super-resolution was valid for depth data.
Lidarboost [12] improved on this method by using a different regularization prior that was better suited to curved shapes. Their regularization term is designed to minimize local gradients and greatly improves quality.
The progress made in Hardware quality and affordability with the Kinect sensors enabled new applications. Kinectfusion [13] uses motion tracking to estimate the movements of a handheld sensor within a static environment. Their algorithm uses GPU acceleration to process the data in real time. Voxels are tracked between frames and their position can be better estimated through a running average. This method is however only applicable to a rigid scene. Any non-rigid motion will cause tracking failure as shown in figure 2.3.

Figure 2.3: Kinect fusion on a non-rigid scene. From left to right: before a user comes into frame, once the user arrive he is partially reconstructed due to fast motion, non-deformable motion produces tracking failure

This shortcoming was addressed by KinectDeform by Afzal et al. [14] which supports non-rigid deformations. In KinectDeform, motion tracking is processed locally and in full 3D (scene flow), as a consequence, their algorithm supports 3D motion that is locally rigid but globally deformable. The scene flow is then applied to the 3D representation computed for the previous frame which is then fused with the current measurement. However, as

described above movements are represented as a voxel to voxel translation and rotation, this does not account for some more complex movement types such as compression and dilatation.

2.2.2 Depth CNN

As convolutional neural networks showed good results in many image processing tasks, attempts have been made to apply them to sparse inputs. One major advantage of CNN is that they do not require any assumptions on the data, and only learn from example. In the case of depth completion, it means that any type of movement is supported, as long as it is represented in the training set.

One early approach by Chen et al. [15] (for object detection from LIDAR depth map) is to replace missing values in the input by a constant. The number 0 is often chosen because it is neutral in addition. In a convolution, which is a linear combination of the input pixels using the filter weights, a value of zero would be "neutral". However this simple technique produces poor results because the value 0 is ambiguous, it could represent either a very close object or a missing value. For depth completion in the context of autonomous driving, these two values need to be handled differently.

To solve this ambiguity, the sparsity map needs to be known. The sparsity map can be represented as a matrix with the same dimensions as the input, where a value of 0 at location (i, j) means that the depth value at location (i, j) has not been measured. Similarly, a value of 1 in the sparsity map indicates that the depth value comes from a valid measurement. Kohler et al. [16] used a CNN trained with those two channels (measurement and sparsity map) to perform image inpainting. Uhrig et al. [17], showed that this technique produced better results than the naive approach of using 0 as default value. However, noise is still very visible on the output depth map.

Uhrihg et al. [17] introduced a sparsity invariant CNN that takes the sparse depth mea-

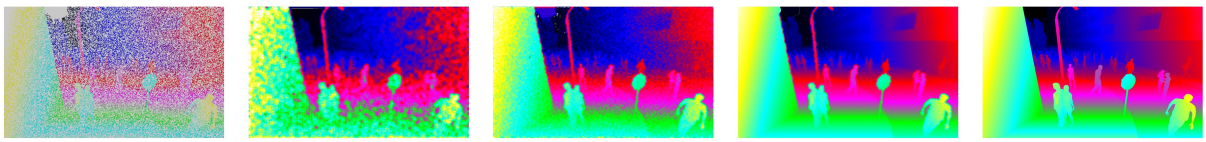


Figure 2.4: Comparison of different depth completion algorithms. From left to right: input, convolution, convolution on the depth map and sparsity map, sparse convolution, groundtruth

surement and sparsity map as input and uses the sparsity map to regularize the output of the convolution. In practice, they perform a normal convolution on the depth measurement, then use the number of valid measurements in the convolution window (computed from the sparsity map) to regularize the output of the convolution. They obtained some state of the art results in terms of root mean squared error and mean absolute error. Their algorithm also produces some visually consistent output. Furthermore, they showed that their network generalized well to new sparsity rates and to the real world KITTI dataset [17].

Chapter 3

Theory

3.1 Convolution

Convolutional networks are modeled after the human brain's visual cortex. In the visual cortex, some neurons are specialized to recognize patterns in a small portion of the visual field, other neurons react to stimuli in different locations of the visual field. Convolution in the context of an artificial neural network is a dot product between a patch in the input image and a filter containing learnable weights. This dot product is computed for many different patches across the input.

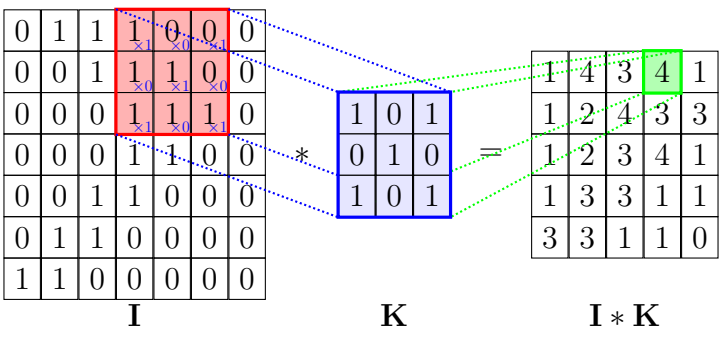


Figure 3.1: Convolution operation

The main parameters of a convolution are the filter shape, the step between each convolution and padding which determines the behavior of convolution around the input boundaries. Padding enables us to have the same shape for the input and the output. The input image can be padded (or widened) using zero-padding (added pixels will have value 0), by mirroring the input around the border or by copying the closest value from

the input image. The filter weights are usually initialized randomly and learned through gradient descent.

The output of a convolution centered at pixel (u, v) on an input x with a square filter w of size $2 \times k + 1$ is:

$$f_{u,v}(x) = \sum_{i,j=-k}^{k} x_{u+i,v+j} w_{i,j} b \tag{3.1}$$

From this formula, it is apparent that CNNs are a type of neuron that enforces local connectivity because the only non-zero weights apply to the pixels around (u, v). Another important aspect of CNNs is translation invariance: the filter is global so the output of a dot product does not directly depend on the location where the filter is applied within the image. Instead, the output of the dot product only depends on the pixel values covered by the filter kernel.

Convolution can be generalized to an arbitrary number of input channels c by using a different filter per input channel and summing the results of each 2D dot product over the channel dimension. This enables processing of multi-channel inputs such as RGB images. Let's denote $x_{u,v,l}$ the input at pixel (u, v) for the channel l and $w_{i,j,l}$ the filter weight at location (i, j) for the input channel l. With this notation, the output of a convolution is defined:

$$f_{u,v}(x) = \sum_{l=1}^{c} \sum_{i,j=-k}^{k} x_{u+i,v+j,l} w_{i,j,l} + b \tag{3.2}$$

Also deriving from this idea, a convolution layer can have an arbitrary number of output channels, the filter has 4 dimensions: the first two dimensions represent the spatial location in the image, the third dimension represents the input channel and the fourth dimension is the output channel. With these two generalizations, it is possible to create a multi-channel picture as an intermediate representation within the network regardless of the number of input channels and output channels. For simplicity, we will omit the output dimensions in the following formulas but multiple output dimensions are used in our model.

Convolutional Neural Networks have been applied to many image processing tasks including single super-resolution [9][18]. In this case, the image is usually pre-processed using a classic method such as bi-cubic interpolation to produce the desired resolution. Then the image is sharpened with a Neural Network that preserves image dimensions.

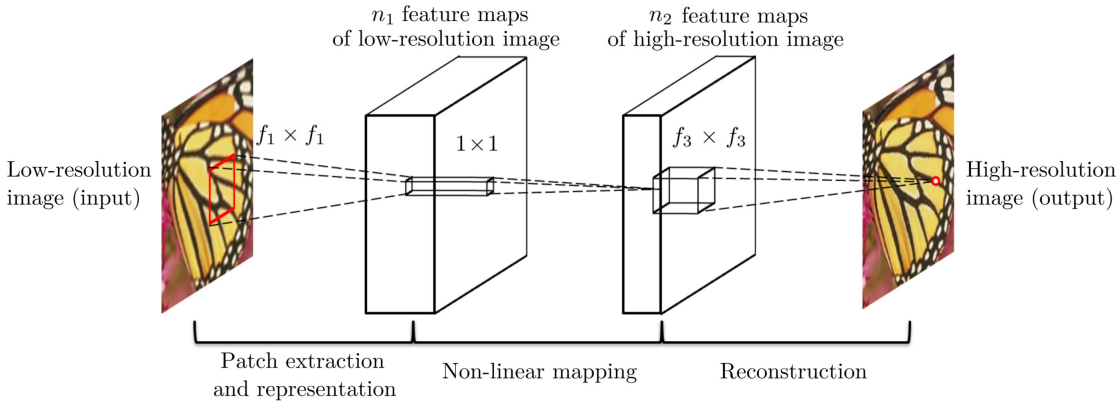


Figure 3.2: Architecture of SRCNN, a single-image super-resolution CNN source: [9]

3.2 Recurrent Neural Network

A Recurrent network is a type of Artificial Neural Network specialized in cases where the input is a sequence, often a time series. The Recurrent Neural Network uses multiple inputs from the sequence to make a prediction at a given step of the sequence. Recurrent Neural Networks have a variety of applications in speech-to-text, Natural Language processing, computer vision, time series.

There are many different techniques to save a representation of past inputs in the network. The simplest one is to save the past inputs and use them directly. Other Techniques such as LSTM and GRU learn a representation of past inputs and update this representation at every step.

In the case of a Convolutional Neural Network for video super-resolution, the scene can change quickly so we are more interested in nearby frames than long term recurrence. VSRNet [10] is storing intermediate representation at different time stamps and concatenating those representations on the channel dimension (see fig. 3.3). Their method is bi-recurrent meaning that past and future frames, as well as the current frame, are used to super-resolve the current frame. In practice, their recurrent layer is written as:

$$f^t_{u,v}(x) = \sum_{m=-n}^{n} \sum_{l=0}^{c} \sum_{i,j=-k}^{k} x^{t-m}_{u+i,v+j,l} w^{n+m}_{u,v,l} + b \tag{3.3}$$

Where the super script represents time.

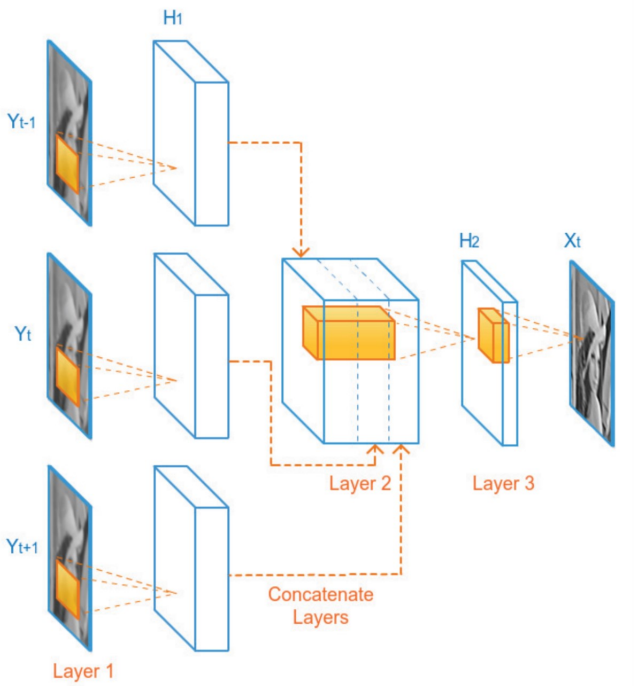


Figure 3.3: VSRNet(2) a recurrent neural network for video super-resolution, source: [10]

3.3 Convolution and sparse data

As stated in part 2. classic Convolutional networks are not well suited to sparse data because the magnitude of the output is directly influenced by the number of measured values in the convoution window. Uhrigh et al. [17] have demonstrated this effect on depth completion from a sparse depth map as shown on the figure 3.4.

The network must have access to the sparsity map in order to handle missing pixels differently. This can be achieved simply by adding the sparsity map as an additional channel of the input. As depicted in figure 3.4, the output of such a network produces a much more realistic output albeit still significantly noisy. This confirms the hypothesis that the sparsity map brings useful information for the task of depth completion. However, the level of noise suggests that the classic convolution is not well suited to a sparse input.

3.4 Sparse convolution

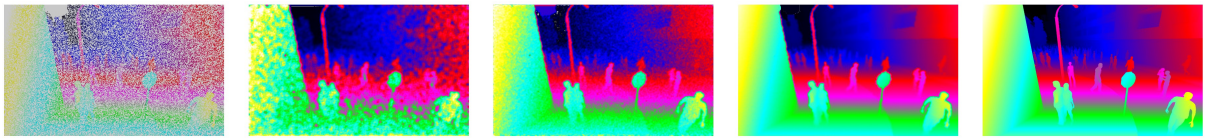


Figure 3.4: Comparison of different depth completion methods, from left to right: Input, classic convolutional network, classic convolutional network with sparsity map, SparseCNN, groundtruth. source: [17]

3.4.1 Definition

The sparse convolution as described by Uhrig et al. [17] considers only observed pixels and regularizes the output of each dot product using the local sparsity level. This can be formulated as

$$f_{u,v}(x, o) = \frac{\sum_{i,j=-k}^{k} o_{u+i,v+j} x_{u+i,v+j} w_{u,v}}{\sum_{i,j=-k}^{k} o_{u+i,v+j}} + b \tag{3.4}$$

The sparsity matrix is updated at each layer using 2D max-pooling over the same window as the feature convolution. This represents information spreading locally at each convolution.

$$g_{u,v}(o) = \max_{(i,j)\in[-k,k]^2} o_{u+i,v+j} \tag{3.5}$$

The formula 3.4 can be generalized to multi-channel sparse inputs when all the channels share the same sparsity map, by summing over the c channels. This case is encountered deeper in the network if one of the previous layers produces a multi-channel intermediate representation. Let's denote $x_{i,j,l}$ the input at pixel (i, j) for the channel l and $w_{i,j,l}$ the filter weight

$$f_{u,v}(x, o) = \frac{\sum_{l=0}^{c} \sum_{i,j=-k}^{k} o_{u+i,v+j} x_{u+i,v+j,l} w_{u,v,l}}{\sum_{i,j=-k}^{k} o_{u+i,v+j} + \epsilon} + b \tag{3.6}$$

Here ϵ is a small number to avoid division by 0.

3.4.2 Interpretation

With formula 3.6, we can see that for a dense input, the proposed operation would be equivalent to a classic convolution scaled by the filter size. Because the weights are learned over the training phase, this would only affect initialization but should produce an equivalent result after training.

Furthermore, information is spreading with each sparse convolution, this is well represented with the sparsity map update in equation 3.5. For every valid input value, all output pixels in the convolution window of size $2k + 1$ centered around the valid input pixel will have an "observed" output value.

3.4.3 Experimental Results

Uhrig et al. [17] used a deep Sparse convolutional network with 6 layers. Each layer preserves the input shape using padding and a convolution step of 1. Each sparse convolution is followed by a ReLu activation function.

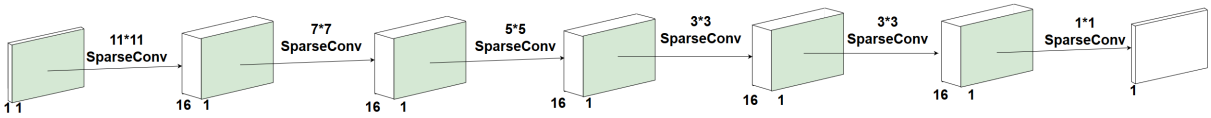


Figure 3.5: Sparse convolutional network as described by Uhrig et al. [17]

One interesting additional result is that this network is invariant to sparsity: for inputs with varying rates of valid measurements from 100% to 5%, the reconstruction error to a dense depth map does not vary significantly. They report that the mean average error dropped only slightly from 0.99m to 0.98m over this range of sparsity rate. The two variants of the classic convolutional network (with and without the sparsity map in the input) did not display sparsity invariance.

This sparsity invariance is of course limited within a defined range, the size of each convolution defines how information can spread. In their architecture described in figure 3.5 the convolution windows are sqares of length 11, 7, 5, 3, 3, and 1. This means that information will spread from one pixel to the all the neighbouring pixels within a square of size $2 \times (5 + 3 + 2 + 1 + 1 + 0) + 1 = 2 \times 12 + 1 = 25$

3.5 Recurrent sparse convolution

As LIDAR sensors record in sequence, producing sparse depth maps at a frequency of 5-10Hz we believe that depth map processing can benefit from the information in neighboring frames. We consider the concatenation technique used by Kappeler et al. [10] for video super-resolution. This technique consists in adding the neighboring frames as additional channels in the input and performing 2D convolution over this multi-channel input. However, due to the changing environment, each frame can have a different sparsity map as reflective objects move and objects block the view of the sky. Hence we cannot use the multi-channel sparse convolution of Uhrig et al. [17] as it supports only channels that have the same sparsity map.

3.5.1 Recurrent Convolution

We propose a new Recurrent Sparse Convolution that can combine information from multiple frames, each with a different sparsity map. We consider inputs taken at regular timestamps and perform sparse convolution on a concatenation of n consecutive inputs of the sequence:

$$f^t_{u,v}(x,o) = \frac{\sum_{m=0}^{n-1}\sum_{l=0}^{c}\sum_{i,j=-k}^{k} o^{t-m}_{u+i,v+j} x^{t-m}_{u+i,v+j,l} w^m_{u,v,l}}{\sum_{m=0}^{n-1}\sum_{i,j=-k}^{k} o^{t-m}_{u+i,v+j} + \epsilon} + b \tag{3.7}$$

In this recurrent layer, the new sparsity matrix combines the values of every time step using 3D maxpooling:

$$g^t_{u,v}(o) = \max_{(i,j,m)\in[-k,k]^2\times[0,n]} (o^{t-m}_{u+i,v+j}) \tag{3.8}$$

3.5.2 Network architecture

We investigate several network architectures based on the SparseCNN by Uhrig et al. [17]. We used the same network depth, filter sizes, convolution step, and padding parameters as described in their paper but replaced one sparse convolution layer by a recurrent layer.

The network architecture is similar to SparseCNN so we used transfer learning as defined by Kappeler et al. [10]. We can directly copy weights and biases from the SparseCNN

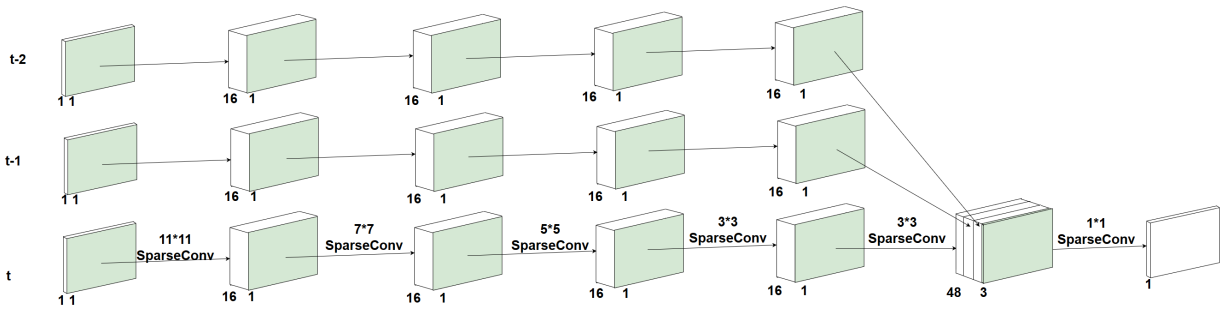


Figure 3.6: A variant of our proposed architecture, here the recurrent layer is located on the 5th layer and combines features from three time-steps.

architecture for all the non-recurrent layers. The convolution kernel of the recurrent layer doesn't have the same shape in the recurrent network as there are more input channels. The solution is to initialize the weights of the recurrent layer by broadcasting the kernel weights of the SparseCNN convolution over the temporal dimension. By initializing weights in this manner, the recurrent sparse convolution and the sparse convolution will produce the same output on a stationary input.

The recurrent layer can then be trained independently on a sequence of sparse frames.

3.6 Bi-recurrent sparse convolution

A variant of the recurrent Sparse CNN is a bi-recurrent algorithm that uses a concatenation of past, current and future frames to produce the output. This convolution operation over $2n+1$ input frames centered around the current frame can be described as:

$$f^t_{u,v}(x,o) = \frac{\sum_{m=-n}^{n}\sum_{l=0}^{c}\sum_{i,j=-k}^{k} o^{t+m}_{u+i,v+j} x^{t+m}_{u+i,v+j,l} w^{m}_{u,v,l}}{\sum_{m=0}^{n-1}\sum_{i,j=-k}^{k} o^{t-m}_{u+i,v+j} + \epsilon} + b \tag{3.9}$$

This operation can be used as a replacement for any of the layers of the SparseCNN and can combine any number of time frames. Bi-recurrence is beneficial when the scene is moving because, for a given number of input frames (centered around the current frame), the inputs are on average closer to the current time. For 3 input frames, the recurrent network inputs are on average taken 1 time-step away from the current frame. In the case of a bi-recurrent network for the same number of input frames, the inputs are on average 2/3 frames away from the current frame. However, this comes with the drawback that

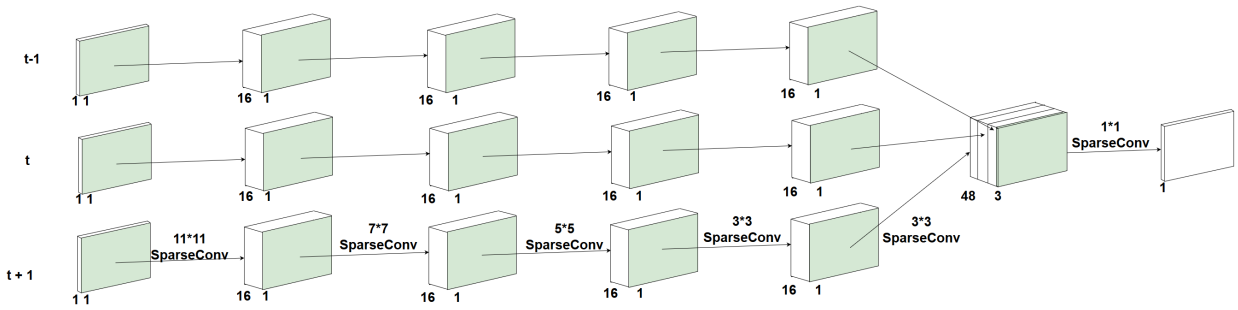


Figure 3.7: Our bi-recurrent sparse architecture, the bi-recurrent convolution is at the 5th layer and combines features extracted from 3 time-frames

bi-recurrent networks can only be used for offline processing because in the case of an online system future frames are not yet available.

Similarly to the Recurrent Sparse Convolution presented above, a bi-recurrent sparse convolutional network can be trained faster using transfer learning from a SparseCNN of the same shape.

Chapter 4

Methodology

4.1 Dataset

We used the SYNTHIA dataset [19] to train and evaluate our models. This dataset contains computer generated scenes of road traffic in various environments from city to highway in different weather and lighting condition: Fall, Winter, Summer, Sunset, Rain. It contains images, dense depth maps and the groundtruth segmentation for several sequences of images. Each sequence contains 4 different views (front, back, left, right) of the vehicle's surroundings from two points of view at the left and right side of the roof. Each sequence is available in all of the four weather and lighting conditions.

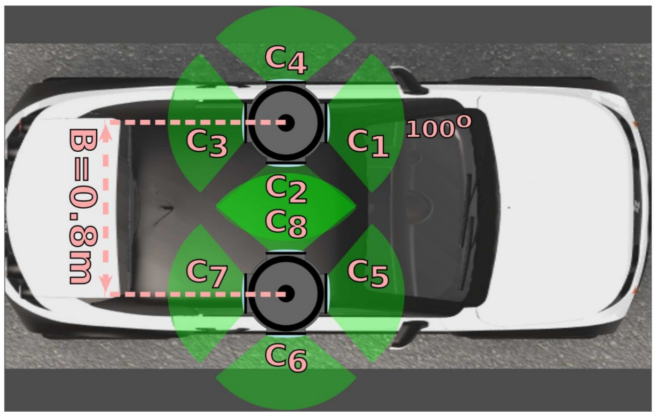
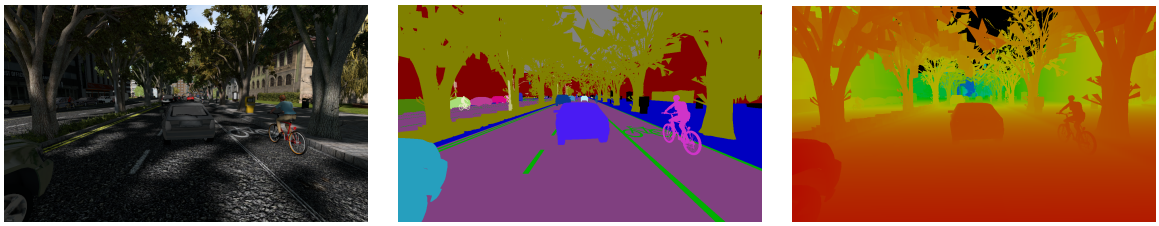


Figure 4.1: The virtual car and sensor locations used in the simulation to create the Synthia dataset, source: [19]

The dense depth maps have a 760*1280 resolution and are encoded as 16 bit unsigned integers where each value represents the distance in centimeters from the sensor to the

object. The absence of object is encoded as the maximum value ($2^{16} - 1$) and is used to represent the sky.



(a) RGB image (b) Scene segmentation (c) Dense depth map

Figure 4.2: Views of one scene through the different data types available in the Synthia dataset

We used all the sequences available but chose to use only one weather condition beacuse weather and lighting do not impact the depth groundtruth. Here are the subsequences used:

- SYNTHIA-SEQS-01-DAWN: 1451 frames of highway scene
- SYNTHIA-SEQS-02-DAWN: 941 frames in a New-York like city
- SYNTHIA-SEQS-04-DAWN: 850 frames of European city scene
- SYNTHIA-SEQS-05-DAWN: 866 frames of a New-York city scene
- SYNTHIA-SEQS-06-DAWN: 909 frames of a highway scene

Using a synthetic dataset enables us to work with a dense groundtruth depth map, which is currently not possible to obtain with a sensor in a real-world dynamic scene.

4.2 Experiments

We ran two main experiments to test our algorithm, we trained and tested our algorithms independantly for both scenarios of depth map compression and LIDAR augmentation:

4.2.1 Depth completion for depth map compression

Depth completion is fundamentaly a lossy compression algorithm. A dense depth map can be compressed to a sparse representation using sub-sampling. Then the dense depth map can be recovered with some level of error using a depth completion algorithm.
Since decompression is typically performed a posteriori, when all the sparse depth maps are available, we can use a bi-recurrent algorithm to perform depth completion. We

performed depth-completion on randomly sub-sampled depth maps with various sparsity levels ranging from 75% to 99.5% which translates to compression ratios between 4 and 200. We used our bi-recurrent sparse convolution and compared it to the SparseCNN algorithm by Uhrig et al. [17]. We used one bi-recurrent layer as a replacement of the 5^{th} layer of the SparseCNN architecture. This recurrent layer combines inputs from 3 time frames. This architecture is depicted in figure 3.7

4.2.2 Depth completion for LIDAR up-sampling

Depth completion can be used to up-sample a LIDAR frame or sequence of frames. Lidar typically have a low vertical resolution, defined by the number of lasers used in parallel. Efficient vertical up-sampling allows us to maintaining accuracy with a small number of lasers. Since LIDARs are typically used in autonomous systems, depth completion needs to be performed quickly after each measurment is made so that the system can react quickly to a changing environment. For this reason we will use the recurrent sparse convolution instead of the bi-recurrent algorithm. Our recurrent architecture has a similar architecture to SparseCNN by Uhrig et al. [17] but the 5^{th} layer is recurrent and combines features extracted from the current and past two time frames, this architecture is depicted in figure 3.6. We compared our recurrent algorithm with SparseCNN described in figure 3.5 for sparsities ranging from 50% to 96%.

4.3 Pre-processing

The first step of pre-processing is to convert all values to float32 and divide by 100 so that all values are scaled in meters m. To generate the input to our Neural Network we apply a number of sparsity masks to simulate real world data. First we compute the "measurable" sparsity mask which represents all objects whose distance can be measured. The effect of this is to label the sky's distance as unmeasurable or undefined.
Depending on the use case for the data, the depth map is processed differently:

4.3.1 Compression

In order to perform dense depth map compression, we want to use a random sub-sample of the depth map. We create a sampling mask which is a randomly distributed binary mask

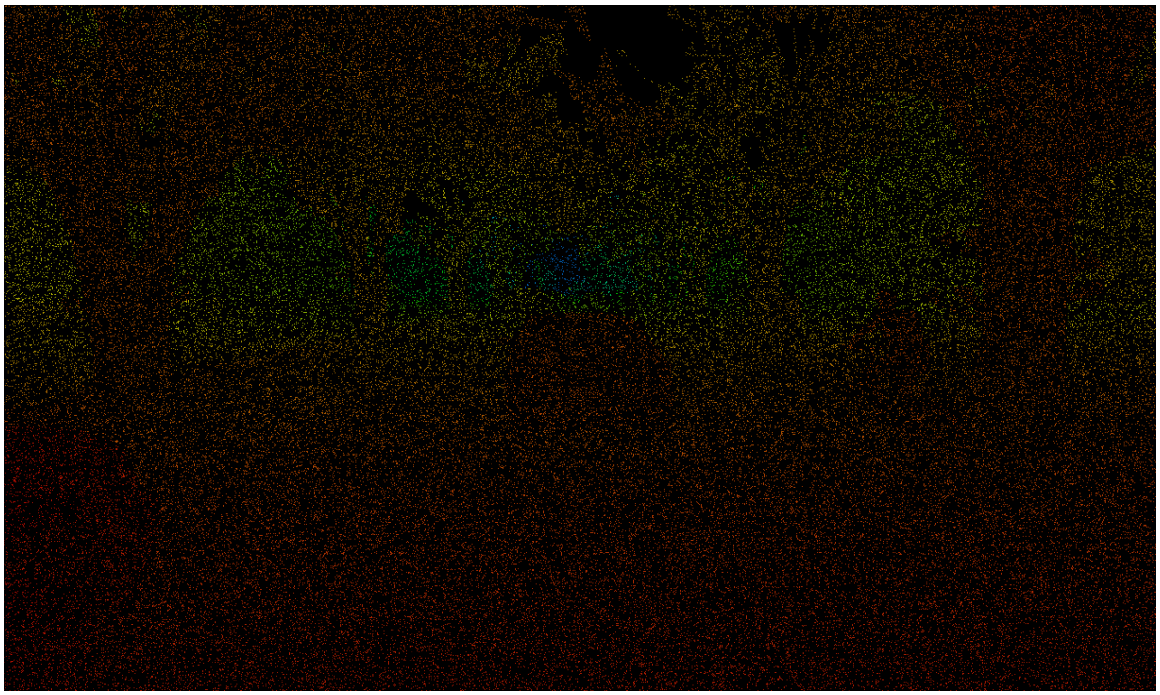


Figure 4.3: Randomly sub-sampled depth map from the Synthia databse

with a given sparsity rate. The sampling mask and the observablility mask are combine with element-wise multiplication into the final sparsity map. Finally this sparsity map is applied to the dense input, this is the compression step. To decompress the sparsity map and recover the dense depth map (with error), we use both the sparse depth map and the sparsity map as input to the Neural Network which performs decompression.

4.3.2 LIDAR super-resolution

In order to simulate a LIDAR, we must create inputs that are similar to what a LIDAR would produce. We create a sampling map to represent which bearings are sampled across the movement of a LIDAR with a given number n of rotationg lasers. The LIDAR mesures n reguarly spaced rows, giving a value of 1 across all columns in those n rows of the sampling map. All the other rows of the sampling map hold the value 0. For example, to simulate a LIDAR with 40 lasers and an image height of 760, the depth will be measured along 1 row every 19 rows and none of the other rows will be observed. . Then observability mask and the sampling mask are combined using element-wise multiplication to create the final sparsity map. Finally this sparsity mask is applied to the dense depth map using element-wise multiplication to produce the sparse depth map. The sparse depth map and the binary sparsity map are both used as inputs to our network.

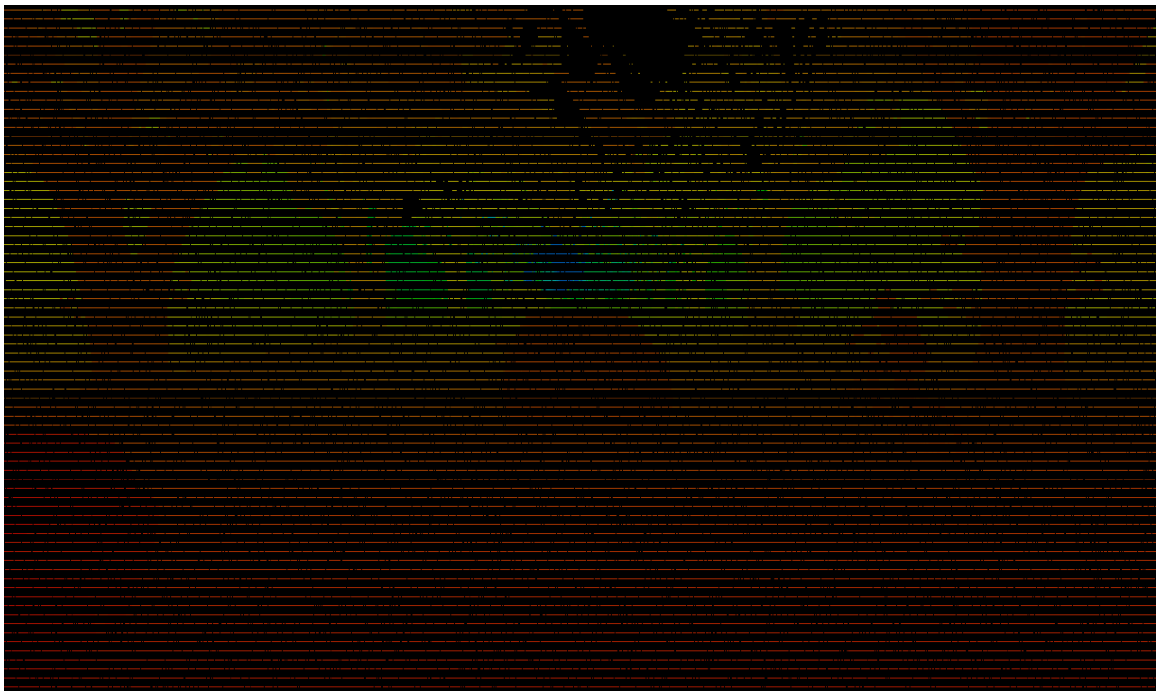


Figure 4.4: LIDAR sub-sampled depth map from the Synthia databse

4.4 Loss and performance metrics

We defined some metrics to comapare our prediction x to the groudtruth y and measure the accuracy of our algorithm. We used Root Mean Square Error (RMSE) as our loss function. However we modified the formula to only take into account the measurable distances and discard all pixels that represent the sky. If we define the Measurablility matrix M(y) of size $h \times w$ such that for all pixel location (i, j)

$$M(y)_{i,j} = \begin{cases} 0 \; if \; y_{i,j} = SKY\_VALUE \\ 1 \; otherwise \end{cases}$$

Then the RMSE can be expressed as:

$$SparseRMSE(x, y) = \sqrt{\frac{\sum_{(i,j)\in[0,h]\times[0,w]} (x_{i,j} - y_{i,j})^2 \times M(y)_{i,j}}{\sum_{(i,j)\in[0,h]\times[0,w]} M(y)_{i,j}}} \tag{4.1}$$

We used Mean Absolute Error (MAE) as an additional performance metric. Similarly, we used a modified version of the MAE to only take into account measurable distances.

This can be expressed as:

$$SparseMAE(x,y) = \frac{\sum_{(i,j)\in[0,h]\times[0,w]} |x_{i,j} - y_{i,j}| \times M(y)_{i,j}}{\sum_{(i,j)\in[0,h]\times[0,w]} M(y)_{i,j}} \tag{4.2}$$

4.5 Training parameters

We trained our algorithm in 3 phases:

- **Non Recurrent Training**

  First a Sparse CNN Network with a similar architecture to our recurrent network is trained on single-depth map. The training involves two epochs over all depth maps from the left camera in Synthia sequences 1, 2 and 4. The model is trained with an Adam optimizer with a learning rate of 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-7}$
- **Recurrence Learning** Then we build a Neural Network where one of the layers is recurrent and transfer weights from the SparseCNN architecture. The weights biases from the non-recurrent layers are directly copied. For the recurrent layer, weights are broadcasted to the new convolution kernel dimensions. The network is trained on the forward view of the sequences 1, 2, and 4 from the Synthia dataset. We train on one epoch and used the same training parameters as stated above. However only the weights from the recurrent layer are learned, other weights are processed as constants
- **Fine-tuning** Finally we train the recurrent network on one more epoch of sequences 1, 2, and 4 from the Synthia dataset. We use the same training parameters as stated above. However this time, we train all the weights and biases in the network. This enables us to fine-tune the weights of other layers for the recurrent network.

4.6 Implementation

We use a number of tools in this work to organize and pre-process data and implement our Machine Learning model. Our implementation can be found on at `https://github.com/Etienne-Gautier/Thesis-public`

- **OpenCV** is an open source image processing library with advanced capabilities in object recognition, motion-tracking that supports GPU acceleration. It is written

in C++ but has APIs in several languages including python. We used it for simple tasks in file IO and pre-processing.

- **Numpy** is an open-source library for numeric operations for python implemented in CPython. It supports classic linear algebra operations and easy manipulation of arrays. We used Numpy to generate the sparsity maps and to compute a visualization of depth maps.
- **Matplotlib** is a visualization library in python. We used it to display performance graphs and colorized views of depth maps.
- **Tensorflow** is an open source Machine Learning framework which is primarily maintained by Google Brain. It contains implementations of many classic Machine Learning algorithms and performs automatic differentiation for backpropagation. We implemented our Neural Network by using and extending the Keras API for Layer, Model, Sequence and Loss classes.

Chapter 5

Results

5.1 Depth map compression error

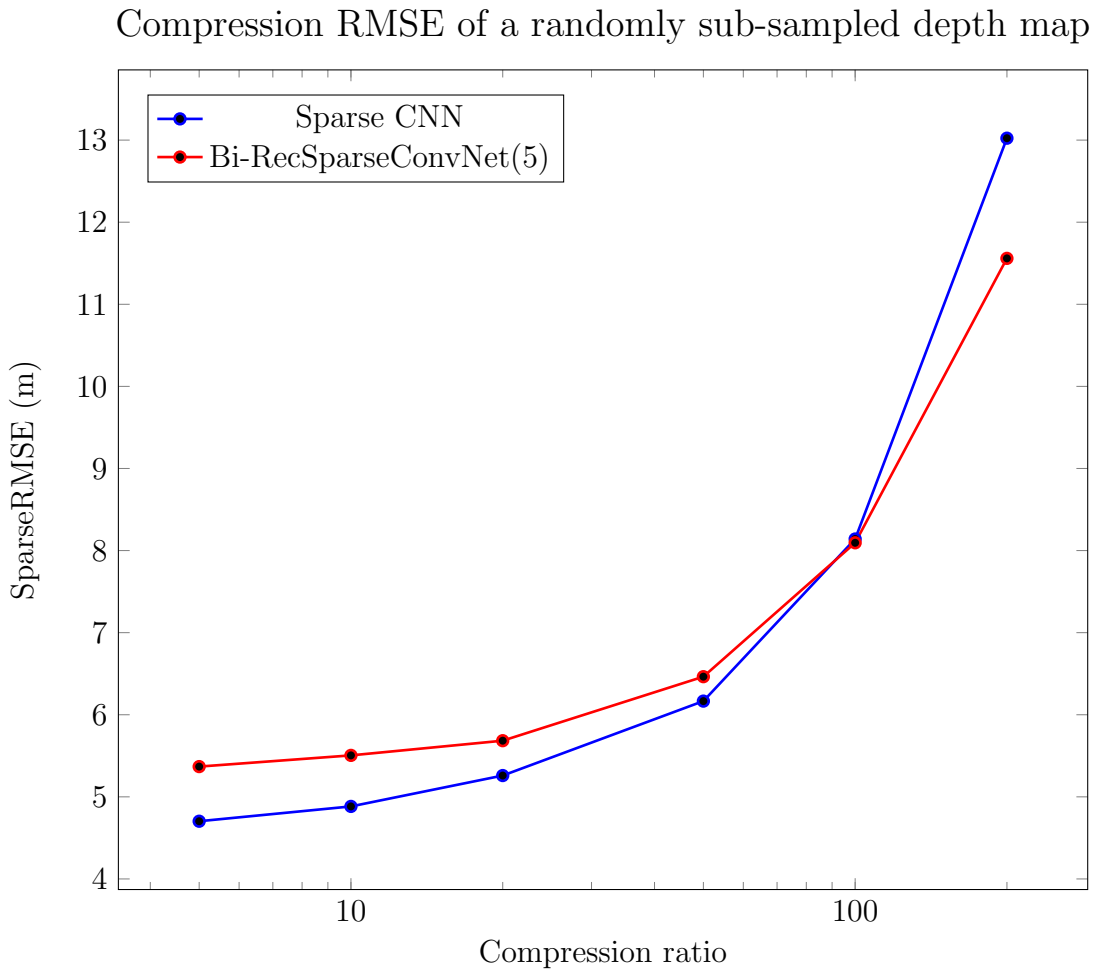


Figure 5.1: Compression loss of the baseline SparseCNN and our bi-recurrent algorithm on the forward view depth maps from the left camera in the Synthia SEQS-06-DAWN sequence

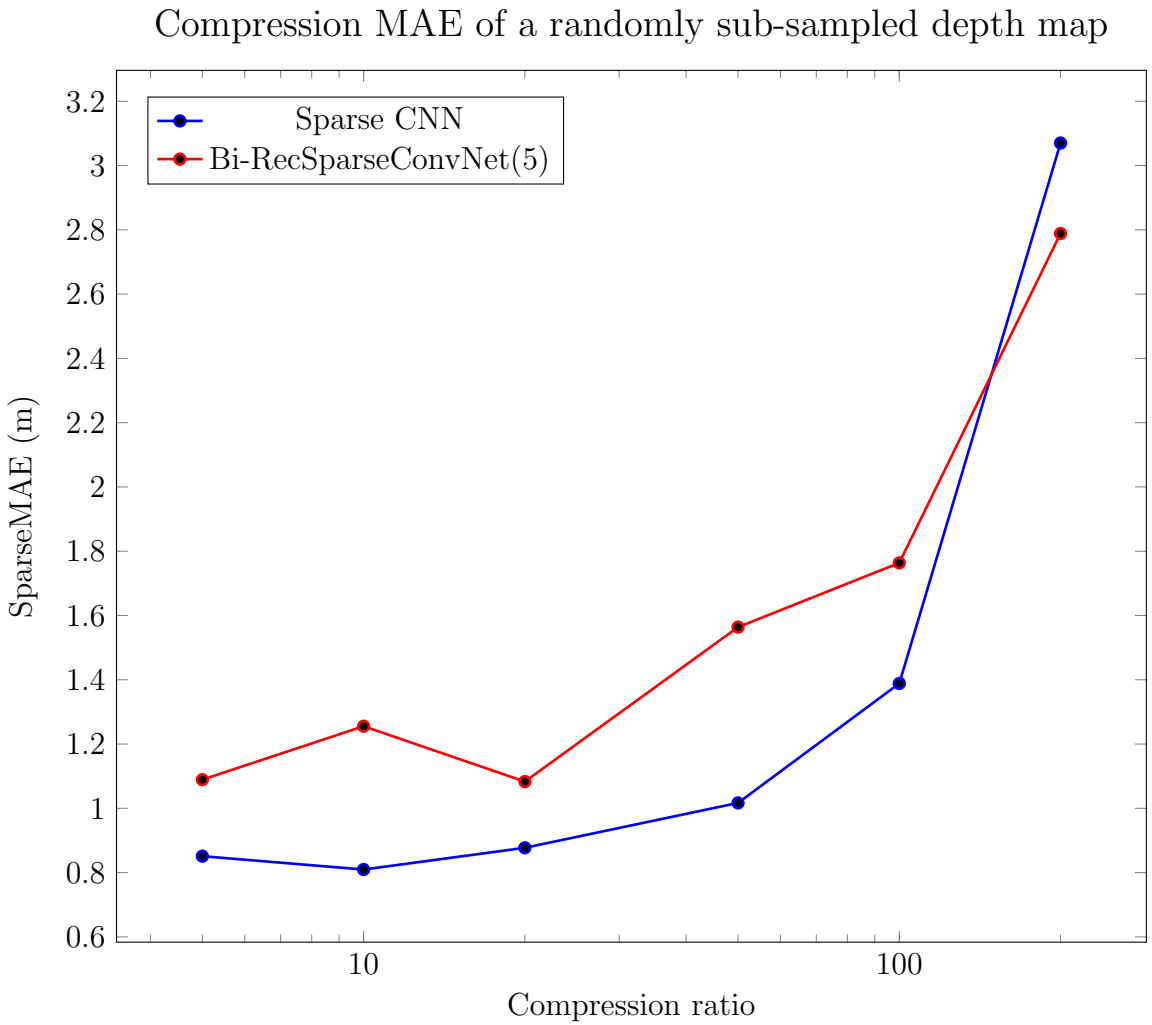


Figure 5.2: Compression loss of the baseline SparseCNN and our bi-recurrent algorithm on the forward view depth maps from the left camera in the Synthia SEQS-06-DAWN sequence

Figure 5.1 shows that for low compression ratios, Sparse CNN outperforms our recurrent algorithms however, for higher compression ratios our algorithm performs better. Bad performance for low compression ratio can be explained by the fact that the sub-sampled current frame already contains enough information for a good reconstruction.

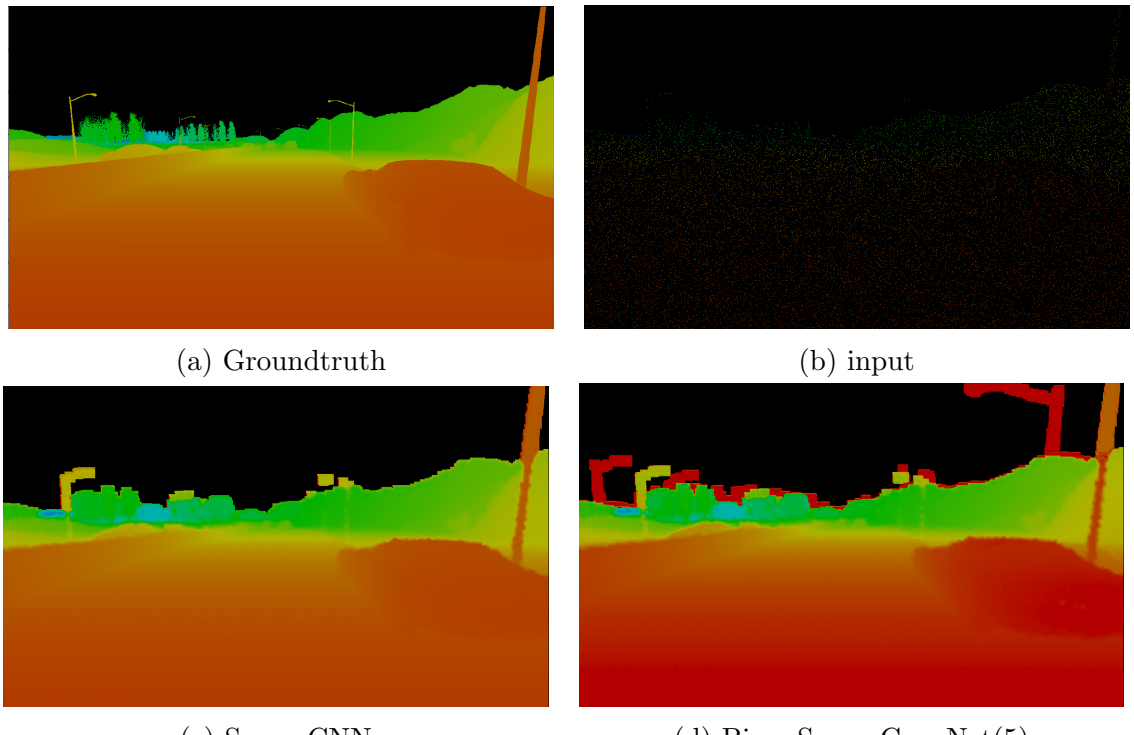
If we take the extreme example of a compression ratio of 1 (no compression) the current input frame contains all the information directly and the simply returning the input frame would be optimal (loss of 0). In this extreme case, the neighbouring frames are irrelevant to decompression. We believe that for low compression ratios, neighbouring frames bring little additional information and the higher number of parameters from the recurrent layer cause the increased error.

However as the compression ratio grows, the amount of information contained in each

input frame decreases. For higher compression ratios, the current frame does not contain enough information to perform a perfect reconstruction and the information contained in the neighbouring frames become more valuable for reconstruction. In fact for a compression ratio varying from 5 to 20, the SparseCNN RMSE increases by 12%, in the case of our Bi-recurrent algorithm, the RMSE increases by only 8% over this range of compression ratios. This slower growth continues as compression rate increases and for compression ratios over 100, the Bi-recurrent algorithm performs better than the SparseCNN.

We noticed that the reconstruction MAE is not monotonically increasing with the compression ratio. This is the case for our algorithm and SparseCNN and we identified the same trend on the validation set (SEQS-05-DAWN). We have found no compelling argument to explain this phenomenon.

5.1.1 Visualization



(a) Groundtruth (b) input

(c) SparseCNN (d) Bi-recSparseConvNet(5)

Figure 5.3: Visual comparison of compression algorithms on depth maps from the SEQS-6-DAWN sequence with a sparsity of 0.05

5.2 LIDAR depth completion

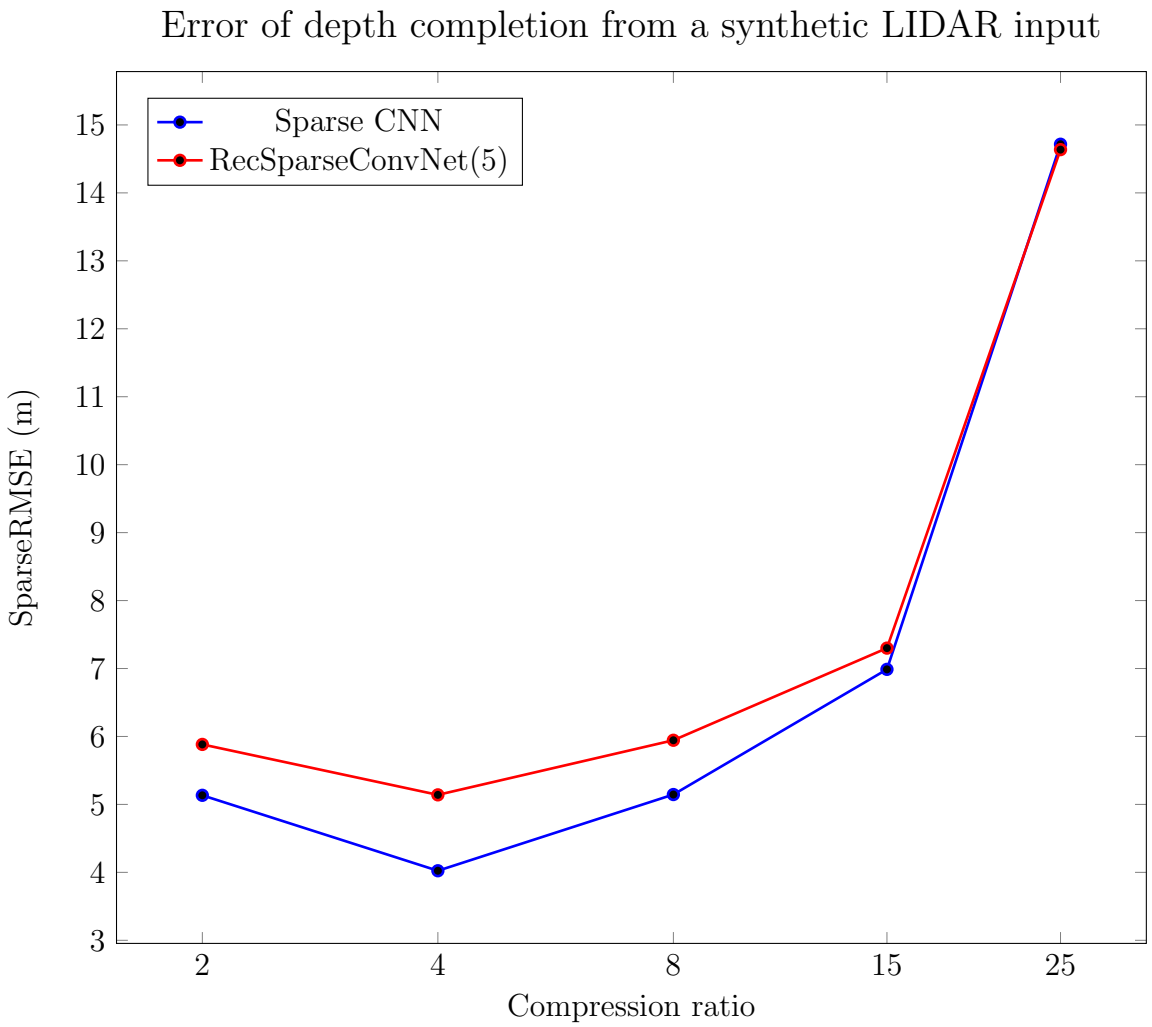
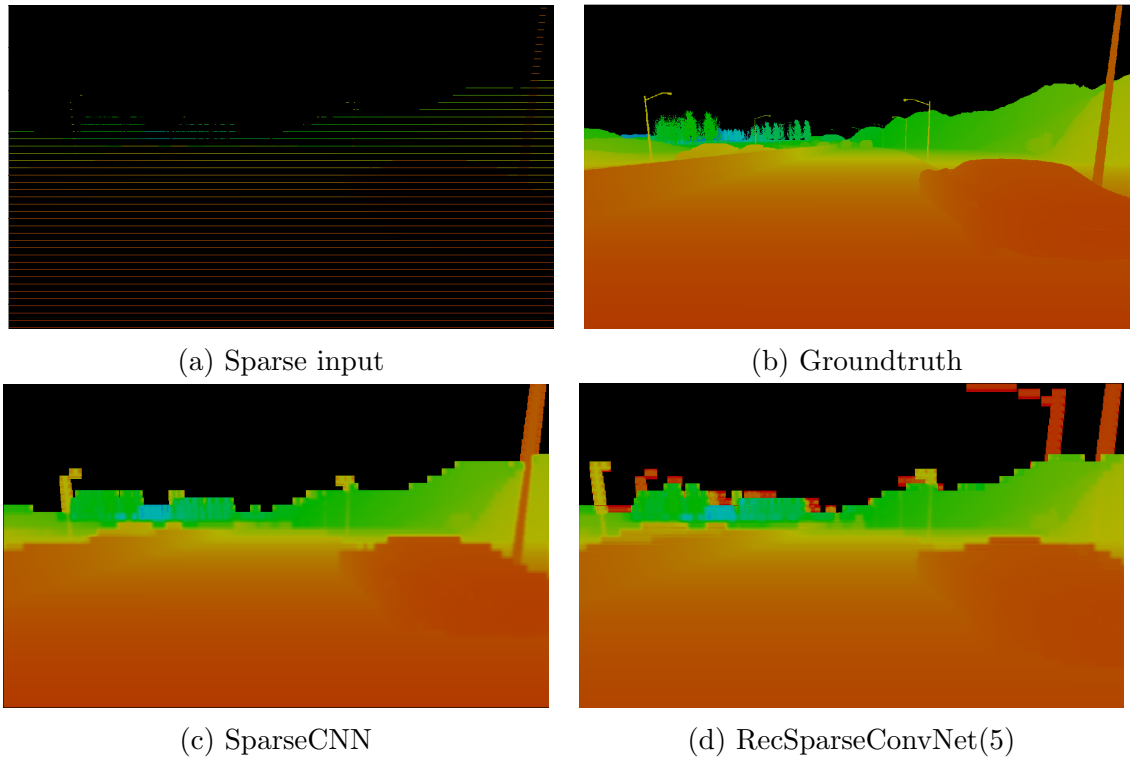


Figure 5.4: Depth Completion error of the baseline SparseCNN and our recurrent algorithm on the forward view depth maps from the left camera in the Synthia SEQS-06-DAWN sequence

We observe the same trend as for the randomly sub-sampled input: the recurrent networks performs worse than SparseCNN for low sparsity levels and as the sparsity levels increaseas, our network error grows slower than SparseCNN. Above a compression ratio threshold, our network performs better than sparseCNN. The compression ratios for which our network outperforms SparseCNN are however different. Our network outperforms SparseCNN starting from much smaller compression ratios in the case of LIDAR sampling.

Similarly to the ramdom subsampling case, the error is not monotonically increasing, in this case the SparseRMSE is decreasing around a compression ratio of 4 for both algo-

rithms.



(a) Sparse input (b) Groundtruth

(c) SparseCNN (d) RecSparseConvNet(5)

Figure 5.5: Visual comparison of lidar depth completion algorithms on depth maps from the SEQS-6-DAWN sequence with a compression ratio of 15

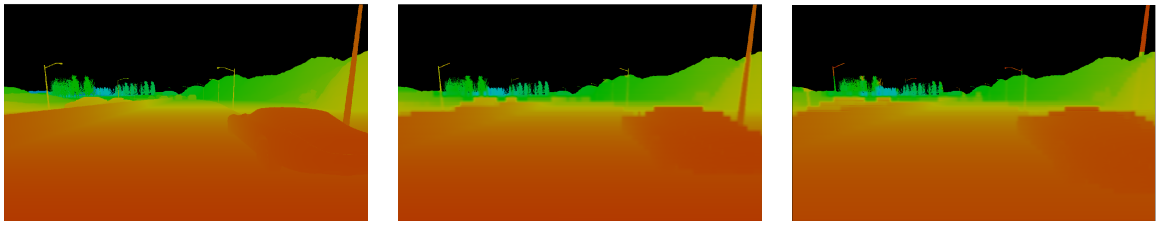
5.3 Visual comaprison and error metric

Visual analysis of the two algorithms in the two scenarios (random sampling and LIDAR sampling) show that one area of significant error for both algorithms is the sky just above the horizon. This error is not taken into account in our SparseRMSE and SparseMAE metrics because the error can only be computed for areas that are observable. Because the sky's distance is not mesurable, these regions are not taken into account.

The reason that these regions are incorrect is that the sparsity map outputed by each algorithms only represents the spread of knowledge of each measurment as defined in section 3.5. In particular, the lamppost is duplicated in the output of our algorithm because the there were valid pixels at around that location of the image in one of the previous frames used in the recurrence.

Implementation of technical solutions to this issue are outside of the scope of this thesis but a possible solution is listed in section 6.2

A different representation of the same results is shown in figure 5.6. This representation uses the groundtruth sparsity map to define the observed regions and the algorithm output to define the depth. This pictures more accurately what our error metric represents.



(a) Groundtruth (b) SparseCNN (c) RecSparseConvNet(5)

Figure 5.6: Visual comparison of lidar depth completion algorithms as seen by our error metric on depth maps from the SEQS-6-DAWN sequence with a compression ratio of 15

In this representation, the differences between both algorithms are minimal and both are very similar to the groundtruth.

Chapter 6

Conclusion

6.1 Conclusion

In this paper we investigated the use of recurrent Convolutional Neural Networks for depth completion with two use cases for depth map compression and LIDAR depth completion. The goal was to demonstrate that a system that uses neighbouring frames can outperform an equivalent non-recurrent system in the task of depth completion. The study involved designing a new recurrent Sparse convolution operation that combines multiple sparse frames, and comparing a network architecture using this recurrent sparse convolution with SparseCNN. We trained and tested both algorithms for two scenarios, depth map compression and lidar depth completion. The experiments showed that in both cases, our alogrithm outperformed sparseCNN for high compression ratios. This shows that our recurrent sparse convolution operation can successfully combine information from nearby frames.

These results are encouraging however, our network underperforms for a range of low compression ratios. While this can be easily explained in the extreme case of no compression (compression ratio of one), the fact that our network underperforms for compression ratios as high as 50 for compression and 15 for LIDAR depth completion remains an issue. Furthermore, the error around the horizon due to the simplistic information spreading model remain a significant issue for the adoption of both algorithms presented here.

Overall the SYNTHIA dataset provides a good benchmark to compare depth completion algorithms. Our experiments shows that neighbouring frames bring valuable informations

for depth completion however, the error analysis shows that there are some clear limitations in terms of the range of sparisties where our algorithm performs well and in how sparsity map from different time frames are combined.

6.2 Future Work

To confirm our findings, it is would be interesting to apply our technique on a real-world dataset such as KITTI depth completion benchmark. This would confirm that the insights developped in this paper still apply and that the proposed algorithm can generalize well.

Apart from this test, other variants of the algorithm can be tested as well. Possible improvements include using several recurrent layers in the same network, exploring different parameters for the network architecture (number of layers, convolution kernel size and shape).

Another intersting aspect is to study the influence of the amount of change between frames including the speed, steering of the car. These information are usually available in an autonomous driving scenario and could be used in different manners. The input frames could be pre-processed with a motion compensation that aligns matching pixels or the parameters could be used as inputs to the machine learning model. Finally in the case of depth map compression, instead of using a random sub-sampling with the same sparsity ratio on all of the frame, image segmentation could be used to decide what sparsity rate is appropriate for this region. For example, the road could be sub-sampled at a lower rate than pedestrians and other road users. This segmentation-guided compression could enable lower error in detail-rich regions of the frame and higher compression ratios elsewhere. Furthermore, this could solve the issue underlined in section **??** where both algorithm fail to represent the horizon accurately. The segmentation can help define which areas of the output should have depth value and which ones are non-measurable (sky).

Bibliography

[1] W. H. Organization *et al.*, *Global status report on road safety 2018*. World Health Organization, 2018.

[2] J. R. Treat, N. Tumbas, S. McDonald, D. Shinar, R. D. Hume, R. Mayer, R. Stansifer, and N. Castellan, "Tri-level study of the causes of traffic accidents: Final report. executive summary.," 1979.

[3] R. Tsai, "Multiframe image restoration and registration," *Advance Computer Visual and Image Processing*, vol. 1, pp. 317–339, 1984.

[4] M. Irani and S. Peleg, "Improving resolution by image registration," *CVGIP: Graphical models and image processing*, vol. 53, no. 3, pp. 231–239, 1991.

[5] M. Elad, "Super-resolution reconstruction of image sequences-adaptive filtering approach," PhD thesis, PhD thesis, The Technion-Israel Institute of Technology, Haifa, Israel, 1996.

[6] S. Baker and T. Kanade, "Limits on super-resolution and how to break them," *IEEE Transactions on Pattern Analysis*, vol. Machine Intelligence, no. 9, pp. 1167–1183, 2002, ISSN: 0162-8828.

[7] W. Yang, J. Feng, G. Xie, J. Liu, Z. Guo, and S. Yan, "Video super-resolution based on spatial-temporal recurrent residual networks," vol. 168, pp. 79–92, 2018, ISSN: 1077-3142.

[8] R. Zeyde, M. Elad, and M. Protter, "On single image scale-up using sparse-representations," in *International conference on curves and surfaces*, Springer, 2010, pp. 711–730.

[9] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *European conference on computer vision*, Springer, pp. 184–199.

[10] A. Kappeler, S. Yoo, Q. Q. Dai, and A. K. Katsaggelos, "Video super-resolution with convolutional neural networks," *Ieee Transactions on Computational Imaging*, vol. 2, no. 2, pp. 109–122, 2016, ISSN: 2333-9403.

[11] S. Schuon, C. Theobalt, J. Davis, and S. Thrun, "High-quality scanning using time-of-flight depth superresolution," in *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, IEEE, 2008, pp. 1–7.

[12] ——, "Lidarboost: Depth superresolution for tof 3d shape scanning," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2009, pp. 343–350.

[13] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, and A. Davison, "Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, ACM, pp. 559–568, ISBN: 1450307167.

[14] H. Afzal, K. Al Ismaeil, D. Aouada, F. Destelle, B. Mirbach, and B. Ottersten, "Kinect deform: Enhanced 3d reconstruction of non-rigidly deforming objects," in *2014 2nd International Conference on 3D Vision*, vol. 2, IEEE, pp. 7–13, ISBN: 147997000X.

[15] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1907–1915.

[16] R. Köhler, C. Schuler, B. Schölkopf, and S. Harmeling, "Mask-specific inpainting with deep neural networks," in *German Conference on Pattern Recognition*, Springer, 2014, pp. 523–534.

[17] J. Uhrig, N. Schneider, L. Schneider, U. Franke, T. Brox, and A. Geiger, "Sparsity invariant cnns," in *2017 International Conference on 3D Vision (3DV)*, IEEE, pp. 11–20, ISBN: 1538626101.

[18] W.-S. Lai, J.-B. Huang, N. Ahuja, and M.-H. Yang, "Fast and accurate image super-resolution with deep laplacian pyramid networks," *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[19] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016.