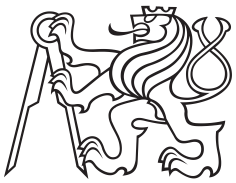
Bachelor Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Convolutional Neural Networks with Local Context Masks

Jakub Paplhám

Supervisor: Ing. Tomáš Petříček, Ph.D.
Field of study: Cybernetics and Robotics
August 2020

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor Ing. Tomáš Petříček, Ph.D. for the continuous support, for his patience and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

I gratefully acknowledge the support of Center for Machine Perception (CMP) and Research Center for Informatics (RCI) with providing resources used for this project. I would also like to express my appreciation of Ing. Tomáš Petříček, Ph.D. arranging access to aforementioned resources.

My sincere thanks also goes to doc. Ing. Tomáš Svoboda, Ph.D. and doc. Ing. Karel Zimmermann, Ph.D. for introducing me to the topic of machine learning.

Lastly I would like to express my deepest appreciation of my family and friends.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Abstract

This work proposes, implements and evaluates modified convolution operation within CNNs which explicitly considers additional context when processing data. This is done through a mechanism where local context corresponds to a local real-valued convolution mask which assigns higher weights to contextually close data and lower weights to contextually distant data.

Basics of machine learning are briefly reviewed and theoretical background for the method is given. Networks with and without the novel operation are trained with a simple, fair training process, utilizing identical training parameters. Many expectations of potential benefits are refuted, some are confirmed.

Traditional convolution is found to achieve better results in the experiments than the proposed novel operation.

Keywords:

Convolutional neural networks, spatially variant convolution, feature fusion, depth completion, semantic segmentation, KITTI, Cityscapes

Supervisor:

Ing. Tomáš Petříček, Ph.D.
Department of Cybernetics,
Faculty of Electrical Engineering,
Czech Technical University in Prague

Abstrakt

Práce navrhuje, implementuje a vyhodnocuje užití modifikované konvoluční operace v konvolučních neuronových sítích, která explicitně uvažuje dodatečný kontext při zpracování dat. Toho je docíleno mechanismem, ve kterém lokální kontext nabývá formy lokální konvoluční masky s reálnými hodnotami, která přiděluje vyšší váhy kontextuálně blízkým datům a nižší váhy kontextuálně vzdáleným datům.

Text shrnuje základy strojového učení a stručně probírá teoretický základ navrhované metody. Jsou natrénovány sítě využívající tradiční a novou operaci pomocí jednoduchého, férového trénovacího procesu. Předpoklady mnoha prospěšných vlastností operace jsou vyvráceny, některé předpoklady jsou potvrzeny.

Tradiční konvoluce v experimentech dosahuje lepších výsledků než nově navržená operace.

Klíčová slova:

Konvoluční neuronové sítě, prostorově proměnná konvoluce, doplnění hloubky, fúze charakteristik, sémantická segmentace, KITTI, Cityscapes

Překlad názvu:

Konvoluční neuronové sítě s lokálními kontextovými maskami

Contents

Figures

Tables

Chapter 1

Introduction

Deep convolutional neural networks (DCNNs) registered large success in processing of data organized in grid-like topology such as discretized audio signals or images. Accomplishments of these networks stem largely from local dependence of the data, where elements which are spatially close in the grid are closely contextually related. This property makes the data ideal for the convolution operation. In most cases the local dependence property holds true: consecutive audio samples are closely related in time, nearby pixels likely share the same color or belong to the same object. However under certain conditions, such as in images at object boundaries, the dependence is broken and spatially close pixels can be contextually distant.

This work proposes and evaluates modification of the standard convolution operation which aims to resolve this issue by explicitly considering additional context. This is done through a mechanism, where local context corresponds to a local real-valued convolution mask which assigns higher weights to contextually close grid elements and lower weights to contextually distant grid elements.

The resulting convolution is spatially variant and context dependent. The operation is implemented within neural networks with state-of-the-art architectures and the approach is evaluated on depth completion and semantic segmentation tasks. The operation allows fusion of intermediate representations of processed data with additional context source data. It also follows a recent trend in the depth completion task to exploit additional information, namely images, surface normals or masks of measurement validity.

Similar approaches were used on a limited number of tasks with state-of-the-art results. This work evaluates the approach on multiple tasks and compares the findings with similar research.

Chapter 2 provides theoretical background for the research. Deep learning and its history are reviewed. Convolutional neural networks are introduced and the convolution operation is examined. Modified, spatially variant convolutions are reviewed. The discussed applications of depth completion and semantic segmentation are defined and the main hypothesis of this work is formulated.

Chapter 3 contains information about the research methods. The proposed convolution operation is explained in detail, the network architecture is described and details about the experiment setup are given.

Chapter 4 summarizes the experimental results. Evaluated metrics of the approach on depth completion are on semantic segmentation are listed.

Chapter 5 discusses the results and mentions potential limitations of the work. Suggestions for future research are also given.

Chapter 6 contains the final summary and concludes the work.

Chapter 2

Convolutional Neural Networks and Theoretical Background

2.1 Introduction to Deep Learning

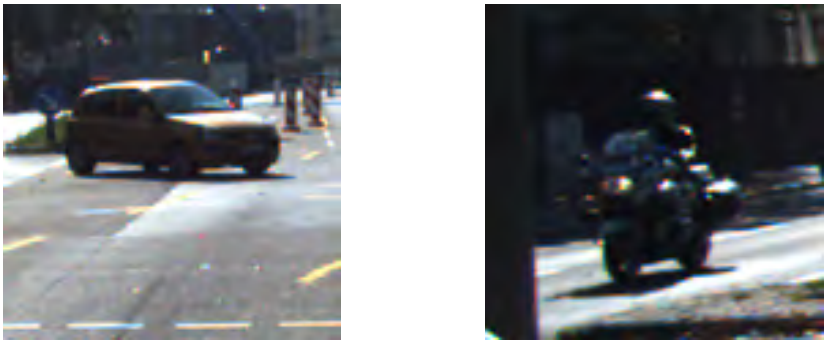
Since the inception of artificial intelligence (AI) as a field of scientific research, following the invention of the programmable digital computer, it was obvious that computers were fully capable of solving problems which were difficult to grasp for humans, but simple to describe formally. Problems which involved a limited set of rules and options, such as playing chess, were shown to be theoretically solvable purely through use of search algorithms and statistical analysis. However problems which are easy for humans, but do not have a formal definition, actions which seem intuitive or even automatic to us, proved to be a difficult task for computers.

The solution was machine learning; allowing the computers to learn. A central idea of AI is the concept of a *rational agent*. An *agent* is anything that perceives its *environment* through *sensors* and acts upon the environment through *actuators*. A human agent has eyes, ears, sense of touch, etc. for sensors, limbs and muscles for actuators and acts upon his surroundings. A robotic agent might have cameras and Light Detection And Ranging (LIDAR) for sensors and motors for actuators. A software agent might receive files as sensory inputs and act upon computer memory by writing new data to it.

The question then becomes: how does an agent process its sensory inputs? The amount of sensory data can be extremely large, yet contain minimal amount of relevant information. Consider an agent which is supposed to differentiate between cars and motorcycles as displayed on figure 2.1. Now compare the difficulty of the task, depending on whether its sensory input is an image of the vehicle or a number of its wheels.

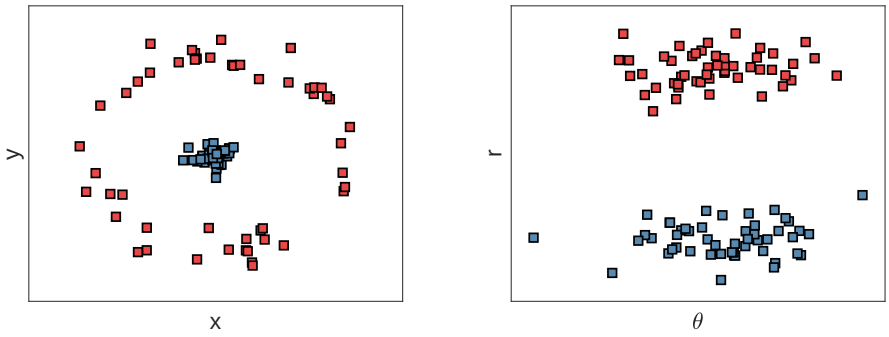
If the picture is appropriately preprocessed, the task is much simpler. The information can be encoded into variables such as number of wheels, formally *features*, which describe knowledge of the real world condensed into relevant information.

With statistical and probabilistic models, such as linear regression or naive Bayes, one can achieve impressive results as long as proper features are used. The difficulty comes from designing these features, as they must not only be



(a) : Car category **(b) :** Motorcycle category

Figure 2.1: Images taken from KITTI Depth Completion Benchmark [1].



(a) : Cartesian coordinates **(b) :** Polar coordinates

Figure 2.2: Difference between two representations of the same data.

relevant, but also appropriately represented for the given task. An epitome of this is shown on figure 2.2, where red and blue squares represent two categories of inputs. When the data is viewed in cartesian coordinates 2.2a, a linear model would be unable to differentiate between the categories. When the data is viewed in polar coordinates 2.2b a linear model is sufficient.

The question then becomes: how to define these features and ensure their validity for the given task? One might point out that it is possible to give these probabilistic models all the available data - RGB images for example - but individual pixels only hold negligible amount of information related to the final result and this approach would require unrealistically large amounts of training data in order to learn correct dependencies and correlations. Feature extraction is therefore necessary in most AI tasks, yet it remains a complicated issue which is still unsolved. A promising idea which shows great results is to allow the AI to learn both: a) mapping from features to output, b) mapping input data to features.

It can however be quite difficult for a computer to extract high-level features directly from raw data. Take for example a network which decides what advertisement to display to someone, based purely on their picture. It would certainly be helpful for the network to know the person's age and

gender, but extracting these features from an RGB image is quite difficult. Deep learning solves this issue by consecutively deducing more and more complicated features from simpler ones, starting with detection of edges, colors and shapes, then deducing textures etc. In the given example, the network might first detect hair color, hair length, presence of facial hair and the amount of wrinkles. From these features, it can then estimate the age and gender.

Typical example of a deep learning network is a feedforward network or a multilayer perceptron (MLP). These models are represented by a single function which is a composition of many simple functions. The name feedforward network comes from the process of feeding output from one function as an input to the next without any backwards connections. The whole network results in a single mapping which becomes progressively more complex as more functions or layers of functions are added.

Many network models more complex than MLP exist. Some combine the learned mapping with probabilistic models. Others expand upon the feedforward idea by taking the output of the network and feeding it back to the network as input.

A type of neural networks which achieved great success in computer vision tasks is the convolutional network. Brief history of convolutional networks, explanation of how they work and motivations which led to their current form are reviewed in the following chapter.

2.2 Brief History of Neural Networks

Earliest predecessors of current deep learning networks were simple linear models inspired by discoveries made during study of the nervous system. These models received n input values $x_1, \ldots, x_n$ and mapped them to an output y, where

$$y = f(\mathbf{x}, \mathbf{w}) = x_1 w_1 + \ldots + x_n w_n. \tag{2.1}$$

Scalars $w_1, \ldots, w_n$ served as a set of model parameters or weights which had to be precisely configured in order to create a function which best maps inputs data to desired outputs.

A simple brain inspired linear model emerged in 1943, the McCulloch-Pitts Neuron[32]. Its basic idea was that the activation of a neuron directly represents validity of a proposition about the observed world. Initially, elementary propositions are measured by sensors. Neurons which are connected to these sensors deduce more refined propositions. McCulloch and Pitts [32] argued that connections between neurons can represent logical *not, and* and *or* functions. Therefore a sufficiently large network of interconnected neurons could learn to represent every conceivable finite logical combination of the initial elementary propositions.

Output of their model was represented as a scalar value $y = f(\mathbf{x}, \mathbf{w})$ and differentiated two categories of inputs, determined by y being positive or

negative. The weights $w_1, \ldots, w_n$ of the model had to be manually set by a human operator. Thus the model was not able to learn by itself, but it was capable of being taught. The first model which could truly learn by modifying its weights was the perceptron developed by Rosenblatt in 1958 [38][39].

Linear models have many limitations, as was shown by Minsky and Papert in 1969 [33]. Most predominant of these limitations is the inability to learn logical *xor* function $y = f(\mathbf{x}, \mathbf{w})$, where

$$f([0,1],\mathbf{w}) = 1, \quad f([1,0],\mathbf{w}) = 1, \quad f([0,0],\mathbf{w}) = 0, \quad f([1,1],\mathbf{w}) = 0. \tag{2.2}$$

After the discovery of these limitations, research of neural networks saw its first major decline in popularity. Many concepts important for the field of deep learning were uncovered during this hiatus, such as Fukushima's self-organizing neural network proposed in 1975, the Cognitron [10]. A simplified version of this concept later led to development of a neuron activation model which is currently the most widely used elementary unit in neural networks, the rectified linear unit.

Major resurgence in popularity of neural networks was seen in the 1980s. A movement called connectionism arose in the field of cognitive science and found ground upon earlier implementations of neural networks. The basic principle of connectionism is that a sufficiently large number of basic computational units connected together can accomplish intelligent behavior. A vital achievement of the movement was demonstration and successful use of the back-propagation algorithm in 1986 by Rumelhart *et al.* [40] and in 1987 by LeCun [31] to train deep neural networks. The back-propagation algorithm remains prevalent in training of neural networks to this day.

Research of neural networks showed its second decline in popularity in the mid-1990s. AI based businesses made unrealistically ambitious claims to attract investors and later found it impossible to fulfill their expectations. Concurrently, other fields of machine learning showed great results. Combination of these two factors shifted attention of researchers away from neural networks for over 10 years.

That is not to say that development of neural networks completely stopped. The Canadian Institute for Advanced Research (CIFAR), well known for its classification benchmarks, continued to fund neural network research during this time and grounded it as a multi-disciplinary subject.

The third and current wave of neural network research began with breakthroughs in 2006, when Hinton *et al.* [19] showed a network training strategy which could be used to train much deeper neural networks than before. The strategy proved functional for many kinds of neural networks and brought spotlight to the theoretical importance of network depth.

Deep neural networks quickly outperformed other machine learning technologies and reinvigorated interest in their study. Furthermore, graphics processing unit (GPU) development enabled great increase in speed of deep neural network training and reduction in its cost.

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a special kind of neural networks which process data organized in a grid-like topology. Audio signals and discretized time-series are an example of 1-D grid data. Images can be thought of as 2-D grids and voxels as 3-D grids. The name of convolutional networks comes from their use of mathematical convolution. Convolution is a linear mathematical operation with many practical applications. Its use in neural networks stems from research done in the 1980s, however the community disagrees on a specific origin. LeCun *et al.* [30] are often credited with the invention of convolutional networks, thanks to their research on handwritten digit recognition published in 1989. The origin could be traced to even earlier work of Fukushima [11], published in 1980, expanding upon his previous research [10].

Mathematical convolution and motivation for its use in neural networks, specifically in the field of computer vision, are reviewed in 2.3.1 and 2.3.2.

2.3.1 Mathematical Convolution

Mathematical convolution is an operation on two functions which produces a new function. Convolution of two functions $x(t), w(t) : \mathbb{R} \longrightarrow \mathbb{R}$ in its continuous form is defined as

$$(x * w)(t) = \int_{-\infty}^{\infty} x(\tau) \cdot w(t-\tau) d\tau, \quad t, \tau \in \mathbb{R}. \tag{2.3}$$

For the purposes of this text, function $x(t)$ will be referred to as *input* and function $w(t)$ as *kernel*. Discrete form of convolution is defined ase

$$(x * w)(t) = \sum_{\tau=-\infty}^{\infty} x(\tau) \cdot w(t-\tau), \quad t, \tau \in \mathbb{Z}. \tag{2.4}$$

Extended to n-dimensional case, discrete convolution is an operation on functions $X(\mathbf{u}), W(\mathbf{u}) : \mathbb{Z}^n \longrightarrow \mathbb{R}$, defined as

$$(X * W)(\mathbf{u}) = \sum_{v_1} \cdots \sum_{v_n} X(\mathbf{v}) \cdot W(\mathbf{u} - \mathbf{v}), \quad \mathbf{u}, \mathbf{v} \in \mathbb{Z}^n. \tag{2.5}$$

Specifically for 2-D case, where $\mathbf{u} = \begin{bmatrix} i \\ j \end{bmatrix}$, $\mathbf{v} = \begin{bmatrix} k \\ l \end{bmatrix}$,

$$(X * W)(i, j) = \sum_{k} \sum_{l} X(k, l) \cdot W(i-k, j-l). \tag{2.6}$$

Functions X, W can be written in a matrix form, where $X(i, j) = X_{i,j}$ is a value on $i-th$ row and in $j-th$ column of matrix X. Similarly for $W(k, l)$ and indices k, l. Example of the matrix form is shown on figure 2.3. It is customary to regard values outside of indices in the matrix as equal to zero. It is also customary to only compute the convolution for indices with said values.

$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$
$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$
$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$
$X_{4,1}$	$X_{4,2}$	$X_{4,3}$	$X_{4,4}$

Figure 2.3: Function $X(\mathbf{u}) : \mathbb{Z}^2 \longrightarrow \mathbb{R}$ written in a matrix form.

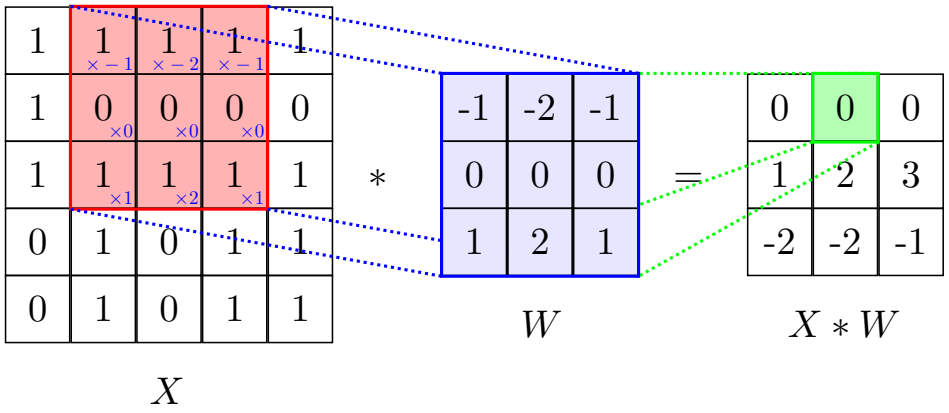


Figure 2.4: Example of a 2-D discrete convolution using input matrix X and Sobel-Feldman 3×3 kernel W to detect vertical edges. The output is limited to only valid kernel positions.

2.3.2 Motivation

Use of convolutional neural networks in computer vision tasks was motivated by findings made in the field of neurophysiology in the 1950s. First, the principle of operation of a neuron was demystified, when A. Hodgkin and A. Huxley [20, 21, 22, 23, 24] measured electrical changes within *Doryteuthis pealeii* giant axon in response to external stimulus. They constructed a mathematical model of the neuron and earned the Nobel Prize in Physiology and Medicine in 1963 for their work. The complicated Hodgkin-Huxley model (output displayed on figure 2.5) inspired creation of simplified neuron models, which though lacking certain biological features were easier to simulate and analyze. These models followed the form of

$$o = f(\sum_i w_i \cdot v_i), \tag{2.7}$$

where f is a nonlinear activation function, traditionally rectified linear unit (ReLU) or sigmoid, v_i is signal from i-th pre-synaptic neuron and w_i is strength of the signal.

A noticeable difference between the Hodgkin-Huxley and a simplified model was, that a model described by equation (2.7) summed up inputs from all pre-synaptic neurons and operated on the sum as a whole. It also lacked any

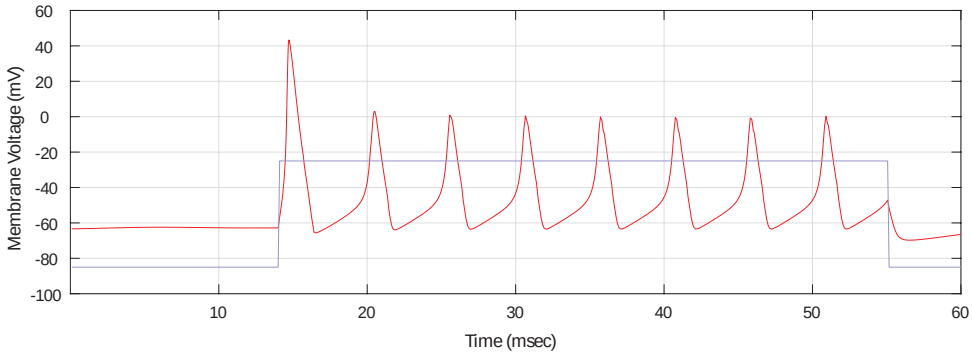


Figure 2.5: Simulation of Hodgkin-Huxley model generated with MATLAB.

dynamic term and therefore could not display output spiking as it can be seen on figure 2.5. This problem could however be circumvented by interpreting the output o of equation (2.7) as a frequency of the spiking, rather than voltage. With this change, humans could now create reasonable neuron simulations.

Second, D. Hubel and T. Wiesel revealed how individual neurons were connected and explained how they processed visual information. Through the use of D. Hubel's invention, the modern tungsten microelectrode, they were able to research striate cortex in paralyzed cats. In 1959 they discovered certain selectivity and columnar organization in the cortex [26] and that nearby neurons process information from nearby visual fields. In 1962 [25], they observed that neurons responded differently to changing light intensity and simple shapes such as rectangles and lines, especially when varying angles of their inclination. They received the Nobel Prize in Physiology and Medicine in 1981 for their work.

Their research proved that neurons are only sparsely connected with spatially restricted interactions. This made convolution an ideal candidate to simulate neuron connections, as the operation was shown to possess many convenient features. When utilizing a relatively small kernel, convolutional connections between neurons were sparse and spatially restricted. Important to detection tasks, convolution was also equivariant to translation. Last but not least, learnt convolution kernels could be used to process inputs of any size. This ultimately led to convolution replacing fully connected layers (FCLs) which were more computationally and memory intensive.

Comparison: Convolution and Fully Connected Layer

When first observing convolutional layer and FCL, 1-D case shown on figures 2.6 and 2.7 respectively, one might believe the layers to be very similar. It is critical, however, to discern their differences. As can be seen, convolutional layer has restricted perceptive field while FCL does not. This naturally leads to FCL having more learnable parameters and requiring more memory and computations. One could be concerned that the inherently small receptive field of convolutions may result in poor network performance. Figure 2.10 shows this not to be the case, as the deeper a neuron is in the network,

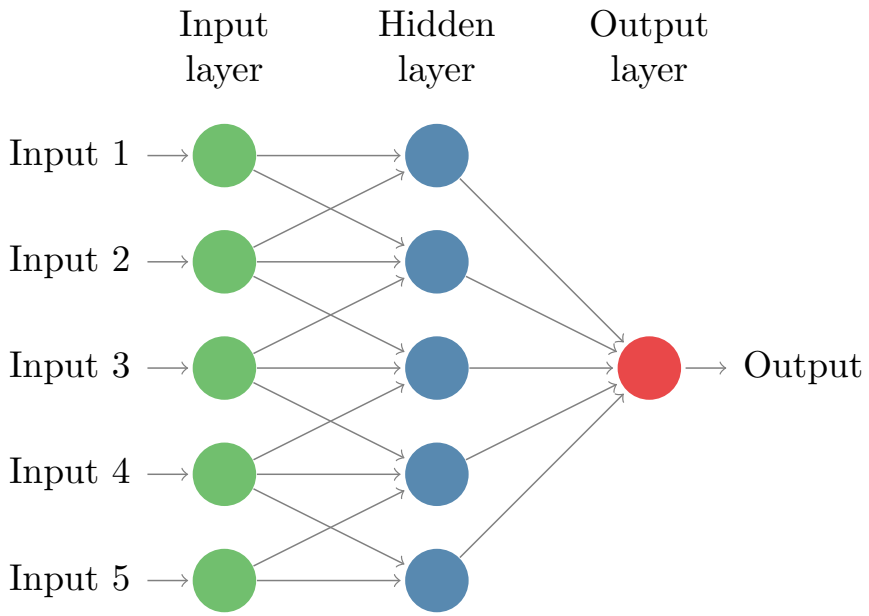


Figure 2.6: Example of a simple neural network for processing 1-D grid data with one convolutional layer.

the larger its receptive field becomes. This structure of connections further mimics the columnar organization of neurons discovered by Hubel and Wiesel.

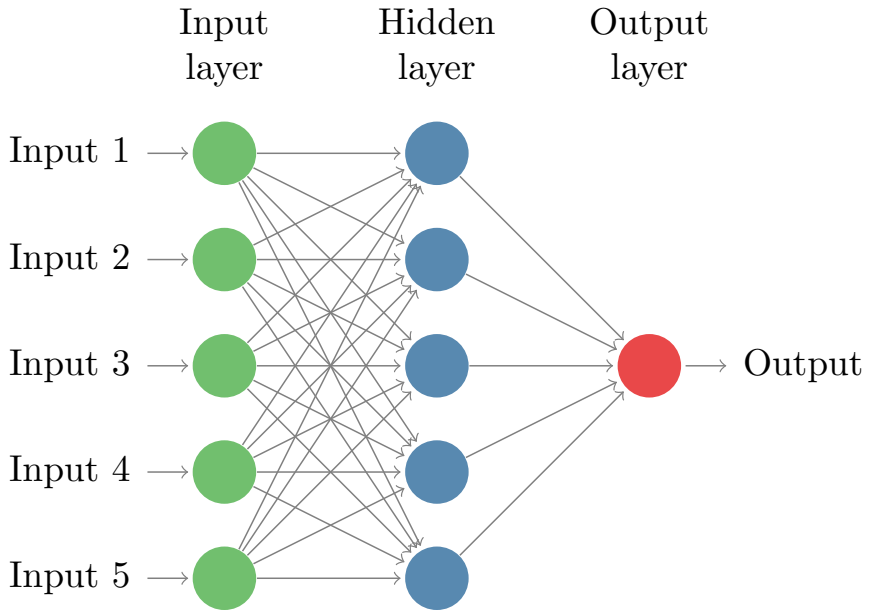


Figure 2.7: Example of a simple neural network for processing 1-D grid data with one fully connected layer.

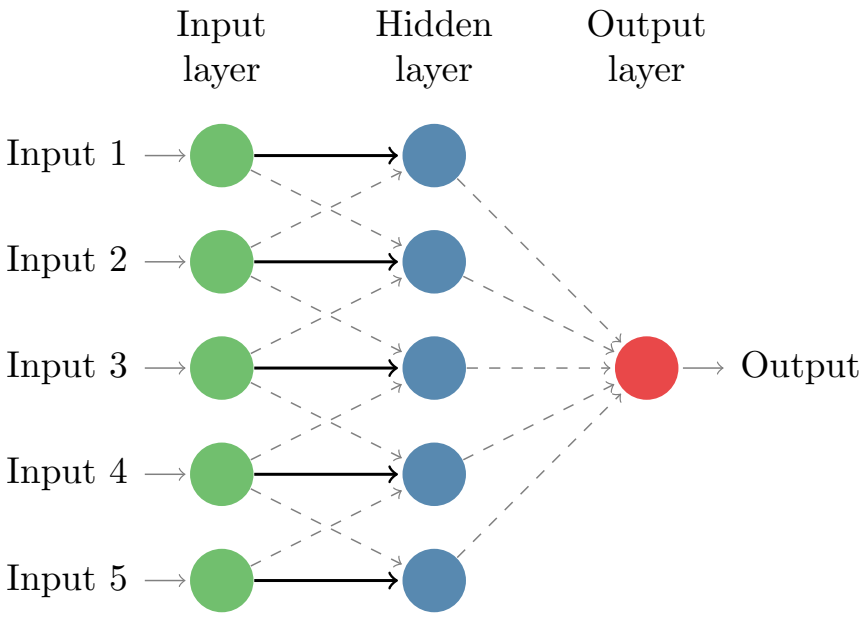


Figure 2.8: A single parameter is shared by multiple computations in a convolutional layer.

Convolution can also be used on nearly any size of data while FCLs only support one fixed size. It has been mentioned that FCLs are also more computationally and memory intensive than convolutional layers. This claim seems reasonable, as FCLs have more connections and therefore require more parameters, however that is not the sole reason. A large increase in memory efficiency of convolutional layers also arises from parameter sharing, as the same kernel is used for every computation in a convolutional layer. This fact is shown on figure 2.8, where bold arrows represent an identical parameter. Observing FCL on figure 2.9, each parameter is only used once.

The following section briefly reviews other operations, beside convolution, which are often utilized in convolutional networks.

2.3.3 Convolution Arithmetic

Convolution and Cross-Correlation

Many deep-learning libraries surprisingly do not include genuine convolution implementations. They instead utilize operation known as cross-correlation, which is identical to convolution, but with reversed kernels. One could argue that this makes cross-correlation non-commutative and he/she would be correct. Convolutional neural networks, however, contain many operations which are not commutative and therefore the loss of this property is not an issue. Additionally, since the kernels are fully learned, the fact that they are reversed is not an issue either. For those reasons, expressions *convolution* and *cross-correlation* are often used interchangeably while discussing neural

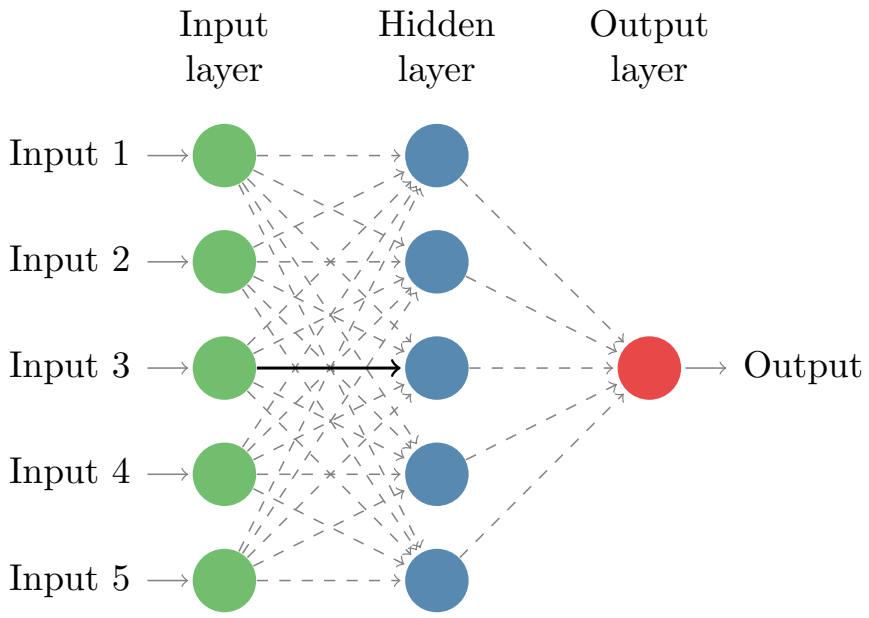


Figure 2.9: Parameters in a fully connected layer are used only once.

networks. This text follows the same practice and will often default to the use of convolution, when referring to cross-correlation.

Deep learning libraries often accept additional parameters for convolution, including, but not limited to, dilation, stride and padding. These parameters, alongside data representation within a neural network, are briefly reviewed and demonstrated on a 2-D discrete case in the following sections. For an in depth understanding refer to [9] which influenced section 2.3.3 heavily.

2-D Discrete Convolution, Channels, Batches

Previously, 1-D case was used to illustrate how convolutional layers are connected within a neural network. Most data in computer vision tasks however comes in form of 2-D grids, such as RGB images or depth maps. An example of 2-D discrete convolution is shown on figure 2.11.

Naturally, both the input data and the kernel here are 2 dimensional matrices. Thus each element in such data can be indexed with 2 numbers. Upon examining RGB images however, one might argue that 3 numbers are needed to index an element, as not only does a position of a pixel have to be specified, but also a color channel - red, green or blue. Therefore an RGB image can not be represented as a matrix, but can be represented as multiple matrices, each color channel represented with a separate one. Another way to represent an image is as a tensor which, though mathematically incorrect, can be thought of as generalized n-D matrix.

RGB images can still be processed with 2-D convolution, even though they can not be represented with a matrix, only the convolution kernel must have the same number of channels as the input data. An example of convolution

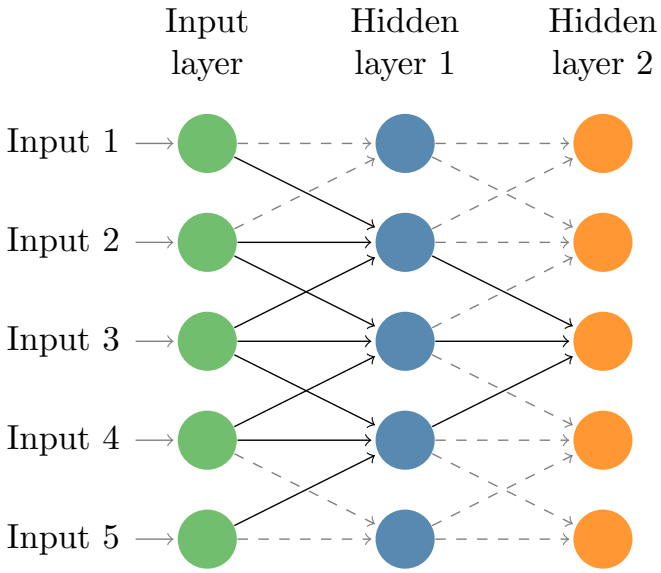


Figure 2.10: Example of a simple neural network for processing 1-D grid data with two convolutional layers, showcasing increased receptive field of view of deeper neurons.

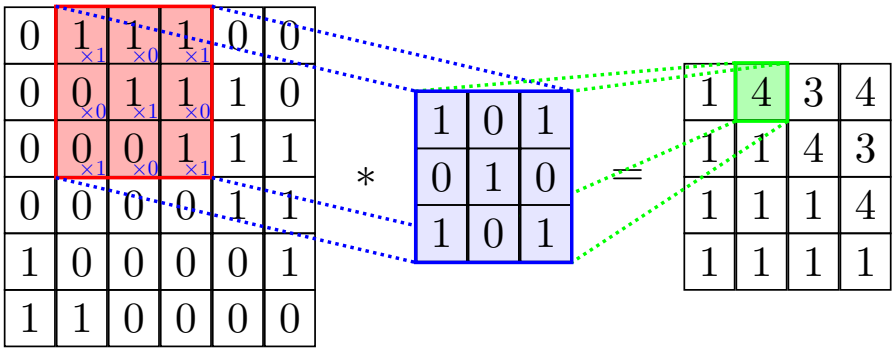


Figure 2.11: Example of a default discrete 2-D convolution with one input channel.

with multiple channels is shown on figure 2.12. This convolution reduces the number of channels from 3 to 1, but when multiple kernels are provided, the number of output channels can be arbitrary.

This is indeed often the case, as it is desirable to process the same data with multiple different kernels. One kernel might detect vertical edges, another one might detect horizontal edges, etc. An example of convolution with multiple kernels is shown on figure 2.13.

For completeness it must also be mentioned that deep learning libraries typically do not use 3 dimensional tensors, while processing 2-D grid data, but use 4 dimensional tensors instead. The last added dimension comes from parallel processing of multiple grids.

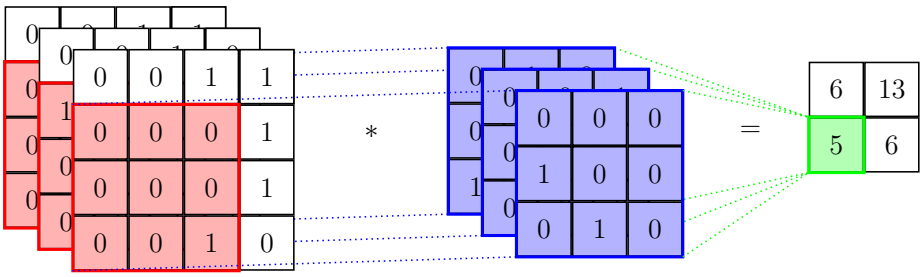


Figure 2.12: Example of a default discrete 2-D convolution with three input channels and one output channel.

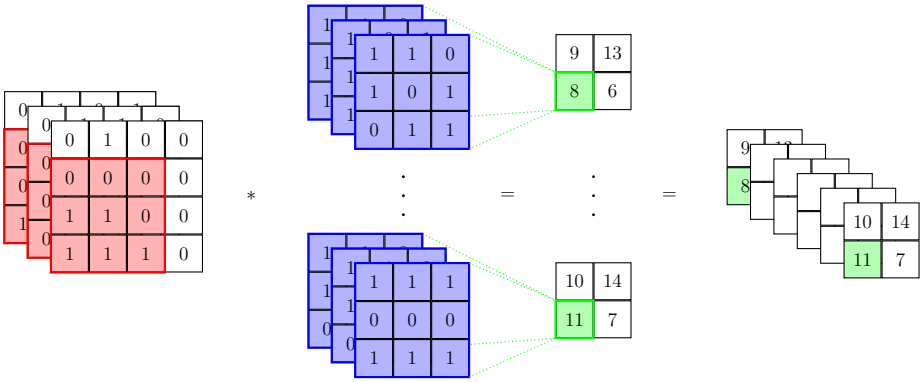


Figure 2.13: Example of a default discrete 2-D convolution with three input channels and multiple output channels.

Neural network computations are usually carried out on graphics processing units (GPUs), leveraging parallelism in their architecture and computing convolutions across the whole data at the same time. This means that the same convolution is being computed for the upper left corner, the bottom right corner, as well as all other remaining parts of the grid, simultaneously. With recent increases in memory size of commercially available GPUs, it became possible to process multiple grids at the same time, and therefore another dimension was added to differentiate between the grids.

A set of grids processed in parallel is called a batch, and is indexed within a tensor as [batch size, channels, height, width]. An important property of convolution is that convolutional layers do not interact across axes, and as such the change of stride, dilation and padding along axis j only affects the output size of axis j. Therefore, only the simplified case, where kernel size k, dilation rate d, stride s and padding p are identical between width and height axes, will be outlined. The resulting properties however still generalize to n-D case with distinct values of k, d, s, p for each axis.

Dilation

The dilation parameter d allows the convolution to attain larger receptive field without needing to increase the amount of parameters. As can be seen on figure 2.10, stacked layers of regular convolution allow the receptive field

of deeper neurons to linearly grow in size. Layers of dilated (or "atrous") convolution allow the receptive field to grow exponentially.

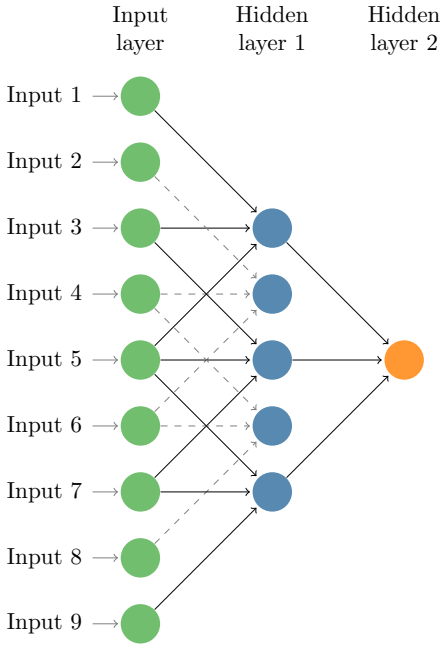
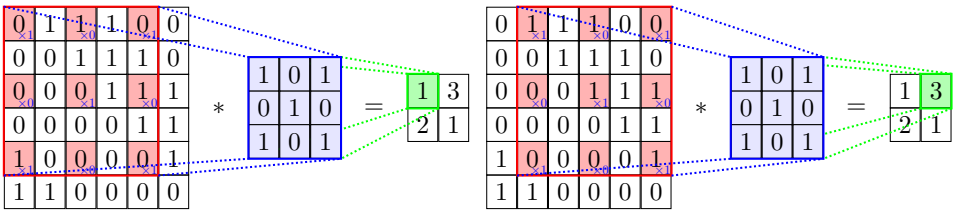


Figure 2.14: Example of a simple neural network for processing 1-D grid data with two convolutional layers with dilation $d = 2$, showcasing exponential receptive field growth of deeper neurons.

Dilated convolution enlarges the provided kernel by inserting spaces between the kernel elements, effectively making the kernel size equal $\bar{k} = k + (k-1)(d-1)$, where k is the original kernel size. Usually, there are $d-1$ spaces inserted between kernel elements, therefore regular convolution corresponds to $d = 1$. A 1-D case where $d = 2$ can be seen on figure 2.14. Notice the spaces between connections and the resulting enlarged field of view compared to figure 2.10. An example of 2-D case where $d = 2$, limited to two output positions, can be seen on figure 2.15.



(a) : Upper left corner. **(b) :** Upper right corner.

Figure 2.15: Example of a discrete 2-D convolution with dilation parameter $d = 2$, visualized for two output positions.

Stride

The stride parameter s allows the convolution to downsample the input. It specifies by how much to move the convolution kernel before computing another element of output. Typically, $s-1$ valid positions of kernel over input data are skipped in both directions, but the horizontal and vertical strides can potentially be distinct. This is equivalent to downsampling the output of a regular convolution by deleting certain elements, as is shown for $s=2$ on figure 2.16. Regular convolution corresponds to $s=1$. A complete example of 2-D case where $s=2$, can be seen on figure 2.17.

Stride $s>1$ is often used to downsample the input, while increasing the number of channels. This allows the network to distinguish larger number of features, though with reduced resolution, without requiring unreasonable amounts of memory.

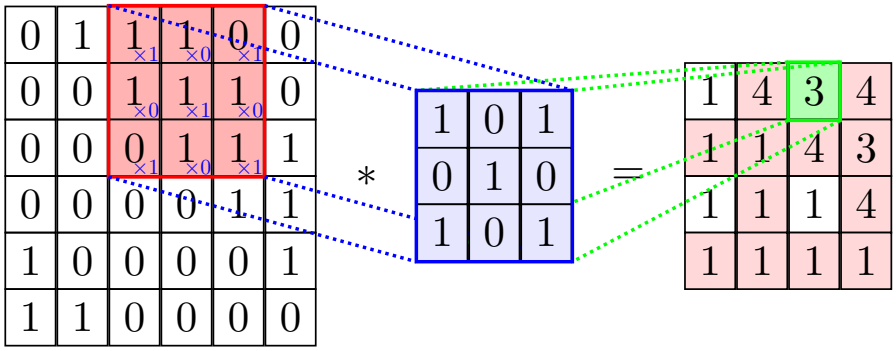
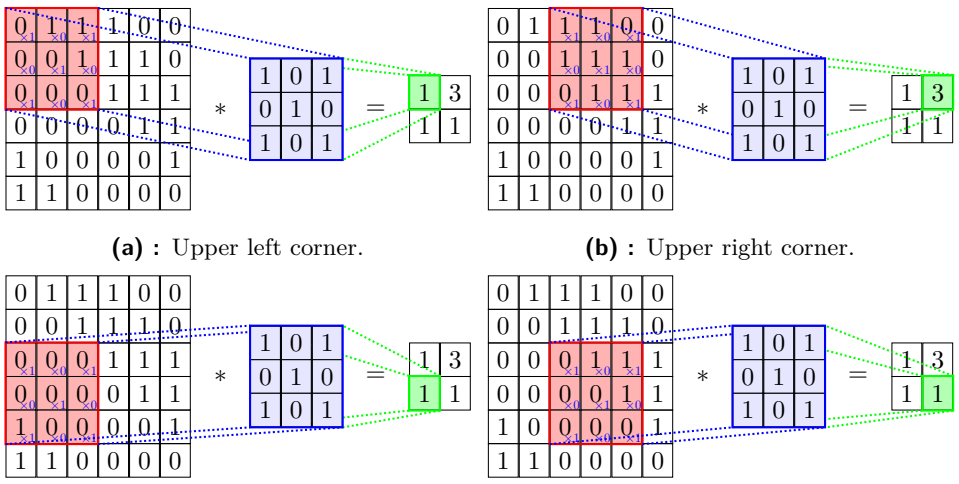


Figure 2.16: Example of a regular discrete 2-D convolution with stride parameter $s=1$ with highlighted elements, which would not be computed for stride $s=2$.



(a) : Upper left corner.

(b) : Upper right corner.

(c) : Bottom left corner.

(d) : Bottom right corner.

Figure 2.17: Example of a discrete 2-D convolution with stride parameter $s=2$, showing all viable positions of kernel over input data to produce output.

Padding

The padding parameter p specifies the number of columns and rows which should be added to input data, before computing the desired convolution. The number of added columns and rows can potentially be distinct. The elements of new rows/columns are usually zero valued.

When convolution kernel of an odd size k is used, padding $p = (k-1)/2$ ensures that the input and output height/width are identical. Such case is shown for $k = 3$ on figure 2.18.

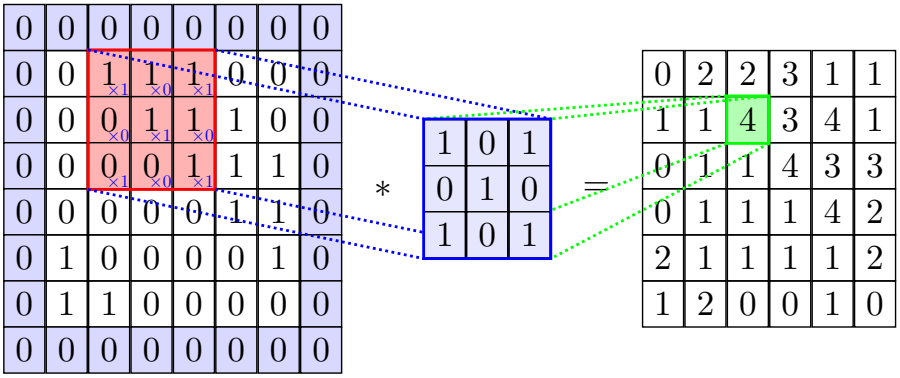


Figure 2.18: Example of a discrete 2-D convolution with padding parameter $p = 1$ and kernel size $k = 3$, resulting in identical size of input and output data.

With all parameters k, d, s, p reviewed, it is possible to write down the following equation which describes the relationship between those parameters, the input size and the output size:

$$o = \left\lfloor \frac{i + 2p - k - (k-1)(d-1)}{s} \right\rfloor + 1, \tag{2.8}$$

where o, i are the output and input sizes along axis j and k, d, s, p are kernel size, dilation rate, stride and padding along axis j. It should also be reminded, that $\lfloor x \rfloor$ denotes integer floor function on x.

2.3.4 Non-Convolutional Layers

In 2.3.3, it has been mentioned that convolutional neural networks comprise multitude of layers which are not commutative. This permits the use of cross-correlation instead of convolution. Convolutional and fully connected layers were introduced in 2.3.2. Other layers which are widely used within convolutional neural networks are briefly reviewed in the following sections.

Activation functions

Advancements in neurobiology in the 20th century allowed scientists to demystify the function of a neuron and subsequently create a simplified model

described by equation (2.7). The equation contains term $\sum_i w_i \cdot v_i$ which models signals from pre-synaptic neurons and can be implemented through a FCL or a convolutional layer, depending on the desired neuron connections.

The equation also contains an unspecified non-linear function f. Let us explore why f must not be linear. Writing an expression which is a composite function of two equations (2.7), we obtain

$$o = f\left(\sum_j f\left(\sum_i w_{j,i} \cdot v_{j,i}\right) \cdot u_j\right), \tag{2.9}$$

where f is an activation function, $v_{j,i}$ is signal from i-th neuron in first layer to j-th neuron in second layer, $w_{j,i}$ is strength of the signal $v_{j,i}$ and u_j is strength of signal from j-th neuron in second layer to a neuron in third layer. The equation (2.9) is a plain extension of equation (2.7) for one additional neuron layer. It should be noted, considering no backwards neuron connections, that for linear f, the whole equation (2.9) is a linear combination of $v_{j,i}$. Even with more layers added, the expression would indeed still remain linear and therefore the network would exhibit shortcoming mentioned in section 2.2.

This fact raises the necessity of f to introduce nonlinearity to the equation. Functions used to model this nonlinearity are called activation functions and they represent the amount of neuron activation based on pre-synaptic signals. Brief review of the most popular activation functions follows.

Logistic Sigmoid

- Zero gradient when saturated.
- Computationally inefficient.
- Positive value bias.

A function which originally registered success as a neuron activation approximation, was the logistic sigmoid function, shown on figure 2.19a:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.10}$$

The function is clearly non-linear and seems to possess the benefit of saturating the neuron activation for large inputs. This intuitively seems advantageous, the activation can not approach infinity and has an upper limit. One must realize, however, that neural networks are trained through the back-propagation algorithm since its successful use in 1986 by Rumelhart *et al.* [40]. This optimization algorithm is gradient based, it computes gradients of learnable weights for each layer and uses them to iteratively update the weights.

Explanation of the algorithm is beyond the scope of this text, it should however be noted that when the gradient is equal to zero, no change in weights can occur. Unfortunately, the sigmoid clearly has zero gradient when saturated. Another drawback of the sigmoid is the range of possible values,

which is positive only. In section 2.3.2 it was mentioned that the neuron output should be interpreted as frequency of deviations in the neuron output magnitude instead of the magnitude itself. This again seems intuitively correct, as the frequency should not reach negative values, however it causes problems during the training process. The last major issue is that the function and its gradient are quite computationally intensive.

Hyperbolic Tangent

- Zero gradient when saturated.
- Computationally inefficient.

Hyperbolic tangent is a sigmoid type function, but centered around zero. Therefore it avoids one of the logistic curve problems, the positive value bias, but retains the remaining - diminishing gradient and computational intensity. It is shown on figure 2.19b and is defined as

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^{2x} - 1}{e^{2x} + 1}. \tag{2.11}$$

Rectified Linear Unit

- Zero gradient when saturated.
- Positive value bias.

The most popular activation function at the time of writing is the rectified linear unit (ReLU), shown on figure 2.19c and defined as

$$ReLU(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}. \tag{2.12}$$

Though the function is very simple and not representing neuron activation accurately, it makes up for its lack of complexity by being computationally inexpensive. Its use enables fast gradient computation for deep networks, speeding up the training process. ReLU however still suffers from zero gradient and zero activation for values of $x \leq 0$ and from positive value bias.

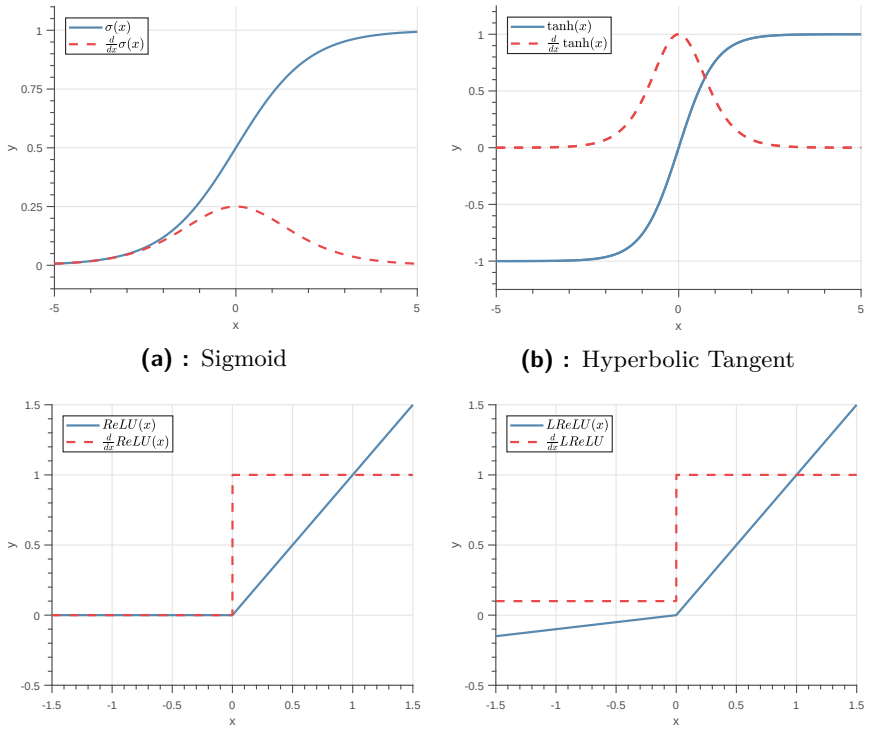
Leaky Rectified Linear Unit

- Slight positive value bias.

Leaky rectified linear unit (LReLU) is a slight variation of ReLU. It improves upon its predecessor by avoiding saturation, zero slope, for values of $x \leq 0$. It is shown on figure 2.19d and defined as

$$LReLU(x) = \begin{cases} ax & x \leq 0 \\ x & x > 0 \end{cases}, \tag{2.13}$$

where a is a decimal smaller than 1, usually $a = \frac{1}{10}$.



(a) : Sigmoid

(b) : Hyperbolic Tangent

(c) : Rectified Linear Unit

(d) : Leaky Rectified Linear Unit.

Figure 2.19: Visualization of the most common activation functions.

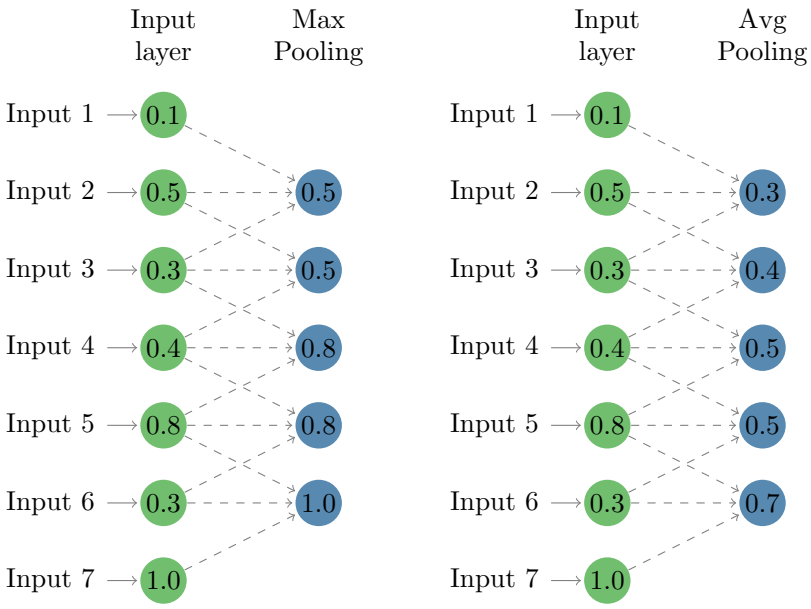
Pooling

Pooling layers are a class of layers which provide summary of neuron activations at a certain location. Different applications favor different types of pooling methods, the most common being: maximum activation, average activation and weighed average activation of spatially close neurons. Pooling methods are specified in advance and posses no learnable parameters.

Examples of pooling over maximum values (Max-Pool) and over average values (Avg-Pool) are shown on figures 2.20a, 2.20b respectively. Pooling layers can under certain circumstances be interpreted as convolution. Average pooling for example can be though of as convolution, where all kernel values are equal to $\frac{1}{k^n}$ for n-D grids. Max pooling can also be thought of as convolution, though spatially variant, and with *zero* valued kernel elements except for a single *one*. Because of this analogy between pooling and convolutional layers, pooling also supports kernel size k, dilation rate d, stride s and padding p parameters.

The reason why pooling layers in convolutional neural networks are used and work so well is debatable. One intrinsic property of pooling is that it makes the neural network invariant to small translations of input. Such property is beneficial for certain applications (classification tasks), but can be detrimental to others (detection and localization tasks). Another argument is that pooling allows the network to switch between smaller activation subnetworks and discriminate between redundant information, while being

computationally simple.



(a) : Pooling maximal values. **(b) :** Pooling average values.

Figure 2.20: Example of a pooling layer applied to 1-D grid data.

2.4 Spatially Variant Convolution

Previously, convolutional neural networks were introduced as a tool for processing grid organized data, where spatially close values are likely to be contextually close as well. For example, when processing an image, nearby pixels have a high probability of belonging to the same object. In certain application however, additional information about the data is provided and can be used to further modify how the data is processed. Take for example a 2-D depth map obtained from Light Detection And Ranging (LiDAR) scans, shown of figure 2.21, where only a small portion of the grid represents valid measurements. The remaining values represent areas which did not have their distance measured. Surely this information could be used to improve the network performance.

2.4.1 Sparse Convolution

J. Uhrig *et al.* [47] exploited the information whether data is valid or not on the large scale KITTI dataset [12] for the task of *depth completion*. They modified the convolution operation so that only valid values are considered and the output is scaled by the amount of observed valid values. Paraphrasing Uhrig *et al.* using notation consistent with this text:

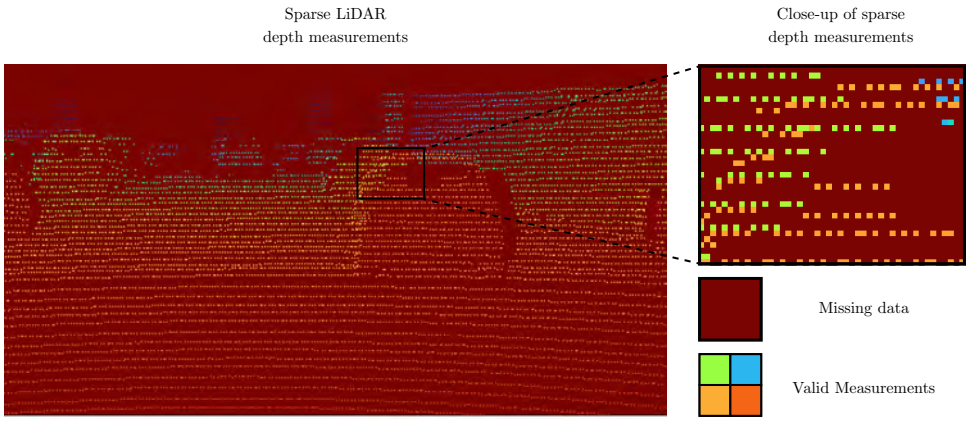


Figure 2.21: Example of a 2-D grid data of sparse LiDAR depth measurements. LiDAR data taken from KITTI Semantic Segmentation Benchmark [47].

Consider the case, where the 2-D grid input data $\mathbf{x} = \{x_{u,v}\}$ are only partially observed. Let $\mathbf{o} = \{o_{u,v}\}$ denote corresponding binary variables indicating if an input is observed ($o_{u,v} = 1$) or not ($o_{u,v} = 0$). The output f of a standard convolutional layer at indices corresponding to height u and width v using this notation is

$$f_{u,v}(\mathbf{x}) = b + \sum_{i,j=-\kappa}^{\kappa} x_{u+i,v+j} \cdot w_{i,j} \tag{2.14}$$

with kernel size $k = 2\kappa + 1$, kernel weights $\mathbf{w} = \{w_{i,j}\}$ and bias b. If the input comprises multiple channels, $x_{u,v}$ and $w_{i,j}$ represent vectors $\mathbf{x}_{u,v,c}$ and $\mathbf{w}_{i,j,c}$ with length c equal to the number of channels. The output for one output channel then is

$$f_{u,v}(\mathbf{x}) = b + \sum_{i,j=-\kappa}^{\kappa} \mathbf{x}_{u+i,v+j,c} \cdot \mathbf{w}_{i,j,c} = b + \sum_{i,j=-\kappa}^{\kappa} \sum_{c} x_{u+i,v+j,c} \cdot w_{i,j,c} \tag{2.15}$$

Making use of the same notation, the output of a sparse convolutional layer, proposed by Uhrig *et al.* [47], where only valid measurements are considered, is

$$f_{u,v}(\mathbf{x}, \mathbf{o}) = b + \frac{\sum_{i,j=-\kappa}^{\kappa} o_{u+i,v+j} \cdot x_{u+i,v+j} \cdot w_{i,j}}{\epsilon + \sum_{i,j=-\kappa}^{\kappa} o_{u+i,v+j}} = b + \frac{\text{num}}{\text{den}} \tag{2.16}$$

where small ϵ is added to denominator to avoid division by zero at filter locations where none of the input pixels are valid.

Visualization accompanying equation (2.16) is shown on figure 2.22 with $*_{sp.}$ denoting sparse convolution. The additional averaging over observed valid values makes the sparse convolution spatially variant and content dependent

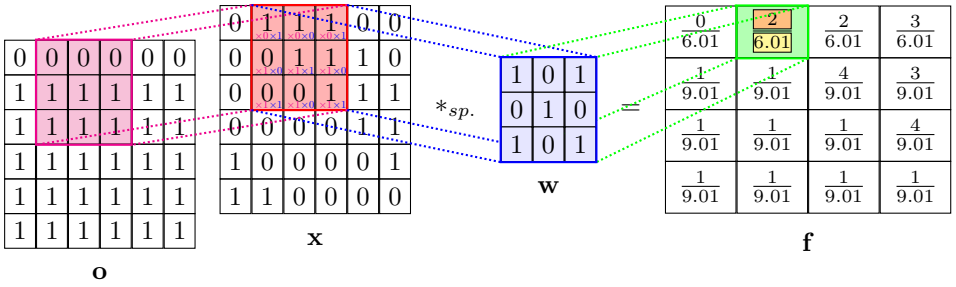
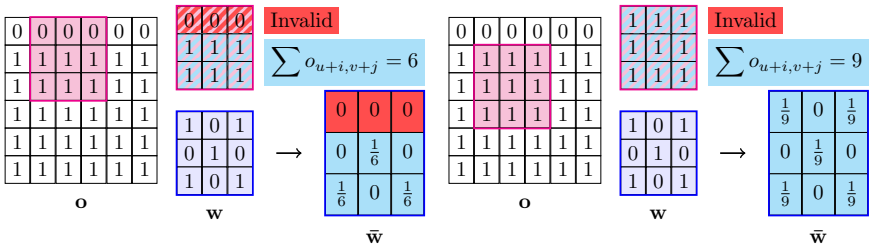


Figure 2.22: Example of a discrete 2-D sparse convolution with one input channel. Input and kernel weights are identical to regular convolution on figure 2.11, however first row of $\mathbf{x}$ is considered invalid, bias $b = 0$ and $\epsilon = 0.01$.

on the matrix $\mathbf{o}$. This means that different parts of the 2-D grid are convolved using a different kernel, based on context of whether the convolved values are valid or not. Figure 2.23 shows that the effective kernel weights $\bar{\mathbf{w}}$ indeed change depending on the position of kernel.



(a) : Position with 6 valid entries. **(b) :** Position with 9 valid entries.

Figure 2.23: Example of effective sparse convolution kernels $\bar{\mathbf{w}}$ being spatially variant based on position in $\mathbf{o}$ and global kernel $\mathbf{w}$.

2.4.2 Guided Convolution

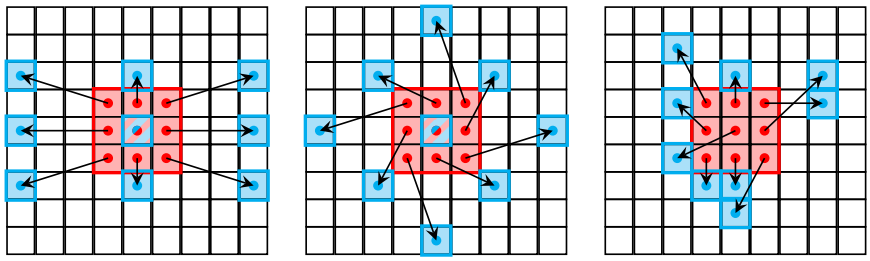
The same task, *depth completion*, was also pursued by J. Tang *et al.* [44], which explored the idea of using a separate neural network to generate spatially variant and content dependent kernels from RGB images with great success. They employed an encoded-decoder network to extract convolution kernels from an RGB image and used the kernels within another encoder-decoder network to process the sparse depth data.

The convolution uses no predetermined rules to obtain the kernels and all of its traits are fully learned. This method was introduced as an alternative to previously popular addition/concatenation of RGB image data with depth data and was shown to produce impressive results, ranking 1st at the KITTI depth completion leaderboard at the time of writing.

2.4.3 Deformable Convolution

Another approach to modifying generic convolution was explored by J. Dai *et al.* [7]. They do not alter the elements of the convolution, instead they generate offsets and use them to deform the regular sampling grid of a kernel. All the offsets are first obtained from channels within the network and then used to deform learned convolution kernels. This shape deformation allows the deformable convolution to describe geometric transformations such as rotation, scaling or other general transformations.

These offsets are local and therefore the convolution can learn to approximately adapt to distortions or camera lens properties.



(a) : Scaling. **(b) :** Rotation. **(c) :** Generic.

Figure 2.24: Example of offsets in discrete deformable 2-D convolution enabling kernels to learn geometric transformations. Position of original kernel is displayed in red. New positions of kernel elements are displayed in cyan.

2.4.4 Pixel-Adaptive Convolution

The last approach to modifying generic convolution, presented in this text, was explored by H. Su *et al.* [42]. They present convolution in which the filter weights are multiplied with a spatially varying kernel that depends on learnable, local pixel features. The spatially varying kernel is extracted through the use of a predefined function which measures difference in pixel features and outputs an appropriate weight - the closer the features, the larger the weight. Analogical approach was devised separately by the thesis supervisor and researched by the author and therefore is explained in detail in chapter 3.

2.5 Discussed Applications

CNNs were introduced as a special kind of neural networks which process data organized in grid-like topology - audio signals, images and voxels. Most common of these are images therefore such the most common CNN applications process 2-D data. The following sections cover two application areas of CNNs which were used to evaluate viability of the proposed convolution, with local masks based on context. They also provide brief literature review of recent development in those applications.

2.5.1 Depth Completion

Depth completion or depth inpainting is an open field of research applicable to a wide range of problems in industrial robotics, automotive industry and general autonomous mobile robots. It is the task of deriving dense depth image from sparse depth measurements. This dense distance data is a prerequisite for obstacle avoidance, object detection or 3-D scene reconstruction. With the recent interest in development of autonomous vehicles, producing accurate dense depth images from sparse inputs became desirable. As low-priced Light Detection And Ranging (LiDAR) scanners become widely available, this trend can be expected to continue. Source of the sparse measurements differs between specific cases. For indoor scenes, depth cameras are often used. For outdoor scenes, LiDAR sensors or algorithmic approaches (Structure-From-Motion, Stereo Vision) are used.

Popular large scale indoors datasets suitable for depth completion are the NYU dataset [34] or Matterport3D [2]. Popular outdoors dataset is the KITTI benchmark [12]. For the purposes of this work, KITTI was chosen as the dataset to be used. Since the measurements are provided by a LiDAR, the data sparsity is far higher in comparison to NYU and Matterport3D, mere 64 scan lines in vertical direction. The indoor datasets are also mostly targeted at segmentation tasks. The KITTI depth completion dataset consist of over 93 thousand ground-truth depth maps with corresponding raw LiDAR scans and synchronized RGB images. An example of depth completion on KITTI [1] is shown on figure 2.25.

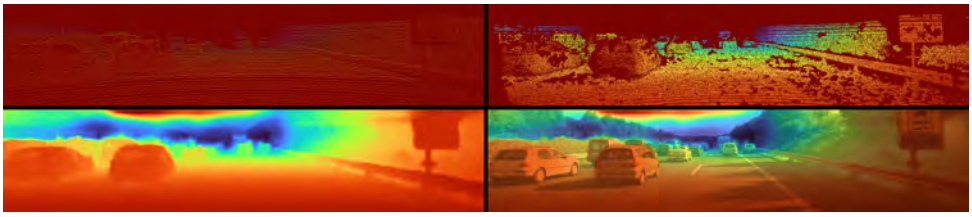


Figure 2.25: Depth completion on KITTI Depth Completion Benchmark [1].
Top left: Network input, **Top right:** Ground truth,
Bottom left: Network output, **Bottom right:** Network output + RGB.

Historically, depth inpainting was performed by handcrafted approaches to achieve upsampling of sparse inputs through interpolation. In the recent years, data driven Deep Convolutional Neural Networks (DCNNs) started to replace these classical approaches, however J. Ku *et al.* [29] show that a well designed algorithmic approach remains viable. Nevertheless, DCNNs show better results on complicated tasks and allow exploitation of additional data, such as RGB images. Recently J. Tang, F. Tian *et al.* [44] achieve state-of-the-art performance by generating context-dependent and spatially-variant convolution kernels from RGB image used as guidance. J. Qiu *et al.* [37] show impressive results by using RGB image and sparse depth to predict surface normals and a confidence map separately, fusing their outputs to produce dense depth prediction. A convolution operation, which explicitly

considers sparsity by evaluating only observed pixels and normalizing the output appropriately is presented by J. Uhrig *et al.* [47]. This approach leads to creation of networks which are invariant to input sparsity.

Noticeably, all of the mentioned approaches exploit additional information to improve the performance. Whether this additional information is an image, normal map or a validity mask. This shows that providing the network with additional context information is essential to achieve state-of-the-art performance. It can also allow the network to gain other beneficial properties. An example of this is the explicit handling of sparsity.

Simple approaches to handle the sparse inputs are to either replace invalid values with zeros or to supply the network with an additional input channel representing pixel validity. These methods were shown to be successful, however they depend on training to develop appropriate interpretation of the sparse input and are spatially invariant. The convolution operation presented by J. Uhrig *et al.* [47] (2.16) explicitly considers the sparsity by evaluating only observed pixels and normalizing the output appropriately. This approach leads to more robust models, which are invariant to sparsity of input and therefore allow sensor (e.g. LiDAR) replacement without the necessity of retraining the model. Similar benefits can be expected when exploiting other context data.

2.5.2 Semantic Segmentation

Semantic segmentation is the task of labeling all pixels within an image as belonging to one of n classes. It can also be understood as pixel-wise classification. Semantic segmentation is necessary for scene understanding of autonomous vehicles and is being used for inspection of industrial products and detection of defects. Lately it also found invaluable application in medical image analysis.

Popular large scale datasets suitable for semantic segmentation are the ADE20K dataset [49], Cityscapes [6] or the recent Audi A2D2 dataset [13]. For the purposes of this work, Cityscapes and A2D2 were considered, since they consist of automotive street scenes, similar to KITTI. This would allow the segmentation network to later be used as a context source for the depth completion network or vice versa. Ultimately Cityscapes was chosen as the dataset to be used, because of its established benchmarks and provided dense stereo depth. A2D2 dataset in comparison only provides sparse LiDAR measurements, which though more accurate, are not suitable as a context source for segmentation. Example of semantic segmentation on the Cityscapes dataset is shown on figure 2.26.

Historically, in classification tasks, DCNNs recognized rapid increase in popularity after AlexNet architecture [28] won ImageNet Large Scale Visual Recognition Challenge 2012. Among continuous improvements in the field, architectures with growing amount of parameters such as VGG-16 [41], Inception [43] and ResNet [17] were developed. These architectures continued to introduce additional modules to DCNNs such as Inception modules and ResNet skip connections. In Xception model, F. Chollet [5] shows that In-

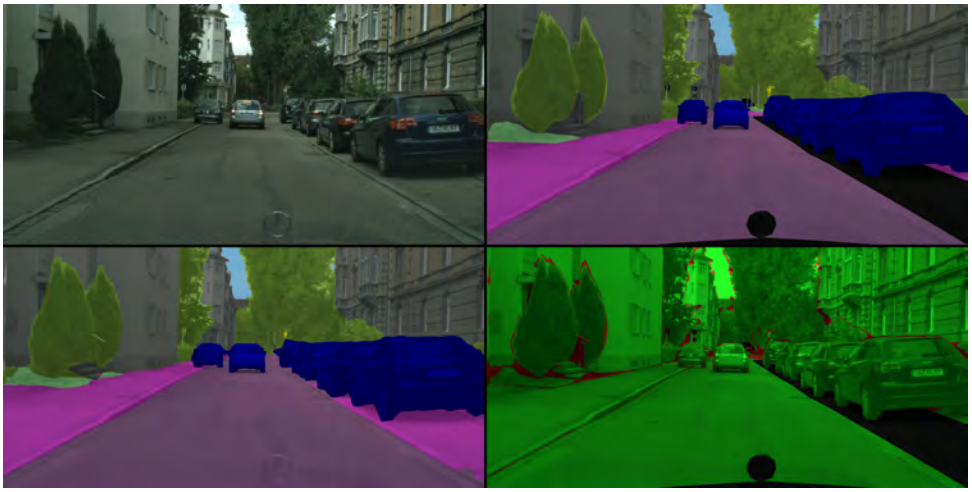


Figure 2.26: Semantic segmentation on Cityscapes dataset [6].
Top left: RGB image, **Top right:** Ground truth,
Bottom left: Network output, **Bottom right:** Validity.

ceptpion modules can be modified to an operation of depthwise separable convolution. This special case of convolution, showcased as early as 2014 by A. Geiger *et al.* [12], significantly improves performance without increasing the number of model parameters. This is extremely desirable for networks utilizing spatially variant convolution, as it has inherent high memory usage during backpropagation in the training process.

In semantic segmentation tasks, L. Chen *et al.* [3] first show that convolution with upsampled filters, or 'atrous convolution', allows explicit control over the resolution at which feature responses are computed. They also present it as a method of arbitrarily enlarging field of view of filters, without increasing the number of network parameters. Second, they propose Atrous Spatial Pyramid Pooling (ASPP) in DeepLab's later iterations, an advanced pooling module, to capture features and objects at multiple scales. They implement proposed operations in a decoder on top of feature-extraction layers from acknowledged networks (encoders), such as ResNet and VGG-16, achieving state-of-the-art performance from 2015 until 2019. Only recently A. Tao, K. Sapra, B. Catanzaro [45] achieve better performance by allowing the network to learn how to best combine predictions from multiple scales of the original image - in contrast to simple averaging over the predictions. They also utilize semi-supervised learning, enlarging the original dataset with outputs from a teacher network.

2.6 Hypothesis

It is possible to improve network performance through the use of local context mechanism based on the sparse convolutions (2.16) [47], where the local context corresponds to local real-valued mask which assigns higher weights to contextually close neighboring pixels and lower weights to contextually

distant pixels. This can be seen as an extension of (2.16) for generic context sources. The context source can vary depending on the application. For semantic segmentation, spatial context based on depth or 3-D distance can be used. For depth completion, semantic context based on a class label distance or learnt inter-class dependencies or appearance context based on RGB distance seem reasonable. It can be expected that the additional information provided in combination with explicit consideration of context will improve performance near object edges or thin objects by placing lower emphasis on contextually distant, though spatially close pixels. It can also allow the network to gain beneficial properties such as the sparsity invariance described by J. Uhrig *et al.* [47].

For task of depth completion, this approach could benefit the network in multiple ways, depending on the context source. It is expected to allow the network to fill in the depth of entire objects, even though only parts of the objects possess valid distance measurements. It is also expected to allow finer predictions at areas with thin objects such as railings. This property would be highly dependent on accurate synchronization between the depth measurements and the context source. The last expected benefit is preservation of textures within the context. Standard depth completion networks often predict smooth depth transitions regardless of varied, irregular depth measurements. This causes irregular and textured objects such as bushes to appear smooth in the prediction. Thus these objects are afterwards difficult to distinguish from truly smooth ones, for example walls.

For segmentation networks with depth context the approach is expected to allow the network to more accurately process areas with multiple overlapping classes. These classes, such as cars, people and background buildings, are spatially close within an image and as such are going to be processed at the same convolution kernel position. Though spatially close within the image, the classes likely are distant in the real world and the depth measurements. The weights generated from depth context are then expected to allow the network to better distinguish between the overlapping classes and to improve accuracy at object boundaries.

Furthermore the proposed masked convolution can be generalized to n-D case and used with diverse context sources appropriate for the task. If proven fruitful, the approach could substitute current methods of providing the network with additional information - such as concatenation or addition.

Detailed explanation of the proposed local context mechanism is provided in section 3.1.

Chapter 3

Methods

3.1 Convolution with Local Context Based Masks

In the equation (2.16), tensor $\mathbf{o}$ serves as a source of validity context. The tensor elements are only allowed two values, indicating valid input ($o_{u,v} = 1$) or invalid input ($o_{u,v} = 0$). We propose a novel convolution operation, Local-Context-Masked Convolution (LCMC), described by equation (3.1), which is an extension and generalization of (2.16).

$$f_{u,v}(\mathbf{x}, \mathbf{o}) = b + \frac{\sum_{i,j=-\kappa}^{\kappa} m_{u,v,i,j} \cdot x_{u+i,v+j} \cdot w_{i,j}}{\sum_{i,j=-\kappa}^{\kappa} m_{u,v,i,j}} \tag{3.1}$$

Expressions (2.16), (3.1) are indeed identical for $m_{u,v,i,j} = o_{u+i,v+j}$. We propose that $m_{u,v,i,j}$ need not be directly equal to values of tensor $\mathbf{o}$. Instead it can be computed through any real valued function $m_{u,v,i,j} = m_{u,v,i,j}(\mathbf{o})$, where $\mathbf{o}$ is a tensor encoding additional context suitable for the task.

Apparent example of such a tensor can be found in the task of depth completion. The sparse depth measurements are often supplemented with a synchronized RGB image. This image is a clear candidate for downsampling and further processing to generate context sources $\mathbf{o}$ for the convolution.

The question then arises: what properties should the function $m_{u,v,i,j}(\mathbf{o})$ have? It seems natural to assign higher weights $m_{u,v,i,j}$ to pixels $x_{u+i,v+j}$, which are contextually close to central pixel $x_{u,v}$ of the convolution. In the same manner lower weights should be assigned to pixels which are contextually distant from the central pixel. A simple expression to model this relation is the weighed inverse distance

$$m_{u,v,i,j} = \frac{1}{a \cdot \Delta o_{i,j} + b}, \quad a, b \in \mathbb{R}, \qquad \Delta o_{i,j} = \sum_{k} \left| o_{u+i,v+j,k} - o_{u,v,k} \right|. \tag{3.2}$$

Note that the index k represents summation over all channels of tensor $\mathbf{o}$. To better suit the type of context source, other distance norms can be used instead of Manhattan distance L_1 (3.2). Reasonable substitutions can be the

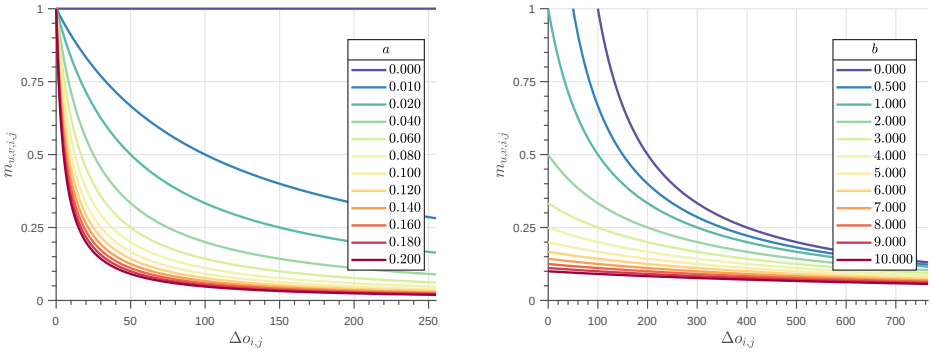
Euclidean distance L_2 (3.3)

$$\Delta_2 o_{i,j} = \sqrt{\sum_k \left(o_{u+i,v+j,k} - o_{u,v,k} \right)^2}, \tag{3.3}$$

or Chebyshev distance L_∞ (3.4)

$$\Delta_\infty o_{i,j} = \max_k \left| o_{u+i,v+j,k} - o_{u,v,k} \right|. \tag{3.4}$$

Weights generated with the inverse distance expression (3.2) are displayed on figure 3.1.



(a) : Varying parameter a and fixed parameter $b = 1$.

(b) : Fixed parameter $a = 0.010$ and varying parameter b.

Figure 3.1: Local mask weights $\mathbf{m}_{u,v}$ generated as the inverse distance of $\mathbf{o}$ elements through expression $w_{u,v,i,j} = \frac{1}{a \cdot \Delta o_{i,j} + b}$.

Another natural choice for the weight mapping are the Gaussian (3.5) and Laplacian (3.6) distributions respectively

$$m_{u,v,i,j} = e^{-\frac{1}{2}\left(\frac{\Delta o_{i,j}}{\sigma}\right)^2}, \quad \sigma \in \mathbb{R}, \tag{3.5}$$

$$m_{u,v,i,j} = \frac{1}{2} e^{-\frac{\Delta o_{i,j}}{\sigma}}, \quad \sigma \in \mathbb{R}. \tag{3.6}$$

Note that (3.5), (3.6) are not exact forms of Gaussian and Laplacian distribution, as they lack the scaling terms $\frac{1}{\sigma\sqrt{2\pi}}$ and $\frac{1}{\sigma}$ respectively. This scaling is omitted in order to ensure $m_{u,v,i,j} = 1$ for $\Delta o_{i,j} = 0$, as to penalize contextually distant pixels, but not inflate weights of contextually close pixels.

The Gaussian weight distribution is shown on figure 3.2. The Laplacian weight distribution is shown on figure 3.3. It is important to note that the proposed convolution is spatially variant and dependent on the context source tensor $\mathbf{o}$. To explain further, figure 3.4 displays an example of how the effective convolution kernel and the context weight kernel $\mathbf{m}_{u,v}$ are generated using Gaussian distribution weights and RGB image as tensor $\mathbf{o}$.

Analogical approach called Pixel-Adaptive Convolution (PAC) was also separately devised by H. Su *et al.* [42] and used for the task of depth prediction

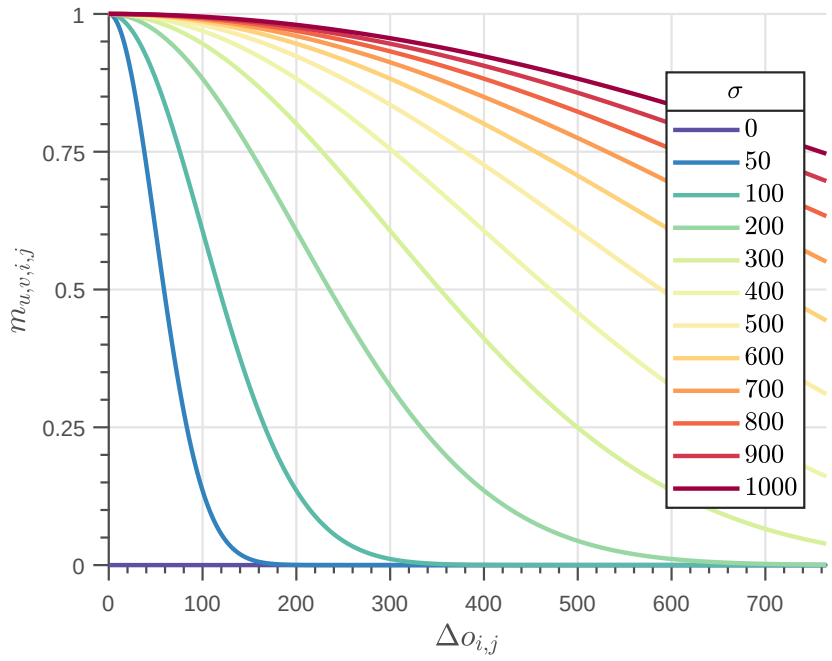


Figure 3.2: Gaussian distribution $m_{u,v,i,j} = e^{-\frac{1}{2}\left(\frac{\Delta o_{i,j}}{\sigma}\right)^2}$ of local mask weights $\mathbf{m}_{u,v}$ depending on context tensor $\mathbf{o}$ and learnable standard deviation σ.

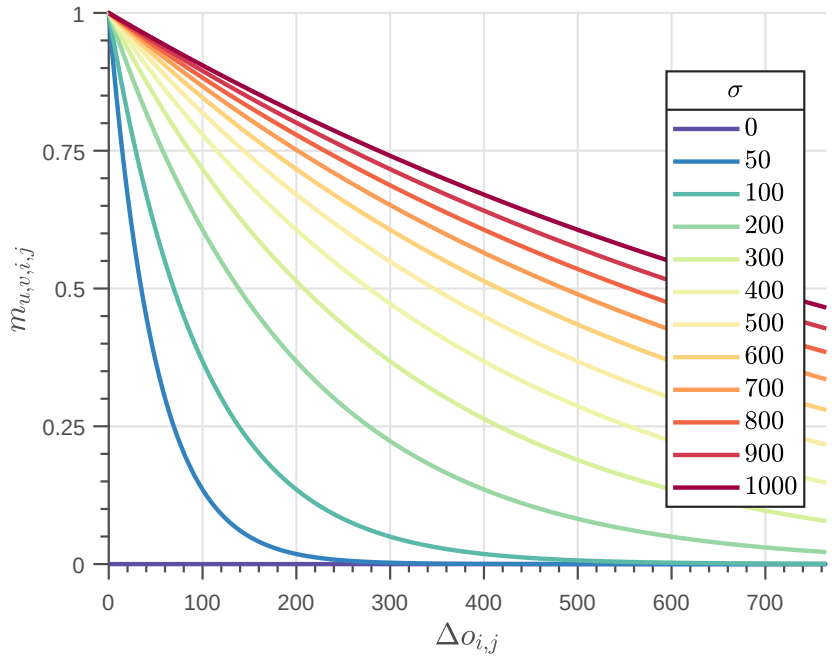


Figure 3.3: Laplacian distribution $m_{u,v,i,j} = \frac{1}{2}e^{-\frac{\Delta o_{i,j}}{\sigma}}$ of local mask weights $\mathbf{m}_{u,v}$ depending on context tensor $\mathbf{o}$ and learnable parameter σ.

on KITTI benchmark [12] with great success by V. Guizilini *et al.* [16]. This task involves predicting dense depth only from an RGB image with no sparse measurements provided and is not to be confused with depth completion.

V. Guizilini *et al.* used PackNet [15] as the depth prediction network and a pretrained, separate Feature Pyramid Network (FPN) with ResNet [17] backbone as the context source network.

Articles of H. Su *et al.* and V. Guizilini *et al.* were discovered while working on late stages of the thesis. Though the proposed convolutions are similar, it is important to note the differences. H. Su *et al.* introduce Pixel-Adaptive Convolution (PAC), transposed Pixel-Adaptive Convolution (PAC^T) and Conditional Random Fields which employ PAC (PAC-CRF).

This work only proposes Local-Context-Masked Convolution (LCMC) which is analogical to PAC, but introduces additional normalization in the form of denominator $\sum_{i,j=-\kappa}^{\kappa} m_{u,v,i,j}$ in equation (3.1). It evaluates the LCMC on KITTI benchmark [12] on the task of depth completion and on Cityscapes [6] on pixel-level semantic labeling task and employs wide range of context sources: RGB image, depth image, feature maps from a pretrained network. Thus, this work could be viewed as an extension of previous work by H. Su *et al.* and V. Guizilini *et al.*

Network architecture used for the experiments is described in section 3.2. Experiment setup, implementation details and the training process are described in section 3.3.

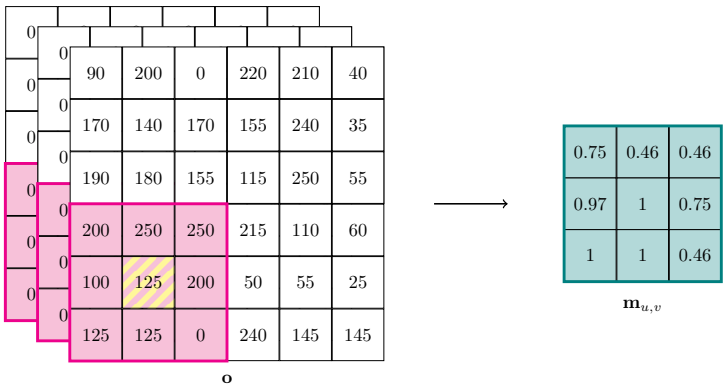
3.2 Network Architecture

Network architecture used for experiments with the LCMC was constructed as a modification of DeepLab [3]. Variation of Xception [5] is used as the backbone for extraction of both low-level and high-level features. High-level features are resampled at multiple scales through ASPP and upscaled through bilinear interpolation. Low-level features are then concatenated with resampled high-level features, upscaled and further convolved to produce the output.

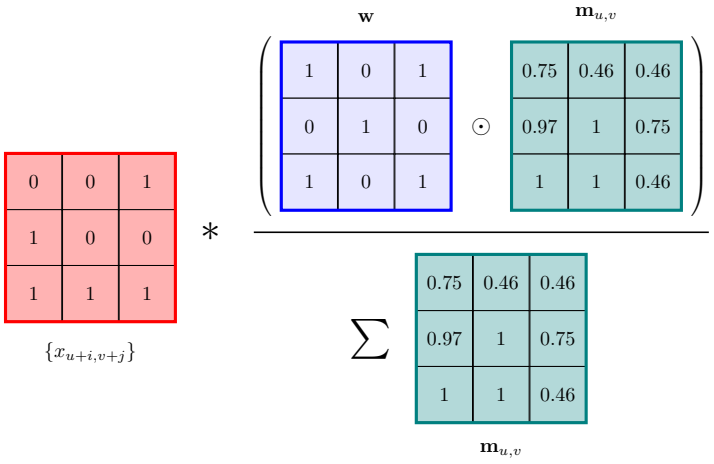
The network can be separated into an encoder and a decoder, shown for the task of depth completion on figures 3.5, 3.6 respectively. Modifications from a traditional DeepLab network are the replacement of all convolutions with LCMCs and reduction of channel numbers of intermediate layers due to the increased memory requirements of spatially variant convolution.

Semantic segmentation networks had channel numbers reduced by a factor of 4 within both the encoder and the decoder. Depth completion networks had channel numbers reduced by a factor of 2 within the first 2 layers of the encoder and by a factor of 4 within the remaining layers of the decoder. Channel numbers within the decoder of depth completion networks were reduced by a factor of 2. Note that such reductions do not result in trivial decrease of the network size by a factor of approximately 4.

Comparing the architecture to V. Guizilini *et al.* [16], one must notice that V. Guizilini *et al.* employ the additional context information only during decoding stages through PAC^T. Our network utilizes the context information throughout the entire forward pass and employs exclusively LCMC to exploit the context. LCMC also supports depthwise separable convolution as proposed



(a) : Local context mask $\mathbf{m}_{u,v}$ generated with Gaussian distribution weights from an RGB image $\mathbf{o}$. Standard deviation $\sigma = 100$. Only upper channel of $\mathbf{o}$ is non-zero for clarity.



(b) : Convolution of input $\{x_{u+i,v+j}\}$ with local mask $\mathbf{m}_{u,v}$ and global kernel $\mathbf{w}$. Displayed division is elementwise. Note that the entire tensor $\mathbf{x}$ is of the same height and width as tensor $\mathbf{o}$, but mask $\mathbf{m}_{u,v}$ is used only locally for $f_{u,v}(\mathbf{x}, \mathbf{o})$.

Figure 3.4: Accompanying visualization for equations 3.1, 3.5, elucidating Gaussian distribution mask generation from RGB image with RGB L_1 distance.

in Xception [5]. Note that the equation (3.1), which describes LCMC, has no effect on pointwise convolution (convolution with kernel size $k = 1$) and pointwise convolution is therefore visualized in a color separate from LCMC.

3.3 Experiment Setup

We utilize 5 GTX 1080Ti, 2 GTX Titan X, 1 GTX Titan Xp GPUs for training, with batch size of 2. Multitude of networks was trained in parallel and GPUs were distributed according to computational and memory requirements. Gaussian distribution was chosen as the default function to generate context weights. The choice was made for two reasons

- the distribution only has one parameter σ to be learned,
- the weights remain in interval $m_{u,v,i,j} \in]0, 1]$.

Parameter σ was initialized to 100. This value is based on experiments with RGB distance context and L1 norm. The largest possible L1 distance $\Delta o_{i,j}$ between two pixels, black and white, assuming 24-bit color, is

$$\Delta o_{i,j} = \sum_{k=1}^{3} |0 - 255| = 3 \cdot 255 = 765. \tag{3.7}$$

Standard deviation of L1 pixel distance, computed on ImageNet [8] dataset, is 172.89. It can then be assumed that about 68% of all pixels are within distance of twice the deviation, 345.78. Gaussian (3.5) with parameter $\sigma = 100$ assigns negligable weights to L1 distance of pixels $\Delta o_{i,j} \gtrapprox 320$ and as such seems a reasonable choice. The same initial σ was used for all experiments, even ones which do not utilize RGB, to keep comparison of network variations simple.

Because of a large observed difference in training times between depth completion networks with and without LCMC, two additional experiments on the depth completion task were conducted with more powerful hardware. Network using RGB context source with batch size of 60 and network using ResNet [17] feature maps context source with batch size of 20 were trained on 8 Tesla V100 GPUs.

The higher batch size and more powerful hardware allowed these networks to reach number of training epochs comparable to networks without LCMC. It should be noted however that the training process of these networks differs significantly from others. The higher batch size allows batch normalization [27] to compute less noisy mean and variance, improve parameter update direction, but causes the network to perform fewer parameter optimization steps per epoch. Since the loss is averaged over batch elements, the training process does not compensate for the decrease in the number of update steps with higher loss.

3.3.1 Programming Language and Libraries

LCMC, the networks and training framework were implemented in PyTorch 1.5.0 [36] with Python 3.7.2 [48] and CUDA 10.0.130. The convolution implementation is flexible, allowing simple substitution of context weight function $m_{u,v,i,j}(\mathbf{o})$. Final implementation of LCMC, provided by the thesis supervisor, is written purely in PyTorch and therefore supports automatic differentiation. If the function $m_{u,v,i,j}(\mathbf{o})$ holds the same property, then the user is not required to manually implement the backward gradient propagation.

Additionally, experiments with PAC [42] were conducted with an alternative setup of PyTorch 1.1.0 and CUDA 9.2, as PAC does not support newer versions. As PAC also does not support depthwise separable convolution, the networks had to be implemented with a backbone other than Xception [5]. Such modifications prevented these networks from being directly compared to the main experiment and therefore the experiments with PAC were terminated.

3.3.2 Training

Loss Function

Depth Completion

Loss between the ground-truth and predicted depth is computed as mean squared error (MSE), formula of which can be seen in equation (3.8). Ground-truth depth of real world scenes, especially outdoor scenes with moving objects, is difficult to obtain. Therefore KITTI provides only partially dense ground truth and the loss is computed over the valid ground-truth pixels only. Let $\mathbf{P} = \{P_{i,j}\}$ be the 2-D depth prediction and let $\mathbf{T} = \{T_{i,j}\}$ be the 2-D ground truth target. The MSE is then computed as

$$\text{MSE} = \frac{\sum_{k,l}(P_{k,l} - T_{k,l})^2}{\sum_{k,l} 1}, \tag{3.8}$$

where $k, l \in \mathbb{N}^2$ are indices of valid measurements within the ground truth $\mathbf{T}$.

Semantic Segmentation

Cross entropy loss, a combination of negative log likelihood and softmax, equations (3.10) and (3.9) respectively, is utilized for training of semantic segmentation networks. The loss is computed separately for all pixels and classes, then reduced to a single number through averaging.

Let $\mathbf{P} = \{P_{i,j,k}\}$ be the network output and let $\mathbf{T} = \{T_{i,j,k}\}$ be the ground-truth probability. Softmax of $P_{i,j,k}$ is

$$S_{i,j,k} = \frac{e^{P_{i,j,k}}}{\sum_k e^{P_{i,j,k}}}. \tag{3.9}$$

The negative log likelihood of $S_{i,j,k}$ then is

$$N_{i,j,k} = -\log(S_{i,j,k}) \cdot T_{i,j,k}. \tag{3.10}$$

The final cross entropy loss can be computed as

$$CEL = \frac{\sum_{i,j,k} N_{i,j,k}}{\sum_{i,j} 1}. \tag{3.11}$$

Optimizer

All networks were trained with Stochastic Gradient Descent (SGD) optimizer with momentum 0.9, using Nesterov momentum [35]. Depth completion network loss was observed to diverge for learning rates higher than 0.02. Semantic segmentation network loss was observed to diverge for learning rates higher than 0.5. Initial learning rates were afterwards chosen as 0.002 and 0.05 respectively, a tenth of the experimentally discovered boundaries. The learning rate was scheduled to decrease by a factor of 0.1 after training for 30, 40, 50, 60 and 70 epochs.

The training process and parameters are designed to deliberately be simple and not introduce additional hyperparameters which could complicate final comparison of network variations.

3.3.3 Datasets and Data Augmentation

Depth Completion

The KITTI depth completion dataset [12] comprises over 86k training images, 1k validation images and 1k test images. The depth measurements are obtained through LiDAR and verified with stereo image pairs. Images in validation and test sets are provided already cropped to size 1216×352, while training images are not. Since LiDAR points at the top of an image are uncommon, all images are cropped to a uniform size 1216×256, discarding the upper part, following the approach of Tang *et al.* [44]. Example of the cropping is shown on figure 3.7. For training the image is horizontally flipped with a probability $p = 0.5$.

Semantic Segmentation

Cityscapes pixel-level segmentation dataset [6] comprises 3475 training and validation images which are both finely and coarsely annotated, 19998 additional coarsely annotated training images and multitude of further data, such as precomputed depth maps using semi-global matching. All images are provided already cropped to size 2048×1024. For both, training and validation, images are cropped at center to a size 1843×921. This is done to remove parts of the image where no depth data is available.

Missing section of depth data in other parts of the image were filled prior to training. Inpainting using cross-bilateral filter (CBF) at multiple scales produced the best results, but was replaced with a technique proposed by A. Telea [46] due to computational complexity of CBF. An example of the inpainting and cropping is shown on figure 3.8. Figure 3.9 shows an example of depth image where defects are present. It should therefore be noted that the depth measurements are of a lesser quality than LiDAR measurements and could potentially be a worse source of context.

For training, images are horizontally flipped with probability $p = 0.5$.

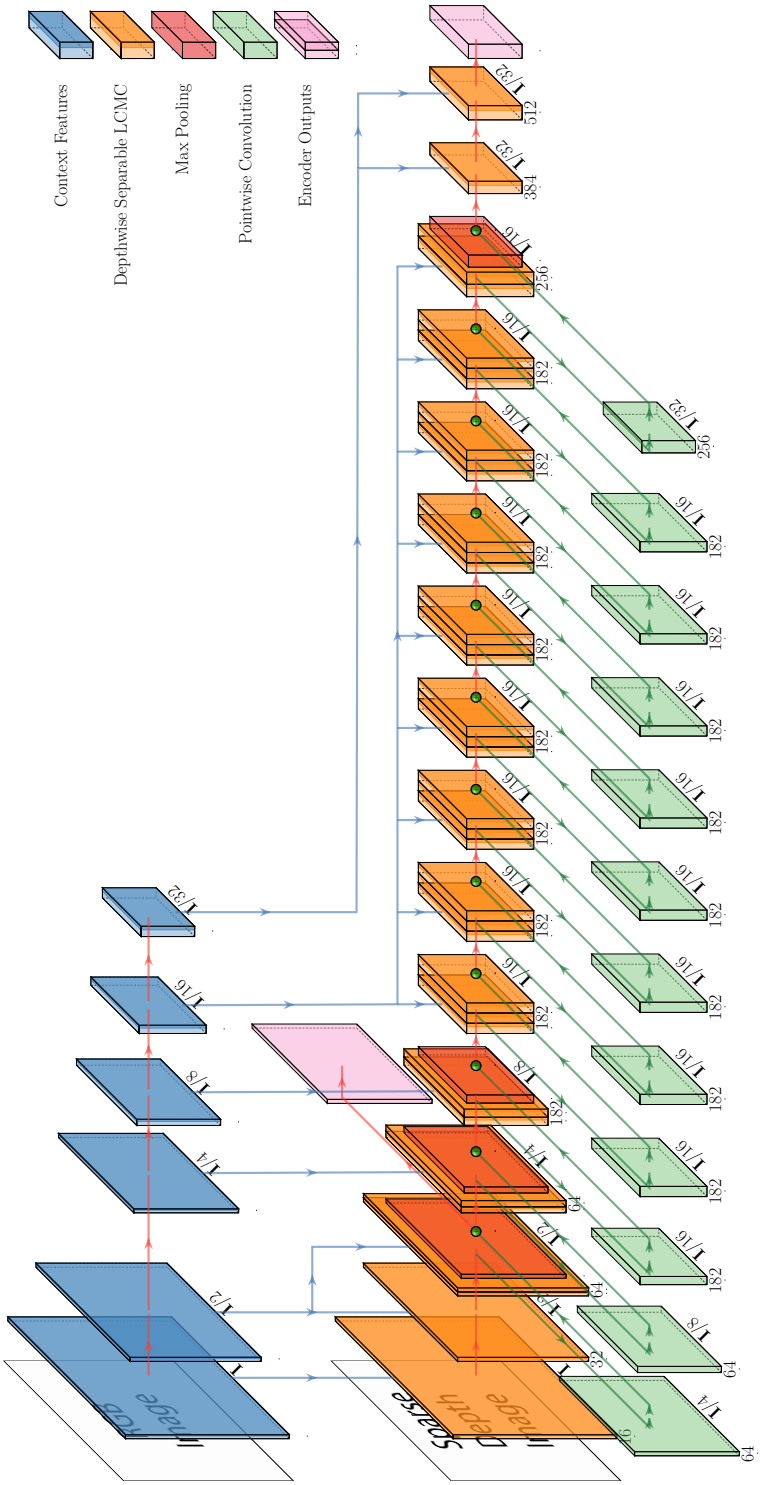


Figure 3.5: Proposed encoder architecture for depth completion task. Modified Xception [5] architecture with LCMC. Feature maps are extracted from synchronized RGB image with a separate network and used as a context source. Convolutional blocks which consist of two layers, lighter and darker, represent convolution followed by batch normalization and ReLU.

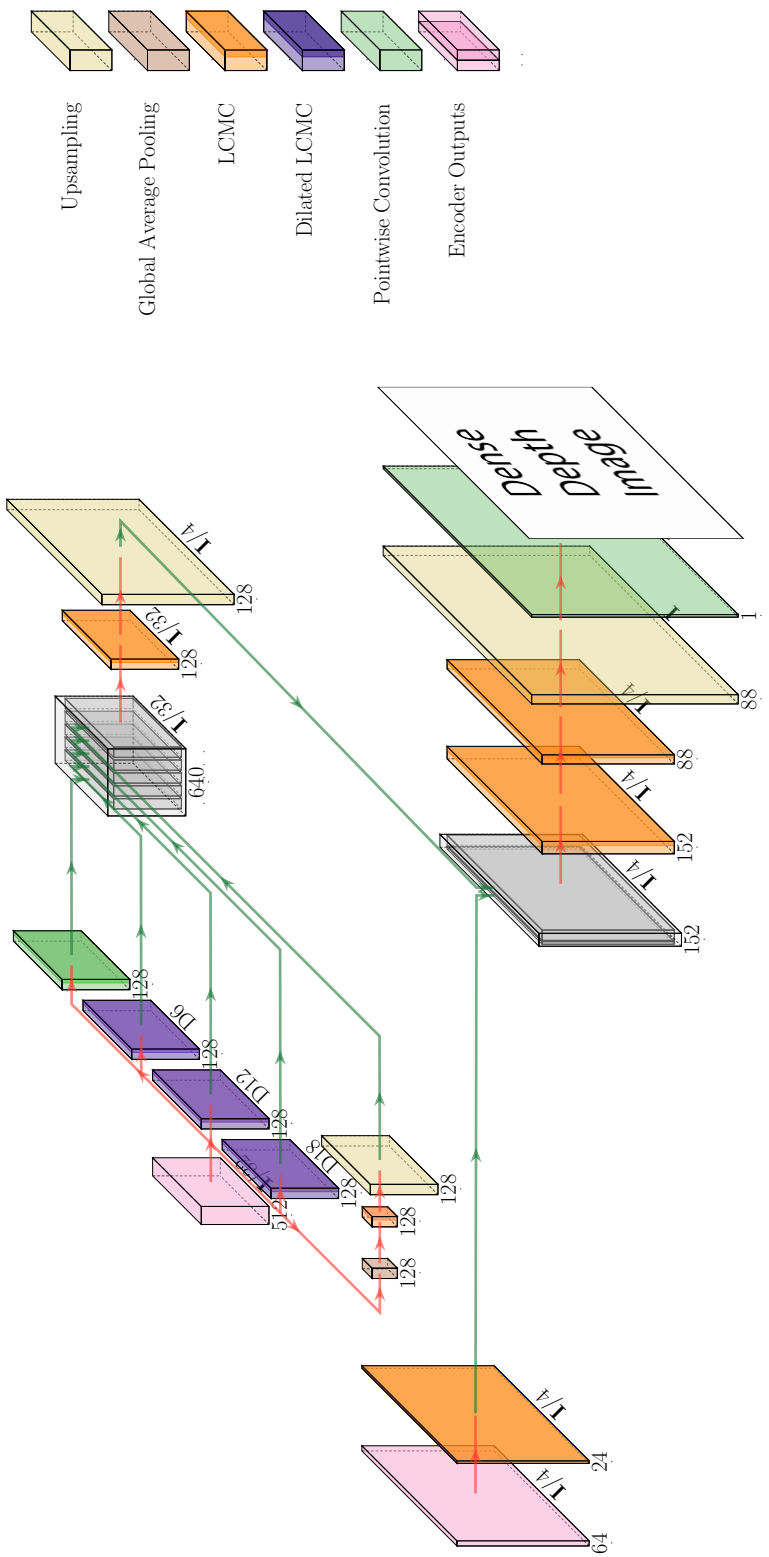
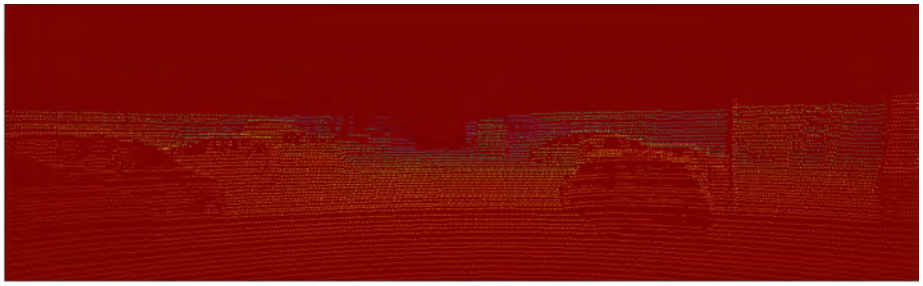
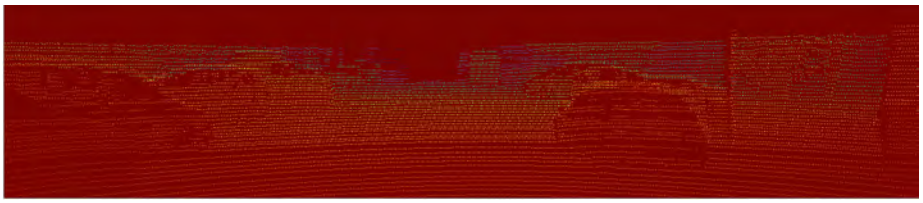


Figure 3.6: Proposed decoder architecture for depth completion task. LCMC makes use of identical context sources as the encoder, though they are not displayed. Upper part of the image displays ASPP [3]. Convolutional blocks which consist of two layers, lighter and darker, represent convolution followed by batch normalization and ReLU.

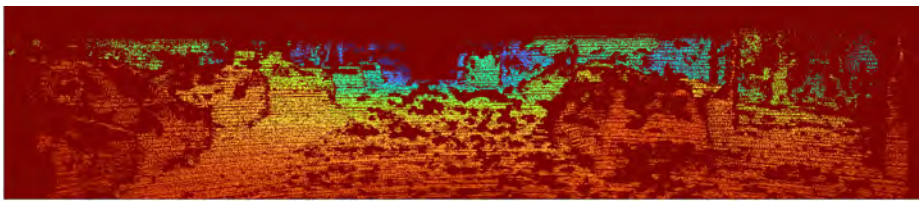
(a) : Original image.



(b) : Original depth.

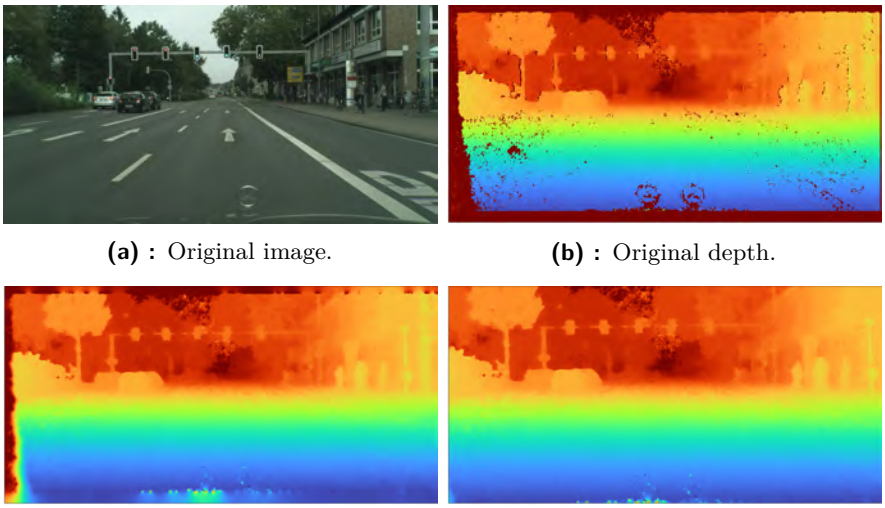


(c) : Cropped depth.



(d) : Cropped ground-truth depth.

Figure 3.7: Crop of KITTI [12] data.



(a) : Original image. **(b) :** Original depth.

(c) : Filled depth. **(d) :** Cropped depth.

Figure 3.8: Preprocessing and crop of Cityscapes [6] depth data.

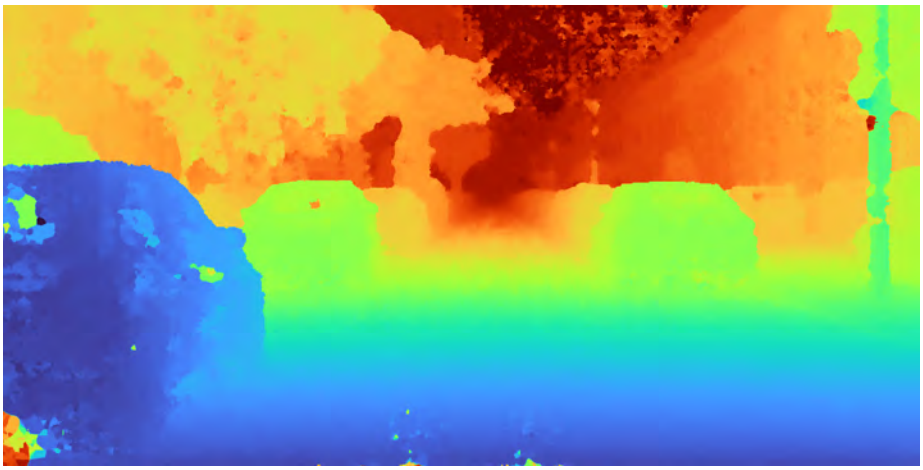


Figure 3.9: Defects of Cityscapes [6] depth data. Most noticeable on the car in the bottom left corner of the image. Further defects are present on the sky at the top of the image.

Chapter 4

Experimental Results

4.1 Results of Depth Completion

Depth completion network performance was measured on a set of 1000 standardized handpicked images. Network outputs were generated, then evaluated with development kit provided as a part of the KITTI dataset.

KITTI benchmark measures multiple metrics, comparing models on basis of Root Mean Square Error and Mean Absolute Error [mm] on depth (RMSE, MAE) and inverse depth (iRMSE, iMAE) [1/km] respectively. Mean, minimum and maximum of these metrics measured on the set of evaluation images are shown in tables 4.1, 4.2, 4.3 respectively. Network parameters for the evaluation were chosen as parameters that minimized RMSE on the validation dataset during training.

Aforementioned metrics can be computed for a 2-D depth prediction $\mathbf{P} = \{P_{i,j}\}$ and a 2-D ground-truth depth target $\mathbf{T} = \{T_{i,j}\}$ as

$$\text{MAE} = \frac{\sum_{k,l}|P_{k,l} - T_{k,l}|}{\sum_{k,l} 1}, \qquad \text{iMAE} = \frac{\sum_{k,l}|P_{k,l}^{-1} - T_{k,l}^{-1}|}{\sum_{k,l} 1}, \tag{4.1}$$

$$\text{RMSE} = \sqrt{\frac{\sum_{k,l}|P_{k,l} - T_{k,l}|^2}{\sum_{k,l} 1}}, \qquad \text{iRMSE} = \sqrt{\frac{\sum_{k,l}|P_{k,l}^{-1} - T_{k,l}^{-1}|^2}{\sum_{k,l} 1}}, \tag{4.2}$$

where k, l are indices of valid measurements within the ground-truth depth.

All depth completion network variations can be identified from a list of properties - *LCMC, rgb, large batch, context*. Their explanation is as follows:

1. Property *LCMC* indicates if the network uses LCMC instead of regular convolution.

2. Property *rgb* indicates if the network was provided with an RGB image.

   - Networks with *context*: ResNet use the image to generate feature maps which serve as a context source.
   - Networks with *context*: RGB use the image as a context source.
   - Networks with *context*: ✗ concatenate the image to input.

3. Property *large batch* indicates if the variation was trained with larger batch size than default batch size of 2.
   - Networks with *context*: ResNet use larger batch size of 20.
   - Networks with *context*: RGB use larger batch size of 60.
4. Property *context* indicates the context source for LCMC.

Network variations with LCMC and batch size of 2 managed to finish only 50 training epochs, due to computational complexity, before the work had to be submitted. The networks can under no circumstance be considered fully trained at that point and tables 4.1, 4.2, 4.3 therefore compare all network results only for the initial 50 epochs.

Network				Mean Metrics				
LCMC	rgb	large batch	context	iRMSE [1/km]	iMAE [1/km]	RMSE [mm]	MAE [mm]	SILog [log(m)·100]
✓	✓	✓	ResNet	4.6	2.19	1473.08	501.52	54.24
✓	✓	✗	ResNet	6.58	4.23	1769.78	797.46	64.82
✓	✓	✓	RGB	4.81	1.98	1449.88	448.78	56.56
✓	✓	✗	RGB	10.98	3.65	1896.57	864.32	63.46
✗	✓	✗	✗	4.75	2.33	1270.95	441.01	48.94
✗	✗	✗	✗	4.36	1.77	1228.32	370.95	48.07
GuideNet [44]				2.25	0.99	736.24	218.83	-
DeepLidar [37]				2.56	1.15	758.38	226.50	-
SparseConvs [47]				4.94	1.78	1601.33	481.27	-

Table 4.1: Mean of metrics on KITTI evaluation. Lower is better.

Network				Min. Metrics				
LCMC	rgb	large batch	context	iRMSE [1/km]	iMAE [1/km]	RMSE [mm]	MAE [mm]	SILog [log(m)·100]
✓	✓	✓	ResNet	1.82	1.21	565.1	290.56	20.05
✓	✓	✗	ResNet	3.14	2.32	865.93	534.47	24.91
✓	✓	✓	RGB	1.59	0.94	630.02	237.18	20.3
✓	✓	✗	RGB	2.7	2.14	794.65	453.25	23.9
✗	✓	✗	✗	1.7	1.17	557.56	250.68	19.09
✗	✗	✗	✗	1.46	0.94	483.04	191.24	17.76

Table 4.2: Minimum of metrics on KITTI evaluation. Lower is better.

Network				Max. Metrics				
LCMC	rgb	large batch	context	iRMSE [1/km]	iMAE [1/km]	RMSE [mm]	MAE [mm]	SILog [log(m)·100]
✓	✓	✓	ResNet	19.71	6.07	8132.75	2095.11	192.26
✓	✓	✗	ResNet	20.99	7.73	8551.68	2495.92	213.12
✓	✓	✓	RGB	20.91	5.43	7074.15	1825.74	209.54
✓	✓	✗	RGB	2525.15	166.98	159360.23	116755.02	1801.8
✗	✓	✗	✗	23.52	6.16	8924.36	2218.11	198.73
✗	✗	✗	✗	21.67	5.7	7571.79	1858.81	180.16

Table 4.3: Maximum of metrics on KITTI evaluation. Lower is better.

4.2 Results of Semantic Segmentation

Semantic segmentation network performance was measured on a *test* set of handpicked images. Network outputs were generated and evaluated with

cityscapesscripts functions provided with the Cityscapes dataset.

In multi-class classification problems, such as semantic segmentation, the traditional performance metric is Intersection over Union (IoU), also called the Jaccard index. IoU of two sets A, B is defined as

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|}. \tag{4.3}$$

The segmentation networks were trained to predict 1 class from a list of 19 classes, a standard setup on Cityscapes, for each pixel. These classes can be seen in tables 4.5 and 4.7. Each class also belongs to one of 6 categories which can be seen in tables 4.4 and 4.6. Network parameters for the evaluation were chosen as parameters that maximized mean IoU on the validation dataset during training.

Metrics measured on the *test* set cropped at center to size 1843×921, removing regions of the image with no depth measurements, are shown in tables 4.5, 4.4. Metrics measured on the same set without cropping are shown in tables 4.7, 4.6.

All semantic segmentation network variations can be identified from a list of properties - *LCMC, coarse, depth*. Their explanation is as follows:

1. Property *LCMC* indicates if the network uses LCMC instead of regular convolution.
2. Property *coarse* indicates if the network was pretrained for 20 epochs on the large set of coarsely annotated images. The training parameters being specified in section 3.3.2.
3. Property *depth* indicates if the network was provided with a depth image.
   - Networks with *LCMC*: ✓ use the depth as a context source.
   - Networks with *LCMC*: ✗ concatenate the depth to input.

All network variations were trained for over 100 epochs. For in-depth examination, confusion tables of some network variations are shown on figures 4.1, 4.2, 4.3.

Network			Category IoU							
LCMC	coarse	depth	construction	flat	human	nature	object	sky	vehicle	mean
✓	✓	✓	0.84	0.97	0.64	0.86	0.45	0.85	0.82	0.78
✓	✗	✓	0.83	0.97	0.63	0.85	0.41	0.85	0.81	0.76
✗	✓	✓	0.88	0.98	0.65	0.87	0.52	0.93	0.87	0.81
✗	✗	✓	0.87	0.98	0.61	0.87	0.46	0.92	0.85	0.79
✗	✓	✗	0.89	0.98	0.68	0.9	0.54	0.93	0.89	0.83
✗	✗	✗	0.88	0.98	0.67	0.9	0.49	0.93	0.88	0.82

Table 4.4: Category IoU on cropped Cityscapes evaluation. Higher is better.

Network			Class IoU																		
LCMC	coarse	depth	road	swalk	build	wall	fence	pole	tlight	tsign	veg	terr	sky	person	rider	car	truck	bus	train	mcycle	bcycle	mean
✓	✓	✓	0.96	0.68	0.83	0.2	0.25	0.36	0.44	0.54	0.87	0.42	0.85	0.61	0.33	0.83	0.24	0.14	0.11	0.22	0.54	0.5
✓	✗	✓	0.95	0.64	0.83	0.25	0.24	0.33	0.37	0.48	0.85	0.43	0.85	0.58	0.23	0.81	0.25	0.22	0.11	0.11	0.49	0.48
✗	✓	✓	0.97	0.76	0.88	0.43	0.48	0.45	0.45	0.51	0.87	0.53	0.93	0.63	0.37	0.89	0.55	0.67	0.55	0.23	0.55	0.62
✗	✗	✓	0.97	0.73	0.87	0.35	0.38	0.39	0.42	0.44	0.87	0.48	0.92	0.58	0.3	0.87	0.47	0.58	0.32	0.15	0.51	0.56
✗	✓	✗	0.98	0.79	0.88	0.43	0.5	0.47	0.45	0.61	0.9	0.59	0.93	0.66	0.41	0.9	0.51	0.64	0.55	0.28	0.63	0.64
✗	✗	✗	0.97	0.76	0.88	0.39	0.42	0.43	0.43	0.56	0.89	0.57	0.93	0.66	0.39	0.89	0.44	0.59	0.4	0.25	0.62	0.6

Table 4.5: Class IoU on cropped Cityscapes evaluation. Higher is better.

Network			Category IoU							
LCMC	coarse	depth	construction	flat	human	nature	object	sky	vehicle	mean
✓	✓	✓	0.82	0.96	0.59	0.85	0.4	0.85	0.78	0.75
✓	✗	✓	0.79	0.94	0.58	0.83	0.36	0.81	0.75	0.72
✗	✓	✓	0.86	0.97	0.61	0.86	0.46	0.9	0.83	0.78
✗	✗	✓	0.83	0.97	0.57	0.84	0.4	0.87	0.79	0.75
✗	✓	✗	0.88	0.98	0.65	0.9	0.51	0.89	0.88	0.81
✗	✗	✗	0.87	0.98	0.65	0.9	0.48	0.9	0.87	0.81

Table 4.6: Category IoU on Cityscapes evaluation. Higher is better.

Network			Class IoU																		
LCMC	coarse	depth	road	swalk	build	wall	fence	pole	tlight	tsign	veg	terr	sky	person	rider	car	truck	bus	train	mcycle	bcycle	mean
✓	✓	✓	0.92	0.59	0.82	0.19	0.27	0.32	0.38	0.51	0.85	0.39	0.85	0.57	0.29	0.77	0.24	0.09	0.14	0.18	0.51	0.47
✓	✗	✓	0.89	0.53	0.79	0.21	0.22	0.28	0.32	0.44	0.83	0.4	0.81	0.55	0.22	0.75	0.19	0.19	0.12	0.09	0.46	0.44
✗	✓	✓	0.96	0.72	0.86	0.4	0.43	0.4	0.39	0.44	0.86	0.52	0.9	0.59	0.35	0.85	0.45	0.57	0.46	0.2	0.51	0.57
✗	✗	✓	0.96	0.69	0.82	0.32	0.33	0.34	0.34	0.36	0.85	0.45	0.87	0.54	0.25	0.84	0.36	0.53	0.16	0.11	0.47	0.5
✗	✓	✗	0.97	0.78	0.88	0.43	0.5	0.44	0.44	0.58	0.89	0.57	0.89	0.64	0.4	0.89	0.55	0.62	0.5	0.29	0.62	0.63
✗	✗	✗	0.97	0.76	0.87	0.39	0.43	0.41	0.41	0.54	0.89	0.56	0.9	0.63	0.36	0.88	0.41	0.58	0.37	0.22	0.61	0.59
H. Multi-Scale Attention [45]			0.99	0.89	0.95	0.72	0.69	0.76	0.82	0.85	0.95	0.75	0.96	0.9	0.79	0.97	0.8	0.94	0.86	0.77	0.81	0.85
Panoptic DeepLab [4]			0.99	0.88	0.95	0.68	0.66	0.75	0.81	0.84	0.94	0.74	0.96	0.89	0.77	0.97	0.79	0.92	0.89	0.76	0.79	0.84

Table 4.7: Class IoU on Cityscapes evaluation. Higher is better.

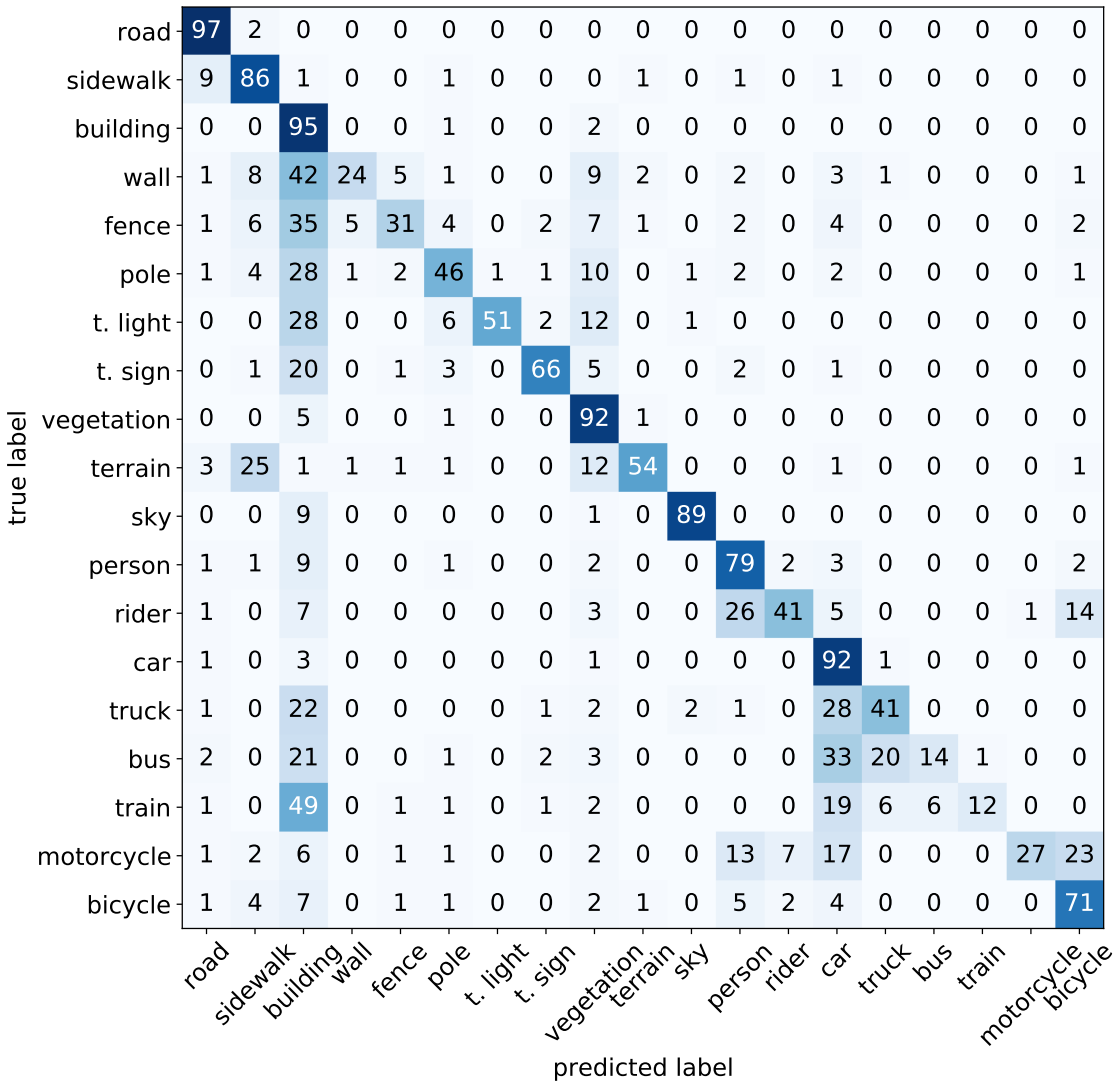


Figure 4.1: Normalized confusion matrix of LCMC ✓, coarse ✓, depth ✓.

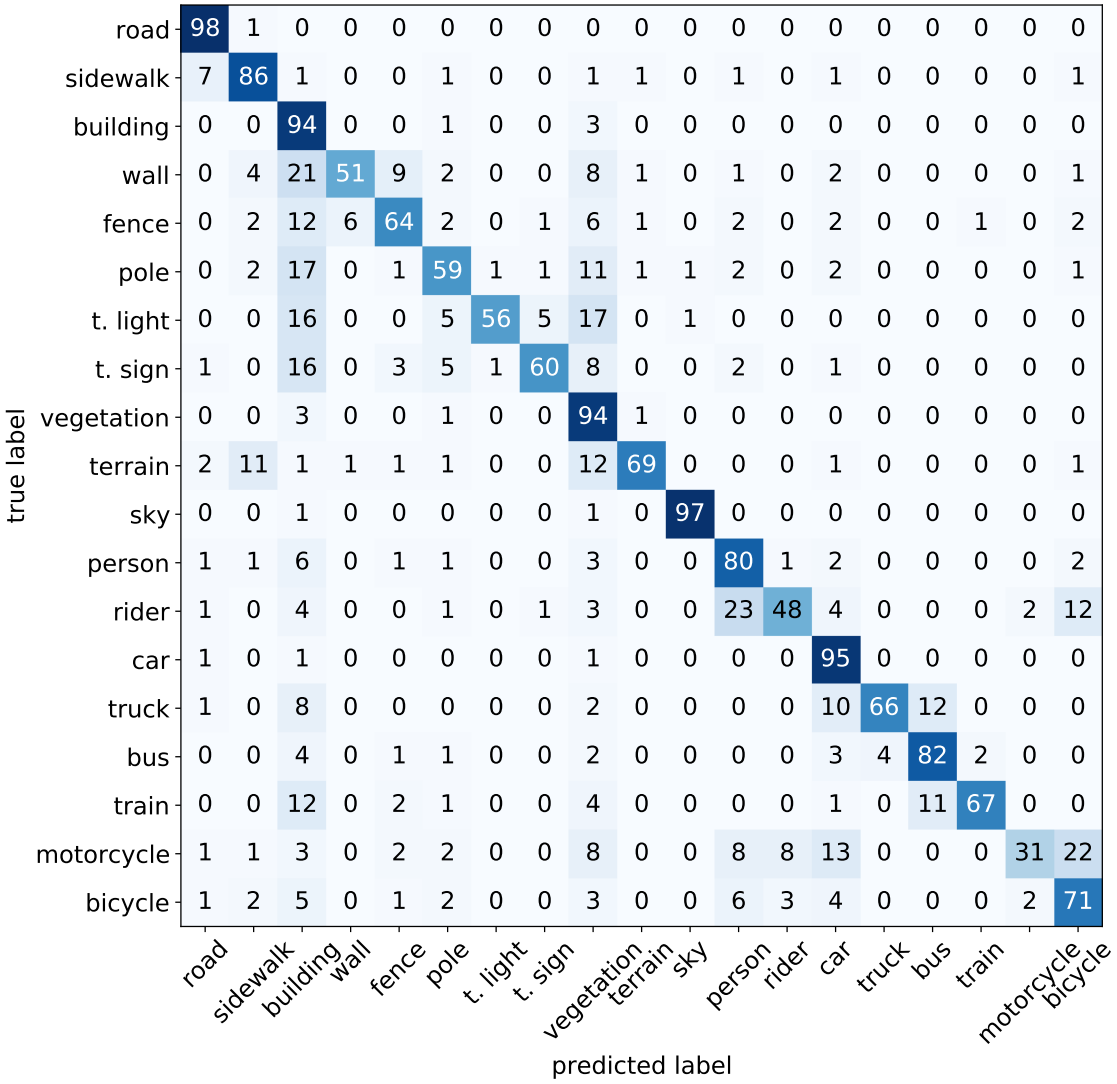


Figure 4.2: Normalized confusion matrix of LCMC ✗, coarse ✓, depth ✓.

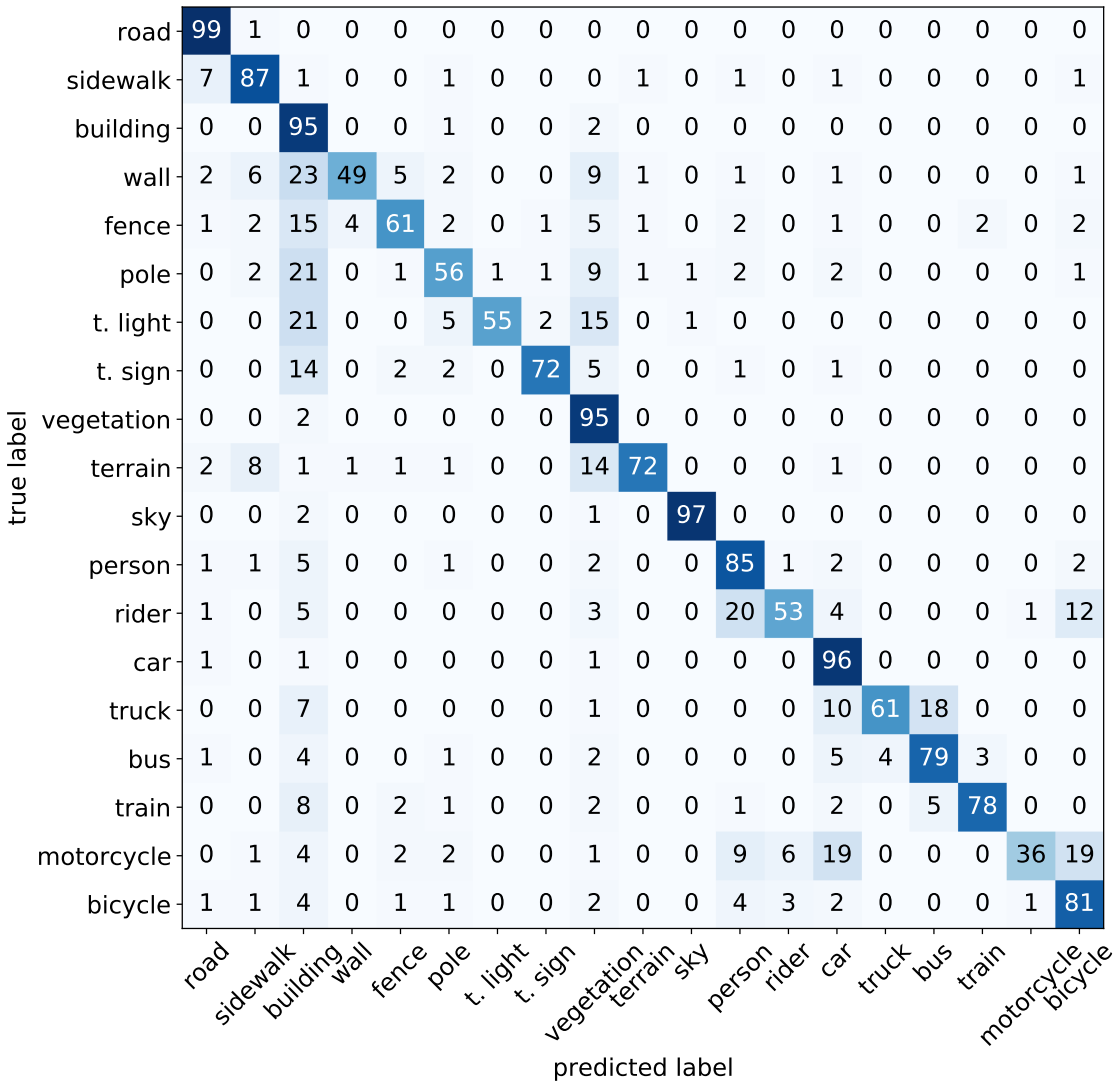


Figure 4.3: Normalized confusion matrix of LCMC ✗, coarse ✓, depth ✗.

Chapter 5

Discussion

This work proposed, implemented and evaluated use of modified convolution operation within CNNs. The operation, named Local Context Masked Convolution (LCMC), explicitly considers additional context when processing data. This is done through a mechanism, where local context corresponds to a local real-valued mask which assigns higher weights to contextually close data and lower weights to contextually distant data. The mask is spatially variant and is applied to standard convolution kernel at each output pixel position. Multitude of networks was trained with parameters described in section 3.3.2.

5.1 Depth Completion

It was expected that the operation could be used for fusion of data within neural networks and potentially result in performance improvements. In the depth completion task, it was expected that the network would fill in depth of whole objects, even though their distance was only partially measured. Basis for the expectation was that the network would recognize pixels belonging to the object as contextually close. This would cause the depth inpainting to consider spatially distant measurements belonging to the object more heavily than spatially close measurements belonging to the background.

Results have shown that this was a false expectation, as can be seen on figure 5.1. None of the networks were able to obtain the property described above and correctly predict depth of tree trunk, highlighted in red at the top of the image, where no depth measurements are present. Comparison of all network outputs on the scene can be seen on figures A.4, A.8. The figures display predicted depth projected to the original RGB image and isolated predicted depth, respectively.

Another expectation was that the operation would allow finer depth predictions at areas with thin objects such as railings. This expectation was confirmed, although not conclusively, by multitude of images. Exemplar case can be seen on figure 5.2, where networks with LCMC were able to more accurately capture the shape of railings on the right side of the image, highlighted in blue. This property however was not highly influential of the measured metrics, as KITTI ground truth images posses only sporadic

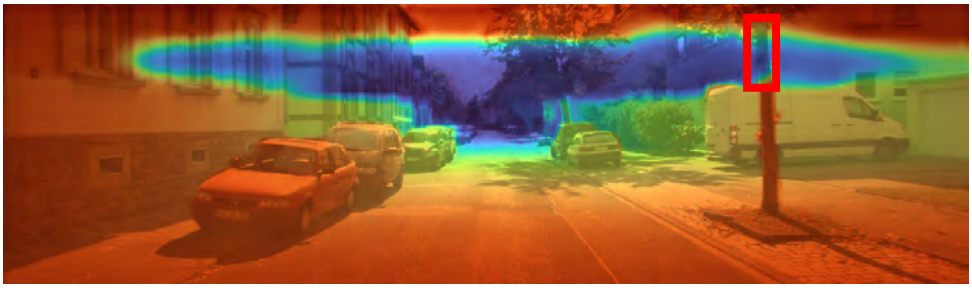


Figure 5.1: Depth completion networks are not able to correctly inpaint depth in areas, where no measurements are present.

measurements of such thin objects. Comparison of networks on the scene are again shown on figures A.3, A.7.

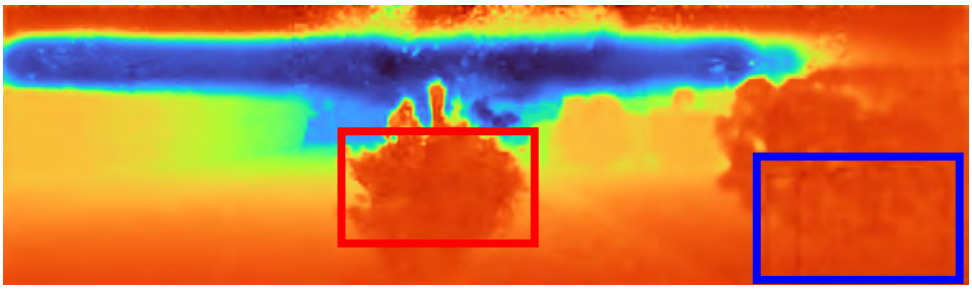


Figure 5.2: Networks utilizing LCMC are able to generate finer predictions of thin objects and are able to transfer texture from the image to depth.

Figure 5.3: RGB image accompanying depth prediction figure 5.2.

Lastly, it was expected that structure from within the context source, such as textures in images, would be transferred to the depth, without needlessly being blurred. This property was partially observed on irregular object, such as bushes, displayed on figure 5.2, highlighted in red.

5.2 Semantic Segmentation

In the semantic segmentation task, it was expected that the network would more accurately process areas with multiple overlapping objects and classes.

These objects, such as cars captured in front of background buildings, are spatially close to the background within the image, but are distant in the depth measurements. It was thus expected that object boundaries would be refined and the overall prediction improved.

Another expectation was that the information about object distance would provide the network with explicit knowledge of object scale. Based on this information, sub-networks could form within the network and process objects at different scales with different convolution kernels. This was expected to majorly improve segmentation of distant scenery.

Neither of these expectations was confirmed by the results, as can be deduced from figures in the appendix B. Instead, object boundary predictions were observed to become coarse and less accurate. Epitome of this is presented on figure 5.4, best seen on boundary between the car and the road. Different network outputs of the same scene are compared on figures B.1, B.2, B.3.

The classification also became highly dependent on quality of the depth image. In many cases, artifacts in the depth images resulted in incorrect and noisy predictions. Example of such classification failures can be seen on figures 5.5 and 5.6. Network outputs of the scenes, where noisy predictions were observed, are compared on figures B.7, B.8, B.9 and B.10, B.11, B.12.

Figure 5.4: Object boundaries predicted by network with LCMC and depth context are often coarse and inaccurate.

5.3 Work Limitations and Suggestions for Future Research

Ultimatelly, evaluation of experiments on depth completion and semantic segmentation tasks suggest that use of regular convolution results in better performance than use of LCMC. This claim is based on performance metrics recorded in sections 4.1 and 4.2. The performance degradation was surprising, as learned parameters of function $m_{u,v,i,j} = m_{u,v,i,j}(\mathbf{o})$ can in theory achieve

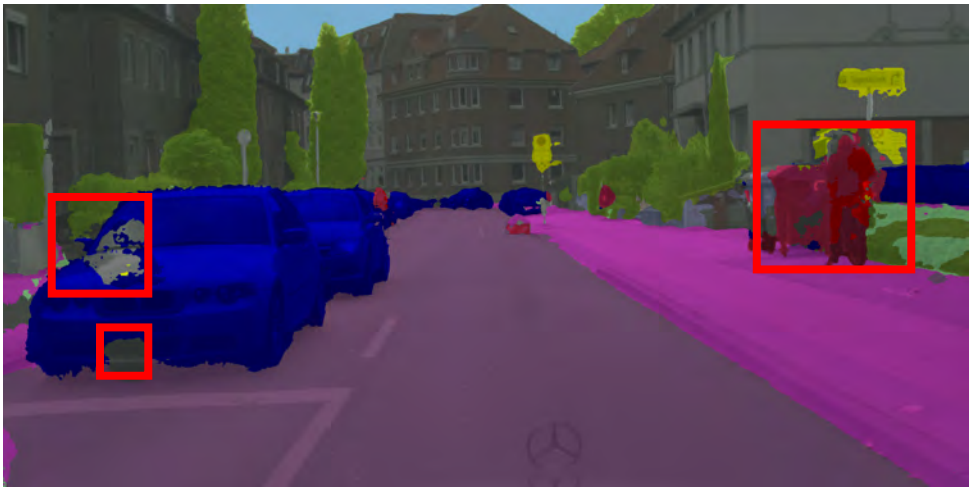


Figure 5.5: Failure of class prediction by networks utilizing LCMC and depth context, caused by anomalies in the depth image.

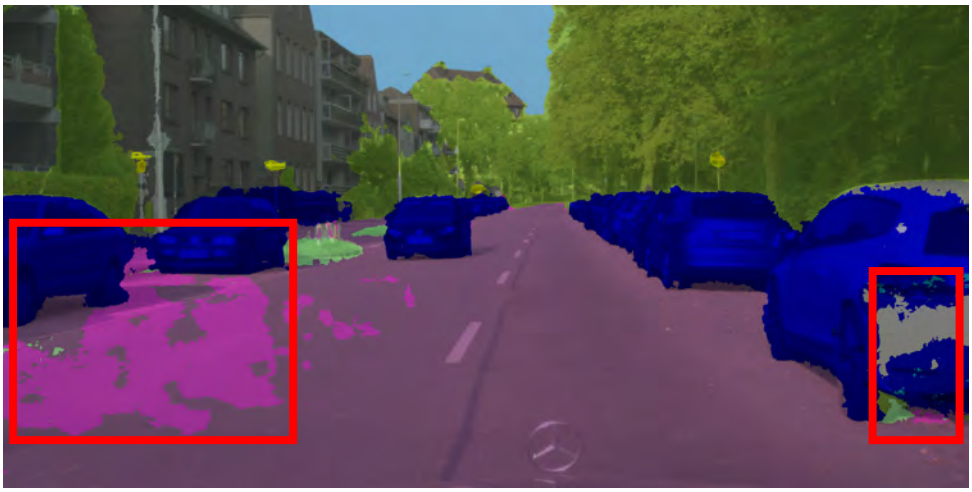


Figure 5.6: Failure of class prediction by networks utilizing LCMC and depth context, caused by anomalies in the depth image.

obliviousness of the operation $f_{u,v}(\mathbf{x}, \mathbf{o})$ to the context source $\mathbf{o}$. Such is the case of Gaussian $m_{u,v,i,j}(\mathbf{o})$, displayed on figure 3.2, for $\sigma \to \infty$ and finite $\Delta o_{i,j}$. It must be noted, however, that such operation is not identical to regular convolution, because it retains the normalization term $\sum_{i,j=-\kappa}^{\kappa} m_{u,v,i,j}$ as denominator. Possible limitations of the work and suggestion for future research are structured in following paragraphs. The author plans to continue research into this topic.

Kernel Size

Major oversight of the network architecture is the use of convolution kernels of size 3. While the original Xception model [5] utilizes this kernel size, it can be expected that larger kernels would be able to capture more dispersed

context differences. These kernels would then be able to register differences in context not only at object boundaries, but also near them.

Simplicity of Training Process

Training of the networks was deliberately kept simple in order to create an equitable comparison. In this comparison, it is clear that regular convolution outperforms LCMC. Effectiveness of the novel operation upon final convergence of the model would be another potential topic for future research. Ideally, the networks would be trained for a longer period of time, with an adaptive optimizer, until reaching convergence. Then they would be trained again with SGD to reach a state of better generalization.

Initialization

It is possible that the kernel weight initialization favored networks with regular convolution. Initialization of standard CNNs has been thoroughly studied, producing methods such a Xavier [14] or Kaiming [18] initialization. Xavier initialization was used within the networks and therefore it is possible that networks were trained in a setting advantageous to regular convolution.

The initial parameter value $\sigma = 100$ was selected for RGB distance context, but was used for all the experiments. Even though the network can adjust value of σ during the training process, the initial value should be chosen specifically for the given task.

Context Source Selection

Only a limited amount of context sources was explored, all with an identical function $m_{u,v,i,j}(\mathbf{o})$. However, function $m_{u,v,i,j}(\mathbf{o})$ should be designed with regard to context source $\mathbf{o}$. Example of a problem which could have been avoided with better design of the function $m_{u,v,i,j}(\mathbf{o})$ was the poor classification on Cityscapes. Anomalies in Cityscapes depth maps were partially caused by the inpainting process and caused the network to produce noisy predictions at times. A different approach, where areas with missing depth measurements are not filled-in and function $m_{u,v,i,j}(\mathbf{o})$ is designed with this knowledge, could have solved this issue.

Author's Error

Lastly, when implementing an idea, it is always a possibility that the execution is flawed. Though functionality of LCMC was tested and confirmed, errors have been made in other parts of the implementation. It was discovered that the learnable context weighing parameter σ was being reset to its initial value of $\sigma = 100$ after loading a model checkpoint. This means that the parameter was not able to converge to an optimal value and attributed to poor performance of models with LCMC. The error was resolved and further results of corrected networks will be made public on `https://github.com/paplhjak/Contextus`.

Comparison to Pixel-Adaptive Convolution

Comparison of LCMC and PAC in a neural network setting would be helpful, as to validate that both methods achieve similar results.

Comparison of the two operations in a different backbone is a possibility for future work - potentially in the task of depth prediction. This is suggested not only to compare performance of LCMC and PAC, but also to compare performance of PAC and transposed PAC. If both methods were to achieve similar results, it would suggest that transposed convolution with context masks is superior to the standard convolution with context masks, as transposed PAC was shown to produce state-of-the-art results in [16].

The reason this comparison was not provided as a part of this work is lack of certain parameters in PAC which are required for efficient implementation of depthwise separable convolution within Xception [5]. Because articles proposing PAC by H. Su *et al.* were discovered while working on late stages of the thesis, it was decided not to change the backbone.

Chapter 6

Conclusion

An operation called Local Context Masked Convolution (LCMC) was proposed, implemented and evaluated. No performance improvements were observed in networks with the proposed novel operation. In fact, performance of networks with the operation degraded. After being compared to research of Guizilini *et al.* [16], it is possible to assume that the method produces better results when applied to transposed convolution instead.

The LCMC operation was inserted into network architectures traditionally used for the depth completion and semantic segmentation tasks. A seemingly fair training environment was used to train network variations with and without the novel operation. Their performance was evaluated and compared.

Multiple potential areas of future research were discovered throughout work on this thesis. Among those were: evaluation of the method with a larger kernel size, training the networks with a more advanced training process and reaching parameter convergence, analyzing weight initiliazation of the method, analyzing use of different functions to simulate the context dependency, analyzing differences between the method and similar, previously succesful methods.

Implementation and results of further experiments will be made public on `https://github.com/paplhjak/Contextus`.

Bibliography

[1] Hassan Alhaija, Siva Mustikovela, Lars Mescheder, Andreas Geiger, and Carsten Rother. Augmented reality meets computer vision: Efficient data generation for urban driving scenes. *International Journal of Computer Vision (IJCV)*, 2018.

[2] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D data in indoor environments. *International Conference on 3D Vision (3DV)*, 2017.

[3] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, Apr 2018.

[4] Bowen Cheng, Maxwell D. Collins, Yukun Zhu, Ting Liu, Thomas S. Huang, Hartwig Adam, and Liang-Chieh Chen. Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation, 2019.

[5] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[6] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding, 2016.

[7] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[9] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2016. cite arxiv:1603.07285.

[10] Kunihiko Fukushima. Cognitron: A self-organizing multilayer neural network. *Biological Cybernetics*, 20:121–136, 1975.

[11] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[12] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[13] Jakob Geyer, Yohannes Kassahun, Mentar Mahmudi, Xavier Ricou, Rupesh Durgesh, Andrew S. Chung, Lorenz Hauswald, Viet Hoang Pham, Maximilian Mühlegg, Sebastian Dorn, Tiffany Fernandez, Martin Jänicke, Sudesh Mirashi, Chiragkumar Savani, Martin Sturm, Oleksandr Vorobiov, Martin Oelker, Sebastian Garreis, and Peter Schuberth. A2D2: Audi Autonomous Driving Dataset. 2020.

[14] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[15] Vitor Guizilini, Rares Ambrus, Sudeep Pillai, and Adrien Gaidon. Packnet-sfm: 3d packing for self-supervised monocular depth estimation. *CoRR*, abs/1905.02693, 2019.

[16] Vitor Guizilini, Rui Hou, Jie Li, Rares Ambrus, and Adrien Gaidon. Semantically-guided representation learning for self-supervised monocular depth, 2020.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec 2015.

[19] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[20] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 1952.

[21] Alan L Hodgkin, Andrew F Huxley, and Bernard Katz. Measurement of current-voltage relations in the membrane of the giant axon of loligo. *The Journal of physiology*, 116(4):424–448, 1952.

[22] Allan L Hodgkin and Andrew F Huxley. The components of membrane conductance in the giant axon of loligo. *The Journal of physiology*, 116(4):473–496, 1952.

[23] Allan L Hodgkin and Andrew F Huxley. Currents carried by sodium and potassium ions through the membrane of the giant axon of loligo. *The Journal of physiology*, 116(4):449–472, 1952.

[24] Allan L Hodgkin and Andrew F Huxley. The dual effect of membrane potential on sodium conductance in the giant axon of loligo. *The Journal of physiology*, 116(4):497–506, 1952.

[25] D. Hubel and T. Wiesel. Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *Journal of Physiology*, 160:106–154, 1962.

[26] David H. Hubel and Torsten N. Wiesel. Receptive fields of single neurons in the cat's striate cortex. *Journal of Physiology*, 148:574–591, 1959.

[27] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[29] Jason Ku, Ali Harakeh, and Steven L. Waslander. In defense of classical image processing: Fast depth completion on the cpu. *2018 15th Conference on Computer and Robot Vision (CRV)*, May 2018.

[30] Y. Le Cun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard. Handwritten digit recognition: Applications of neural network chips and automatic learning. *Comm. Mag.*, 27(11):41–46, November 1989.

[31] Yann Lecun and F. Fogelman Soulie. Modeles connexionnistes de l'apprentissage. *Intellectica*, (special issue apprentissage et machine), March 1987.

[32] Warren Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

[33] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[34] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[35] Y. E. NESTEROV. A method for solving the convex programming problem with convergence rate o(1/k$\hat{2}$). *Dokl. Akad. Nauk SSSR*, 269:543–547, 1983.

[36] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[37] Jiaxiong Qiu, Zhaopeng Cui, Yinda Zhang, Xingdi Zhang, Shuaicheng Liu, Bing Zeng, and Marc Pollefeys. Deeplidar: Deep surface normal guided depth prediction for outdoor scene from sparse lidar data and single color image. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[38] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 386–408, 1958.

[39] F. Rosenblatt. *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Report (Cornell Aeronautical Laboratory). Spartan Books, 1962.

[40] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536, 1986.

[41] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[42] Hang Su, Varun Jampani, Deqing Sun, Orazio Gallo, Erik Learned-Miller, and Jan Kautz. Pixel-adaptive convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[43] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[44] Jie Tang, Fei-Peng Tian, Wei Feng, Jian Li, and Ping Tan. Learning guided convolutional network for depth completion, 2019.

[45] Andrew Tao, Karan Sapra, and Bryan Catanzaro. Hierarchical multi-scale attention for semantic segmentation, 2020.

[46] Alexandru Telea. An image inpainting technique based on the fast marching method. *Journal of Graphics Tools*, 9, 01 2004.

[47] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. Sparsity invariant cnns. In *International Conference on 3D Vision (3DV)*, 2017.

[48] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[49] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *arXiv preprint arXiv:1608.05442*, 2016.

Appendix A

Comparisons of Depth Completion Networks

position	LCMC	rgb	large batch	context
top left	✓	✓	✓	ResNet
top right	✓	✓	✗	ResNet
middle left	✓	✓	✓	RGB
middle right	✓	✓	✗	RGB
bottom left	✗	✓	✗	✗
bottom right	✗	✗	✗	✗

Table A.1: Layout of depth completion network outputs in comparison images.

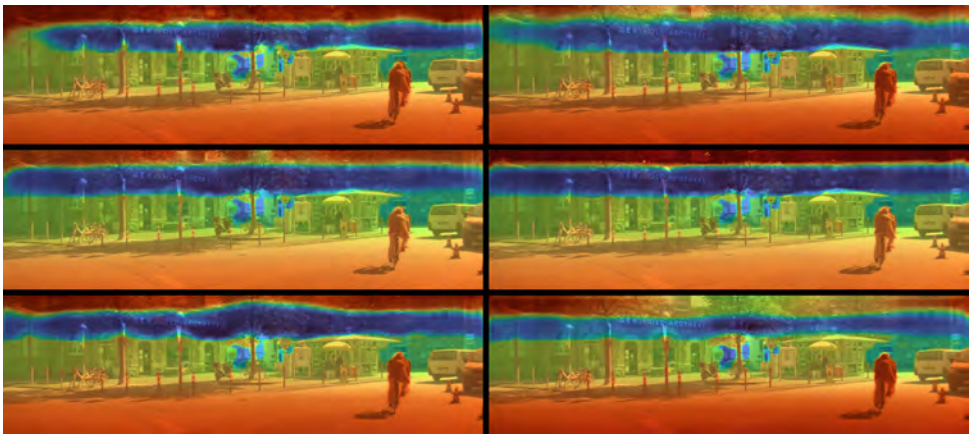


Figure A.1: Comparison of projected depth c. network outputs on KITTI *2011_09_26_drive_0005_sync_image_0000000074_image_02.png*.
Layout of network outputs is specified in table A.1

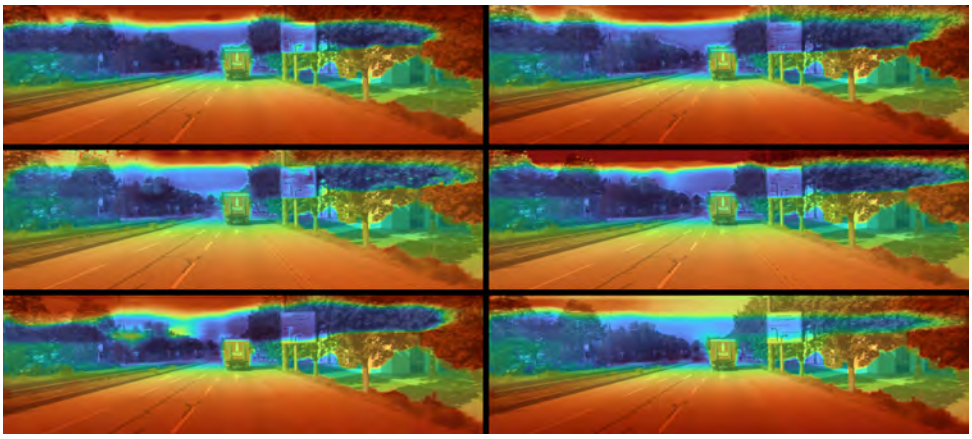


Figure A.2: Comparison of projected depth c. network outputs on KITTI *2011_09_26_drive_0036_sync_image_0000000686_image_02.png*.
Layout of network outputs is specified in table A.1

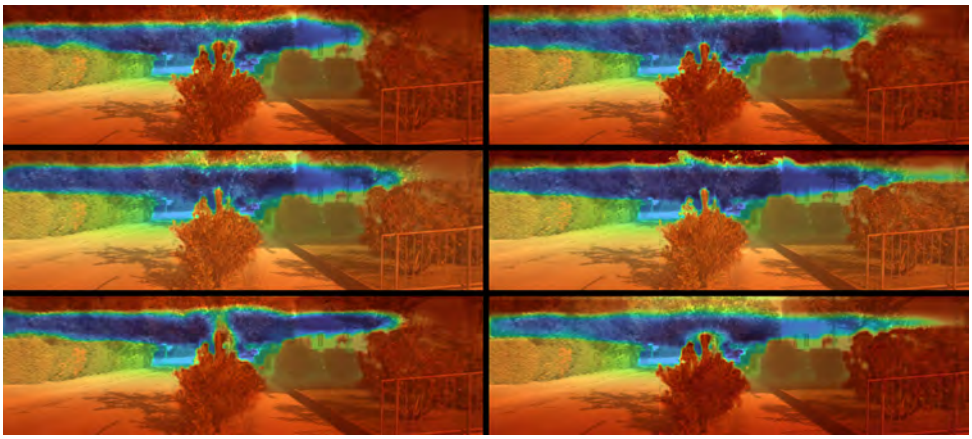


Figure A.3: Comparison of projected depth c. network outputs on KITTI *2011_09_26_drive_0079_sync_image_0000000065_image_03.png*.
Layout of network outputs is specified in table A.1

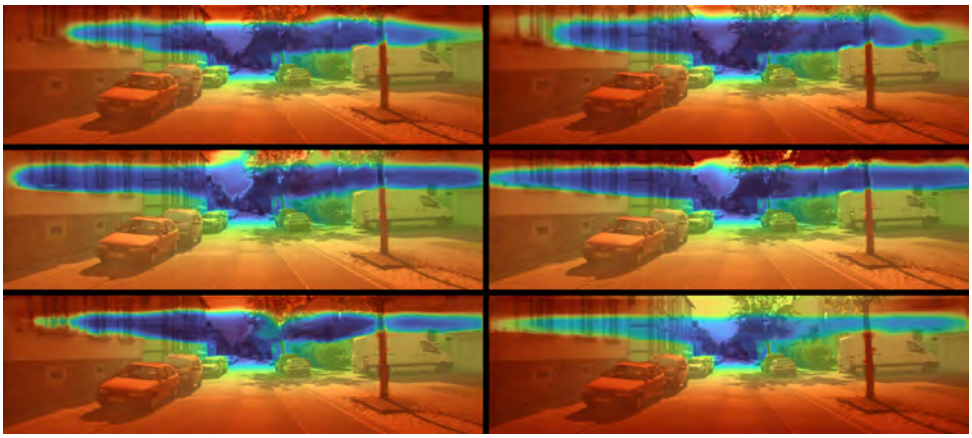


Figure A.4: Comparison of projected depth c. network outputs on KITTI *2011_09_26_drive_0095_sync_image_0000000248_image_02.png*.
Layout of network outputs is specified in table A.1

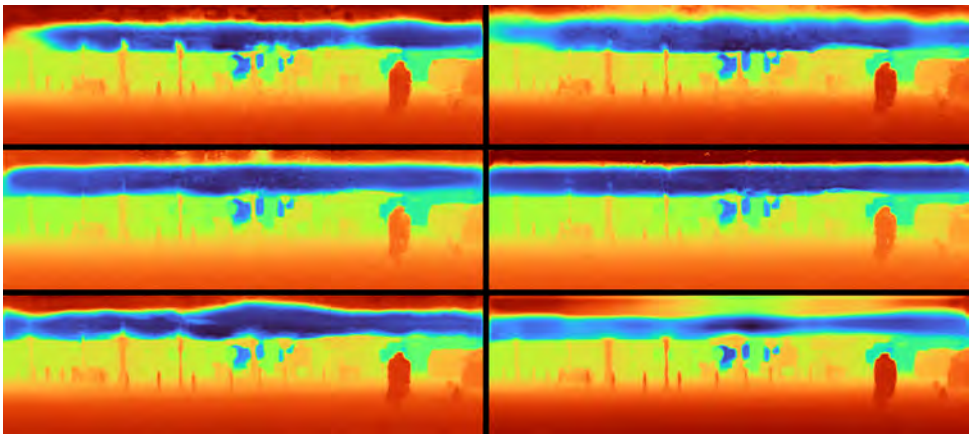


Figure A.5: Comparison of depth completion network outputs on KITTI *2011_09_26_drive_0005_sync_image_0000000074_image_02.png*.
Layout of network outputs is specified in table A.1

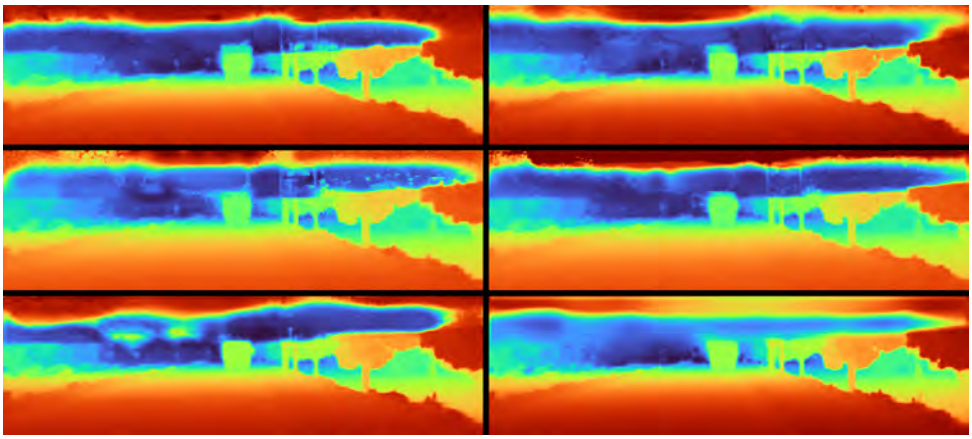


Figure A.6: Comparison of depth completion network outputs on KITTI *2011_09_26_drive_0036_sync_image_0000000686_image_02.png*.
Layout of network outputs is specified in table A.1

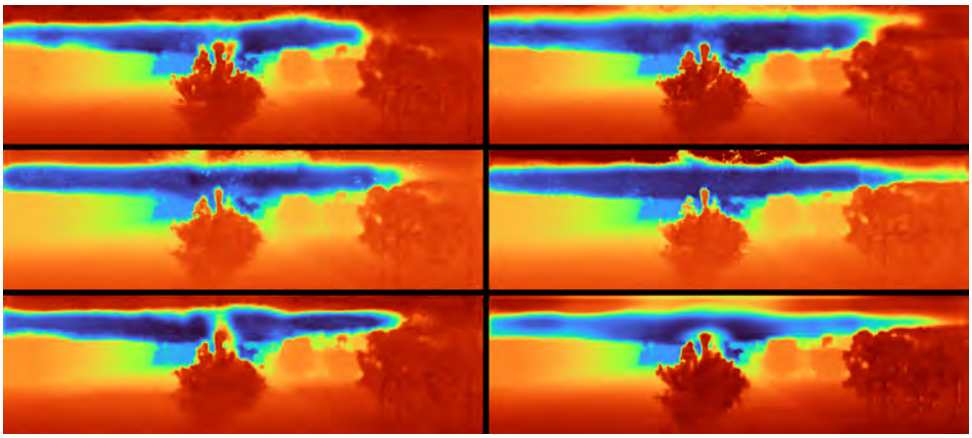


Figure A.7: Comparison of depth completion network outputs on KITTI *2011_09_26_drive_0079_sync_image_0000000065_image_03_pred.png*.
Layout of network outputs is specified in table A.1

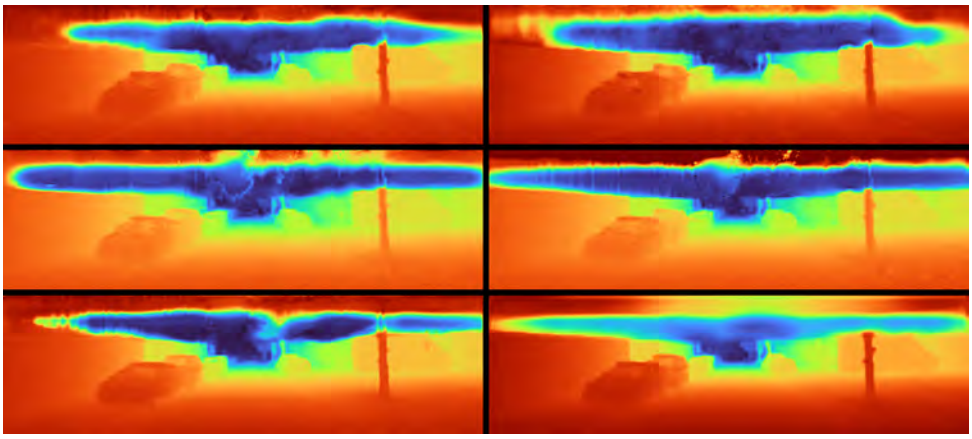


Figure A.8: Comparison of depth completion network outputs on KITTI *2011_09_26_drive_0095_sync_image_0000000248_image_02.png*.
Layout of network outputs is specified in table A.1

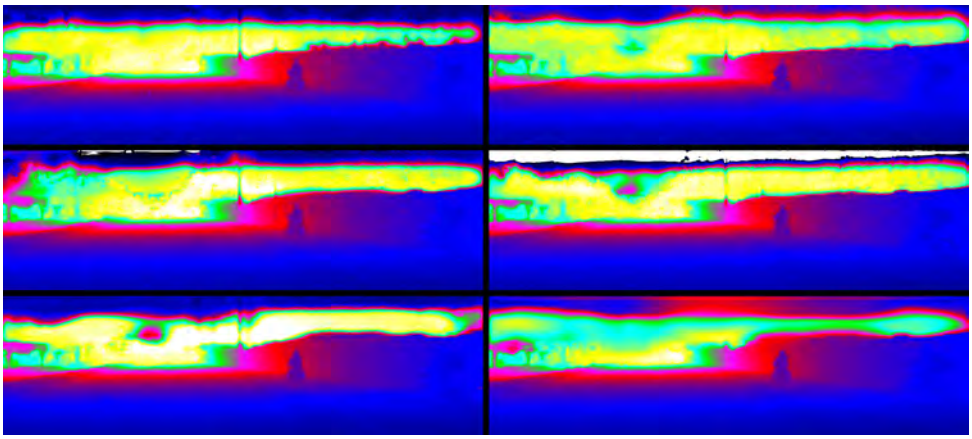


Figure A.9: Comparison of depth completion network outputs on KITTI *2011_09_26_drive_0002_sync_0000000005_image_02.png*.
Layout of network outputs is specified in table A.1

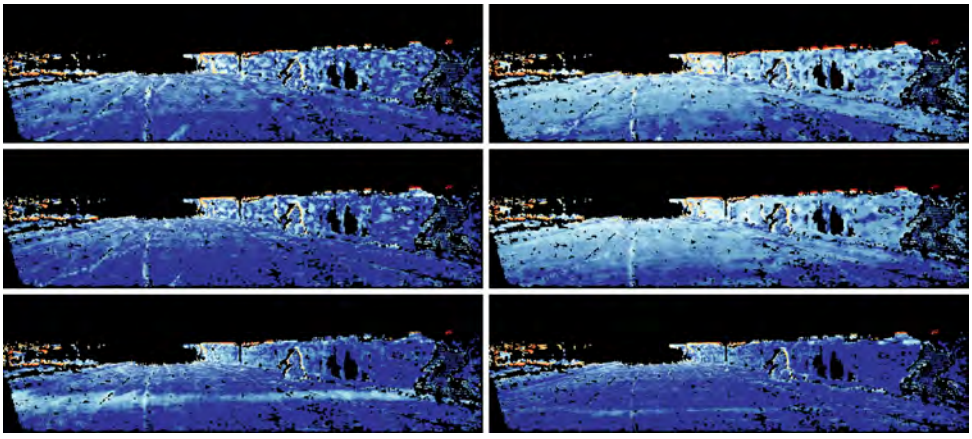


Figure A.10: Comparison of depth c. network output errors on KITTI *2011_09_26_drive_0002_sync_0000000005_image_02.png*.
Layout of network outputs is specified in table A.1

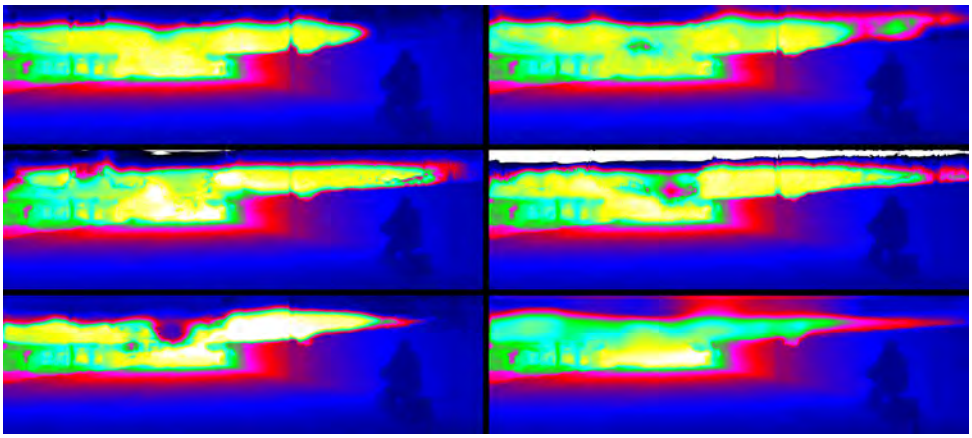


Figure A.11: Comparison of depth completion network outputs on KITTI *2011_09_26_drive_0002_sync_0000000023_image_02.png*.
Layout of network outputs is specified in table A.1

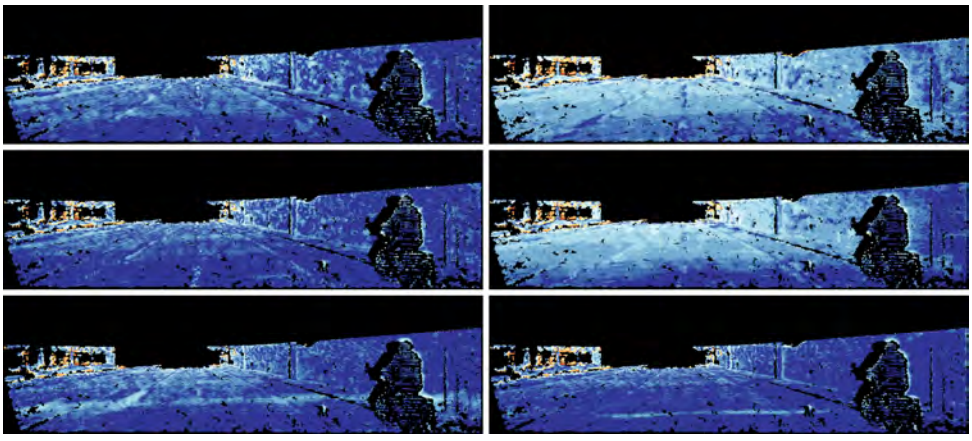


Figure A.12: Comparison of depth c. network output errors on KITTI *2011_09_26_drive_0002_sync_0000000023_image_02.png*.
Layout of network outputs is specified in table A.1

Appendix B

Comparisons of Semantic Segmentation Networks

position	LCMC	coarse	depth
top left	✓	✓	✓
top right	✓	✗	✓
middle left	✗	✓	✓
middle right	✗	✗	✓
bottom left	✗	✓	✗
bottom right	✗	✗	✗

Table B.1: Layout of semantic segmentation network outputs in comparison images.

Figure B.1: Segmentation ground truth of *munster_000055_000019.png*.

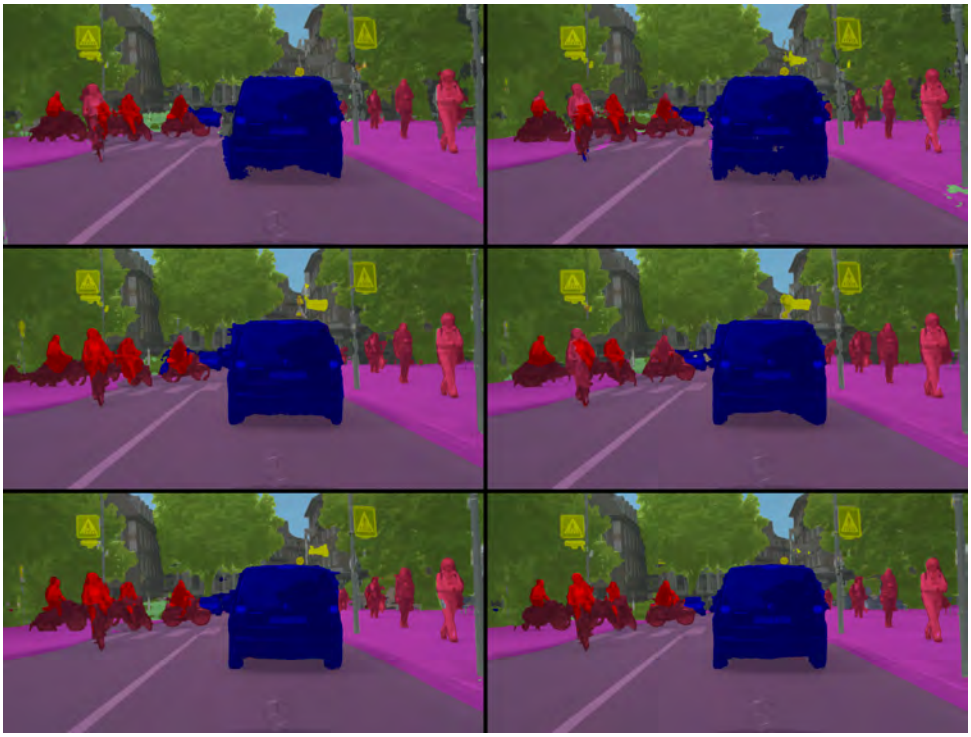


Figure B.2: Segmentation outputs of *munster_000055_000019*.
Layout of network outputs is specified in table B.1

Figure B.3: Segmentation correctness of *munster_000055_000019*.
Layout of network outputs is specified in table B.1

Figure B.4: Segmentation ground truth of *munster_000089_000019.png*.

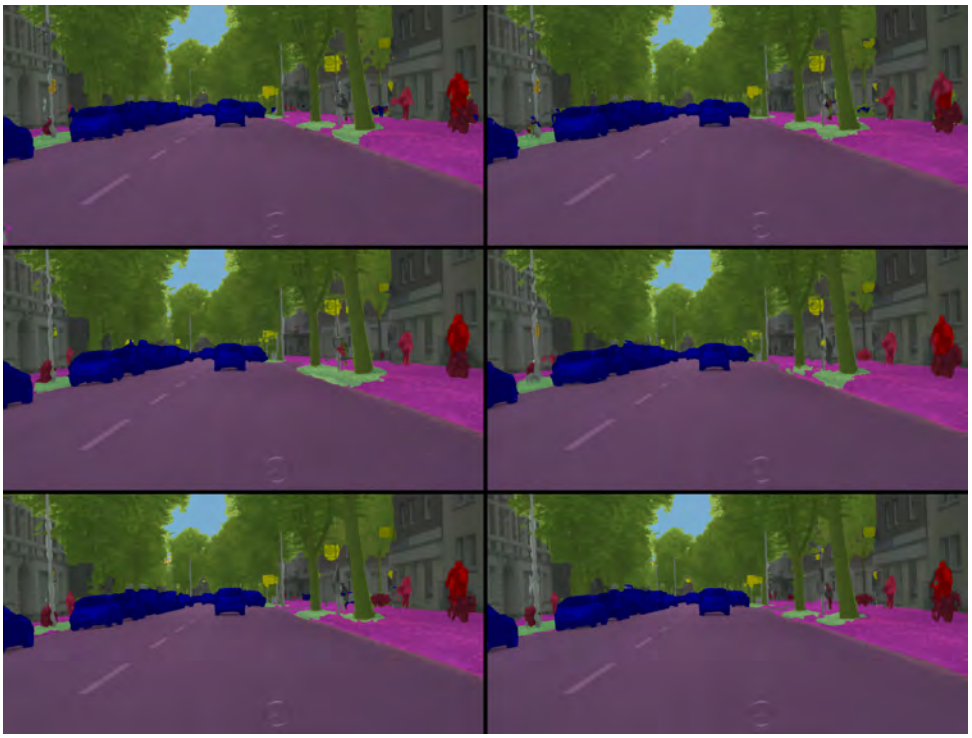


Figure B.5: Segmentation outputs of *munster_000089_000019*.
Layout of network outputs is specified in table B.1

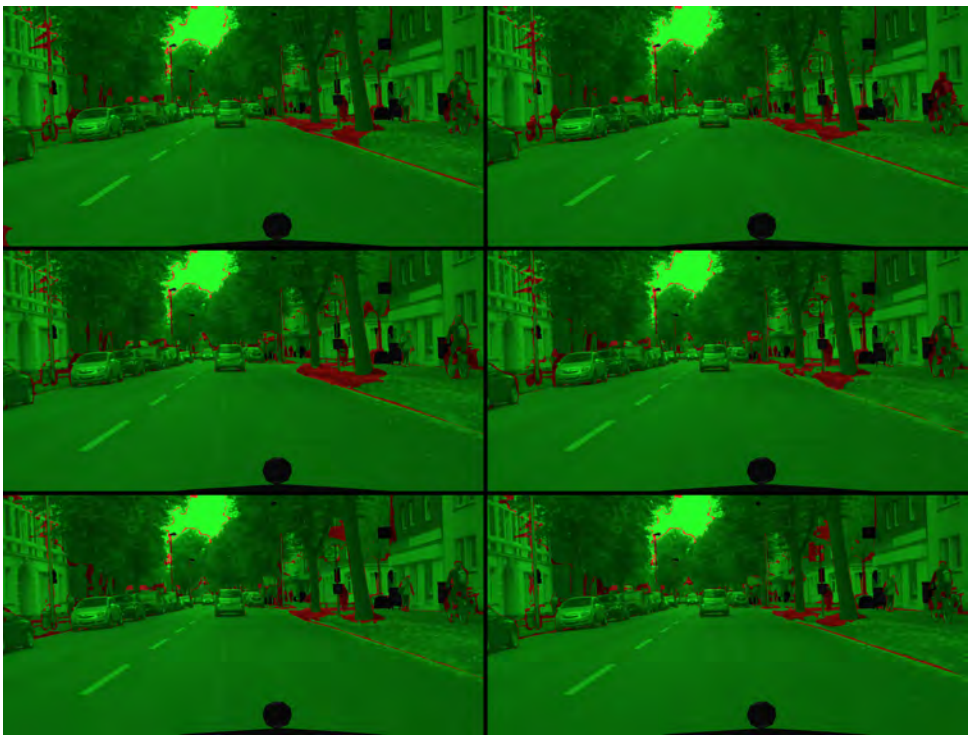


Figure B.6: Segmentation correctness of *munster_000089_000019*. Layout of network outputs is specified in table B.1

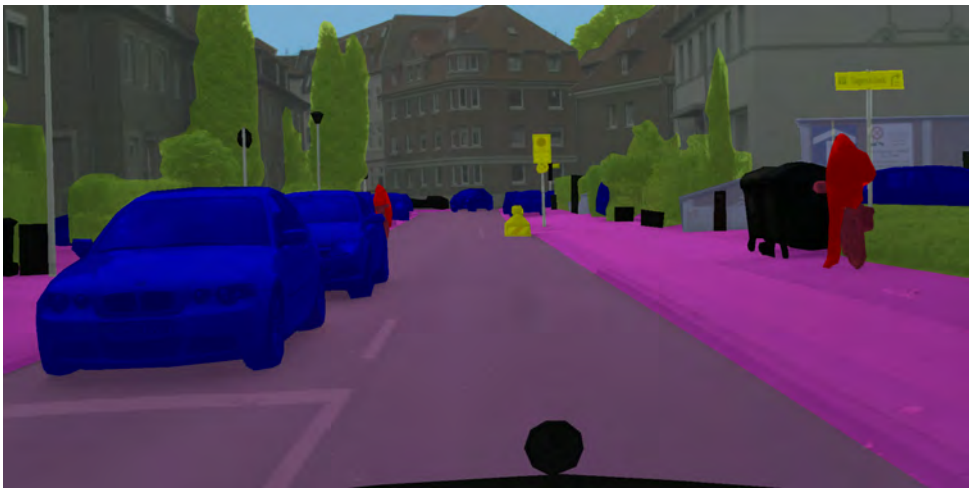


Figure B.7: Segmentation ground truth of *munster_000095_000019.png*.

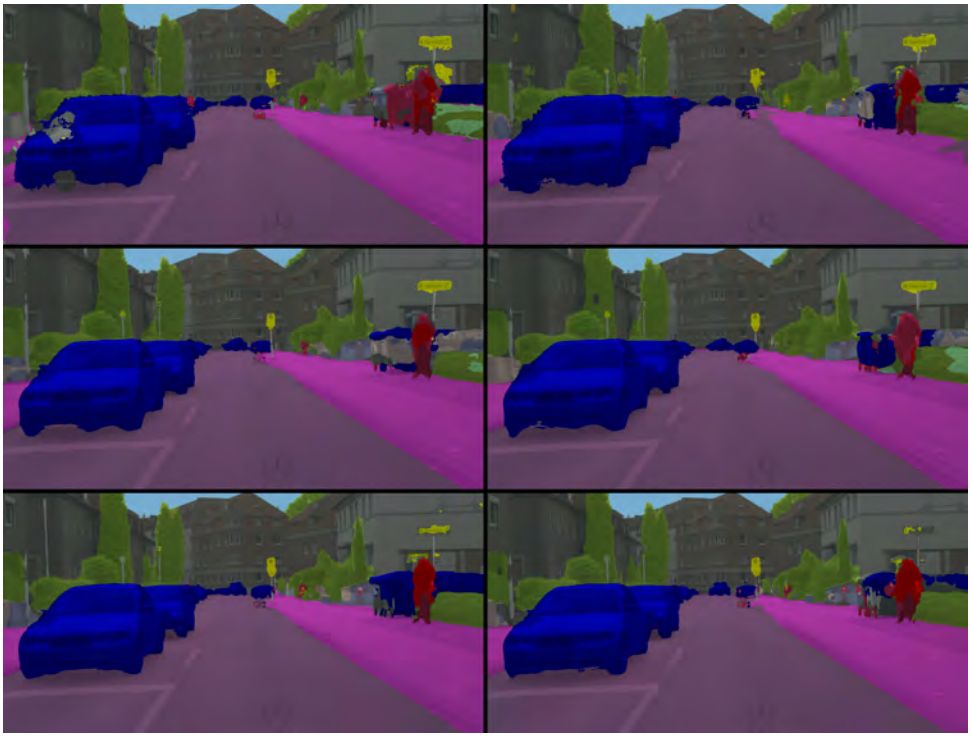


Figure B.8: Segmentation outputs of *munster_000095_000019*.
Layout of network outputs is specified in table B.1

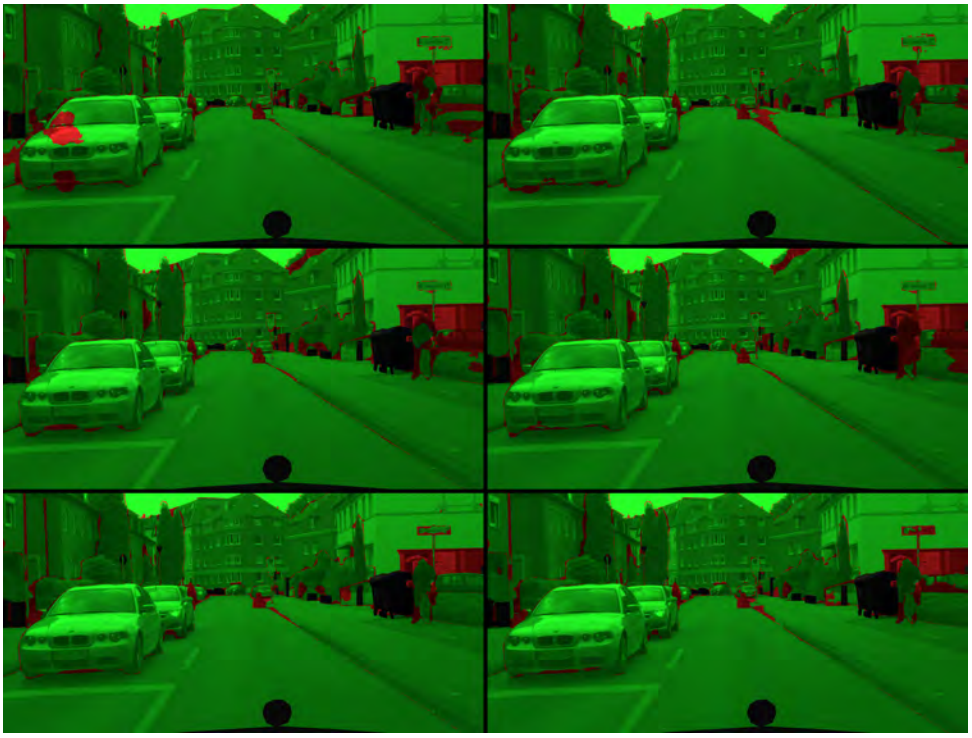


Figure B.9: Segmentation correctness of *munster_000095_000019*.
Layout of network outputs is specified in table B.1

Figure B.10: Segmentation ground truth of *munster_000172_000019.png*.

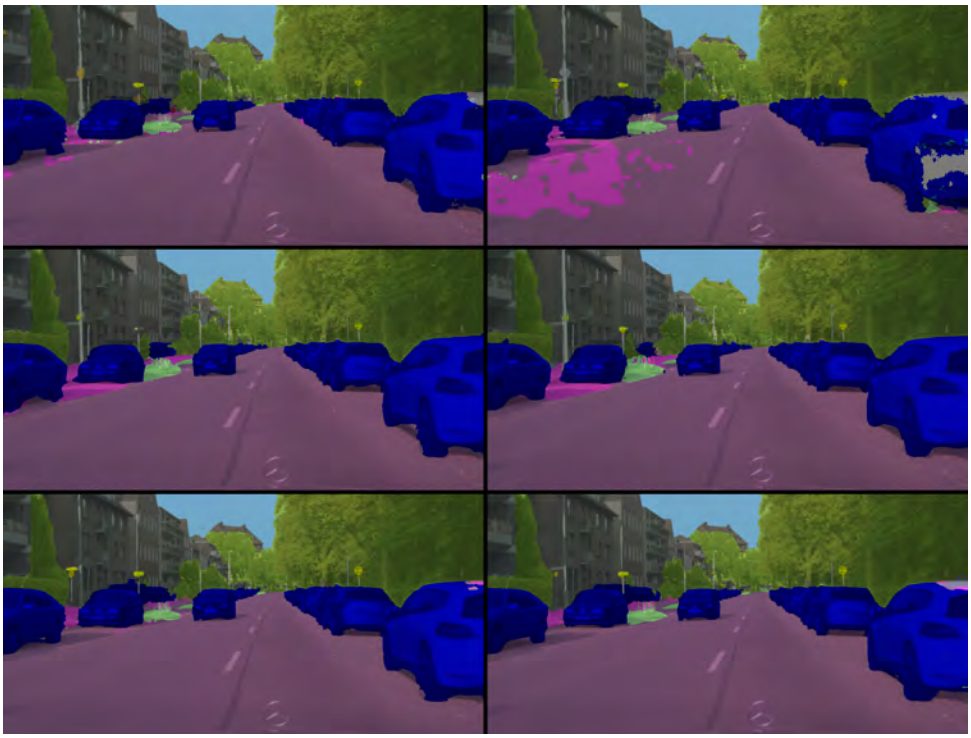


Figure B.11: Segmentation outputs of *munster_000172_000019*.
Layout of network outputs is specified in table B.1

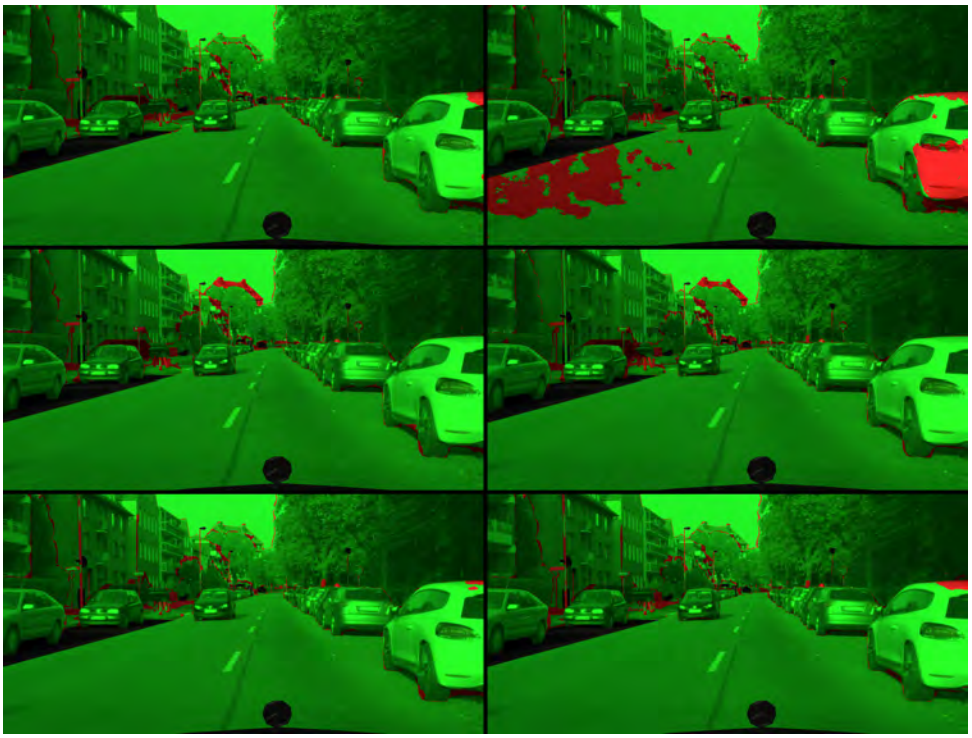


Figure B.12: Segmentation correctness of *munster_000172_000019*. Layout of network outputs is specified in table B.1

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Paplhám Jakub** Personal ID number: **474482**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Convolutional Neural Networks with Local Context Masks

Bachelor’s thesis title in Czech:

Konvoluční neuronové sítě s lokálními kontextovými maskami

Guidelines:

1. Design and implement a local context mechanism based on the sparse convolutions from [1] where the local context corresponds to local real-valued masks which assign higher weights to contextually close neighboring pixels and lower weights to contextually distant pixels.
2. Consider (at least) the following types of the context:
◦ spatial context based on depth or 3D distance (from depth and intrinsic camera parameters) of respective pixels,
◦ semantic context based on a class label (distribution) distance or learnt inter-class dependencies,
◦ appearance context based on RGB distance.
3. Study relevant literature on sparse depth completion and semantic segmentation, such as [2, 3, 4, 5], or other relevant papers from KITTI dataset [6] scoreboards for depth completion and semantic segmentation.
4. Design, implement, train, and evaluate a sparse depth completion and semantic segmentation method using the local context mechanism above. Consider RGB and sparse depth input modalities.
5. Optionally, consider also other application areas besides sparse depth completion and semantic segmentation.
6. In evaluation, consider both externally provided context sources, such as class labels from manual annotation or measured depth, and a recurrent setting where the output of the CNN from the first forward pass is used as the context source for the next forward pass.
7. Investigate marginal effects of employing various types of local context and relevant components of the method on the selected task (i.e., ablation study). Compare the method to state of the art.

Bibliography / sources:

[1] Uhrig, J.; Schneider, N.; Schneider, L.; Franke, U.; Brox, T. & Geiger, A. Sparsity Invariant CNNs. International Conference on 3D Vision (3DV), 2017
[2] Qiu, J.; Cui, Z.; Zhang, Y.; Zhang, X.; Liu, S.; Zeng, B. & Pollefeys, M. DeepLiDAR: Deep Surface Normal Guided Depth Prediction for Outdoor Scene From Sparse LiDAR Data and Single Color Image. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019
[3] Tang, J.; Tian, F.-P.; Feng, W.; Li, J. & Tan, P. Learning Guided Convolutional Network for Depth Completion. arXiv:1908.01238. 2019
[4] Chen, L. C.; Papandreou, G.; Kokkinos, I.; Murphy, K. & Yuille, A. L. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2018, 40, 834-848
[5] Chollet, F. Xception: Deep Learning with Depthwise Separable Convolutions. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, 1800-1807
[6] Geiger, A.; Lenz, P.; Stiller, C. & Urtasun, R. Vision meets Robotics: The KITTI Dataset. The International Journal of Robotics Research, 2013, 32, 1231-1237

Name and workplace of bachelor's thesis supervisor:

Ing. Tomáš Petříček, Ph.D., Vision for Robotics and Autonomous Systems, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **09.01.2020** Deadline for bachelor thesis submission: **14.08.2020**

Assignment valid until: **30.09.2021**

Ing. Tomáš Petříček, Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature