Thesis for the degree of Licentiate of Engineering

Deep Learning Applications for Autonomous Driving

LUCA CALTAGIRONE

Department of Mechanics and Maritime Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2018

Deep Learning Applications
for Autonomous Driving
LUCA CALTAGIRONE

THESIS FOR LICENTIATE OF ENGINEERING no 2018:02

Department of Mechanics and Maritime Sciences
Chalmers University of Technology
SE–412 96 Göteborg
Sweden
Telephone: +46 (0)31–772 1000

Chalmers Reproservice
Göteborg, Sweden 2018

Deep Learning Applications for Autonomous Driving

Luca Caltagirone
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

This thesis investigates the usefulness of deep learning methods for solving two important tasks in the field of driving automation: (i) Road detection, and (ii) driving path generation. Road detection was approached using two strategies: The first one considered a bird's-eye view of the driving scene obtained from LIDAR data, whereas the second carried out camera-LIDAR fusion in the camera perspective. In both cases, road detection was performed using fully convolutional neural networks (FCNs). These two approaches were evaluated on the KITTI road benchmark and achieved state-of-the-art performance, with MaxF scores of 94.07% and 96.03%, respectively.

Driving path generation was accomplished with an FCN that integrated LIDAR top-views with GPS-IMU data and driving directions. This system was designed to simultaneously carry out perception and planning using as training data real driving sequences that were annotated automatically. By testing several combinations of input data, it was shown that the FCN having access to all the available sensors and the driving directions obtained the best overall accuracy with a MaxF score of 88.13%, about 4.7 percentage points better than the FCN that could use only LIDAR data.

Keywords: deep learning, computer vision, autonomous driving, robotic perception and planning, sensor fusion

To Jaime, Isaac, and Dahlia

Acknowledgements

I would first like to express gratitude to my supervisor Prof. Mattias Wahde for his faith in me and for his guidance in taking my academic career in the directions that I have.

My heartfelt thanks also go to my co-supervisor Prof. Lennart Svensson for his thoughtful advice and many stimulating discussions. I would like to thank Dr. Mauro Bellone for sharing his knowledge and for helping me keep my Italian alive.

I am very grateful to the PRELAT project for funding the work presented in this thesis and to Dr. Martin Sanfridson for keeping the project running. Last but not least, I wish to thank all my colleagues at VEAS for creating a very welcoming and friendly working environment.

List of included papers

This thesis consists of the following papers. References to the papers will be made using Roman numerals.

I. Caltagirone, L., Scheidegger, S., Svensson, L., and Wahde, M., *Fast LIDAR-based Road Detection Using Fully Convolutional Neural Networks.* In: Proceedings of the 28th IEEE Intelligent Vehicles Symposium, Los Angeles, CA, pp. 1019-1024, 2017.

II. Caltagirone, L., Bellone, M., Svensson, L., and Wahde, M., *LIDAR-based Driving Path Generation Using Fully Convolutional Neural Networks.* In: Proceedings of the 20th IEEE International Conference on Intelligent Transportation Systems, Yokohama, Japan, pp. 573-578, 2017.

III. Caltagirone, L., Bellone, M., and Wahde, M., *LIDAR-Camera Fusion for Road Detection using Fully Convolutional Neural Networks.* Submitted to Robotics and Autonomous Systems, January 2018.

Technical terms used in the thesis

Table of Contents

Chapter 1

Introduction and motivation

Every year about 1.3 million people die worldwide in road traffic accidents [27] and 20 to 50 million are injured or disabled. Road traffic accidents are currently the leading cause of death among young people aged 15 to 29 years. Besides human suffering, road traffic accidents also cause the loss of about 3% of global GDP [6]. According to the *Tri-level study of the causes of traffic accidents* [26], human factors were a probable cause in about 93% of the examined accidents. The above numbers clearly indicate that if human errors and deficiencies were removed from driving, there would be enormous savings both economically and in terms of human lives [2]. This consideration is one of the main motivating factors for the development of vehicles that drive autonomously, that is, vehicles that can understand the world and make decisions by themselves, without human input.

Driving a vehicle is quite a complex skill. Broadly speaking, it can be split into three main components: perception, planning, and control. Perception is the act of apprehending the environment in the immediate surroundings of the vehicle and it includes tasks such as recognizing whether there are other traffic participants and, if so, estimating their relative position and velocity, locating the road and its lanes and estimating the vehicle position within them, monitoring the vehicle state, and looking for traffic signs, traffic lights, and other relevant landmarks. Given this knowledge, a driver can plan the desired vehicle trajectory (planning) and then follow it by applying appropriate actions to the vehicle's steering wheel, brakes, throttle, and gear shift (control).

The ultimate goal of driving automation is to develop computer systems that can carry out the previously described functions in all situations, effectively replacing the human driver. This is called level 5 automation, or *full automation*, according to the driving automation levels for on-road vehicles defined by SAE [25]. Many automotive and technology companies (e.g., Volvo Cars, Toyota, Google-Waymo, Apple), universities (e.g., Chalmers University of Technology, Oxford University, Stanford University), and startups (e.g., Otto, Drive.ai) are currently working towards this goal and much progress has been made in the past two decades. However, there are still many open problems to solve, in particular regarding the perception function in challenging scenarios (e.g., rainy and snowy weather, nighttime, etc.).

Given that the transportation infrastructure has been tailored for human drivers who mainly rely on vision for sensing the environment, a natural choice is to equip autonomous vehicles with cameras in order for them to perceive the world. The information content of camera images is, however, strongly dependent upon the external lightning conditions and can be degraded to a large degree in certain circumstances. It is therefore common to equip autonomous vehicles with additional sensors such as radars, sonars, and LIDARs that can provide a complementary or alternative view of the environment. For example, whereas cameras provide information about emitted or reflected light from surrounding objects (passive sensing), LIDARs generate a sparse 3D representation of the scene by using their own emitted light (active sensing).

The information acquired by the sensors must then be processed using computer algorithms that extract the information necessary to control the vehicle. Unfortunately, camera images and 3D point clouds are high-dimensional objects that are quite expensive to process. Only in recent years, with the advent of fast GPUs and novel machine learning techniques, has it become possible to accomplish that goal in real time and with portable computing devices.

As previously mentioned, autonomous driving is dependent on the concerted action of many different functions. Furthermore, each function can be approached using a wide variety of algorithms. In this work, two tasks are investigated: (1) Road segmentation, and (2) driving path generation. Both were addressed using a family of machine learning algorithms known as deep learning [14]. These functionalities were developed and evaluated using the KITTI data set [10]. Finally, two main sensors have been considered, a monocular RGB camera and a multi-layer rotating LIDAR.

The author was the main contributor to all the papers in this thesis.

Chapter 2

Convolutional neural networks

2.1 Artificial neural networks

Artificial neural networks (ANNs) are a family of computing systems that are inspired by biological neural networks. The basic building block of all ANNs is the **artificial neuron**. Artificial neurons (or **units**) receive inputs from other neurons in the network and then generate their own output. The computation of the output as well as the connectivity among neurons depend on the specific application. In fact, many strategies have been developed since McCulloch and Pitts first proposed this type of computational model in 1943 [17]. This work focuses on multi-layer feedforward neural networks, that is, networks where the units are organized in multiple distinct layers, such that the information propagates only in one direction. The right panel of Fig. 2.1 shows an example of a feedforward ANN with three layers: (1) The leftmost layer is called the **input layer** and its nodes simply return the input data. (2) The inner layer processes the information coming from the input layer and its result is forwarded to the following layer. (3) The rightmost layer, or **output layer**, carries out further processing and generates the ANN's final output. Layers that are placed between the input and output layers are called **hidden layers**. The ANN example just described is a **fully connected network**, that is, each unit in a given layer is connected to all the units in the previous and following layers. This connectivity is not efficient for working with high-dimensional inputs, such as images, because the number of connections becomes very large. Moreover, the fully connected architecture does not explicitly exploit the spatial structure of the information.

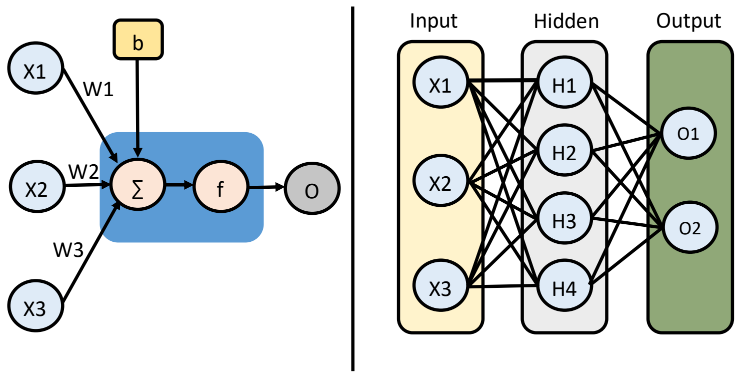

Figure 2.1: *(Left panel): A simple illustration of an artificial neuron receiving inputs from three units. The artificial neuron computes the sum of its inputs* $\{x_i\}_{i=1,2,3}$ *multiplied by their corresponding connection weights* $\{w_i\}_{i=1,2,3}$*, adds a bias term* b*, applies an activation function* f*, and then sends its output forward to other units. (Right panel) Illustration of a three-layer feedforward neural network. The information flows from the input to the output layer passing through one intermediate layer, called the hidden layer. The ANN is fully connected meaning that each unit in a given layer is connected to all the units in the preceding and following layer.*

2.1.1 Artificial neuron

The output of an artificial neuron (see, for example, the left panel of Fig. 2.1) is computed by applying a function f, also known as an **activation function**, to the weighted sum of all its K inputs plus a bias term b, that is:

$$o = f\left(\sum_{i=1}^{K} w_i x_i + b\right), \tag{2.1}$$

where the coefficients w_i represent the strength of the connections (or **weights**) between the artificial neuron and its input units.

Over the years, many activation functions have been proposed, from the step function used in the perceptron [23] to the sigmoid logistic function. All of them, however, are nonlinear: This is a crucial requirement since otherwise, any number of layers could always be reduced to one, given that the composition of multiple linear transformations can be expressed as a single linear transformation. Nowadays, the most commonly used activation function is the rectifier [14] that is defined as follows:

$$f(x) = \max(0, x). \tag{2.2}$$

A neuron that utilizes the above transfer function is called a **rectified linear unit** (ReLU) [19]. Another popular activation function that has been shown to speed up learning and that is used in this work is the **exponential linear unit** (ELU) [5] :

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ a(\mathrm{e}^x - 1) & \text{otherwise} \end{cases}$$

where the parameter a is commonly set to one.

2.2 Convolutional neural networks

As mentioned in Sect. 2.1, fully connected neural networks are not efficient for processing high-dimensional inputs such as images. By studying the human visual cortex, it was observed that individual neurons only have access to a small portion, called the **receptive field**, of the entire visual field [12]. Information is then aggregated by neurons that have overlapping receptive fields and neurons in deeper layers of the visual cortex. Biological considerations then suggest that local connectivity and hierarchical layers might be better suited for processing visual information. These observations were the inspiration for the development of the **convolutional layer** which is the building block of **convolutional neural networks** (CNNs) [14].

2.2.1 Convolutional layer

In contrast to a fully connected layer, artificial neurons in a convolutional layer only have access to a limited region of the preceding layer, a region specified by their receptive field size. If this size is small, the number of connections is reduced significantly in comparison with a fully connected layer. Another important feature of convolutional layers is that the weights are shared among units which further decreases the number of parameters. The rationale behind this design choice is that discriminative features can be found anywhere in the visual field; a pattern of connectivity that detects vertical edges will likely be useful in the top right corner of an image just as much as in its bottom left corner.

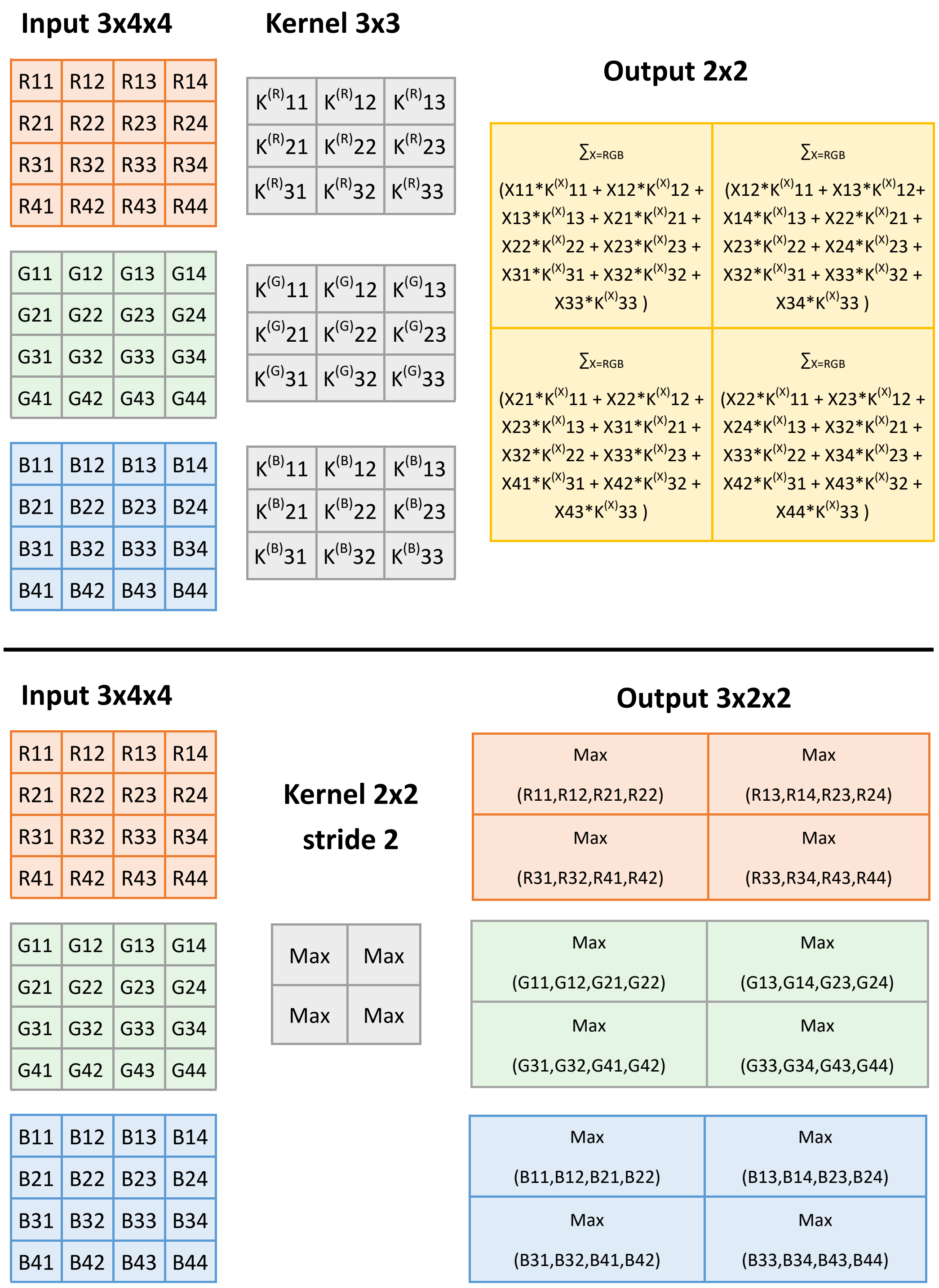
Input 3x4x4
R11 R12 R13 R14
R21 R22 R23 R24
R31 R32 R33 R34
R41 R42 R43 R44
G11 G12 G13 G14
G21 G22 G23 G24
G31 G32 G33 G34
G41 G42 G43 G44
B11 B12 B13 B14
B21 B22 B23 B24
B31 B32 B33 B34
B41 B42 B43 B44
Kernel 3x3
$K^{(R)}11$ $K^{(R)}12$ $K^{(R)}13$
$K^{(R)}21$ $K^{(R)}22$ $K^{(R)}23$
$K^{(R)}31$ $K^{(R)}32$ $K^{(R)}33$
$K^{(G)}11$ $K^{(G)}12$ $K^{(G)}13$
$K^{(G)}21$ $K^{(G)}22$ $K^{(G)}23$
$K^{(G)}31$ $K^{(G)}32$ $K^{(G)}33$
$K^{(B)}11$ $K^{(B)}12$ $K^{(B)}13$
$K^{(B)}21$ $K^{(B)}22$ $K^{(B)}23$
$K^{(B)}31$ $K^{(B)}32$ $K^{(B)}33$
Output 2x2
$\sum_{X=RGB}$ (X11*$K^{(X)}$11 + X12*$K^{(X)}$12 + X13*$K^{(X)}$13 + X21*$K^{(X)}$21 + X22*$K^{(X)}$22 + X23*$K^{(X)}$23 + X31*$K^{(X)}$31 + X32*$K^{(X)}$32 + X33*$K^{(X)}$33)
$\sum_{X=RGB}$ (X12*$K^{(X)}$11 + X13*$K^{(X)}$12+ X14*$K^{(X)}$13 + X22*$K^{(X)}$21 + X23*$K^{(X)}$22 + X24*$K^{(X)}$23 + X32*$K^{(X)}$31 + X33*$K^{(X)}$32 + X34*$K^{(X)}$33)
$\sum_{X=RGB}$ (X21*$K^{(X)}$11 + X22*$K^{(X)}$12 + X23*$K^{(X)}$13 + X31*$K^{(X)}$21 + X32*$K^{(X)}$22 + X33*$K^{(X)}$23 + X41*$K^{(X)}$31 + X42*$K^{(X)}$32 + X43*$K^{(X)}$33)
$\sum_{X=RGB}$ (X22*$K^{(X)}$11 + X23*$K^{(X)}$12 + X24*$K^{(X)}$13 + X32*$K^{(X)}$21 + X33*$K^{(X)}$22 + X34*$K^{(X)}$23 + X42*$K^{(X)}$31 + X43*$K^{(X)}$32 + X44*$K^{(X)}$33)
Input 3x4x4
R11 R12 R13 R14
R21 R22 R23 R24
R31 R32 R33 R34
R41 R42 R43 R44
G11 G12 G13 G14
G21 G22 G23 G24
G31 G32 G33 G34
G41 G42 G43 G44
B11 B12 B13 B14
B21 B22 B23 B24
B31 B32 B33 B34
B41 B42 B43 B44
Kernel 2x2
stride 2
Max Max
Max Max
Output 3x2x2
Max (R11,R12,R21,R22)
Max (R13,R14,R23,R24)
Max (R31,R32,R41,R42)
Max (R33,R34,R43,R44)
Max (G11,G12,G21,G22)
Max (G13,G14,G23,G24)
Max (G31,G32,G41,G42)
Max (G33,G34,G43,G44)
Max (B11,B12,B21,B22)
Max (B13,B14,B23,B24)
Max (B31,B32,B41,B42)
Max (B33,B34,B43,B44)

Figure 2.2: *(Top panel) Illustration of the convolution operation carried out using a kernel of size 3×3 and stride 1. (Bottom panel) Max pooling operation performed using a filter of size 2×2 and stride 2.*

A convolutional layer processes information structured as multidimensional arrays (henceforth: tensors). An RGB image, for example, can be represented as a 3D tensor of size $D \times H \times W$, where D denotes the depth (or number of channels, which in this case equals three), H is the height, and W is the width. Given a **kernel** K of size $F \times F$ and an input tensor I, the **convolution operation** is defined as follows:

$$O(i, j) = (I * K)(i, j) = \sum_{l=1}^{D} \sum_{m=1}^{F} \sum_{n=1}^{F} I^l(i + m, j + n)K^l(m, n), \tag{2.3}$$

The top panel of Fig. 2.2 illustrates this operation: given an input of size $3 \times 4 \times 4$ and a kernel of size 3×3, an output of size 2×2 is produced by sliding the kernel window over the input tensor and applying the convolution operation. The **stride** of the convolution, denoted as S, controls how many cells the kernel should slide over each time it is moved. In the example above, the **stride** is set to one. Notice that only the spatial size of the kernel is specified, in this case $F = 3$; its depth is always constrained to be equal to the number of channels of the tensor it is applied to. As can be seen in Fig. 2.2, the spatial size of the output is smaller than the input. In general, the output size can be computed according to the following equation:

$$W_{\text{out}} = \frac{(W_{\text{in}} - F)}{S} + 1 \tag{2.4}$$

$$H_{\text{out}} = \frac{(H_{\text{in}} - F)}{S} + 1 \tag{2.5}$$

In some cases, it is desirable to control the spatial size of the output; a common approach to achieve that is to pad the input with zeros before carrying out the convolution, a procedure known as **zero padding**. Letting P denote the amount of zero padding, Equations (2.5) and (2.4) then become:

$$W_{\text{out}} = \frac{(W_{\text{in}} - F + 2P)}{S} + 1 \tag{2.6}$$

$$H_{\text{out}} = \frac{(H_{\text{in}} - F + 2P)}{S} + 1 \tag{2.7}$$

Here, it is assumed that zero padding is applied symmetrically on both spatial dimensions; for example, $P = 1$ corresponds to adding two rows of zeros to the input, one at the top and one at the bottom, and two columns, one to the left and one to the right. This procedure must be applied to each input channel.

The example shown in the top panel of Fig. 2.2 considers only one kernel which transforms the 3D input into a 2D tensor. In practice, however, a convolutional layer usually consists of many kernels and its output is also a 3D tensor. Each channel (i.e., 2D output obtained by applying one kernel) of the output tensor is called a **feature map**. To summarize, the output in channel z of a convolutional layer with Z feature maps is obtained as follows:

$$O_z(i,j) = (I * K_z)(i,j) = \sum_{l=1}^{D}\sum_{m=1}^{F}\sum_{n=1}^{F} I^l(i+m, j+n)K_z^l(m,n) + b_z, \quad (2.8)$$

where now also the scalar bias term, b_z, of feature map z and $z \in (1, \ldots, Z)$ has been included.

2.2.2 Max pooling layer

In order to decrease the spatial size of the tensors processed with convolutional layers, it is common practice to insert in the network one or more **max pooling** layers. In this way, it is possible to reduce the memory requirements for training and running the CNN. Additionally, this operation provides partial translation invariance. A max pooling layer is entirely specified by its filter size F and its stride S. The bottom panel of Fig. 2.2 shows an example of the max pooling operation, considering a filter of size 2×2 and stride 2. The output tensor is obtained by sliding the max pooling filter across the input and by choosing the maximum value within the selected region; this operation is carried out on each feature map individually, so that the output will have the same depth as the input.

2.2.3 Encoder-decoder network

The previous sections presented some of the components that can be used when designing CNNs. It is easy to see that there are countless ways of selecting the layers' **hyperparameters** (e.g., kernel size, stride, number of feature maps, etc.) and their mutual connectivity. This work focuses on one type of architecture known as **encoder-decoder**, an illustration of which is provided in Fig. 2.3. The key ideas of this architecture are (i) to decrease the spatial size of the input through a sequence of max pooling layers while

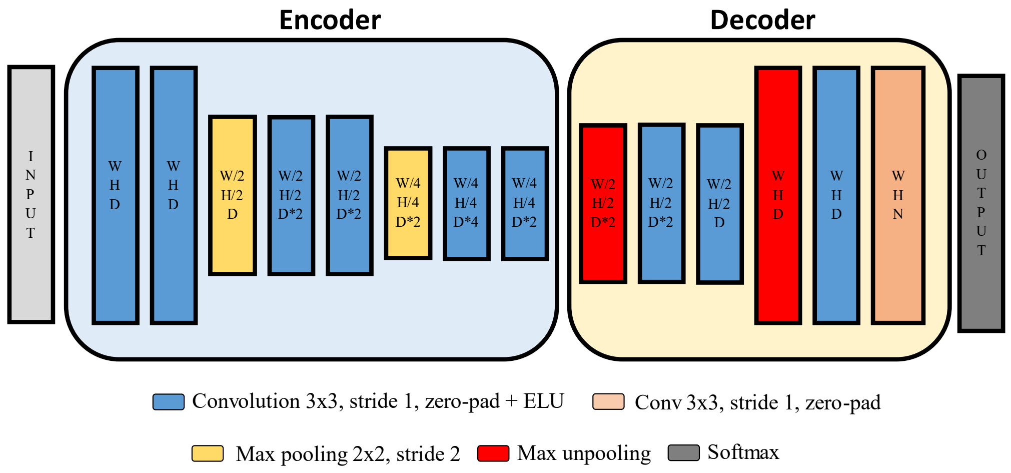

Figure 2.3: *Encoder-decoder CNN. The encoder consist of a sequence of convolutional and max pooling layers. The input spatial size is decreased after each pooling operation, thus reducing the memory requirements. At the same time, the number of trainable parameters is increased by doubling the number of feature maps in the convolutional layers. The decoder carries out further processing while also upsampling the spatial size of the input tensor. W is the input width, H is the input height, and D is the initial number of feature maps. N is the depth of the output tensor.*

at the same time (ii) increasing the depth of the convolutional layers. With this strategy the memory requirements are reduced and the number of tunable parameters is increased with each downsampling stage. The example in Fig. 2.3 shows a network with two downsampling and two upsampling stages. The max pooling layers also store the positions of the maximum values (**pooling indices**) in their input tensors. Those indices can then be used for carrying out the **max unpooling** operation in the decoding part of the network [1]. When a CNN, such as the encoder-decoder presented in this section, does not contain any fully connected layer (see Sect. 2.1), it is commonly referred to as a **fully convolutional neural network** (FCN).

2.3 Training and objective function

Machine learning can be used to solve many different tasks [11]. Two of the most common ones are **classification** and **regression**. In a classification task, the algorithm should specify the class $k \in 1, \ldots, N$ that identifies a given input X. As an example, given an image, the task of the program might be to decide if there is a cat, or a dog, or a fox in it. Regression is similar to classification but, in this case, the output is a continuous variable instead of a discrete one. An example of regression is predicting the price of a house or a stock.

The work presented here is concerned with a task known as **semantic segmentation**, which can be seen as a generalization of the classification example described above. In this case, a program assigns to each pixel of a given image a probability distribution over the possible classes (e.g., road, car, pedestrian, etc.) to which the pixel could belong.

2.3.1 Softmax function

The **softmax function**, here denoted $\boldsymbol{\sigma}$, is used to ensure that the output of an algorithm can be interpreted as a probability distribution. Each component of a vector $z \in R^N$ is transformed as follows:

$$\boldsymbol{\sigma}(z)_i = \frac{\mathrm{e}^{z_i}}{\sum_{j=1}^{N} \mathrm{e}^{z_j}}, \quad i \in 1, \ldots, N \tag{2.9}$$

It is easy to see that the components of the transformed vector are always positive and sum to one. For a semantic segmentation task, a **softmax layer** is usually inserted after the last convolutional layer of a CNN. Its purpose is to apply the softmax function along the depth dimension at each spatial location in order to generate a probability distribution for each pixel.

2.3.2 Cross-entropy loss and optimization

Consider a classification task solved with an ANN parametrized by a vector of weights w and a training example (x, y), where x denotes the input and y represents its corresponding ground truth class. Let $f_w(x) = f(x) = [p_1, \ldots, p_N]$ denote the ANN-predicted probability distribution over the possible classes. The cross-entropy error is then computed as follows:

$$H(x, y) = -\log(f(x)_y) = -\log(p_y) \tag{2.10}$$

The loss function L used for tuning the ANN's weights is commonly defined as the average cross-entropy error computed for the entire training set, that is:

$$L(w) = -\frac{1}{M}\sum_{i=1}^{M}\log(f(x^i)_{y^i}) = -\frac{1}{M}\sum_{i=1}^{M}\log(p^i_{y^i}) \qquad (2.11)$$

where M denotes the total number of training examples.

The minimization of the loss function L is usually carried out using gradient descent and the backpropagation algorithm [24]. In practice, in order to reduce the computational burden, at each iteration the loss function is computed by considering only a randomly selected subset of the entire training set and the optimization algorithm is then known as **stochastic gradient descent** (SGD). The size of the subset is a hyperparameter called the **batch size**.

Chapter 3

Data set and preprocessing

This work makes use of the well-known KITTI Vision Benchmark Suite [10] which consists of many driving sequences logged over a period of several days in and around Karlsruhe, Germany, and includes urban, rural, and freeway scenarios. The data collection was carried out in day time and in fair weather conditions, using several sensors mounted on a Volkswagen Passat B6. Details about the setup can be found in [10]. The sensors relevant for this work are (i) an RGB camera (Point Grey Flea 2 FL2-14S3C-C), (ii) a LIDAR (Velodyne HDL-64E), and (iii) a GPS-IMU system (OXTS RT 3003). Fig. 3.1 shows a detailed illustration of the KITTI sensor platform[1].

3.1 Point cloud top-view representation

In order to process a point cloud efficiently with a CNN, a preprocessing step is carried out for transforming it into a structured 3D tensor. The procedure consists of the following steps: (1) a 2D grid is specified in the x-y plane of the LIDAR coordinate system. Each grid cell has a size of $L \times L$, where L is set to 0.1 meters in this case. The region of space covered by the grid is set according to the specific problem under consideration: In Paper I, it covered $x \in [6, 46]$ and $y \in [-10, 10]$ meters; in Paper II, it covered for $x \in [-30, 30]$ and $y \in [-30, 30]$ meters. (2) Each of the 4D data points, which contains the position $[x, y, z]$ and the reflectivity ρ, is then assigned to its corresponding

[1]http://www.cvlibs.net/datasets/kitti/setup.php

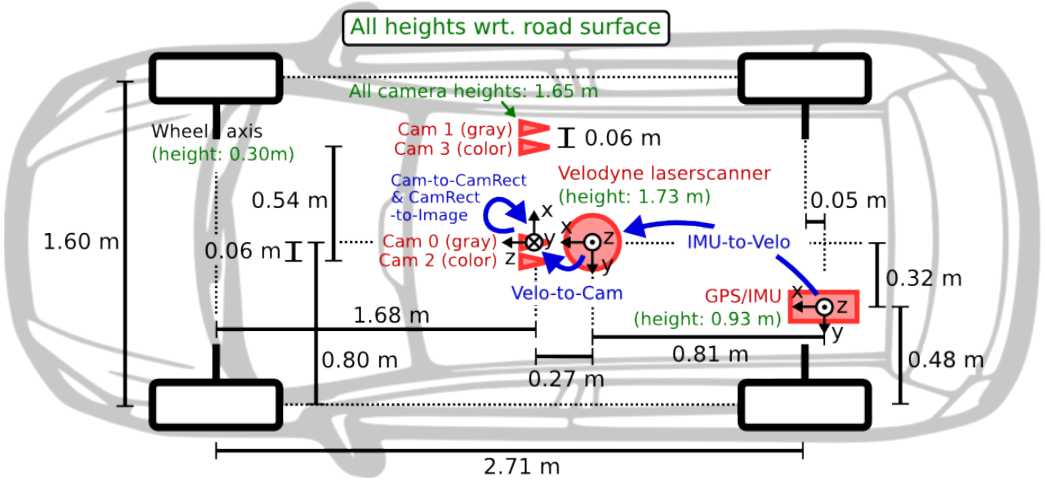

Figure 3.1: *Top view of the sensor platform used for data collection of the KITTI Vision Benchmark Suite [10]. The sensors relevant for this work are Cam 2, which is an RGB camera, the Velodyne laserscanner, and the GPS/IMU.*

cell by projecting it onto the grid:

$$\begin{bmatrix} x \\ y \\ z \\ \rho \end{bmatrix} \xrightarrow[\text{projection}]{\text{grid}} \begin{bmatrix} u \\ v \\ z \\ \rho \end{bmatrix} \tag{3.1}$$

where u and v specify the column and row cell positions in the grid, respectively. (3) In each grid cell, several statistical measures are then computed: Number of points, mean reflectivity, as well as mean, standard deviation, minimum, and maximum height. (4) The complete point cloud top-view input is obtained by stacking together the 2D tensors corresponding to each statistical measures. Fig. 3.2 shows the top-view images obtained by applying the above procedure to an example from the KITTI road training set.

3.2 GPS-IMU top-view representation

This section describes how GPS-IMU information can be transformed into a spatial representation that is suitable for data fusion with the point cloud top-views described above, and that can be efficiently exploited by a CNN for pattern recognition.

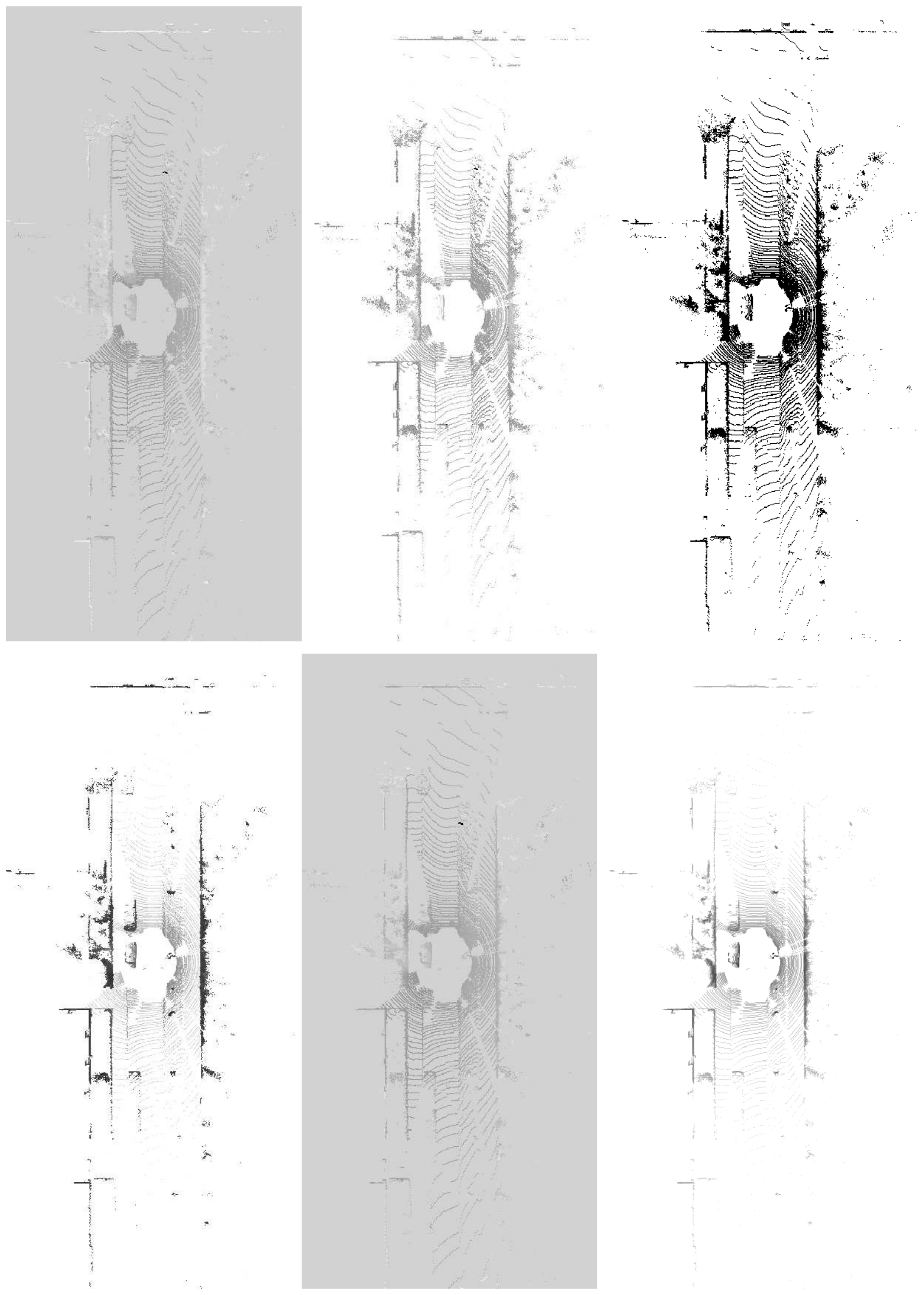

Figure 3.2: *Each image encodes a specific statistic computed inside the grid cells using the procedure described in Sect. 3.1. (Top left) Maximum height. (Top center) Mean reflectivity. (Top right) Occupancy. (Bottom left) Elevation standard deviation. (Bottom center) Minimum height. (Bottom right) Number of points.*

Consider a generic driving sequence $\mathcal{D}$ consisting of N time steps. At each time step, the GPS-IMU system acquires the following information (only the relevant quantities are reported) about the vehicle state, (1) latitude, (2) longitude, (3) altitude, (4) roll angle, (5) pitch angle, and (6) heading. The data can then be summarized as a list of positions, $\{[x_i^0, y_i^0, z_i^0]^{\mathrm{T}}\}_{i=1}^N$ and their corresponding pose matrices $\{\mathbf{M}_i\}_{i=1}^N$, with respect to a reference coordinate system denoted by the superscript 0.

By selecting k as the current time step, it is then possible to define two sets of 3D points expressed in the current coordinate system, denoted by the superscript k, the *future path* Π^+, and the *past path* Π^-. For example, the vehicle relative future position at time step $k+2$ can be obtained by applying the following transformation:

$$\begin{bmatrix} x_{k+2}^k \\ y_{k+2}^k \\ z_{k+2}^k \\ 1 \end{bmatrix} = \mathbf{M}_k^{-1}\mathbf{M}_{k+2} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} . \tag{3.2}$$

Each point in Π^+ and Π^- can then be projected onto the 2D grid specified in Sect. 3.1:

$$\begin{bmatrix} x_i^k \\ y_i^k \\ z_i^k \end{bmatrix} \xrightarrow[\text{projection}]{\text{grid}} \begin{bmatrix} u_i \\ v_i \end{bmatrix} . \tag{3.3}$$

The top left panel of Fig. 3.3 shows an example of past and future paths rendered as disks in a point cloud top-view. The next step of the procedure is to join neighboring points in order to obtain a continuous curve (see the top middle panel of Fig. 3.3). In this case, the thickness of the driving paths is set to 18 pixels or 1.8 meters, which is the vehicle's nominal width.

3.2.1 IMU augmentation

The future and past paths, so far, only contain occupancy information about whether the vehicle covered or will cover a certain grid cell (or pixel). The shape of the path already conveys useful knowledge that can, however, be augmented with additional information about the vehicle's motion, that is, its forward speed, forward acceleration, and yaw rate. For each of those

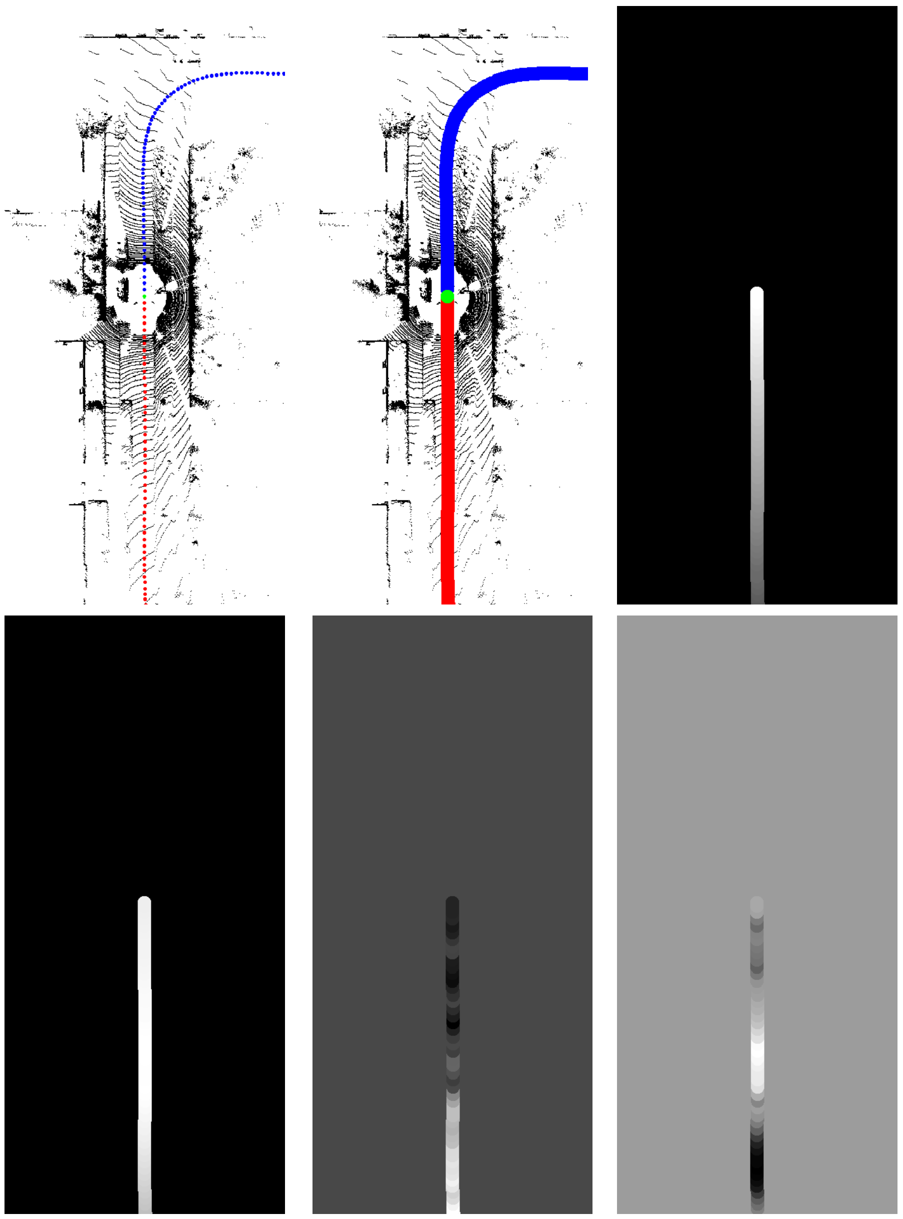

Figure 3.3: *Illustration of the results generated using the procedure described in Sect. 3.2. (Top left) Discrete past and future paths represented by red and blue disks, respectively. The current position is drawn as a green disk. (Top middle) Continuous driving paths with thickness of 1.8 meters corresponding to the vehicle's width. (Top right) Driving intention proximity encoding the approximate distance where a turning action should be taken. (Bottom) From left to right: Images representing the vehicle's forward speed, forward acceleration, and yaw rate. The grayscale intensities are proportional to the numerical values of the illustrated quantities; the intensity of the background color corresponds to a value of zero.*

quantities, a top-view image is generated by replacing the occupancy state (1 or 0) with its corresponding value at a given position. Some examples are presented in the bottom three panels of Fig. 3.3.

3.2.2 Driving intention

Lastly, given the coordinates of the current position, Google Maps is queried in order to get the driving directions for reaching a desired destination. The driving instructions consist of sequences of actions, such as *turn left* or *go straight*, and the corresponding approximate distance to the point where they should be enacted. Both the action and the approximate distance are then used for augmenting the past path thus generating two additional top-view images.

3.3 Point cloud projection and upsampling

Whereas the previous two sections discussed a bird's-eye view representation of LIDAR point clouds, here a camera perspective is considered that provides a view of the world similar to the one experienced by the driver. Camera images are, of course, directly acquired in this perspective. By contrast, LIDAR point clouds must be preprocessed using the multi-step procedure described in the following. The first step is to apply the transformation matrix $\mathbf{T}_l^c$ and rectification matrix $\mathbf{R}$ to transform a 3D point $p^l = [x^l, y^l, z^l, 1]^{\mathrm{T}}$ in the LIDAR coordinate system to a 3D point $p^c = [x^c, y^c, z^c, 1]^{\mathrm{T}}$ in the rectified camera coordinate system:

$$p^c = \mathbf{R}\,\mathbf{T}_l^c\,p^l \tag{3.4}$$

Given p^c and the camera projection matrix, $\mathbf{P}$, it is then possible to calculate its corresponding column and row position in the image, which are denoted by u and v, respectively, as:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{P} \begin{bmatrix} x^c \\ y^c \\ z^c \\ 1 \end{bmatrix} \tag{3.5}$$

where λ is a scaling factor that is also determined by solving the above system of equations. The projection is applied to every point in the point

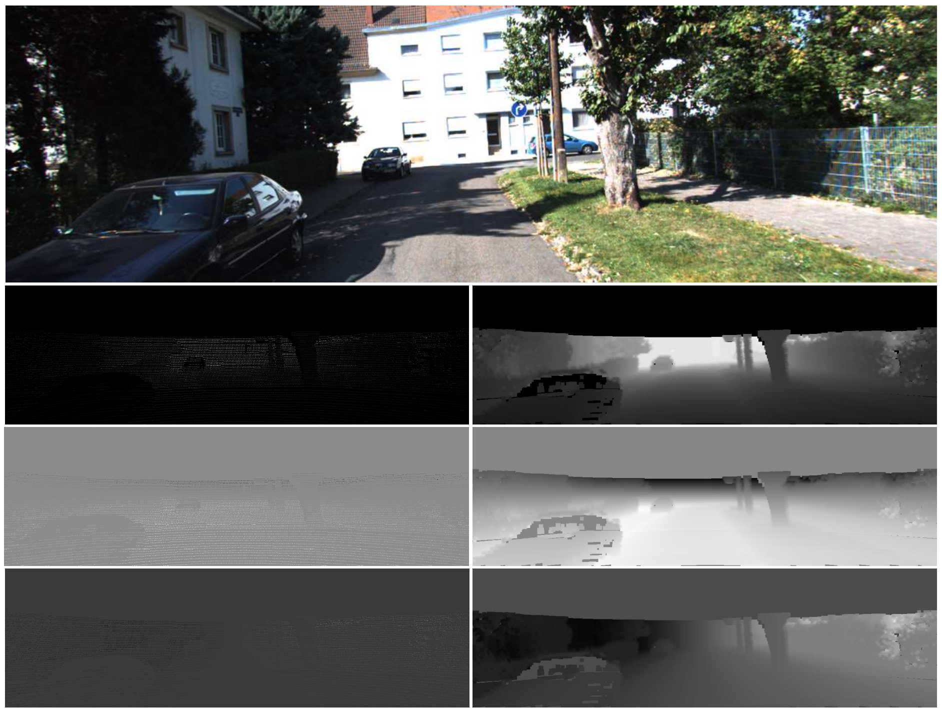

Figure 3.4: *Illustration of the projection and upsampling procedure described in Sect. 3.3. The topmost panel shows an RGB image. The left column shows the sparse LIDAR images (from top to bottom: Z, Y, X) obtained by projecting the point cloud onto the image plane. The right column shows the corresponding upsampled images. The grayscale intensities are proportional to the numerical values of the represented quantities.*

cloud, while discarding points such that $\lambda < 0$ or when $[u, v]$ falls outside the image. This procedure generates three images, denoted as X, Y, and Z where each pixel contains the x^c, y^c, and z^c coordinates of the 3D point that is projected into it. Given that the laser scanner has a limited number of channels, many pixels in the image are not associated with a valid detection, that is, there are no 3D points projected into them. In those cases, the pixel values are simply set to zero. The sparse LIDAR images are lastly upsampled using the procedure introduced by Premebida *et al.* in [22] in order to obtain a dense input which is more suitable for fusion with the corresponding RGB images. Fig. 3.4 illustrates some examples of sparse and dense LIDAR images obtained using the above procedure.

Chapter 4

Applications

4.1 Road detection

A road detector is a system that estimates the free road surface that is available for the future motion of the vehicle. Most state-of-the-art approaches carry out road detection using camera images and employ some kind of **deep neural network** (DNN) [13, 20, 18]. However, in order to achieve higher levels of driving automation, it will probably be necessary to use additional sensors besides cameras. One reason for this is that cameras are passive sensors and therefore cannot be relied upon in poor lighting conditions or at nighttime. Laser scanners (or LIDARs) on the other hand carry out sensing by emitting laser light pulses and timing their reflections. With this sensing mechanism, they can measure distances with high accuracy in any lighting condition. As shown in Fig. 4.1, the same scene captured by an RGB camera at different times during a day might show significant variability. On the other hand, those same scenes would appear rather similar to a laser scanner. With these considerations in mind, Paper I approached the problem of road detection using exclusively LIDAR point clouds, whereas Paper III investigated how to fuse RGB images and LIDAR data.

Road detection is commonly framed as a binary semantic segmentation task (see Sect. 2.3). The system is fed an image (e.g., RGB, point cloud top-view, etc.) and produces as output a confidence map that assigns to each pixel its probability of belonging to the road. Given the confidence map and the ground truth, several measures can then be computed for evaluating the system's performance [7]. This work considers three in particular, namely

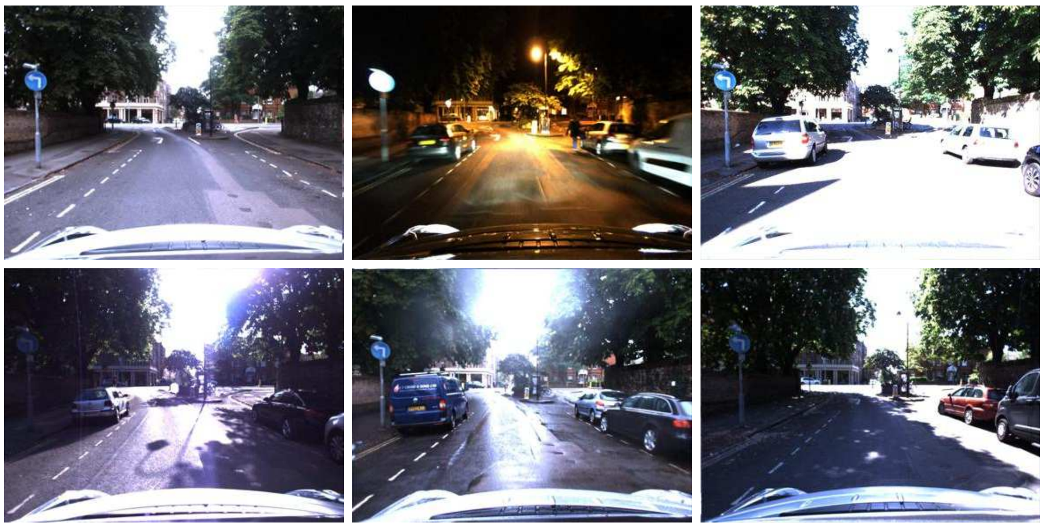

Figure 4.1: *Images of a scene captured at various times during the day [16]. As can be seen, there is significant appearance variability which makes the task of road detection using only RGB images particularly challenging or even impossible in certain conditions.*

precision (PRE), **recall** (REC), and **maximum F1-score** (MaxF). These measures are computed as follows:

$$\mathrm{PRE} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}, \tag{4.1}$$

$$\mathrm{REC} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \tag{4.2}$$

$$\mathrm{MaxF} = \max_{\tau} \; 2 \times \frac{\mathrm{PRE}(\tau) \times \mathrm{REC}(\tau)}{\mathrm{PRE}(\tau) + \mathrm{REC}(\tau)}, \tag{4.3}$$

where, TP denotes the true positives (road pixels correctly classified as road), FN represents the false negatives (road pixels classified as not-road), and FP denotes the false positives (not-road pixels classified as road). Lastly, τ represents the classification threshold: Pixels in the confidence map with probability smaller than τ are assigned to the class not-road, whereas pixel with probability larger than τ are classified as road.

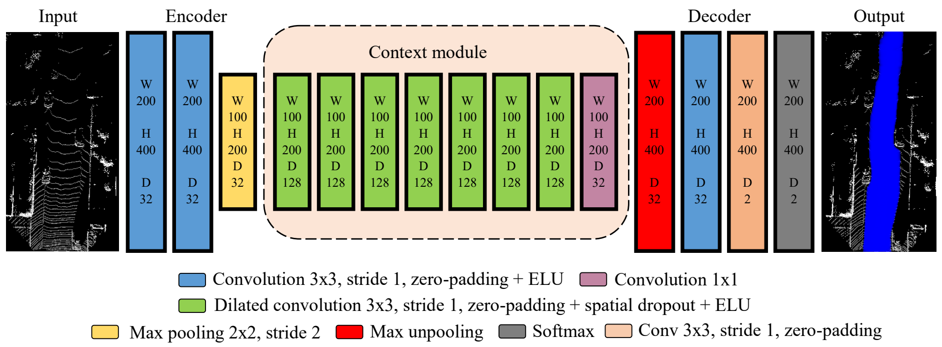
Input
Encoder
Context module
Decoder
Output
W 200 H 400 D 32
W 100 H 200 D 32
W 100 H 200 D 128
W 200 H 400 D 2
Convolution 3x3, stride 1, zero-padding + ELU
Convolution 1x1
Dilated convolution 3x3, stride 1, zero-padding + spatial dropout + ELU
Max pooling 2x2, stride 2
Max unpooling
Softmax
Conv 3x3, stride 1, zero-padding

Figure 4.2: *FCN architecture used in Paper I for road segmentation in point cloud top-views. The input consists of 6 stacked images encoding height and reflectivity information (see Sect. 3.1). The output is a road confidence map. W represents the width, H denotes the height, and D is the number of feature maps.*

4.1.1 LIDAR-only segmentation

Paper I considered only LIDAR point clouds, which were then transformed into top-view images using the procedure described in Sect. 3.1. A top-view perspective was chosen because both path planning and vehicle control are performed in this 2D world. Algorithms competing in the KITTI road benchmark are also evaluated in this perspective [8]. Additionally, classification is simplified in comparison with the camera perspective where, for example, objects farther away appear smaller; in that case, an FCN would have to learn to recognize patterns at multiple scales thus requiring more training data and larger capacity. Fig. 4.2 shows the FCN architecture that was used for road segmentation. As can be seen, the general architecture resembles the encoder-decoder network that was described in Sect. 2.2.3. In this case, however, the encoder and decoder are shallower and include only one max pooling and one max unpooling layer, respectively. By keeping the feature maps at high resolution, it is possible to achieve higher segmentation accuracy. The FCN also includes several **dilated convolutional layers** which are collectively denoted as a **context module**. Dilated convolutions are similar to standard convolutions with the difference that one has to specify a parameter called **dilation factor** that determines how many units to skip when carrying out the convolution operation. By progressively increasing the dilation factor, it is possible to grow the receptive field of the network with fewer layers than would be required using standard convolutions.

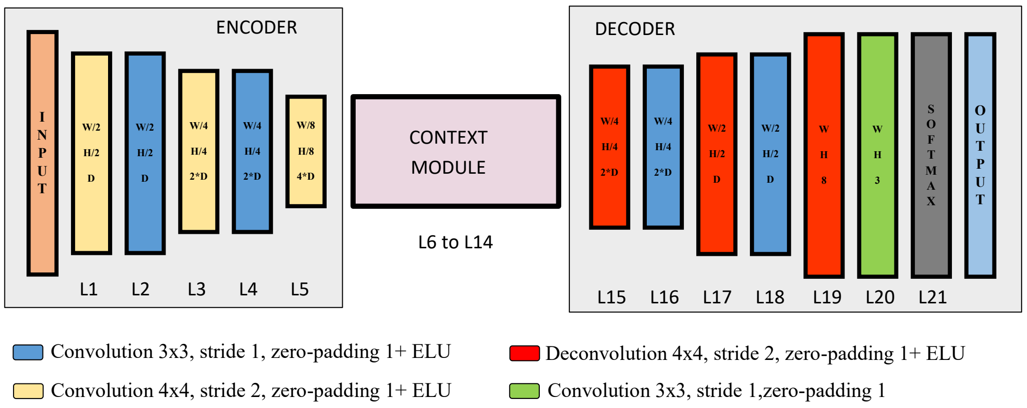
ENCODER
INPUT
W/2 H/2 D
W/2 H/2 D
W/4 H/4 2*D
W/4 H/4 2*D
W/8 H/8 4*D
L1 L2 L3 L4 L5
CONTEXT MODULE
L6 to L14
DECODER
W/4 H/4 2*D
W/4 H/4 2*D
W/2 H/2 D
W/2 H/2 D
W H 8
W H 3
SOFTMAX
OUTPUT
L15 L16 L17 L18 L19 L20 L21
Convolution 3x3, stride 1, zero-padding 1+ ELU
Convolution 4x4, stride 2, zero-padding 1+ ELU
Deconvolution 4x4, stride 2, zero-padding 1+ ELU
Convolution 3x3, stride 1,zero-padding 1

Figure 4.3: *Base FCN architecture used in Paper III. The network consists of 20 layers and it is entirely convolutional. Both downsampling and upsampling are implemented using convolutional layers instead of max pooling and max unpooling layers as was the case in Paper I and II. Details about the context module can be found in Paper III.*

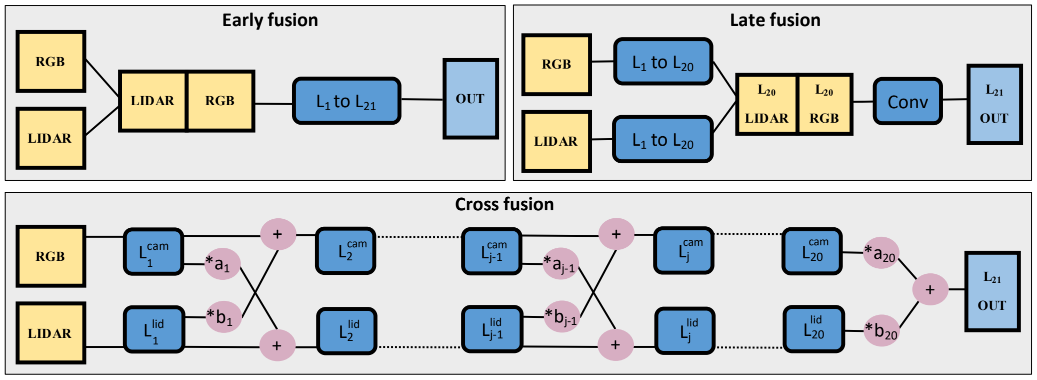

Figure 4.4: *Fusion strategies investigated in Paper III for carrying our LIDAR-camera data fusion. In early fusion, the input tensors are concatenated and then processed together. In late fusion, two parallel branches each process one input tensor. Fusion happens deeper in the network at the decision level, just before the output layer. Cross fusion is also carried out by two parallel branches, however, in this case, trainable connections allow data integration at multiple depth levels.*

4.1.2 LIDAR-camera segmentation

Paper III also considered the road segmentation task using FCNs. In this case, however, LIDAR and camera data were fused together to achieve higher accuracy and robustness, in particular when dealing with challenging lighting conditions. To establish a direct correspondence between camera and LIDAR data, in this paper the point clouds were projected onto the camera plane and then upsampled using the procedure presented in Sect. 3.3. The base FCN architecture is shown in Fig. 4.3. As can be noticed, downsampling was performed by using 4×4 convolutions with stride 2 instead of max pooling layers. Given that the input images had a larger size compared to those used in Paper I, the encoder included three downsampling layers. Similarly, the decoder contained three upsampling layers.

Two common approaches for fusing multimodal data with CNNs are **early fusion** and **late fusion**. As shown in Fig. 4.4, early fusion simply involves concatenating the input tensors of each modality to create a multimodal input tensor. In the specific case of LIDAR-camera fusion considered here, the input tensors are the RGB camera image and the ZYX dense image which, once stacked, become a tensor with 6 channels denoted as RGBZYX. This tensor is then processed with a single branch FCN.

Late fusion instead is carried out with a two-branch FCN (see Fig. 4.4), where each branch processes only one of the two input tensors. The actual fusion happens right before the output layer by feeding the processed single modality tensors to one last convolutional layer after concatenating them.

A contribution of Paper III was the introduction of a novel fusion strategy named **cross fusion** which is illustrated in the bottom panel of Fig. 4.4. As can be seen, the cross fusion network also consists of a two-branch FCN. In this case, however, the two branches are connected by trainable cross connections, denoted as a_j and b_j, $j \in \{1, \dots, N\}$, where N is the number of layers. The underlying idea behind this strategy is to allow the FCN to learn to integrate multimodal information at any level of abstraction.

KITTI road benchmark results

The road detector developed in Papers I and III were evaluated on the KITTI road benchmark [8] to assess their performance against state-of-the-art algorithms. The results are reported in Table 4.1. LoDNN stands for *LIDAR-only deep neural network* and it is the system developed in Paper I. This approach

Table 4.1: *KITTI road benchmark results (in %) on the urban road category. LidCamNet is the cross fusion FCN developed in Paper III. LoDNN is the LIDAR-only FCN introduced in Paper I. Only results of published methods are reported.*

Rank	Method	MaxF	PRE	REC	Time (s)
1	DFFA [15]	96.35	96.02	96.69	0.4
2	**LidCamNet**	96.03	96.23	95.83	0.15
3	RBNet [4]	94.97	94.94	95.01	0.18
4	StixelNet II [9]	94.88	92.97	96.87	1.2
6	**LoDNN**	94.07	92.81	95.37	0.018
14	LidarHisto [3]	90.67	93.06	88.41	0.1

obtained a MaxF score of 94.07% and outperformed by 3.4 percentage points the second best LIDAR-only system, LidarHisto. Furthemore, it is currently the algorithm with fastest inference. The cross fusion FCN introduced in Paper III is called LidCamNet which stands for *LIDAR-Camera fusion network*. This approach is currently the second best algorithm on the benchmark.

LidCamNet performed significantly better than LoDNN, the system developed in Paper I. There might be several factors behind this gap: First and foremost, LidCamNet also used camera images, which in good lighting conditions provide rich discriminative clues for road detection. Another factor that favored LidCamNet was that, in Paper III, the LIDAR images were upsampled, whereas the top-view images used in Paper I were sparse. One of the arguments put forward in Paper I was that road detection is more complex in the camera perspective because an FCN has to learn multiple scales. Given that in Paper III a LIDAR-only FCN (called ZYX-FCN) was also included in the experiments, it was possible to make some preliminary comparisons between LIDAR-based road detectors working in different perspectives. ZYX-FCN could not be evaluated on the KITTI road benchmark (multiple submissions are discouraged) so a direct comparison of ZYX-FCN and LoDNN on the test set was not possible. However, the two FCNs were compared on the validation set used in Paper III: LoDNN achieved a MaxF score of 94.62% whereas ZYX-FCN obtained a score of 94.96%. The gap between the two systems is not very large but it is noteworthy that ZYX-FCN performed better. In light of these results, it is clear that further experiments are warranted to determine whether a top-view perspective is indeed advantageous for carrying out road detection.

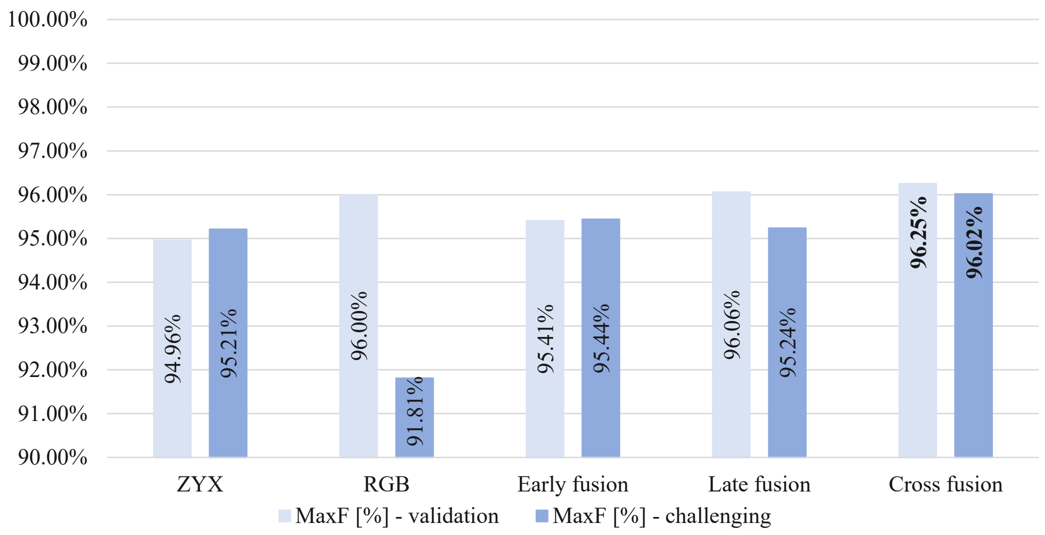

Figure 4.5: *Comparison of MaxF scores on the validation set and the challenging set.*

Challenging driving scenes

An additional contribution of Paper III was to extend the KITTI road data set with additional scenes that were visually challenging. The purpose was to highlight the benefits of fusing LIDAR and camera data for achieving high accuracy in a wider spectrum of external conditions. The fusion FCNs previously described and two single modality FCNs were evaluated on a validation set selected from the KITTI road data set (fair weather and good lighting conditions) and on the challenging set mentioned above. The results of this experiment are reported in Fig. 4.5. As can be seen, the RGB-FCN performance was rather good on the validation set (96.00% MaxF) but, as expected, it decreased significantly on the challenging set (91.81% MaxF). All the FCNs that had access to LIDAR data achieved scores above 95% on the challenging set, thus confirming the usefulness of this sensor for robust road detection. The cross fusion FCN was again the most successful fusion strategy achieving a MaxF score above 96% in both the validation set and the challenging set.

4.2 Driving path generation

The previous section dealt with the problem of road detection using camera and LIDAR sensors. Road detection, however, is just one among many tasks that must be carried out within the perception component of autonomous vehicles. The final goal is to detect and recognize all the driving factors in the vehicle's surroundings so that a world model can be generated and then used for carrying out higher level functions such as path planning, decision making, and control.

An alternative paradigm, known as *behavior reflex*, instead proposes to learn all the driving functions with one single system. A famous example of this approach is ALVINN [21] developed by Pomerlau in 1989; in that work, a fully connected feedforward neural network with one hidden layer was trained to map low-resolution camera and range images to desired heading angles. An advantage of this type of approach is that the training data, which usually consists of sequences of camera images together with the corresponding actions carried out by the driver (e.g., steering angle and acceleration), can be obtained automatically without the need of time-consuming hand-labeling. On the negative side, given that the system carries out all the driving functions, from perception to control, it is much more difficult to gain insight into its decision-making process and to predict how it will behave when presented with novel situations.

Paper II described an alternative approach where an FCN was trained to predict human-like driving paths for the future motion of a vehicle. Training examples were obtained automatically by processing the data logged during real driving sequences (e.g., LIDAR point clouds, GPS positions, etc.). In this way, a large data set was obtained without much effort. Also in this case, LIDAR top-view images were used as input for the FCN (see Sect. 3.1), however a larger region of interest was considered which covered the range $[-30, 30]$ meters in both the x and y directions. Additional input channels encoding information about the vehicle's past motion and driving directions were generated by using the procedure described in Sect. 3.2. The FCN's architecture was similar to the one used in Paper I, one of the main differences being that there were two max pooling layers instead of one in order to reduce the computational requirements because of the larger size of the input images.

Several FCNs were trained using different combinations of input channels for evaluating the benefits provided by individual sensor or information modalities. The FCN that only used LIDAR data obtained a score of 83.44%.

Adding IMU provided a gain of 1.7 percentage points, whereas considering driving directions resulted in a gain of 2.3 percentage points. The FCN that used all the available data, that is, LIDAR point clouds, IMU information, and driving directions, obtained the best MaxF score of 88.13%. These results showed that each modality contributed to improve performance and that the adopted data representation was appropriate for integrating multimodal information.

Chapter 5

Conclusion and future work

5.1 Conclusion

This thesis has explored several applications of deep learning methods in the context of driving automation. It has been shown that CNNs, which are commonly used for processing camera images, can also be used effectively for working exclusively with LIDAR data (Paper I) or in combination with other sensors or information modalities (Papers II and III). The emphasis on the LIDAR sensor was motivated by its capacity to provide highly accurate range measurements in any lighting conditions; this alternative view of the environment can then be leveraged for making driving automation possible in a broader spectrum of external conditions.

It has been shown that sparse bird's-eye view images can directly be used as input to an FCN for carrying out road segmentation (Paper I) with high accuracy and fast detection. The LIDAR-only deep neural network (LoDNN), achieved state-of-the-art performance with a MaxF score of 94.07% on the KITTI road benchmark, outperforming by a wide margin the second best LIDAR-based system.

By working in the same top-view perspective, a novel end-to-end learning system has been developed for generating driving paths (Paper II). Whereas a behavior reflex approach simply predicts the next driving action, a driving path specifies the vehicle's future positions over a longer look-ahead horizon (30 meters, in this work). This makes the output of the proposed approach more interpretable and informative for further planning and decision-making. Additionally, the ground truth annotations for training the FCN are obtained in an automatic fashion thus allowing the generation of large training sets with no additional human effort and the possibility of continuous learning, that is, training the system while it is running in order for it to adapt to entirely novel situations. In order to exploit the visual pattern recognition capability of CNNs, information about the vehicle's past motion and future destination were transformed into a spatial representation that allowed a direct integration with the point cloud top-views (Paper II).

Multimodal sensor fusion has also been investigated for the task of road detection. It has been shown that by combining camera images and LIDAR data, an FCN can carry out road segmentation in a wider range of external conditions than possible using only RGB images (Paper III). Besides evaluating two established fusion FCNs, early and late fusion, a novel multimodal FCN has been developed that can learn at what abstaction level and to what extent LIDAR and camera information should be fused. This approach was evaluated on the KITTI road benchmark for which it achieved state-of-the-art performance. In fact, it outperformed by almost two percentage points the system introduced in Paper I, a result that highlights the importance of developing multimodal approaches in the quest towards full driving automation.

5.2 Future work

The approaches described in this thesis were all memory-less, that is, they processed each input independently of the previous ones. The integration of time in the system, either in the form of aggregated inputs (e.g., combining multiple LIDAR scans within one image) or by using networks with an internal memory (such as recurrent neural networks) would likely improve the overall performance and robustness for all the considered tasks. For example, in its current formulation, the driving path generation FCN described in Paper II cannot know which objects are stationary and which ones are not, something that limits its applicability mostly to traffic-free scenarios.

Another important issue is the annotation of data. Supervised learning approaches, such as the ones presented here, need annotated training examples which in general are very expensive to obtain in terms of time, money, and human effort. In Paper II, the problem of creating annotations was solved by using the GPS to recover the vehicle's future driving path. A relevant topic for future work would be to devise similar strategies for automatic or semi-automatic data labeling.

As was shown in Papers II and III, fusing multiple sensors and information modalities is generally beneficial for achieving better performance and robustness. In future work, it will be of high interest to investigate how additional sensors (such as radars, stereo cameras, etc.) and information sources (e.g., high-definition maps) can be integrated within the proposed systems. A related topic will be the development of additional data representations that are suitable for multimodal data fusion.

Chapter 6

Summary of included papers

6.1 Paper I

This work was concerned with road detection using a LIDAR as the main sensor. The problem was cast as a semantic segmentation task and it was approached using a fully convolutional neural network (FCN). The FCN was designed to have a large receptive field and used high-resolution feature maps in order to achieve higher accuracy. Top-view images encoding several basic statistics were generated to summarize the information content of unstructured point clouds into a format that was suitable for processing with the FCN. The proposed system carried out real-time inference at about 55 Hz using a modern GPU and achieved state-of-the-art performance on the KITTI road benchmark reaching a MaxF score of 94.07% and outperforming by over 3.4 percentage points the second best LIDAR-only system.

6.2 Paper II

In this paper, a deep learning approach was developed to carry out driving path generation in LIDAR top-view images. This was accomplished by fusing LIDAR point clouds with GPS-IMU information and driving directions using an FCN. The system was trained using as ground truth driving paths followed by human drivers. An advantage of this approach was that the annotations were obtained automatically and therefore a large data set for supervised learning could be collected with limited effort. Several combinations of inputs were considered to determine the effect of individual information modalities. The main result was that the FCN trained with all the available information (i.e., LIDAR, GPS-IMU, and driving directions) achieved the highest accuracy thus confirming that the chosen architecture and data representation were suitable for carrying out information fusion and that each modality contributed to the overall system performance.

6.3 Paper III

As Paper I, this work was also concerned with the task of road detection. However, in this case, the problem was solved by integrating RGB images and LIDAR point clouds that were projected onto the image plane. One of the main contributions of this work was the introduction of a novel fusion FCN architecture, called *cross fusion*, that allowed multimodal information fusion at any depth level in the network by using trainable cross connections. The cross fusion FCN outperformed two other established fusion approaches, early fusion and late fusion, as well as the single modality (RGB or LIDAR) FCNs. The system was evaluated on the KITTI road benchmark and ranked second with a MaxF score of 96.03%.

Bibliography

[1] V. Badrinarayanan, A. Handa, and R. Cipolla, *Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling*, arXiv preprint arXiv:1505.07293, (2015).

[2] M. Blanco, J. Atwood, S. Russell, T. Trimble, J. McClafferty, and M. Perez, *Automated vehicle crash rate comparison using naturalistic data*, tech. rep., Virginia Tech Transportation Institute, 2016.

[3] L. Chen, J. Yang, and H. Kong, *Lidar-histogram for fast road and obstacle detection*, in 2017 IEEE International Conference on Robotics and Automation (ICRA), May 2017, pp. 1343–1348.

[4] Z. Chen and Z. Chen, *Rbnet: A deep neural network for unified road and road boundary detection*, in International Conference on Neural Information Processing, Springer, 2017, pp. 677–687.

[5] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, *Fast and accurate deep network learning by exponential linear units (elus)*, CoRR, abs/1511.07289 (2015).

[6] S. Dahdah and K. McMahon, *The true cost of road crashes: valuing life and the cost of a serious injury*, International Road Assessment Programme, World Bank Global Road Safety Facility, (2008).

[7] T. Fawcett, *An introduction to roc analysis*, Pattern Recognition Letters, 27 (2006), pp. 861 – 874. ROC Analysis in Pattern Recognition.

[8] J. Fritsch, T. Kuhnl, and A. Geiger, *A new performance measure and evaluation benchmark for road detection algorithms*, in Intelligent Transportation Systems-(ITSC), 2013 16th International IEEE Conference on, IEEE, 2013, pp. 1693–1700.

[9] N. Garnett, S. Silberstein, S. Oron, E. Fetaya, U. Verner, A. Ayash, V. Goldner, R. Cohen, K. Horn, and D. Levi, *Real-time category-based and general obstacle detection for autonomous driving*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 198–205.

[10] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, *Vision meets robotics: The kitti dataset*, The International Journal of Robotics Research, 32 (2013), pp. 1231–1237.

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.

[12] D. H. Hubel and T. N. Wiesel, *Receptive fields, binocular interaction and functional architecture in the cat's visual cortex*, The Journal of physiology, 160 (1962), pp. 106–154.

[13] A. Laddha, M. K. Kocamaz, L. E. Navarro-Serment, and M. Hebert, *Map-supervised road detection*, in Intelligent Vehicles Symposium (IV), 2016 IEEE, IEEE, 2016, pp. 118–123.

[14] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature, 521 (2015), pp. 436–444.

[15] X. Liu, Z. Deng, and G. Yang, *Drivable road detection based on dilated fpn with feature aggregation*, in 29th International Conference on Tools with Artificial Intelligence (ICTAI), IEEE, 2017.

[16] W. Maddern, G. Pascoe, C. Linegar, and P. Newman, *1 Year, 1000km: The Oxford RobotCar Dataset*, The International Journal of Robotics Research (IJRR), 36 (2017), pp. 3–15.

[17] W. S. McCulloch and W. Pitts, *A logical calculus of the ideas immanent in nervous activity*, The bulletin of mathematical biophysics, 5 (1943), pp. 115–133.

[18] R. Mohan, *Deep deconvolutional networks for scene parsing*, arXiv preprint arXiv:1411.4101, (2014).

[19] V. Nair and G. E. Hinton, *Rectified linear units improve restricted boltzmann machines*, in Proceedings of the 27th International Conference on Machine Learning (ICML), 2010, pp. 807–814.

[20] G. L. Oliveira, W. Burgard, and T. Brox, *Efficient deep models for monocular road segmentation*, in 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct 2016, pp. 4885–4891.

[21] D. A. Pomerleau, *Alvinn: An autonomous land vehicle in a neural network*, in Advances in Neural Information Processing Systems 1, D. S. Touretzky, ed., Morgan-Kaufmann, 1989, pp. 305–313.

[22] C. Premebida, J. Carreira, J. Batista, and U. Nunes, *Pedestrian detection combining rgb and dense lidar data*, in 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Sept 2014, pp. 4112–4117.

[23] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton*, Technical Report 85-460-1, (1957).

[24] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors*, Nature, 323 (1986), pp. 533–538.

[25] SAE On-Road Automated Vehicle Standards Committee and others, *Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems*, SAE Standard J3016, (2014), pp. 01–16.

[26] J. R. Treat, N. Tumbas, S. McDonald, D. Shinar, R. D. Hume, R. Mayer, R. Stansifer, and N. Castellan, *Tri-level study of the causes of traffic accidents: final report*, (1979).

[27] World Health Organization, *Global status report on road safety 2015*, World Health Organization, 2015.