AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

FIELD OF SCIENCE: ENGINEERING AND TECHNOLOGY

SCIENTIFIC DISCIPLINE: AUTOMATION, ELECTRONICS, ELECTRICAL ENGINEERING AND SPACE TECHNOLOGIES

DOCTORAL DISSERTATION

Depth Completion for FMCW Radars

Author:	*Mariusz Karol Nowak*
Supervisor:	*Dr hab. inż. Paweł Skruch, prof. AGH*
Completed in:	*AGH University of Science and Technology* *Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering* *Department of Automatic Control and Robotics*

Kraków, 2023

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

DZIEDZINA NAUK INŻYNIERYJNO-TECHNICZNYCH

DYSCYPLINA AUTOMATYKA, ELEKTRONIKA, ELEKTROTECHNIKA I TECHNOLOGIE KOSMICZNE

ROZPRAWA DOKTORSKA

Zagęszczanie map głębi dla radarów FMCW

Autor:	*Mariusz Karol Nowak*
Promotor pracy:	*Dr hab. inż. Paweł Skruch, prof. AGH*
Praca wykonana:	*Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie* *Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej* *Katedra Automatyki i Robotyki*

Kraków, 2023

I would like to express my deepest gratitude to my PhD supervisor dr hab. inż. Paweł Skruch, prof. AGH and my industrial supervisor dr inż. Mateusz Komorkiewicz for their guidance during the work on my PhD, helpful discussions and review of this dissertation.

I'm grateful to my colleagues from Aptiv Cracow for their work on equipping the data collection vehicle used in this work and data collection efforts.

Finally, I would like to thank my parents Marian A. Nowak and Krystyna Nowak for their support while I worked on my PhD.

Preamble

This research is a result of an industrial PhD program, funded by the Polish Ministry of Science and Higher Education (MNiSW), and carried out in cooperation with Aptiv Services Poland S.A., Technical Center Kraków and AGH University of Science and Technology, Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering.

Abstract

A model of the environment which includes distance to the nearest obstacle is necessary for a system tasked with autonomous movement. Therefore, it is reasonable to say that every autonomously-navigating system must be able to estimate depth (i.e. the distance to the nearest object). However, there is a trade-off between the size, complexity and cost of a depth sensor on one hand, and the density and accuracy of its measurements on the other hand.

The goal of this dissertation is to examine the feasibility of depth completion (i.e. producing a dense depth-map from a sparse depth input) for extremely sparse depth measurements. The bulk of this work deals with depth completion on automotive Frequency Modulated Continuous Wave (FMCW) radars. Additionally, I present a novel way to perform depth completion on lidar point cloud, which I developed for the purpose of creating the training dataset for the radar depth completion task, an algorithm utilizing zero-centered, additive weight perturbations during neural network training and simple simulations showing the capability of a neural network to perform angle finding in the presence of 2 targets.

The first two chapters of the dissertation describe the problem formulation, motivation and automotive sensors, with emphasis put on FMCW radars. The next chapters describe my original contributions.

The first contribution (described in Chapter 3) is an algorithm utilizing zero-centered, additive weight perturbations during neural network training. I show why it helps to increase the amount of information the neural network can learn for a given size and demonstrate its usefulness for some commonly used neural network architectures. Weight perturbations were utilized in the WeaveNet training and in the training of the radar angle finding model on simulated data.

The second contribution (described in Chapter 4) is the definition of WeaveNet - a neural network whose architecture is designed to perform well in the task of lidar depth completion on variable input sparsity depth measurements (name inspired by the convolutional kernels pattern, which looks like a woven fabric). It was trained and tested utilizing the data from the KITTI Depth Completion challenge.

WeaveNet was instrumental in creating a dense depth dataset, that was later used in my radar depth completion solution.

The third contribution (described in Chapter 5) consists of a set of idealized simulations, showing that neural networks are capable of finding angles to two targets when radar antennae receive a superposition of reflected waves (a necessary step for a radar depth completion network).

The fourth contribution (described in Chapter 6) is the creation of a dataset used to train and validate algorithms for radar depth completion. It consists of over 1,000,000 Radar Data Cubes (RDCs) from a forward-facing radar, together with corresponding camera images and dense depth maps (produced using WeaveNet).

The fifth contribution (described in Chapter 7) is the design of a neural network architecture capable of transforming the low-level RDC input into abstract channels in the azimuth-elevation plane. Consequently, it was possible to train this neural network to predict dense depth maps using RDC as input. They were trained and tested on the dataset that was created for the purpose of the work on this PhD dissertation. The output of the radar-only depth completion networks is visually reasonable and much better than the linear interpolation of the point cloud obtained using standard radar processing algorithms. To the best of my knowledge, this is the first solution producing a dense depth map on the basis of automotive FMCW radar RDC.

The final contribution (also described in Chapter 7) is the design of the neural network architecture capable of fusing the RDC-derived data from the radar and the data derived from RGB camera images for the purpose of radar depth completion (also trained and tested on the same dataset). I have shown that the networks utilizing RDC in addition to visual data obtain results between 3% and 21.5% better than analogous networks trained to use visual data only (during different data collection drives).

Keywords: radar, FMCW, depth completion, weight perturbation

Streszczenie

Model otoczenia, który obejmuje odległość do najbliższej przeszkody, jest niezbędny dla systemu jazdy autonomicznej. Dlatego też można uznać, że każdy system nawigujący autonomicznie musi być w stanie oszacować głębokość (czyli odległość do najbliższego obiektu). Istnieje jednak konflikt pomiędzy rozmiarem, złożonością i kosztem czujnika głębokości, z jednej strony, a gęstością i dokładnością pomiarów, z drugiej strony.

Celem tej rozprawy jest zbadanie wykonalności zagęszczania map głębi (czyli tworzenia gęstej mapy głębokości na podstawie rzadkich pomiarów głębokości) dla wyjątkowo rzadkich pomiarów głębokości. Większość tej pracy dotyczy uzupełniania głębokości w radarach samochodowych typu *Frequency Modulated Continuous Wave* (FMCW). Dodatkowo przedstawiam nowy sposób zagęszczania map głębi na chmurze punktów lidaru, który opracowałem w celu stworzenia zestawu danych szkoleniowych do zadania uzupełniania głębokości radarowej, algorytm dodający szum do wag sieci neuronowej podczas treningu i prostą symulację pokazującą, że sieć neuronowa jest w stanie estymować kąt do celu w obecności dwóch celów.

Pierwsze dwa rozdziały rozprawy opisują rozwiązywany problem, motywację oraz sensory motoryzacyjne, ze szczególnym uwzględnieniem radarów FMCW. Kolejne rozdziały opisują osiągnięcia doktoratu.

Pierwszym osiągnięciem (opisanym w Rozdziale 3) jest algorytm wykorzystujący centrowane wokół zera, addytywne perturbacje wag w trakcie szkolenia sieci neuronowej. Pokazuję, dlaczego pomaga to zwiększyć ilość informacji, której sieć neuronowa może nauczyć się przy danym rozmiarze sieci. Dodatkowo demonstruję przydatność metody dla niektórych powszechnie stosowanych architektur sieci neuronowych. Perturbacje wag były wykorzystane w szkoleniu sieci WeaveNet oraz przy treningu sieci przeprowadzających *angle finding* na symulowanych danych.

Drugim osiągnięciem (opisanym w Rozdziale 4) tej pracy jest zdefiniowanie Sieci WeaveNet - sieci neuronowej, której architektura została zaprojektowana w celu skutecznego zagęszczania map głębi dla danych lidarowych, o zmiennej gęstości wejściowych pomiarów głębokości. Nazwa sieci

została zainspirowana wzorem jąder konwolucyjnych, który wygląda jak tkanina. Sieć WeaveNet została przeszkolona i przetestowana przy użyciu danych z konkursu KITTI Depth Completion. Sieć WeaveNet odegrała kluczową rolę w tworzeniu gęstego zestawu danych głębokości, który później został wykorzystany w moim rozwiązaniu uzupełniania głębokości radarowej.

Trzecim osiągnięciem (opisanym w Rozdziale 5) jest zestaw wyidealizowanych symulacji, które pokazują, że sieci neuronowe są zdolne do znajdowania kątów do dwóch celów, gdy anteny radarowe odbierają superpozycję fal odbitych (niezbędny krok dla sieci zagęszczania map głębi na danych radarowych).

Czwartym osiągnięciem (opisanym w Rozdziale 6) jest stworzenie zestawu danych używanego do szkolenia i walidacji algorytmów zagęszczania map głębi na danych radarowych. Zestaw składa się z ponad 1 000 000 *Radar Data Cubes* (RDC) z radaru skierowanego w przód, wraz z odpowiadającymi im obrazami z kamery i gęstymi mapami głębokości (tworzonymi przy użyciu sieci WeaveNet).

Piątym osiągnięciem (opisanym w Rozdziale 7) jest zaprojektowanie architektury sieci neuronowej zdolnej do przekształcania niskopoziomowego wejścia RDC w abstrakcyjne kanały na płaszczyźnie azymut-elewacja. W rezultacie możliwe stało się przeszkolenie tej sieci neuronowej do przewidywania gęstych map głębokości przy użyciu RDC jako wejścia. Sieci te były szkolone i testowane na zestawie danych stworzonym na potrzeby pracy nad tym doktoratem. Wyjście sieci zagęszczania map głębi używającej wyłacznie danych radarowych jest wizualnie sensowne i znacznie lepsze niż interpolacja liniowa chmury punktów uzyskanej przy użyciu standardowych algorytmów przetwarzania radarowego. O ile mi wiadomo, jest to pierwsze rozwiązanie generujące gęstą mapę głębokości na podstawie RDC z motoryzacyjnego radaru FMCW.

Ostatnim osiągnięciem (również opisanym w Rozdziale 7) jest projekt architektury sieci neuronowej zdolnej do łączenia danych pochodzących z radaru (RDC) i obrazów kamery RGB w celu zagęszczania map głębi (również przeszkolonej i przetestowanej na tym samym zestawie danych). Pokazałem, że sieci wykorzystujące RDC w połączeniu z danymi wizualnymi uzyskują wyniki od 3% do 21,5% lepsze niż analogiczne sieci szkolone tylko do korzystania z danych wizualnych (na różnych podzbiorach zbioru danych).

Słowa kluczowe: radar, FMCW, zagęszczanie map głębi, perturbacja wag

Contents

List of Abbreviations

The following table describes the meaning of various abbreviations and acronyms used throughout the thesis, in alphabetical order. The page on which each one is defined or first used is also given. Common abbreviations or acronyms used once are not on this list.

Abbreviation	**Meaning**	**Page**
ADAS	Advanced Driver Assistance Systems	124
CA CFAR	Cell Averaging Constant False Alarm Rate	37
CFAR	Constant False Alarm Rate	37
CNN	Convolutional Neural Networks	71
CPU	Central Processing Unit	45
CUT	Cell Under Test	37
FMCW	Frequency Modulated Continuous Wave	24
FFT	Fast Fourier Transform	36
FOV	Field of View	40
GPS	Global Positioning System	21
GPU	Graphics Processing Unit	46
GRU	Gated Recurrent Unit	122
GT	Ground Truth	27
IFOV	Instantaneous Field of View	40
IMU	Inertial Measurement Unit	21
KPI	Key Performance Indicator	95
MAE	Mean Absolute Error	74
MLE	Maximum Likelihood Estimator	39

Abbreviation	Meaning	Page
OS CFAR	Order Statistic Constant False Alarm Rate	38
RDC	Radar Data Cube	24
RGB	Red Green Blue	67
SGD	Stochastic Gradient Descent	47
SOCA CFAR	Smallest-Of Cell-Averaging Constant False Alarm Rate	38
TOF	Time-of-Flight	31

Mathematical Notation

The following tables describe the mathematical notation used throughout the thesis (latin and greek letters separately).

Symbol	Meaning
$\odot$	Element-wise product operation
B	Bandwidth
c	Speed of light
d	Distance projected on the antenna axis
D_{max}	Maximal relative lidar input density (when training WeaveNet)
D_{min}	Minimal relative lidar input density (when training WeaveNet)
f	Frequency
f_c	Carrier frequency
f_T	Instantaneous chirp frequency
Δf	Frequency difference
f_{prune}	Prunning function used in weight prunning
f_{score}	Scoring function used in weight prunning
m	Vector containing mask of 0s and 1s of the size of neural network weights
p	Probability output at a neural network head
r	Distance
t	Time
T	Period
Δv	Relative radial velocity

Symbol	Meaning
α	Angle to target
β	Focal loss parameter
γ	Focal loss parameter
ϵ	Distance between antennae
θ	Vector containing neural network weights
λ	Wavelength
ϕ	Phase offset
ψ	Wavefunction
ω	Angular frequency

1 Introduction

1.1 Needs of Autonomous Driving Systems

1.1.1 Decision Pipeline

The task of autonomous driving can be broadly divided into two sub-tasks, the first one is perception of the environment and the other one is planning and performing actions in the environment Such division is implicit for example in MobilEye's Responsibility Sensitive Safety framework (Shalev-Shwartz et al. [2018]). After planning is performed, proper commands can be sent to the actuators, which execute them and cause the car to move according to the desired plan. A simplified, schematic representation of the hierarchy of autonomous driving execution is presented in Figure 1.1.

The autonomous driving algorithm stream starts with the sensors. There are 3 main sensors used for external perception in automotive: camera, radar and lidar (described closer in Chapter 2). Each of the sensors has different capabilities that suit different perception tasks best. Likewise, each of the sensor types has different optimal locations on the car, typical sensor locations on an autonomous car are presented in Figure 1.2. Apart from external perception, information about the localization of the car itself is also needed. Typically it is provided by the Global Positioning System (GPS) unit in cooperation with the inertial measurement unit (IMU) .

The next step in the autonomous driving algorithm stream consists of the perception algorithms. Planning is best performed when it operates on an abstract representation of the environment. Perception algorithms aim to transform the sensors' raw output into an abstract representation useful for the planning module. External perception algorithms typically output a list of objects (such as cars or pedestrians around the car) or an occupancy grid - a map-like representation of the surroundings where each small cell (e.g. a 0.5 m x 0.5 m) is classified as belonging to a particular class (e.g. driveable space, static obstacle or moving obstacle). Localization algorithms use the GPS and IMU output to locate the vehicle on a global map, containing information about the road plan.

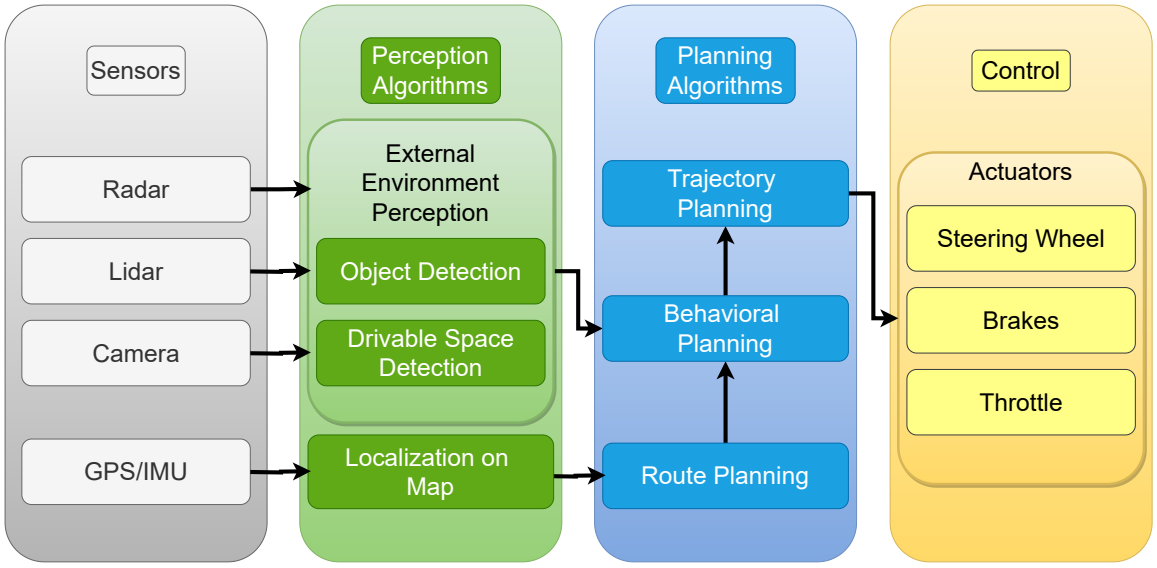


Figure 1.1. Simplified, schematic representation of the hierarchy of autonomous driving execution.

Planning algorithms use the perception output and produce commands passed to the control unit. A simple example of the planning pipeline can proceed according to the Algorithm 1.

Algorithm 1 Planning Algorithm Pipeline

1: Use map and localization data to plan the route to the destination.
2: Use the map, planned rout and output of the external perception algorithms to plan high level behavior (e.g. go forward at 20 $\frac{\text{m}}{\text{s}}$ speed, turn right, etc.).
3: Translate the high level behavior into a specific car trajectory (e.g. expressed as a set of parameters defining a curve).

The final step consists of the control algorithms which influence the actuators, an in turn the physical world. The control algorithms translate the car trajectory to the exact commands to the actuators, and the actuators affect car systems, such as the steering wheel, brakes or throttle.

1.1.2 Sensor Characteristics

As mentioned previously, different sensors have different capabilities (described closer in Chapter 2). Consequently, they are placed at different locations on the car (shown in Figure 1.2). The next few paragraphs succinctly present the main differences.

Cameras are outstanding at mimicking the way a human driver would perceive the environment. If it is easy to distinguish between 2 classes of objects (e.g. a car and a pedestrian) for a human using his eyes, then the task is also easy for a camera-based algorithm. Hence they are particularly good at tasks

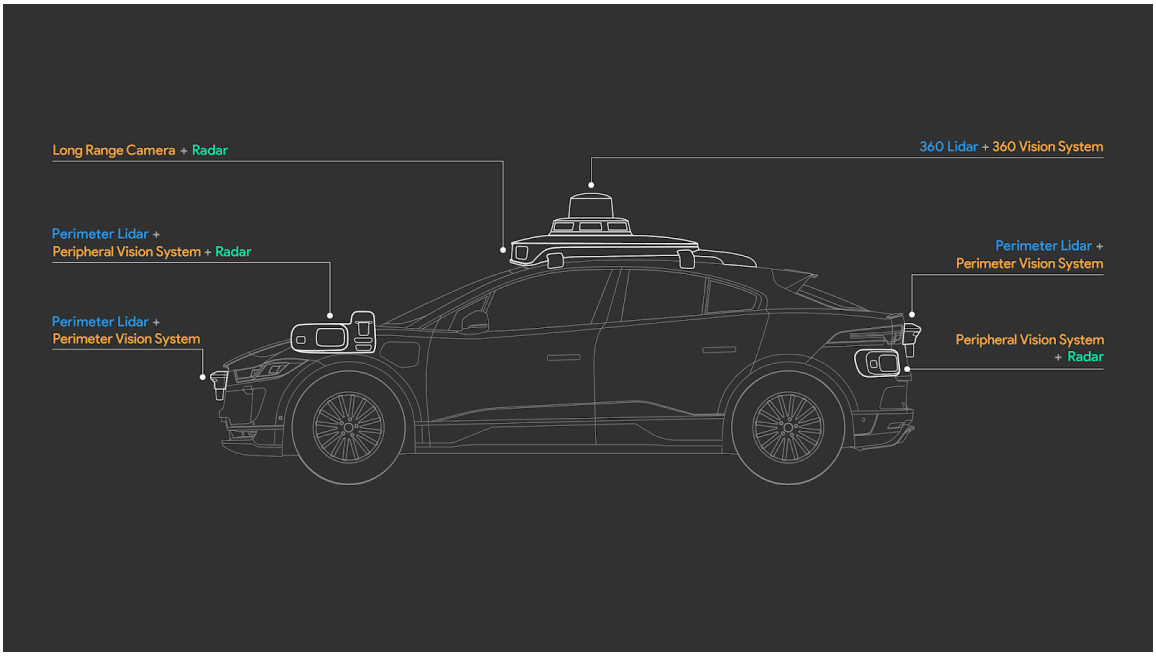


Figure 1.2. Diagram showing sensors and their localizations in a road-approved autonomous car from Waymo. Diagram was taken from the Waymo blog https://waymo.com/blog/2020/03/introducing-5th-generation-waymo-driver.html.

such as recognizing traffic signs. Additionally, cameras are quite cheap and provide excellent resolution in azimuth and elevation. On the other hand, camera-based algorithms are notoriously bad at measuring the absolute distance to an object.

Lidars are great at providing high-resolution depth maps, however, they are prohibitively expensive to use in most cars. In practice, they are used mostly in research projects, such as Waymo, (Sun et al. [2020]), NuScenes (Caesar et al. [2019]), or KITTI (Geiger et al. [2013]), and top-of-the-line vehicles.

On the other hand, automotive radars are relatively cheap (even 2 orders of magnitude cheaper than lidars). They produce reasonably good distance estimates. Yet radars produce relatively sparse output (on the order of 64 reflections), furthermore, the reflections are irregularly spaced. Most reflected waves come from metal targets or targets whose geometry is most suitable for reflections towards the emitter (e.g. perpendicular planes). Reflection accuracy in the horizontal plane (azimuth) is usually approximately 1 deg (Bialer et al. [2021]) and significantly worse in the vertical plane (elevation).

When one has an unlimited budget, there is a strong case to be made to use all three of the aforementioned sensors in an autonomous vehicle. However, as previously mentioned, the lidars are quite expensive. Therefore, it would be highly beneficial to produce a lidar surrogate on the basis of the data from the 2 cheaper sensors - camera and radar. One form of representation of such a surrogate is a dense depth map. A dense depth map is a scene representation akin to a picture, where each pixel encodes dis-

tance to the object. I believe it is a more natural form of representation than a pointcloud, as it shows the world from the sensor's point of view. In this work, my main goal is to create a dense depth map using data from an automotive radar alone, or in conjunction with a camera. To the best of my knowledge, this is the first solution producing a dense depth map on the basis of solely automotive FMCW radar output.

1.2 Problem Formulation

1.2.1 Motivation

In a very broad sense, the goal of this work is to use neural networks to extract more information from an automotive Frequency Modulated Continuous Wave (FMCW) radar. In order to achieve this goal I decided to use a low-level radar output - specifically I used a Radar Data Cube (RDC) as the network input. RDC is a three dimensional representation of the radio wave reflections detected by the radar at a particular point in time. First dimension represents the distance between the sensor and the reflection source, second dimension represents the relative radial velocity between the sensor and the reflection source and the third dimension represents the signal at the different antennae. In Chapter 2 I present a closer look at the principle of operation of the FMCW radars and the way RDC is calculated.

More specifically, my goal was to check, whether it is possible to use a neural network to extract a lidar-like scene representation (a dense depth map) from an automotive FMCW radar. I chose the depth map as the representation which I trained the network to produce for 2 main reasons.

The first reason is that the task is ambitious - the standard automotive radar output is much different than a dense depth map (presented in Figure 1.4). Simple interpolations between radar depth measurements fail to produce a representation of a scene that would be interpretable by a human. Hence, if the model is able to produce a visually recognizable scene representation, we can conclude that it is much better at extracting the information from the signal than the standard algorithms used in radars. In fact, my algorithm achieves this goal, as evidenced by the Figure 1.5. In that figure, I present a camera image, dense depth map obtained using lidar and the output of my radar model. Contrary to the raw reflections interpolation, my depth map reconstruction from low-level radar data produces a visually understandable scene. I consider it to be a good marker of the performance of my solution.

The second reason for choosing to predict a dense depth map is more mundane - the ease of dataset construction. A dataset of dense depth maps can be created without the need for costly human labeling. I created a 1,000,000 frames dataset using a rooftop-mounted lidar and a WeaveNet neural network (that was purpose-designed by me in the preliminary phase of the research described in this dissertation). As

the depth completion datasets can be created in an automated way (hence relatively cheaply), they can be used for pretraining a neural network whose final task is different.

1.2.2 Inputs and Outputs

As stated in Section 1.2.1, this dissertation deals with the problem of transforming the output of an automotive FMCW radar into a dense depth map. I achieve it using a custom-designed neural network. I created two versions of the network: a radar-only version and radar+camera version. The radar-only version of the network uses only RDCs to create a dense depth maps, and the radar+camera version utilizes both RDC and camera images. Schematic representation of its inputs and outputs is presented in Figure 1.3.

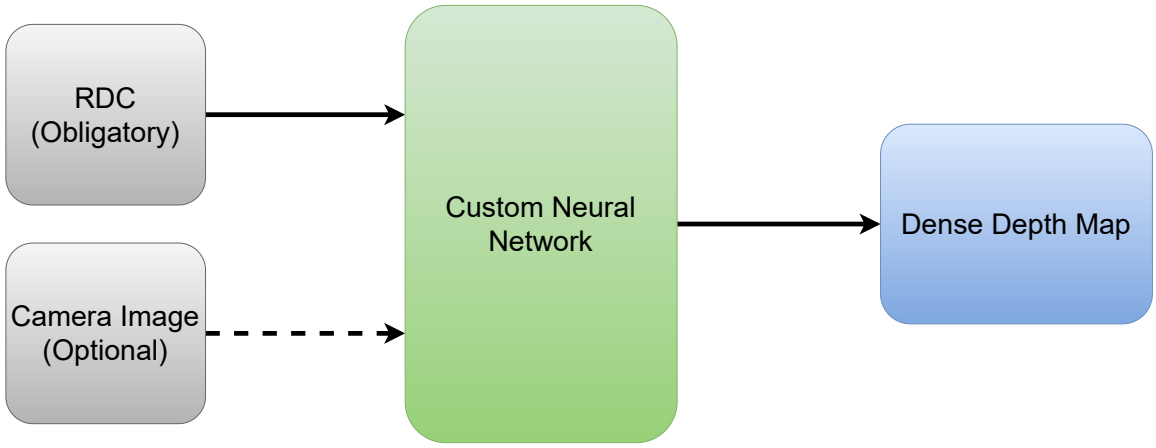


Figure 1.3. Schematic representation of the inputs and outputs of the radar depth completion network. The camera image input is optional.

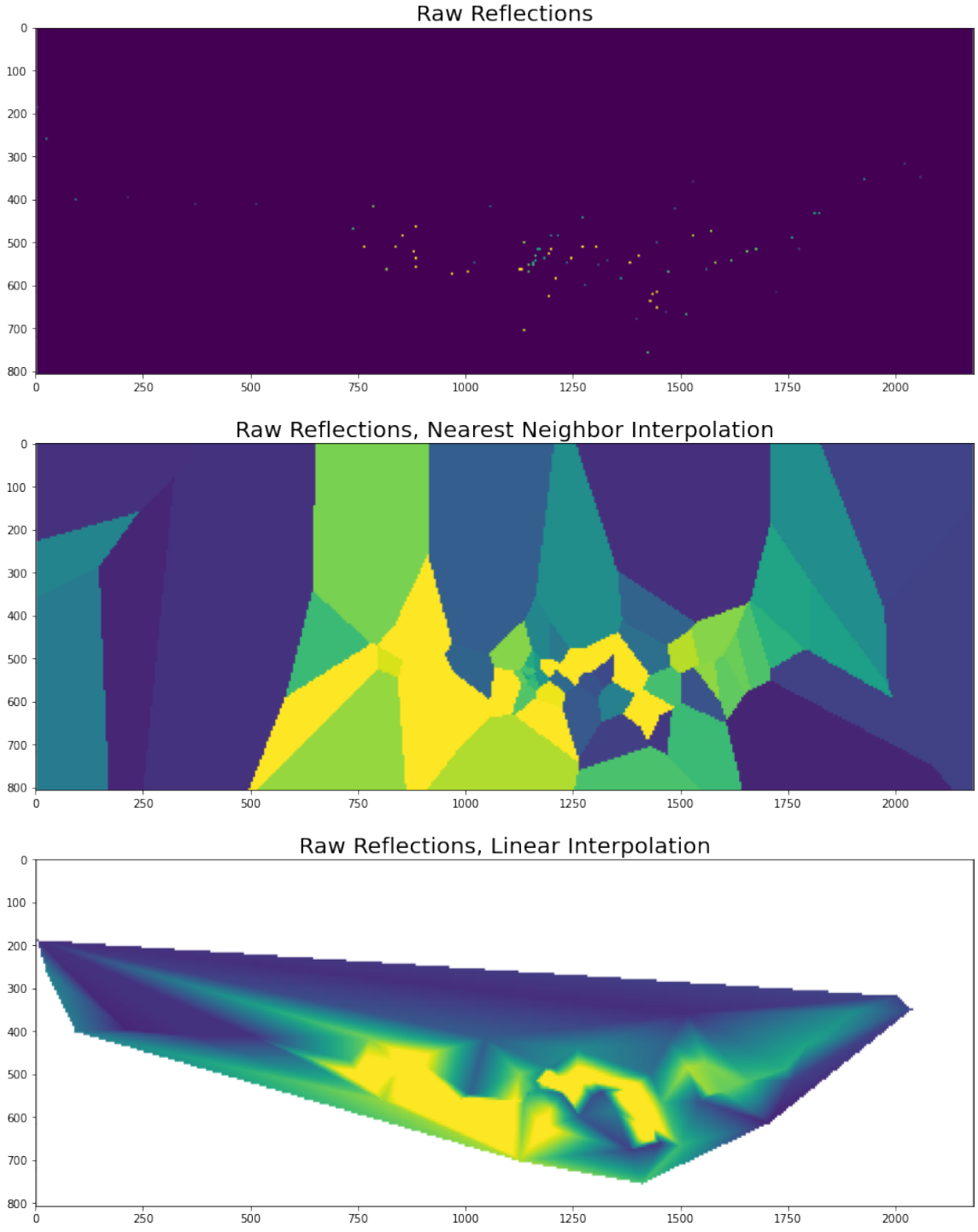


Figure 1.4. Depth maps created using raw radar reflections and naive interpolations (linear and nearest neighbor), distance is color-coded. Numbers on the axes denote pixel coordinates.

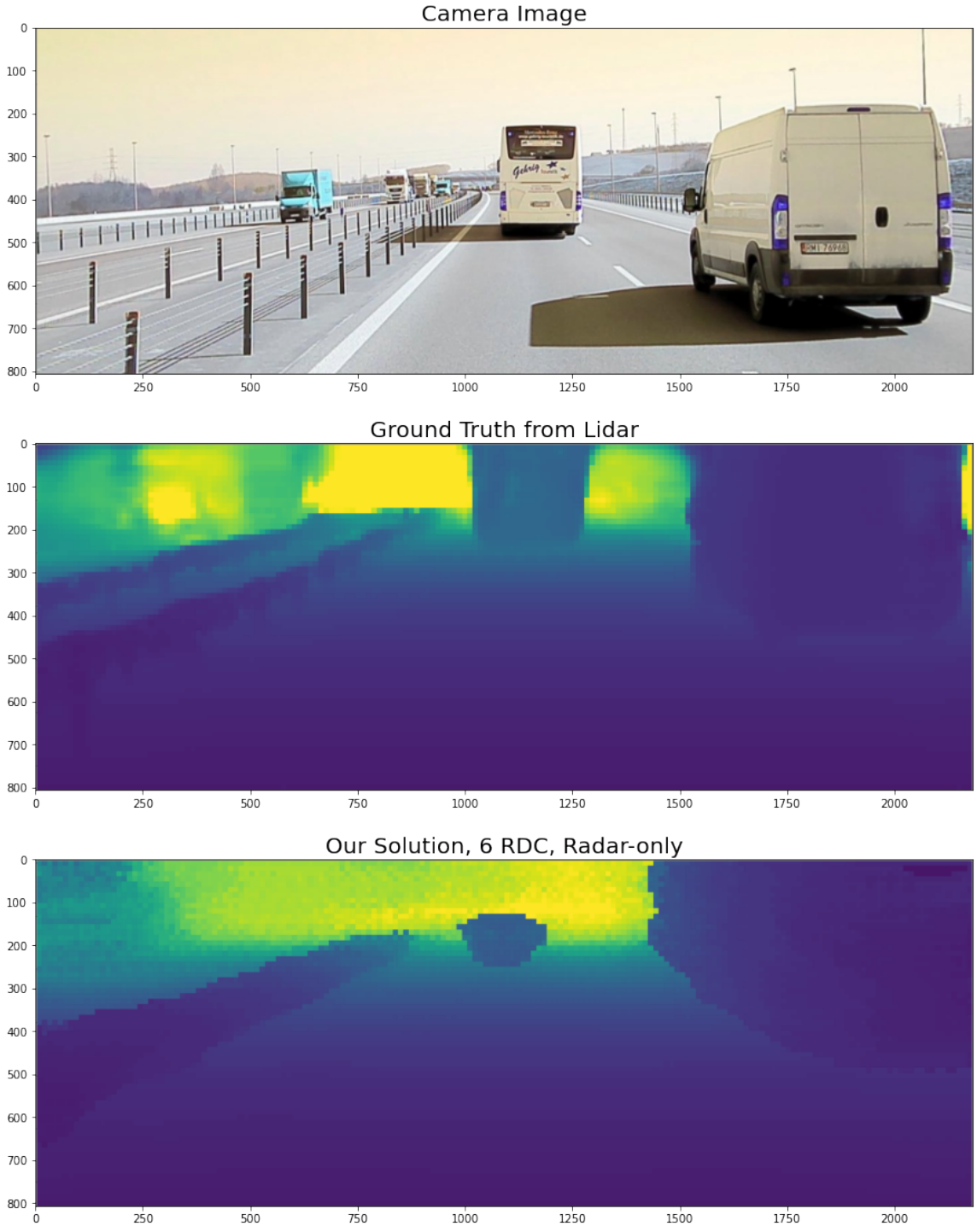


Figure 1.5. Camera image, ground truth (GT) from lidar and a depth map reconstruction using low-level radar data (my solution), distance is color-coded. Numbers on axes denote pixel coordinates.

1.3 Theses

The goal of this dissertation is to show the feasibility of depth completion (i.e. producing a dense depth map from a sparse depth input) for automotive Frequency Modulated Continous Wave (FMCW) radars and the feasibility of camera - FMCW radar fusion for depth completion. It can be distilled in the following theses:

a Low-level FMCW radar output can be processed in such a way, that a dense depth map is produced.

b A model fusing low-level FMCW radar output with camera images can produce a higher quality depth map than an analogous model utilizing only camera images.

1.4 Scope of Work and Organization of the Thesis

The end goal of this work has always been creating a dense depth map using the outputs of an automotive radar. However, to achieve this goal, I had to use quite a circuitous path. Graphically, it is represented in Figure 1.6. The first thing needed to develop a machine learning model is the dataset. I had access to data collected using a car with a forward-facing FMCW radar, a forward-facing camera, and a lidar on top. The 'independent variables' (RDCs from radar and camera images) were easy to measure directly, however, the lidar output is not dense, but only semi-dense. For example in the KITTI dataset (Uhrig et al. [2017], Geiger et al. [2013]), Velodyne HDL-64E lidar provides approximately 20,000 detections in the field of view of the front camera, while the camera itself provides an image with 465,750 pixels (375x1242). Lidar depth completion is a known problem, with known solutions, however, none of them could be safely applied to my case directly. I needed to create a dense depth map from the pointcloud from a Pandora lidar projected on a plane located closer to the grille (radar location) than the lidar itself. Such arrangement results in an unequal distribution of the measurements in the depth completion plane. Therefore, I had to create a lidar depth completion solution that is indifferent to the measurement density. I did it, by defining a WeaveNet neural network architecture, and a specialized universal sparsity training procedure. I trained that network on the publicly available KITTI lidar depth completion dataset (Uhrig et al. [2017]). The results I obtained are described in Chapter 4. During the work on WeaveNet, I noticed that an uncommon training procedure was very beneficial in its training. This procedure consisted of utilizing zero-centered, additive weight perturbations during neural network training. I dug deeper into it, demonstrated its usefulness for a range of neural architectures, and showed why it helps to better utilize the neural network weights (described in Chapter 3). When I completed the work on WeaveNet, I used it to create dense depth maps using the data collected using the test car.

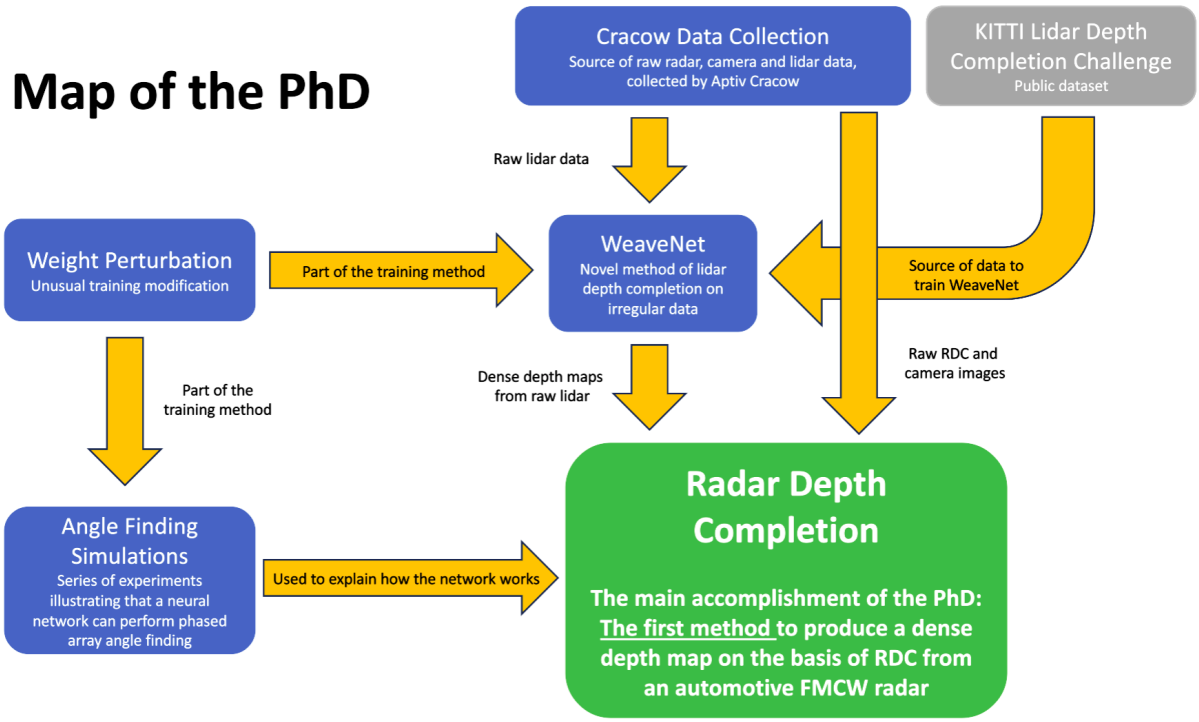


Figure 1.6. A graphical representation of the contents of the PhD.

Once I had the dataset (the process of dataset collection and processing is described in Chapter 6), I could delve into developing a solution to the problem of radar depth completion. The direct work on the radar depth completion problem is described in Chapter 7. In particular, I defined a novel neural network architecture suitable for transforming the data from RDC into abstract channels in the azimuth-elevation plane and trained it for the task of depth completion. I also created a neural network fusing the RDC and visual data and trained it to produce dense depth maps. I have demonstrated that my network is able to extract useful information from RDC by showing that a radar+vision network achieves significantly better results than an analogous network utilizing only the camera images. Finally, I wanted to do something to 'illuminate the black box' of my radar depth completion solution. I created a toy model of the FMCW radar anglefinding when reflections from 2 different targets are superimposed. I have shown that a neural network is capable of disentangling the 2 sources of signal and successfully perform anglefinding in such a scenario (Chapter 5). The zero-centered, additive weight perturbations were also used for training this network.

This dissertation relies on the author's previously published works and expands them into a standalone dissertation. The published works on which it relies are:

- Weight Perturbation as a Method for Improving Performance of Deep Neural Networks (Nowak and Lelowicz [2021]). The paper describes an improvement in a neural network training method and forms the basis of Chapter 3. It was authored by Mariusz K. Nowak and Kamil Lelowicz.
- WeaveNet: Solution for Variable Input Sparsity Depth Completion (Nowak [2022]). This paper describes a lidar depth completion solution that was instrumental in creating my dataset and forms a basis of Chapter 4. Mariusz K. Nowak was the sole author of this paper.
- Novel Neural Network Architecture for Obtaining Dense Depth Map from Radar Data Cube (to be published, **?**). The paper describes my solution for creating the dense depth map from RDC data and forms the basis of Chapter 7. It was authored by Mariusz K. Nowak, dr Mateusz Komorkiewicz and Prof. Paweł Skruch.

2 Automotive Depth Sensors

In order to complete tasks that require movement in 2D or 3D space, it is necessary to know the positions of other objects. Humans can estimate the distance to an object (depth) using their eyes as sensors and processing the information using stereovision or visual cues. Similarly, if we want an autonomous or semi-autonomous system to have depth estimation capacity, we need to equip it with appropriate sensors and algorithms to process the raw data from the sensors. In this chapter, I present the most popular depth sensors used in automotive settings.

2.1 Lidar

Lidar (originally an abbreviation of Light Detection and Ranging) is a sensor utilizing visible light to measure the distance to the nearest object. Typically, it is done by using multiple, vertically spaced lasers placed in a rotating shell (like the one shown in Figure 2.1).

The distance to target r can be determined using the following formula (to see why, please look at Figure 2.2):

$$r = \frac{ct}{2}, \tag{2.1.1}$$

where c is the speed of light, and t is the measured time.

The azimuth (angle in the horizontal plane) to target can be determined because the position of rotating lasers at the time of sending laser pulses is known. Elevation (the angle in the vertical plane) may be determined because lasers sending pulses are themselves slightly slanted in the vertical plane. An example of lidar measurements superimposed on a camera image can be seen in Figure 2.3.

Time-of-Flight (TOF) camera is a version of lidar in which there are no scanning lasers - a different modulated light source is used instead. Then, the camera imager is used to measure the time of flight of light from the illumination source to the illuminated object and back (Li [2014b]).

Figure 2.1. Pandora lidar Source: Pandora lidar user manual, available at: https://www.symphotony.com/wp-content/uploads/20181015-Pandora-Users-Manual.pdf

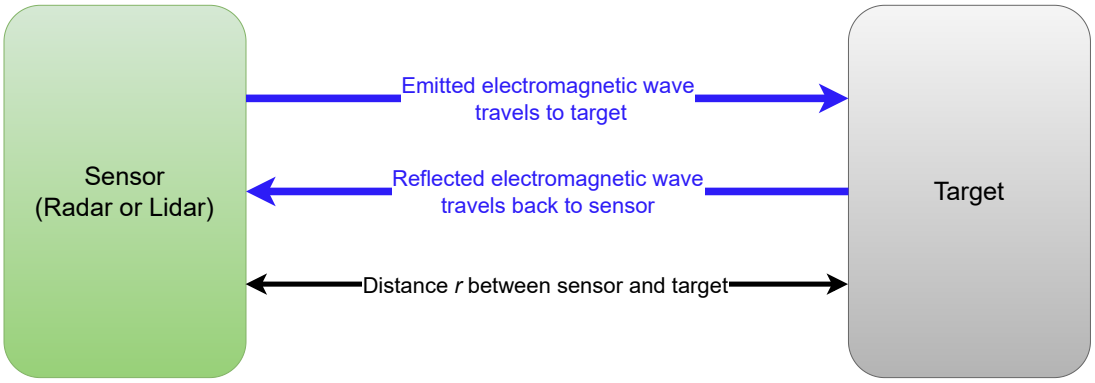


Figure 2.2. Radar and lidar ranging principle.

Figure 2.3. Pandora lidar measurements cast on camera image, color encodes distance. Source: Pandora lidar user manual, available at: https://www.symphotony.com/wp-content/uploads/20181015-Pandora-Users-Manual.pdf

2.2 Radar

2.2.1 Pulse Radar

Pulse radar is the simplest and oldest version of radar (originally an abbreviation of Radio Detection and Ranging). It can measure the distance to the target, by sending an electromagnetic pulse and measuring the time it takes for the waves to travel to the target, reflect, and be detected by the radar. Then, the distance r can be determined using the same formula (2.1.1) as for lidar (to see why, please look at Figure 2.2).

Simultaneously, the relative radial velocity of the target with respect to the radar can be obtained by looking at the Doppler shift in the received signal. Relative radial velocity Δv can be found using the following formula:

$$\Delta v = \frac{\Delta f c}{2f}, \tag{2.2.1}$$

where c is the speed of light, Δf is the measured frequency difference and f is the base frequency of the emitted wave.

The simplest way to detect the angle to target in a pulse radar is to use a rotating directional antenna.

2.2.2 Frequency Modulated Continuous Wave Radar

In this section I cover the basics of FMCW radars with multiple antennae. Such radar is shown in Figure 2.4. A more detailed description of FMCW radars can be found in (Wolff [1998]), (Markel [2022]) or in (Richards [2014]).

FMCW radar emits 'chirps' - pulses of radio waves whose frequency changes over the transmission time. Typically, chirp frequency increases linearly with time, as described in the Equation (2.2.2) (Markel [2022]) :

$$f_T(t) = f_c + \frac{B}{T}t \quad \text{for } t \in [0, T), \tag{2.2.2}$$

where f_c is the carrier frequency, B is the bandwidth, and T is the length of the chirp. An example of chirp frequency modulation is presented in Figure 2.5. Please note, that chirps are not emitted directly one after another - they are spaced by time intervals, during which radar emits no radio waves. These non-emitting intervals are necessary for signal processing.

When the radar receives the echo of the transmitted pulse, it is possible to calculate the time it took for the radio wave to travel to the reflector and back by looking at the difference between the frequency

Figure 2.4. An image of an automotive FMCW radar (not the one used in this work)
Source: Presentation of Aptiv radars, available at:
https://www.aptiv.com/en/solutions/advanced-safety/adas/radars

of the currently emitted wave and the frequency of the echo signal. It is achieved by mixing the received echo signal with the currently transmitted radio waves and looking at the resulting waveform. Consider a waveform that arises from the superposition of two sine waves of frequencies f_1 and f_2 and equal amplitude. It is described by the following equation:

$$\begin{aligned} x(t) &= \sin(2\pi f_1 t) + \sin(2\pi f_2 t + \phi) \\ &= 2\cos(2\pi \frac{f_1 + f_2}{2} t + \frac{\phi}{2}) \sin(2\pi \frac{f_1 - f_2}{2} t - \frac{\phi}{2}). \end{aligned} \tag{2.2.3}$$

The resulting waveform is a product of one wave oscillating with high-frequency $\frac{f_1+f_2}{2}$ and another wave oscillating with low frequency $\frac{|f_1-f_2|}{2}$, as can be seen in Figure 2.6.

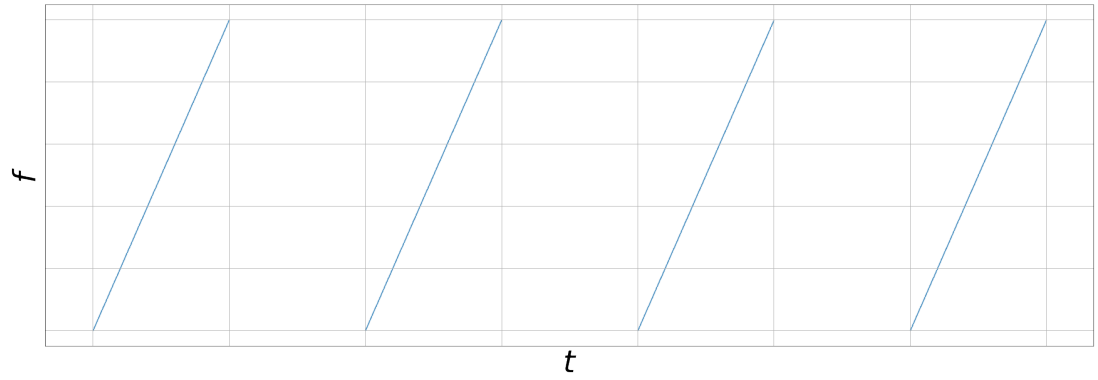


Figure 2.5. Chirp frequency vs time.

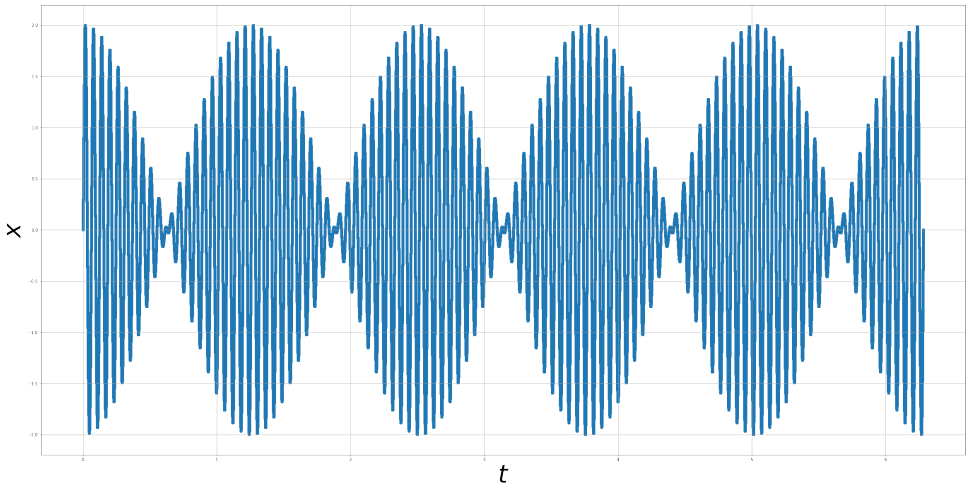


Figure 2.6. Example of superposition of sine waves.

Moreover, the resulting waveform is dependent on the phase difference between the emitted and received waves (ϕ). Looking at the difference between phase difference ϕ of consecutive chirps, it is possible to determine the relative velocity of the radar and the reflector with high accuracy (due to the Doppler effect).

In practice, the distance to the target and its relative velocity are found using Fast Fourier Transforms (FFT) (Markel [2022]). In order to determine the distance to the target, the signal is sampled in multiple places during one chirp, and a Fourier transform is calculated for this discrete time series (in literature, it is usually called fast time FFT (Markel [2022])). Beating frequency is obtained as the result of the fast time FFT. Similarly, to obtain the relative velocity of the target, the signal is sampled across multiple chirps, and FFT is performed on this time series (slow time FFT (Markel [2022])). The rate of the phase difference change between chirps is obtained as the result of the slow time FFT. This consequently, makes it possible to estimate the relative radial velocity of the target. After obtaining the target distance and relative radial velocity, the measurements are placed in a Radar Data Cube (RDC).

RDC is a way to represent the low-level radar output. It is a 3D structure inhabited by radar detections. The first axis consists of the range bins (each detection is placed in a specific bin on the basis of the distance between the radar and the target), the second axis consists of the relative radial velocity bins (detections are segregated on the basis of relative radial velocity between radar and target), and the third axis contains information regarding the signal detected at particular antennae (which allows for determining azimuth and elevation to target in downstream processing, as described in Section 2.2.2.2). A vector containing the information about the received wave on all the antennae, for a given range bin and relative radial velocity bin is called a beamvector. Automotive FMCW radars typically operate in alternating long and short-range modes, which differ by the size of the range bins, and consequently maximal operating distance and range resolution.

2.2.2.1 RDC Compression and Clutter Removal

The radio waves received by radar are a superposition of signal - the echo reflected from intended radar targets (e.g. other vehicles) and noise (e.g. thermal noise on the receiver or reflections from unintended targets - for example, ground reflections in automotive setting). Typically, a specific range-velocity cell of an RDC is treated as containing signal if the power of the measured radio waves is above some threshold. While some components of the noise are relatively easy to estimate (e.g. thermal noise on the receiver can be estimated if we know the sensor temperature), other components of the noise, such as the reflections from the ground, cannot be easily measured (Richards [2014]).

One approach to noise level estimation is the so-called Cell Averaging Constant False Alarm Rate (CA CFAR) algorithm (Richards [2014]). For the purpose of the algorithm, it is assumed, that noise can be modeled with a zero-mean normal probability distribution (Richards [2014]). Then, a threshold is chosen in such a way, that noise-only cells (without actual reflections from valuable targets) exhibit measured power above the threshold at a constant rate (hence the Constant False Alarm Rate name). This requires an estimation of the noise level. For each cell (Cell Under Test (CUT)) the noise level is estimated independently. It is done by averaging the power of measured radio waves in the other cells in CUT's neighborhood in the RDC (CUT is not included). Usually, the closest cells (e.g. the ones which differ by no more than one range or radial velocity bin) are also excluded from the calculation (Richards [2014]). These 'guard cells' are not used to estimate the noise level, under the assumption that if the CUT contains echo from an actual target, the closest cells are also likely to contain reflections from that target. The width of the zone of the guard cells and the width of the zone of the cells used in noise level estimation may differ, depending on the implementation of the CA CFAR algorithm. A schematic

representation of an RDC, showing which cells are used to estimate noise in the CA CFAR algorithm is presented in Figure 2.7.

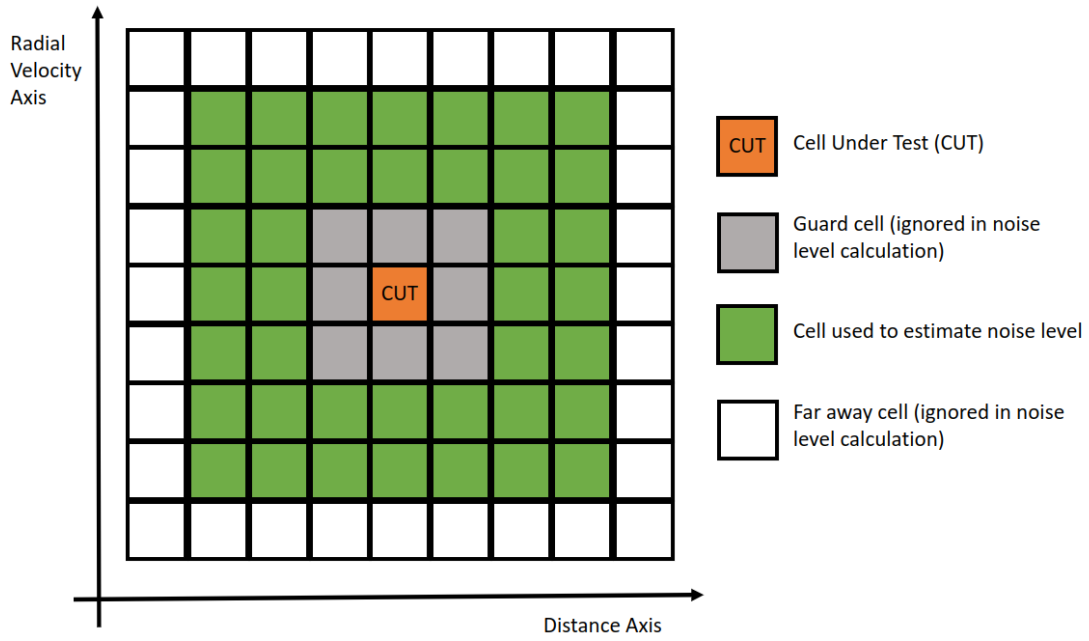


Figure 2.7. Schematic representation of cells used in CA CFAR calculations. The width of the zone of the guard cells and the width of the zone of the cells used in noise level estimation may differ, depending on the implementation of the algorithm.

After estimating the standard deviation of the normal distribution with which the noise-generating process is modeled for CUT, a CFAR threshold is chosen. If the power of the radio wave measured in CUT is above the threshold, CUT is considered to contain signal. Otherwise, it is considered to contain only noise and is masked with zeros.

CFAR algorithm can be used with methods of noise estimation other than looking at neighboring cells in the range-velocity plane of RDC. One such method is Smallest-Of Cell-Averaging CFAR (SOCA CFAR). In SOCA CFAR, multiple windows for noise estimation are used, and then, the window with the smallest estimated noise is used to determine the CFAR threshold (Richards [2014]). SOCA CFAR helps in case there are multiple targets with overlapping noise estimation windows.

Another alternative to CA CFAR is Order Statistic CFAR (OS CFAR) (Blake [1988]). If a sequence of samples is sorted in ascending order, then the k^{th} element of that sequence is called the k^{th} order statistic. OS CFAR estimates the noise level using k^{th} order statistic (for some predetermined k) power measurement in the noise estimation window, instead of averaging the power measurements from the

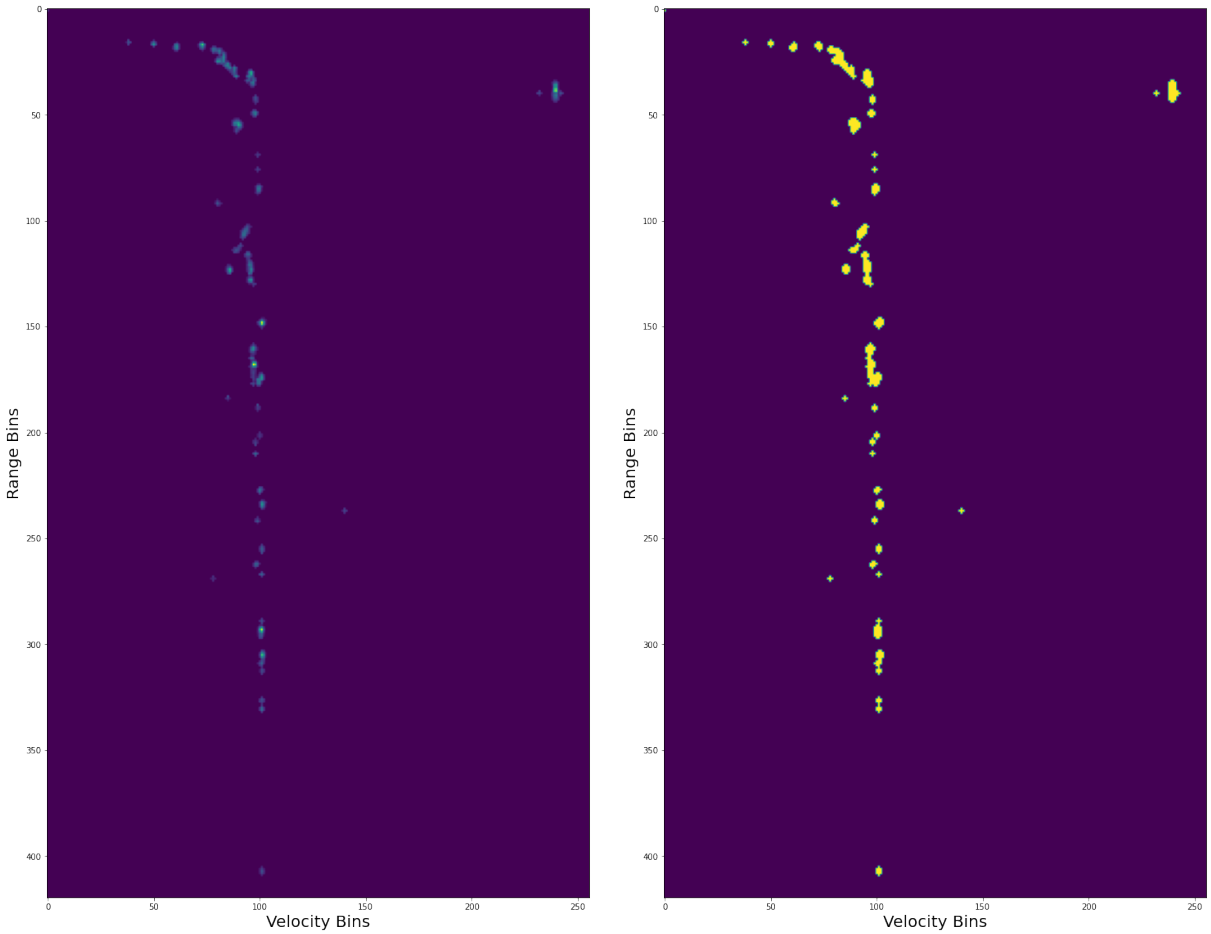


Figure 2.8. Figures showing an example RDC after clutter removal. The figure on the left shows the sum of absolute values of the signals at all the antennae for a given range-velocity bin combination. The figure on the right shows the same RDC, albeit instead of the sum of absolute values at antennae it shows all the nonempty range-velocity bin combinations.

whole window (Blake [1988]). Such approach makes OS CFAR more robust to the existence of multiple targets neighboring each other in the range-velocity plane than vanilla CA CFAR.

CFAR algorithms that change the size or shape of the noise estimation window are called adaptive CFAR (Richards [2014]). The basic version of the adaptive CFAR was described by (Finn [1986]). In his approach, it is assumed that there are 2 (continuous) clutter zones in the CFAR window. Noise is estimated in all possible combinations of the clutter zones (e.g. first 4 cells in clutter zone 1, next 7 cells in clutter zone 2). Then, a maximum likelihood estimator (MLE) is used to choose the split of the noise estimation window into clutter zones, that maximizes the log-likelihood, given the measurements in different cells. Finally, the noise level is estimated using only the cells from the clutter zone containing CUT.

Instead of using the data from neighboring RDC cells to estimate noise level, it is possible to utilize the data from the same cell, gathered over multiple radar frames. For example, it can be done using an exponential moving average to aggregate noise estimates over time (Richards [2014]).

Clutter removal has a two-fold advantage in the current generation of automotive radars. Firstly, it removes the false detections, which would decrease the quality of the output of the downstream algorithms (e.g. tracker). Secondly, it significantly decreases the computational needs of downstream processing (e.g. if 99% of beamvectors are removed before angle finding, then angle finding needs to perform 99% less calculations). In Figure 2.8 I show an example of an RDC after the clutter removal. Please note 3 features of a typical RDC in an automotive FMCW radar:

a Overwhelming majority of range-velocity combinations are empty - in our dataset typically more than 99 % of bins are empty - signals at that bins are too weak to pass the CFAR noise thresholds.

b Plurality of the nonempty bins are arranged along a vertical line - they represent targets moving with similar radial velocity in the frame of reference of the radar. This effect is due to the fact that the vehicle on which the radar is mounted is moving with nonzero velocity.

c Each nonempty bin borders another nonempty bin - in the figure on the right one can clearly see that the nonempty bins form little + signs, with the primary reflection in the center and secondary reflections in the bordering bins. Apart from that, the reflections tend to form bigger clusters - likely coming from a single target.

2.2.2.2 Angle Finding

An automotive FMCW radar does not scan the surrounding area with a radio beam. Instead, it sends the radio wave in all directions, and uses multiple receiving antennae to determine the angle to the target from which the wave reflected. Thanks to this approach, it has Instantaneous Field of View (IFOV) equal to its Field of View (FOV). Consider the situation shown in Figure 2.9. Radar antenna receives signal reflected from a target at azimuth α. It follows from basic trigonometry, that:

$$r = \frac{d + \epsilon}{\sin(\alpha)}. \tag{2.2.4}$$

Consider a radar that has 2 antennae spaced ϵ from each other in the horizontal plane - one of them at a horizontal distance d from the target, the other one $d + \epsilon$. In such case, the difference in path length $(r_2 - r_1)$ that the wave travels can be expressed as:

$$r_2 - r_1 = \frac{(d + \epsilon) - d}{\sin(\alpha)}, \tag{2.2.5}$$

consequently:

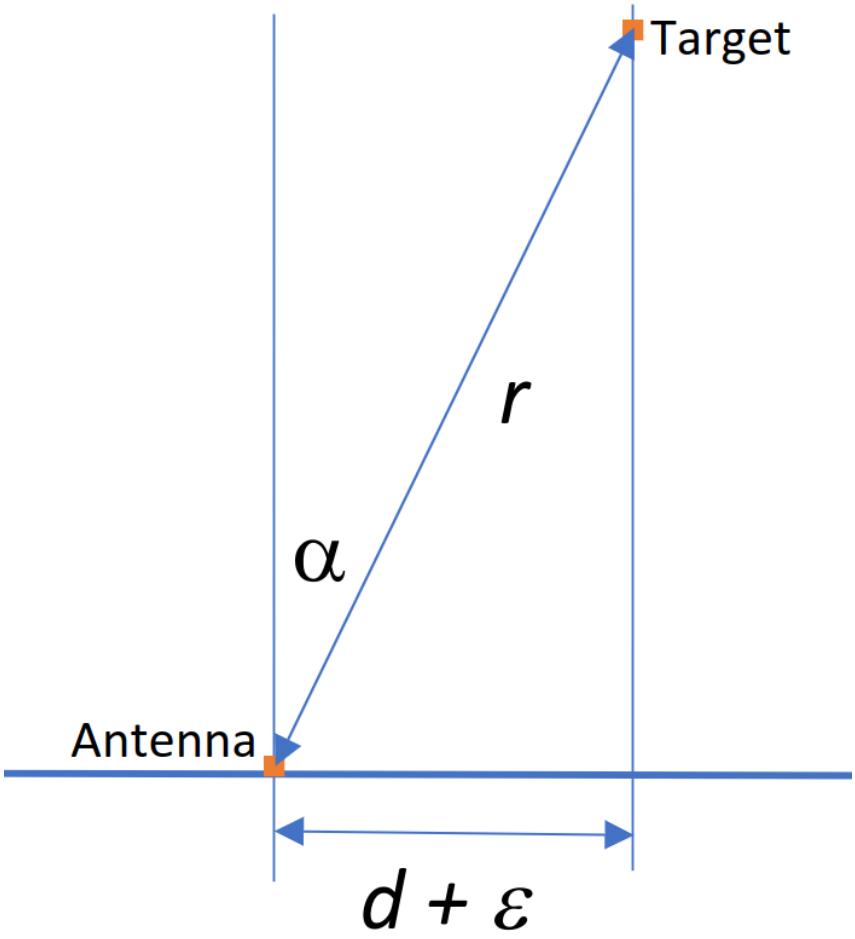


Figure 2.9. Notation used in the description of FMCW radar angle-finding.

$$\alpha = \arcsin(\frac{\epsilon}{r_2 - r_1}) = \arcsin(\frac{\epsilon}{c(t_2 - t_1)}), \tag{2.2.6}$$

where c is the speed of light.

We cannot observe the time of arrival of the reflected wave directly, however, we can observe the phase difference between the waves received at different antennae. Given that the distance between antennae in a single radar is on the order of the wavelength of the emitted wave, the phase difference information can be used as a substitute for direct measurement of radio wave travel time. Hence, if the radar has more than one antenna, it is possible to determine the azimuth (angle in the horizontal plane) to the target by comparing the phase difference between received echo waves at horizontally spaced antennae and elevation (angle in the vertical plane) by comparing the phase difference between received echo waves at vertically spaced antennae. Azimuth is much more important than elevation in automotive settings, therefore typical automotive radars have more horizontally spaced antennae than vertically spaced antennae, and consequently, the azimuth accuracy is much higher than elevation accuracy.

In practice, finding phase difference between antennas (and consequently angle finding) can be most easily done by performing an FFT along the antenna axis in RDC. There are also more advanced algorithms for angle finding, such as MUSIC (Schmidt [1986]) or ESPRIT (Roy and Kailath [1989]). In Chapter 5, I show that it can be performed using a neural network.

2.3 Ultrasonic Range Finder

Ultrasonic range finder, like radar and lidar, also uses emitted waves to measure the distance to the target. However, unlike the other sensors, it uses sound waves, rather than electromagnetic waves. Because of that, it is much worse at sensing far-away targets. In automotive, ultrasonic range finders are almost exclusively used as parking-assist sensors. Wang et al. [2014] provides a literature review of such applications.

2.4 Estimating Depth from Non-Depth Sensors

2.4.1 Stereo Vision

Comparing images from two or more closely mounted cameras may be used to determine the distance to an object in the image. To do it, the images from both cameras are first rectified (i.e. lens distortions are removed, making the images effectively pinhole camera projections). More on the process of rectification can be read in Lelowicz [2019]. After the rectification step, matching points (i.e. pixels corresponding to one point in 3D space) are localized. If the relative positioning of the cameras is known, then by comparing the localizations of those pixels in the imagers of the cameras it is possible to calculate the distance to the target. A more in-depth discussion of computer stereo vision may be found in Li [2014a]. Stereovision naturally produces a very dense depth representation (it is possible to determine depth for each pixel). Therefore it is in some applications used in conjunction with very accurate sparse sensors to produce higher quality output (e.g. stereo vision was used in conjunction with lidar measurements to produce KITTI depth completion dataset labels (Uhrig et al. [2017])). Quality of stereo vision depth measurements deteriorates for distances much bigger than the distance between cameras, which limits its application for far-away objects.

Most modern works on stereo vision utilize it in connection with neural network-based models (notably, used for matching the points belonging to the same physical object on the images from 2 cameras). Some of the models achieving state of the art performance on KITTI Stereo Evaluation challenge (Menze and Geiger [2015]) include Weinzaepfel et al. [2023], Xu et al. [2022a], Sun et al. [2022], Xu

et al. [2023], Li et al. [2022a], Liu et al. [2022], Chen et al. [2023], Sommer et al. [2022] and Cheng et al. [2020]).

2.4.2 Monocular Depth Estimation

When the camera takes a picture, information from the 3D world is projected into a 2D (azimuth-elevation) plane of the camera imager. It is impossible to directly recover depth information from this 2D representation, hence monocular depth estimation is an ill-posed problem. However, it is still possible to extract some depth information from images - KITTI Depth Prediction Challenge (Uhrig et al. [2017]) is a prime example of a dataset that deals with monocular depth estimation in an automotive setting.

There are 2 main approaches to training a neural network for the depth prediction task - treating it as a regression problem or as a classification-regression problem (Li et al. [2022b]). When treating the problem as a regression task, a regression loss is directly used for each pixel for which the depth was predicted. On the other hand, when the problem is approached as a classification-regression task, at each predicted pixel the depth is assigned to one of the discrete bins. Final depth prediction is then computed on the basis of the probabilities computed for different bins (e.g. as the argmax or a linear combination of the depth from different bins). Currently, the models taking the classification-regression approach seem to perform better in the KITTI Depth Prediction Challenge (Uhrig et al. [2017]).

Work by Eigen et al. [2014] is among the most highly cited in the field of monocular depth estimation. They utilize a convolutional neural network to predict depth, taking the direct regression approach. Their network consists of 2 streams - one stream quickly down-samples the image to extract coarse global information, while the other one processes the image at target resolution to extract fine local details. The information from the coarse stream is fed to the fine stream before making predictions.

Ranftl et al. [2021] have produced another seminal work in the field of monocular depth estimation. Rather than using a fully convolutional network, they transform the image into a set of tokens (extracted from non-overlapping image patches), use a transformer-style network on the tokens, and finally use a convolutional decoder to produce the output. They also formulate the problem as a regression task.

BinsFormer (Li et al. [2022b]) is currently the best-performing depth prediction model in the KITTI Depth Prediction Challenge (Uhrig et al. [2017]). The network takes the classification-regression approach, utilizing adaptive bins. The main novelty in their work is related to utilizing a transformer-style network to generate distance bins. Information from the image is also processed in parallel using a convolutional neural network. The final output is produced by using the information from the convolutional stream to classify the pixels into the adaptive depth bins determined by the transformer-style part of the network.

Some other of the top performing monocular depth estimation algorithms include (in the order of KITTI leaderboard positions): Shao et al. [2023b], Shao et al. [2023c], Liu et al. [2023], Piccinelli et al. [2023], Liu et al., Shao et al. [2023a], Ning and Gan [2023], Agarwal and Arora [2023], Shim et al. [2023] and Yuan et al. [2022].

3 Weight Perturbation

3.1 Introduction

Deep neural networks require significant computational power both to train and use them. In fact, the increase in size of the neural networks is closely linked with increase in their performance. Work by LeCun et al. [1989a] on applying gradient backpropagation to training a convolutional neural network to recognize handwritten digits is widely credited as one of the first applications of neural networks to a practical task. In LeCun et al. [1998] paper, authors show that a convolutional neural network can be used to perform best-in-class in the task of reading hand-written digits on bank checks. When GPUs arrived, Chellapilla et al. [2006] have shown that convolutional neural network inference on GPU can be up to 4.1 times faster than on CPU (also on the task of handwritten digit recognition). Work by Krizhevsky et al. [2012] (AlexNet) was a crucial moment for neural network development. AlexNet was trained on multiple GPUs and used a novel ReLU activation function to outperform all previous competitors on ImageNet (Deng et al. [2009]) competition (recognising objects on photos). VGG architecture (Simonyan and Zisserman [2014]) went further with increasing the convolutional network depth (up to 19 layers). ResNet architecture, utilizing additive skip connections to improve gradient propagation, (He et al. [2016]) was the next step for convolutional neural networks - both in terms of size and performance, with its biggest version having 152 layers. ResNet152 may mark the height of the focus on increasing the network depth in CNN research. Further papers, have largely focused on the network micro-structure, without the overt desire to increase its size. Examples of such approach can be seen for example in Inception family networks (Szegedy et al. [2015], Szegedy et al. [2016], Szegedy et al. [2017]), or MobileNet family networks (Howard et al. [2017], Sandler et al. [2018], Howard et al. [2019]).

The increase in the size of the neural networks unfortunately leads to increase in the computational complexity of the inference. The high computational cost of inference is particularly important when the network is deployed on an embedded device. On the other hand, the work done on weight pruning (LeCun et al. [1989b], Han et al. [2015], Li et al. [2016]) and lottery ticket hypothesis in particular

(Frankle and Carbin [2018]), shows that large part of network weights can be safely removed from the network, while affecting model performance in a negligible way. However, the weight pruning methods described in the literature do not decrease the real-life computational cost of running the networks on Graphics Processing Units (GPUs), since neural networks utilize parallel computations of the GPUs, where it is not efficient to remove just some connections from the network.

In this chapter, I propose an alternative approach to the problem of non-useful weights within deep neural networks. My solution, instead of removing the non-useful weights, tries to make them learn useful features. In particular, I propose a method to deal with cases in which all the weights in a particular network layer converge to extremely similar value, and thus fail to realize their learning potential. This method will be used during the training of the networks performing lidar depth completion and radar angle finding (described in chapters 4 and 5).

3.2 Related Work

3.2.1 Using Noise as a Regularizer

Using noise added to inputs or hidden neurons of the network is a common knowledge data augmentation procedure (it is referenced in numerous deep learning tutorials, such as Diaz [2020], Nelson [2020], Rusak [2020], Nair [2018]). Tools for adding noise to the input images are also implemented in commonly used libraries such as scikit-image (van der Walt et al. [2014]) or MATLAB (MATLAB [2021]). Adding Gaussian noise to the images fed to the neural network is probably the most common method of noise-based regularization (referred to in Diaz [2020], Nelson [2020], Rusak [2020], Nair [2018]). Other commonly used methods of noise-based regularization include adding uniformly distributed noise to the inputs (also called speckle noise (Nelson [2020])) or making some input pixels completely white or completely black (also called salt-and-pepper noise (Nelson [2020])). Adding noise to the output of the hidden layers of the network is a much less common regularization method, however, it is still a widely known technique (Diaz [2020], Nair [2018]), with an appropriate noise adding layer already implemented in Keras (Chollet et al. [2015]). An interesting example of adding adversarial noise to layer output is presented in You et al. [2018]. A different line of work concerns adding the noise to gradients during the learning process, as is presented in Neelakantan et al. [2015].

Adding noise directly to the weights of the network is mentioned on pages 238 and 239 of a famous Deep Learning textbook Goodfellow et al. [2016]. It is described as a method to make the model invariant to small-scale variation in weights, used particularly in recurrent neural networks (Goodfellow et al. [2016]). Adding noise to the weights of a deep convolutional network is currently not the usual tool

for regularization (it is not mentioned by authors of top-performing network architectures, e.g. Sandler et al. [2018], Chollet [2017], Tan and Le [2019], Huang et al. [2017], He et al. [2016]). The use of noise added to the weights in the literature is significantly different from what my chapter proposes, as my research focuses on additive noise as a tool to increase the learning capability of the network by allowing Stochastic Gradient Descent (SGD) based network optimizers to make the weights in non-performing (low weight variance) layers diverge and learn useful features.

3.2.2 Weight Pruning

Much progress in machine learning performance has been a result of creating deeper neural networks. Thus, most of the best-performing neural networks require a great deal of computational resources (Huang et al. [2019a]). This makes the deployment of networks on mobile phones or other resource-constrained devices challenging (Yang et al. [2017], Sze et al. [2017]). One of the most common methods for reducing the resource consumption of a neural network is pruning, which is the methodical removal of connections and parameters from the network. Network pruning can significantly reduce energy consumption when the network is deployed on a custom accelerator (Yang et al. [2017]).

Typically, the network has a layered structure, where the outputs of previous layers are the inputs for the next one and each layer is described by a separate set of trainable parameters (weights). The architecture can be defined as a family o functions $f(x, \cdot)$ and consists of a set of operations allowing to transform input into output, given some specific set of trainable parameters. The neural network in this case is a detailed parameterization of the given architecture $f(x, \theta)$ for a specific vector of parameters θ. The effect of the pruning operation is the transformation of the network $f(x, \theta)$ to the form $f(x, m \odot \theta')$, where $\odot$ is element-wise product operation and the vector $m \in \{0, 1\}^{|\theta'|}$ is a binary mask (the same size as θ and θ') which fixes certain parameters of θ' to zero.

Please note, that the parameters θ' can be different from the initial θ.

The pruned model $f(x, m \odot \theta')$ can be obtained in many different ways from the original form $f(x, \theta)$, however, almost all pruning methods are derived from the Algorithm 2 (Blalock et al. [2020]):

Multiple variations on the presented algorithm have been proposed. For example, the parameters can be pruned already during the initialization (Lee et al. [2019]) or the network can be modified by adding additional elements that are used to score the network weights (Molchanov et al. [2017]). Usually, weights are scored based on their absolute value or contributions to the network gradient or activation functions.

Algorithm 2 Generic Weight Pruning Algorithm

1: Randomly initialize the weight vector θ.

2: Train the network to convergence, yielding $f(x, \theta')$.

3: Initialize a mask with ones, $m = \{1\}^{|\theta'|}$.

4: **for** $iteration = 1, 2, \ldots$ **do**

5: Issue a score to each element of θ (a network weight), the score is used to prune the network:

$$f_{\text{score}} : \mathbb{R}^n \rightarrow \mathbb{R}^n,$$
$$f_{\text{prune}} : \mathbb{R}^n \times, \mathbb{R}^n \rightarrow \mathbb{R}^n$$
$$m = f_{\text{prune}}\left(m, f_{\text{score}}\left(\theta'\right)\right)$$

6: **end for**

7: After pruning, train the network $f(x, m \odot \theta')$ further.

8: The algorithm produces the final θ' and m.

3.2.3 Lottery Ticket

Pruning techniques can reduce the number of weights of a trained neural network by over 80%, without harming the accuracy (LeCun et al. [1989b], Han et al. [2015], Li et al. [2016]). However, training a pruned neural network model from scratch usually leads to worse performance than retraining a pruned model. Frankle and Carbin [2018] presented a methodology for finding subnetworks that can be trained in isolation and achieve performance on par with the original network. On this basis, they proposed the lottery ticket hypothesis:

Hypothesis 1 *A randomly-initialized, dense neural network contains a subnetwork that can be trained in isolation, in such a way that:*

a the subnetwork is initialized with the same set of weights that it had as a part of the original network.

b it can match the test accuracy of the original network after training for at most the same number of iterations as the original network.

Let the dense feed-forward neural network $f(x, \theta_i)$ with initial weights θ reach the minimum validation loss l at i^{th} iteration of the optimization on the training set. The hypothesis conjectures the existence of a mask $m \in \{0, 1\}^{|\theta|}$, such that when optimizing subnetwork $f(x, m \odot \theta)$ with fixed m on the same training set, it reaches validation loss l' at $i'th$ iteration for which $i' \leq i, l' \leq l$ and the number of nonzero elements in m is much smaller than size of θ .

In order to identify the winning ticket during neural network training, smallest-magnitude weights can be pruned iteratively, according to the Algorithm 3:

Algorithm 3 Lottery Ticket Training

1: Create an empty mask m_0 and randomly initialize the weights for the neural network $f(x, \theta)$.
2: **for** $i = 1, 2, \ldots$ **do**
3: Train the network for k iterations, yielding parameters θ_k.
4: Prune $p\%$ of smallest-magnitude weights in $m_{i-1} \odot \theta_k$, creating a mask m_i.
5: Reset the remaining weights to their values in θ.
6: **end for**
7: After the desired number of iterations the final mask m and the winning ticket $f(x, m \odot \theta)$ are ready.
8: Train the winning ticket $f(x, m \odot \theta)$ using a standard approach, getting the network $f(x, m \odot \theta')$ after the end of training.

The best results of this approach can be obtained for fully-connected networks. For convolutional networks, finding a winning lottery ticket is generally much harder. Consequently, the performance on the validation dataset is not necessarily better than when using the original network (Frankle and Carbin [2018], Zhou et al. [2019]).

3.3 My Method

The goal of my training method is to maximize the learning capability of the network, while keeping its size and inference time on a GPU constant. The weights of a deep neural network are frequently not utilized in an efficient way, i.e. some layers have weight variance very close to zero, such as 10^{-6} (e.g. in case of weights of MobileNetV2 (Sandler et al. [2018]) trained on ImageNet dataset (Deng et al. [2009]), as shown in Fig. 3.1). This indicates, that those layers are not learning useful features. Moreover, as all the weights in a particular layer become extremely close to each other, the standard neural network training methods (based on stochastic gradient descent) become unable to cause the weights to diverge, as they push all the weights in the same direction. Thus, if a network during training lands in a state where weights in some layers are extremely close to each other, it tends to stay in such state. My method was conceived as a remedy to that problem. It is designed to enable the standard, SGD-based neural network optimizers to make the weights in non-performing layers diverge and learn useful features.

I achieve it by repeatedly adding normally distributed, zero-mean noise to model weights mid-training. I chose the normally distributed noise since it is a symmetrical distribution, and I have no reason to treat perturbing weights in the positive direction differently than perturbing them in the negative direc-

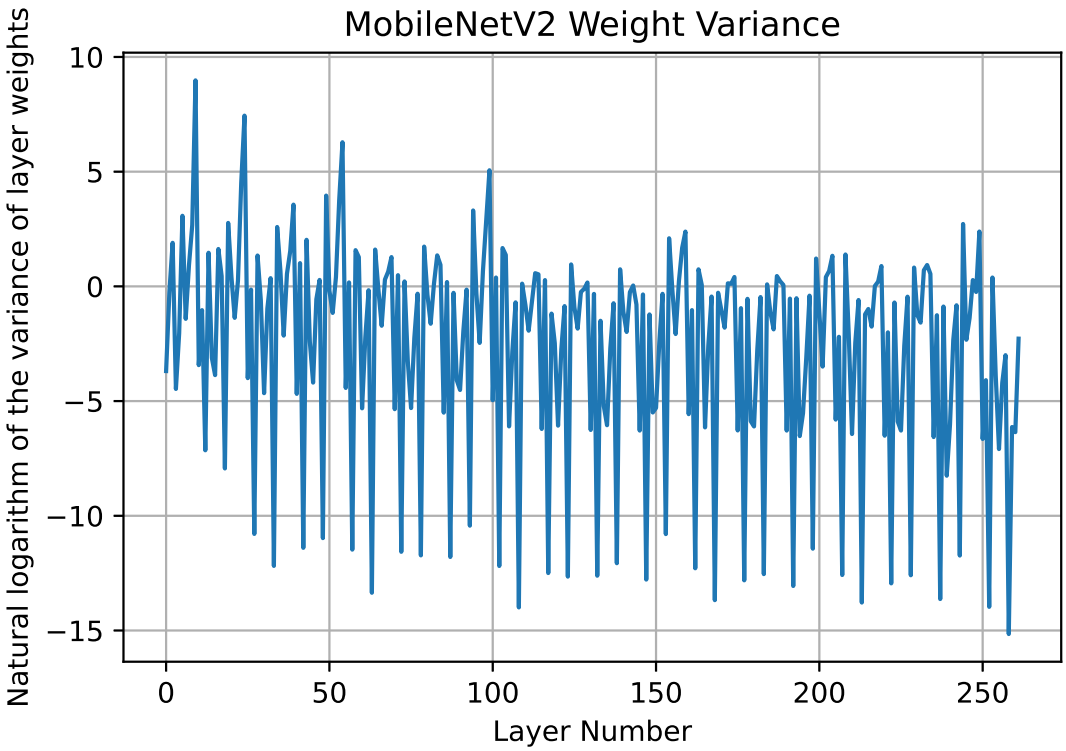


Figure 3.1. The natural logarithm of the variance of weights in different layers of MobileNetV2 network trained on ImageNet. Please note, that some layers have weight variance very close to zero

tion. It is also crucial, that the expected value of the distribution is equal to zero - this way the cumulative effect of applying the noise scales with the square root of the number of times of application, rather than linearly. The standard deviation of the additive noise is chosen based on the variance of weights in the layers of the trained network before the noise injection. In the experiments made for this chapter, I used standard deviations equal to 0.01 and 0.05 of the 25th percentile of weight standard deviations within each layer of the network.

The high-level algorithm for training a neural network using weight perturbations is shown as Algorithm 4:

Algorithm 4 Weight Perturbation Training

1: Train the network to convergence in a standard way.
2: **for** $iteration = 1, 2, \ldots$ **do**
3: Add zero-centered, normally distributed noise to network weights.
4: Continue training the network until validation loss is at the same level as before the noise injection.
5: **end for**
6: Train the network until convergence.

3.4 Why Weight Perturbation Might Work

The main conjecture behind the weight perturbation method is that the weights in a particular layer of the network must be variable enough for the gradient-descent-based optimizers to work properly and for the network to learn useful features. In order to test this conjecture, I defined a toy problem: Let there be a neural network, with 2 input neurons, 1 layer with some number of hidden neurons and a single output neuron. The inputs at the input neurons are each either 0 or 1. The network is tasked with learning the exclusive or (XOR) gate, i.e. the network should output 1 if the inputs are different from each other and 0 if they are equal. I chose the XOR gate, since the XOR problem is not linearly separable (so it could not be solved by a linear model or without the hidden neurons). Please note, that the best accuracy that can be achieved in a XOR-approximating neural network without any hidden neurons is 75% (e.g. by implementing it as an OR gate). The 75% accuracy seems to be a baseline which was achieved or surpassed in all experiments.

I trained this simple network, while varying the number of hidden neurons (from 2 to 128), and varying the standard deviation of the starting weights (from 10^{-20} to 1). In each experiment the training was performed for 30 epochs (each with 10 batches), using Adam optimizer, batches of 400 samples and learning rate 10^{-3}. Samples were equally distributed between (0,0), (0,1), (1,0) and (1,1). For each set of number of hidden neurons and starting weights standard deviations the experiment was repeated 100 times.

In Figs. 3.3-3.8 I show the mean results of those experiments for different lengths of training (from 5 to 30 epochs). In the plots on the left, I present mean accuracy as a function of initial weights standard deviation (the higher, the better) and in the plots on the right I present cross entropy as a function of initial weights standard deviation (the lower, the better). When looking at the plots, one can draw several conclusions:

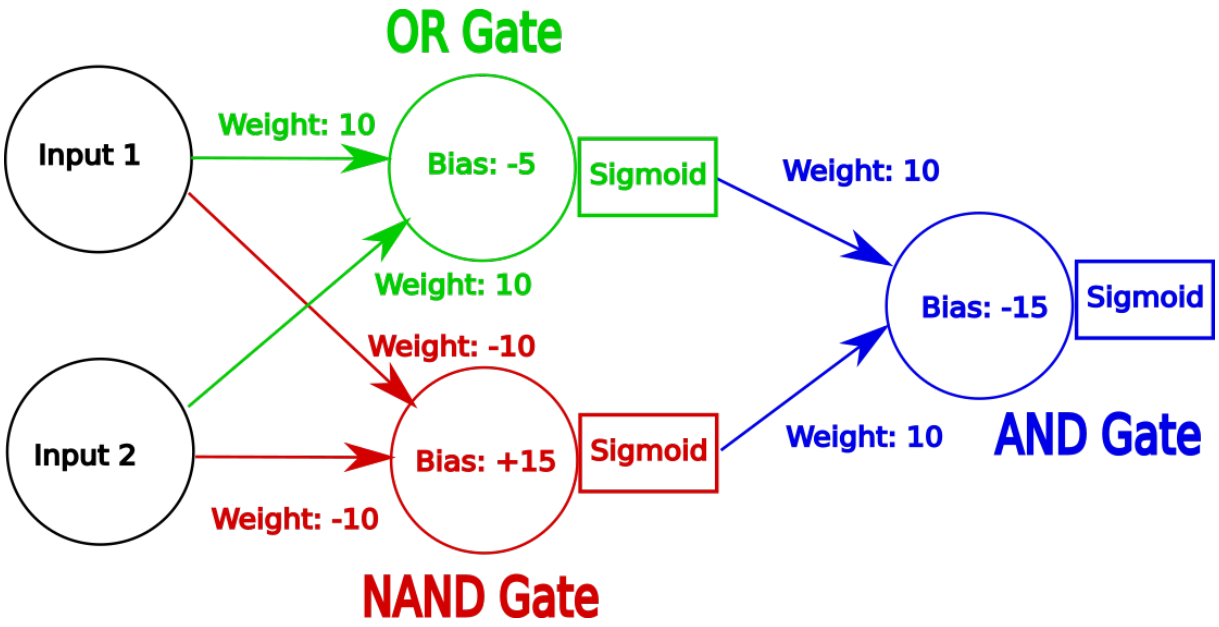


Figure 3.2. XOR gate implemented as a 2 layer neural network consisting of OR, NAND and AND gates

a The results confirm my initial conjecture - if the weights are too similar to each other, the network fails to learn useful feature representations and consequently fails to learn the XOR gate.

b For each network architecture, there seems to be a minimal level of diversity between weights, necessary for the SGD-based optimizer to perform well, i.e. achieve mean accuracy meaningfully higher than 75% baseline (e.g. for the network with 32 hidden neurons the minimal standard deviation appears to be approximately 10^{-12}, while for 2 hidden neurons network it seems to be between 10^{-4} and 10^{-3}).

c This minimal weight standard deviation threshold is stable between different lengths of training, it shows that if the weights are too similar to each other, the SGD-based optimizers seem not only to work slow, but rather not work properly at all.

d The threshold for weights standard deviation enabling SGD-based optimizer to work properly is smaller for bigger networks (i.e. it is possible to train relatively big neural networks even if weights are very similar to each other). This result is consistent with the practice of using large neural networks.

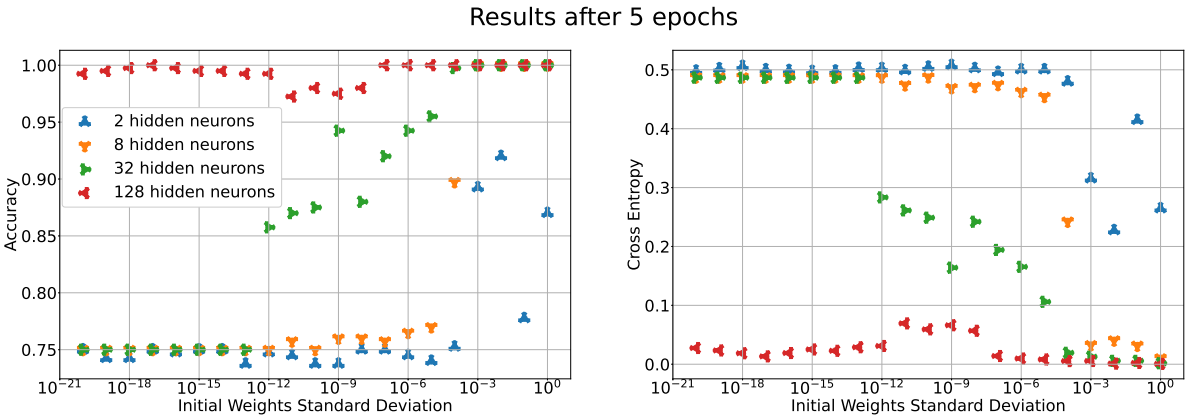


Figure 3.3. Accuracy and cross-entropy after training the neural network to imitate a XOR gate for 5 epochs, presented for different initial weights standard deviations and different numbers of hidden neurons.

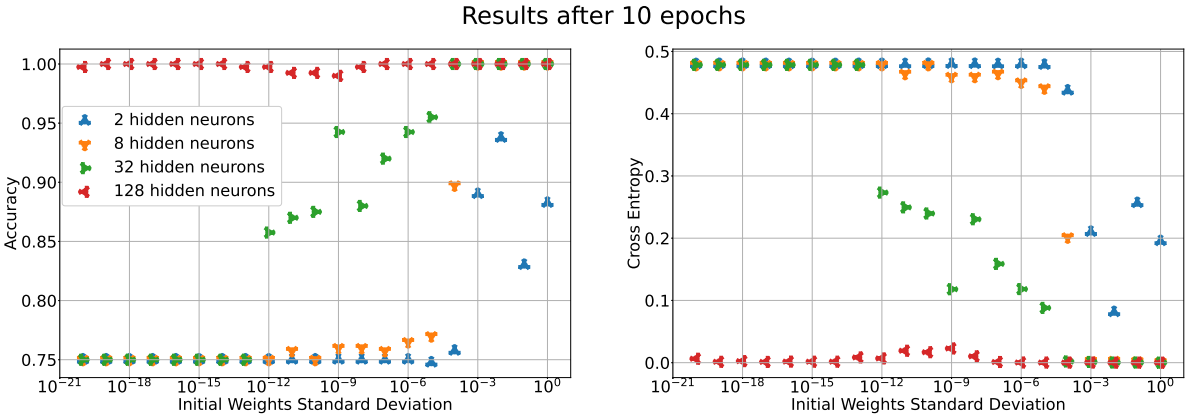


Figure 3.4. Accuracy and cross-entropy after training the neural network to imitate a XOR gate for 10 epochs, presented for different initial weights standard deviations and different numbers of hidden neurons.

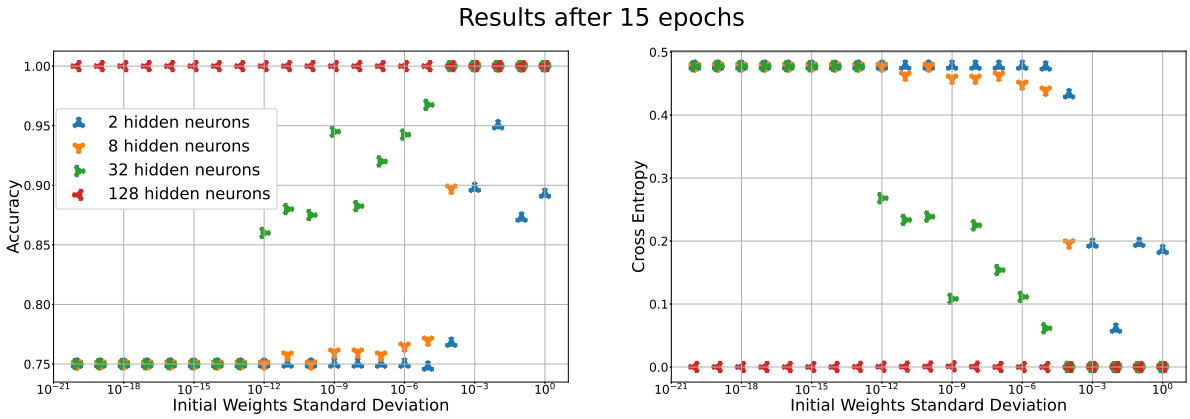


Figure 3.5. Accuracy and cross-entropy after training the neural network to imitate a XOR gate for 15 epochs, presented for different initial weights standard deviations and different numbers of hidden neurons.

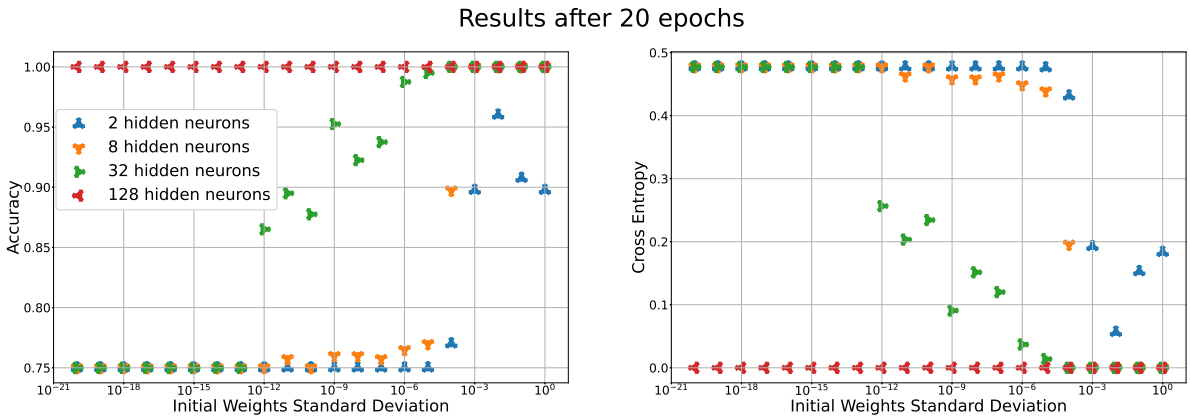


Figure 3.6. Accuracy and cross-entropy after training the neural network to imitate a XOR gate for 20 epochs, presented for different initial weights standard deviations and different numbers of hidden neurons.

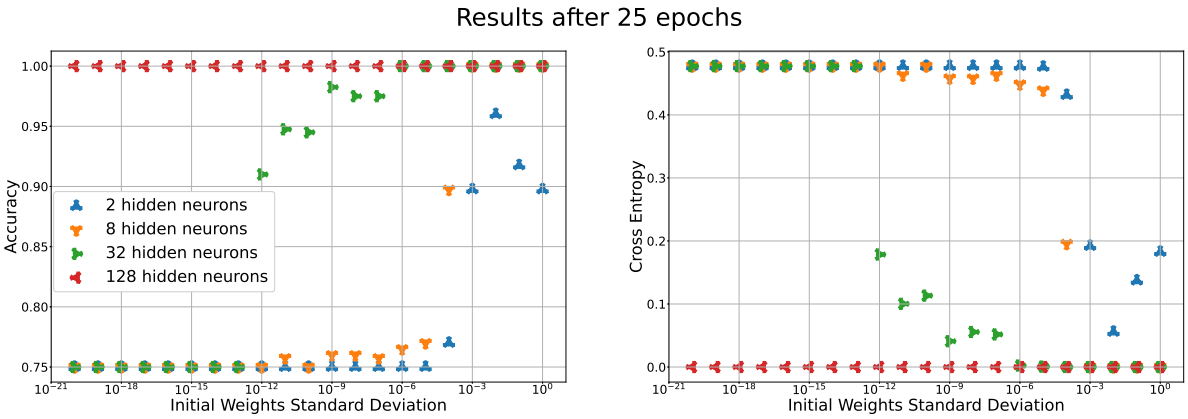


Figure 3.7. Accuracy and cross-entropy after training the neural network to imitate a XOR gate for 25 epochs, presented for different initial weights standard deviations and different numbers of hidden neurons.

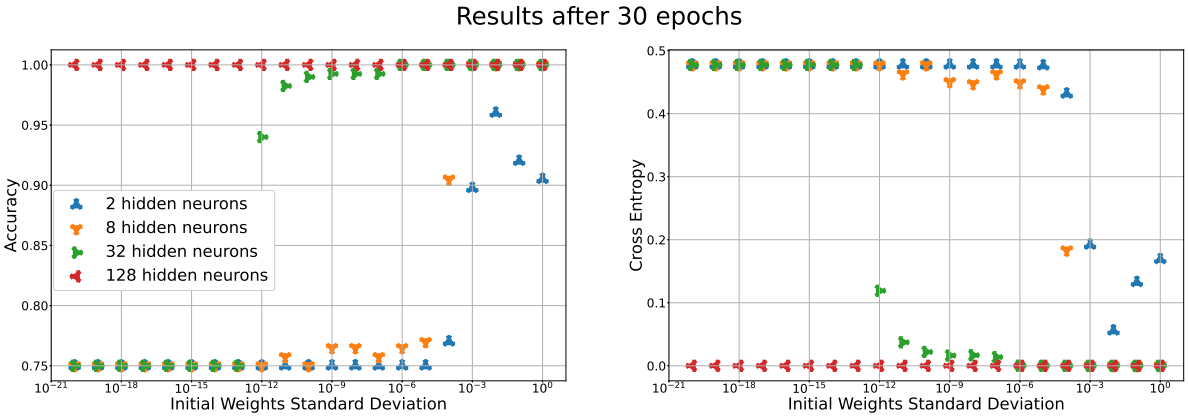


Figure 3.8. Accuracy and cross-entropy after training the neural network to imitate a XOR gate for 30 epochs, presented for different initial weights standard deviations and different numbers of hidden neurons.

3.5 Experiments

3.5.1 Experimental Setup

The goal of my experiments was to measure the influence of weight perturbations on neural network performance across a range of network architectures and compare the results that are obtained using weight perturbations to the results obtained in a vanilla training process and a training process involving a lottery ticket pruning procedure described in Section 3.2.3. To that end, I performed experiments on five neural network architectures:

- MobilenetV2 (Sandler et al. [2018]),
- DenseNet201 (Huang et al. [2017]),
- ResNet152V2 (He et al. [2016]),
- Xception (Chollet [2017]),
- EfficientNet B0 (Tan and Le [2019]).

For each of the above-mentioned networks, I used their Keras (Chollet et al. [2015]) implementation, with the pretrained ImageNet (Deng et al. [2009]) weights. I trained and tested the aforementioned networks on the Cifar100 dataset (Krizhevsky et al. [2009]). In case of all networks I removed the first 2 strides, since the CIFAR100 (Krizhevsky et al. [2009]) images are much smaller (32x32 px) than the ImageNet (Deng et al. [2009]) images (for most networks trained on ImageNet, the network assumes the input to have size 224x224 px). During training I used mixup (Zhang et al. [2017]) regularization, which feeds to the network a linear combination of inputs, and makes the label used while training a linear combination of input labels. All the training was performed with cross entropy as the loss function.

For each of the networks I tried 4 different training procedures:

- Vanilla training,
- Training utilizing weight perturbations,
- Training utilizing lottery ticket,
- Training utilizing weight perturbations and lottery ticket.

The **vanilla training** was performed in the following way:

1. 200 epochs training with high learning rate,
2. 100 epochs training with 10 times smaller learning rate.

The **training using weight perturbation** was performed in the following way:

1. 150 epochs training with high learning rate,
2. 50 epochs training with high learning rate, with additive noise injection every 10 epochs,
3. 100 epochs training with 10 times smaller learning rate.

The **training using lottery ticket** was performed in the following way:

1. 10 epochs of training with high learning rate during which lottery ticket is applied (finding and pretraining subnetwork),
2. 190 epochs training with high learning rate,
3. 100 epochs training with 10 times smaller learning rate.

The **training using lottery ticket and weight perturbation** was performed in the following way:

1. 10 epochs of training with high learning rate during which lottery ticket is applied (finding and pretraining subnetwork),
2. 140 epochs training with high learning rate,
3. 50 epochs training with high learning rate, with additive noise injection every 10 epochs,
4. 100 epochs training with 10 times smaller learning rate.

In the case of the training utilizing the lottery ticket subroutine, for each tested network I performed exactly 5 iterations of pruning, each time pruning 12% of the remaining weights. Only the weights from convolutional and fully-connected layers were pruned. The parameters of batch norm layers were not modified in the pruning process. At the end, I obtained a network in which approximately 48% of the weights are equal to zero.

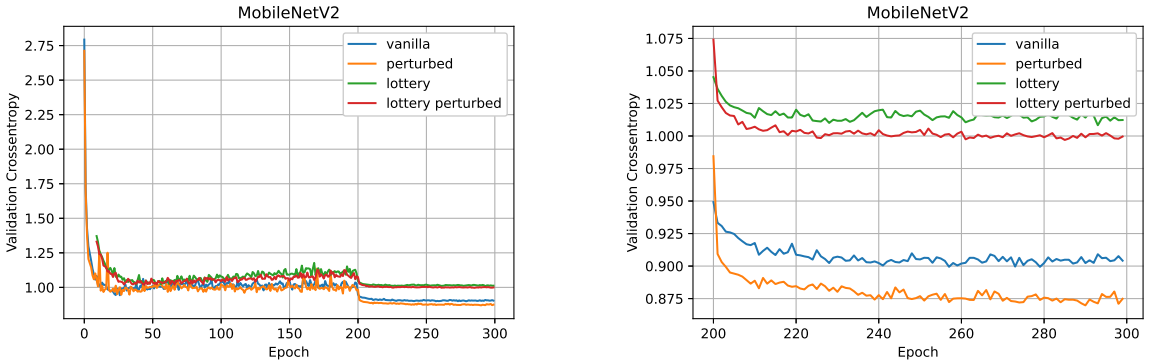
For MobileNetV2 and Xception the starting learning rate was set to 10^{-4}. For DenseNet201, Resnet152V2 and EfficientNet B0 it was set to 10^{-5}.

For MobileNetV2, Xception and EfficientNet B0 the standard deviation of weight perturbation was set to 0.05 of the 25th percentile of weight standard deviations within layers of the network. For ResNet152V2 and DenseNet201 the standard deviation of weight perturbation was set to 0.01 of the 25th percentile of weight standard deviations within layers of the network.

3.5.2 Results

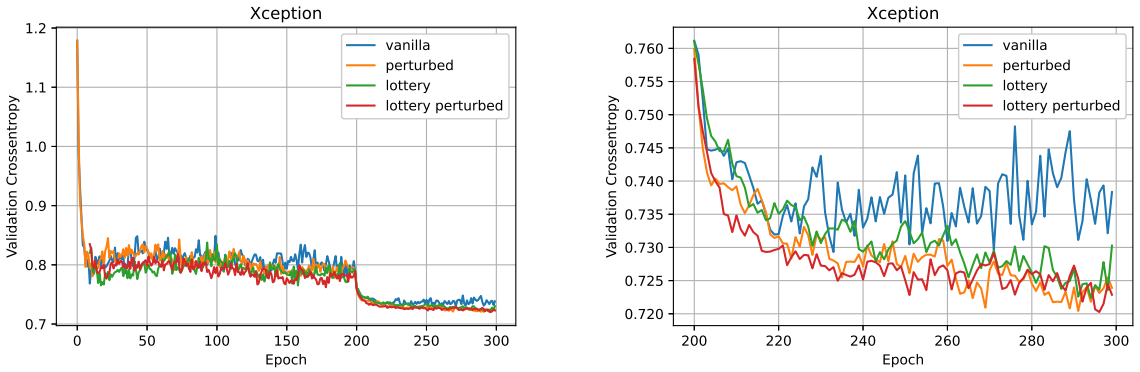
For each of the tested network architectures and training procedures I present a plot showing how the validation cross entropy (the lower the better) changed during training. I chose to present validation cross-entropy, rather than validation accuracy, as the cross entropy is a more robust metric than just accuracy, and consequently enables better comparison of the training procedures. Specifically, the cross entropy metric better captures the intuition that I want the network to be more confident in the classification cases when it correctly classifies a particular image.

For each network architecture I present results at 2 scales for the sake of clarity (one plot shows the validation cross entropy over the course of whole training procedure and the other plot shows validation cross entropy over the last 100 epochs of training - after the learning rate was reduced). In case of the



(a) Whole (b) Zoomed

Figure 3.9. MobileNetV2 validation cross-entropy for different training procedures (different scales)

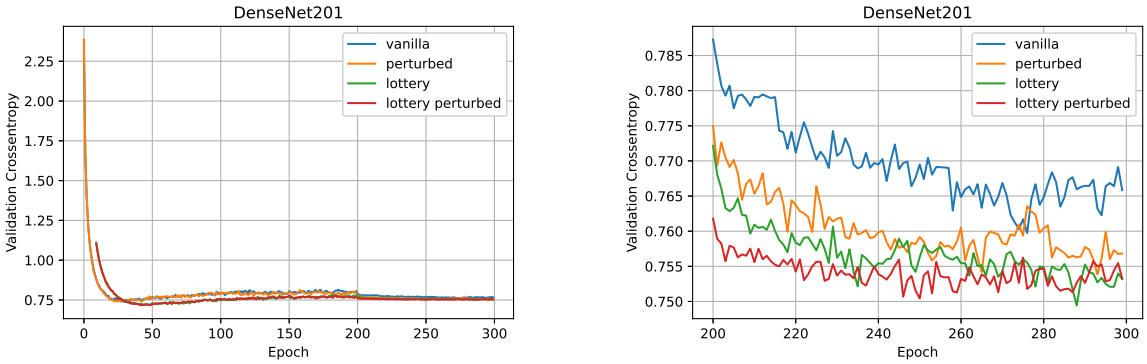


(a) Whole (b) Zoomed

Figure 3.10. Xception validation cross-entropy for different training procedures (different scales)

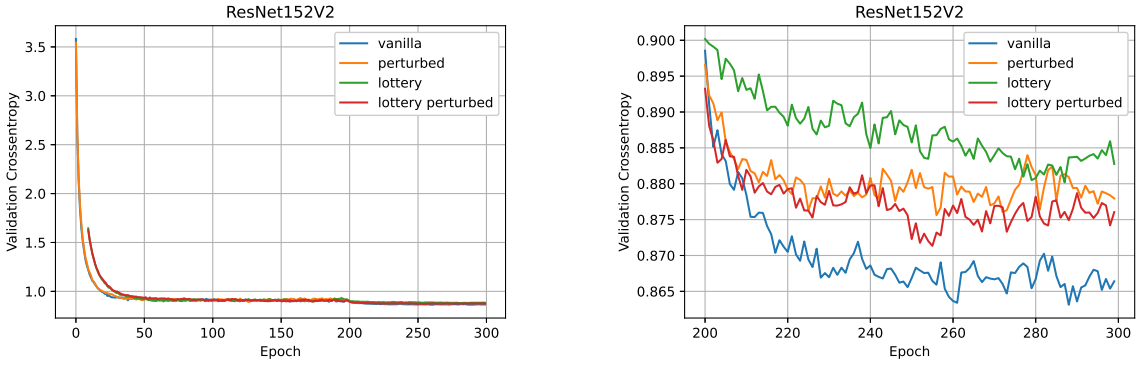
trainings utilizing the lottery ticket subroutine I start plotting the validation cross entropy only after the first 10 epochs since the weights are reset several times in the first training epochs as a part of the subroutine to find the optimal subnetwork.

As can be seen in Fig. 3.9 the additive weight perturbations work well with the MobileNetV2 (Sandler et al. [2018]) architecture. The training with perturbations achieved validation loss reduction by more than 2%, when compared to the vanilla training. Please note, that the network performance is increased both when perturbations are applied to the vanilla network structure and the lottery-ticket-determined subnetwork.



(a) Whole (b) Zoomed

Figure 3.11. DenseNet201 validation cross-entropy for different training procedures (different scales)



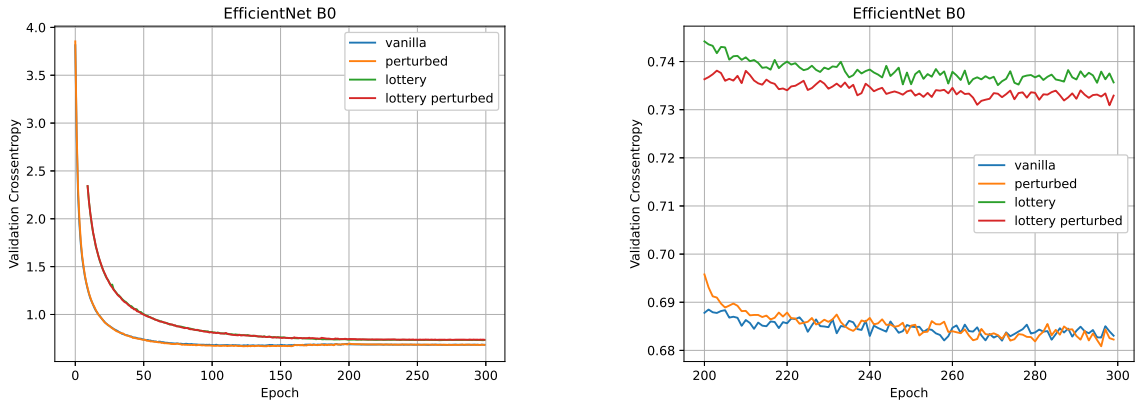
(a) Whole (b) Zoomed

Figure 3.12. ResNet152V2 validation cross-entropy for different training procedures (different scales)

In the case of the Xception (Chollet [2017]) network (Fig. 3.10), weight perturbation shows clear improvement over the vanilla training and does not affect the lottery-ticket-determined subnetwork performance in any significant way.

For the DenseNet201 (Huang et al. [2017]) network, weight perturbations show improvement over the vanilla training, and might cause slightly faster convergence for the lottery-ticket-determined subnetwork.

For the ResNet152V2 (He et al. [2016]) additive weight perturbations failed to improve network performance over the vanilla training procedure, but the training procedure where both the lottery ticket and perturbations were applied achieved better results than the lottery ticket alone.



(a) Whole

(b) Zoomed

Figure 3.13. EfficientNet B0 validation cross-entropy for different training procedures (different scales)

For the EfficientNet B0 (Tan and Le [2019]) additive weight perturbations failed to improve network performance when compared with the vanilla training, however, weight perturbations improved the performance of the network to which the lottery ticket subroutine was applied.

3.6 Conclusions

Using the toy example of a neural network approximating a XOR gate I have shown that SGD-based optimizers are unlikely to achieve good results if the weights in a particular layer are too similar to each other. This effect is more pronounced in smaller neural networks, but does not disappear in the larger versions. Moreover, the effect appears to be stable, regardless of the length of training (so it does not only slow training down, but rather makes it impossible in some cases).

To combat this effect I proposed to apply noise to network weights during training. I have shown that using a training procedure utilizing additive weight perturbation causes visible performance improvement in some of the tested network architectures (MobileNetV2, DenseNet and Xception), when compared to the vanilla training procedure. The results are particularly significant for MobileNetV2, with weight perturbations clearly improving the performance of both the standard network and a network transformed according to the lottery ticket training subroutine.

The weight perturbation procedure is easily adaptable to the training of any neural network and consequently provides a potentially useful addition to a deep learning practitioner's toolbox. The place of the weight perturbation procedure in such toolbox is comparable with the place of the lottery ticket subroutine. While for some network architectures, the use of lottery tickets brings significant improvements, it is conjectured not to bring performance gains (Frankle and Carbin [2018]) in other architectures. My experiments show the same to be true for weight perturbations (it improved on vanilla training for MobileNetV2, DenseNet and Xception - 3 out of 5 cases). It is also worth noting, that training with weight perturbations works very well in tandem with the lottery ticket subroutine (it causes very clear improvement over the lottery ticket alone when applied to MobileNetV2, ResNet152V2 and EfficientNet B0 - 3 out of 5 cases, in the remaining 2 cases it does not cause the performance of the network to drop).

I used the method of training described in this chapter to train WeaveNet (described in Chapter 4) and to train the neural networks used in the angle finding experiments (described in Chapter 5).

This chapter has dealt with showing the usefulness of weight perturbation in CNN training. Given the popularity and high training and inference cost of the transformer-style (Vaswani et al. [2017]) networks (e.g. Devlin et al. [2018], Lewis et al. [2019], Raffel et al. [2020], Brown et al. [2020], OpenAI [2023], Touvron et al. [2023a], Touvron et al. [2023b], Geng and Liu [2023] or Biderman et al. [2023]), it might be beneficial to test weight perturbationas a tool for their pretraining or finetuning.

4 Lidar Depth Completion - WeaveNet

4.1 Introduction

As I stated in Section 1.4 the lidar output is not dense, but only semi-dense. While building the radar depth completion dataset (described in Chapter 6) I needed dense depth maps, but I couldn't use off-the-shelf lidar depth completion solutions, as the problem I was facing was quite unique. I needed to create a dense depth map from the pointcloud from a Pandora lidar (different from the Velodyne used in the KITTI dataset) projected on a plane located closer to the grille (radar location) than the lidar itself. Sensor locations are shown in Chapter 6. It results in an unequal distribution of the measurements in the depth completion plane. This chapter describes my work on creating a lidar depth completion solution that is indifferent to the distribution of lidar depth measurements, so that in Chapter 6 I could use it to create a dataset to train the radar depth completion model. I present the results for two versions of the network - unguided (lidar-only) and guided (lidar+camera).

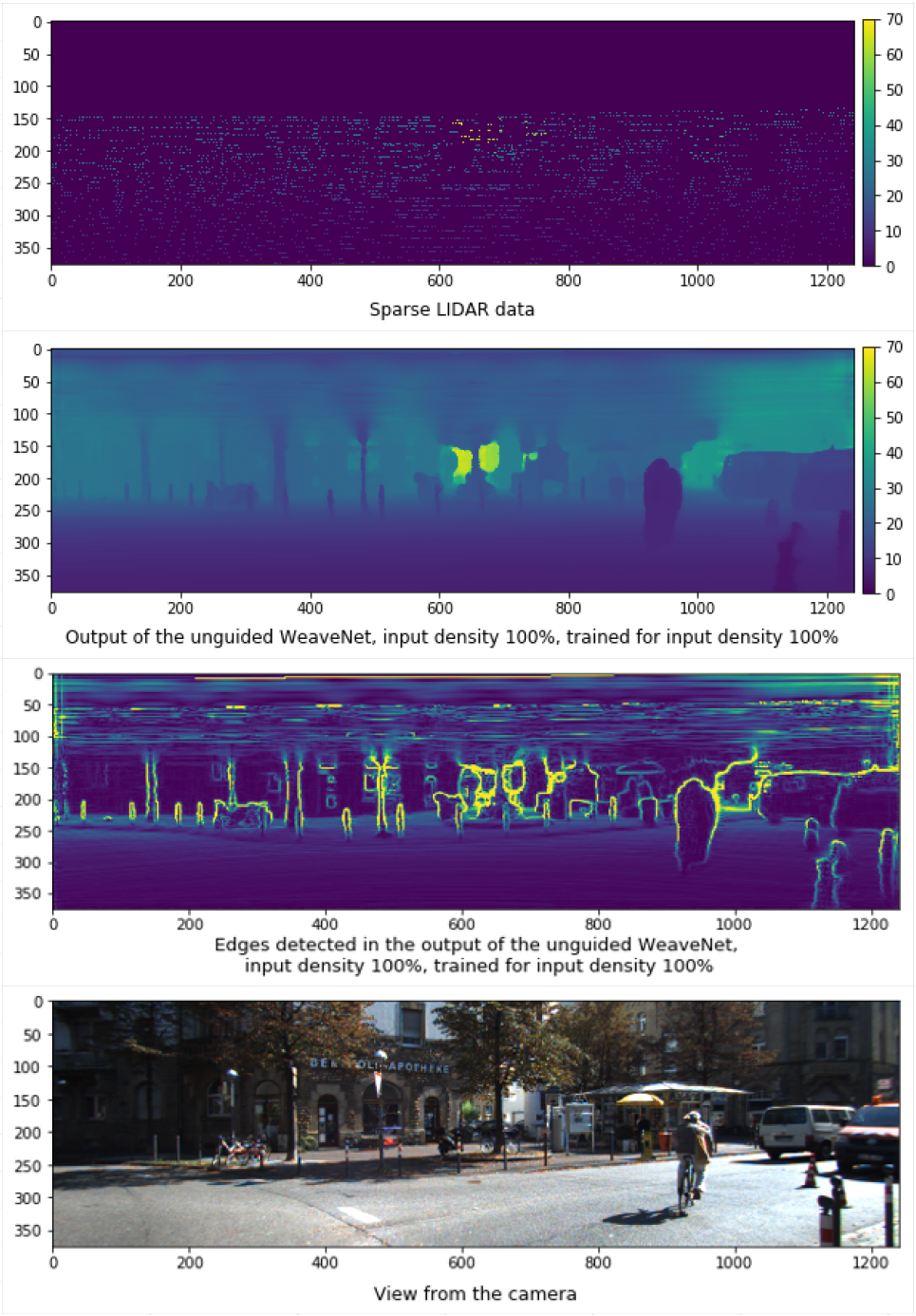


Figure 4.1. The output of the unguided WeaveNet together with the sparse lidar data and corresponding camera image. Training for input density 100% refers to training without masking any lidar depth measurements. The depth is color coded, with the scale shown to the right of the relevant images. The edges in the output of the network are shown to demonstrate the capability of network to accurately represent fine details (they were found using a version of the Sobel filter (Kanopoulos et al. [1988])). The same frame is used as the basis of all future images in this paper. Numbers on the axes denote pixel coordinates.

4.2 Related Work

4.2.1 Methods Based on Sparsity Invariant Convolutions

Uhrig et al. [2017] published the KITTI depth completion dataset together with the definition of the sparsity invariant convolutions. The point behind the KITTI depth completion dataset is to provide a way to train and test models capable of transforming sparse depth measurements into a dense depth map. This dataset consists of 93k frames, where each frame consists of a camera image, sparse depth data obtained by projecting the lidar pointcloud onto the camera plane and a semi-dense depth annotation on the camera plane. The semi-dense depth annotations were produced by accumulating 11 lidar scans and subsequently removing outliers (such as the outliers related to moving objects) by comparing the accumulated lidar data points to the depth map obtained using stereo vision. The sparsity invariant convolutions (Uhrig et al. [2017]) are a method to perform 2D convolutions on an input with variable sparsity. This is achieved by utilizing a mask which contains ones for valid input pixels (e.g. pixel where a lidar point is projected) and zeros for invalid pixels. Before performing a convolution the input is multiplied (pixel-wise) by the validity mask. After performing the convolution (but before adding bias) the output is normalized by dividing it by the fraction of the valid pixels that were used to compute it. The mask is propagated in such a way, that a pixel is considered valid if and only if at least one valid pixel was used to compute it. Using the notation from Uhrig et al. [2017], the result of the convolution with kernel $(2k+1, 2k+1)$ at the position (u, v) is:

$$f_{u,v}\left(\mathbf{x}, \mathbf{o}\right) = \frac{\sum_{i,j=-k}^{k} o_{u+i,v+j} x_{u+i,v+j} w_{i,j}}{\sum_{i,j=-k}^{k} o_{u+i,v+j} + \epsilon} + b, \tag{4.2.1}$$

and the value of the validity mask is propagated according to:

$$g_{u,v}\left(\mathbf{x}, \mathbf{o}\right) = \max_{i,j=-k}^{k} o_{u+i,v+j}, \tag{4.2.2}$$

where **x** denotes the array of pixel values at the previous layer, **o** denotes the array storing information on whether the pixel was valid in the previous layer (1 if it was valid, 0 if it was not), **w** denotes the array storing information about convolutional weights and b is the bias.

Huang et al. [2019b] perform the task of depth completion using an encoder-decoder, hourglass-style network utilizing sparsity invariant convolutions. To that end, they introduce sparsity invariant averaging and sparsity invariant upsampling. These are operations that enable the network to perform averaging and upsampling in a manner consistent with the sparsity invariant framework (and define the way in which validity masks should be propagated).

Eldesokey et al. [2018] significantly expand on the sparsity invariant convolution framework by lifting the requirement that the validity mask is binary. Instead, they allow the confidence mask to take

values between 0 and 1. Additionally, they restrict the convolutional weights to be non-negative and interpret them as the affinity between different pixels, hence the propagated depth for a specific pixel is an average of depths from neighboring pixels weighted by their affinities. They propagate the confidence for a specific pixel making it an average of the confidences of neighboring pixels weighted by their respective affinities. Eldesokey et al. [2018] found it necessary to use a loss function which aims to maximize confidence as part of the training procedure.

Yan et al. [2020] utilize in their work an encoder-decoder hourglass-style network to produce dense depth output. They use separate, yet identical encoders for image data and lidar data. Interestingly, both lidar encoder and the image encoder use sparsity invariant convolutions together with masks generated using lidar datapoints (albeit the mask for the image encoder switches each zero to one and each one to zero). Yan et al. [2020] provide an ablation analysis of the performance of the network when it is fed with data sparser than usual, however, they decrease the data density at most 4 times.

4.2.2 Methods Not Based on Sparsity Invariant Convolutions

A big part of the best-performing methods for guided depth completion borrow heavily from the Convolutional Spatial Propagation Network (CSPN) by Cheng et al. [2019b], which was itself influenced by the non-depth completion work of Liu et al. [2017]. Cheng's neural network consists of a conventional encoder-decoder network and an iterative depth propagation layer. The output of the encoder-decoder network are initial depth predictions, and affinities to other pixels. The affinities between pixels are used as convolutional weights to propagate depth information. The values of affinities used when propagating depth information to a particular pixel are normalized, so that the sum of their absolute values equals 1. The depth propagation step is performed iteratively. In another paper Cheng et al. [2019a] define a CSPN++ block which improves upon CSPN by making the number of propagation step iterations and size of the kernel used during the propagation step dependent on the input (in a learnable way).

The work of Park et al. [2020] uses very similar approach, but introduces pixel-wise confidence and relaxes constraints on affinity normalization. In the case of Park's network, the output of the encoder-decoder network are initial depth prediction, affinities to other pixels and additionally pixel-wise confidence. The iterative depth propagation layer performs depth propagation by using affinities and pixel-wise confidence. The affinities between pixels multiplied by pixel-wise confidence are used as convolutional weights to propagate depth information. The sum of absolute values of affinities used when propagating depth information to a particular pixel is constrained to be no greater than 1. Unlike in the work of Eldesokey et al. [2018] Park does not use confidence when calculating the loss function.

The method developed by Tang et al. [2019] focuses on effectively utilizing the RGB input to propagate the sparse depth information. To that end, the encoder part of their network consists of the vision encoder, which outputs convolutional weights that are later used by the sparse depth encoder.

The work of Chen et al. [2019] shows a different path to depth completion. Their network utilizes 2 processing streams - a 2D stream in which lidar and RGB data are processed using conventional 2D convolution and a 3D processing stream in which they utilize 3D continuous convolutions (Wang et al. [2018]). As Chen et al. [2019] point out in their work, the lidar points close together on the 2D projection can be far away in 3D space, therefore utilizing both 2D and 3D processing allows for a much better feature extraction process.

Qiu et al. [2019] introduce a network that utilizes a conventional processing stream, that takes sparse depth and image inputs and directly outputs the dense depth information and an alternative stream that accepts the same inputs but produces surface normals as an intermediate step towards outputting depth prediction. The final depth prediction is a learnable weighted average of the previous 2 outputs. An interesting part of the work performed by Qiu et al. [2019] is that they used a synthetic dataset obtained using the Carla simulator (Dosovitskiy et al. [2017]) to pretrain the network on the task of surface normal prediction.

The work of Ma et al. [2018] uses a conventional encoder-decoder hourglass-style network (the depth and image data are fused at the beginning of the encoder). The most important contribution of this work is introducing a self-supervised way to train a depth completion network. Ma et al. [2018] introduce a training process utilizing spatiotemporal properties of consecutive lidar and image frames to train the network without the semi-dense annotations. Specifically, they utilize a loss function taking into account depth prediction consistency with depth measurement at a specific pixel, photometric loss connected to image warping between consecutive frames, and a smoothness term that penalizes second-order derivatives in the depth prediction. Notably, Ma et al. provide an ablation analysis of the performance of the network when it is fed with data sparser than usual.

4.3 My Method

4.3.1 WeaveBlock

The main idea behind sparsity invariant convolutions (Uhrig et al. [2017]) was that the network can produce a valid depth output for a particular picture only if at least one valid depth input pixel was used in the calculations. It follows, that in order to process very sparse depth input it is necessary to allow easy flow of information between pixels located far away from each other. In my network, this

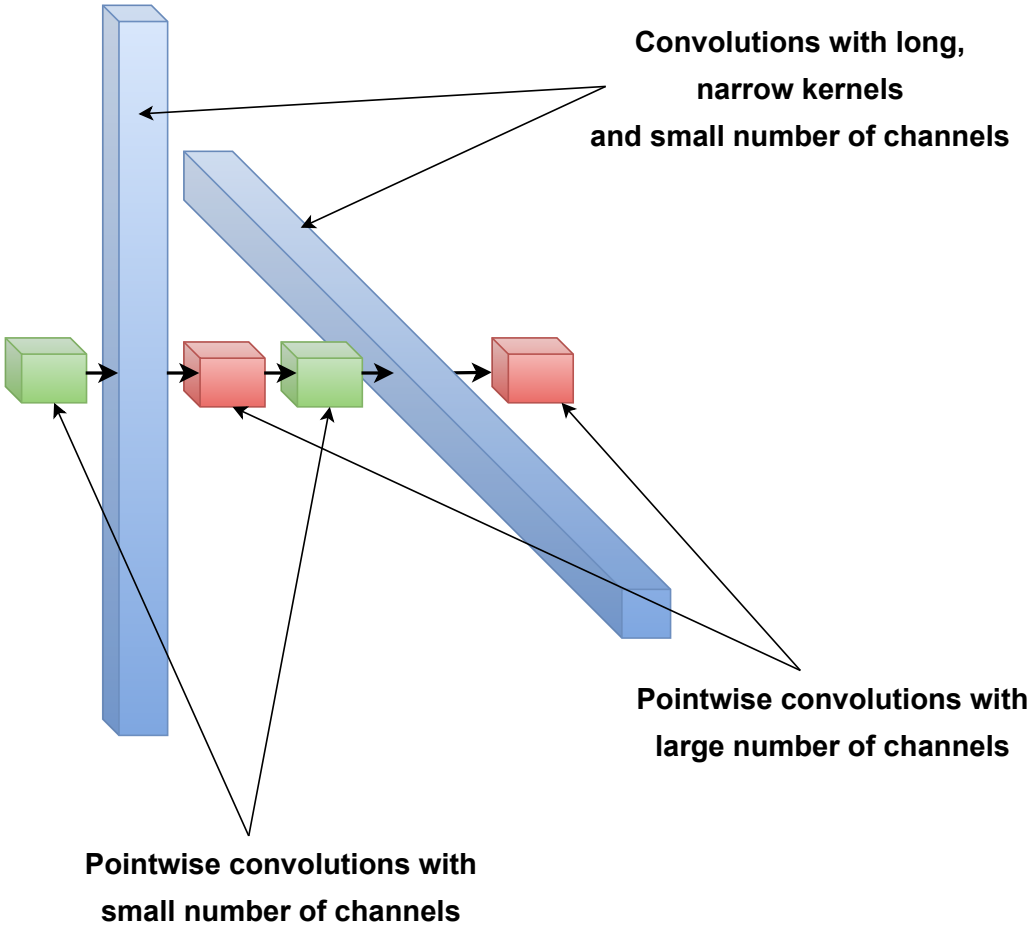


Figure 4.2. A schematic representation of the WeaveBlock

is achieved by utilizing WeaveBlocks. The main inspirations for the WeaveBlock design were sparsity invariant convolutions (Uhrig et al. [2017]), and general structure loosely inspired by the depthwise separable convolutions in MobileNet (Howard et al. [2017]). Each WeaveBlock (schematically shown in Figure 4.2) consists of the following elements:

1. Pointwise convolution with a small number of channels,
2. Sparsity invariant convolution with long and narrow vertical kernel and a small number of channels,
3. Pointwise convolution with a large number of channels,
4. Pointwise convolution with a small number of channels,
5. Sparsity invariant convolution with long and narrow horizontal kernel and a small number of channels,
6. Pointwise convolution with a large number of channels.

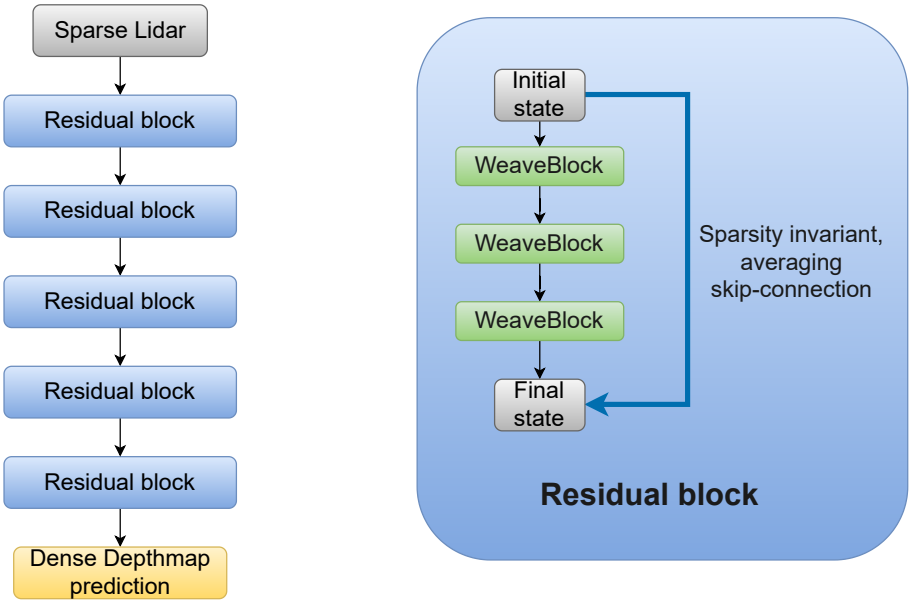


Figure 4.3. A schematic representation of the unguided WeaveNet architecture on the left, and a close-up on a single residual block on the right.

All mentioned convolutions use relu (Xu et al. [2015]) activation. In the case of the network whose results are presented in this chapter, the small number of channels was set to 12, large number of channels was set to 128, and the long and narrow convolutional kernels were in the shapes 31x1 and 1x31. The design of WeaveBlock achieves the goal of propagating the depth information at a big distance (15 pixels in each direction). Additionally it enables the network to perform more complicated computations utilizing 128 channels pointwise convolutions, while also being reasonably frugal with resources - please note that when computations are performed in kernels bigger than 1x1, the number of channels in the current layer and the number of channels in the preceding layer is small.

4.3.2 Unguided WeaveNet Architecture

The unguided version of the WeaveNet architecture (shown in Figure 4.3) follows a simple, modular structure. Inside the network, WeaveBlocks are grouped in groups of 3, forming a single residual block. The residual blocks are wrapped in an averaging skip connection, implemented similarly as in Huang's sparsity invariant averaging (Huang et al. [2019b]). The network consists of 5 identical residual blocks, without any upsampling or downsampling - the network processing always happens at input and output resolution. The results obtained using the unguided WeaveNet, together with the corresponding lidar input and camera image are presented in Figure 4.1. In the same figure, the edges in the output of the network are also shown to demonstrate the capability of the network to accurately represent fine details.

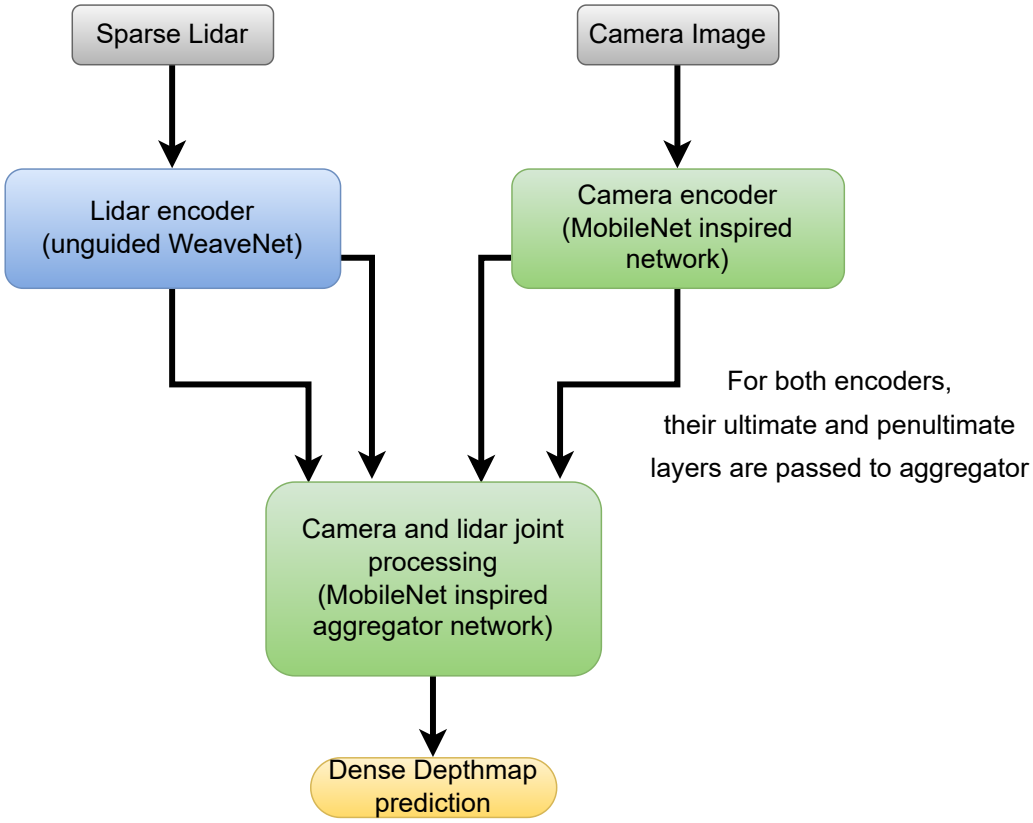


Figure 4.4. A schematic representation of the guided WeaveNet architecture.

The unguided version of the network is the main part of this work, the guided version of the network does not contain significant improvements of its own and was prepared only to make the results obtained using WeaveNet comparable to other, guided solutions used in the KITTI depth completion task.

4.3.3 Guided WeaveNet Architecture

The RGB-guided version of the network (shown in Figure 4.4) utilizes the unguided architecture to preprocess the lidar depth information. The RGB data are preprocessed using MobileNet (Howard et al. [2017]) inspired convolutional network, where a single functional convolution consists of:

1. Pointwise convolution with a small number of channels,
2. 3x3 convolution with a small number of channels,
3. Pointwise convolution with a large number of channels.

All mentioned convolutions use relu activation. Small number of channels for this subnetwork was set to 16 and large number of channels was set to 128. A set of 3 such convolutions makes one residual block. At the beginning of a residual block, there is also an additional pointwise convolution with small number of channels. There are also 2 batch normalizing layers (after the first pointwise convolution and at the

end of the residual block). The skip connection is implemented via concatenation of the input taken from the batch normalization layers. Please note, that the concatenation input consists of 16 channels from the front of the block and 128 channels from the end of the block. I believe that such network structure forces the model to compress the already learned features to a small channel space, while allowing for processing new features in a larger channel space. The vision preprocessing network consists of 3 such residual blocks and ends with a 1 channel convolution.

The preprocessed data from the lidar and camera are concatenated - for both subnetworks their ultimate and penultimate layer outputs are concatenated. Please note this concatenation uses much more channels from the camera preprocessing (129 channels) than from lidar preprocessing (13 channels) - I found it necessary for the network to extract useful camera features.

The fused data is processed by an aggregator network which is nearly identical to the vision preprocessing network - the only difference is that the aggregator network does not include batch normalization layers.

4.3.4 Notes on the Input and Output Modality

An important feature of unguided depth completion is that the inputs and outputs have the same modality - it is the distance to the nearest obstacle projected on the camera plane. Therefore it is possible that inside the network the information is also processed in the form of depth information, rather than abstract channels. In such a case, it would be possible that the network weights are just a learnable way to interpolate between points. In fact, the good results in Eldesokey's work on propagating confidence information in Convolutional Neural Networks (CNNs) (Eldesokey et al. [2018]) may suggest that. Their method of depth completion achieves competitive results on the KITTI depth completion challenge while effectively restricting the convolutional weights to be nonnegative. During the work on designing my depth completion network, I found several indications consistent with the possibility that the channels in the middle of the network are not abstract, but rather consistently represent depth.

a I found that using batch normalization in the lidar processing stream leads to much inferior results.

b I performed experiments using the same network architecture and activation functions other than relu. I found that using activation functions whose output may be negative such as swish (Ramachandran et al. [2017]) and elu (Clevert et al. [2016]) leads to worse performance in the depth completion task than simple relu.

Such results are consistent with the possibility that at least some of the channels in intermediate layers of the network are directly representing depth. In fact, when examining the output of each residual block I found that there are only 2 types of channels - channels whose output represents depth (up to an affine

transformation) and channels whose output represents offsets at object edges. Example channels are shown in Figure 4.5.

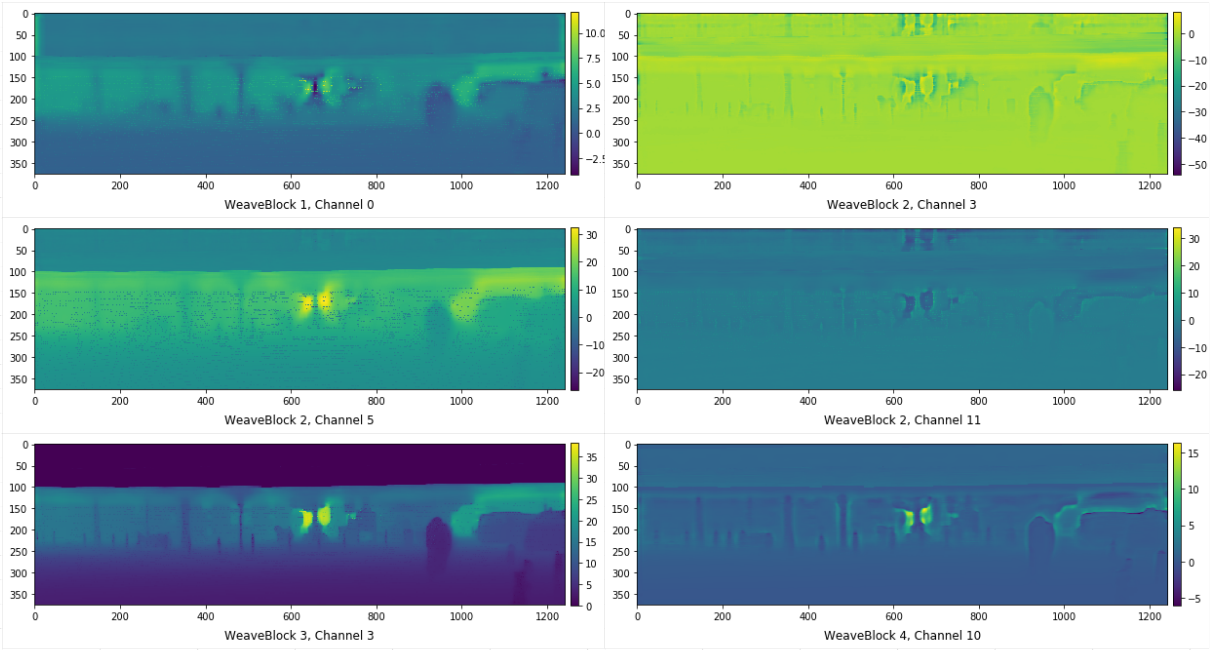


Figure 4.5. Example channels at the intermediate layers of WeaveNet (channels storing depth are on the left and channels storing edges are on the right). Numbers on the axes denote pixel coordinates.

4.3.5 Training Process

In this chapter I present 3 distinct versions of WeaveNet. Architecturally they are the same, but they differ in terms of how they were trained:

- The first version was trained in a standard way (utilizing 2 training phases: pretraining and standard input density training).
- The second version was trained using universal sparsity training, that consisted of 3 training phases: pretraining, standard input density training and a variable input sparsity training (wherever in the text I refer to training with $x\%$ density, I mean training where $x\%$ of the lidar depth measurements were kept).
- The third version was trained in a 4 phase specific sparsity training (pretraining, standard input density training, variable input sparsity training and a final phase of retraining the network at a specific input sparsity).

Full training procedure for the specific sparsity unguided version of the network was conducted in 4 distinct phases:

1. Pretraining phase during which instead of using lidar pointcloud as the input of the network, the network was fed with semi-dense depth labels. To be exact, the network was fed with KITTI Depth Completion labels with 50% of pixels masked with zeros. The pretraining phase was relatively short - only one epoch, utilizing approximately 20% of training set frames (initial experiments had shown that it is a sufficient pretraining length). This phase was performed using 10^{-3} learning rate. I found that this short pretraining phase greatly increases the speed of convergence in the future phases.
2. The main training phase, during which the network was fed standard sparse input. This phase lasted 30 epochs. The optimizer learning rate was decreased from 10^{-3} to 10^{-4} to 10^{-5} to 10^{-6} during this phase. I found it beneficial to apply additive, normally distributed noise to the network weights during this phase of training (approach closer examined in Chapter 3).
3. The variable sparsity training phase. During this phase, the network was fed with input for which between 0% and 99.2% of the standard sparse input was masked with zeros. For a particular frame, the input density (i.e. the number of pixels that were not masked with zeros) was randomly sampled according to a formula:

$$D(D_{\text{min}}, D_{\text{max}}) = \mathrm{e}^{\mathrm{U}(\ln(D_{\text{min}}),\ln(D_{\text{max}}))} \tag{4.3.1}$$

   where D_{min} and D_{max} are the lowest (0.008) and the highest (1.0) possible density levels respectively and $\mathrm{U}(\ln(D_{\text{min}}), \ln(D_{\text{max}}))$ is a random variable sampled uniformly between $\ln(D_{\text{min}})$ and $\ln(D_{\text{max}})$.

   Such sampling method achieves the goal that the probability of sampling a number from the interval $[a, b]$ and the probability of sampling a number from the interval $[c, d]$ is equal if and only if $b/a = d/c$. This phase lasted 10 epochs and was performed with 10^{-6} learning rate.
4. The fixed sparsity training phase. The goal of the last training phase was to train the network at the target data sparsity (50% masked pixels, 75% masked pixels, 90% masked pixels, 95% masked pixels and 99% masked pixels). This phase lasted 2 epochs for each version of the network for a particular sparsity and was performed with 10^{-6} learning rate.

For the training of the RGB-guided network, the weights obtained from training the unguided version for the first 2 phases were used as the starting weights of the lidar encoder. Additionally, 2 extra phases were employed after phase 2 and before phases 3 and 4:

1. The RGB-encoder training phase. During this phase, the weights of the lidar encoder were frozen. This phase lasted 10 epochs. The optimizer learning rate was decreased from 10^{-4} to 10^{-5} to 10^{-6} during this phase.
2. Joint training phase. During this phase, the whole network was trained (no weights were frozen). This phase lasted 10 epochs and was performed with 10^{-6} learning rate.

Huber loss was utilized during the entirety of the training. The loss was quadratic for errors smaller than 5 m and linear for errors greater than 5 m. The whole training was performed using Adam (Kingma and Ba [2014]) optimizer.

4.4 Experiments

4.4.1 Results on Standard Input Density

In this section, I present how WeaveNet compares to other guided and unguided solutions available in the literature (on the basis of Root Mean Square Error (RMSE) and Mean Absolute Error (MAE)). All the comparisons are based on the results on the KITTI validation set.

The RGB-guided version of WeaveNet is reasonably close to state-of-the-art in terms of MAE in the task of guided depth completion (Table 4.2). The RGB-guided version of WeaveNet is significantly worse than the current state-of-the-art RGB-guided depth completion solutions in terms of RMSE (Table 4.2). It should be noted, that as of today the solutions not relying on sparsity invariant convolution seem to be significantly better at the task of guided depth completion.

Unguided WeaveNet achieves state-of-the-art results on the task of lidar-only depth completion. To the best of my knowledge, WeaveNet is currently the best lidar-only depth completion solution in terms of MAE (Table 4.1). Unfortunately, WeaveNet is inferior to other unguided solutions (Huang et al. [2019b], Ma et al. [2018]) in terms of RMSE.

Table 4.1. Performance of the unguided depth completion methods on the KITTI validation set

Model	RMSE [m]	MAE [m]
Uhrig et al. [2017]	2.01	0.68
Ma et al. [2018]	**0.99135**	not provided
Huang et al. [2019b]	0.99414	0.26241
WeaveNet (standard training)	1.09596	**0.24512**
WeaveNet (universal sparsity training)	1.17372	0.27323
WeaveNet (specific sparsity training)	1.08774	0.25310

Table 4.2. Performance of the RGB-guided depth completion methods on the KITTI validation set

Model	RMSE [m]	MAE [m]
Eldesokey et al. [2018]	1.37	0.38
Tang et al. [2019]	0.77778	0.22159
Park et al. [2020]	0.88410	not provided
Cheng et al. [2019a]	0.72543	**0.20788**
Chen et al. [2019]	0.75288	0.22119
Qiu et al. [2019]	**0.68700**	0.21538
Yan et al. [2020]	0.79280	0.22581
Ma et al. [2018]	0.856754	not provided
Huang et al. [2019b]	0.88374	0.25711
WeaveNet (standard training)	0.97175	0.22060
WeaveNet (universal sparsity training)	1.05978	0.26249
WeaveNet (specific sparsity training)	0.94849	0.22623

4.4.2 Input Density Ablation Study

For the purpose of checking the performance of the network while using very sparse input, I tested it on the validation set with randomly masked depth input pixels. I performed the tests with 50% masked pixels (approximately 10 000 depth measurements left in the FOV), 75% masked pixels (approximately 5 000 depth measurements left in the FOV), 90% masked pixels (approximately 2 000 depth measurements left in the FOV), 95% masked pixels (approximately 1 000 depth measurements left in the FOV) and 99% masked pixels (approximately 200 depth measurements left in the FOV). For each of the sparsity levels I present the results obtained by:

a The network trained on the default KITTI data.

b The network trained with variable input sparsity.

c The network trained with the input at the specific sparsity level.

I present the quantitative results obtained using WeaveNet in Figures 4.6 and 4.7 for the unguided version of the network and in Figures 4.8 and 4.9 for the guided version. As can be seen in those plots, there is very little difference between the performance of the network trained for the specific input sparsity and the network trained using variable input sparsity (universal sparsity training). It points to the possibility of using the weights obtained during the universal training in systems where the number of measurements can change from one frame to another or in systems where the density of measurements may change between different parts of the FOV (such as the future imaging radars).

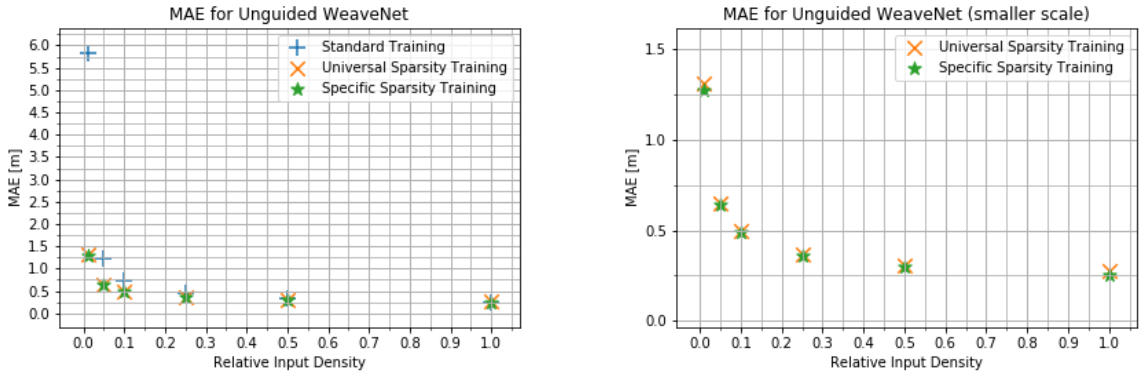


Figure 4.6. MAE for the unguided WeaveNet for various input densities for different training modes (different scales)

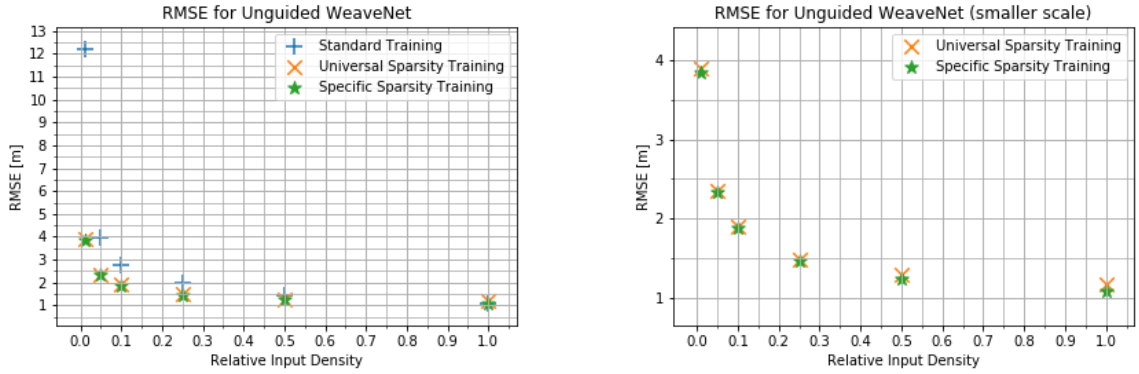


Figure 4.7. RMSE for the unguided WeaveNet for various input densities for different training modes (different scales)

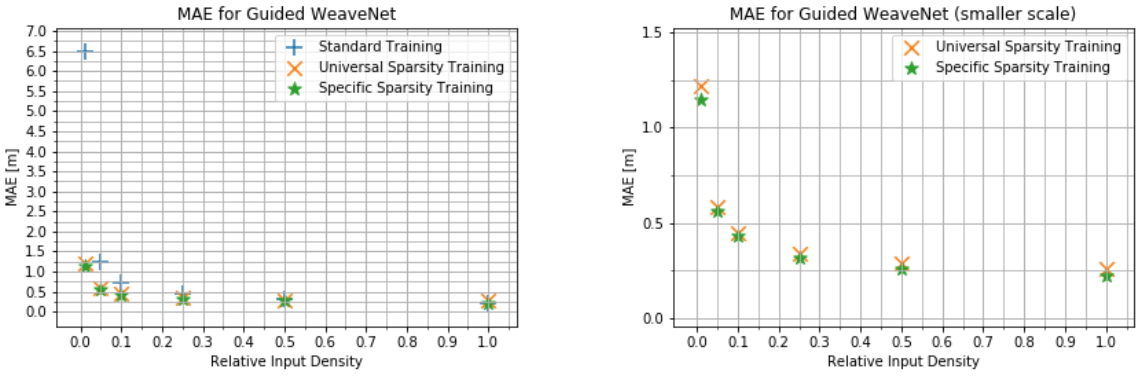


Figure 4.8. MAE for the guided WeaveNet for various input densities for different training modes (different scales)

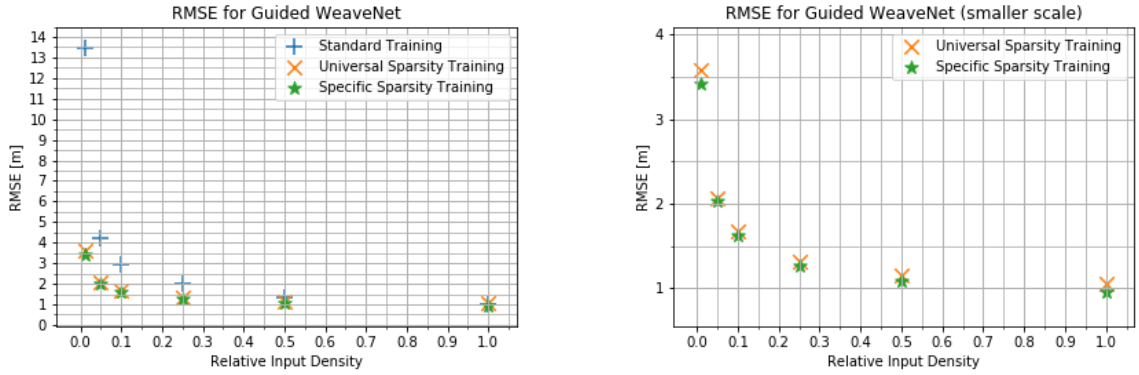


Figure 4.9. RMSE for the guided WeaveNet for various input densities for different training modes (different scales)

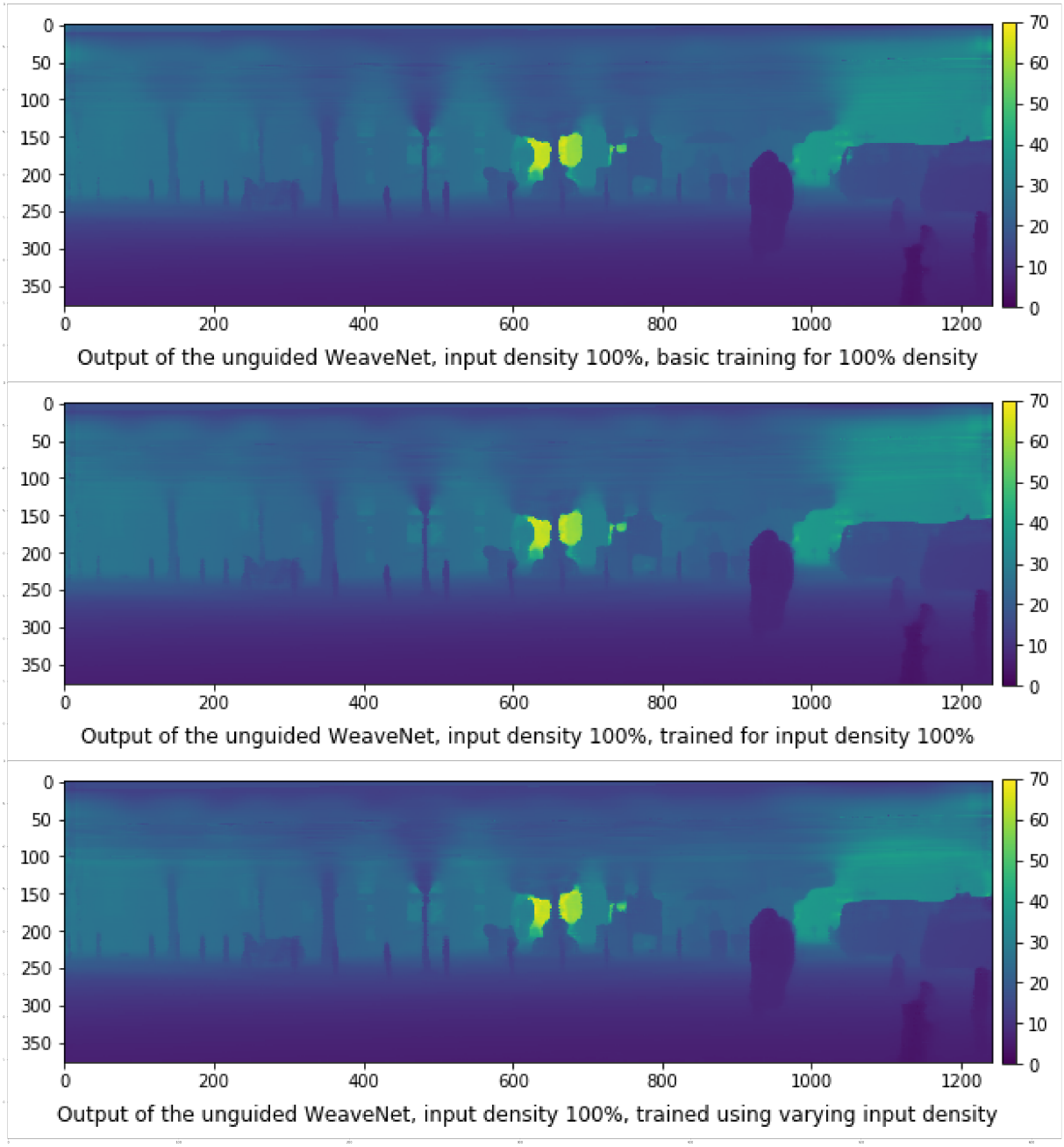


Figure 4.10. Output of the unguided WeaveNet for 100% density for different training modes (basic training on the top, training for specific density in the middle, training for variable density on the bottom). The depth is color coded, with the scale shown to the right of the images. Numbers on the axes denote pixel coordinates.

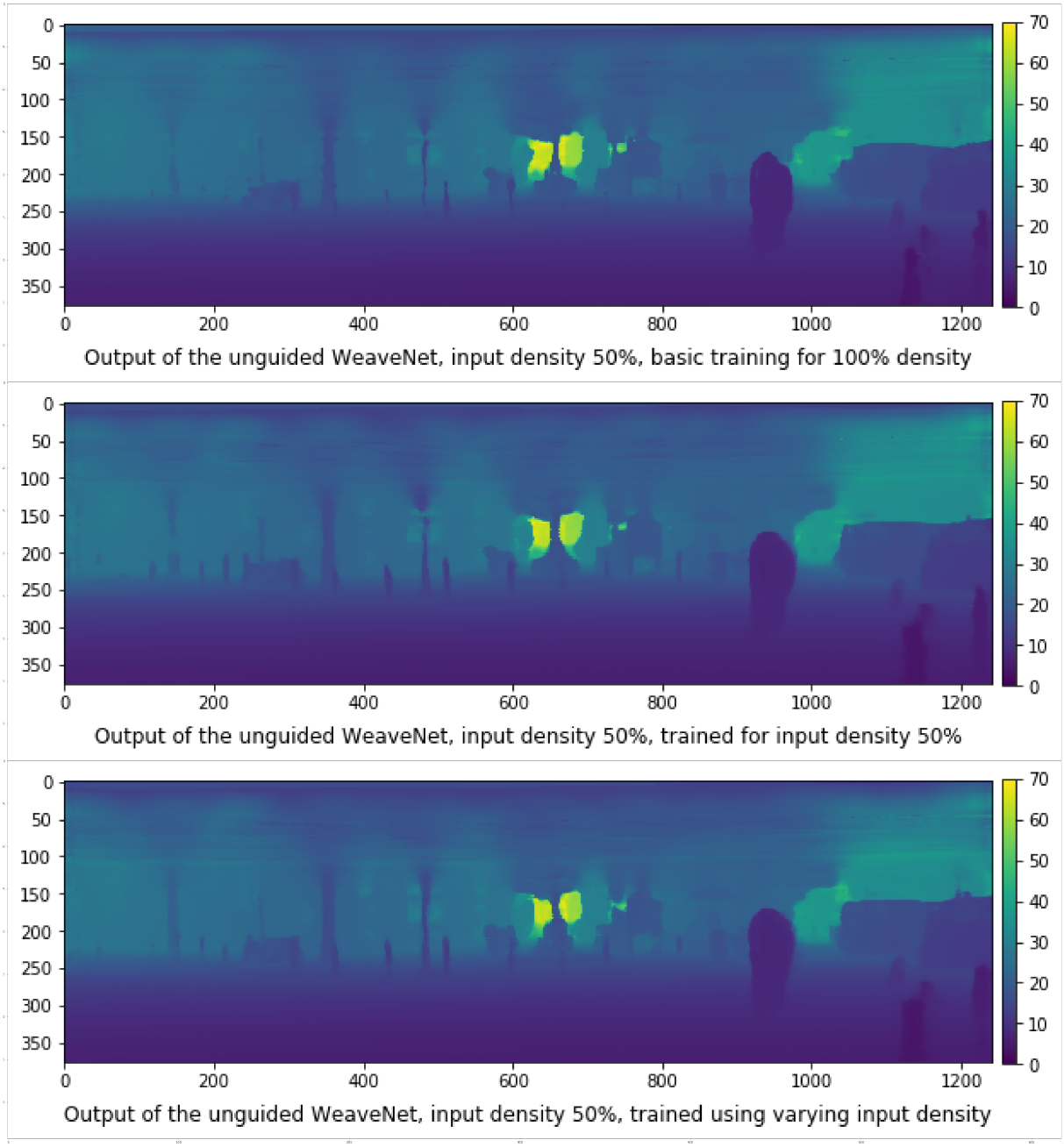


Figure 4.11. Output of the unguided WeaveNet for 50% density for different training modes (basic training on the top, training for specific density in the middle, training for variable density on the bottom). The depth is color coded, with the scale shown to the right of the images. Numbers on the axes denote pixel coordinates.

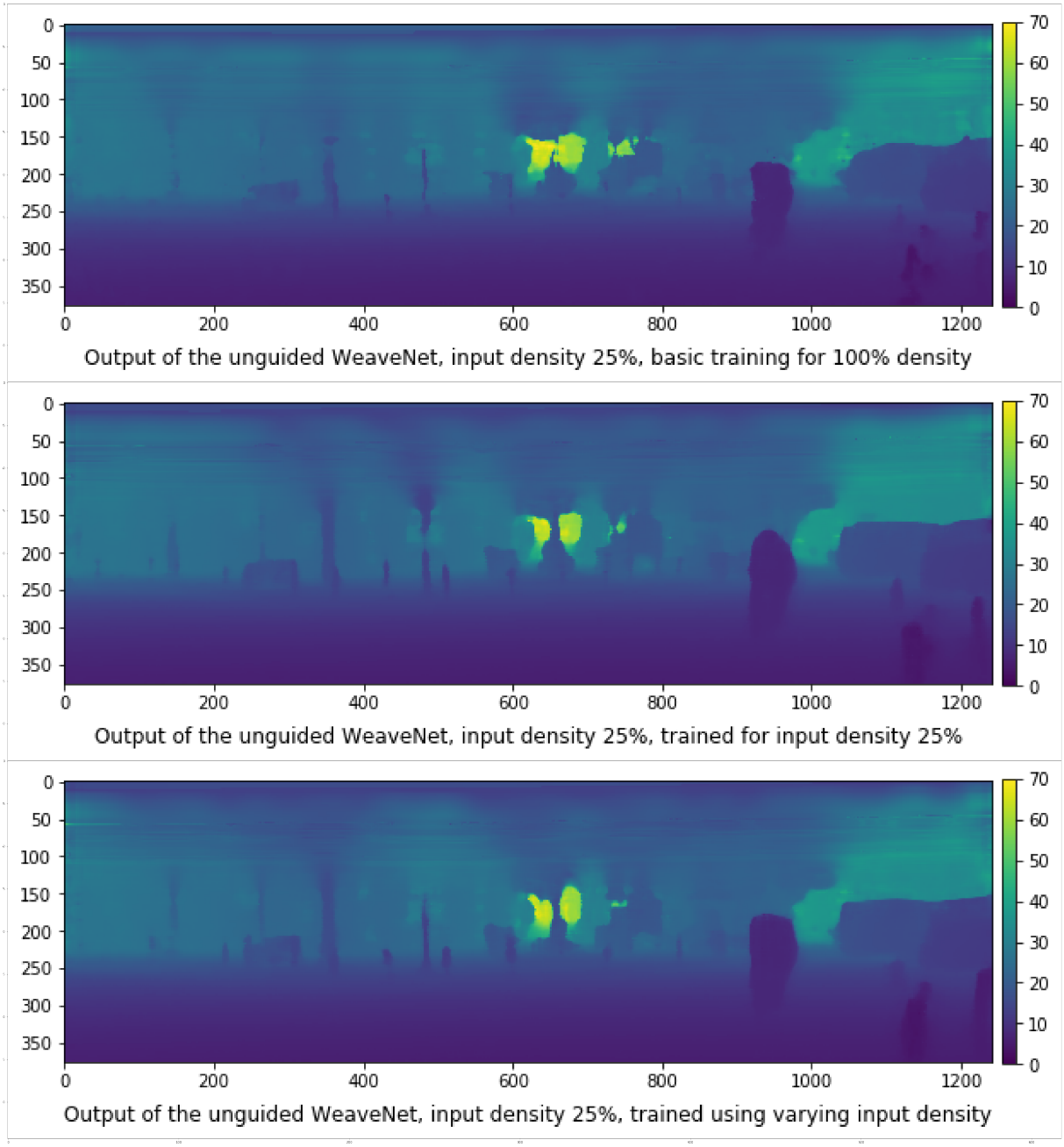


Figure 4.12. Output of the unguided WeaveNet for 25% density for different training modes (basic training on the top, training for specific density in the middle, training for variable density on the bottom). The depth is color coded, with the scale shown to the right of the images. Numbers on the axes denote pixel coordinates.

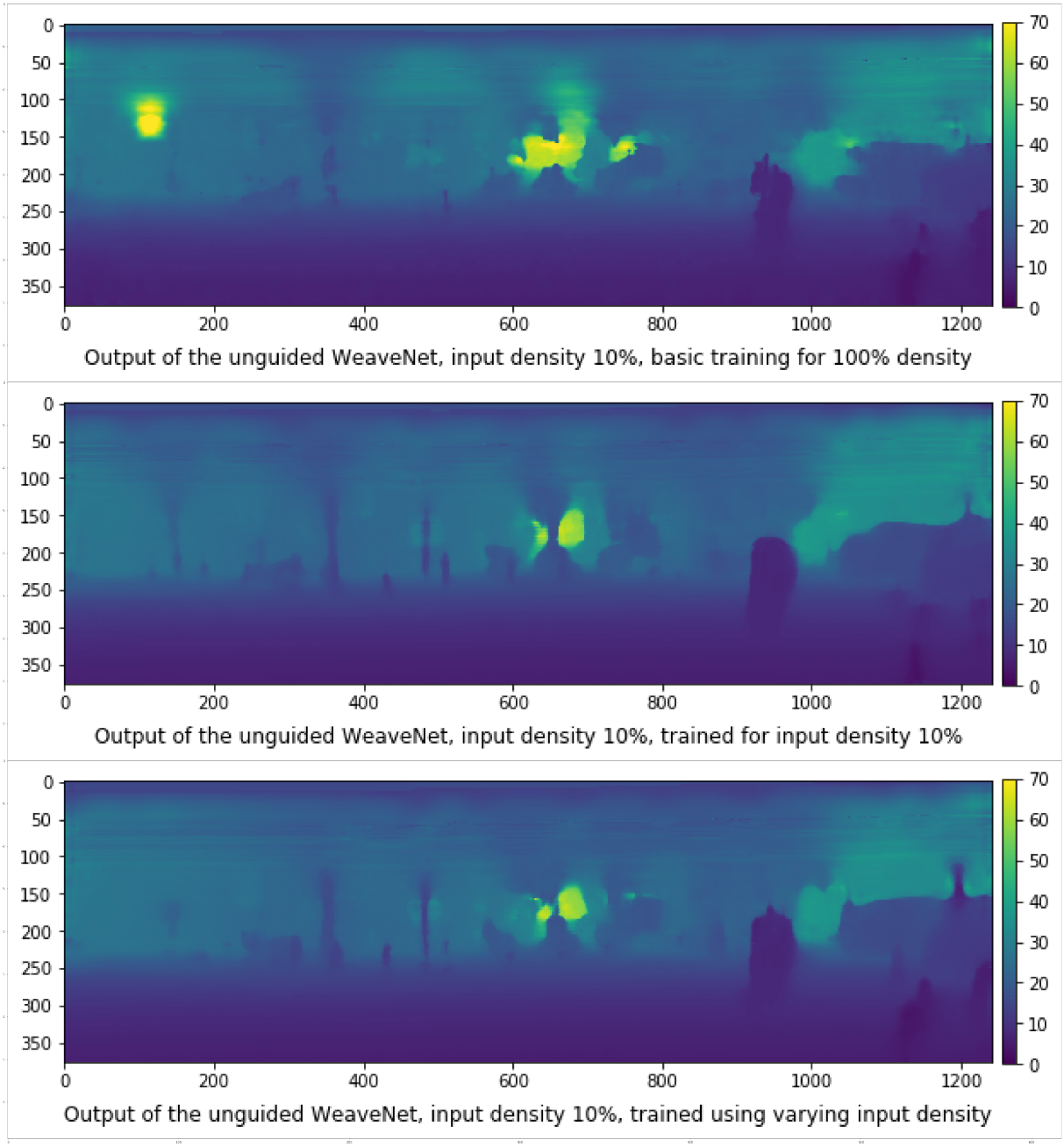


Figure 4.13. Output of the unguided WeaveNet for 10% density for different training modes (basic training on the top, training for specific density in the middle, training for variable density on the bottom). The depth is color coded, with the scale shown to the right of the images. Numbers on the axes denote pixel coordinates.

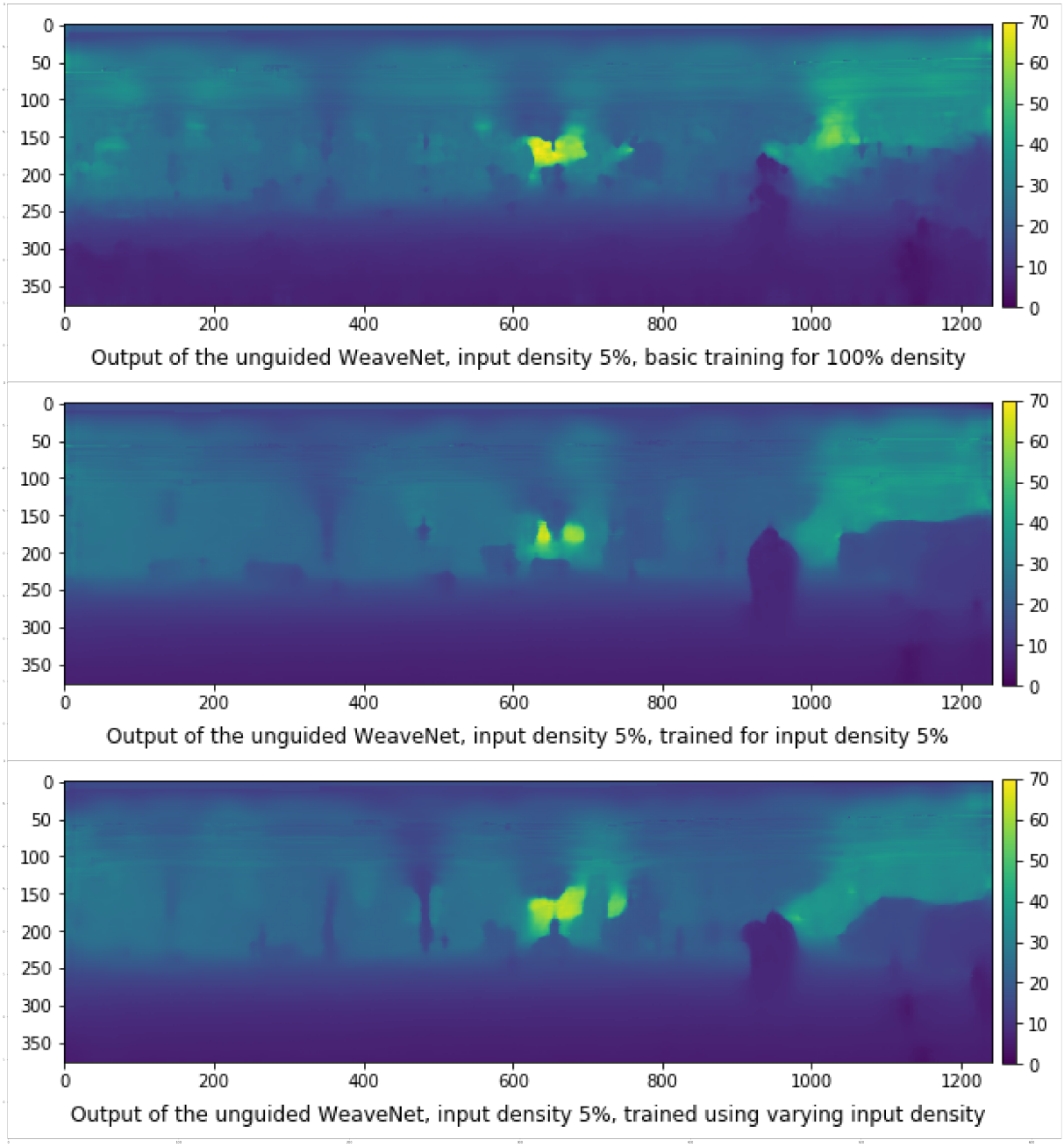


Figure 4.14. Output of the unguided WeaveNet for 5% density for different training modes (basic training on the top, training for specific density in the middle, training for variable density on the bottom). The depth is color coded, with the scale shown to the right of the images. Numbers on the axes denote pixel coordinates.

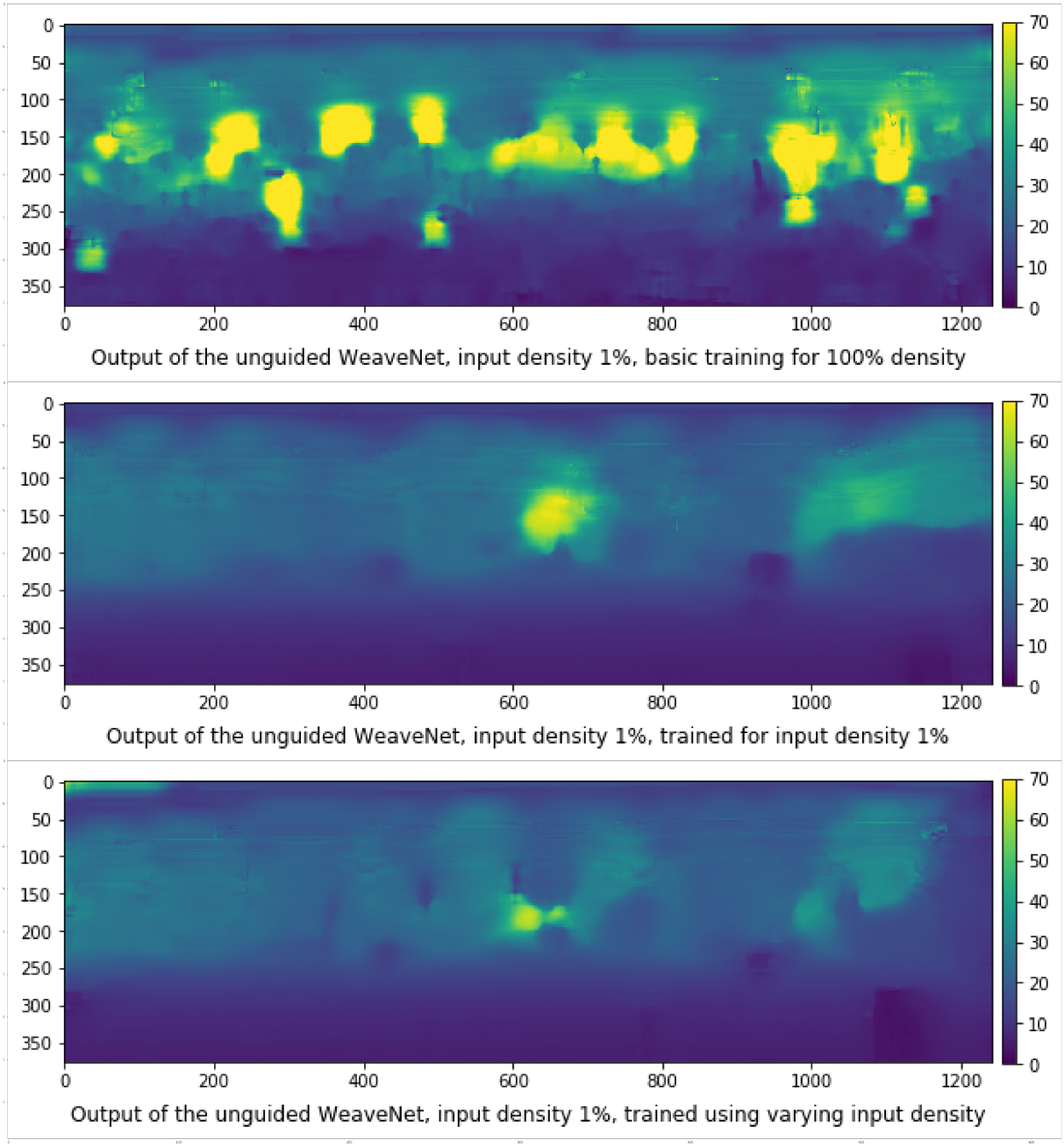


Figure 4.15. Output of the unguided WeaveNet for 1% density for different training modes (basic training on the top, training for specific density in the middle, training for variable density on the bottom). The depth is color coded, with the scale shown to the right of the images. Numbers on the axes denote pixel coordinates.

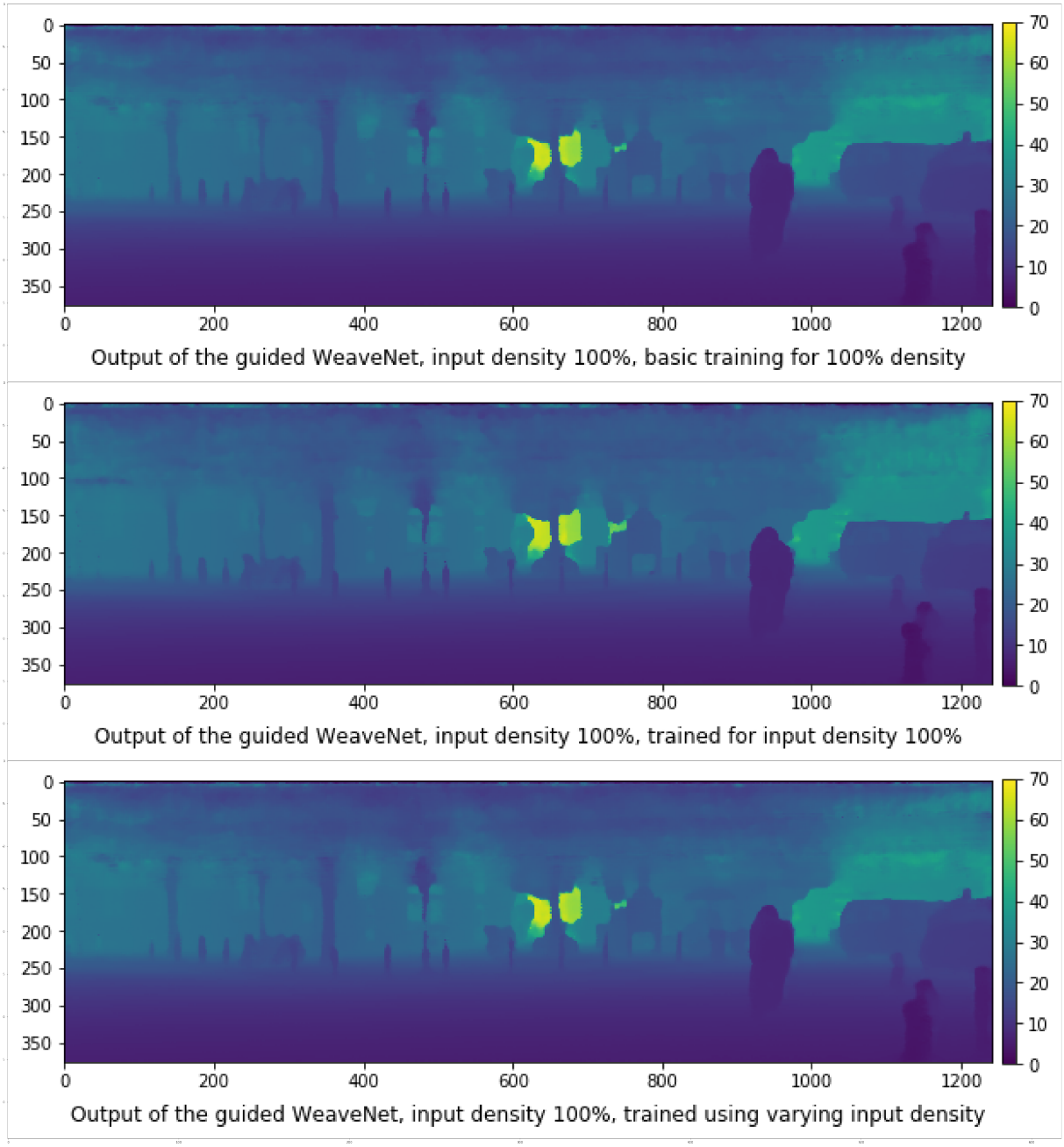


Figure 4.16. Output of the guided WeaveNet for 100% density for different training modes (basic training on the top, training for specific density in the middle, training for variable density on the bottom). The depth is color coded, with the scale shown to the right of the images. Numbers on the axes denote pixel coordinates.

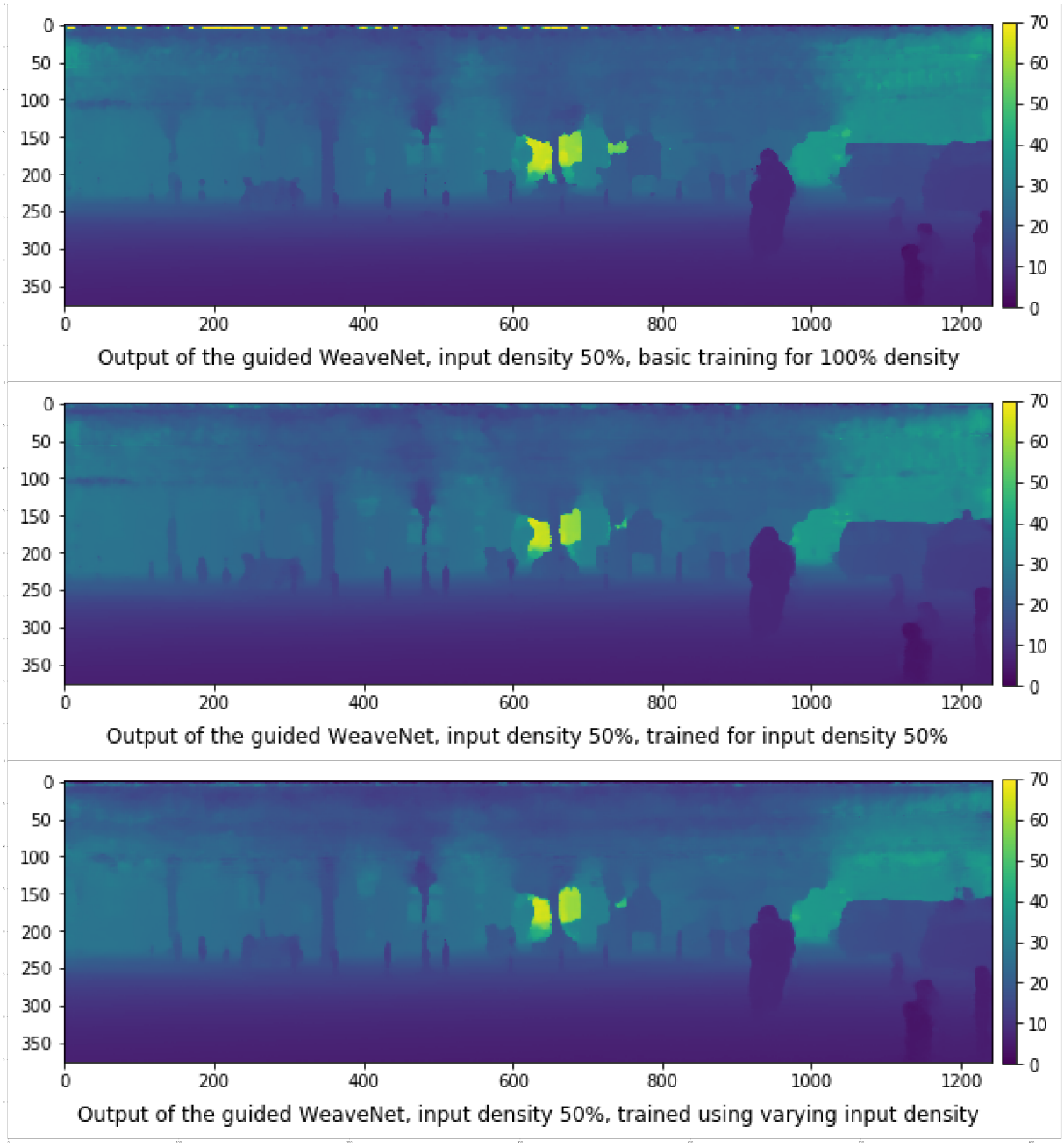


Figure 4.17. Output of the guided WeaveNet for 50% density for different training modes (basic training on the top, training for specific density in the middle, training for variable density on the bottom). The depth is color coded, with the scale shown to the right of the images. Numbers on the axes denote pixel coordinates.

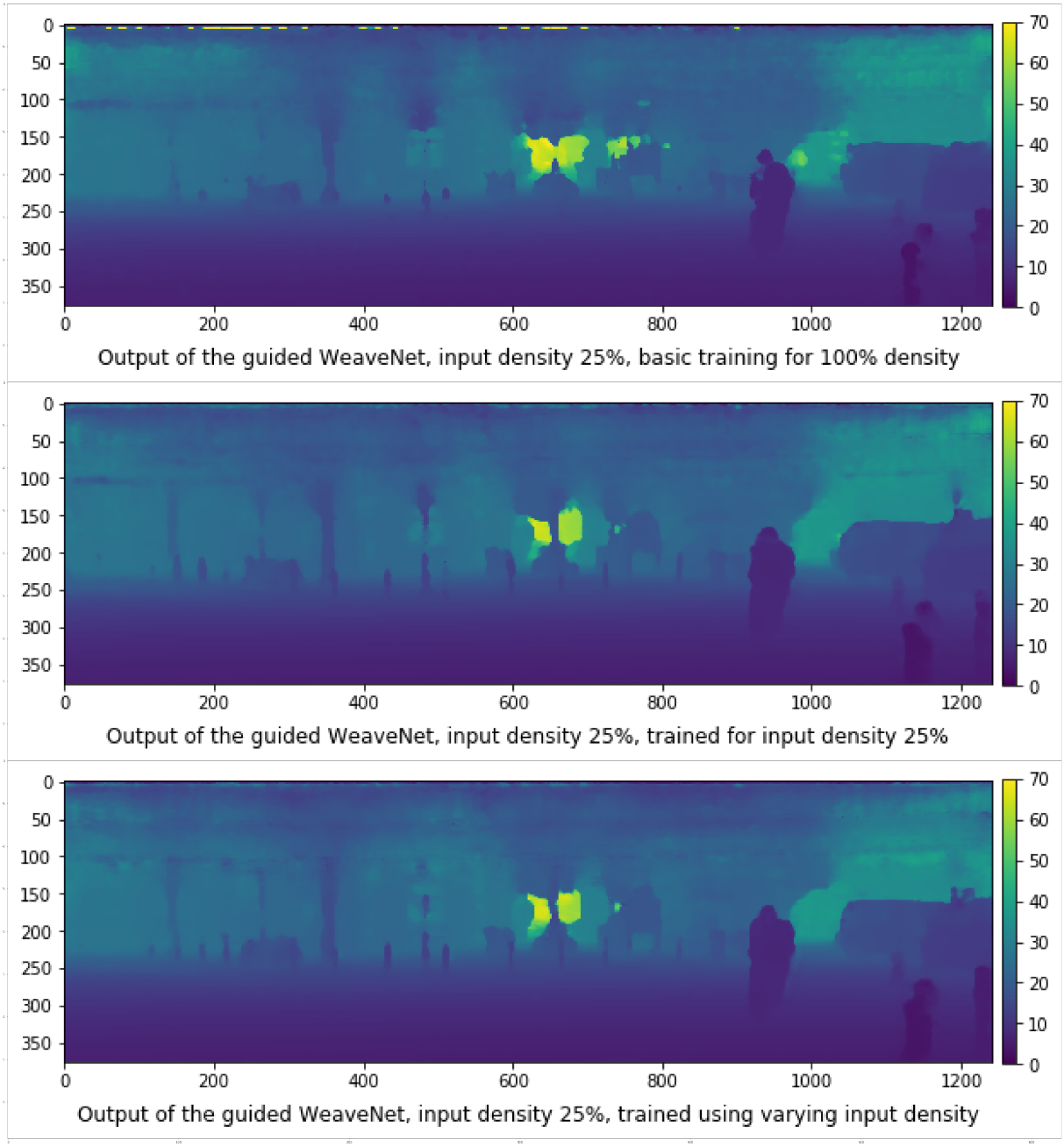


Figure 4.18. Output of the guided WeaveNet for 25% density for different training modes (basic training on the top, training for specific density in the middle, training for variable density on the bottom). The depth is color coded, with the scale shown to the right of the images. Numbers on the axes denote pixel coordinates.

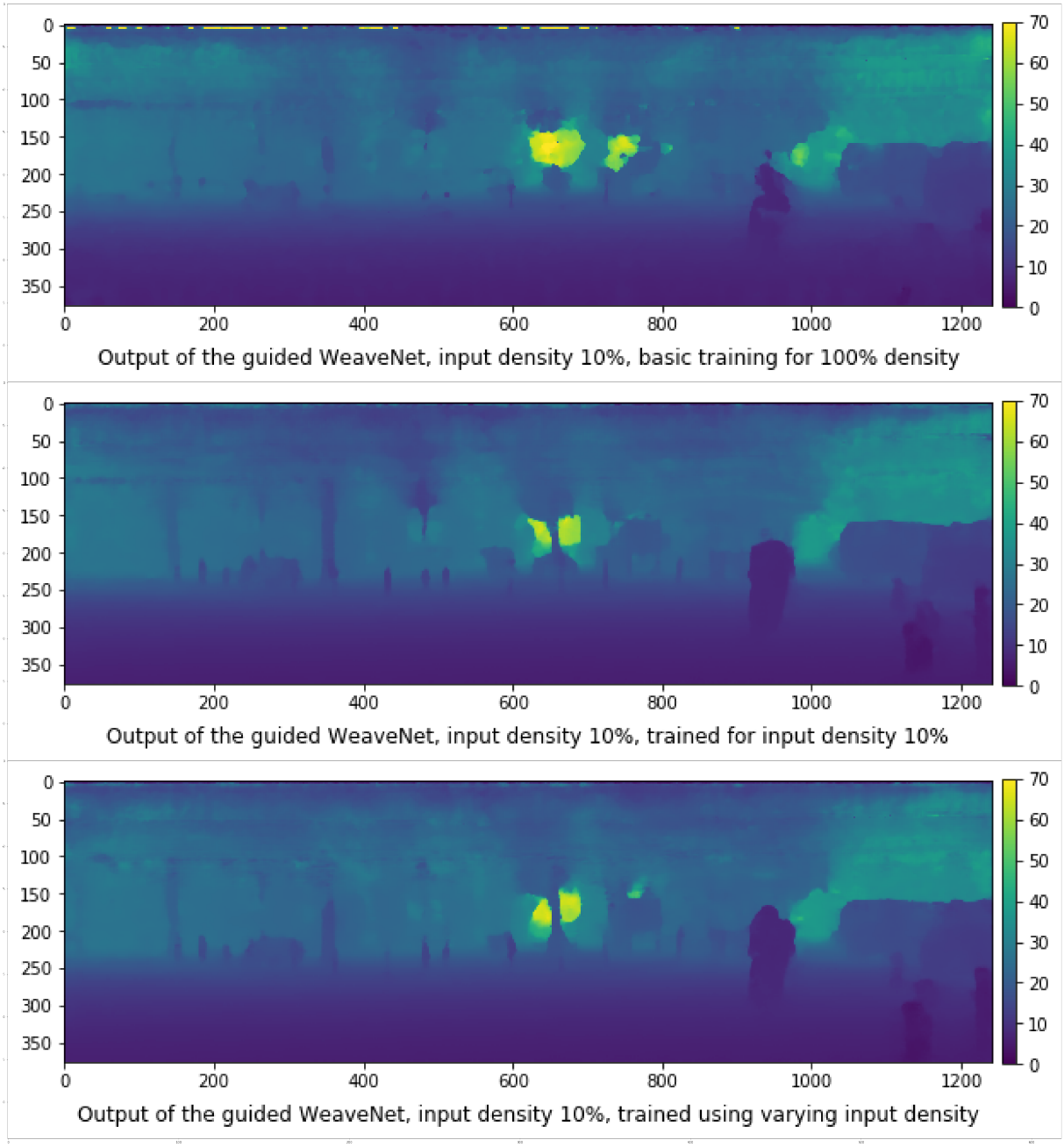


Figure 4.19. Output of the guided WeaveNet for 10% density for different training modes (basic training on the top, training for specific density in the middle, training for variable density on the bottom). The depth is color coded, with the scale shown to the right of the images. Numbers on the axes denote pixel coordinates.

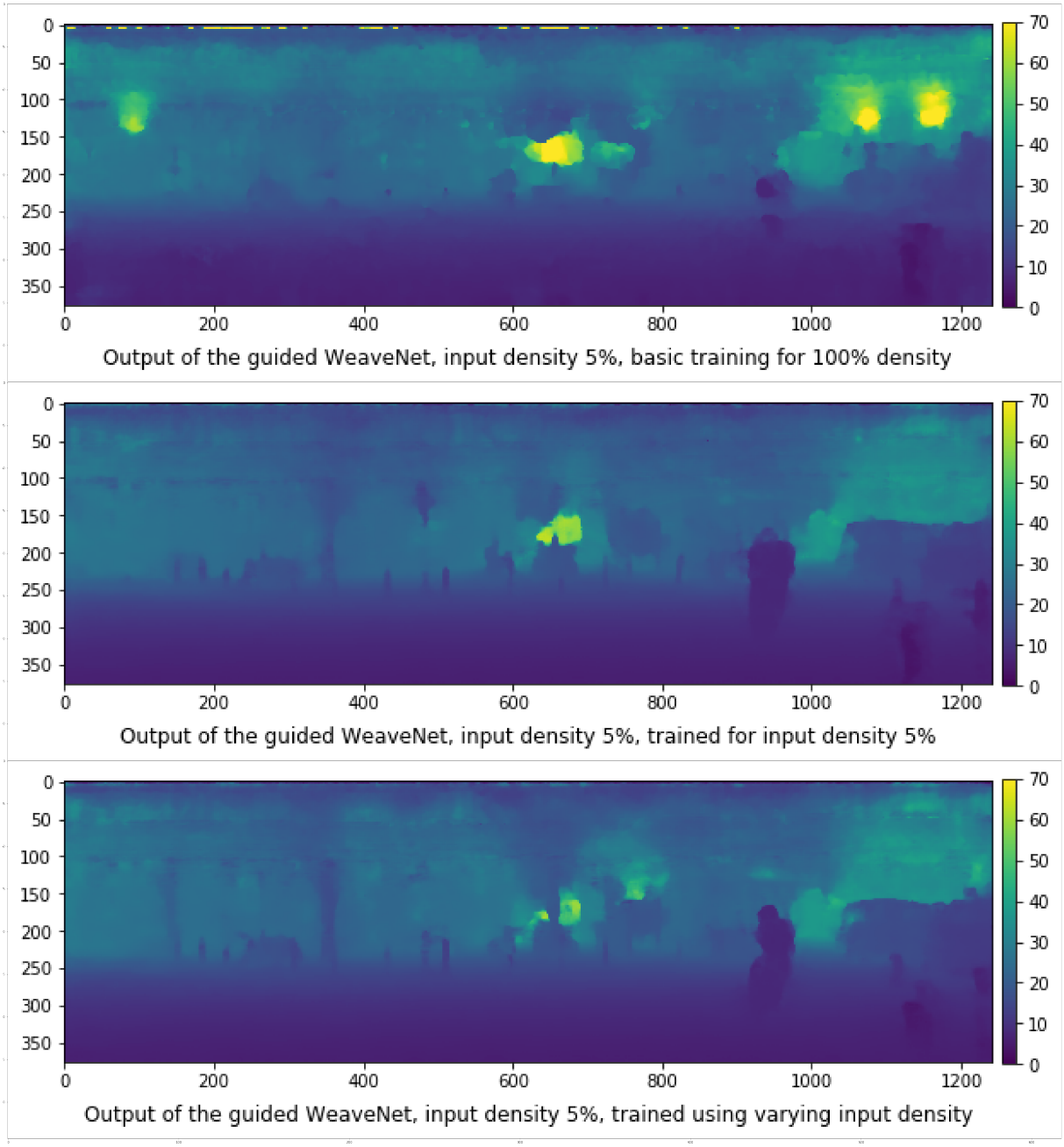


Figure 4.20. Output of the guided WeaveNet for 5% density for different training modes (basic training on the top, training for specific density in the middle, training for variable density on the bottom). The depth is color coded, with the scale shown to the right of the images. Numbers on the axes denote pixel coordinates.

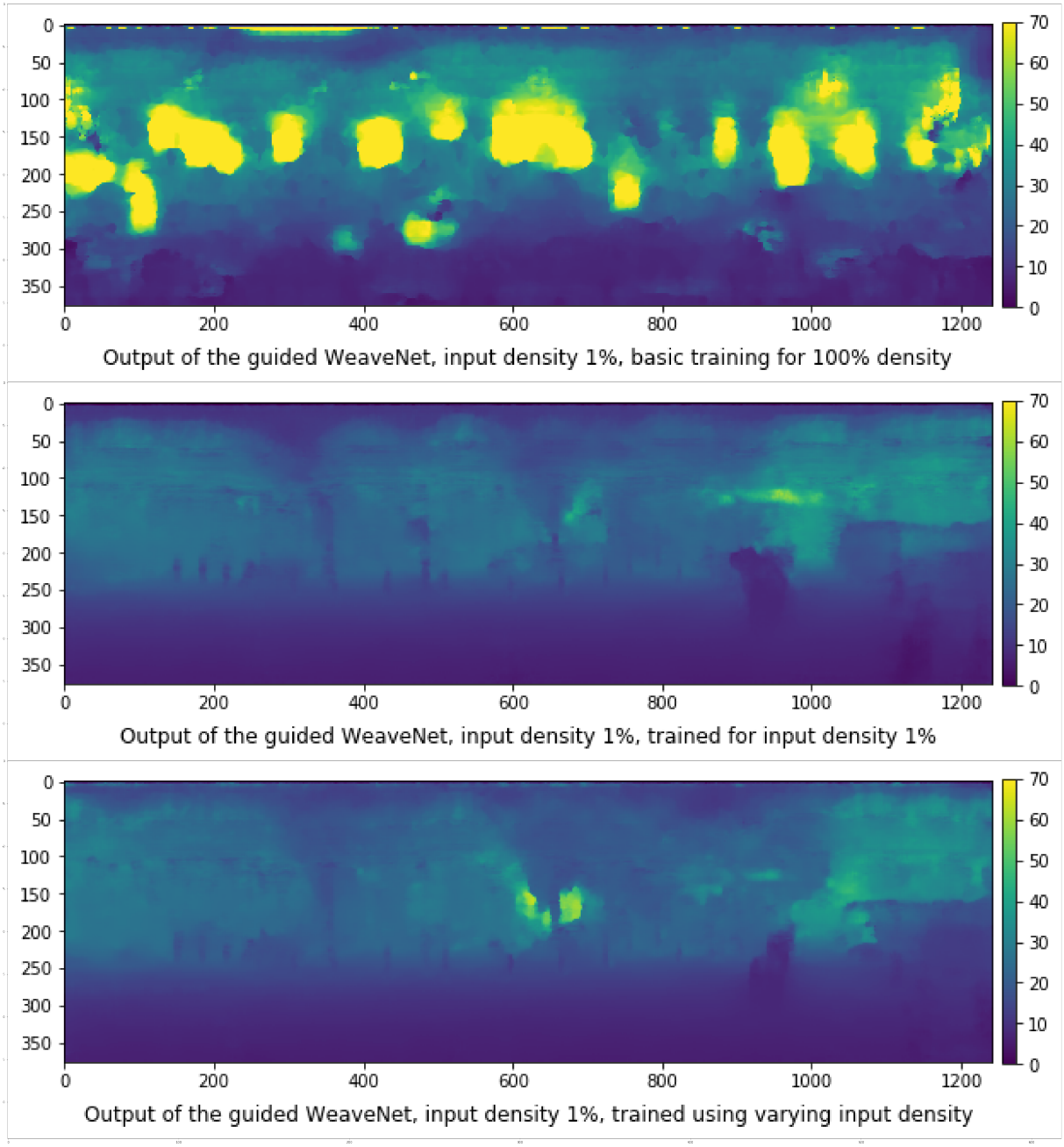


Figure 4.21. Output of the guided WeaveNet for 1% density for different training modes (basic training on the top, training for specific density in the middle, training for variable density on the bottom). The depth is color coded, with the scale shown to the right of the images. Numbers on the axes denote pixel coordinates.

In Figures 4.10-4.15 and 4.16-4.21 I show the depth output of the WeaveNet (its unguided and guided versions respectively) for various input densities and for the 3 aforementioned training methods. The depth is color coded, with the scale shown to the right of each small image. In all cases, I use the same frame as in Figure 4.1. The main reason for including all these pictures is to show the reader the good performance of the network trained using variable sparsity. Please note that there is little qualitative difference between the output produced by the version of the network trained at specific input density and the version of the network trained using variable input density (universal sparsity training). For the unguided version of the network, the fine details, such as the protection poles or street lamps are reasonably distinguishable even at the relative input density of 10%. In the guided version, the details are still distinguishable even at 1% input density.

4.5 Conclusions

In the chapter, I presented a novel WeaveNet architecture capable of performing the depth completion task on very sparse input data. The results obtained by the version of the network trained using variable input sparsity are particularly promising since they point to the possibility of using depth completion methods using data from sensors producing a highly variable number of irregularly spaced measurements. An example of such a sensor that might in the future benefit from depth completion is a high-resolution imaging radar.

Another contribution of this work is the interpretation of the channels at the intermediate layers of the network. The knowledge that the channels in the intermediate layers of the network either represent the interpolated depth data or are estimating the positions of object boundaries might help improve future depth completion networks by incorporating auxiliary losses using the intermediate layer output.

I used the WeaveNet network described in this chapter to create the radar depth completion dataset (described closer in Chapter 6). The robustness of the method to the varying density of depth measurements was of crucial importance, as to collect that dataset we used the Pandora lidar (instead of the Velodyne one). This enabled the training of the radar depth completion network, described in Chapter 7.

5 Neural Network Angle Finding Simulations

5.1 Can a Neural Network Perform Angle Finding?

In the chapter on automotive sensors, Section 2.2.2.2, I described the conventional algorithms used for angle finding. In Chapter 7, I describe the core part of this work - a radar depth completion network operating directly on RDC. For that network to work, a necessary requirement is to be able to perform angle finding. In particular, it is crucial that the neural network is able to predict the angle of arrival when multiple waves arrive at the same time. In order to 'illuminate the black box' of my model from Chapter 7, I decided to perform some very simple simulations of radio waves irradiating a phased array antenna and trained small neural networks to extract angle and amplitude information from the signal on antennae elements.

5.2 Simulation

5.2.1 Antennae

In order to avoid leaking any proprietary information about antennae design, I decided to base all the experiments on the spacing of antenna elements described in the literature (specifically the unequal spacing antenna from Li and Li [2022]).

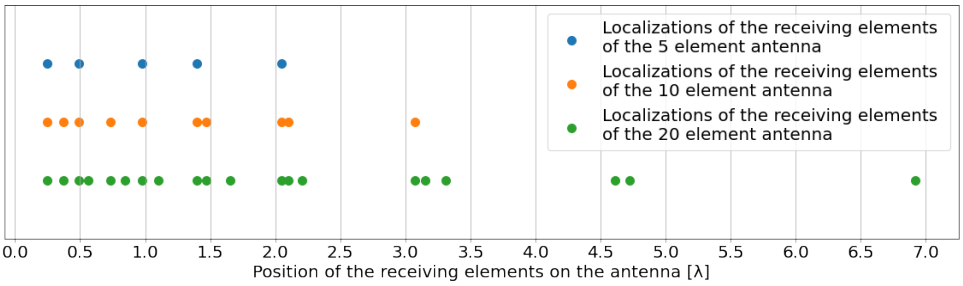


Figure 5.1. Localizations of the receiving elements of the antennae

I performed the experiments on three versions of the antenna:

- 5-element base antenna, with receiving elements at 0.25λ, 0.49λ, 0.98λ, 1.4λ and 2.05λ, where λ is the wavelength. All receiving elements were placed along one axis;
- 10-element antenna, which was created by concatenating the base antenna with another antenna whose elements are located at coordinates equal to 1.5 times the coordinates from the base antenna. In this case the receiving elements were located at 0.25λ, 0.375λ, 0.49λ, 0.735λ, 0.98λ, 1.4λ, 1.47λ, 2.05λ, 2.1λ and 3.075λ, all along one axis;
- 20-element antenna, which was created by concatenating the base antenna with 3 other antennae whose elements are located at coordinates equal to 1.5, 1.5^2 and 1.5^3 times the coordinates from the base antenna. In this case the receiving elements were located at 0.25λ, 0.375λ, 0.49λ, 0.5625λ, 0.735λ, 0.84375λ, 0.98λ, 1.1025λ, 1.4λ, 1.47λ, 1.65375λ, 2.05λ, 2.1λ, 2.205λ, 3.075λ, 3.15λ, 3.3075λ, 4.6125λ, 4.725λ and 6.91875λ, all along one axis;

Localizations of the receiving elements of the antennae are illustrated in Figure 5.1.

Thanks to using the 3 antennae of different sizes, I was not only able to show that a neural network can extract the angle of arrival and amplitude information from the signal on a phased array but also illustrate the influence of the size of the antennae on the quality of angle of arrival estimation. It matters, because automotive FMCW radars typically have more receiving elements located on the horizontal axis than on the vertical axis, hence the accuracy of azimuth estimation is much better than the accuracy of elevation estimation.

5.2.2 Waves

I assume, that the electromagnetic wave detected at the antenna reflects at a single point far away (at a distance far greater than the distance between receiving antennae). We can denote the distance to the reflection source by r. If the target is located at the angle α from the antenna, then the distance d in the angle plane can be phrased as (notation as in Figure 5.2):

$$d = r\sin(\alpha). \tag{5.2.1}$$

So the difference in the distance the wave has to travel between two antennae located ϵ from each other is:

$$\Delta r = \frac{\epsilon}{\sin(\alpha)}. \tag{5.2.2}$$

Let $k = \frac{2\pi}{\lambda}$ be the wavenumber, then the wave at r, at time t can be described by the following equation (describing a traveling sinusoidal wave):

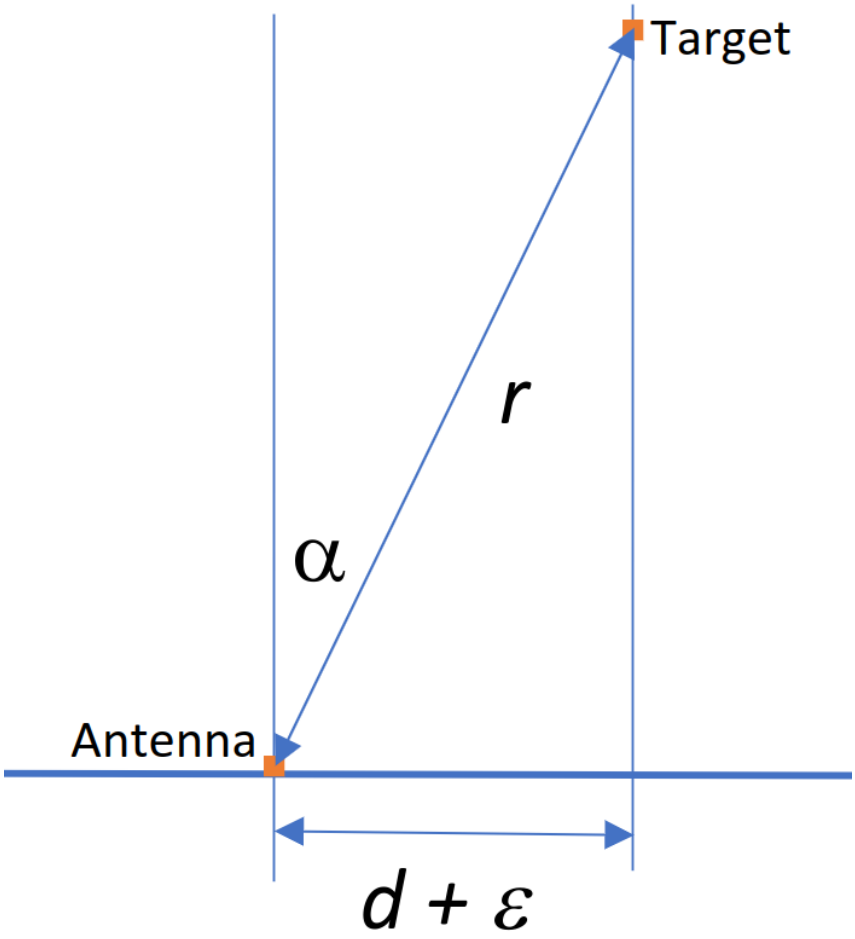


Figure 5.2. Notation used in the description of simulation

$$\psi(r,t) = \exp(ikr - i\omega t + i\phi), \tag{5.2.3}$$

where ϕ is some phase offset. Let the phase of the wave at the time of the measurement at $d_0 = 0$ be ϕ_0. Then for any d, the wave can be described by the following equation:

$$\psi(d) = \exp(ik\frac{d}{\sin(\alpha)} + i\phi_0), \tag{5.2.4}$$

and slightly more explicitly:

$$\psi(d) = \exp(i\frac{2\pi d}{\lambda\sin(\alpha)} + i\phi_0). \tag{5.2.5}$$

I used equation (5.2.5) to simulate the wave at the receiving antennae.

5.2.3 Simulation Mechanics

At each step, two waves were simulated (a higher amplitude wave and a lower amplitude wave). The form of experiments in which 2 waves are superimposed was chosen because it makes the angle determination task nontrivial - classical algorithms often fail in such cases. The angle to the source of each wave was sampled independently from a uniform distribution between -30 deg and 30 deg. Similarly, the phase ϕ_0 at $d = 0$ was sampled independently for each wave from a uniform distribution between 0 and 2π. The amplitudes were sampled in the following manner - first, the amplitude of the bigger wave was sampled from a uniform distribution between 0 and 1, and later the amplitude of the smaller wave was sampled from a uniform distribution between 0 and the amplitude of the bigger wave. Equation (5.2.5) was then used to calculate the state of the wave at the location of each receiving antenna. Then, the resulting wave (superposition of the small wave and big wave) was calculated by summing the signal from the small wave and big wave.

5.3 Neural Network Structure and Training

Inputs of the neural network consist of the real and imaginary part of the signal at each receiving antenna. The network is trained to produce the following outputs:

a angle to the source of the bigger wave (as a single number),

b angle to the source of the smaller wave (as a single number),

c amplitude of the bigger wave (as a single number),

d amplitude of the smaller wave (as a single number),

e angle to the source of the bigger wave (as a probability distribution),

f angle to the source of the smaller wave (as a probability distribution).

Outputs a-d were optimized using mean squared error and outputs e-f were optimized using binary cross-entropy calculated for 1 deg bins (each bin separately).

The middle part of the network consisted of 4 hidden layers, each with 128 neurons and swish (Ramachandran et al. [2017]) activation function, this structure is very similar to the initial angle finding module of the network described in Chapter 7.

Each training epoch consisted of 10,000,000 independent simulations. After each training epoch validation loss was calculated using 1,000,000 independent simulations. The training was performed using the following procedure:

1. Train the network using 10^{-3} learning rate for 100 epochs or until validation loss fails to improve for 10 epochs.

2. Perform additive weight perturbation, as described in Chapter 3.
3. Train the network using 10^{-4} learning rate for 100 epochs or until validation loss fails to improve for 10 epochs.
4. Perform additive weight perturbation, as described in Chapter 3.
5. Train the network using 10^{-5} learning rate for 100 epochs or until validation loss fails to improve for 10 epochs.
6. Perform additive weight perturbation, as described in Chapter 3.
7. Train the network using 10^{-6} learning rate for 100 epochs or until validation loss fails to improve for 10 epochs.

5.4 Results

I chose the Mean Absolute Error (MAE) as the Key Performance Indicator (KPI) for both angle of arrival estimation and amplitude estimation tasks. As evidenced by the data in Table 5.1, the network was able to learn useful information (i.e. perform better than chance) for all antennae sizes, however, the results are significantly better for larger antennae. In Figures 5.3-5.5 I show the error histograms for both tasks, both waves and all antennae sizes. In Figure 5.6 , I separately show the histogram of angle finding errors for weak wave on the 5 element antenna. I show it separately, because of its unusual shape - I believe, it is caused by the fact that the task of estimating the angle of arrival of the weak wave on the small antenna is so hard, that the error distribution is relatively flat (without a major peak at the center). When it comes to the other plots, somewhat surprisingly, the histograms of the amplitude estimation error for the weak wave and the strong wave were almost identical. They are also very similar to each other across the different antennae sizes. The differences in the histograms of errors for the angle of arrival estimation tasks are markedly different between the strong wave and the weak wave. For every antenna size, the distribution of errors for the weak wave has much fatter tails than for the strong wave.

Table 5.1. Performance of simulated radar antennae of different sizes

Antenna size	Strong wave angle of arrival MAE [deg]	Weak wave angle of arrival MAE [deg]	Strong wave amplitude MAE	Weak wave amplitude MAE
5 element antenna	6.85	12.02	0.046	0.048
10 element antenna	3.24	7.93	0.035	0.040
20 element antenna	2.50	7.18	0.032	0.042

Similarly, it can be noticed that as the antenna size increases, the error distributions become much more concentrated around zero.

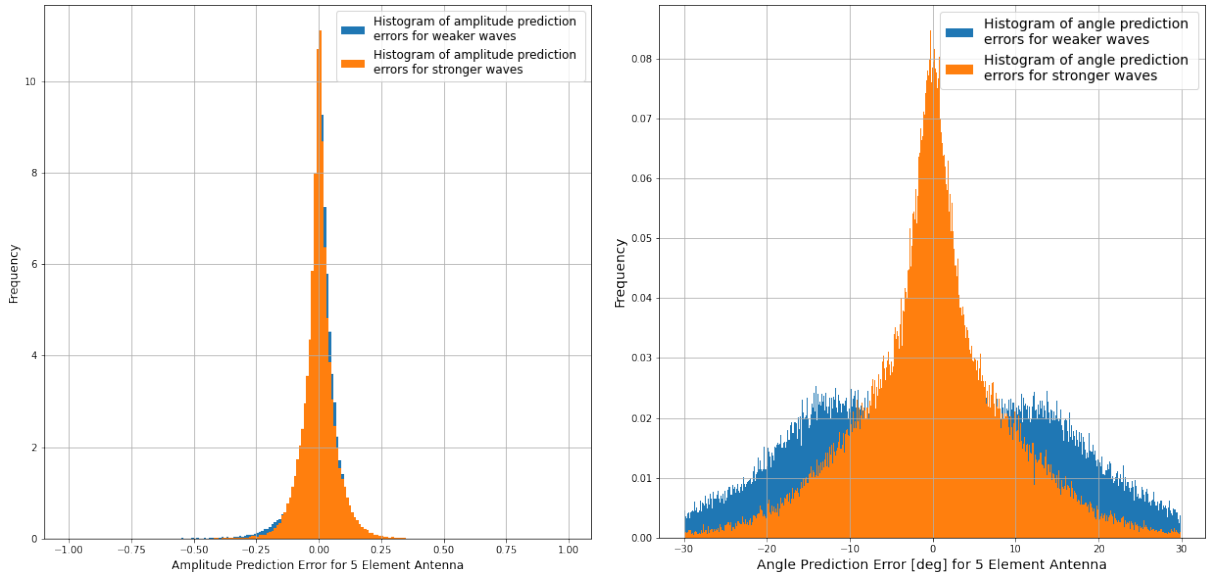


Figure 5.3. Histograms of amplitude prediction errors (left) and angle prediction errors (right) for the 5 element antenna.

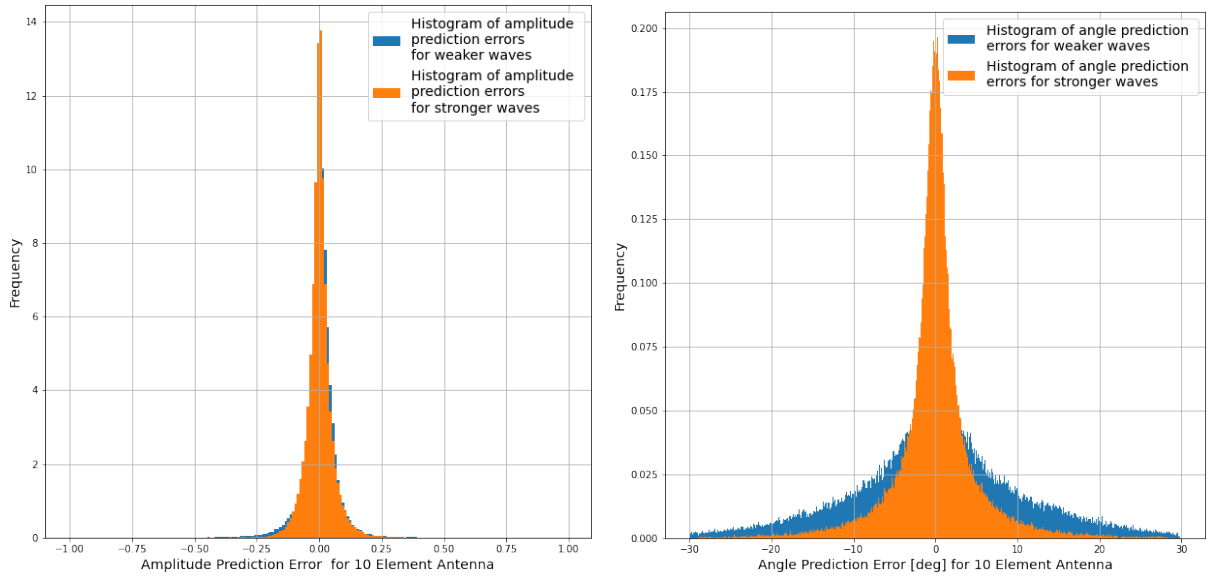


Figure 5.4. Histograms of amplitude prediction errors (left) and angle prediction errors (right) for the 10 element antenna.

Figure 5.7 presents the results of an experiment in which the amplitude of the strong wave was kept constant at 1 and the amplitude of the weak wave was set at different values between 0 and 1. At each level of the weak wave amplitude, 1000 simulations were performed and the KPIs for the angle of arrival

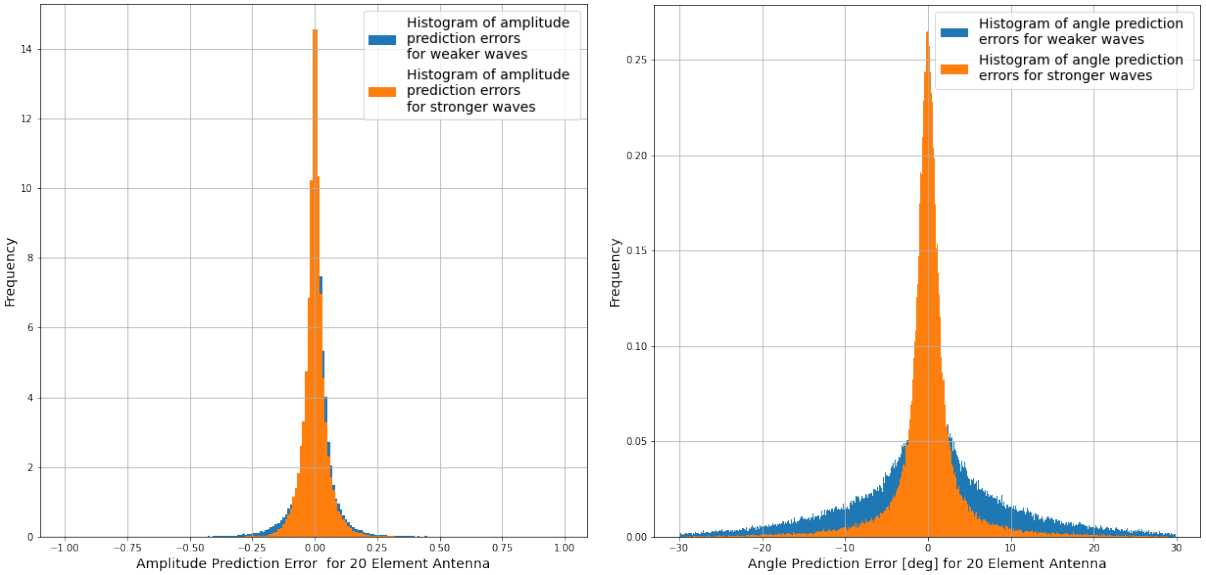


Figure 5.5. Histograms of amplitude prediction errors (left) and angle prediction errors (right) for the 20 element antenna.

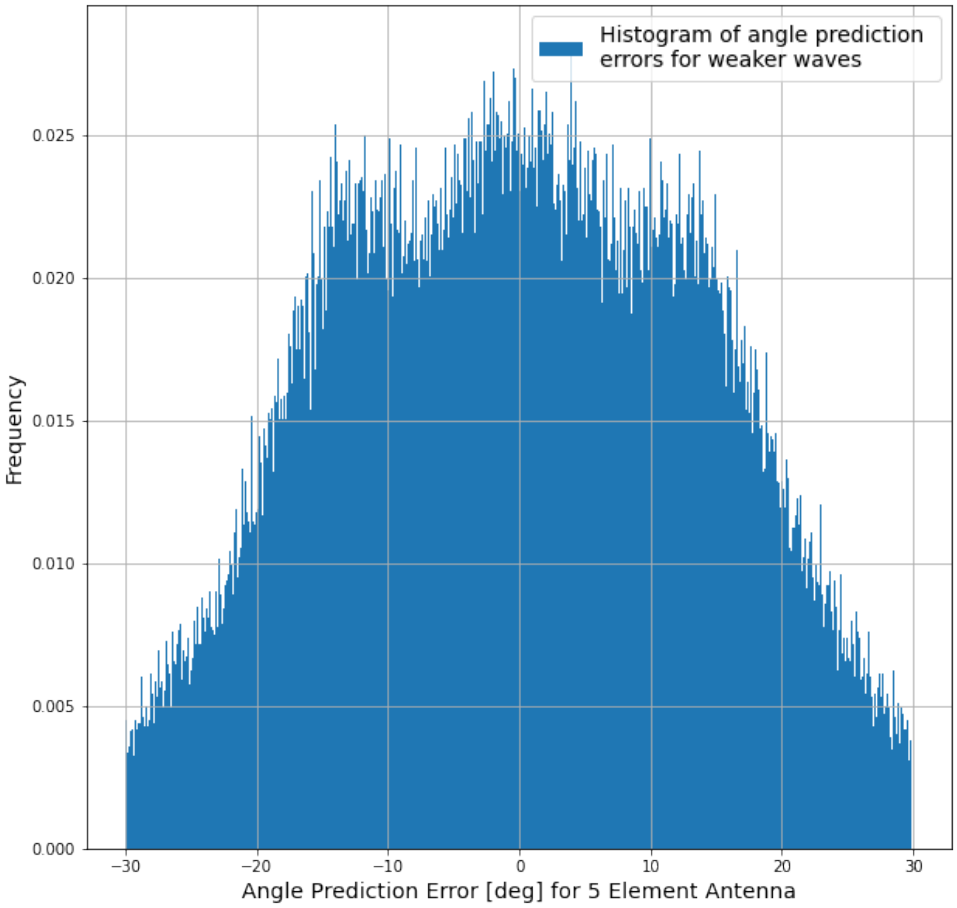


Figure 5.6. Estimation of the weak wave angle of arrival using the 5 element antenna.

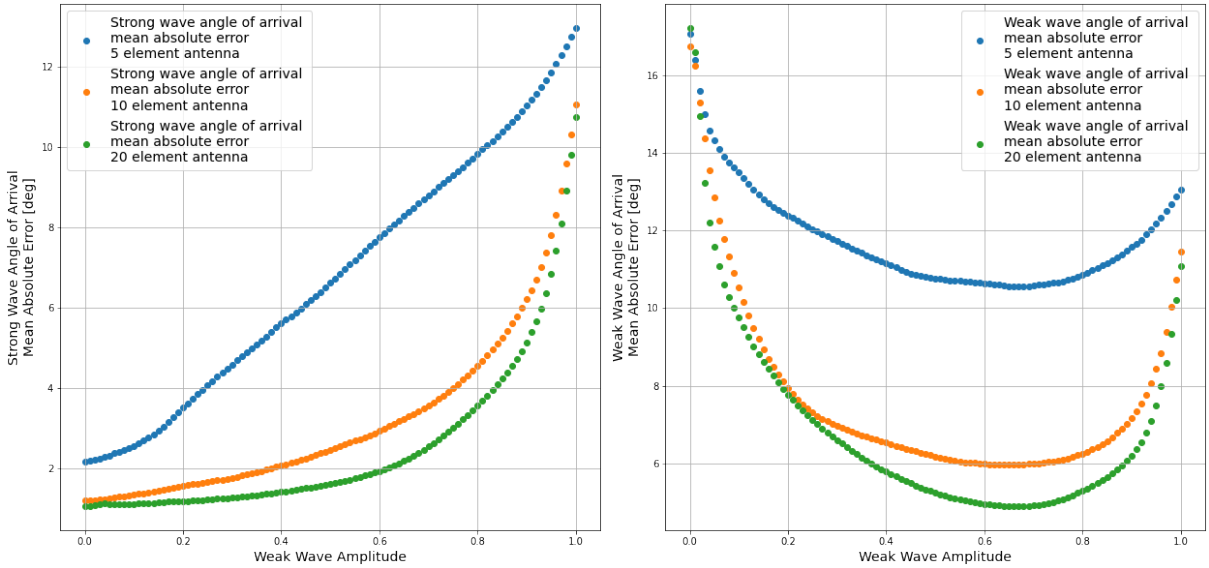


Figure 5.7. Plots of the mean absolute error of the angle of arrival estimation for different weak wave amplitudes (amplitude of the strong wave held constant at 1). Plot for the strong wave on the left, and the weak wave on the right.

estimation task were calculated. The plot on the left shows results for the estimation of the angle of arrival of the strong wave and the plot on the right shows results for the estimation of the angle of arrival of the weak wave. We can see, that as the amplitude of the weak wave increases, the quality of the angle finding for the strong wave decreases. The deterioration of performance is much more graceful for the bigger antennae. Somewhat surprisingly, the network achieves the best results for the weak wave not when the weak wave amplitude is close to 1, but when it is around 0.7. I believe, that this is an artifact caused by the fact that during training the weak wave only infrequently had higher amplitudes.

5.5 Conclusions

In this chapter, I have shown that a neural network can deal with the task of angle finding in a phased array radar. I have shown, that angle finding can be performed reasonably well even when 2 waves are superimposed at the antenna. I have analyzed the network performance at different levels of the amplitude of the weaker wave (which can be thought of as noise from the perspective of estimating strong wave parameters). I have illustrated the well-known fact that larger antennae are better at angle finding. The angle finding performance is better for larger antennae, since for a small antenna the same pattern at receiving elements may be the result of waves coming from different directions. This is illustrated in Figure 5.8. Please note, that for the small antennae the network predictions have multiple peaks, for different possible angles of arrival, and for the 20 element antenna, there is one very pronounced peak,

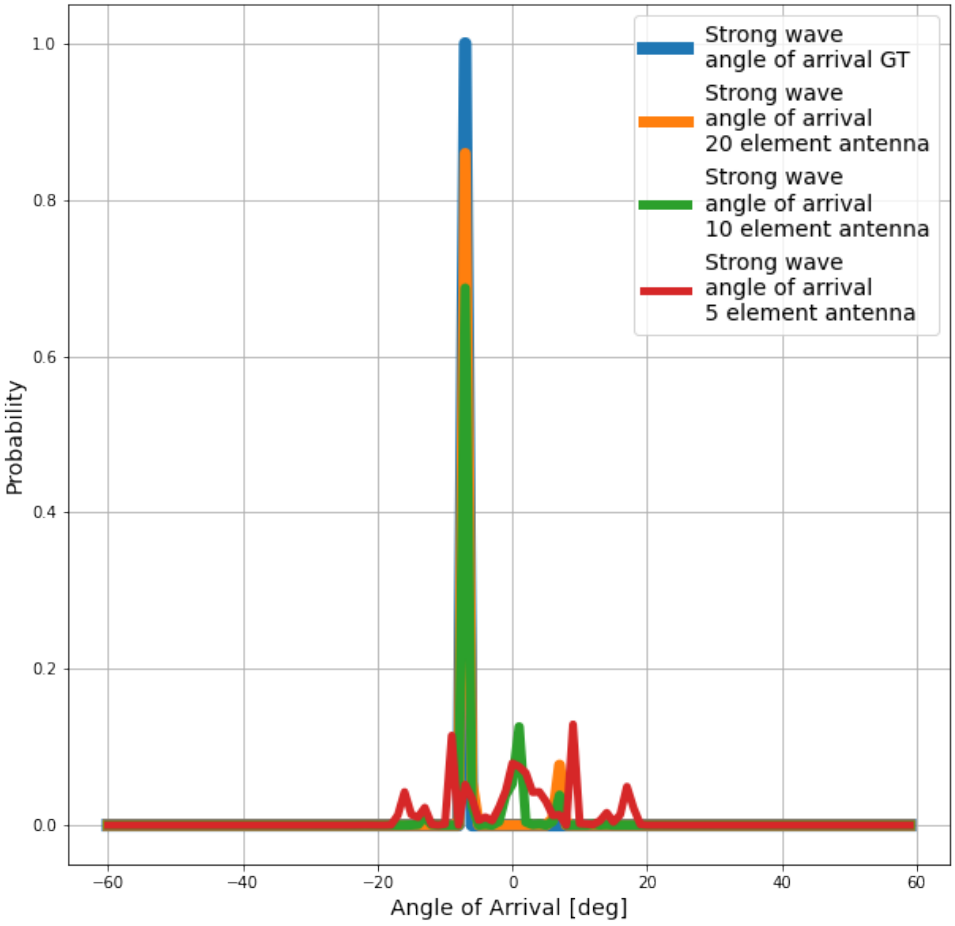


Figure 5.8. Estimation of the strong wave angle of arrival using antennae of different sizes.

at the correct angle. This result is very relevant to my experiments from Chapter 7, as the automotive radar antennae are significantly larger in the horizontal plane than in the vertical plane, thereby making the azimuth estimation much easier than elevation estimation. I believe that the methods presented in this chapter can be easily used to perform a fast evaluation of different radar antenna designs. Neural networks are universal approximators (Hornik et al. [1989]). Hence, they can be used to quickly put some bounds on the level of performance that can be obtained using a specific antenna design.

6 Dataset

6.1 Setup

We have collected a dataset that consists of approximately 1,000,000 radar frames of a forward-facing radar, which corresponds to roughly 14 hours of driving. Radar had 12 receiving antennae and utilized 77 GHz (3.9 mm) waves. The dataset was collected during 7 drives, 2 of which were done on a highway and 5 in an urban setting. All the drives were in good weather conditions. For each of the rides, the first 90% of frames (chronologically) were assigned to the train set, and the last 10% were assigned to the test set. The data collection vehicle was equipped with a Pandora lidar on top (lidar manual and specification available at https://www.symphotony.com/wp-content/uploads/20181015-Pandora-Users-Manual.pdf). Apart from the low-level radar data, images from a forward-looking RGB camera and lidar pointclouds were also collected. The car used for data collection is shown in Figure 6.1. A schematic representation of the car with marked sensor positions is shown in Figure 6.2.

For the purpose of the depth prediction task, the labels were created in the following process:

1. Lidar reflections from a forward-facing cone (about 50 deg in the azimuth plane, 20 deg in the elevation plane; roughly corresponding to the field of view of the radar) were chosen and projected onto a plane parallel to Pandora camera plane, but located closer to the radar.
2. Lidar depth completion was performed using a pretrained neural network on the depth data on the projection plane.

The lidar depth completion was performed using a neural network, which was pretrained on the KITTI Depth completion dataset in a way that makes the network invariant to the density of lidar detections, as described in Chapter 4. It is acceptable to use lidar and a lidar depth completion network to create labels for the radar depth completion task, as the quality of lidar depth completion is much better than the quality of my radar depth completion solution. It is important to note that unfortunately lidar and radar were mounted relatively far from each other (radar was mounted in the grille of the car and

Figure 6.1. Pictures of the car. Lidar is mounted on top and radar is mounted on the grille, facing forward.

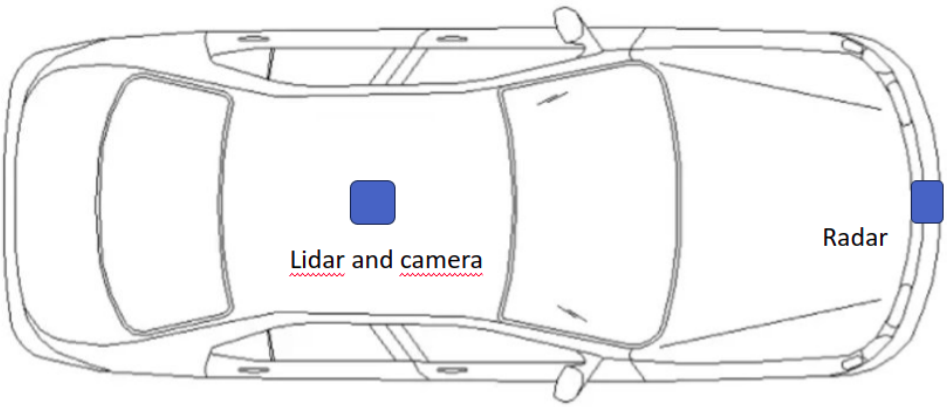


Figure 6.2. Drawing of a car with marked sensor positions, original drawing from https://cadbull.com/detail/92969/Sedan-model-top-view-model-of-car

lidar was mounted on top of the car), therefore the depth maps created from the perspective of lidar and radar do not align very well for close objects.

6.2 What Does the Radar See and What Does the Lidar See?

Radar and lidar operate in a fundamentally different way. The lidar mounted on top of the data collection vehicle uses a set of rotating lasers to illuminate targets around the sensors (described in Section 2.1). Almost every time the laser shines on an object that is within the operating range, the resulting reflection is strong enough to be detected (shown in Figure 2.3). Thanks to that, lidar produces a regular, semi-dense grid of depth measurements. On the other hand, the probability that a radar wave reflected from an object is recorded by the sensor depends heavily on the object material and geometry. Additionally, because of the way radar reflections are processed (described in more detail in 2.2.2) there can be only one non-ambiguous target for a given distance-velocity combination. Additionally, CFAR thresholding (described in Section 2.2.2.1) causes majority of the cells to be masked with zeros. The combined effects can be seen clearly in Figure 2.8 - there is a clearly visible vertical line, along the velocity bin associated with stationary objects. Please note, that this line has similar width along all distance bins - it means that radar is very good at detecting far-away objects. I present the mean distances measured by different sensors in Table 6.1. The full histogram of these measurements is shown in Figure 6.3. It should be noted, that the mean distances measured by the lidar are much smaller than the mean distances measured by the radar. For the whole dataset, the mean distance measured by lidar was approximately 27.9 m, while the mean distances measured by radar in its long look operating mode was 75.5 m and in the short look operating mode 71.1 m. Please look at the histogram in Figure 6.3, the overwhelming majority of lidar detections are clustered up to approximately 30 m from the sensor (corresponding to 50^{th} radar range bin). On the other hand, the distribution of the radar detections has a much fatter right range tail - the decrease of the number of radar detections with distance is almost a straight line. This points to the fact, that while lidar is great at providing information about the close surroundings of the car, radar is much better at detecting objects located far away.

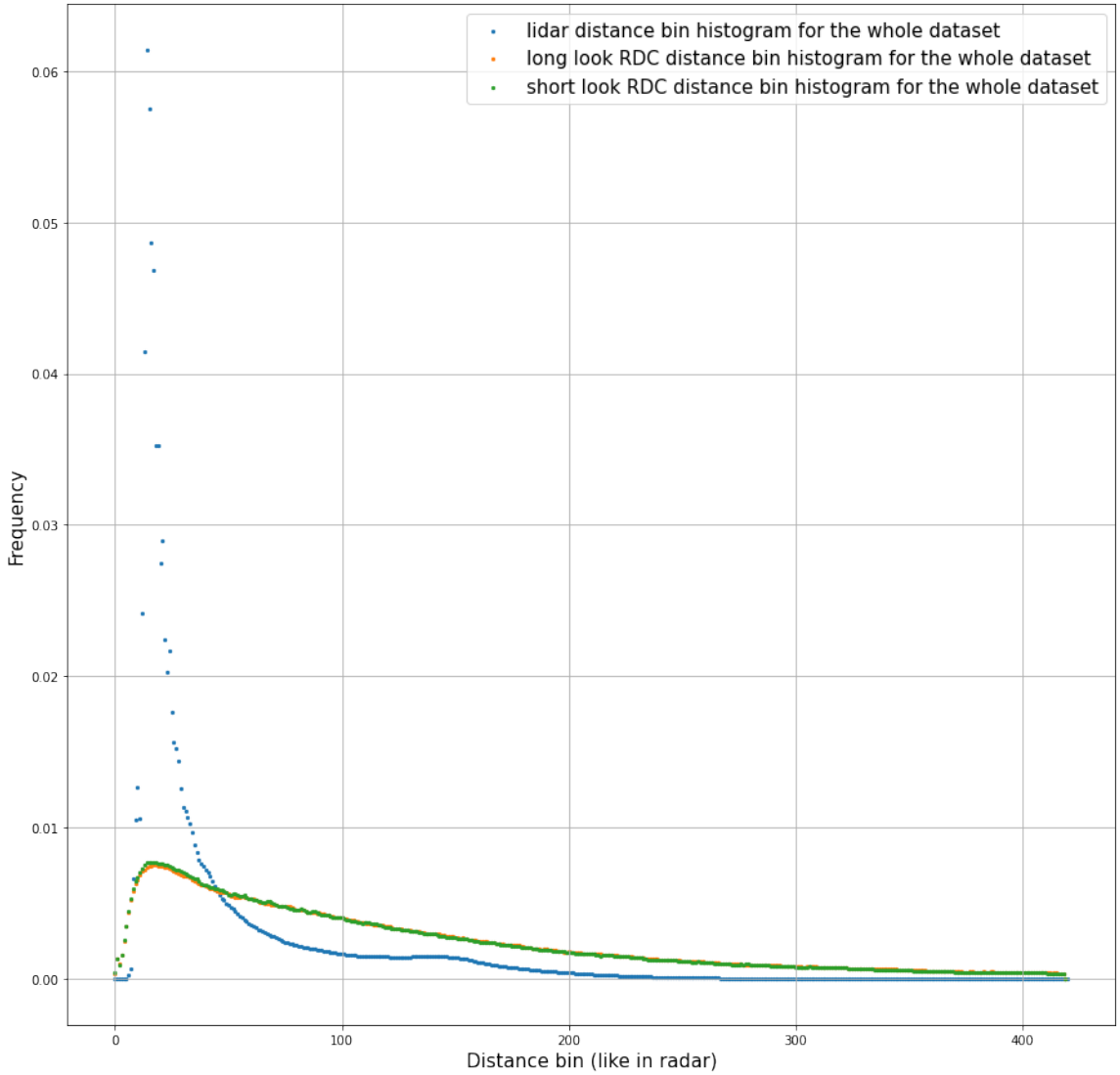


Figure 6.3. Histogram of the lidar-measured and radar-measured distances (bins like in radar), for all drives combined.

Table 6.1. Mean distances measured by sensors for different datasets.

	Highway drive 1	Highway drive 2	Urban drive 1	Urban drive 2	Urban drive 3	Urban drive 4	Urban drive 5	Whole dataset
Mean distance measured by lidar	30.5 m	31.5 m	26.5 m	24.4 m	22.8 m	28.6 m	28.7 m	27.9 m
Mean distance measured by radar (long look)	78.2 m	84 m	84.8 m	63.3 m	56.8 m	73.6 m	74.5 m	75.5 m
Mean distance measured by radar (short look)	73.3 m	78.7 m	80.4 m	59.6 m	53.4 m	69.7 m	70.37 m	71.1 m

6.3 Comparison of the Drives

There is a clear difference between the data collected during the highway drives (collected on a highway around Cracow and a highway between Katowice and Cracow) and urban drives (collected in Cracow). For one, the urban drives were recorded in a much more cluttered environment - this is evidenced by the fact, that the mean distances measured by the lidar during the urban drives are much shorter than the mean distances measured during the highway drives (Table 6.1). This is caused by the existence of more lidar-detectable obstacles near the car in the city, than on the highway. For radar, we don't see such a pronounced difference in the mean measured distances between urban and highway drives. It is caused by the fact, that the reasons for not detecting far-away objects are different between sensors. For the lidar, the dominant reason why the sensor does not detect an object far away is that the farther-away object is occluded by something nearer. On the other hand, the main effect limiting the number of radar measurements is the existence of empty RDC cells. The radar can unambiguously detect only one object at a specific distance and traveling at the same speed. Therefore, if there are some far-away objects in the radar FOV, the sensor will detect them, and the number of far-away radar measurements will not be greatly affected by the existence of closer objects.

Somewhat surprisingly, the existence of many objects cluttering the car surroundings in the city, does not translate into more final radar measurements (nonempty RDC cells), but less (see Table 6.2). This is probably caused by the CFAR thresholding - reflections from the multitude of cluttering objects are

raising the estimated noise level, and consequently decreasing the number of nonempty RDC cells. This, in turn, makes the task of radar depth completion harder in an urban setting.

Table 6.2. Mean number of non-empty RDC cells for different datasets.

	Highway drive 1	Highway drive 2	Urban drive 1	Urban drive 2	Urban drive 3	Urban drive 4	Urban drive 5	Whole dataset
Mean nonempty RDC bins (long look)	1277.6 1.2%	1780.7 1.7%	916.1 0.9%	666.8 0.6%	625 0.6%	816.2 0.8%	849.4 0.8%	938.2 0.9%
Mean nonempty RDC bins (short look)	1253.5 1.2%	1722.6 1.6%	916.8 0.9%	660.0 0.6%	618.2 0.6%	813.0 0.8%	846.3 0.8%	926.5 0.9%

Figures 6.4-6.6 show the histograms of the radar and lidar depth measurements for different drives. In the histograms for radar (6.4 and 6.5), we should notice that the differences in the number of measurements per frame between the drives are very pronounced, especially for reflections from objects located close to the sensor. On the other, the histograms for lidar measurements in different drives (Figure 6.6) are very similar to each other when it comes to the number of measurements. There is one qualitative difference for the highway drives - we can see an increase in the number of measurements between bins 125 and 150 (approximately 75 -100 m). These measurements probably represent reflections from cars that the data-collection vehicle is following.

Apart from the distance measurements, the radar provides also the velocity measurements. Their histograms for different drives are presented in Figures 6.7-6.8. The differences between the drives are very pronounced, practically every drive shows a qualitatively different distribution of measurements. Figure 6.9 shows the histograms of the velocity measurements from the radar short look and long look in one picture. We can see that the two histograms are qualitatively similar to each other.

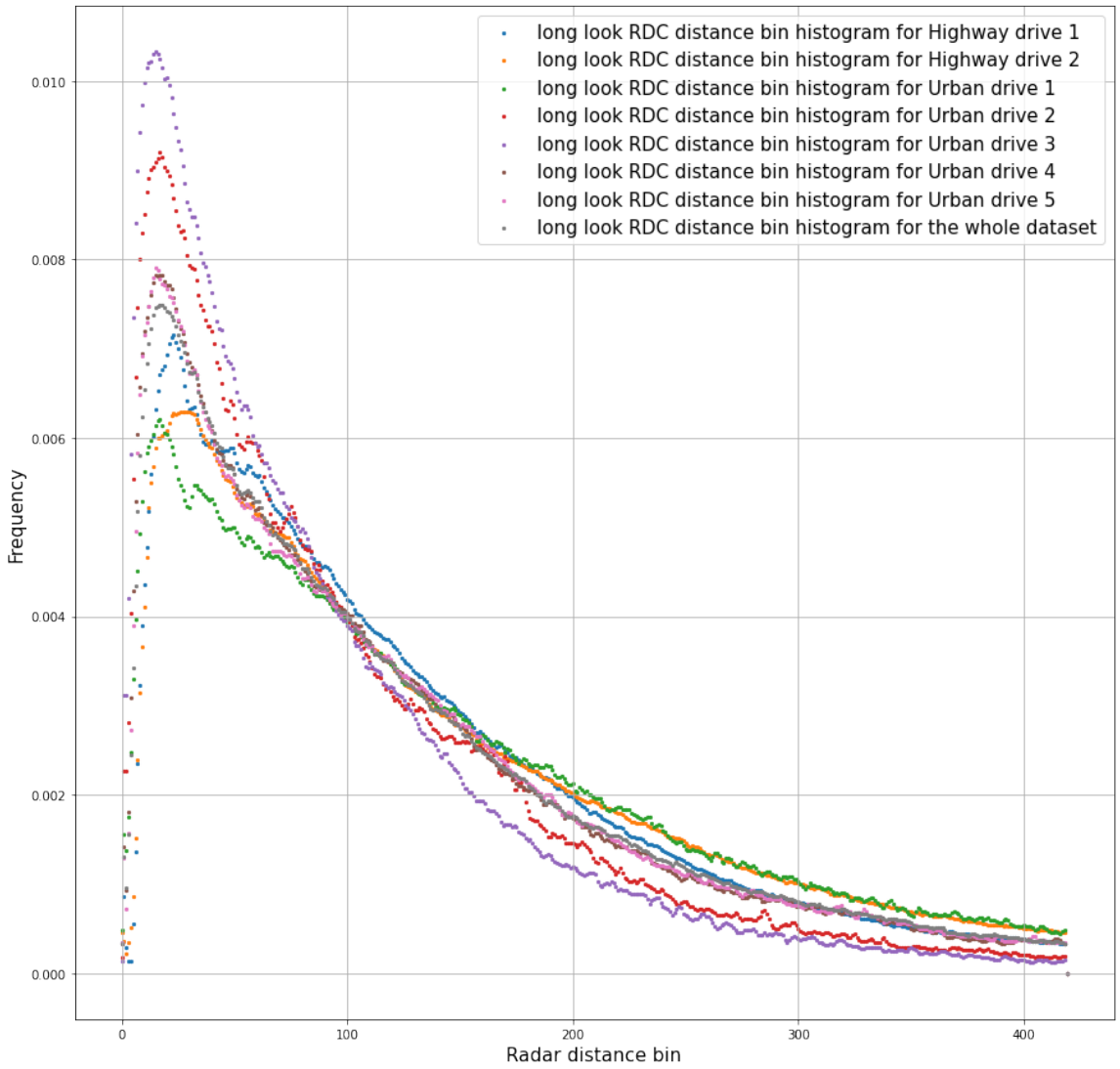


Figure 6.4. Histogram of the radar-measured distances for long look RDC, for different drives.

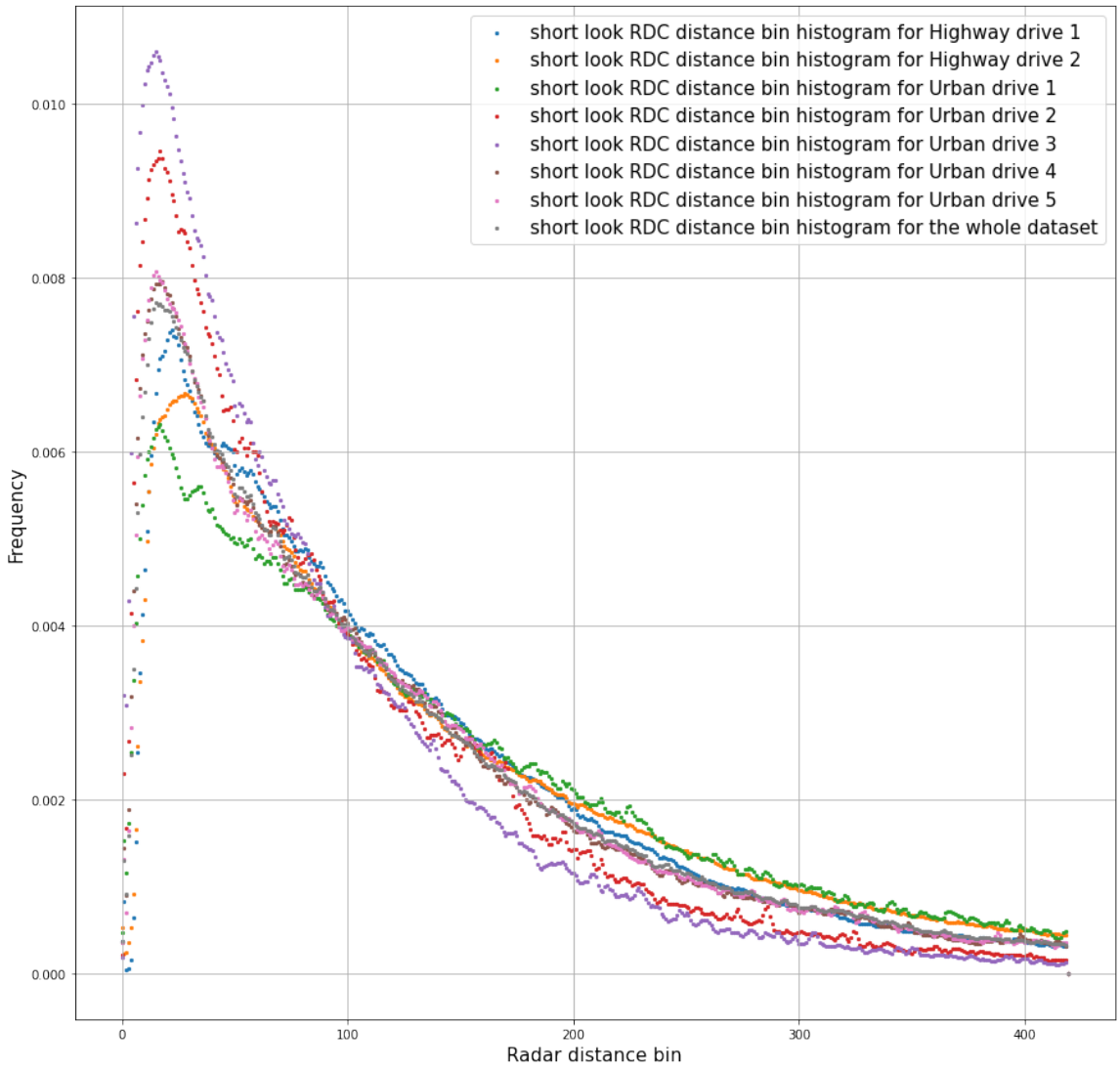


Figure 6.5. Histogram of the radar-measured distances for short look RDC, for different drives.

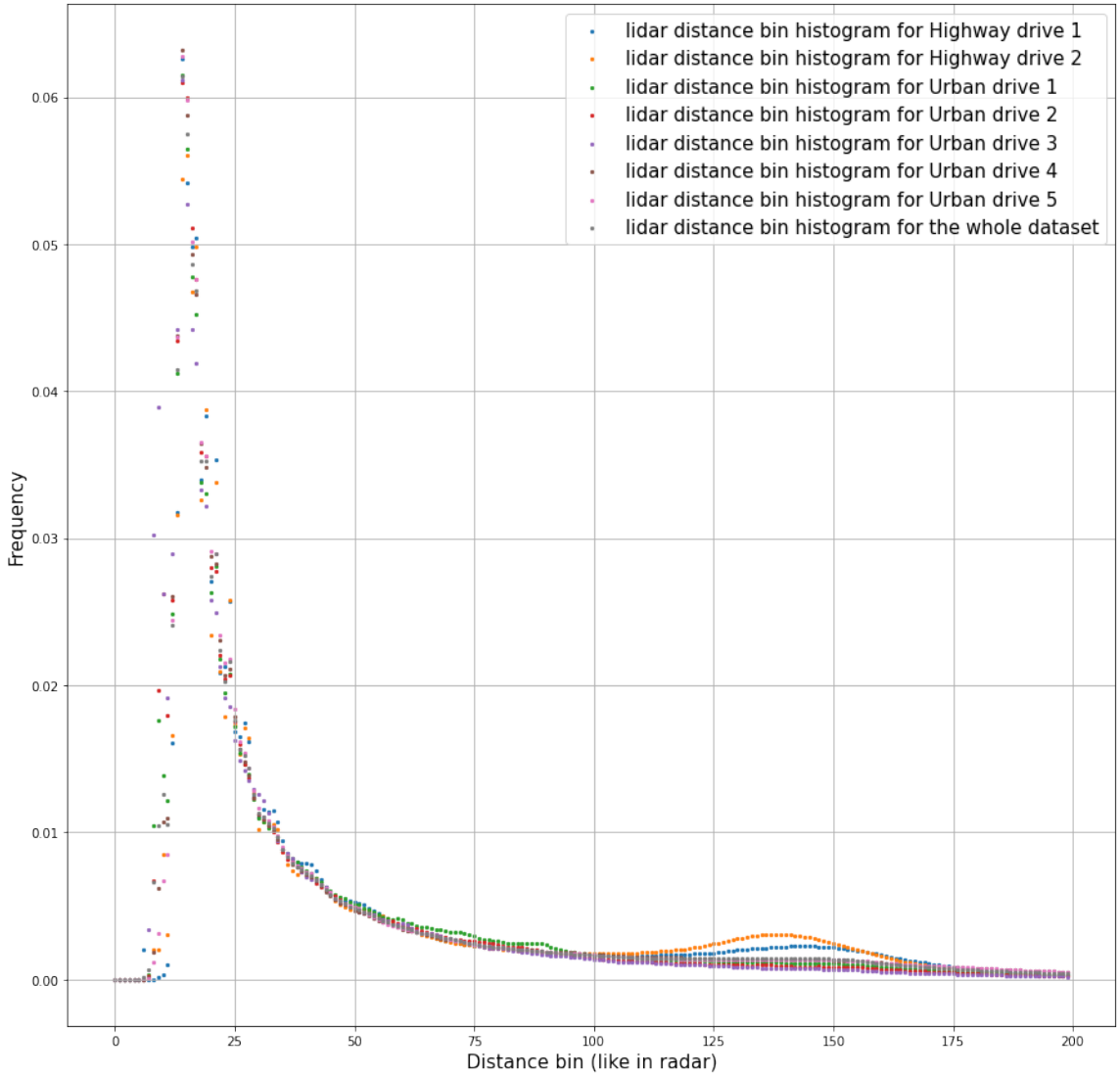


Figure 6.6. Histogram of the lidar-measured distances (bins like in radar), for different drives.

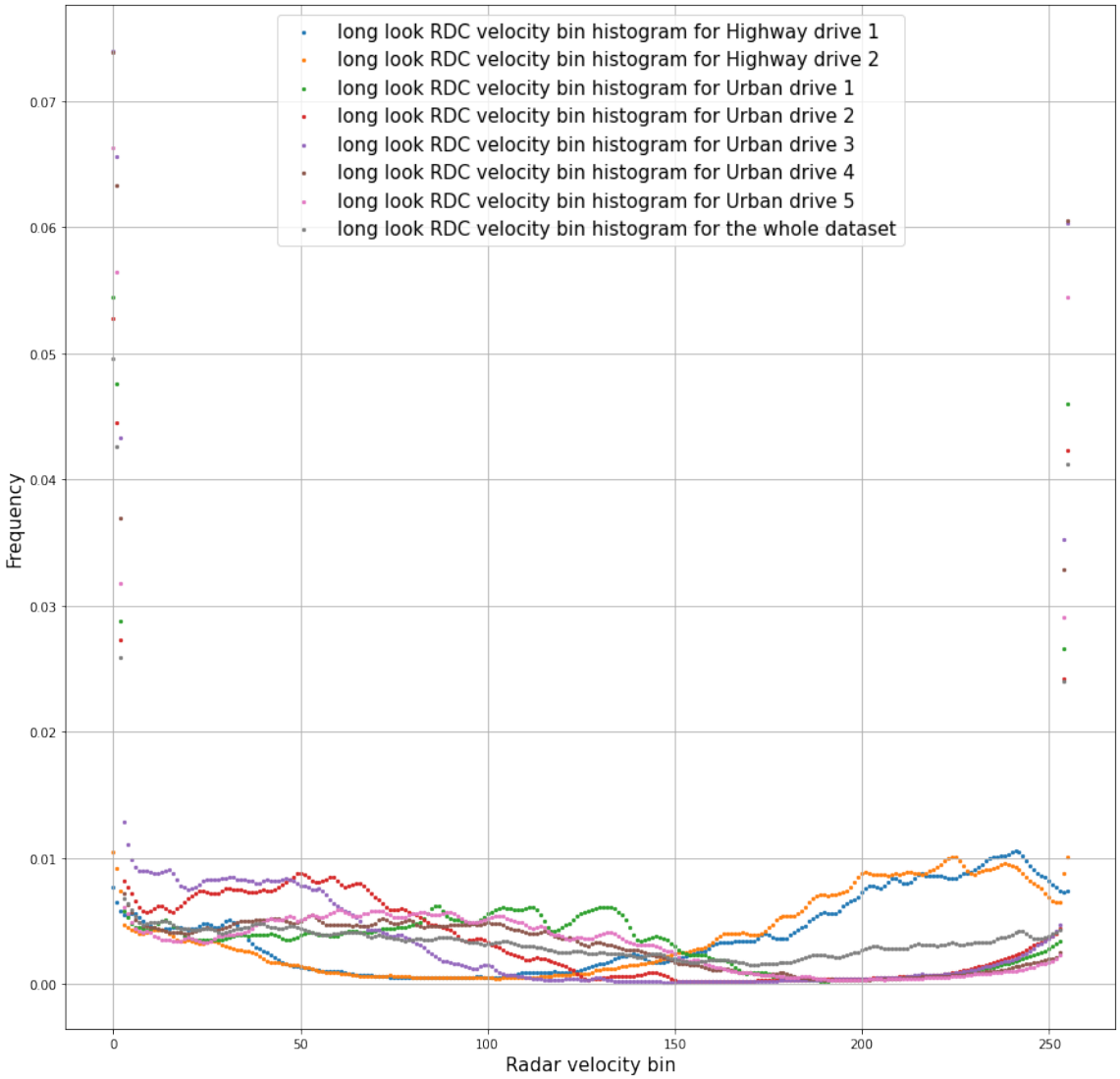


Figure 6.7. Histogram of the radar-measured velocities for long look RDC, for different drives.

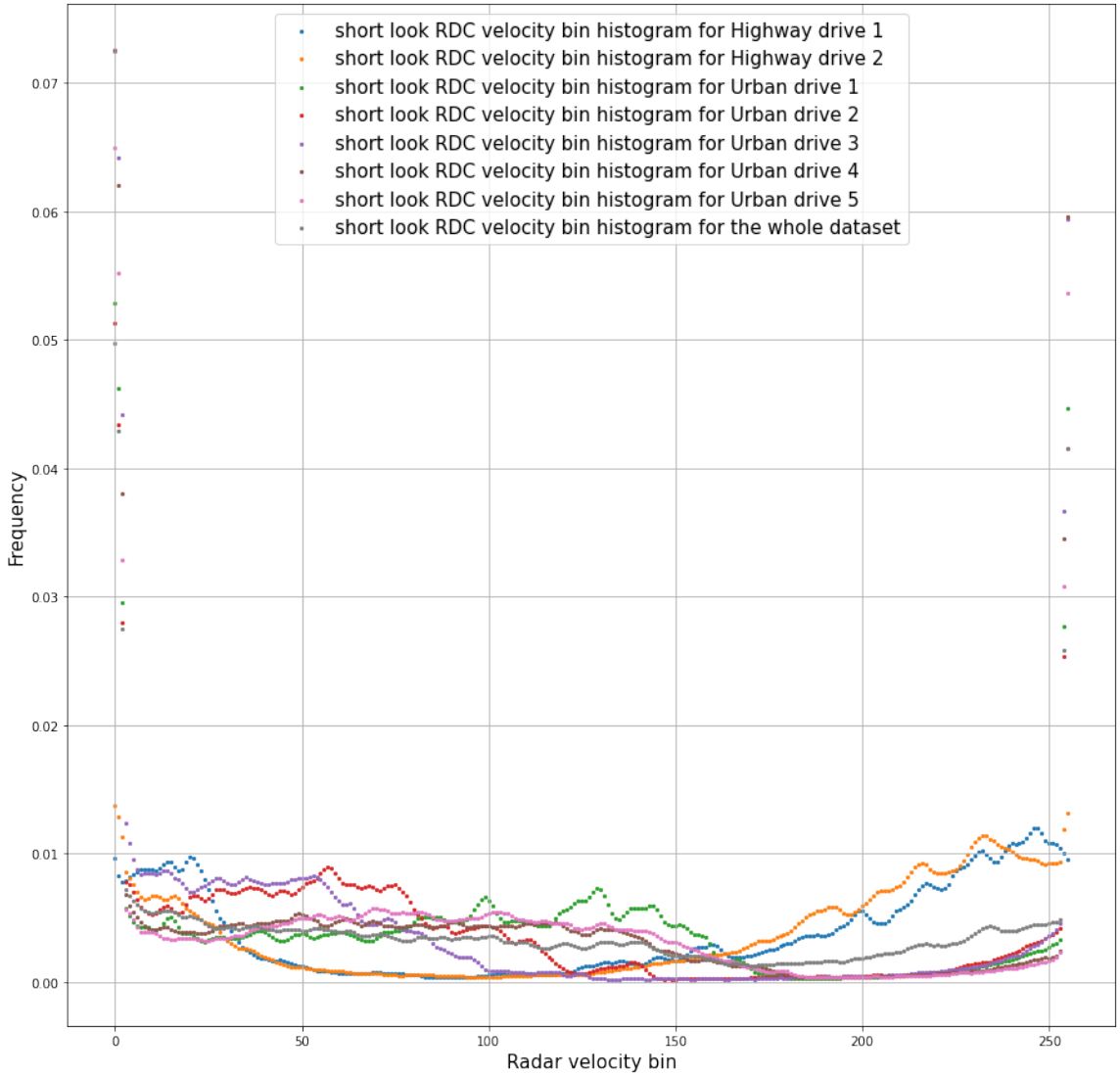


Figure 6.8. Histogram of the radar-measured velocities for short look RDC, for different drives.

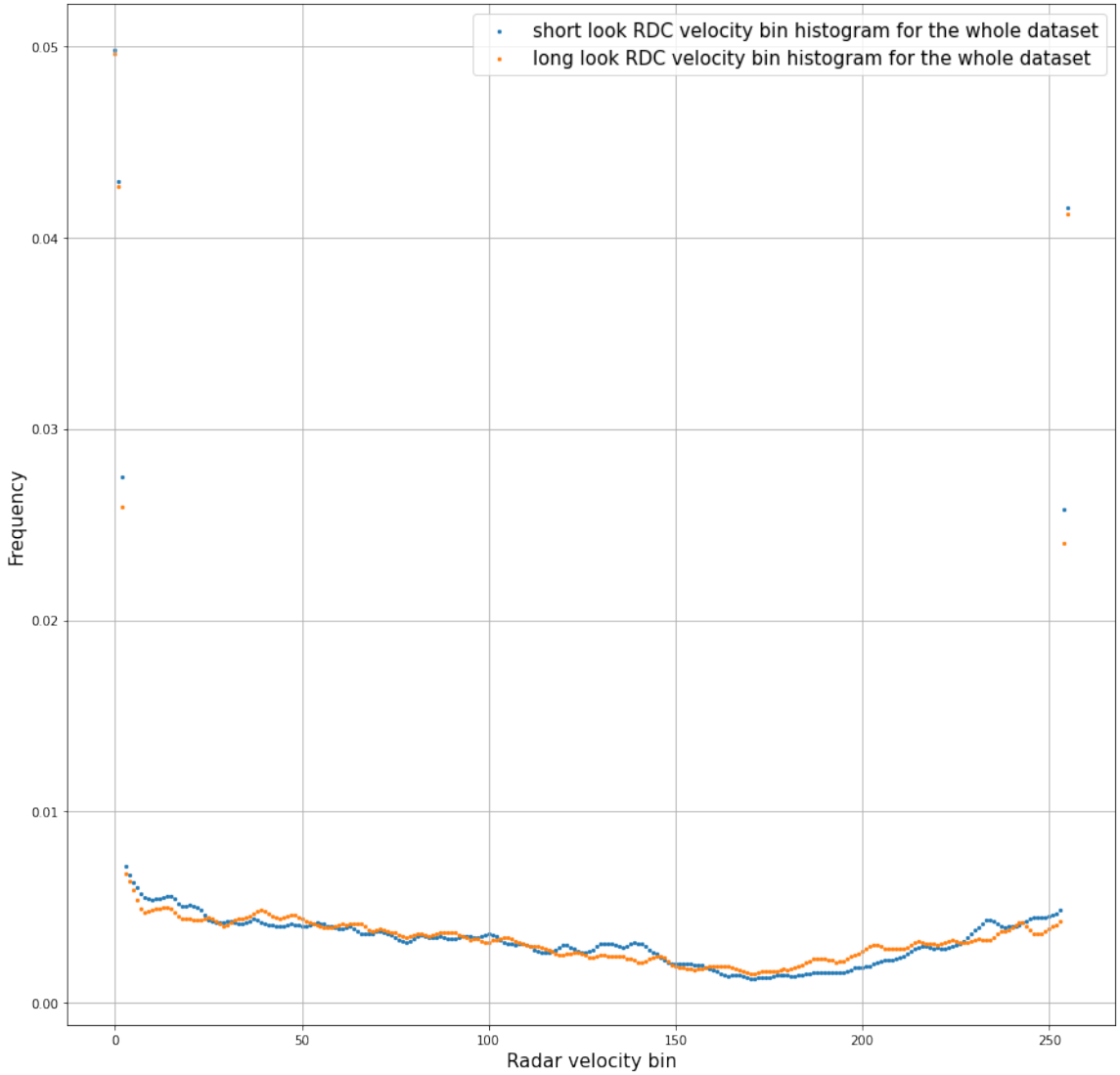


Figure 6.9. Histogram of the radar-measured velocities for long and short look RDC, all drives combined.

6.4 Conclusion

The created dataset is big and diverse. The biggest differences are between the urban and the highway drives. The distribution of the radar distance measurements is very different than the distribution of the lidar distance measurements. This is mainly caused by the fact that radar can detect only one unambiguous object for a given distance-velocity combination and by the CFAR thresholding. The size of the range bins is slightly different for the radar short look and the long look, but the distribution of the detections in bins is very similar in both operating modes. Similarly, the distribution of velocities measured in both radar operating modes is very similar.

7 Radar Depth Completion

7.1 Introduction

Producing a dense depth map on the basis of FMCW radar output is the main goal of this dissertation. In the introductory chapter (Chapter 1) I explained why it is important to produce it without utilizing an expensive lidar. In this chapter, I present a method to create dense depth maps using radar alone or radar fused with camera. To the best of my knowledge, this is the first solution producing a dense depth map on the basis of automotive FMCW radar RDC.

A detailed analysis of the dataset is presented in Chapter 6.

7.2 Related Work

7.2.1 Camera-only Solutions

When the camera takes a picture, information from the 3D world is projected into a 2D (azimuth-elevation) plane of the camera imager. It is impossible to directly recover depth information from this 2D representation, hence monocular depth estimation is an ill-posed problem. However, it is still possible to extract some depth information from images - KITTI Depth Prediction Challenge (Uhrig et al. [2017]) is a prime example of a dataset that deals with monocular depth estimation in an automotive setting. A more in-depth discussion of monocular depth estimation is provided in the chapter on automotive depth sensors, Section 2.4.2.

7.2.2 Radar-based and Radar+Camera-based Solutions

The main contribution of my work is a method of processing the RDC-level sparse output of an FMCW radar into a dense depth map. I am unaware of any other solutions dealing with radar depth completion (i.e. predicting dense depth map on the basis of sparse radar output) that are based on RDC.

There has been some work related to depth completion on radar point cloud. Due to poor quality of radar point clouds (especially when it comes to measuring elevation of a radar reflection), almost all of them utilize radar-camera fusion (e.g. Long et al. [2021], Lin et al. [2020], Lo and Vandewalle [2021], Singh et al. [2023], Zheng et al. [2022]). NuScenes dataset (Caesar et al. [2019]) is most frequently used for development of such methods.

Lin et al. [2020] project radar reflections onto a camera plane and later process them jointly with the RGB data using 2D convolutional neural networks. They test several fusion methods and provide ablation studies on the NuScenes dataset (Caesar et al. [2019]).

Lo and Vandewalle [2021] process the camera images using a state-of-the-art, Resnet-based (He et al. [2016]) monocular depth estimation network. In parallel they process the radar point cloud projected onto the camera plane using a Resnet-based encoder. At a later stage they fuse the two data streams, upsample and predict depth in a quantized fashion. They also utilized the NuScenes dataset (Caesar et al. [2019]).

Zheng et al. [2022] also use parallel convolutional processing of camera image and radar reflection projection on the camera plane. Additionally, they perform semantic segmentation of the camera image and fuse the semantic segmentation stream back into the depth estimation stream. They also used the NuScenes dataset (Caesar et al. [2019]).

Long et al. [2021] take note of the limitations of radar point cloud, that make it hard to accurately project the reflections onto the camera plane. Instead, they associate radar reflections with pixels in a one-to-many fashion. Later, they represent the fused radar and camera frames as RGB image with additional channels representing confidence that depth is in a particular range. Their experiments were performed on the NuScenes dataset (Caesar et al. [2019]).

Singh et al. [2023] took approach similar to that of Long et al. [2021]. They also utilized a 2 stage depth estimation network, which first associated radar reflections with particular pixels on camera, and later performed depth estimation itself. They trained and tested on the NuScenes dataset (Caesar et al. [2019]).

The previously mentioned methods rely very heavily on camera and in practice use radar only as an addition to monocular depth estimation.

Xu et al. [2022b] did not use radar point cloud to enhance depth estimation based on a camera image, but rather developed a stand-alone radar solution. They enhance the output of an imaging radar used for indoor mapping. Their represent their radar point cloud in 3D spherical coordinates (azimuth x elevation x range), with additional channels containing reflection intensity and relative velocity. Then, they use a convolutional neural network to produce a depth map in cylindrical coordinates. The output of their network has very low elevation resolution and hence is more suitable for tasks such as obstacle

detection, rather than scene understanding. Please also note, that imaging radars produce output of much higher quality than typical automotive FMCW radars.

On a related note, there are Synthetic Aperture Radars (SAR) (Moreira et al. [2013]) which produce dense depth maps as a result of their operations, however, they are completely unrelated to my method, both in terms of method of operation and their applications. SAR uses the movement of a radar (e.g. on a plane or a satellite imaging ground) to create an imaginary radar with a large antenna and in this way can produce high resolution, dense depth map (Moreira et al. [2013]).

My solution for radar depth completion is significantly different than all the previously described methods for two main reasons. Firstly, it is capable of producing a dense depth output using only the output of an automotive radar, without the need of a corresponding camera image. Moreover, using radar alone, it is able to produce depth maps that can be used for scene understanding. Secondly, it operates on RDC-level data, an earlier step in radar signal processing than the pointcloud, that is used by other published methods.

7.3 My Solution

In this section, I use the dimensions of the network operating on 3 short-range RDCs and 3 long-range RDCs (RDC structure is described in Section 2.2.2). In this chapter I present also the results of a network using 4 short-range RDCs and 4 long-range RDCs - in that case, only the first dimension is affected (changing from 3 to 4 before the fusion module).

7.3.1 Inputs

The main idea behind my solution was to create a trainable module capable of processing RDC data and transforming them into the azimuth-elevation plane. The radar-vision fusion network has 3 inputs:

- Stacked RDCs from the long-range radar operating mode - it has dimension [3,420,256,24] which corresponds to 3 stacked RDCs, 420 range bins, 256 relative velocity bins and 24 antennae channels (12 antennae, signal at each one described by a complex number).
- Stacked RDCs from the short-range radar operating mode (dimension [3,420,256,24])
- Stacked corresponding RGB camera images - it has dimensions [3,720,1280,3], which corresponds to 3 stacked images, 720 height pixels, 1280 width pixels and 3 RGB channels.

I present 2 versions of the network - one utilizing only the low-level radar data, and another one fusing the low-level radar data with RGB images in the azimuth elevation plane. Moreover, in the ablation studies, I present a version of the network in which the stream processing radar data is blocked and

depth predictions are made solely on the basis of the RGB images, to determine to what extent radar is beneficial when it is fused with the RGB images from camera.

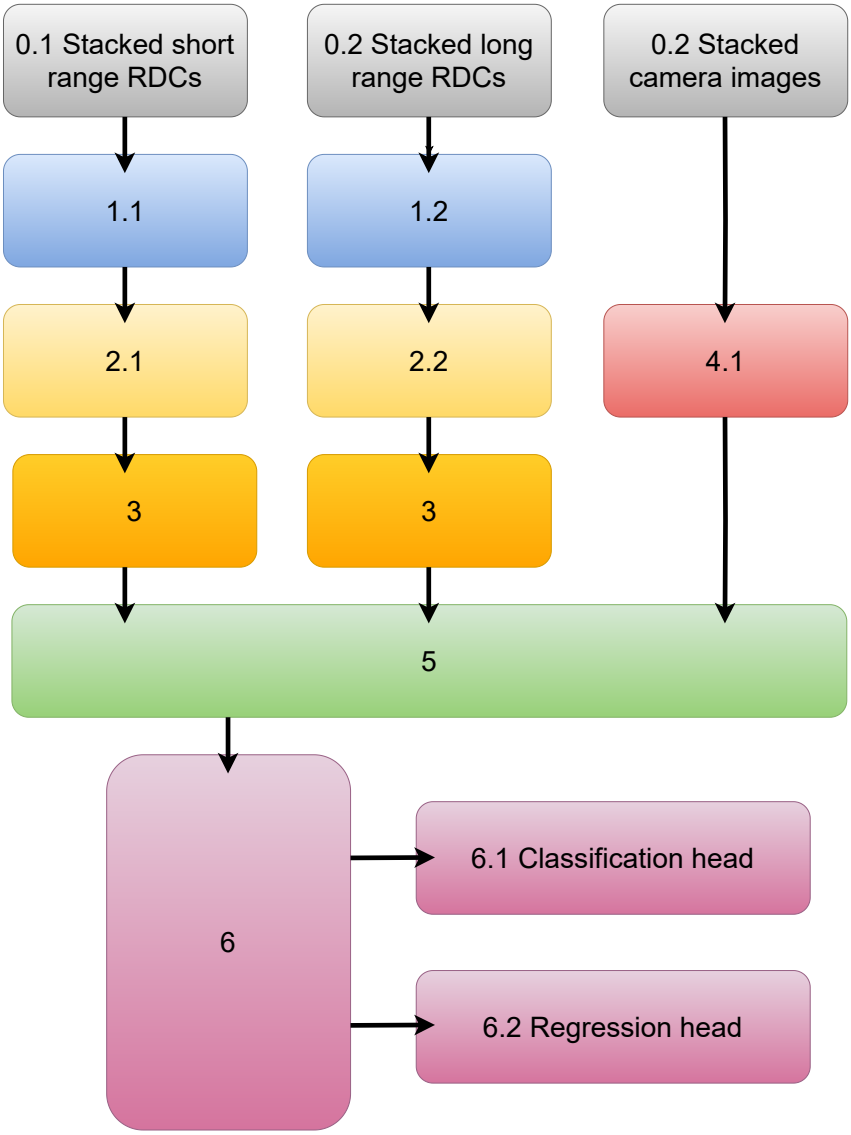


Figure 7.1. Diagram of the high-level network structure. Elements of the diagram are numbered according to the convention from subsection 7.3.2. In the case of the elements whose number consists of 2 digits (e.g. 1.1), the second digit is introduced to emphasize that the multiple elements sharing the same leading digit do not share weights.

7.3.2 Network Architecture

My neural network (shown in Figure 7.1) may be logically divided into 6 modules (not counting inputs as one of them), they are enumerated below.

0. Inputs, as described in subsection 7.3.1.
1. Trainable angle-finding module.
2. Trainable encoder for the range and relative velocity dimensions of the RDC.
3. Fixed module dealing with reshaping the output of the RDC encoder to project it onto a 2D azimuth-elevation plane.
4. Trainable encoder for the camera images.
5. Trainable sensor and temporal fusion module.
6. Trainable module jointly processing the encoded RDC data and encoded camera image in the azimuth-elevation plane.

In the case of the radar-only experiments, the network structure was kept, but the vision processing stream was fed with constant dummy input. Similarly, in the case of vision-only experiments, the network structure was kept, but the radar processing stream was fed with constant dummy input.

In the diagrams in the following sections (describing network modules in more detail), trivial layers, such as those whose purpose is cropping the image into the desired shape, are omitted. Likewise, the temporal axis (axis storing the data from consecutive RDCs and camera images) is omitted to avoid cluttering the diagrams.

7.3.2.1 Trainable Angle-Finding Module

As mentioned in the section on FMCW Radars (Section 2.2.2) and later shown in Chapter 5, it is possible to find the angle to the target by comparing the phase of the reflected radiowave between the receiving antennae. The information about the received radiowave on different antennae is recorded in the third dimension of the RDC, therefore the angle-finding module is implemented as a set of convolutional layers with a 1x1 kernel.

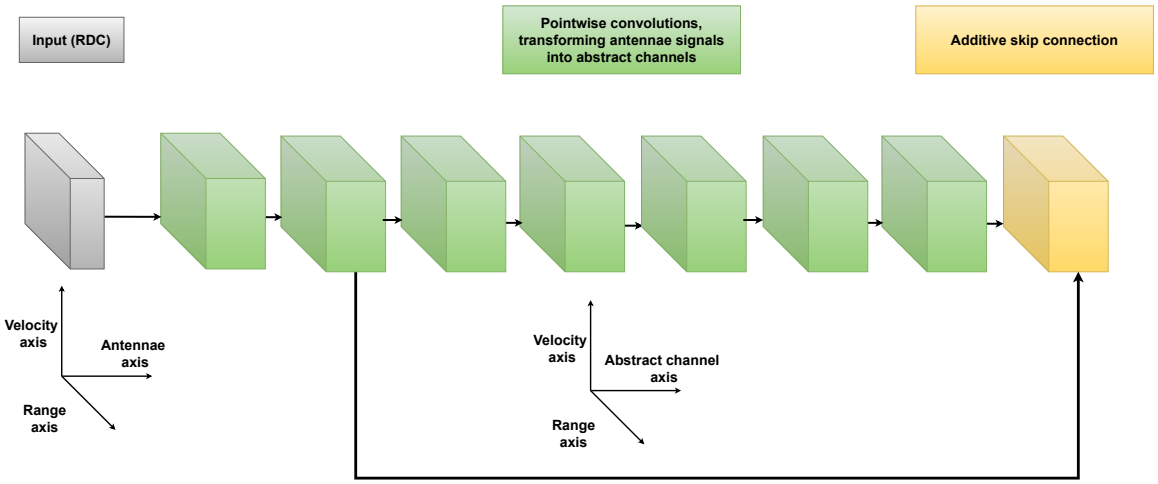


Figure 7.2. Diagram of the trainable angle-finding module.

The module (represented in Figure 7.2) effectively operates on each beamvector separately and projects the last dimension of RDC into an abstract, higher dimensional space. Input to this module has dimension [3, 420, 256, 24] and output has dimension [3, 420, 256, 96].

7.3.2.2 Trainable Encoder for Range and Relative Velocity

RDC of my radar contains 420 range bins and 256 relative velocity bins, which corresponds to 107,520 range-velocity combinations. In practice, the overwhelming majority of the combinations (usually more than 99%) contain empty beamvectors - signals for that particular range-velocity combinations were below the noise threshold at lower-level data processing in the radar. This module of the network (presented in Figure 7.3) deals with encoding the RDC data into a denser representation. It is performed in 3 steps.

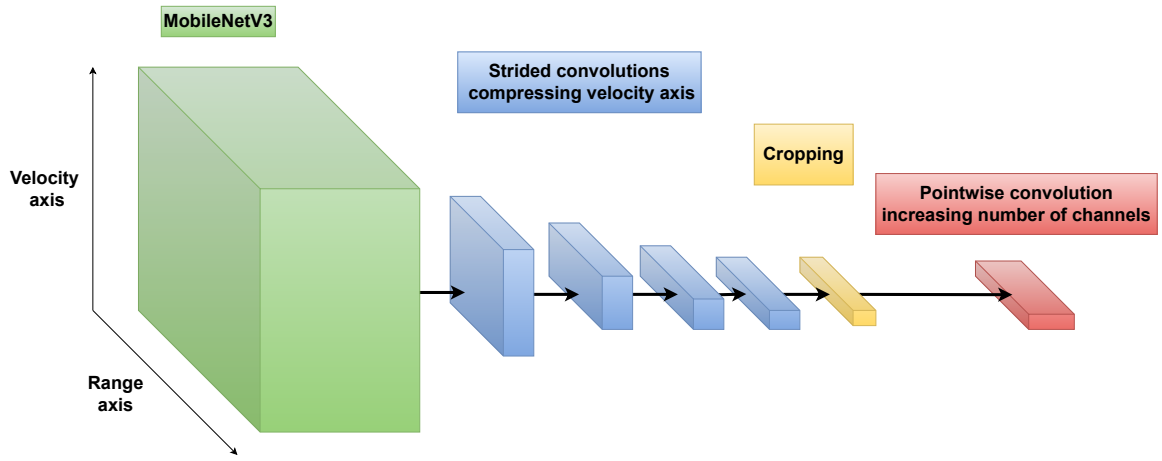


Figure 7.3. Diagram of the trainable encoder for range and relative velocity

In the first step, a 2D convolutional neural network, relying heavily on MobileNetV3 (Howard et al. [2019]), operates on each RDC separately and transforms it from dimension [420, 256, 96] into dimension [14, 8, 576]. The first 2 dimensions are treated as spatial dimensions and the last one as channels by the 2D convolutions. Beamvectors from the neighboring range and velocity bins likely come from reflections from the same object. Treating the first 2 dimensions of RDC as spatial dimensions and applying 2D convolutions to such input allows the network to learn interactions between beamvectors from neighboring cells in the RDC (an example RDC can be seen in Figure 2.8).

The second step of this module uses 2D convolutions with appropriate strides to compress the dimension corresponding to the velocity information. The task of depth prediction does not rely directly on the velocity information, hence I decided to reduce the amount of information about velocity to conserve computational resources. This part of the encoder also operates on each RDC separately, its input has dimension [14, 8, 576] and output [14, 1, 1024].

The last encoder processing step uses cropping and pointwise convolution to transform the shape of the output into [11, 1, 33280].

In total, the encoder for the range and relative velocity dimensions accepts input of dimension [3, 420, 256, 96] and produces output of dimension [3, 11, 1, 33280].

7.3.2.3 Reshaping the Data into Azimuth-Elevation Plane

This module (represented in Figure 7.4) may be the most critical innovation in the network. It applies reshaping and permutation operations to the RDC-based data in order to transform them into the azimuth-elevation plane. For a single RDC, this module performs the following steps:

1. Reshape input of dimension [11, 1, 33280] into data of dimension [11, 8, 40, 104] - at this stage, the dimension containing range information (the one with 11 bins) stays as it was, abstract channels are divided into 3 groups.
2. Permute the order of the axis to get output of shape [40, 104, 11, 8]
3. Reshape the data again, this time to the form [40, 104, 88], where the first 2 dimensions represent height and width in the azimuth-elevation plane and the last dimension are abstract channels.

Please note that the information from the dimension that used to contain range information is propagated to all the pixels, thus enabling efficient learning for the depth prediction task.

This module operates on 3 RDCs in total, so its input dimension is [3, 11, 1 ,33280] and output dimension is [3, 40, 104, 88].

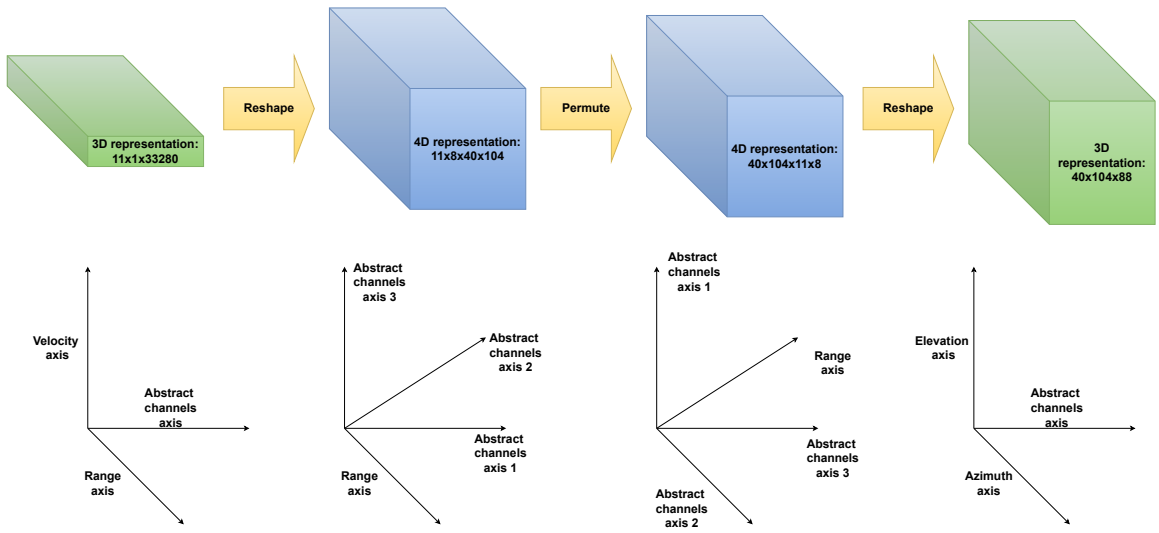


Figure 7.4. Diagram of the fixed module dealing with reshaping the output of the RDC encoder to project it into a 2D azimuth- elevation plane. In this diagram, the cuboids correspond to data representation between layers, rather than layers (layers are shown as arrows).

7.3.2.4 Trainable Encoder for Camera Images

This module operates separately on each camera image fed to the network. It utilizes a MobilenetV3-based (Howard et al. [2019]) network along with some image cropping to extract features from the camera images and transform the data from shape [3, 720, 1280, 3] into shape [3, 40, 104, 80].

7.3.2.5 Trainable Sensor and Temporal Fusion Module

This module (represented in Figure 7.5) first deals with fusing the data from separate streams (radar short-range look, radar long-range look, camera) for a specific frame, and later performs temporal fusion - that is, fuses the data from all the frames into one output.

The first operation in the module is concatenating the 3 streams of data (dimensions [3, 40, 104, 88], [3, 40, 104, 88], [3, 40, 104, 80]) along the last axis to get output of dimension [3, 40, 108, 256].

Then, the data are processed using another neural network based heavily on MobileNetV3 (Howard et al. [2019]) (the first axis is treated as batch dimension so each frame is processed separately), to get the output of dimension [3, 20, 52, 576].

The temporal fusion is performed by 7 consecutive Conv2D Gated Recurrent Unit (GRU) layers (Chung et al. [2014]) (there is a concatenate skip connection enveloping 4 of them). The first 6 layers output sequences, last one collapses the output to a single frame of shape [20, 52, 256]. In order to make the GRU layers more computationally efficient, I hard-coded the number of frames (instead of the usual implementation allowing for a varying number of frames).

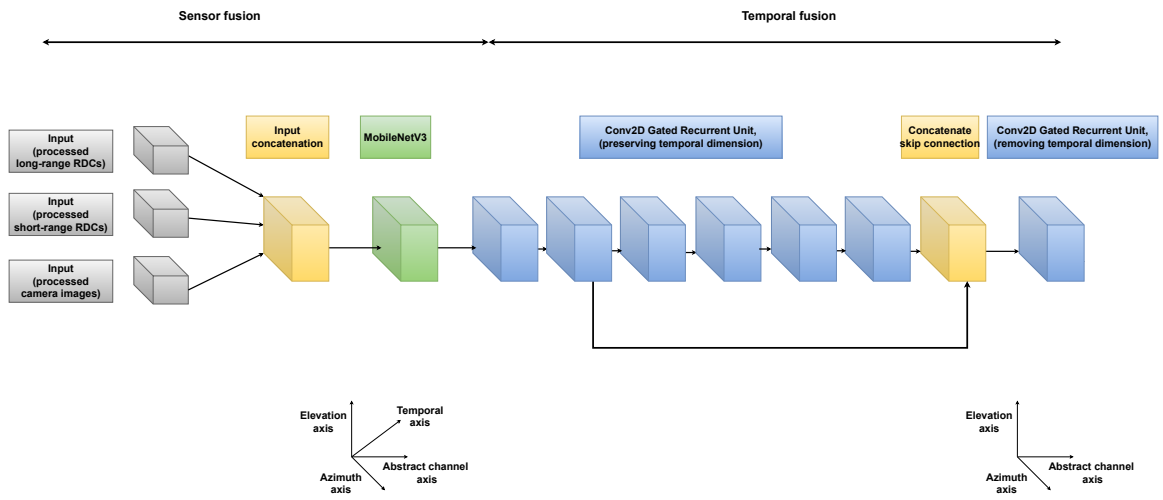


Figure 7.5. Diagram of the sensor and temporal fusion module.

7.3.2.6 Joint Processing in the Azimuth-Elevation Plane

The last part of the network (represented in Figure 7.6) operates using 2D convolutional layers on the data in the azimuth-elevation plane. The network produces 2 outputs - classification, where it classifies each pixel into one of 421 range bins (each roughly 0.6 m wide), corresponding to the range bins of the radar, and a regression output, producing a depth map directly. Outputs have dimensions [65, 176, 421] and [65, 176, 1] respectively.

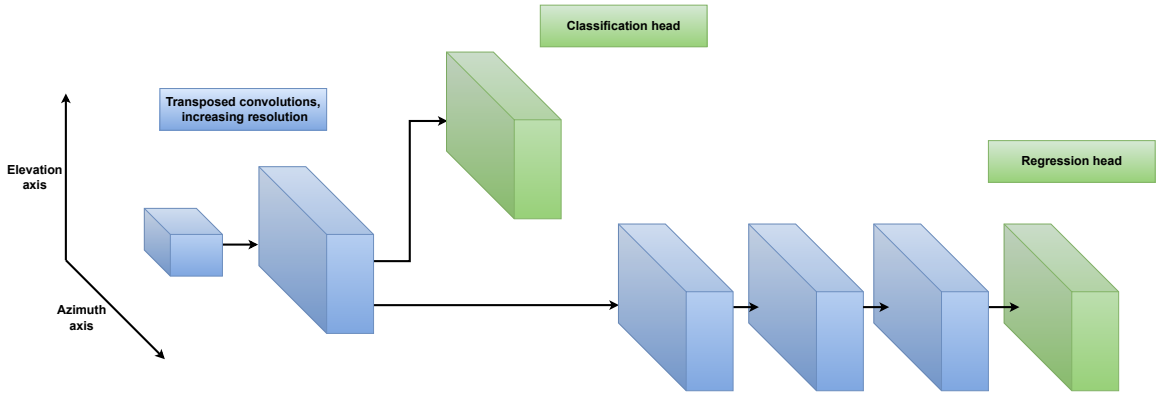


Figure 7.6. Diagram of the module processing data in azimuth-elevation plane and network heads.

7.3.3 Training

All the networks were trained for 200 epochs. Each epoch consisted of training on 5,000 training examples sampled from the training set. For the purpose of sampling, the random seed was each time set to the number of the epoch. Adam (Kingma and Ba [2014]) was used as the network optimizer. The learning rate was decreased throughout the training according to the following schedule:

$$\mathrm{lr}(i) = 10^{-4} \cdot 0.972^{i}. \tag{7.3.1}$$

Trainings were performed using a focal loss (Lin et al. [2018]) for the classification head and Huber loss (switching at 20 m) for the regression head. Focal loss was calculated according to (Lin et al. [2018]):

$$\mathrm{FL}(p_t) = \beta(1-p_t)^{\gamma}\log(p_t), \tag{7.3.2}$$

where:

$$p_t = \begin{cases} p & \text{if } y > 0 \\ 1-p & \text{otherwise} \end{cases}. \tag{7.3.3}$$

β was set to 500 and γ was set to 2.

7.4 Results

I present the results for 4 versions of the network:

- Radar-only network, utilizing 3 RDCs from short-range look and 3 RDCs from long-range look (6 RDCs in total).
- Radar-only network, utilizing 4 RDCs from short-range look and 4 RDCs from long-range look (8 RDCs in total).
- Radar + Vision network, utilizing 3 RDCs from short-range look, 3 RDCs from long-range look (6 RDCs in total) and an equivalent number of video frames (i.e. video frames closest to the collection time of RDCs).
- Vision-only network, utilizing a number of video frames equivalent to 6 RDCs in total. This network shares the structure with Radar + Vision network to make it possible to estimate what is the influence of radar data on KPIs in case of fusion with the camera.

I decided to use Relative L1 error (see Equation (7.4.1)) at the output of the regression head as my main KPI. It is an easily interpretable measure that is also used in KITTI Depth Prediction Challenge (Uhrig et al. [2017]) (where the goal is to estimate depth using RGB camera images). Relative L1 error was calculated according to the formula (7.4.1):

$$\text{Relative L1}(y_{true}, y_{pred}) = \frac{L1(y_{true}, y_{pred})}{y_{true}}, \tag{7.4.1}$$

where L1 is defined in a way analogous to the norm in the $L^{p=1}$ space:

$$\text{L1}(y_{true}, y_{pred}) = |y_{true} - y_{pred}|, \tag{7.4.2}$$

where y_{true} is the true distance and y_{pred} is the predicted distance.

The results for different parts of the dataset (different drives) were different enough, that I decided to show the KPI results for each drive separately. I present the results for uncapped distance (up to approximately 250 m) and capped distance (capped at 70 m) in tables 7.1, 7.2 respectively. I present the results for distance capped at 70 m to both show the performance of my networks at closer distances, which are more relevant from the perspective of Advanced Driver Assistance Systems (ADAS) algorithms and to facilitate comparison with KITTI Depth Prediction Challenge (Uhrig et al. [2017]) results. Apart from the numerical results, I also show the output of the networks for an example frame from an urban drive and an example frame from a highway drive (Figures 7.7-7.20).

Table 7.1. Relative L1 results for different datasets (no distance cap).

Dataset	Radar only 6 RDCs	Radar only 8 RDCs	Radar+Vision 6 RDCs	Vision only	Percentage improvement over vision only
Highway drive 1	0.2803	0.1414	0.0842	0.0895	5.92%
Highway drive 2	0.161	0.1553	0.1076	0.1164	7.56%
Urban drive 1	0.2699	0.2862	0.1135	0.1207	5.97%
Urban drive 2	0.3462	0.2265	0.1087	0.1121	3.03%
Urban drive 3	0.2789	0.2914	0.1020	0.1162	12.22%
Urban drive 4	0.5179	0.4112	0.1057	0.1347	21.53%
Urban drive 5	0.3847	0.2546	0.1048	0.1114	5.92%

Table 7.2. Relative L1 results for different datasets (distance capped at 70 m).

Dataset	Radar only 6 RDCs	Radar only 8 RDCs	Radar+Vision 6 RDCs	Vision only	Percentage improvement over vision only
Highway drive 1	0.2505	0.1114	0.0600	0.0664	9.64%
Highway drive 2	0.1365	0.1247	0.0857	0.0928	7.65%
Urban drive 1	0.2553	0.2647	0.0971	0.1047	7.26%
Urban drive 2	0.3284	0.2046	0.0927	0.0958	3.24%
Urban drive 3	0.2679	0.2764	0.0916	0.1055	13.18%
Urban drive 4	0.4935	0.3766	0.0858	0.1149	25.33%
Urban drive 5	0.3598	0.2268	0.0814	0.0891	8.64%

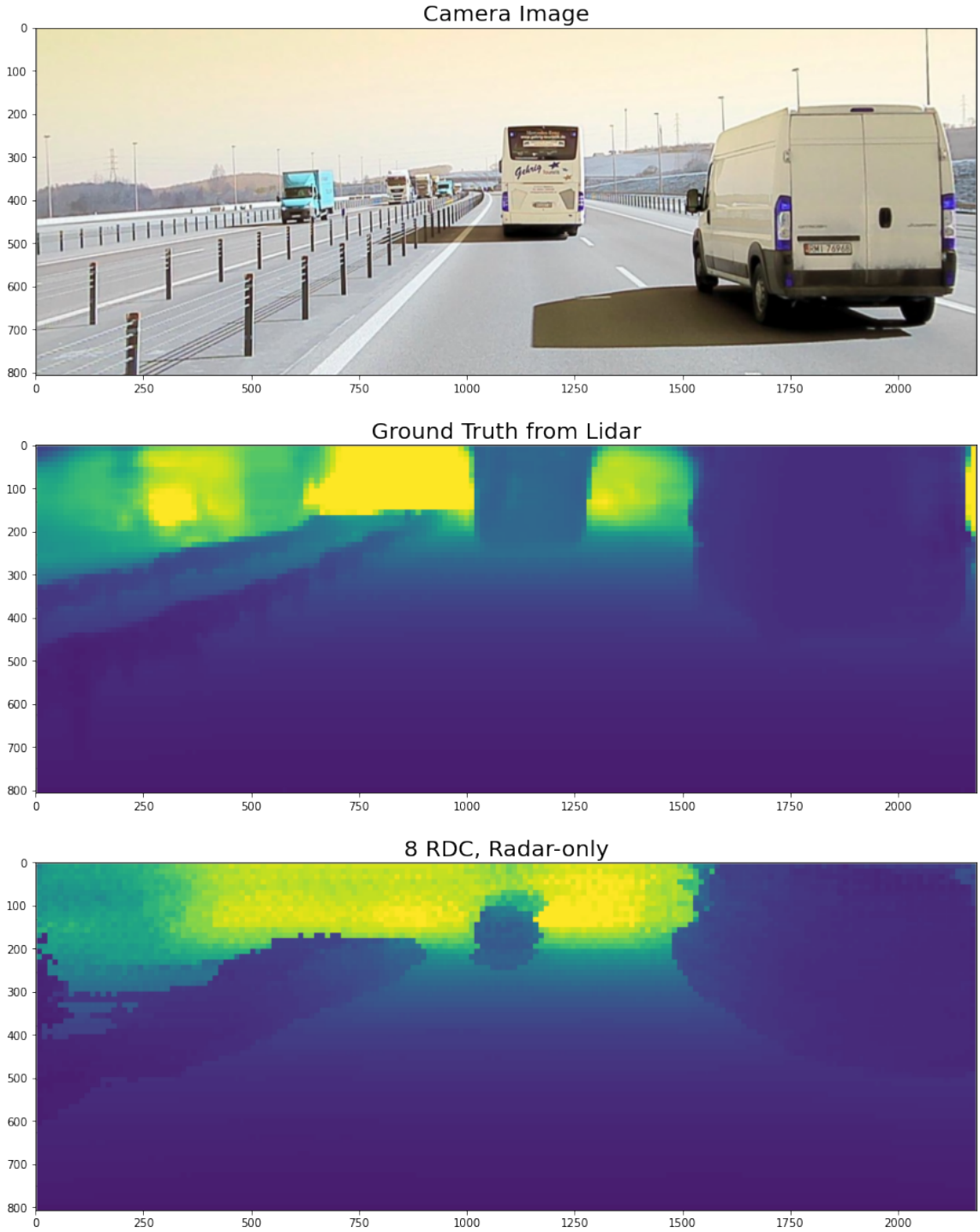


Figure 7.7. Comparison of the output of different versions of the network, frame from Highway Drive 1 (part 1). Depth is color coded, numbers on the axes denote pixel coordinates.

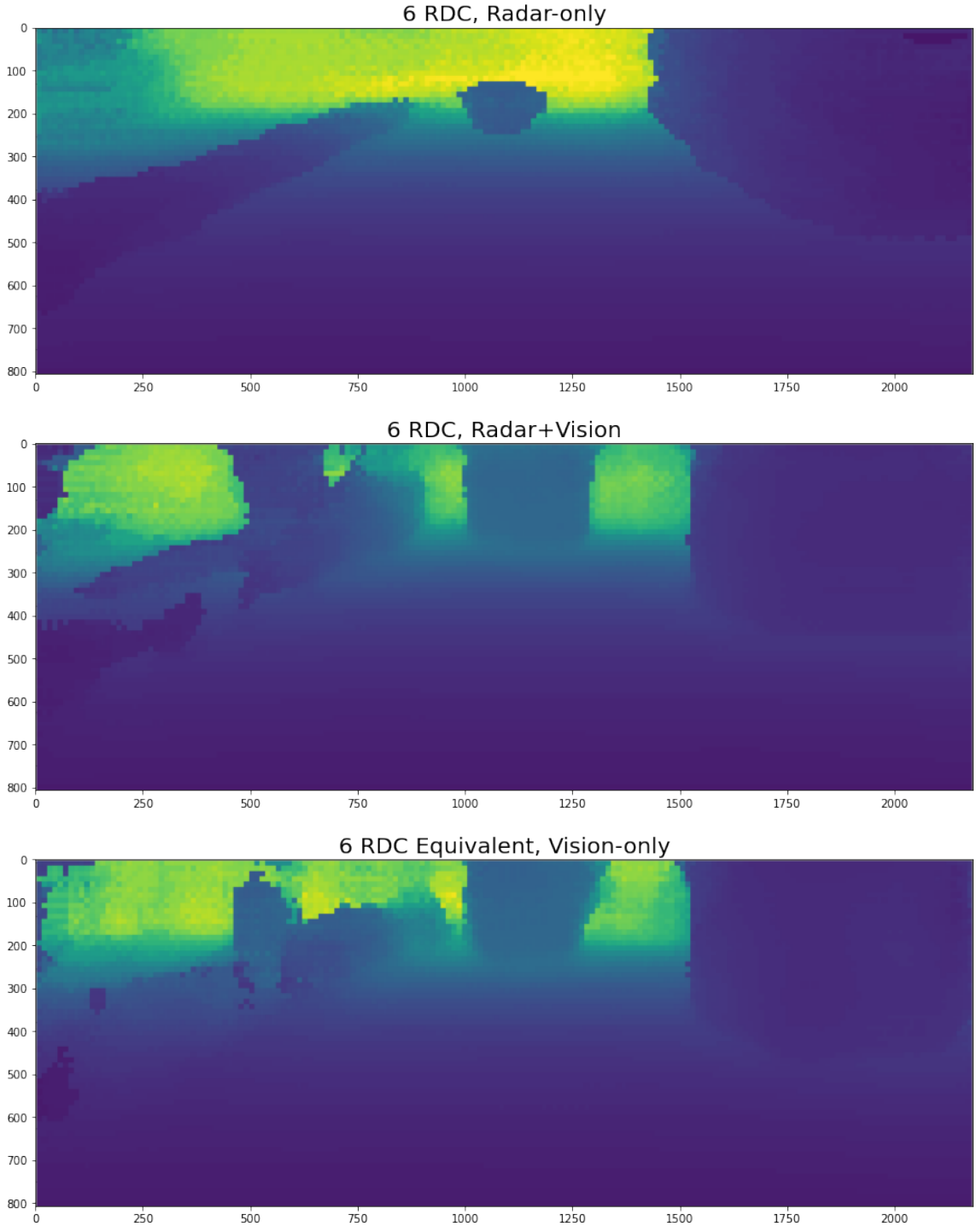


Figure 7.8. Comparison of the output of different versions of the network, frame from Highway Drive 1 (part 2). Depth is color coded, numbers on the axes denote pixel coordinates.

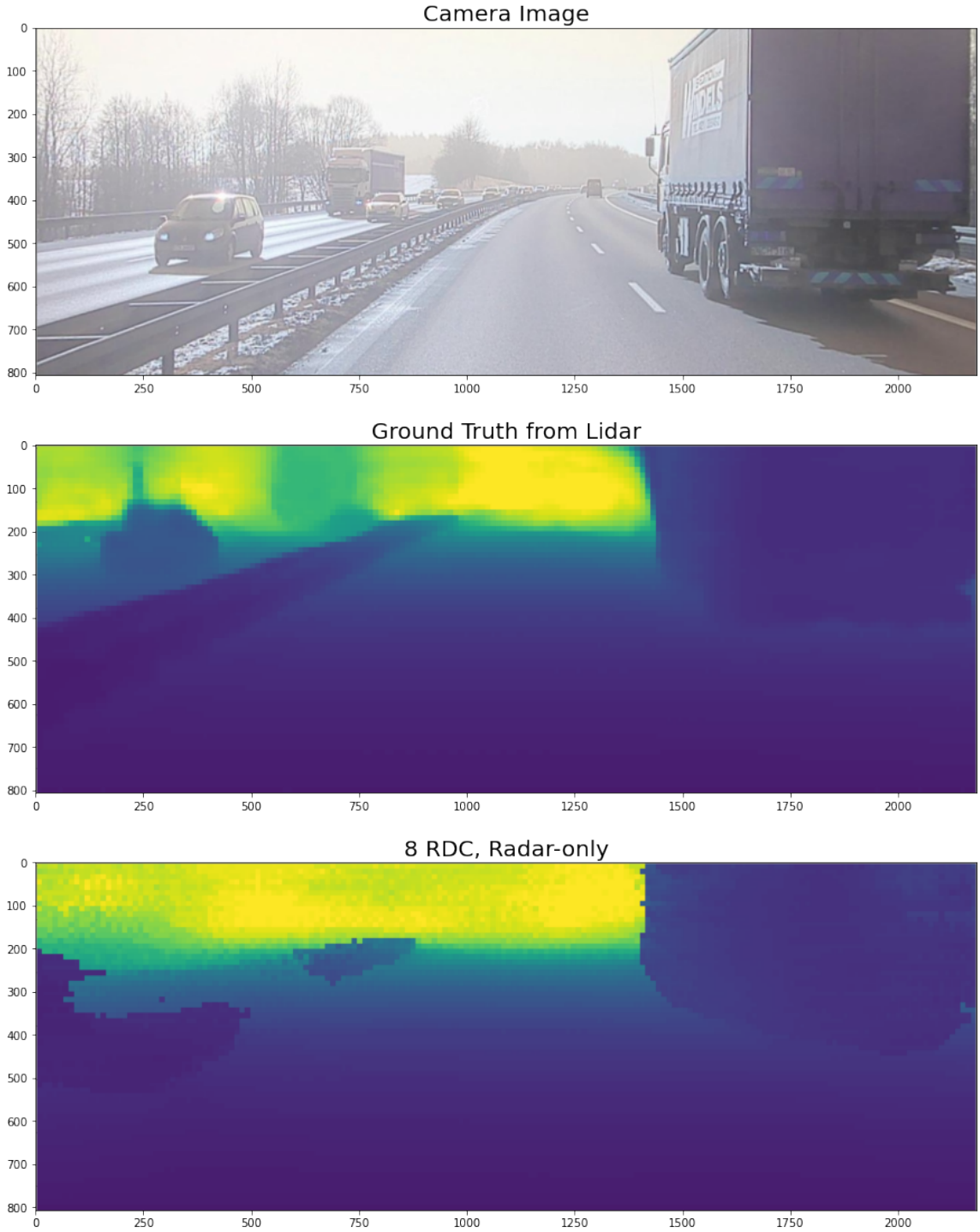


Figure 7.9. Comparison of the output of different versions of the network, frame from Highway Drive 2 (part 1). Depth is color coded, numbers on the axes denote pixel coordinates.

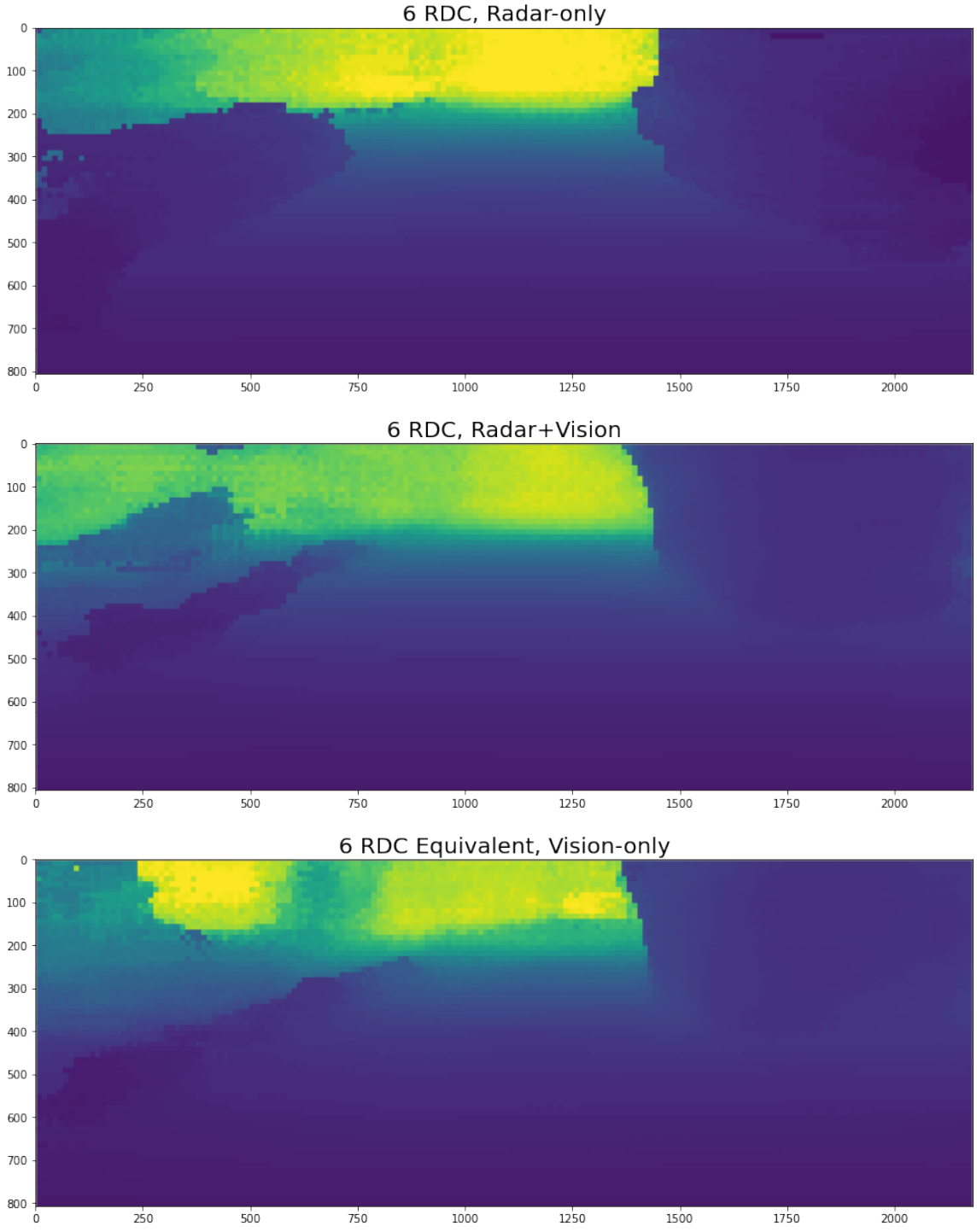


Figure 7.10. Comparison of the output of different versions of the network, frame from Highway Drive 2 (part 2). Depth is color coded, numbers on the axes denote pixel coordinates.

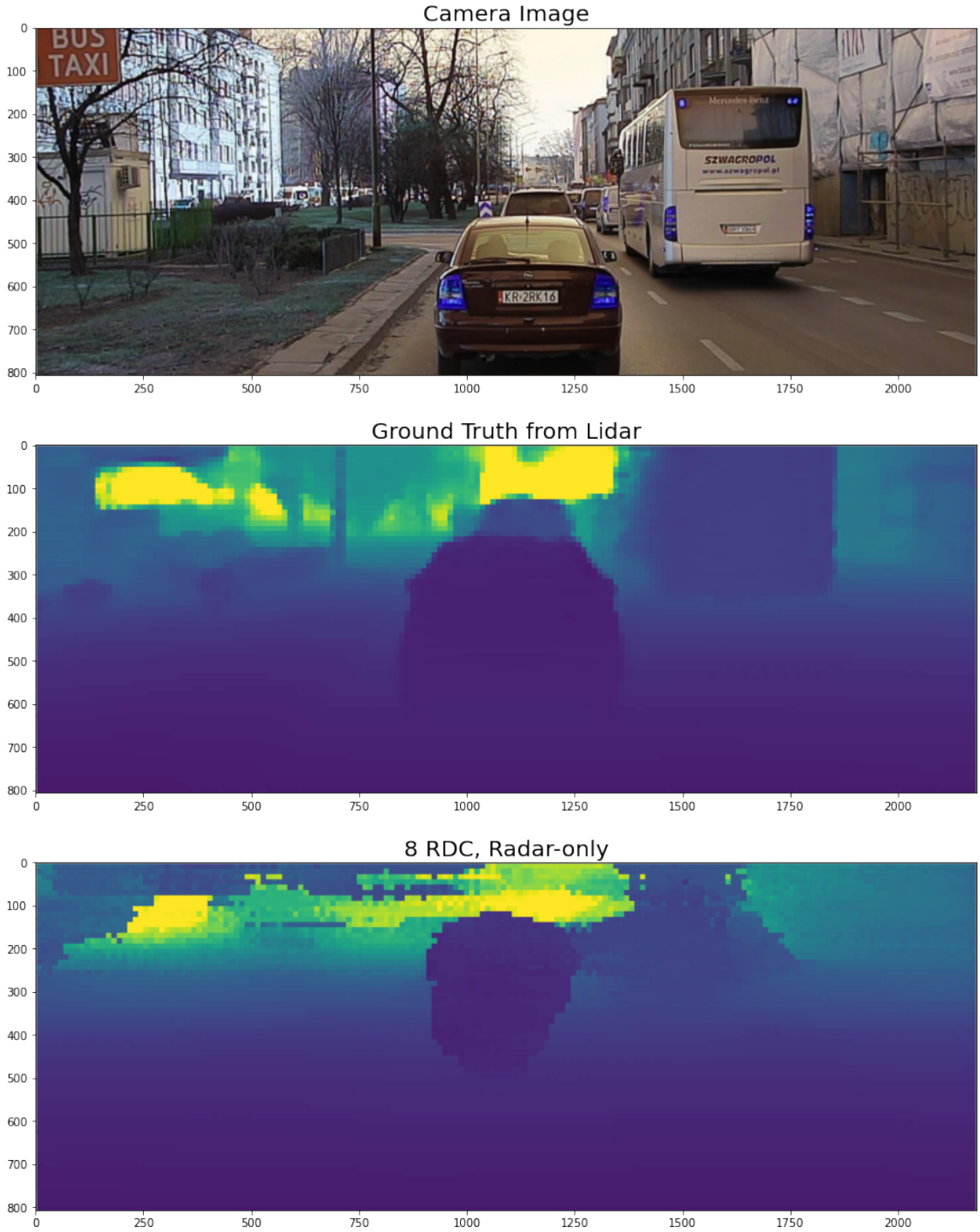


Figure 7.11. Comparison of the output of different versions of the network, frame from Urban Drive 1 (part 1). Depth is color coded, numbers on the axes denote pixel coordinates.

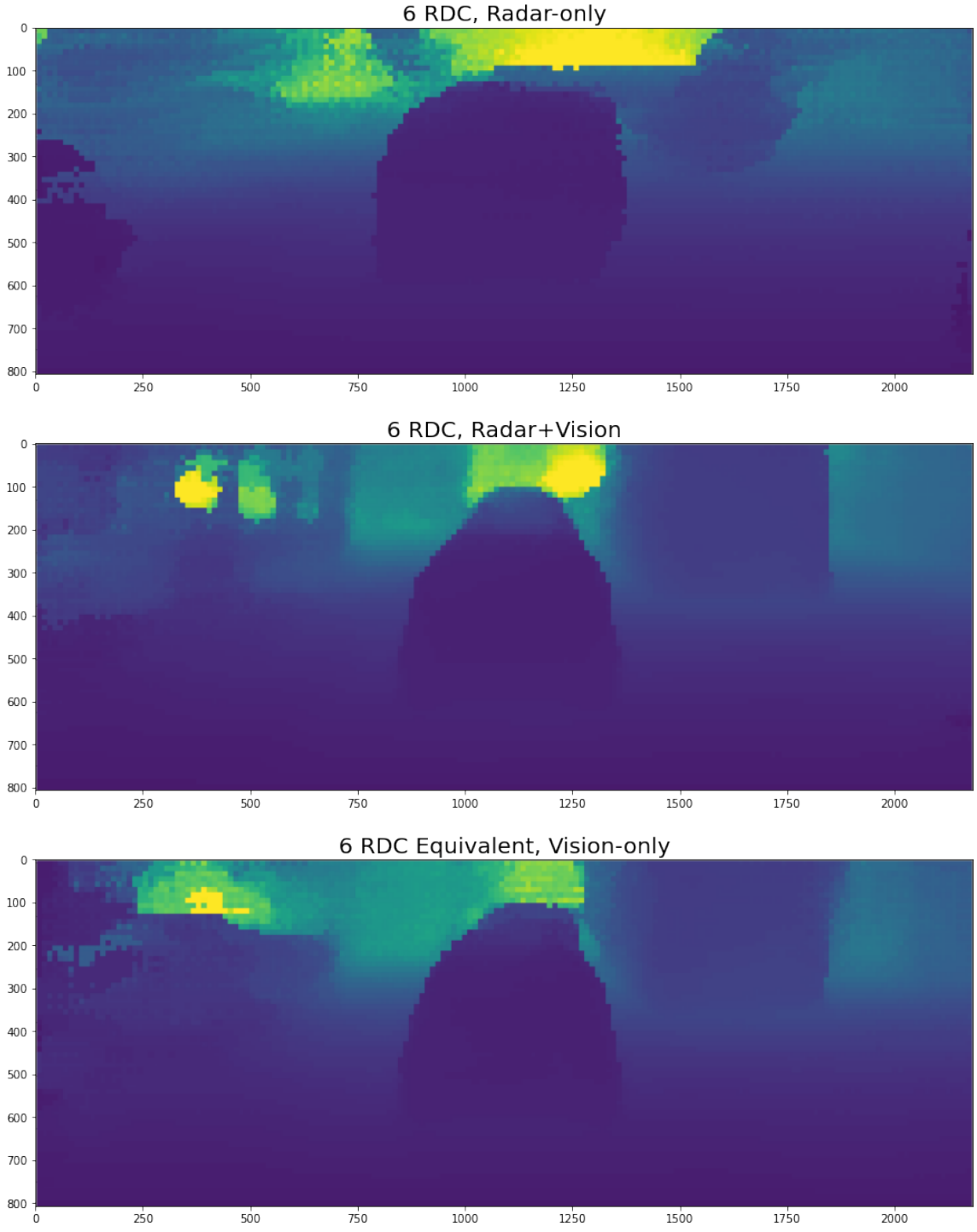


Figure 7.12. Comparison of the output of different versions of the network, frame from Urban Drive 1 (part 2). Depth is color coded, numbers on the axes denote pixel coordinates.

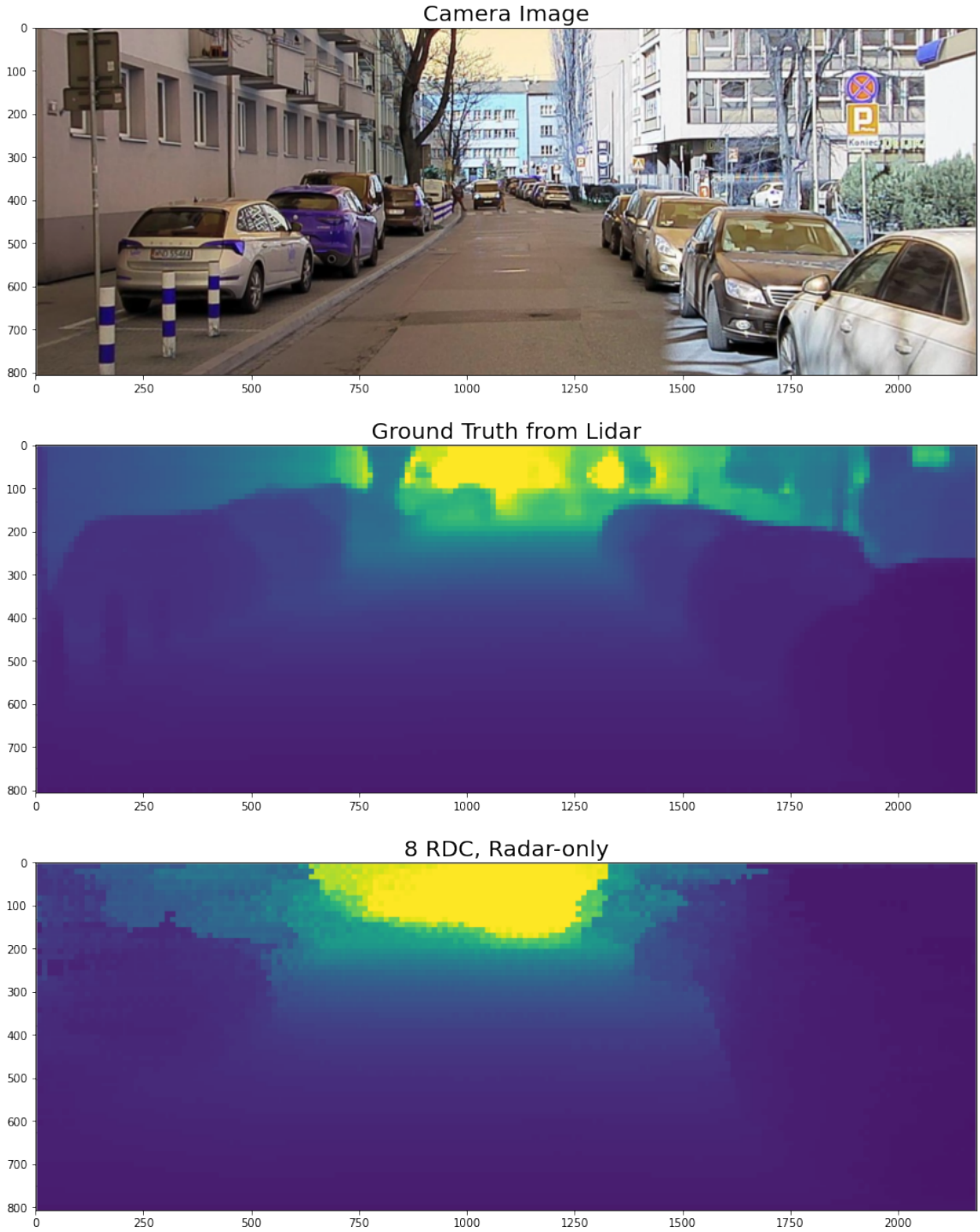


Figure 7.13. Comparison of the output of different versions of the network, frame from Urban Drive 2 (part 1). Depth is color coded, numbers on the axes denote pixel coordinates.

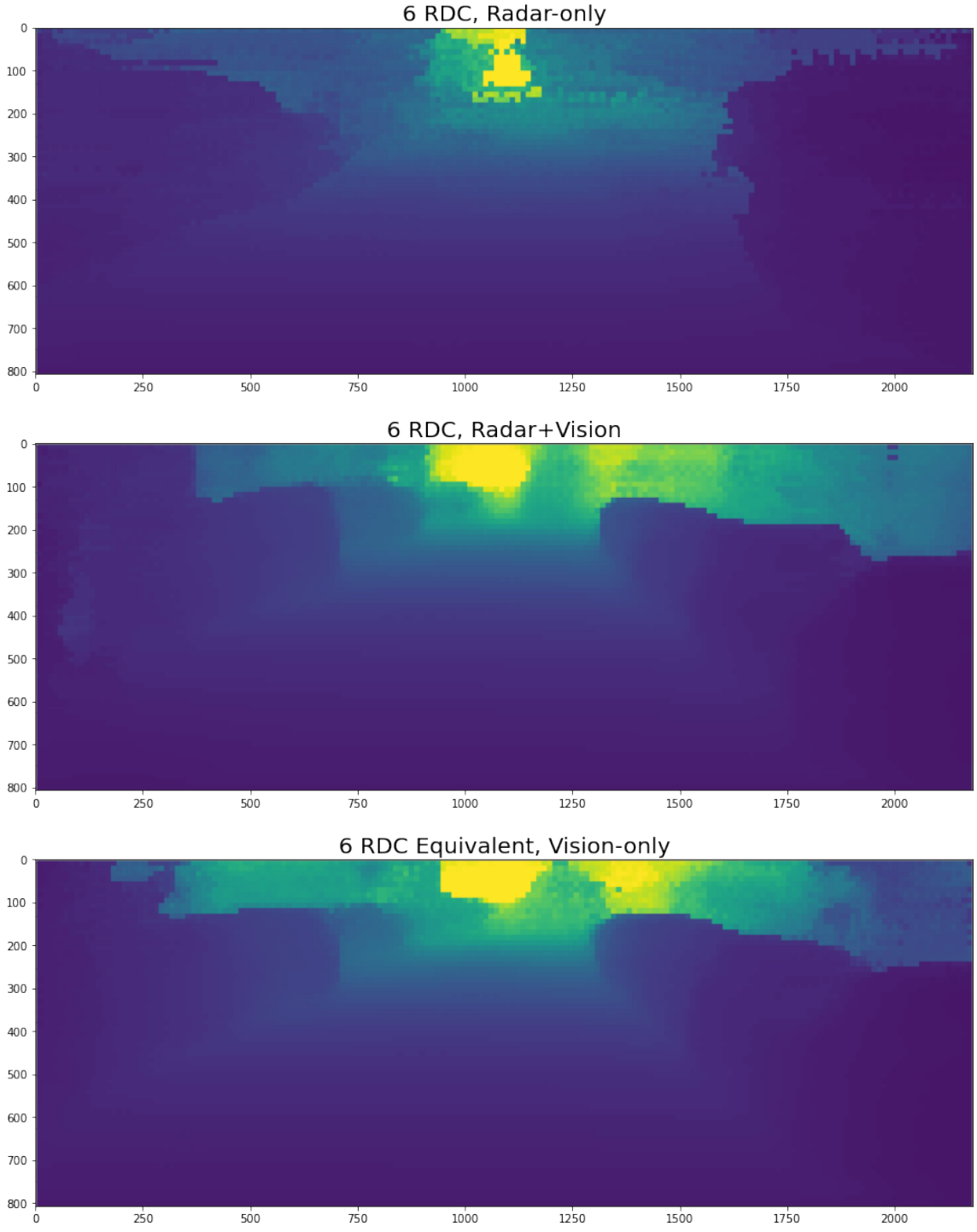


Figure 7.14. Comparison of the output of different versions of the network, frame from Urban Drive 2 (part 2). Depth is color coded, numbers on the axes denote pixel coordinates.

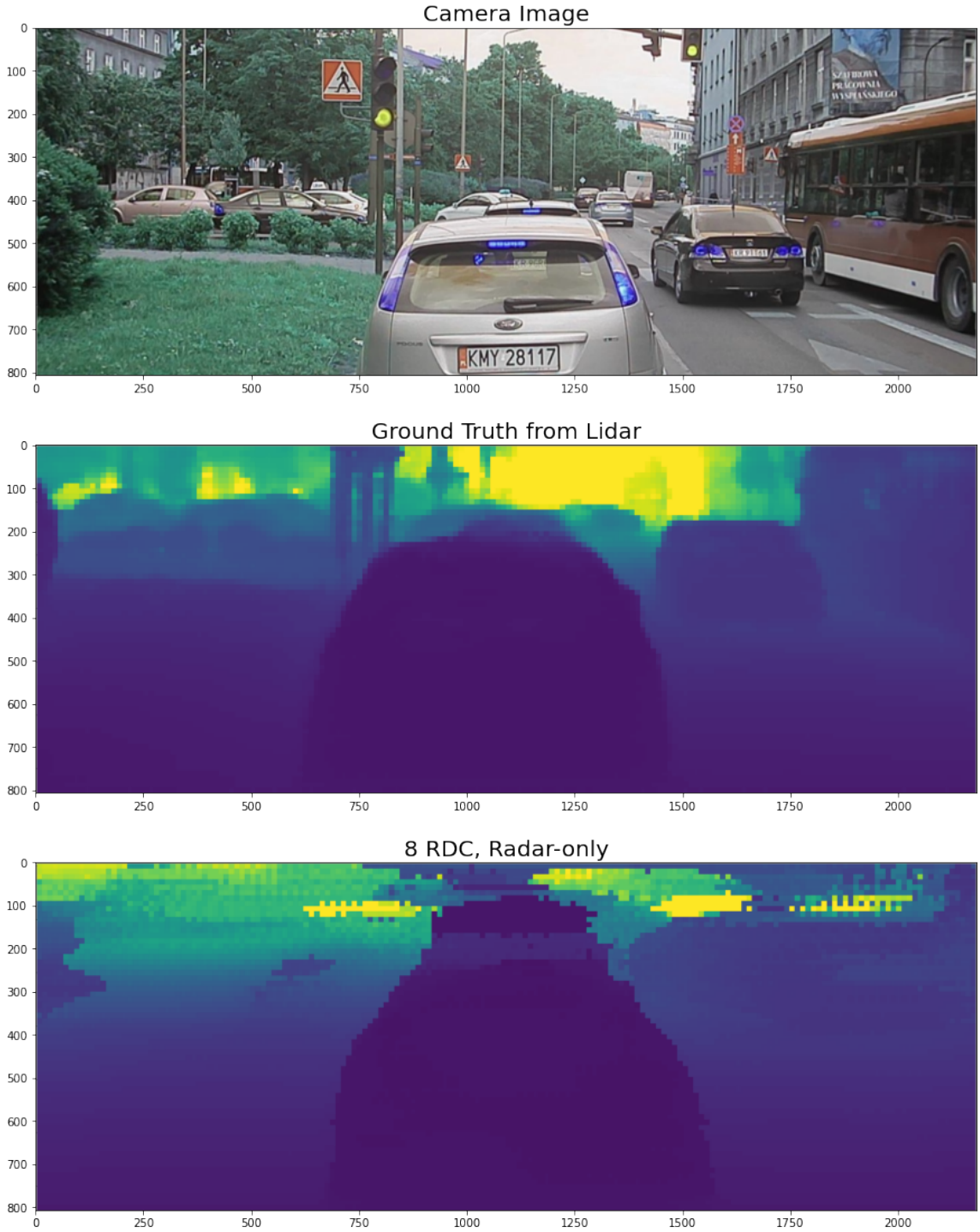


Figure 7.15. Comparison of the output of different versions of the network, frame from Urban Drive 3 (part 1). Depth is color coded, numbers on the axes denote pixel coordinates.

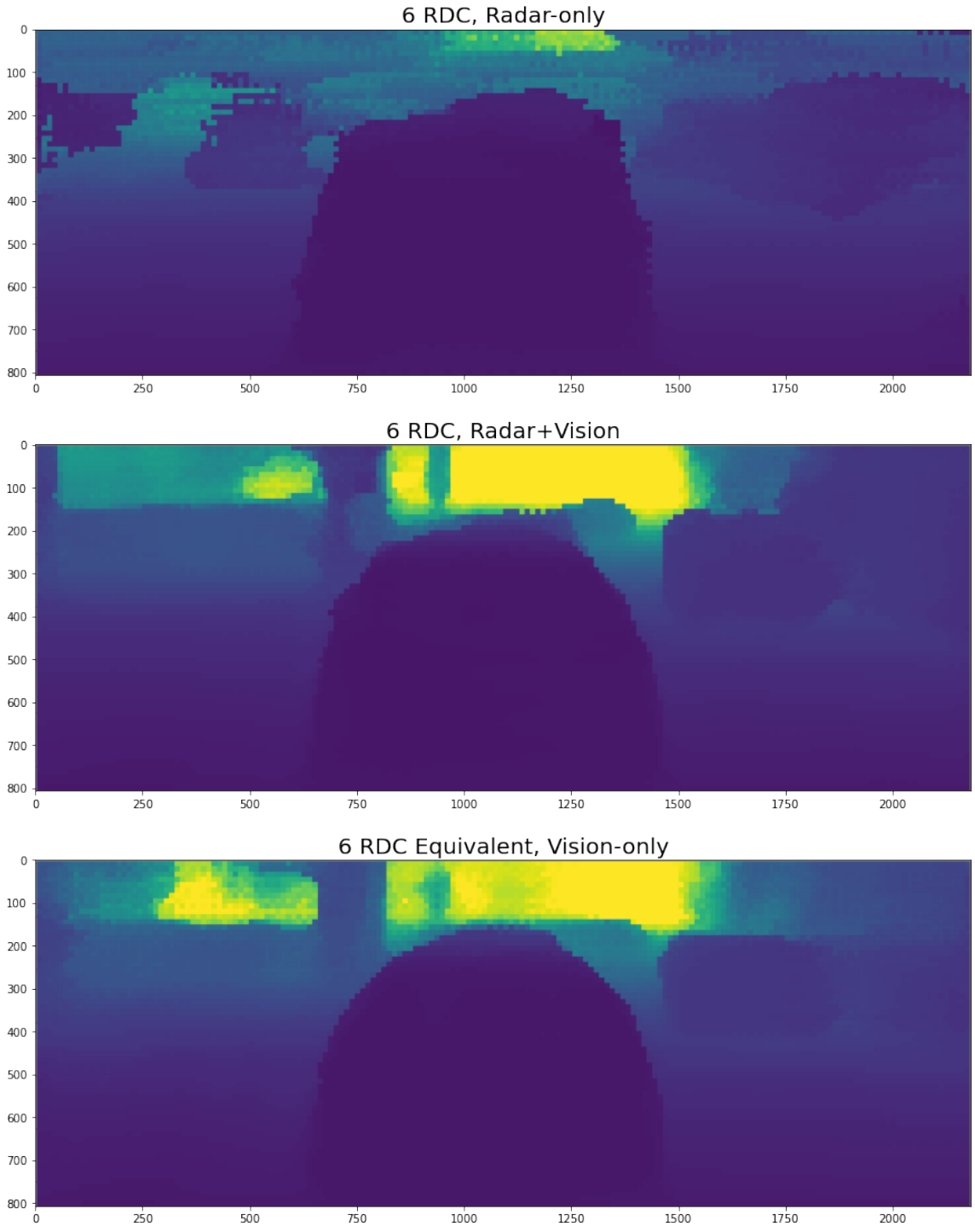


Figure 7.16. Comparison of the output of different versions of the network, frame from Urban Drive 3 (part 2). Depth is color coded, numbers on the axes denote pixel coordinates.

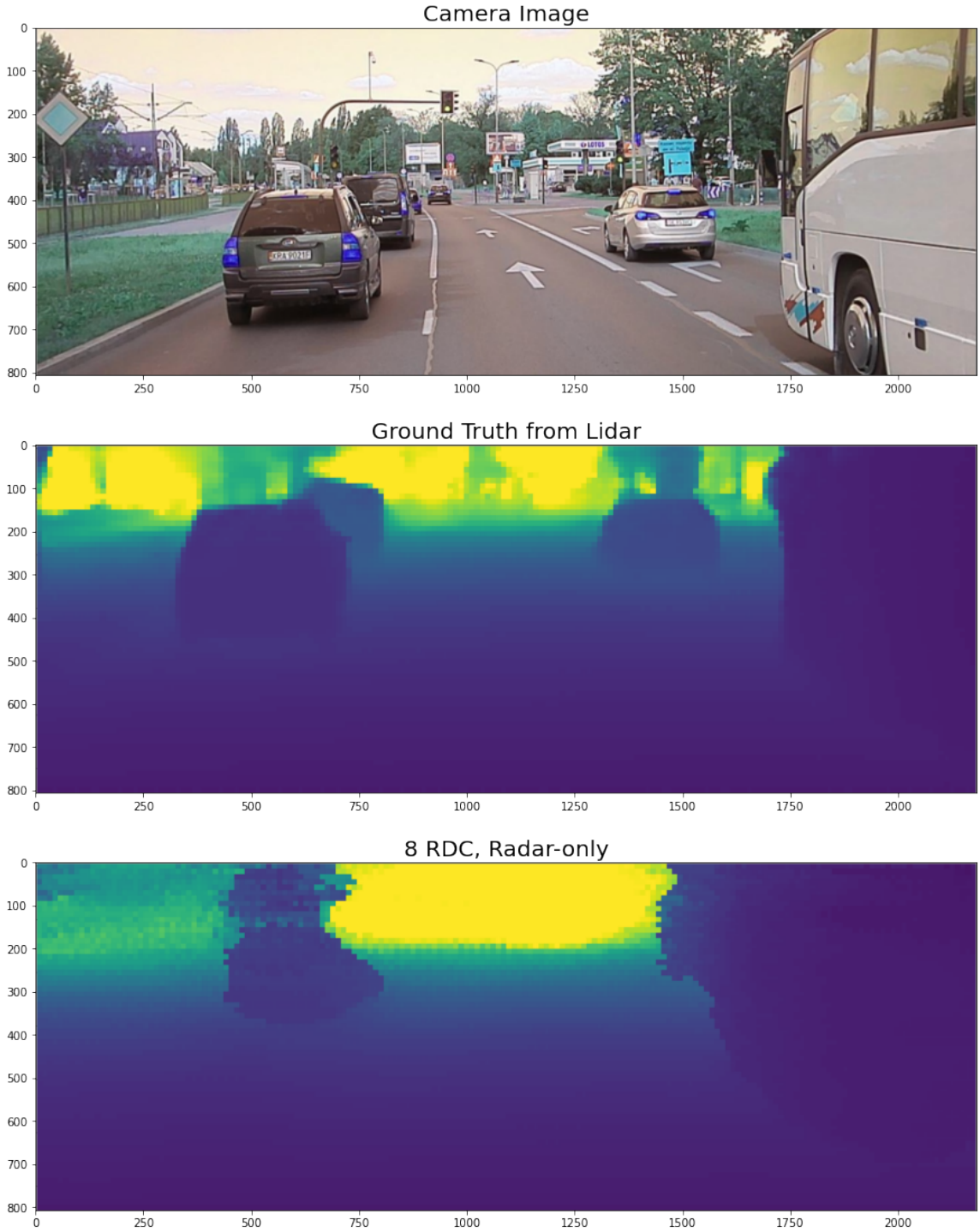


Figure 7.17. Comparison of the output of different versions of the network, frame from Urban Drive 4 (part 1). Depth is color coded, numbers on the axes denote pixel coordinates.

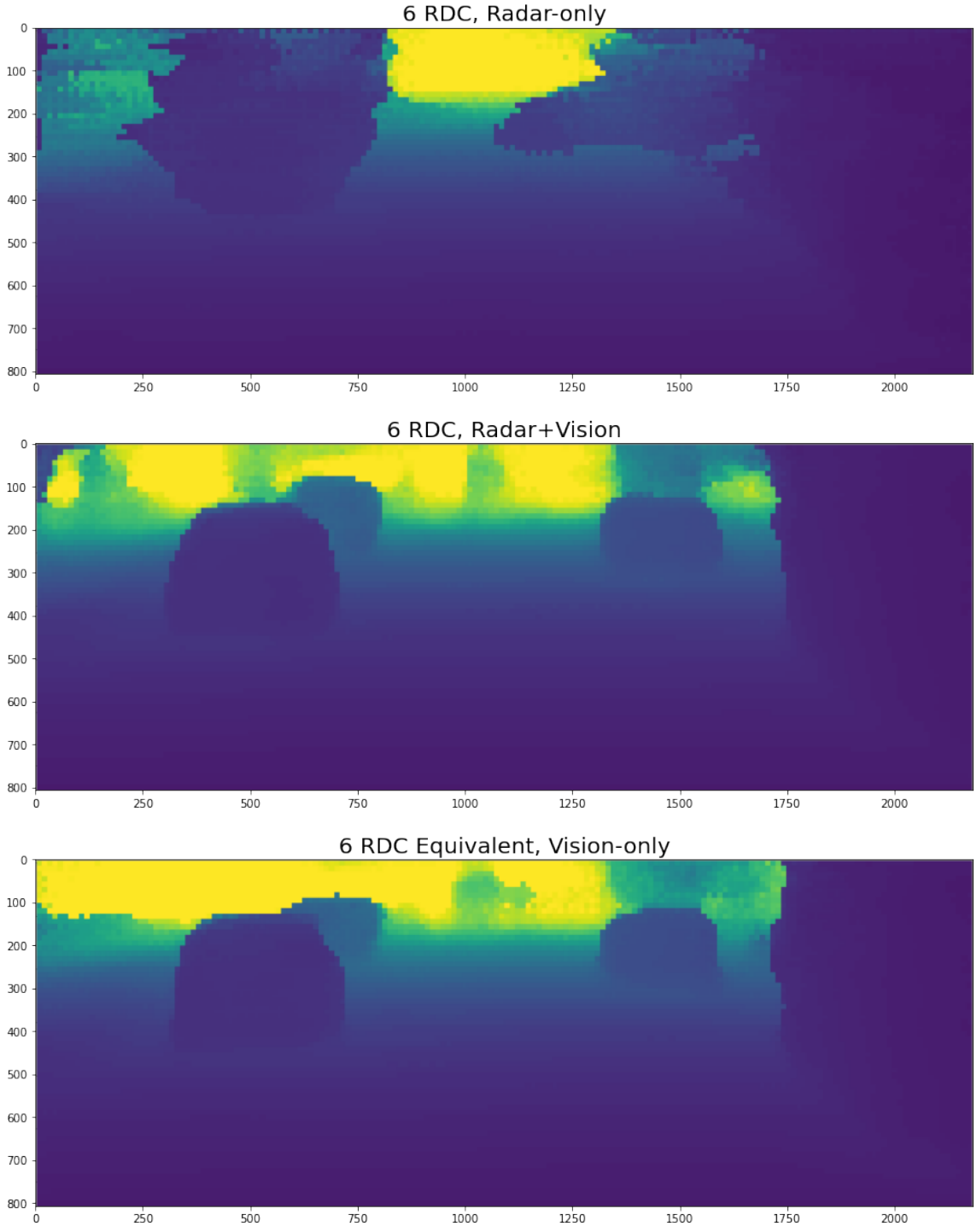


Figure 7.18. Comparison of the output of different versions of the network, frame from Urban Drive 4 (part 2). Depth is color coded, numbers on the axes denote pixel coordinates.

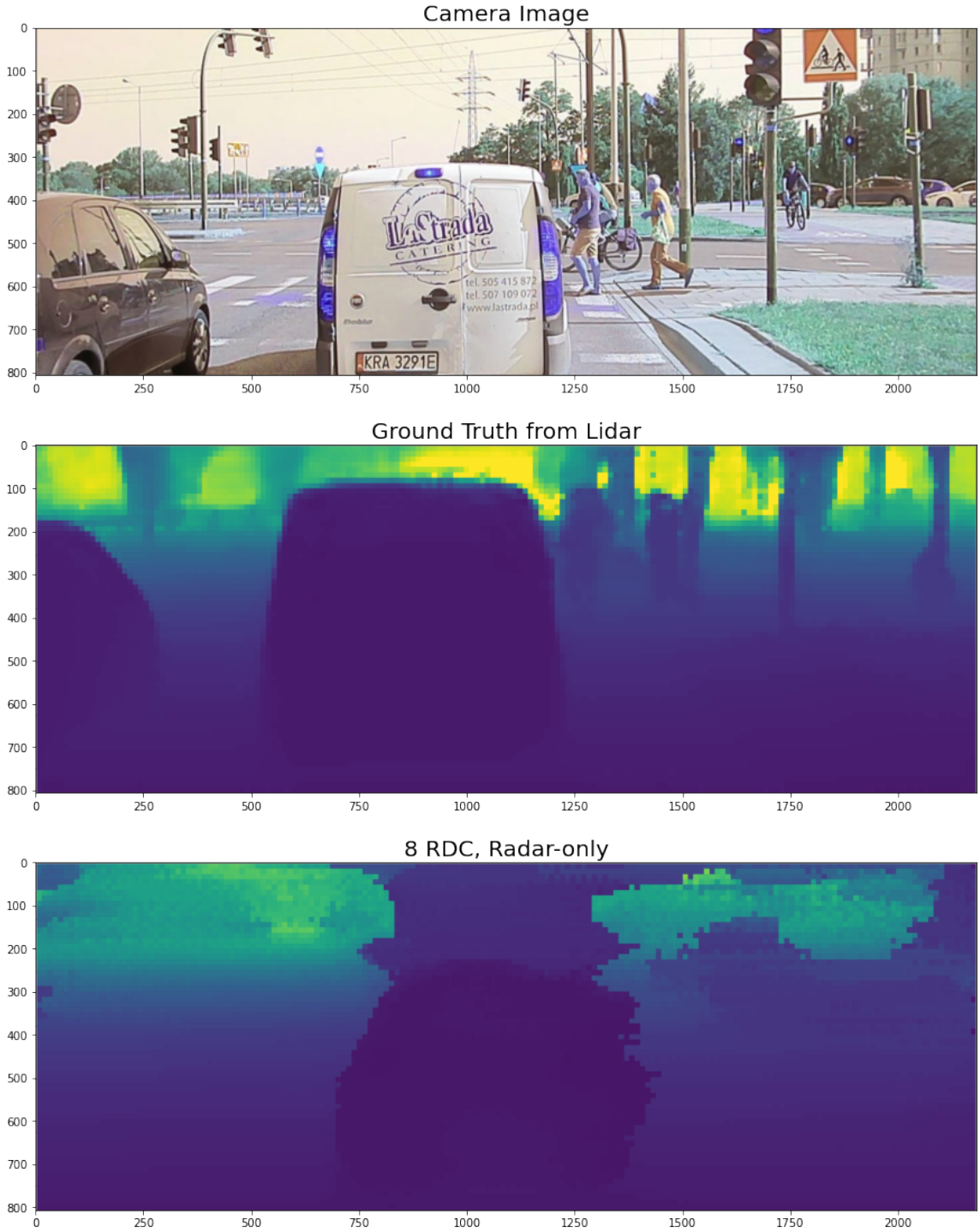


Figure 7.19. Comparison of the output of different versions of the network, frame from Urban Drive 5 (part 1). Depth is color coded, numbers on the axes denote pixel coordinates.

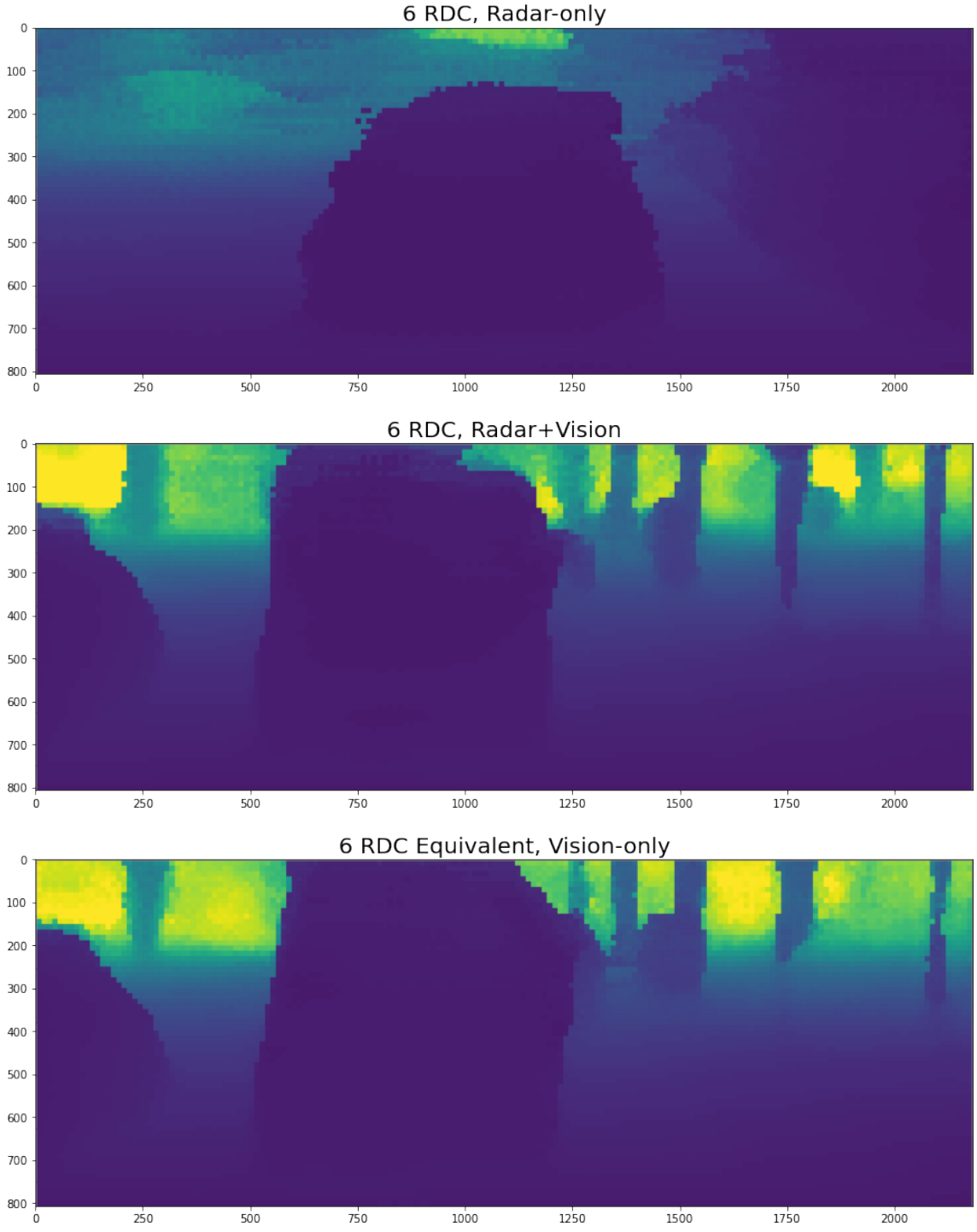


Figure 7.20. Comparison of the output of different versions of the network, frame from Urban Drive 5 (part 2). Depth is color coded, numbers on the axes denote pixel coordinates.

7.4.1 Discussion of the Radar-only Results

The performance of the radar-only versions of the network in highway settings is significantly better than in urban settings. I believe, that it is caused by the fact that there are significantly more objects from which radiowaves might reflect in an urban setting. When paired with the poor elevation accuracy of the radar and CFAR thresholding, multitude of the radar reflections in an urban setting causes the final RDC to contain fewer non-empty cells, which in turn causes network performance to drop. It should also be noted that the performance of the radar-only network is visibly improved when 8 RDCs are used instead of 6 RDCs. The author is particularly satisfied that the radar-only solution is capable of producing reasonably looking depth maps using a sensor normally not tasked with producing image-style output (please compare Figures 7.7-7.20 with Figure 1.4).

7.4.2 Discussion of the Radar + Vision Results

Fusion of radar and camera data leads to measurable improvement over vision-only solution for all the drives in our dataset (improvements between 3% and 25%). It shows that the presented network structure and training method are capable of extracting useful information from the RDC. Unlike the radar-only solution, the network using both radar and vision produces stable results in both highway and urban settings. It is difficult to compare the results between different datasets, but I believe the results of my solution are comparable with the state-of-the-art vision-only solutions for depth prediction participating in the KITTI Depth Prediction Challenge (Uhrig et al. [2017]).

7.4.3 Influence of the Number of Reflections on the Output Quality

There are large differences in the KPIs between test drives, especially for radar-only models. One possible source of the differences may be the different number of non-empty RDC cells in different environments. I decided to test that hypothesis. In Figures 7.21 and 7.22 (for radar-only and radar+vision networks respectively) I present the mean Relative L1 error calculated only for frames with at least some fraction of non-empty RDC cells. The improvement of KPIs with the number of non-empty RDC cells is very pronounced for the radar-only networks, and much less visible for radar+vision networks. This is probably caused by the fact, that some baseline depth prediction can be made using only visual data in case of scarcity of the radar measurements. In Figures 7.23-7.26 I present scatter plots, where Relative L1 error is shown on the y-axis and the fraction of the non-empty RDC bins for a particular frame is presented on the x-axis. Fitted lines are presented in the same plots. In Table 7.3 I present what fraction of KPI variance can be explained by a linear model using a fraction of the non-empty RDC cells as the

independent variable. Let $\mathrm{f}_{\mathrm{linear}}(frac_{nonempty})$ be the best-fit linear function predicting the true Relative L1 error for a particular frame, using the number of nonempty RDC cells as input. Let Relative_L1$_k$ be the Relative L1 error for the k^{th} frame, moreover, let there be n frames in total. Then, the fraction of Relative L1 variance that can be explained by the fraction of nonempty RDC cells can be calculated using the following Equation (7.4.3):

$$\mathrm{R}^2 = 1 - \frac{\sum_{k=1}^{n}(\text{Relative\_L1}_k - \mathrm{f}_{\mathrm{linear}}(frac_{nonempty}(k))^2}{\sum_{k=1}^{n}(\text{Relative\_L1}_k - \frac{\sum_{k=1}^{n}\text{Relative\_L1}_k)}{n})^2}, \tag{7.4.3}$$

where $frac_{nonempty}(k))$ is the fraction of nonempty cells at k^{th} frame.

As can be seen in the plots and the table, a linear model using the fraction of the non-empty RDC cells fails to adequately explain the KPI differences between frames. But it can be also noted, that 'catastrophic failures' (i.e. very high Relative L1 error) of the radar-only models are much more frequent for frames with a lower fraction of non-empty RDC cells. It suggests, that in case of a more lenient CFAR thresholding (resulting in a larger number of non-empty RDC cells), my models could perform much better.

Table 7.3. Fraction of Relative L1 variance explained by the number of non-empty RDC cells for different model versions

Model version	Fraction of Relative L1 variance explained by the number of non-empty RDC cells
Radar-only, 3RDC	0.085
Radar-only, 4RDC	0.245
Radar+Vision, 3RDC	0.086
Radar+Vision, 4RDC	0.019

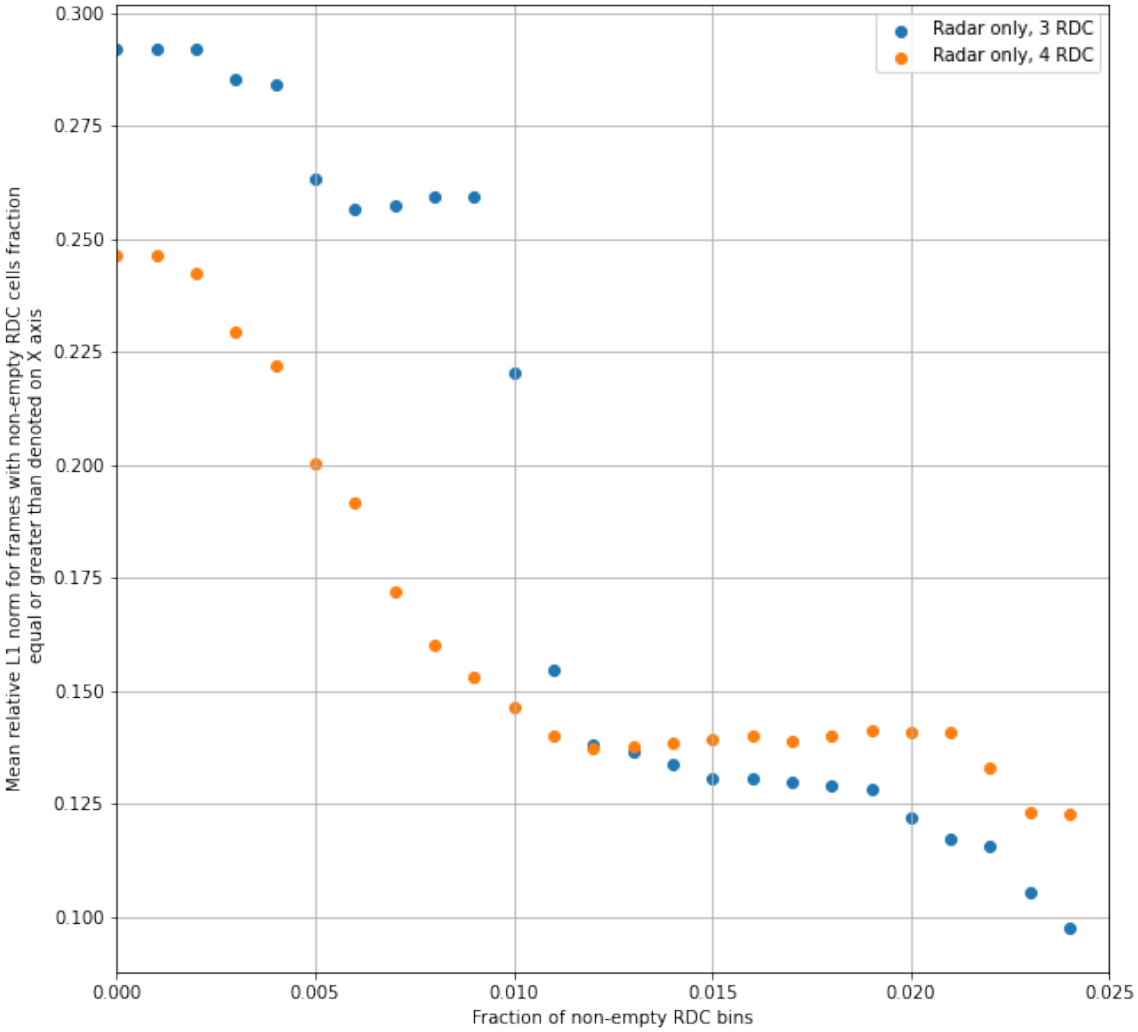


Figure 7.21. Comparison of the KPIs of different versions of the network as a function of the fraction of the non-empty RDC cells in a particular frame (for radar-only networks)

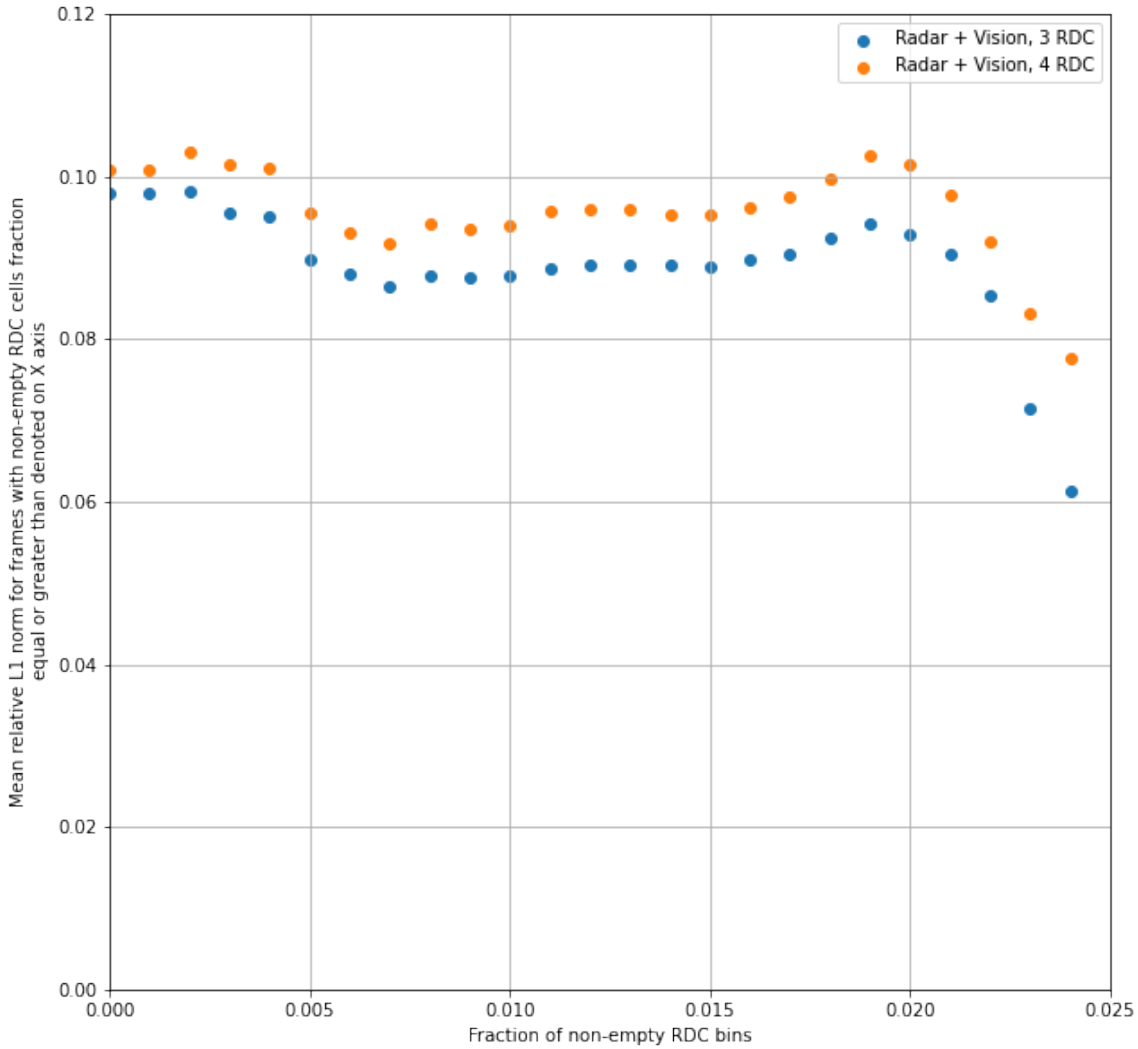


Figure 7.22. Comparison of the KPIs of different versions of the network as a function of the fraction of the non-empty RDC cells in a particular frame (for radar+vision networks)

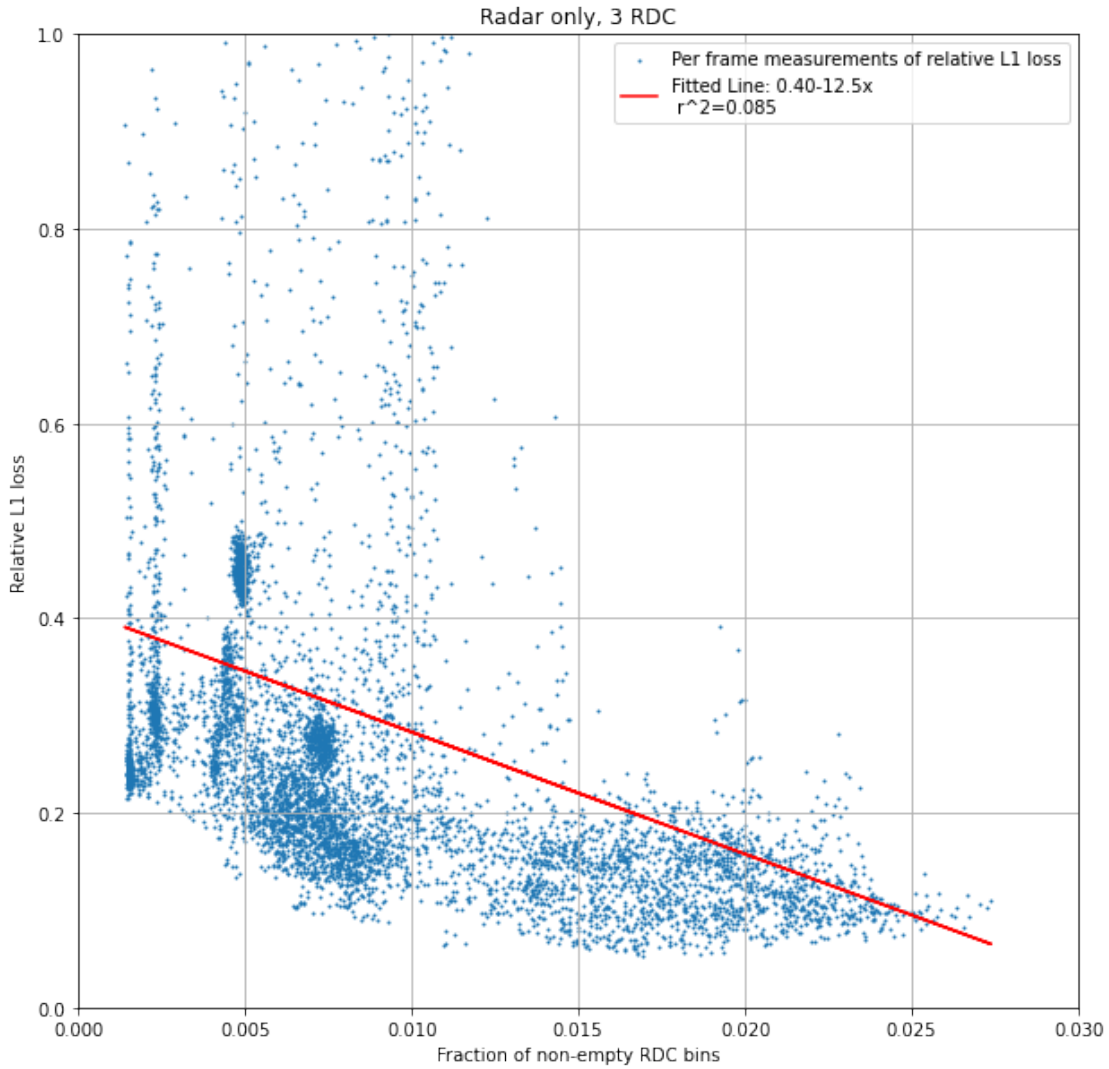


Figure 7.23. Relative L1 loss as a function of the number of non-empty RDC cells (one data point per frame), together with a fitted line, for a 3 RDC, radar-only network

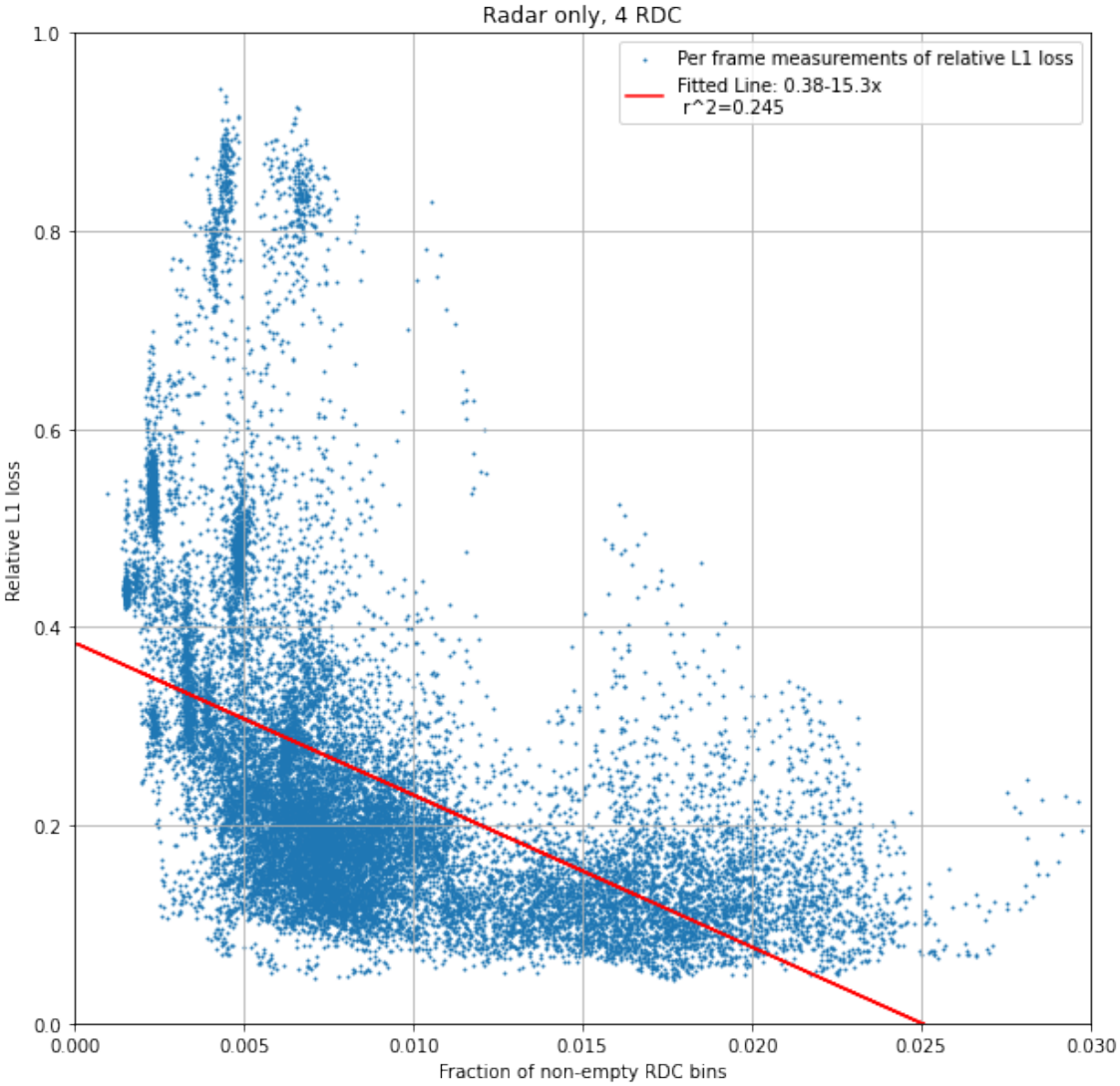


Figure 7.24. Relative L1 loss as a function of the number of non-empty RDC cells (one data point per frame), together with a fitted line, for a 4 RDC, radar-only network

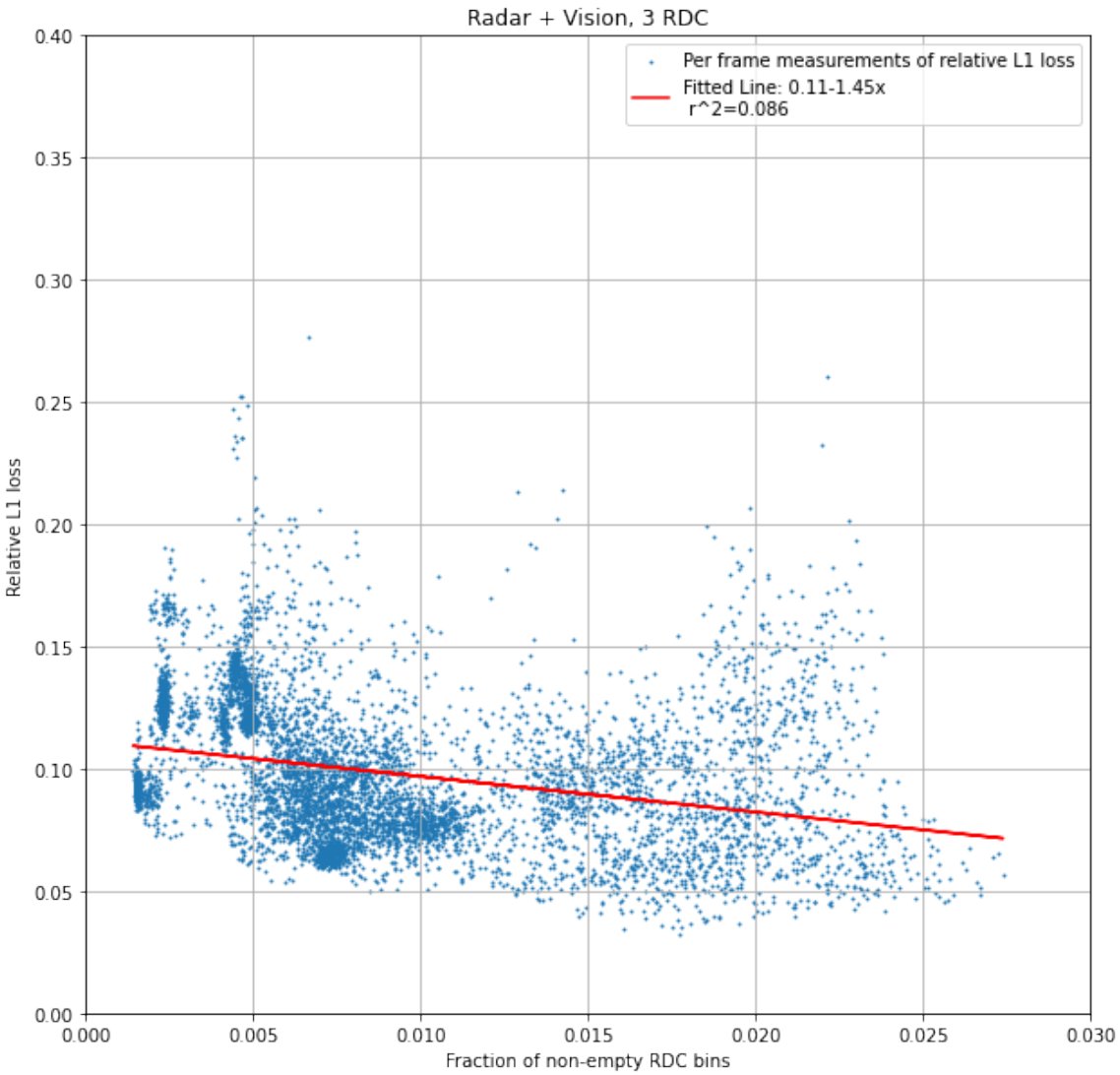


Figure 7.25. Relative L1 loss as a function of the number of non-empty RDC cells (one data point per frame), together with a fitted line, for a 3 RDC, radar+vision network

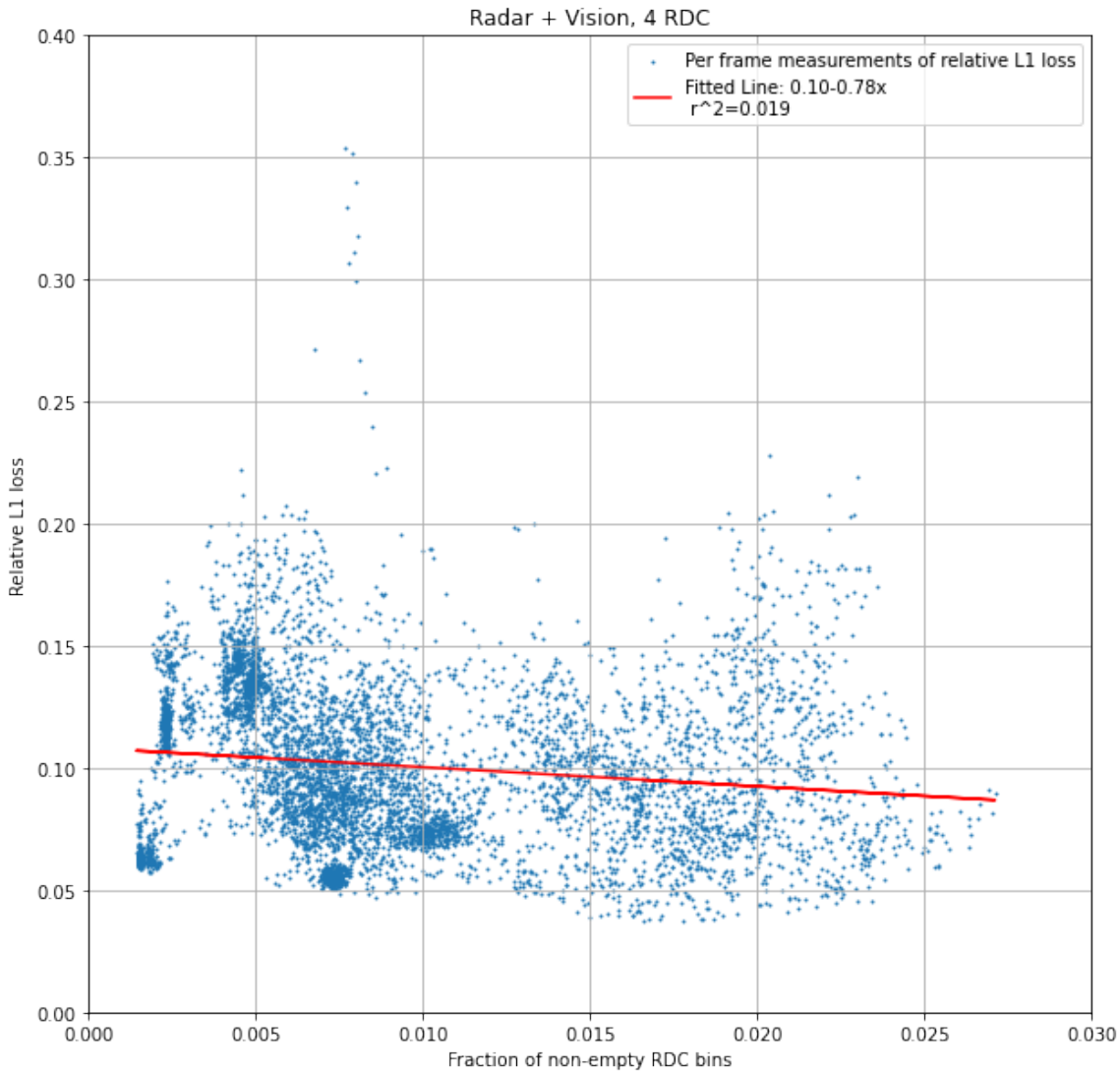


Figure 7.26. Relative L1 loss as a function of the number of non-empty RDC cells (one data point per frame), together with a fitted line, for a 4 RDC, radar+vision network

7.5 Conclusions

In this chapter, I presented a novel neural network architecture capable of processing low-level radar data from an FMCW radar commonly used in automotive. The proposed network architecture is capable of inferring significantly more information than standard algorithms used in automotive radars. In particular, I would like to note that my radar-only networks are able to produce visually understandable scene images using a sensor that was not designed to produce such output. To the best of my knowledge, this is the first solution producing a dense depth map on the basis of automotive FMCW radar RDC.

My solution transforms the RDC data into the azimuth-elevation plane, making fusion with visual data relatively easy. My radar+vision models perform measurably better than vision-only controls, proving that my model architecture is capable of extracting useful information from RDC.

I performed an analysis of the influence of the number of non-empty RDC cells on my model performance. The results of the analysis point out that the number of 'catastrophic failures' (i.e. very high Relative L1 error) can be reduced by providing the networks with more radar measurements. In practice, it can be done by using more lenient CFAR thresholding, resulting in more non-empty RDC cells.

The labels for my depth prediction network were created without the need for manual labeling, which makes the creation of large-scale datasets very affordable. I believe that pretraining on a large-scale depth prediction dataset may be beneficial for training radar-processing neural networks for different tasks (e.g. occupancy grid estimation), thereby reducing the need for expensive manual labeling.

8 Contributions

This work has succesfully proven the truthfulness of the theses formulated in Section 1.3. To end the dissertation, I would like to recapitulate the statement of contributions from the abstract.

The first contribution (described in Chapter 3) is an algorithm utilizing zero-centered, additive weight perturbations during neural network training. I show why it helps to increase the amount of information the neural network can learn for a given size and demonstrate its usefulness for some commonly used neural network architectures. Weight perturbations were utilized in the WeaveNet training and in the training of the radar angle finding model on simulated data.

The second contribution (described in Chapter 4) is the definition of WeaveNet - a neural network whose architecture is designed to perform well in the task of lidar depth completion on variable input sparsity depth measurements (name inspired by the convolutional kernels pattern, which looks like a woven fabric). It was trained and tested utilizing the data from the KITTI Depth Completion challenge. WeaveNet was instrumental in creating a dense depth dataset, that was later used in my radar depth completion solution.

The third contribution (described in Chapter 5) consists of a set of idealized simulations, showing that neural networks are capable of finding angles to two targets when radar antennae receive a superposition of reflected waves (a necessary step for a radar depth completion network).

The fourth contribution (described in Chapter 6) is the creation of a dataset used to train and validate algorithms for radar depth completion. It consists of over 1,000,000 Radar Data Cubes (RDCs) from a forward-facing radar, together with corresponding camera images and dense depth maps (produced using WeaveNet).

The fifth contribution (described in Chapter 7) is the design of a neural network architecture capable of transforming the low-level RDC input into abstract channels in the azimuth-elevation plane. Consequently, it was possible to train this neural network to predict dense depth maps using RDC as input. They were trained and tested on the dataset that was created for the purpose of the work on this PhD dissertation. The output of the radar-only depth completion networks is visually reasonable and much better than

the linear interpolation of the point cloud obtained using standard radar processing algorithms. To the best of my knowledge, this is the first solution producing a dense depth map on the basis of automotive FMCW radar RDC.

The final contribution (also described in Chapter 7) is the design of the neural network architecture capable of fusing the RDC-derived data from the radar and the data derived from RGB camera images for the purpose of radar depth completion (also trained and tested on the same dataset). I have shown that the networks utilizing RDC in addition to visual data obtain results between 3% and 21.5% better than analogous networks trained to use visual data only (during different data collection drives).

Bibliography

[1] A. Agarwal and C. Arora. Attention attention everywhere: Monocular depth prediction with skip attention. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 5861–5870, 2023.

[2] O. Bialer, A. Jonas, and T. Tirer. Super resolution wide aperture automotive radar. *IEEE Sensors Journal*, 21(16):17846–17858, 2021. doi: 10.1109/JSEN.2021.3085677.

[3] S. Biderman, H. Schoelkopf, Q. G. Anthony, H. Bradley, K. O'Brien, E. Hallahan, M. A. Khan, S. Purohit, U. S. Prashanth, E. Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR, 2023.

[4] S. Blake. Os-cfar theory for multiple targets and nonuniform clutter. *IEEE Transactions on Aerospace and Electronic Systems*, 24(6):785–790, 1988. doi: 10.1109/7.18645.

[5] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020.

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[7] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuscenes: A multimodal dataset for autonomous driving. *CoRR*, abs/1903.11027, 2019. URL *http://arxiv.org/abs/1903.11027*.

[8] K. Chellapilla, S. Puri, and P. Simard. High performance convolutional neural networks for document processing. In *Tenth international workshop on frontiers in handwriting recognition*. Suvisoft, 2006.

[9] Q. Chen, B. Ge, and J. Quan. Unambiguous pyramid cost volumes fusion for stereo matching. *IEEE Transactions on Circuits and Systems for Video Technology*, 2023.

[10] Y. Chen, B. Yang, M. Liang, and R. Urtasun. Learning joint 2d-3d representations for depth completion. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 10023–10032, 2019.

[11] X. Cheng, P. Wang, C. Guan, and R. Yang. Cspn++: Learning context and resource aware convolutional spatial propagation networks for depth completion. *arXiv preprint arXiv:1911.05377*, 2019a.

[12] X. Cheng, P. Wang, and R. Yang. Learning depth with convolutional spatial propagation network. *arXiv preprint arXiv:1810.02695*, 2019b.

[13] X. Cheng, Y. Zhong, M. Harandi, Y. Dai, X. Chang, H. Li, T. Drummond, and Z. Ge. Hierarchical neural architecture search for deep stereo matching. *Advances in Neural Information Processing Systems*, 33:22158–22169, 2020.

[14] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[15] F. Chollet et al. Keras. *https://keras.io*, 2015.

[16] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[17] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2016.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[20] A. Diaz. Regularization method: Noise for improving deep learning models, 2020. URL *https://towardsdatascience.com/noise-its-not-always-annoying-1bd5f0f240f*.

[21] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[22] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. *CoRR*, abs/1406.2283, 2014. URL *http://arxiv.org/abs/1406.2283*.

[23] A. Eldesokey, M. Felsberg, and F. S. Khan. Propagating confidences through cnns for sparse data regression. *arXiv preprint arXiv:1805.11913*, 2018.

[24] H. M. Finn. A cfar design for a window spanning two clutter fields. *IEEE Transactions on Aerospace and Electronic Systems*, AES-22(2):155–169, 1986. doi: 10.1109/TAES.1986.310750.

[25] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[26] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[27] X. Geng and H. Liu. Openllama: An open reproduction of llama, May 2023. URL *https://github.com/openlm-research/open_llama*.

[28] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. *http://www.deeplearningbook.org*.

[29] S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, 2015.

[30] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[31] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[32] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam. Searching for mobilenetv3. 2019. doi: 10.48550/ARXIV.1905.02244. URL *https://arxiv.org/abs/1905.02244*.

[33] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[34] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[35] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *arXiv preprint arXiv:1811.06965*, 2019a.

[36] Z. Huang, J. Fan, S. Cheng, S. Yi, X. Wang, and H. Li. Hms-net: Hierarchical multi-scale sparsity-invariant network for sparse depth completion. *IEEE Transactions on Image Processing*, 29:3429–3441, 2019b.

[37] N. Kanopoulos, N. Vasanthavada, and R. L. Baker. Design of an image edge detection filter using the sobel operator. *IEEE Journal of solid-state circuits*, 23(2):358–367, 1988.

[38] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[39] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[40] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[41] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989a. doi: 10.1162/neco.1989.1.4.541.

[42] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel. Optimal brain damage. In *NIPs*, volume 2, pages 598–605. Citeseer, 1989b.

[43] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[44] N. Lee, T. Ajanthan, and P. Torr. SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY. In *International Conference on Learning Representations*, 2019. URL *https://openreview.net/forum?id=B1VZqjAcYX*.

[45] K. Lelowicz. Camera model for lens with strong distortion in automotive application. In *2019 24th International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 314–319, 2019. doi: 10.1109/MMAR.2019.8864659.

[46] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[47] F.-F. Li. Stereo vision, 2014a. URL *http://vision.stanford.edu/teaching/cs131_fall1415/lectures/lecture9_10_stereo_cs131.pdf*.

[48] H. Li and Y. Li. Low-sidelobe antenna array based on evanescent mode of cutoff waveguide. *IEEE Transactions on Antennas and Propagation*, 70(12):11608–11616, 2022. doi: 10.1109/TAP.2022.3209274.

[49] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[50] J. Li, P. Wang, P. Xiong, T. Cai, Z. Yan, L. Yang, J. Liu, H. Fan, and S. Liu. Practical stereo matching via cascaded recurrent network with adaptive correlation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16263–16272, June 2022a.

[51] L. Li. Time-of-flight camera - an introduction, 2014b. URL *https://www.ti.com/lit/wp/sloa190b/sloa190b.pdfts=1693310204437&ref_url=https%253A%252F%252Fwww.google.com%252F#:~:text=A%203D%20tme%2Dof%2Dflight,measured%20and%20translated%20to%20distance.*

[52] Z. Li, X. Wang, X. Liu, and J. Jiang. Binsformer: Revisiting adaptive bins for monocular depth estimation. *arXiv preprint arXiv:2204.00987*, 2022b.

[53] J.-T. Lin, D. Dai, and L. Van Gool. Depth estimation from monocular images and sparse radar data. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10233–10240. IEEE, 2020.

[54] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*, 2018.

[55] B. Liu, H. Yu, and Y. Long. Local similarity pattern and cost self-reassembling for deep stereo matching networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 1647–1655, 2022.

[56] C. Liu, S. Kumar, S. Gu, R. Timofte, and L. Van Gool. Single image depth prediction made better: A multivariate gaussian take—supplementary material—.

[57] C. Liu, S. Kumar, S. Gu, R. Timofte, and L. Van Gool. Va-depthnet: A variational approach to single image depth prediction. *arXiv preprint arXiv:2302.06556*, 2023.

[58] S. Liu, S. D. Mello, J. Gu, G. Zhong, M.-H. Yang, and J. Kautz. Learning affinity via spatial propagation networks. *arXiv preprint arXiv:1710.01020*, 2017.

[59] C.-C. Lo and P. Vandewalle. Depth estimation from monocular images and sparse radar using deep ordinal regression network. In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 3343–3347. IEEE, 2021.

[60] Y. Long, D. Morris, X. Liu, M. Castro, P. Chakravarty, and P. Narayanan. Radar-camera pixel depth association for depth completion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12507–12516, 2021.

[61] F. Ma, G. V. Cavalheiro, and S. Karaman. Self-supervised sparse-to-dense: Self-supervised depth completion from lidar and monocular camera. *arXiv preprint arXiv:1807.00275*, 2018.

[62] M. Markel. *Radar for Fully Autonomous Driving*. Artech House, Norwood, MA, 2022.

[63] MATLAB. *version R2021a*. The MathWorks Inc., Natick, Massachusetts, 2021.

[64] M. Menze and A. Geiger. Object scene flow for autonomous vehicles. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3061–3070, 2015. doi: 10.1109/CVPR.2015.7298925.

[65] D. Molchanov, A. Ashukha, and D. Vetrov. Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*, 2017.

[66] A. Moreira, P. Prats-Iraola, M. Younis, G. Krieger, I. Hajnsek, and K. P. Papathanassiou. A tutorial on synthetic aperture radar. *IEEE Geoscience and Remote Sensing Magazine*, 1(1):6–43, 2013. doi: 10.1109/MGRS.2013.2248301.

[67] A. Nair. Improving deep learning model robustness by adding noise using keras, 2018. URL *https://analyticsindiamag.com/improving-deep-learning-model-robustness-by-adding-noise-using-keras/*.

[68] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

[69] J. Nelson. Why to add noise to images for machine learning, 2020. URL *https://blog.roboflow.com/why-to-add-noise-to-images-for-machine-learning*.

[70] C. Ning and H. Gan. Trap attention: Monocular depth estimation with manual traps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5033–5043, 2023.

[71] M. K. Nowak. Weavenet: Solution for variable input sparsity depth completion. *Electronics*, 11 (14), 2022. ISSN 2079-9292. doi: 10.3390/electronics11142222. URL *https://www.mdpi.com/2079-9292/11/14/2222*.

[72] M. K. Nowak and K. Lelowicz. Weight perturbation as a method for improving performance of deep neural networks. In *2021 25th International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 127–132, 2021. doi: 10.1109/MMAR49549.2021.9528460.

[73] OpenAI. Gpt-4 technical report, 2023.

[74] J. Park, K. Joo, Z. Hu, C.-K. Liu, and I. S. Kweon. Non-local spatial propagation network for depth completion. *arXiv preprint arXiv:2007.10042*, 2020.

[75] L. Piccinelli, C. Sakaridis, and F. Yu. idisc: Internal discretization for monocular depth estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21477–21487, 2023.

[76] J. Qiu, Z. Cui, Y. Zhang, X. Zhang, S. Liu, B. Zeng, and M. Pollefeys. Deeplidar: Deep surface normal guided depth prediction for outdoor scene from sparse lidar data and single color image. *arXiv preprint arXiv:1812.00488*, 2019.

[77] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[78] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[79] R. Ranftl, A. Bochkovskiy, and V. Koltun. Vision transformers for dense prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 12179–12188, October 2021.

[80] M. A. Richards. *Fundamentals of Radar Signal Processing*. McGraw-Hill, New York, 2014.

[81] R. H. Roy and T. Kailath. Esprit-estimation of signal parameters via rotational invariance techniques. *IEEE Trans. Acoust. Speech Signal Process.*, 37:984–995, 1989.

[82] E. Rusak. A surprisingly simple way to make dnns robust against many types of image corruptions, 2020. URL *https://medium.com/bethgelab/increasing-the-robustness-of-dnns-against-image-corruptions-by-playing-the-game-of-noise-4566b5c2c8d5*.

[83] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[84] R. Schmidt. Multiple emitter location and signal parameter estimation. *IEEE Transactions on Antennas and Propagation*, 34(3):276–280, 1986. doi: 10.1109/TAP.1986.1143830.

[85] S. Shalev-Shwartz, S. Shammah, and A. Shashua. On a formal model of safe and scalable self-driving cars. *arXiv preprint arXiv:1708.06374*, 2018.

[86] S. Shao, Z. Pei, W. Chen, R. Li, Z. Liu, and Z. Li. Urcdc-depth: Uncertainty rectified cross-distillation with cutflip for monocular depth estimation. *arXiv preprint arXiv:2302.08149*, 2023a.

[87] S. Shao, Z. Pei, W. Chen, X. Wu, and Z. Li. Nddepth: Normal-distance assisted monocular depth estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7931–7940, 2023b.

[88] S. Shao, Z. Pei, X. Wu, Z. Liu, W. Chen, and Z. Li. Iebins: Iterative elastic bins for monocular depth estimation. *arXiv preprint arXiv:2309.14137*, 2023c.

[89] K. Shim, J. Kim, G. Lee, and B. Shim. Depth-relative self attention for monocular depth estimation. *arXiv preprint arXiv:2304.12849*, 2023.

[90] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[91] A. D. Singh, Y. Ba, A. Sarker, H. Zhang, A. Kadambi, S. Soatto, M. Srivastava, and A. Wong. Depth estimation from camera image and mmwave radar point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9275–9285, 2023.

[92] L. Sommer, P. Schröppel, and T. Brox. Sf2se3: Clustering scene flow into se (3)-motions via proposal and selection. In *DAGM German Conference on Pattern Recognition*, pages 215–229. Springer, 2022.

[93] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[94] S. Sun, R. Liu, and S. Sun. Range-free disparity estimation with self-adaptive dual-matching. *IET Computer Vision*, 2022.

[95] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer. Efficient processing of deep neural networks: A tutorial and survey. *arXiv preprint arXiv:1703.09039*, 2017.

[96] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[97] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[98] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[99] M. Tan and Q. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[100] J. Tang, F.-P. Tian, W. Feng, J. Li, and P. Tan. Learning guided convolutional network for depth completion. *arXiv preprint arXiv:1908.01238*, 2019.

[101] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

[102] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

[103] J. Uhrig, N. Schneider, L. Schneider, U. Franke, T. Brox, and A. Geiger. Sparsity invariant cnns. In *International Conference on 3D Vision (3DV)*, 2017.

[104] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014. ISSN 2167-8359. doi: 10.7717/peerj.453. URL *https://doi.org/10.7717/peerj.453*.

[105] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[106] S. Wang, S. Suo, W.-C. Ma, A. Pokrovsky, and R. Urtasun. Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2589–2597, 2018.

[107] W. Wang, Y. Song, J. Zhang, and H. Deng. Automatic parking of vehicles: A review of literatures. *International Journal of Automotive Technology*, 15:967–978, 2014.

[108] P. Weinzaepfel, T. Lucas, V. Leroy, Y. Cabon, V. Arora, R. Brégier, G. Csurka, L. Antsfeld, B. Chidlovskii, and J. Revaud. Croco v2: Improved cross-view completion pre-training for stereo matching and optical flow. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 17969–17980, October 2023.

[109] D.-I. F. C. Wolff. Radar basics, Nov 1998. URL *https://www.radartutorial.eu/02.basics/Frequency%20Modulated%20Continuous%20Wave%20Radar.en.html*.

[110] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[111] G. Xu, J. Cheng, P. Guo, and X. Yang. Attention concatenation volume for accurate and efficient stereo matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12981–12990, 2022a.

[112] G. Xu, X. Wang, X. Ding, and X. Yang. Iterative geometry encoding volume for stereo matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21919–21928, June 2023.

[113] R. Xu, W. Dong, A. Sharma, and M. Kaess. Learned depth estimation of 3d imaging radar for indoor mapping. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 13260–13267. IEEE, 2022b.

[114] L. Yan, K. Liu, and E. Belyaev. Revisiting sparsity invariant convolution: A network for image guided depth completion. *IEEE Access*, 8:126323–126332, 2020.

[115] T.-J. Yang, Y.-H. Chen, and V. Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. *arXiv preprint arXiv:1611.05128*, 2017.

[116] Z. You, J. Ye, K. Li, Z. Xu, and P. Wang. Adversarial noise layer: Regularize neural network by adding noise. *arXiv preprint arXiv:1805.08000*, 2018.

[117] W. Yuan, X. Gu, Z. Dai, S. Zhu, and P. Tan. New crfs: Neural window fully-connected crfs for monocular depth estimation. *arXiv preprint arXiv:2203.01502*, 2022.

[118] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[119] K. Zheng, S. Li, K. Qin, Z. LI, Y. Zhao, Z. Peng, and H. Cheng. Depth estimation via sparse radar prior and driving scene semantics. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, pages 911–927, December 2022.

[120] H. Zhou, J. Lan, R. Liu, and J. Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. *arXiv preprint arXiv:1905.01067*, 2019.

List of Figures

List of Tables