Contributions to Machine Learning in Automotive Camera and Radar Perception Systems

Applications of the Sampling Method in Image Classification and Deep Learning for Radar-Based Object Detection

der Fakultät für Elektrotechnik, Informationstechnik und Medientechnik
der Bergischen Universität Wuppertal vorgelegte

Dissertation

---

zur Erlangung des akademischen Grades
eines Doktors der Ingenieurwissenschaften

von

M. Sc. Weimeng Zhu

aus

Peking, China

Wuppertal 2023

Tag der Prüfung:	05.05.2023
Hauptreferent:	Prof. Dr.-Ing. Anton Kummert
Korreferent:	Prof. Dr. rer. nat. Jörg Frochte

Abstract

Autonomous driving requires very accurate perception of the surrounding environment. Multiple sensors are used to achieve highly accurate object detection and environment perception. Among all types of sensors, cameras and automotive radars are two major sensors used in most high-end commercial vehicles.

To achieve good performance in perception tasks, machine learning and deep neural networks are utilized and have shown the advantages of robustness and generalization. Although deep neural networks were largely developed in the context of image processing, they are rarely seen in radar perception systems. In this dissertation, a new radar perception neural network with several new methods and techniques is proposed.

In recent years, the sampling methods applied in neural networks have attracted much research interest. Sampling increases the flexibility of rigid convolution kernels inside convolutional neural networks. It samples the feature map of intermediate output based on a pre-defined or dynamic data-dependent sampling grid to rearrange the location of feature vectors. Several new approaches and designs were proposed in previous works to further utilize sampling methods in advanced neural network architectures. These approaches have shown good performance in both image perception task and radar perception tasks.

Many challenges and issues are discussed and solved through the proposal of new methods, network designs or processing techniques. These applications and solutions are general, and as they are instructive in automotive use cases, they can be easily extended to other use cases.

Acknowledgement

The content has been removed for data safety reason.

Contents

1 Introduction

The automotive industry has shown great interest in autonomous driving. Various studies have been done or are in progress to reach the ultimate goal of autonomous driving. Among all methods applied in autonomous driving, machine learning (ML) and deep neural networks have shown the advantages in robust and effective modeling and control. In the recent years, along with the rapid development of computational hardware, deep learning has shown its power in problem solving and cognitive science. With respect to the automotive use cases, perception systems with cameras are largely dominated by deep neural network algorithms. In the past few years, not only perception systems, but also planning and control systems in driver assistants or semi-autonomous driving systems have been steadily improved and refreshed by deep neural networks.

Automotive radar is widely used in newly produced cars for its high accuracy in distance and velocity measurement. Compared to the high cost of high-resolution cameras or the even higher cost of LiDARs, radars have shown a good price–performance ratio in perception systems. Although they are still limited by their ability to recognize visual properties (e.g., traffic sign recognition), they are efficient and effective in detecting moving objects on the road.

Due to the increasing requirements for active safety functions on commercial vehicles, many vehicle manufacturers and suppliers are putting enormous effort and resources into research and development of advanced driver assistance systems. Automotive radars and cameras are contributing to many of these systems, providing high-quality perception results to support high-level feature functions. In the past decade, the capability of these assistance systems has grown quickly, advancing from adaptive cruise control and lane-keeping assistants to traffic jam pilots, active pedestrian protection, and robotic taxis.

In these advanced autonomous driving functions, cameras contribute to many textural detection tasks, such as traffic sign recognition, traffic light recognition, weather condition perception, and general object detection. With different design objectives, radars also contribute to various movement detection tasks, such as moving vehicle detection, speed measurement, and, in the most recent developments, general object detection by radar.

To perform complex perception tasks (e.g., general object detection), especially on camera and radar data, ML and deep learning techniques are widely used, and they are largely improving the capability of such perception systems. Convolutional neural networks are used in object detection in images. Point-cloud segmentation can be performed by neural networks on radar detection. This research topic is garnering much interest, and many new proposals are being made every day.

This dissertation will make several contributions to automotive perception tasks. In Chap. 2, fundamental knowledge in ML, computer vision, and automotive radars is first presented. Most of the basis of these research fields will be summarized, excluding detailed information on the state-of-the-art, which will be covered in more relevant chapters in later parts of this dissertation.

The following chapter will first present research in the field of image classification for an automotive use case: traffic sign recognition. Research in this application began with sampling methods in neural networks, which is a flexible technique. Sampling methods are able to largely modify the rigid manner of processing in common convolutional neural networks without a significant increase in much complexity. These findings inspired subsequent research and made major contributions to this dissertation.

After traffic sign recognition, the direction of research moved to automotive radar, a completely different world. In Chap. 4, challenges are discussed, including a lack of annotated data for automotive radar. Several methods used to collect data for ML on radars are presented, and several issues during data collection and annotation, as well as their solutions, are discussed.

Based on the above data collection, Chap. 5 proposes various methodologies and techniques for processing radar signals with neural networks. Also, a novel sampling-based recurrent network structure that has shown to be effective for automotive perception needs is introduced.

Most of the contributions discussed in this dissertation are primarily or partly based on our previous publications. These publications may have contributed to the work of other researchers in similar or relevant fields. Finally, the findings and contributions are summarized in Chap. 6. The work discussed and presented in this dissertation is not definitive but is a milestone in this research field. The following and future work are progressing toward the ultimate goal of fully autonomous driving.

The relations between the chapters of this dissertation are shown in Fig. 1.1. Chap. 2 provides the fundamentals for the research work proposed in Chap. 3 and Chap. 5. Chap. 3 introduces the proposed sampling method to the idea of sampling-based

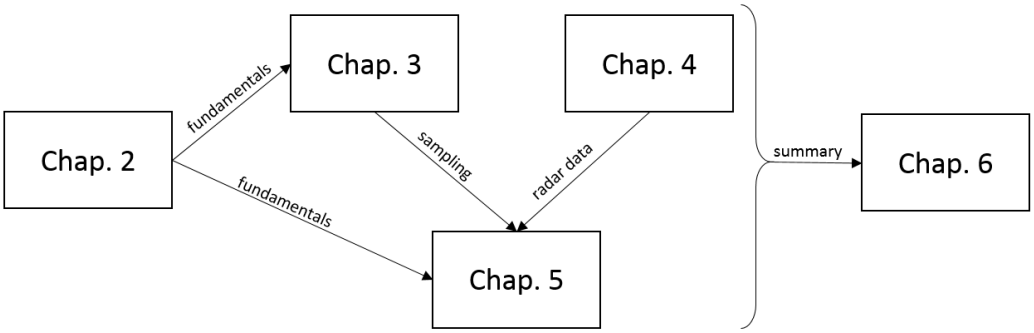

Fig. 1.1.: The relations between chapters of this dissertation.

recurrent network in Chap. 5. Chap. 4 presents the radar data used in the research work of Chap. 5. Finally, Chap. 6 summarizes all previous work and discusses the next steps.

2 Fundamentals and Related Works

In this chapter, the fundamental technologies essential to this dissertation are presented. These technologies are referred to in the following chapters and are either the basis of the advanced technologies applied or introduced in this dissertation, the basis of the input signals of the systems, or the definition of the tasks that the system solves. The introduction addresses three major fields covered in the following chapters.

First, this chapter discusses the field of ML, reviewing some of the technologies widely used for different types of learning tasks.

Second, the field of computer vision is presented, covering one of the key tasks in computer vision, that is, feature embedding. In this section, the technologies that are used in this dissertation or which this work is based on are introduced. in addition to feature embedding, several common problems are introduced that computer vision is intended to solve.

Finally, this chapter will briefly cover the field of automotive radar. Specifically, the fundamentals of this type of radar are introduced, which constitute a major part of this dissertation. This section will present the basics of radar signal processing in automotive radar. Moreover, it will present several advanced technologies to solve issues in practical use. Ultimately, the most fundamental representation of radar data will used in the latter chapters of this dissertation.

In the following chapters, the reviews of these fundamental technologies will be abridged, as they are not the main concentrations or contributions of this dissertation. The advanced developments and contributions introduced in this dissertation will correspond to only a small part of these fields, but they will still refer to or be based on these technologies.

2.1 Fundamentals of Machine Learning

Machine learning (ML) is a key sub-field of artificial intelligence (AI). The common goal of ML is to enable machines to understand data and learn to model data-related problems. Differing from normal rule-based algorithms, ML concentrates on machines generating or learning models on their own by feeding them a pile of data. Most technologies and methods within the field of ML can dynamically change their internal data models and adapt them to the data they are faced with.

ML entails programming computers to optimize a performance criterion using example data or past experience. A model is defined by a set of parameters, and learning involves the execution of a computer program to optimize the parameter of the model using the training data or past experience [Ayo10]. In this sense, machine learning programs can be separated into three major types: supervised learning, unsupervised learning, and reinforcement learning.

Supervised learning involves optimizing the model parameters using a set of annotated data, where a set of input data and a set of output targets are defined and paired. The mapping function between the input-output pairs is learned during the optimizing process [RN02].

In unsupervised learning, a model is optimized without a set of output targets paired to the input data [HS+99]. In contrast to supervised learning, the algorithm must find a common pattern within the given set of input data. The detection of a pattern is normally guided by the past experience of the programmer. A subset of the parameters for modeling is given by a human being, and the rest of the parameters are optimized by the algorithm during the learning process.

Reinforcement learning is targeted at learning of action-taking decision to maximize cumulative reward through a series of actions [SB18]. In contrast to supervised learning, labeled input-output pairs are not needed, nor a sub-optimal action series is needed to be corrected explicitly. The learning is focused on a balance between current knowledge and unknown territory. This learning method is widely used in control systems, e.g., robot control, gaming AI, and scheduling algorithms.

The following sections will explore the first two types of machine learning in greater depth, discussing more widely used technologies upon which this dissertation is based or that are used in this work. This brief introduction covers neither all research fields within ML nor all methodologies under each type. More detail about the fundamentals of ML can be found in textbooks [Mit97][RN02][Alp20].

2.1.1 Supervised Learning

Supervised learning is an ML task of learning a function or model that maps from an input data to an output data based on a set of input-output data pairs. The set of data pairs used during the optimization of model parameters is called training data. The set of input data is called input features, or simply input data, and the set of output data is called labels. The mapping function

$$f : X \to Y \tag{2.1}$$

is defined as a map from the input space X to the output space Y. A set of training examples $N = (x_1, y_1), ..., (x_n, y_n)$ is defined by the examples sampled as pairs from X and Y, such that x_i is paired with y_i for the i-th example in N. For example, if x_i were an image of a car, y_i would be its label "car".

To find the optimized parameters of function f, a loss function

$$l : Y \times Y \to \mathbb{R} \geq 0 \tag{2.2}$$

is defined to evaluate the loss between the predicted y'_i and the label y_i given input x_i. The learning objective

$$L = \frac{1}{n} \sum_{i=1}^{n} l(f(x_i), y_i) \tag{2.3}$$

is to estimate the loss based on the given training set N. The process of minimizing L over an optimization function and finding the optimized parameters of f is supervised learning.

Supervised learning requires a set of labels given as input data. The annotation of data is done mainly by human annotators or by definition. The source of annotation depends on the problem. For example, a task of defining whether the content of a picture is a car or a train is called an image classification task. Such tasks normally require annotation by humans because understanding content is a complex activity and cannot be easily pre-defined. In extreme cases, expert knowledge is necessary, for example, to distinguish between dog roses (*Rosa canina*) and field roses (*Rosa arvensis*).

Another source of annotation is to retrieve it by definitions. In the field of data mining, data are collected using a list of features. The task is to use a set of features to predict some other features. For example, knowing the age and gender of the customer combined with the current date and time, the machine should predict whether a kind of product will be purchased. In such tasks, the data from past orders

will contain purchase activity, and past visiting entries can be used as negative examples. These annotations are already given when the data are used during the definition of tasks. In such cases, no extra human effort is needed to annotate each data sample.

The annotation process is very important to supervised learning. It defines the output space of a given learning task. The complexity of output space can affect the complexity of the optimization process of finding the mapping function. It is common to separate a large complex task into multiple sub-tasks for simplicity. For instance, driving a car is a very complex task. Even for an educated human being, the process of learning to drive a car can last several weeks, even years. As a result, autonomous driving (AD) enabling a machine to drive a car is comparably or even more complex than teaching a human to drive.

In theory, it is possible to learn a model by feeding the environment information (e.g., surrounding camera images and environmental sounds) and the paired maneuver of the car to a supervised learning process. In practice, such a model is beyond the capability of any existing modeling approach. There exists research topic on end-to-end autonomous driving learning [Xia+20], but they are still not yet satisfying safety requirement and standard on real-world driving scenarios. As a fallback, a common approach is to separate driving into multiple sub-tasks [Pen+17].

As with human beings, the first major sub-task is perception. When driving a car, a person needs to keep looking ahead and around to understand the environment and situation. More than that, a person needs to hear sounds to get information that is invisible or in blind spots. Similarly, the machine needs to understand the environment and the current status of ego vehicle.

Then, a second sub-task called planning comes onto the stage. The next steps must be planned. For example, before turning left, a driver must see that there are no other cars or pedestrians to their left and have already seen the lane marks indicating the path for left turning. The driver then thinks about reducing their speed and steering the wheel to the left. A machine will do similar things. After understanding the environment, it needs to plan the maneuver of the car to achieve the target (e.g., turning left).

Finally comes the sub-task of control. After planning the next steps, a driver proceeds with a sequence of activities to perform a left turn, namely, step on brake pedal and steer the wheel. A machine needs to perform similar activities. To follow the planned path, it needs to decide how much the brake needs to be applied and how

many degrees the steering needs to be turned. Even a simple left turn is not simple for a machine.

For each task, input data needs to be collected and the objective defined. Then, one needs to annotate the data to produce labels. Finally, supervised learning is applied to obtain the mapping function. In the end, the machine uses the functions to perform AD. This dissertation concentrates on several sub-tasks in the perception field of AD. Most of the proposed approaches are supervised learning. Supervised learning is a very important method for ML of perception tasks [Gru+17]. It is not the ultimate solution for perception but is widely used in the academic and industrial worlds of AD research [Mou+18].

In general cases, supervised learning datasets are separated into a training set and a test set and, in some cases, a validation set. This separation is intended to reduce overfitting issues during learning, so the ML model can perfectly memorize all samples and their labels through the training process. In this case, the model can perform perfectly on the training set, but the learnt model is almost useless when this model is only a dedicated mapping function. The validation set is used to identify whether overfitting is happening and the test set is used to evaluate the generalization and performance of the trained model. These two sets contain similar but not identical training sets of data, so if the model is able only to memorize the labels of the training set, it will not be able to perform well on a validation or test set. When a perfect performance occurs during training but very poor performance occurs during validation, the model is already overfitting. For the three-set data (training, validation, and test), the validation is used to fine-tune the hyper-parameters and the test set is used to provide generalized performance. In this case, the hyper-parameters are not overfitting to a validation set, so this may still be a matter of an overfitting case.

In the following parts of this section, a few supervised learning approaches that are fundamental or highly related to the methods in this dissertation are introduced.

2.1.1.1. Decision Tree

Decision tree is one of the most widely used technologies in statistics, data mining and ML [Fri17]. The concept is to use a set of descriptions of a sample as features and then to make decisions based on a subset of these features and through a tree-like graph of different decision makers.

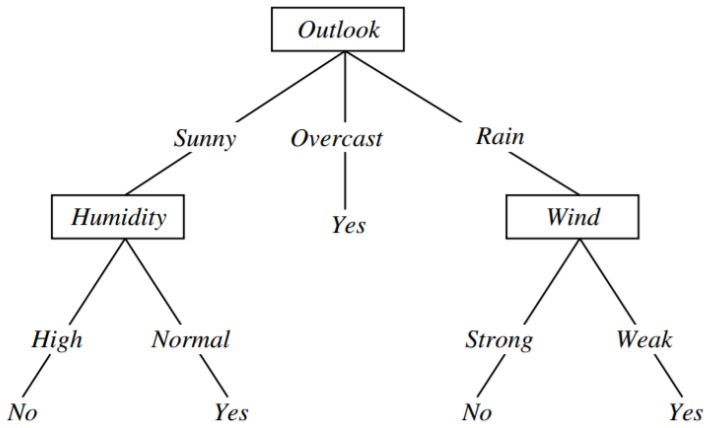

Fig. 2.1.: An example of classification tree deciding on *playtennis* [Mit97].

Fig. 2.1 shows a classification tree for deciding whether to play tennis based on a set of weather conditions. Each internal tree node specifies the feature it is analyzing, and the branch shows the value of the decision. For example, when the outlook is sunny and the humidity is normal, the person will go play tennis.

There are many modifications of and advanced approaches to decision trees, such as classification and regression tree (CART) [Bre+17], C4.5 [Qui93], chi-squared automatic interaction detection (CHAID) [Kas80], and quick, unbiased, efficient, statistical Tree (QUEST) [LS97]; these can be combined with ensemble methods, for example, random forest [Ho95][Bre01] and adaptive boosting (AdaBoost) [FS+96]. All of these methods are based on different methods of building one or multiple decision trees, but all maintain the basic concept of making decisions on a subset of features through a tree-like graph.

2.1.1.2. Support Vector Machine

Support vector machine (SVM) is a widely used binary classifier that is highly robust in classification tasks. The basic concept behind SVM is to map the training samples into a hyper space and find best separation hyperplane with the maximum margin between the positive and negative samples [Vap99].

Fig. 2.2 shows a sample hyperplane of an SVM on a linear separable dataset. The data samples are classified by a margin defined by support vectors. Research into SVM has tested different methods to find the best margin on a given dataset [STS11]. For example, an open-source linear SVM algorithm package LIBLINEAR [Fan+08] uses the coordinate descent method [Hsi+08] and the Newton method [GL21] as its advanced optimization algorithms.

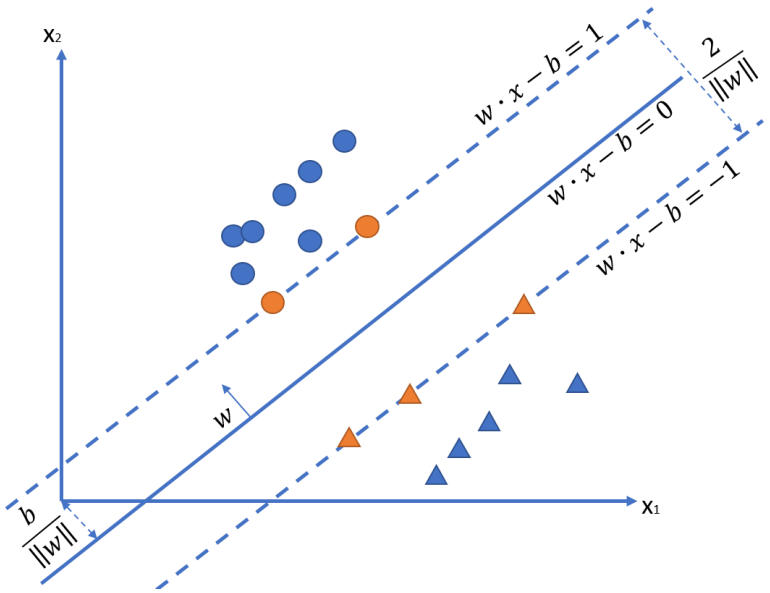

Fig. 2.2.: An example SVM on a linear separable dataset. $\mathbf{w}$ is the normal vector of the separating hyperplane. $\frac{b}{\|\mathbf{w}\|}$ is the offset of this hyperplane to the space origin. The two dashed hyperplanes are hard-margins of the classifier which is having a distance of $\frac{2}{\|\mathbf{w}\|}$.

2.1.1.3. Artificial Deep Neural Network

An artificial neural network (ANN) is a computational model inspired by the modeling of a biological neural network inside animal brains. Fig. 2.3 shows the structure of a typical neuron. Each neuron receives neural signals from other neurons via several dendrites. The signal is processed in the cell body and transferred via impulses through a unique axon. The axon has several terminals at its end and will spread signals to other neurons' dendrites. The neurons are connected via the junction between dendrites and axon terminals, forming a large network structure for signal processing and transferring. For each neuron, the signals caught by dendrites can be seen as input and the signal spread via axon terminals can be seen as output.

To simulate a similar network structure and signal processing model, the first computational model was proposed by [MP43]. Fig. 2.4 shows the modern structure of a typical ANN neuron inside an ANN. Comparable to a biological neuron, it has several input signals and several output signals. The input signals are processed in the neuron by a function f and learnable weights $\mathbf{w}$ and b. The connections between neurons by flows of input and output signals form a network structure. The gray circles indicate other neurons in this example network. Moreover, modern ANNs are designed and formed mostly in a layer-wise structure, where there is no internal connection between neurons within a layer (dashed orange box in the figure).

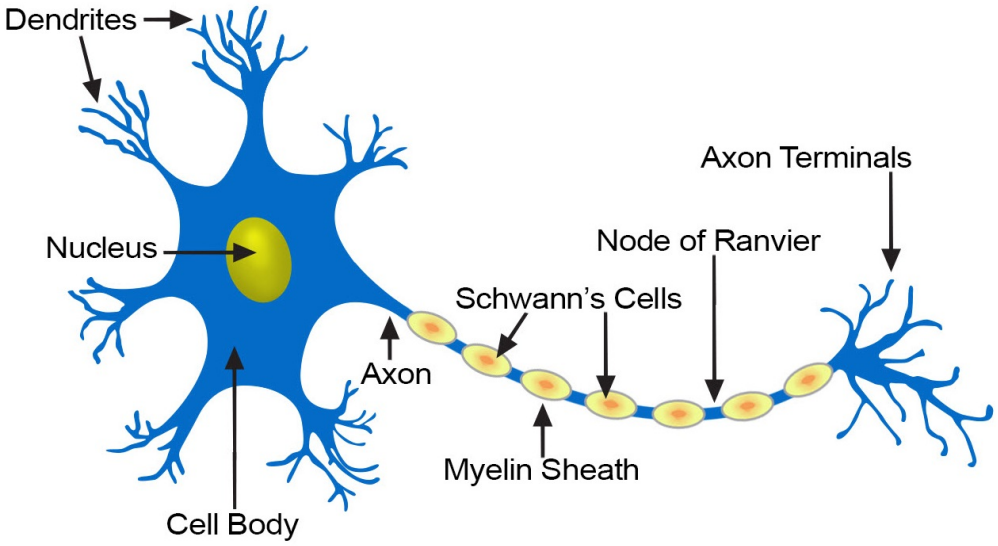

Fig. 2.3.: Structure of a typical neuron [Hea21].

In the following decades, many researchers have contributed to the development of neural network models. Even in modern ANNs, many proposed concepts and fundamental algorithms are in very early stages. The concept of a perceptron was introduced by [Ros58]. Then, the most important training approach, backpropagation, was proposed by [Wer74]. In the past decade, thanks to the high-speed development of computer hardware and computational devices (e.g., graphics processing unit (GPU)), many new research contributions have been made to the field of ANN [MS10][Abi+18].

The first and simplest type of ANN is a feedforward neural network. The data flows from input to the hidden layers and then to the output without any cyclic connections. Feedforward neural networks are constructed by a single or multiple perceptron layers. Each layer consists of a number of neurons as a function

$$y = f(\mathbf{w} \cdot \mathbf{x} + b), \tag{2.4}$$

where $\mathbf{x}$ is the input signals, f is an activation function, $\mathbf{w}$ is the learnable weight vector for weighted sum, b is the learnable bias, and y is one output of this neuron. Each neuron can accept more than one input signal from the input neurons and have more than one output that is connected to an output neuron. The activation function is normally non-linear, such as a hyperbolic tangent function or sigmoid function.

When there exists more than one perceptron layer in a feedforward neural network, it is in the form of multi-layer perceptron (MLP). The learning of model parameters

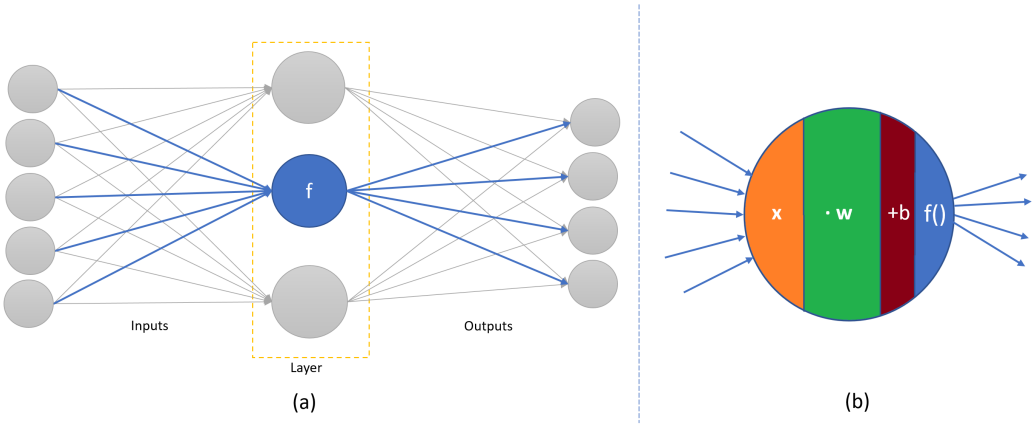

Fig. 2.4.: (b) Structure of a typical modern ANN neuron based on Eq. 2.4, (a) highlighted in blue in an ANN.

(e.g., $\mathbf{w}$, and b in Eq. 2.4) in an MLP is carried out by backpropagation. Each output of an input sample to an MLP model can be formed as

$$\hat{y}(x) = f^L(\mathbf{w}^L f^{L-1}(\mathbf{w}^{L-1} \cdots f^1(\mathbf{w}^1 x + b^1) \cdots) + b^L), \tag{2.5}$$

where $\mathbf{w}^L$ and b^L are the neuron parameters of layer L and f^L is the activation function of this layer. A loss function to evaluate the cost between the MLP prediction $\hat{y}(x)$ and the label y is defined as

$$L(\hat{y}(x), y) \tag{2.6}$$

for the given data sample (x, y).

Backpropagation is applied to minimize the loss through a gradient decent process on $\frac{\partial L}{\partial w^l}$ under chain rule [RHW86]. In this way, the gradient of loss is propagated back through the whole network. Recently, automatic differentiation packages (e.g., TensorFlow [Aba+16] and PyTorch [Pas+17]) are widely used for ANN design and model learning via backpropagation [Bay+18]. Due to the need for a loss function evaluating the cost of a known paired target of given input, most ANNs are supervised learning models. Exceptions are when the target is self-determined (e.g., Autoencoder [Kra91]) or implicit through training strategy (e.g., generative adversarial network (GAN) [Goo+14]).

The convolutional neural network (CNN) is another class of ANN that is widely used in image classification and pattern recognition. Unlike the MLP, whose neurons are fully connected, CNN uses convolution kernels to slide along the input features. The kernels are applied as sliding windows on the input features, and the weights in these kernels are kept during application. In this way, CNNs provide translation

equivalent responses to the input data [GBC16]. This property contributes to the advantage of CNN to MLP in image processing approaches. One of the earliest CNN applications is on handwritten digit recognition [LeC+89]. They claim that weight-sharing CNNs are better at generalizing features by disregarding precise locations, which are irrelevant to the classification task.

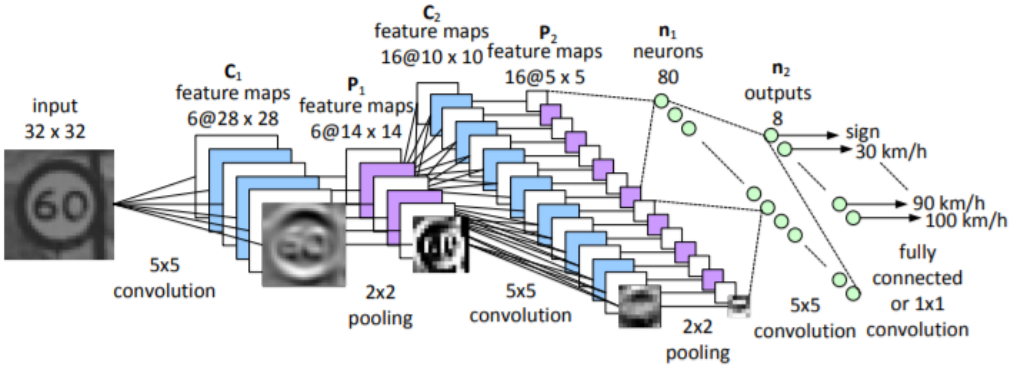

Fig. 2.5.: A typical example of CNN for speed sign classification [Pee+16]. The output of convolutional layer (noted with kernel size) is feature map **C** noted with number of maps @ map size. The output of pooling layer (noted with kernel size) is feature map P noted with number of maps @ map size. When the feature map size is reduced to only one pixel, it is noted as **n** with number of neurons or outputs.

Fig. 2.5 shows a typical example of a CNN. It consists of several convolutional layers that output feature maps, one or a few fully connected layers at the end of the network structure, and several pooling layers. Although CNN has a very small number of parameters compared to normal MLP, the computational efforts are large due to the many times the convolution kernels are applied by sliding window manner. To reduce the total computational complexity and also increase the model capability in generalization, sub-sampling of the feature maps is preferred as steps between convolutional layers. As a third important component, a non-linear function is normally applied on each feature map. Typical non-linear functions are sigmoid, hyperbolic tangent, and rectified linear unit (ReLU) [NH10].

Given the small number of parameters, CNNs are less overfitting than MLP on training data. To increase model capability, modern CNNs are becoming deeper and deeper. These ANNs all follow the universal approximation theorems [HSW89][Zho20]. Increasing the number of hidden layers in the network increases the approximation capabilities. Some recent studies have found that increasing the depth of network will lead to poorer generalization of the objective task [NRU20]. In practice, balancing between the generalization ability and the depth of the network, as well as

the width of the convolution kernels, is an important topic and task for the network structure design.

In most recent research, even CNNs without any fully connected layers have been proposed for many tasks. Semantic segmentation is a very typical task for a fully convolutional network (FCN) [LSD15]. Other than outputting a single class of the whole image, semantic segmentation outputs the classes of each pixel of the image. FCN shows the possibilities and abilities of CNN in image processing and the advantages of convolution kernels. More and more modern CNNs are no longer using fully connected layers for pixel or part-level tasks [Dai+16a].

When processing with time series data, recurrent neural network (RNN)s are among the most used ANN types. As an extension of ANN, RNN adds connections between nodes in a temporal sequence graph. In the design of RNNs, the network has one or several temporal states as their memory of the processed input data. The network is capable of processing the input data of the current time together with their memory of the history. In theory, RNNs are Turing-complete in that they are capable of processing arbitrary inputs with arbitrary programs if perfectly designed with appropriate parameters [Hyö96].

In applications, RNNs are used in many speech recognition, machine translation, and video processing tasks [VKJ11][Sin+17][LG15]. Given the ability to process arbitrary lengths of sequential input, it is easily applied in these tasks without much overhead on sequential input pre-processing.

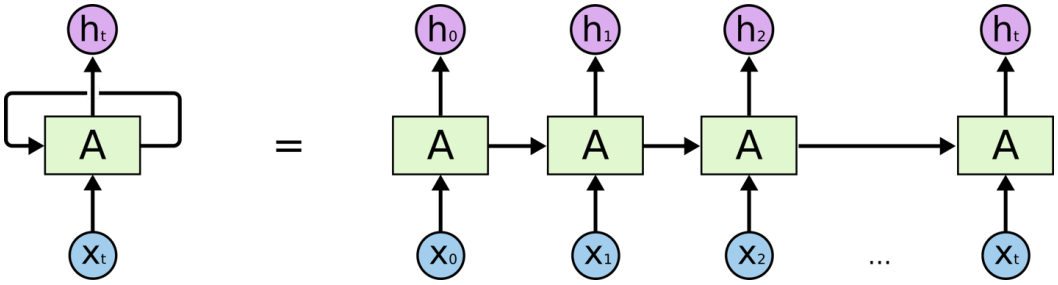

Fig. 2.6.: A typical example of RNN with an unrolled version.

Fig. 2.6 shows a typical RNN module and its unrolling. It has a sequential input X_t, a processing function A (e.g., a single layer perceptron), a state memory in recurrent form, and an output h_t. After unrolling the recurrent connection, it is clearer how the memory is transferred in the temporal space in an RNN.

One limitation of RNN is the complexity of backpropagation during the training phase. There are two major issues with backpropagation in RNNs, namely vanishing gradients and exploding gradients [PMB13]. To address these issues, several

research works have proposed new structures of RNN modules, e.g., LSTM (details in Sec. 5.4.1.1), GRU [Cho+14][DH20] (details in Sec. 5.4.1.2), and IndRNN [Li+18].

When having more layers in ANN and combining with CNN, RNN, and MLP, the large deep neural network is in the field of deep learning (DL) [DY14]. DL is still an ML research field based on ANN. Most of the development and proposals in these ANN fields are also beneficial for DL improvements.

In later chapters of this dissertation, several research topics on CNN, RNN, and DL are covered. The most relevant details of some previous research works will be discussed there.

2.1.2 Unsupervised Learning

When training data are not provided with labels, unsupervised learning algorithms are considered and utilized to perform ML tasks. One of the most common tasks is clustering. Clustering is a task that needs to group similar samples within a dataset and separate dissimilar samples from these groups. It is a common technique for statistical data analysis and is widely used in many fields (e.g., pattern recognition, information retrieval, and ML).

One type of clustering algorithms is centroid-based clustering. These algorithms represent the clusters by central vectors in the feature space. The typical method is to assign each sample to the nearest center through a distancing function. Optimization processes for finding the best central vectors are the keys in these clustering algorithms.

K-means is a well-known centroid-based clustering algorithm [Llo82]. Given a dataset $X = x_1, x_2, \ldots, x_n$ and a pre-defined number of clusters $C = c_1, c_2, ..., c_k$ sharing the same space definition of X, the objective of k-means is to find an optimal set of clusters C from all possible sets of clusters $\mathbb{C}$ by

$$\underset{\mathbf{C}\in\mathbb{C}}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{\mathbf{x}\in c_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 \tag{2.7}$$

where μ_i is the mean of the samples in c_i.

In the standard optimization, K-means initializes each cluster centroid by a random sample in the dataset. Each sample can only be assigned to one cluster. During optimization iteration t, each sample x_p is assigned to each cluster by

$$c_i^{(t)} = \left\{ x_p : \left\| x_p - \mu_i^{(t-1)} \right\|^2 \leq \left\| x_p - \mu_j^{(t-1)} \right\|^2 \ \forall j, 1 \leq j \leq k \right\}. \tag{2.8}$$

After assignment, each cluster centroid is updated by

$$\mu_i^{(t)} = \frac{1}{\left| c_i^{(t)} \right|} \sum_{x_j \in c_i^{(t)}} x_j. \tag{2.9}$$

When the assignment is no longer updated, the optimization algorithm converges. However, the algorithm is not guaranteed to always converge, and the algorithm is NP-hard [Alo+09]. A variety of researchers have proposed new optimizations or advanced algorithms for k-means (e.g., k-means++ [AV06], k-medoids [KR90], and fuzzy c-means [Dun73][Bez13]).

Another type of clustering algorithm is density-based clustering. Unlike the centroid-based methods, they define the clusters by grouping the samples using data density. In this way, the samples in the same cluster may not necessarily be nearest to the cluster centroid. The most famous density-based clustering algorithm is density-based spatial clustering of applications with noise (DBSCAN) [Est+96].

The DBSCAN algorithm can be simplified in the following steps [Sch+17]:

1. Compute neighbors of each point and identify core points (*Identify core points*)
2. Join neighboring core points into clusters (*Assign core points*)
3. **foreach** non-core point **do**
4. Add to a neighboring core point if possible (*Assign border points*)
5. Otherwise, add to noise (*Assign noise points*)

Each sample point in the dataset is either assigned to a cluster as a core point or border point, or is treated as a noise point. It has robustness toward outliers and does not require a pre-defined number of clusters. The computational complexity is also approximately as low as $O(n^2)$ in basic implementation and it can be reduced to $O(n * log(n))$ in average with R*-tree query.

In automotive sensor processing, DBSCAN is widely used in processing point-cloud data and object detection tasks [Hua+19][KKD12][LLK18]. For object detection

by automotive radars, DBSCAN has an advantage in non-rigid object detection and clustering [Inc21].

The latter chapters of this dissertation will use some of the unsupervised learning techniques to improve the proposed methods and solve faced issues.

2.2 Fundamentals of Computer Vision

Computer vision (CV) is a scientific research field that solves questions about how computers can understand images and videos like human beings. The main goal of CV is to perform tasks that human vision systems can do [BB82]. CV research covers several different fields, including image acquisition, image processing, image analysis, and understanding. Digitization of images is one of the most important topics in CV. It first defines the raw representation and the systematic limitation of the input data to any following image processing pipelines. For example, the design of the lens will affect the field of view (FOV) of the image sensor. Different FOV will lead to different image quality, hence, different performance of image analysis [Miy+20]. Such differences are large enough to define different data distributions or even data domains for the same CV task [Car+19].

Another important research field is image representation. Digital images can contain metadata and can also be compressed for data storage and transfer. Image representation is about how the information in an image is formed and structured. The most common method is to save images in pixel form, which stores the image in a large matrix where each value represents the information of a specific location in the image. The information can be, for example, color coding or brightness. To save a large valued matrix is very costly, several compression methods are proposed to reduce the size of digital images [VSP13], most commonly JPEG, GIF, and PNG.

When analyzing the image content, features are extracted from images and formed in a way that machines can make use of for the understanding tasks. There are various methods for image feature extraction [KB14]. Feature extraction will help the image processing algorithms understand the most important information in each image and ignore the variances and noises. A good feature extraction should provide generalization of the content that is shared with the same objective. One of the most famous applications is optical character recognition (OCR). It is an application that can process digital documents and digitize its content into characters. Various researchers have been involved in and contributed to OCR tasks [IIN17]. The development of OCR has also reflected the development of image feature extraction

[SD18]. The following section will introduce a few feature extraction methods that are most related to the methods in this dissertation.

CV is a very large scientific research field. This dissertation covers and contributes to only a very small subset of the methods and tasks. The following sections will also introduce a few common tasks in CV that are highly involved in the contribution of this dissertation.

2.2.1 Image Feature Embedding

Image feature embedding techniques are very important in the generalization of image information and compression of the large matrix-based image representation. It is normally used in image processing to map the image into a feature space that an ML algorithm can be applied. The quality of such embeddings has a large effect on the ML algorithm performance.

One traditional type of image embedding is the image descriptor. These descriptors take an image as input, and output feature vectors of this image. The feature vector is designed with consideration for the need of the task (e.g., Canny edge detector [Can86] for detecting edges, Harris corner detector for localization of corners [HS+88], and histogram of oriented gradients (HOG) for shape description [DT05]).

With the development of computational hardware, CNNs dominate image feature embedding techniques [O'M+19]. One advantage of CNNs is end-to-end training. Dissimilar to the traditional methods that require hand-craft features or design of functions, CNN parameters are trained in a "black-box" manner. Even without much expert knowledge inference, CNN can still benefit from supervised learning. Also, with the development of DL and improvements in GPUs, CNN shows high capability in image processing and feature extraction.

In the following sections, more details about HOG and CNN in feature embedding of images will be shown. These two methods are highly related to this dissertation in data pre-processing or in the contribution of this dissertation.

2.2.1.1. Histogram of Oriented Gradients

HOG is designed to describe the shape of an object in an image. Differing from edge detection, HOG is capable not only of finding the edges but also giving the

orientation of the edges. In this manner, it can also describe the shape in statistical form.

After normalizing the image in pre-processing, the gradient and the orientation of each pixel in the image are calculated through Sobel filters

$$[-1, 0, 1] \text{ and } [-1, 0, 1]^\top \tag{2.10}$$

on image I, whose gradients will result in

$$\begin{aligned} G_h(x, y) &= I(x, y+1) - I(x, y-1) \\ \text{and } G_v(x, y) &= I(x+1, y) - I(x-1, y) \end{aligned} \tag{2.11}$$

for the horizontal gradient G_h and vertical gradient G_v at the image pixel (x, y). The magnitude of gradient G is

$$G = \sqrt{{G_h}^2 + {G_v}^2}, \tag{2.12}$$

and the orientation angle θ is

$$\theta = \text{atan2}(G_v, G_h), \tag{2.13}$$

where $\text{atan2}()$ is the operation of the 2-argument arctangent defined as

$$\text{atan2}(y, x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{if } x > 0, \\ \frac{\pi}{2} - \arctan\left(\frac{x}{y}\right) & \text{if } y > 0, \\ -\frac{\pi}{2} - \arctan\left(\frac{x}{y}\right) & \text{if } y < 0, \\ \arctan\left(\frac{y}{x}\right) \pm \pi & \text{if } x < 0, \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0. \end{cases} \tag{2.14}$$

The orientation angle is binned into a pre-defined angle binning vector, where the voting weight is from the magnitude.

The image is separated into several cells, and the statistics of the angle binning vector are calculated for each pixel in the cell. Then, the histograms of the binning vectors of each cell are gathered. A sliding-window block on the cells gathers and normalizes the histograms of underlying cells. In the end, summing up or concatenating the histograms of all blocks results in a HOG feature vector of the image. Fig. 2.7 shows the steps of calculating the HOG feature vector in an example pedestrian detector.

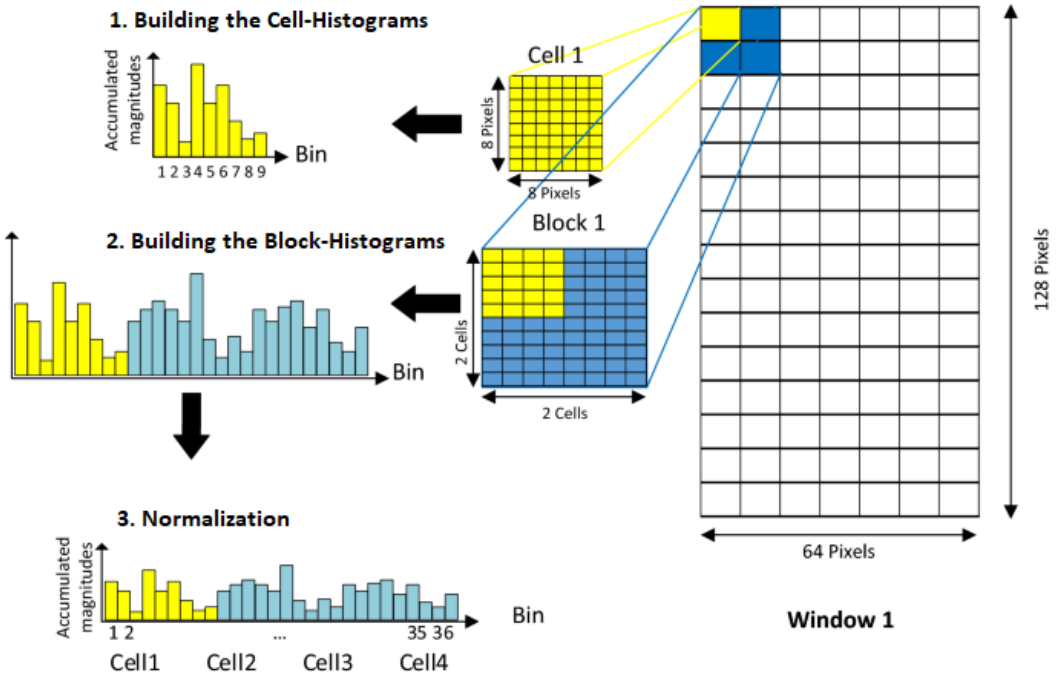

Fig. 2.7.: An example of HOG in a pedestrian detector [Hel+20].

HOG is widely used in object detection tasks due to its robustness of localization and ability to describe shapes. Also, the feature vector of HOG can be easily used in multiple classifiers (e.g., SVM [DT05]).

2.2.1.2. Convolutional Neural Network

In the past decade, CNN has shown its advantage in image feature extraction that has outperformed the traditional methods (e.g., HOG [Sul+17]). CNN has many convolution kernels sliding through images, which is similar to Sobel filters or other filters for pattern recognition. Some researchers have found that the end-to-end learned convolution kernels show some sort of pattern recognition (e.g., edge detection [KZS+15]). Although, as a "black-box" model and non-linear functions, it is very tricky for human beings to understand exactly how CNN works.

Nevertheless, CNN has shown its great capability in image feature extraction and advantage over hand-crafted feature descriptors. Thanks to end-to-end supervised learning, the performance of CV algorithms has been significantly increased by the contribution of CNNs [Kha+20]. Researchers have shown that DL has comparable performance to [Rus+15] or outperforms human beings [Gei+17] in several CV tasks.

2.2.2 Computer Vision Tasks

Typical tasks in CV are spread in all sub-fields in the pipeline, including image acquisition, low-level feature extraction, multi-sensor fusion, segmentation, object detection, image classification, and so on [FP11].

This section briefly introduces a few CV tasks that are highly related to the contributions in this dissertation. Details and examples of each task are presented in Sec. A.1.

2.2.2.1. Image Classification

Image classification is the task of assigning a label from a fixed set of categories to an image, normally to the major object in the image. The objective of this task is to identify which thing is presented in an image.

2.2.2.2. Image Object Detection

Image object detection (OD) is a task to detect objects and localize them in an image. It may also be combined with an image classification task to identify the category of a detected object. The objective of this task is to identify the different things presented in an image.

2.2.2.3. Image Segmentation

Image segmentation is the task of partitioning an image into different segments. Differing from OD, the objective is to understand the organization of pixels in an image. Instead of identifying objects, the task is targeted to identify areas.

2.3 Fundamentals of Automotive Radar

This section will cover some basics of automotive radar sensors. Radars are one of the most well-known sensors in the military world. Radio detection and ranging (RADAR) is a detection system that uses radio waves to determine the distance, detection angle and/or velocity of objects. It was majorly developed in World War II by the UK and US for military use [Goe13]. Radar systems consist of a transmitter

generating radio waves, sending out the waves by a transmitting antenna, a receiving antenna receiving the reflected waves after hitting the objects and a processor to determine the various properties of the objects.

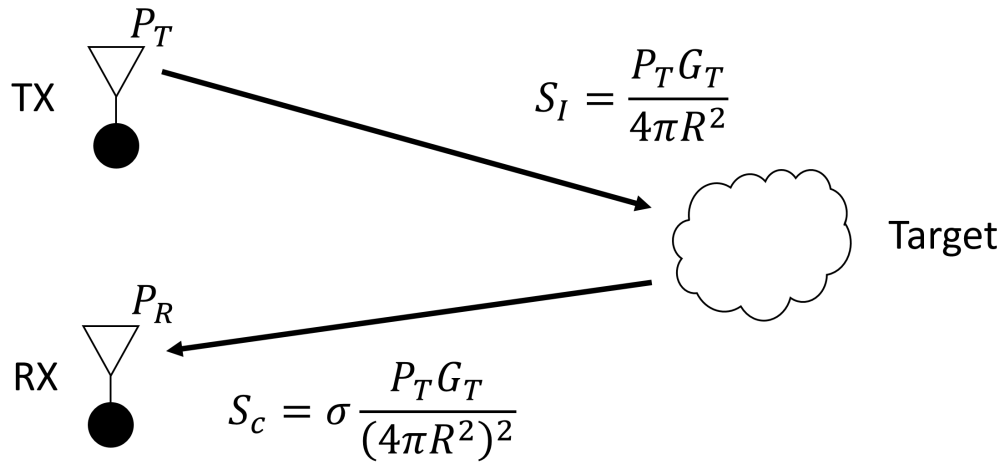

Fig. 2.8.: An illustration of radar power transmitting and receiving by antennas.

The power received by the receiving antenna of a given transmitting antenna is determined by the radar equation, where power

$$P_R = \frac{P_T G_T}{4\pi R^2} \frac{\sigma}{4\pi R^2} A_e = \frac{P_T G_T A_e \sigma}{(4\pi)^2 R^4} \tag{2.15}$$

is determined by several aspects [Mer+01]. Fig. 2.8 shows a brief illustration of the power transmitting and receiving. First, the power hitting the object

$$S_I = \frac{P_T G_T}{4\pi R^2} \tag{2.16}$$

is determined by the transmitter power P_T, the gain of transmitting antenna (TX) G_T and the distance from the antenna to the object R. It is clear that the power is decayed by R^2 when hitting the target. Then the power at receiving antenna (RX)

$$S_c = S_I \frac{\sigma}{4\pi R^2} = \sigma \frac{P_T G_T}{(4\pi R^2)^2} \tag{2.17}$$

is further affected by the reflection surface property radar cross section (RCS) σ which is an electromagnetic signature of the object. It is affected by the size, the material, and the relative surface angles to the waves of the object [Ali17]. The

power is again decayed by R^2 when receiving. In the end, the receiving antenna further affects the received power by effective aperture

$$A_e = \frac{G_r \lambda^2}{4\pi}, \tag{2.18}$$

where the gain of receiver antenna G_r and the wavelength λ are the factors. The received power on a specific radar system is proportional to

$$P_R \propto \frac{\sigma}{R^4}, \tag{2.19}$$

where the object property σ and distance R are the factors. When considering radar systems, it is obvious that the objects are easily detectable when they have a good reflection surface with enough size, good material (normally metal) and good surface angle (perpendicular to the wave, or in the form of a corner reflector). Also, the further the object is, the much harder it can be detected as the decay is in the fourth power of the distance.

2.3.1 Automotive Radar Signal Processing

In the modern automotive industry, the most commonly used type of radar is frequency modulated continuous wave (FMCW) radar. These radars have the advantages of measuring range, angle, and Doppler velocity at the same time, low cost in manufacture, stable enough without mechanical moving parts, and also small enough to be seamlessly integrated into car design [Sch05].

Continuous wave (CW) radars transmit a continuous radio signal with a given frequency. In this way, the Doppler effect is recognizable when the object is in relative movement to the radar by the detection of change of the frequency. In such a manner, CW radars are capable of measuring the relative velocity of the object, but not the distance when transmitting only one single frequency with unmodulated CW.

2.3.1.1. Signal Processing of FMCW Radars

The signal processing of FMCW radars is very complex and is not the major contribution of this dissertation. Only a very brief introduction will be covered in

this section. Technical and mathematical details can be found in the literature [Mer+01][Win07].

FMCW radars transmit signals with frequency changes over time. Fig. 2.9 shows an example of such a frequency change. Within the time frame of T, in total, K chirps are sent out and received. The frequency changes over bandwidth B from a base frequency of f_0. With multiple such (virtual) antennas, a multiple input multiple output (MIMO) FMCW radar is capable of detecting objects with aspects of distance (range), speed (Doppler), and direction of arrival (DOA).

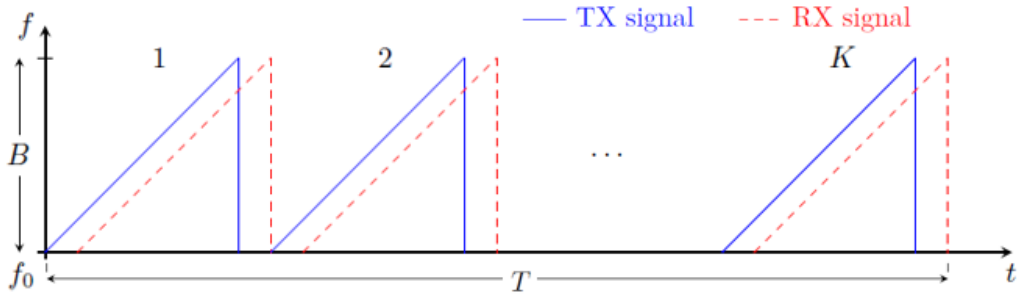

Fig. 2.9.: An example of frequency changes over time in FMCW radar.

The FMCW radar signal processing can be considered as a 3-D fast Fourier transform (FFT) process on a cube of each chirp, chirps in each time frame, and (virtual) antennas. Fig. 2.10 shows an illustration of how the cube is transformed by FFT. After processing, the cube forms a range-Doppler-angle (RDA) cube.

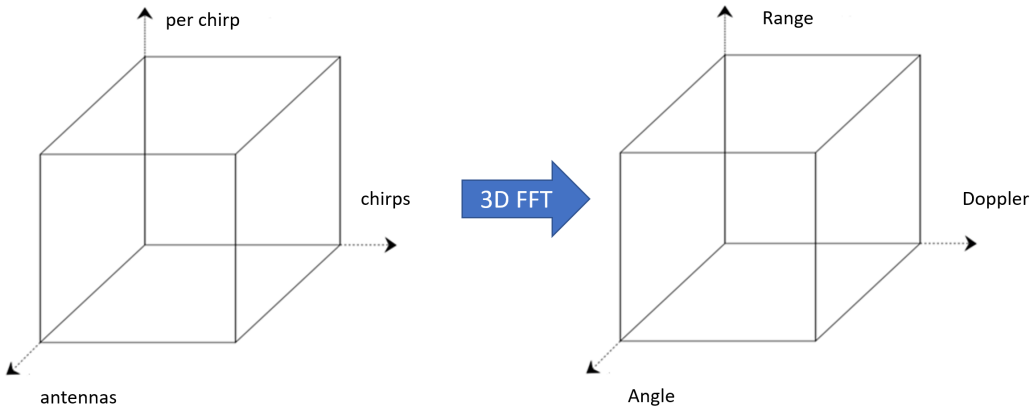

Fig. 2.10.: 3-D FFT on FMCW radar signals.

2.3.1.2. Object Extraction

Although the RDA cube is dense and large, the actual objects are sparse and few. To extract objects from this cube, the constant false alarm rate (CFAR) algorithm is commonly used. This algorithm attempts to identify the peaks in an array with an adaptive noise floor. Fig. 2.11 shows an example of how CFAR works on array data. The algorithm works in a sliding window manner. In cell-averaging CFAR, it takes the surrounding data as a training set to determine the noise floor

$$F_n = \frac{1}{2N_T} \sum_{n=1}^{2N_T} x_n, \tag{2.20}$$

where all samples x_n in the training set at left $(1 \dots N_T)$ and training set at right $(N_T + 1 \dots 2N_T)$ are averaged. If the cell under test (CUT) is above the threshold

$$T = \alpha F_n, \tag{2.21}$$

the CUT is kept. The α is a threshold scaling factor; in a simple form, it can be defined as

$$\alpha = P_{fa}^{-\frac{1}{2N_T}} - 1, \tag{2.22}$$

where P_{fa} is the desired false alarm rate. Several improvements to call-averaging CFAR have been proposed by researchers in the past decades (e.g., ordered-statistic CFAR [Roh83][Roh11]).

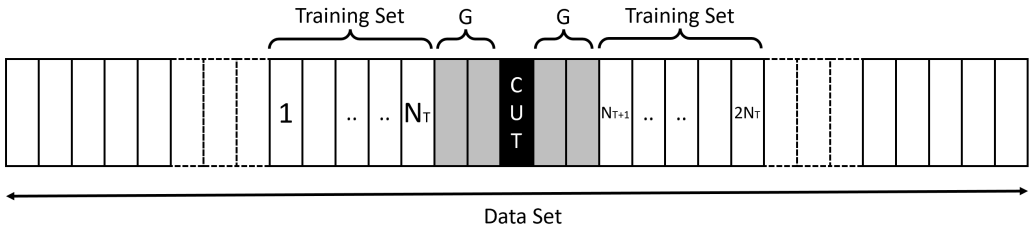

Fig. 2.11.: An example of the CFAR algorithm.

The CFAR algorithm is very efficient in target extraction in a dense array. After CFAR processing, the RDA cube is then very sparse and contains only interesting objects. Fig. 2.12 shows an example range-Doppler matrix after CFAR. The targets

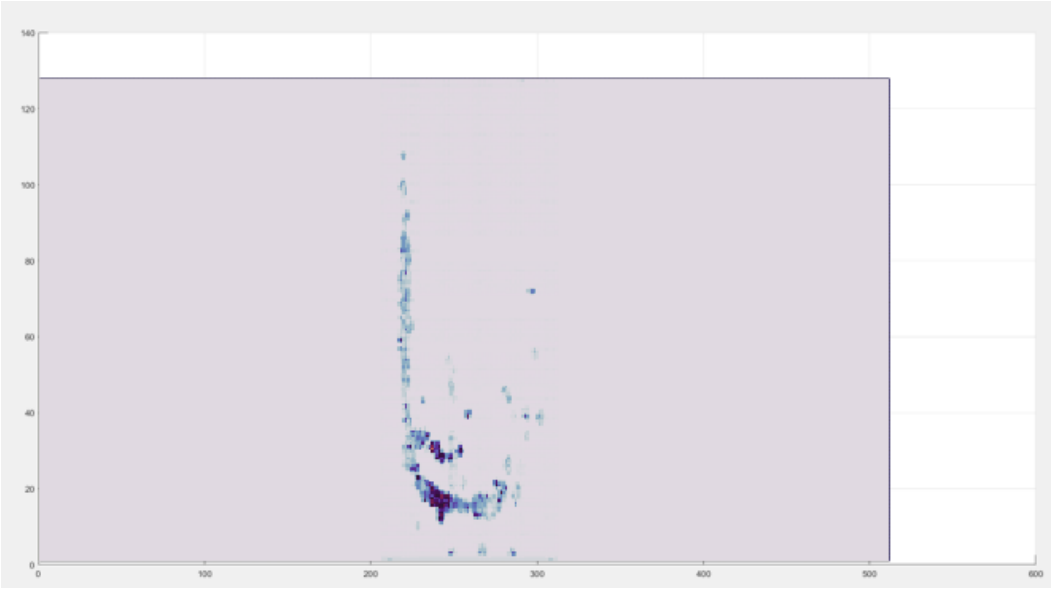

Fig. 2.12.: An example of a range-Doppler matrix after CFAR.

that survived after CFAR are presented in non-purple colors. The color corresponds to the average magnitude of the antenna array.

In practice, due to the computational complexity of multi-object solutions and super-resolution for angle estimation, CFAR is processed on a range-Doppler-antenna cube before angular FFT is applied. The extracted targets will then be applied with angular FFT or other angle estimation algorithms. The range-Doppler-antenna cube is called compressed data cube (CDC) in this dissertation. After angle estimation, the targets in the CDC are radar detections. They consist of information of distance (range), velocity (Doppler), DOA (angle), and magnitude (RCS). This information will be further used in the following pipeline to determine the actual objects in the environment.

In the following chapters in this dissertation, several contributions to signal processing and object detection on radar detection data are presented and discussed. These data are presented in the form of a point cloud of the surrounding environment of the sensor. This point cloud is in a polar coordinate system where the distance and angle are axis descriptors. Each data sample (radar detection) consists of a few meta information, such as velocity and RCS. The detailed data description and post-processing will be discussed and presented in Chap. 4.

3 A New Sampling Method for Image Classification with Neural Networks

Image classification is one of the most important perception tasks in AD. The task concentrates on classifying an image into a pre-defined category. This task is also commonly combined with the OD task. In automotive scenarios, traffic sign recognition and head/tail light recognition are common perception tasks used in several advanced driver-assistance system (ADAS) applications. Compared to other sensors like ultrasonic, infrared, or even advanced radar sensors, cameras are still essential in recognition of vehicle lights or traffic signs.

With recent hardware improvements, DL using ANN shows its advantages in CV tasks. With the characteristic of maintaining spatial relations during processing, CNNs produce top performance [LB98], especially in visual recognition tasks, such as image classification [Kri+12], semantic segmentation [LSD15], and object detection [Gir+14]. Not only in academic research, CNNs are also widely used in the automotive industry. Its strength in image classification is improving performance in ADASs (e.g., traffic sign recognition [Li+16] or pedestrian detection [ZT00]).

This chapter introduces a novel sampling method for CNNs in image classification tasks. Most of the content is based on our previous publications and contributions of dense spatial translation networks (DSTNs) [ZSK18][ZS19a][ZS19b][ZS19c].

3.1 Traffic Sign Recognition and Computer Vision Challenges

Traffic sign recognition (TSR) is an image classification task in ADASs. The task is to detect and classify a traffic sign into its category. It is useful in speed-limit warning, driving assistant, or even advanced AD. A traffic sign can vary its appearance in the output of an image sensor (e.g., camera) for multiple reasons. Weather conditions,

environment, mounting pose, and viewing angle can change the appearance. However, one of the common cases is the difference of design across countries. Fig. 3.1 shows such differences for an end of speed limit of 80km/h sign. Signs (a) and (b) are taken in different situations with the variance in environments or perspective. But even in the case that situations match perfectly, the sign may still look quite different in different countries. For example, the Italian sign (c) has a significant variance not only in the font of the number 80 but also in the rotation angle and width of the termination line.

Although CNNs make strong visual recognition systems, the design of their convolutional kernels has the drawback of processing the input data using a rigid spatial pattern. It lacks generalization in exploiting unknown and complex spatial neighborhood relations. For the variation in traffic signs, a vanilla CNN lacks the capacity to deal with it, other than through considerably increasing the network complexity.

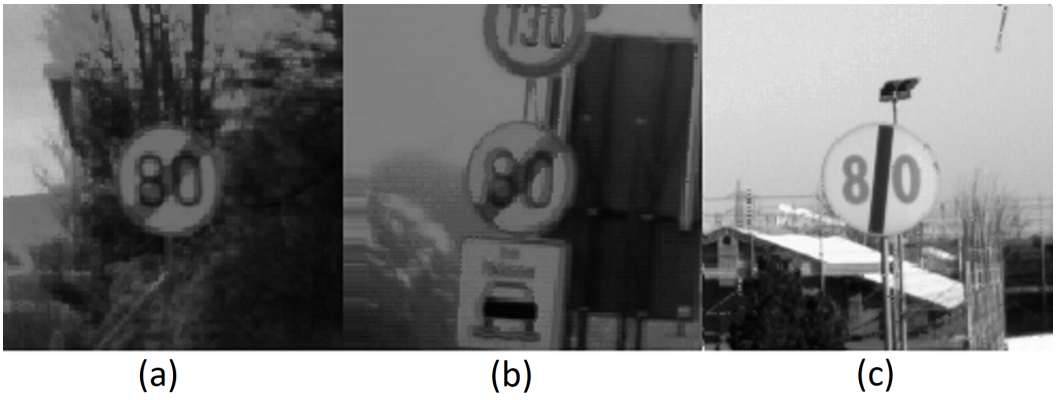

Fig. 3.1.: Real-world examples of an end-of-speed-limit of 80km/h sign. (a)&(b): Images of a German sign taken in different situations; (c): An Italian sign.

Recently, much research has been working to overcome this drawback in ordinary CNN. For example, a pooling layer reduces spatial resolution by a fixed ratio and invariant kernel. Region of interest (ROI) pooling can pool the values based on ROIs, which are automatically proposed by a network [Dai+16b][Gir15]. Dynamic filter networks apply different convolutional kernels conditioned on input to make convolutional layers flexible [Jia+16].

In the following sections, a new method will be introduced to increase the flexibility of CNN kernels to improve the performance of classification of highly variant images.

3.2 A New Sampling Method in a Convolutional Neural Network: Dense Spatial Translation Network

In this section, one of the major contributions to the improvement of CNN kernels by a sampling method is presented. In this work, a sampling method is added inside a CNN to increase the flexibility of feature embedding on highly variant image data.

The network structure of this method and several strategies for supervised learning of integrated neural networks are also presented. The experiment results show that the problems from such variations can be reduced, and the performance of image classification is improved with the use of the new method. Apart from TSR tasks, an experiment is conducted on a number of classification tasks with high variance, where the task is also improved by this method.

3.2.1 State of the Art

Knowing the high rate and variability of geometric transformations in real-world images, a spatial transformer network (STN) can compensate for a pre-defined type of transformation with the parameters conditioned on their input [Jad+15]. Fig. **??** shows the structure of STN. STNs are trained in an end-to-end manner, which could be integrated into another general recognition framework. A significant improvement is shown in a general real-image number recognition task on the street view house number (SVHN) dataset by using STN.

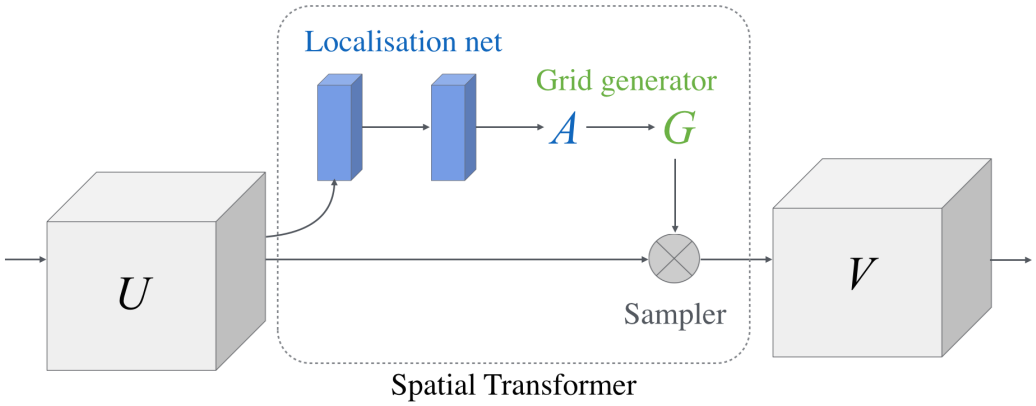

Fig. 3.2.: Network structure of STN [Jad+15]. U is the input feature tensor. A is affine transformation parameters generated from localization network. G is grid generation based on given affine transformation parameters. V is the output feature tensor after affine transformation.

A deformable convolutional network (DCNN) can produce a more generalized spatial transformation in convolutional kernels and ROI pooling kernels [Dai+17]. The authors of DCNN changed the value acquisition in these kernels by applying offsets conditioned on the input. Such deformable convolutional kernels (Fig. **??**) and deformable ROI pooling kernels can see and read feature values from a location outside of the fixed kernel window. It increases the network's capability to deal with complex deformations. DCNNs can also be trained in an end-to-end manner, and can directly replace any existing convolutional layer or ROI pooling layer.

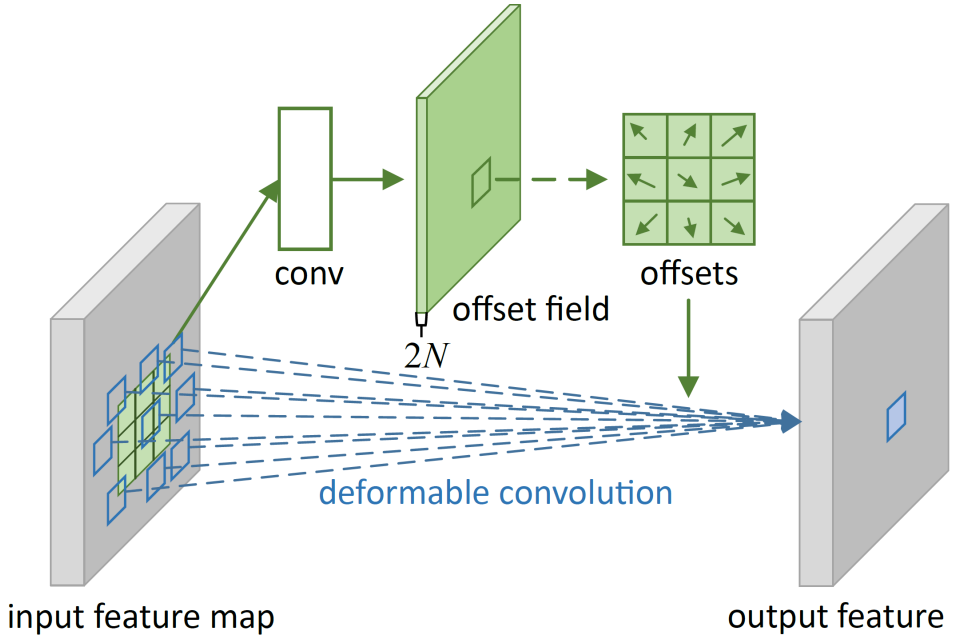

Fig. 3.3.: Network structure of DCNN [Dai+17]. The single convolution layer creates offset field for N pixels in two dimensions ($2N$), and applies deformable convolution based on the offsets.

The main contribution in this chapter uses the advantages of input-conditioned transformation in STN and generalized deformation in DCNN. Considering the efficiency of a vehicle-embedded system, and to combine these advantages, a novel network module is introduced called DSTN. DSTN can apply generalized dense translations conditioned on the input, and produce a reusable feature map that can be fed into multiple following network components without re-computation. This lowers computational effort. Furthermore, for the sake of simplicity, DSTN is designed in an end-to-end trainable manner and discusses beneficial training strategies.

According to experiments on public datasets and a private industrial real-image dataset, DSTN shows significant improvements in image classification tasks. Further-

more, it consumes only a little more computation resources, so it is still applicable to industrial ADAS products, especially on the low-power embedded hardware found in vehicles nowadays.

3.2.2 Problems and Challenges

In real-life automotive scenarios, one usually has difficulties using very deep networks or a large number of channels for intermediate feature matrices due to the limitation of embedded hardware. If the neural network uses fewer feature channels or is shallow in depth, it will decrease its capacity for modeling or representation [Sun+16]. To increase this capacity without using a deeper network or more feature channels, the layers can be designed to be more suitable for the target application. Thus, if knowing an application might benefit from spatial rectification of the input data (or intermediate channels), a dedicated sampling methodology can be applied. Inspired by STN [Jad+15], such a sampling structure will reduce the variance in input images. This will help the network to produce similar intermediate representations for the images in the same (sub-)class.

One of the drawbacks of STNs is that they can only deal with a pre-defined type of global image transformation (e.g., affine, or similarity transformation). Such a transformation cannot easily solve or reduce the problem described above, as it requires highly local displacements to warp the target object into a common appearance. To utilize the sampling pipeline structure in STNs, DSTN is proposed. DSTN changes the transformation part in STN, replacing it with a dense offset-based translation, which is similar to the offset idea in DCNN [Dai+17]. One advantage of this design is that it is a stand-alone network layer that could be integrated into any existing CNN.

3.2.3 DSTN Structure

The structure of the DSTN is shown in Fig. 3.4. An output feature map V (5) is sampled from the input feature map I (1). Due to this sampling, a DSTN layer could be added between any two network layers. It will not perform any other complex calculation considerably changing the shape of the feature maps (e.g., convolution or pooling). During the sampling, a localization network (2) is applied to the input map, and outputs an offset field F_δ (3) of the sampling grid in the size of the output map. A sampling grid G_s is finally generated (4) based on these offsets and is used

by sampler (6) to produce the output map. The special component Stop Gradient (7) is described in detail in Sec. 3.2.5.

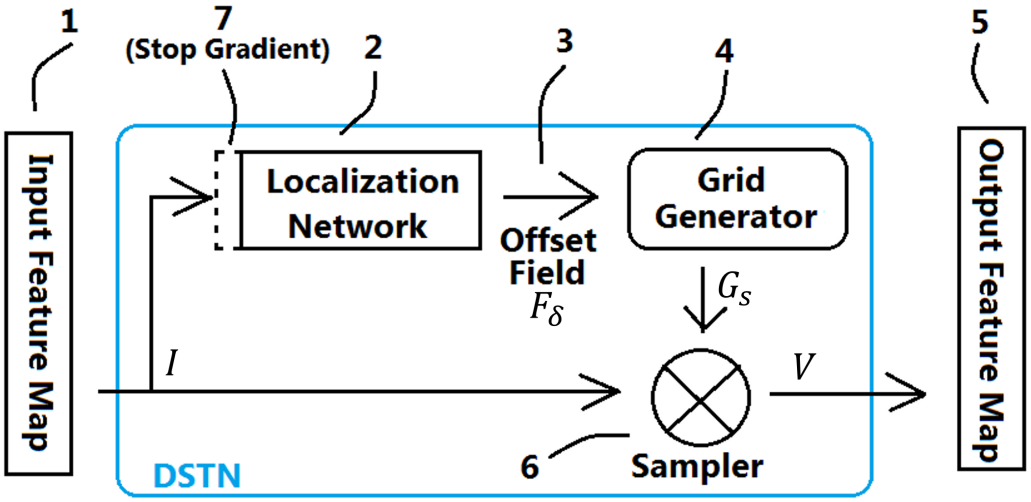

Fig. 3.4.: An abstract structure of DSTN. (Numerical marks are described in the text.)

The localization network receives the input feature map, and produces an offset field according to the size of the output feature map. One simple way of designing the localization network is a sub-network producing a two-channel offset field F_δ in the size of the output map ($H_O \times W_O$), where each channel corresponds to horizontal and vertical offset. This localization network is trained in an end-to-end manner together with the major task of the neural network (e.g., image classification).

The grid generator produces a sampling grid

$$G_s = G_0 + F_\delta, \tag{3.1}$$

which is adding the offset field F_δ to the uniform basic grid G_0. The uniform basic grid contains coordinate indices of each pixel of output feature map, uniformly distributed with regard to the input feature map size. For example, an all-zero offset field would result in a resizing of the input map (if the output map size varies), or an identical input map (if the output map size remains constant). This generated sampling grid will be used in the sampler as indication of value acquisition location from the input feature map.

The sampler in DSTN is under the same design as in [Jad+15]. The sampled value V_i^c of channel c at pixel i (corresponding to x_i, y_i coordinates in the uniform basic grid) in the output feature map (5) can be calculated using bi-linear interpolation as

$$V_i^c = \sum_h^{H_I} \sum_w^{W_I} I^c_{(h,w)} max(0, 1 - |x_i^s - h|) max(0, 1 - |y_i^s - w|), \tag{3.2}$$

where H_I and W_I are the dimensions of input feature map, $I^c_{(h,w)}$ is the value of channel c at pixel (h, w) in the input feature map, and x_i^s and y_i^s are the coordinate values at pixel i in the generated sampling grid G_s with size of $(H_O \times W_O)$. The details on the proof that this sampler is differentiable can be found in [Jad+15]. This differentiable sampler allows gradient calculation during back-propagation, which supports an end-to-end training without requisite of ground-truth offset field to train the localization network.

This DSTN design was inspired by the offset-field idea from DCNN. By design, their structure restricts the deformation to only convolutional layers and ROI pooling layers. Additionally, they lack the reuse of processed data (i.e., sampled data), which could reduce redundant re-processing in the optimization of an embedded application. By producing only a pure sampling mechanism without forcing any additional processing layers, the sampled data can be reused multiple times directly for the following networks or data processing structures after DSTN. In addition, the offset-field production in DCNN is limited to one single convolutional layer only or a single fully connected layer. In this design, DSTN can perform more complex offset-field production under different requirements (e.g., small-patch translation or global-determined offsets).

A DCNN block can be reproduced by a DSTN layer when it has only one convolutional layer in the localization network, followed by a single convolutional layer. In this special case, DSTN is mathematically equal to DCNN. Still, the DSTN can use different localization networks for different purposes. For example, adding a pooling layer in the localization network can make the DSTN perform dynamic pooling, which cannot be done in DCNN.

3.2.4 DSTN Functionality

The general objective of DSTN is to increase the network's capability. Given an input with small differences in the same (sub-)class (e.g., same sign from different countries), DSTNs can produce similar output matrices for these inputs without increasing distinctly computational complexity in terms of network depth and channel number. Fig. 3.5 shows a functionality example of the internal components processing an input. In this figure, blue arrows show the feed-forward data flow

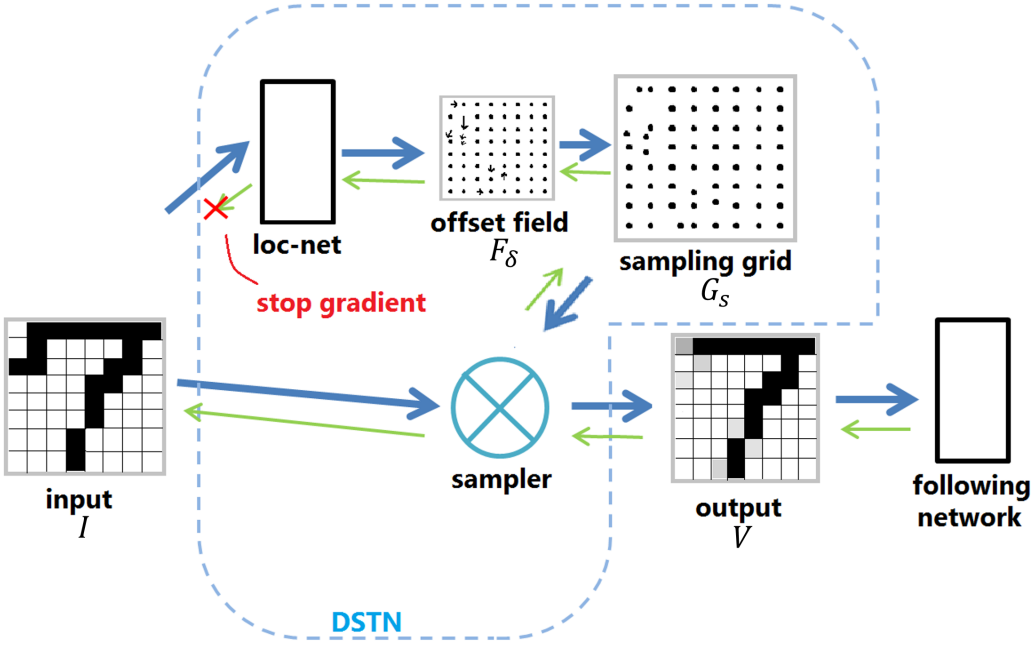

Fig. 3.5.: The structure of DSTN with a concrete example. Notations are consistent to Fig. 3.4.

and green arrows show the back-propagation gradient flow. By sampling from the input with respect to the produced offsets (illustrated as arrows) based on this input, the following network is now seeing very similar inputs for the same (sub-)class. Fig. 3.6 gives another example of this effect, where the DSTN produces similar image outputs for the "7"s in different fonts. The output column shows visually similar images compared to the visually different images in the input column. It is possible to decrease such differences by means of a sampling method.

3.2.5 Training Strategies

DSTN has the ability to produce a sampled feature matrix with regard to the input feature matrix without changing its content by any means of convolution. It can be easily integrated into any existing CNN structure. Also, as the sampling method is differentiable [Jad+15], DSTNs can be trained end-to-end, which requires no further changes to present training procedures and only small changes in the whole structure of a network. This convenience is also very important, as it requires little modification for an existing optimized CNN on an embedded system. To make the training procedure faster and more stable, two beneficial training strategies are described below.

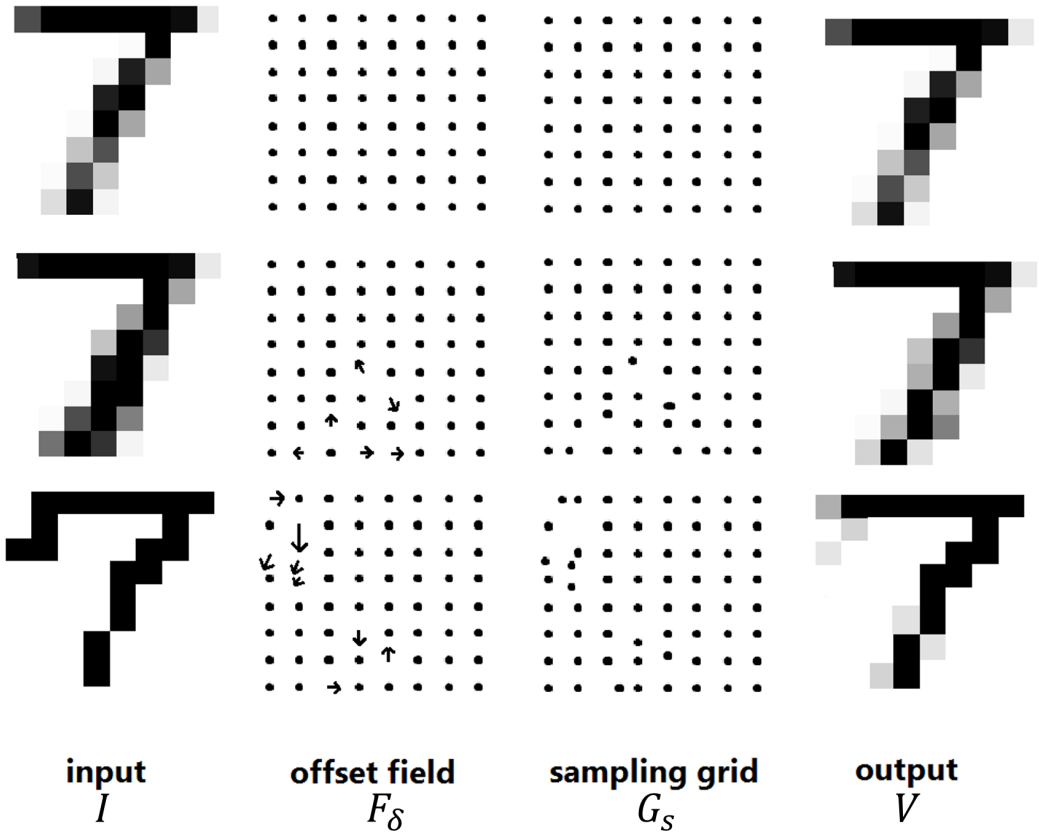

Fig. 3.6.: An example of how DSTN deals with small difference of images for the same class. Row: three inputs of "7" with different fonts; Column: the visible components in Fig. 3.5, as input I, offset field F_δ, sampling grid G_s, and output V, respectively.

3.2.5.1. Stop Gradient

The gradient is calculated, and the error information is passed in the back-propagation phase [RHW86]. In general, the error with regard to the objective of a network is represented in this information. In DSTN, a localization network should be trained to produce sampling grid offsets, while the other layers prior or posterior to the respective DSTN layer in the network are intended to perform other tasks (e.g., image classification). Because the DSTN is designed to be trained in an end-to-end manner, all prior layers to the DSTN layer can receive updates due to errors in sampling offsets. If one wants to train an embedded network with limited capacity, passing such an error from the localization task may increase the complexity of the training. The network might resolve using the error gradients from the localization network for training other tasks instead of localization only.

Therefore, an extra component is added, Stop Gradient, prior to the localization network inside the DSTN layer. It blocks the gradients passing from the localization

network back to the preceding layers. In Fig. 3.5, the green arrows represent the gradient passing during back-propagation, and the red cross shows where the gradient-passing is blocked. Although this blocking of gradients is not mandatory, it is still suggested to be applied to protect the layers prior to the DSTN from the errors of the localization network. This is to keep the main task of the whole network apart from the subordinate localization task. In experiments with slim networks suitable for embedded hardware, the advantages of applying Stop Gradient were promising.

3.2.5.2. Post-activation

In fact, the dense translation produces a very dynamic change in the feature matrix. In end-to-end training, the parameters of the localization network tend to get out-of-control, leading to a non-convergence. Early in the training, while the rest of the network is unsettled, estimated displacements may be far too large, or even outside the input map. To reduce such an effect, a post-activation of DSTN is suggested inside a neural network (i.e., learning rate of DSTN is set to zero in early epochs). In this way, the rest of the network can converge easier on the main task (e.g., image classification) before the activation of DSTN. After that, activating DSTN will increase the capabilities of the network to easily deal with the small variances within one (sub-)class.

With post-activation in place, the DSTN becomes more stable in end-to-end training. This distinctly reduces the chance of non-convergence. Also, since the main task of the whole network is trained a few times beforehand, the network is expected to have a general view of the task.

3.2.6 Analysis of computational complexity

DSTN is a light-weighted network unit that consumes only limited computational resources. The main consumption is by the localization network, which is at least a convolutional layer with a two-channel output, for horizontal and vertical offsets. The sampler produces minor resource consumption. For example, to perform a bi-linear interpolation based on the offset, a 2×2 weighted averaging kernel is used. Considering that it already has horizontal and vertical offsets, the proportions of four sampled pixels can be easily calculated for each sampling location. Summarized, the DSTN in general only consumes the computational time for the localization net plus

a 2×2 weighted kernel-averaging, which is less than adding one additional input channel.

If a two-channel-output convolutional layer is applied as the localization network, the total computation time of DSTN (including sampling, etc.) should be less than applying a three-channel-output convolutional layer on the same input. Compared to a naive method of increasing the capacity of a network to deal with those small intra-class differences, the additional consumption by DSTN is neglectable. More details and analyses are presented in the next section of the evaluations.

3.3 Experiment Results

The evaluation concentrates on image classification tasks. Image classification is a common ML task in ADAS. Since DSTN is focused on increasing the capability of neural networks on embedded systems, all image classification tasks are performed by small convolutional neural networks that are capable of running in real time on embedded hardware. For example, consider a customary embedded ADAS system composed of two digital signal processor (DSP) cores, each having 750MHz, the system has 96KB of L2-cache and 32MB of random-access memory (RAM). Computational power of such hardware is significantly smaller than a workstation PC with multiple high-end graphic cards, and tens of Gigabytes of RAM. Considering only the upper bound of resources in such hardware, it could store up to 24K single-precision floats in the L2 cache or 8M single-precision floats in the RAM. When running in 30fps, it can have up to 50M cycles in total on each frame theoretically when neglecting overhead. As such an embedded hardware is used by the whole ADAS, any single task (e.g., traffic sign recognition) can only take a small portion of the computational resources. In the following section, such hardware is used for runtime analyses.

3.3.1 Street View House Numbers Database

To evaluate a theoretical expectation that DSTNs increase the performance of a network when small differences are found within (sub-)classes, the SVHN public dataset is used (see Sec. A.1.1). In this dataset, the task is to recognize digits 0–9 from real images cropped from Google Street View. SVHN contains significantly more variation in the input images than in the modified national institute of standards and technology (MNIST) dataset. MNIST has only hand-written digits with a very clean

background (i.e., pure black). All numbers in MNIST are in white, and the images are in grayscale. SVHN provides a real-world scenario where numbers appear in different fonts and colors. They may also contain texture in the foreground and other objects in the background. Furthermore, in image patches, multiple digits may appear, where the classification should only concentrate on the central digit.

The SVHN dataset contains 604,388 digits for training and 26,032 for testing. All experiment networks are based on the same structure as a small classification network. The network has three 3×3 convolutional layers with output channels of 16, 32, and 64, respectively. It also has two 2×2 max-pooling layers in-between, followed by two fully connected layers with 128 and 10 neurons, respectively. To evaluate the performance of a model benefited by DSTN, the following three networks are compared:

1. a network adding one DSTN unit using a bi-linear sampler after the first max-pooling layer;

2. a network with comparable computational demand. It changes the output channel size of the first convolutional layer to 19 to match the additional effort of DSTN; and

3. a baseline network without any modifications.

The experiments further test some variances of the DSTN unit, adding Stop Gradient SG and post-activation PA. In addition, it examines how a complex localization network can produce better grids, which in turn leads to better performance. The second max-pooling layer is replaced in the baseline network with a more complex localization network in the DSTN layer (noted as "DSTN-CL"). This complex localization network consists of a max-pooling layer and two convolutional layers in order to perform dynamic pooling.

The classification error rate of all networks can be found in Tab. 3.1. The error rate shows the network performance on the test set. "Baseline" is the basic network structure, and "Same-Consume" is the network with comparable computational complexity to DSTN. The DSTN networks use a DSTN unit, with its variances. The "Improvement" column denotes the decrease of error rate compared to baseline.

According to Tab. 3.1, the DSTN network outperforms the complexity-comparable baseline. The improvement from the baseline to the complexity-comparable baseline is minor because the network cannot benefit much from such a slight increase in computational resources. However, using the DSTN can even improve more. Although the prediction error on the test set only decreases to 7%, considering the

Tab. 3.1.: Classification error rates of different networks on SVHN and the improvement to the baseline.

Network	Error	Improvement
Baseline	4.59%	0%
Same-Consume	4.48%	2.4%
DSTN	4.40%	4.1%
DSTN+SG	4.34%	5.4%
DSTN+PA	4.47%	2.6%
DSTN+SG+PA	**4.25%**	**7.4%**
DSTN-CL	**3.95%**	**13.9%**

low computation consumption, it is still a reasonable improvement. Especially on an embedded system, using a complex network structure directly (as used in alternative approaches) is not practical, though the performance would improve. The DSTNs are very useful for improving network performance on limited computational resources. Additionally, the "DSTN-CL" network, with a complex localization network for dynamic pooling purposes, can reach even better performance.

To show how hardware limitations can affect performance, three networks are chosen from a result list[1] to estimate computational complexity. These networks are ranked in the top, middle, and bottom of the scoring list. They also have a low number of customized layers to allow for a comprehensible and fair estimation of their computational complexity. Following the description of these papers proposing the three networks, the theoretical number of operations and the number of parameters are calculated. Because some of the networks use customized layers, the calculation mostly ignores these layers, so the real numbers can be even larger. The chosen networks are noted as "Sermanet" [Ser+12], "Goodfellow" [Goo+13], and "Liao" [LC15]. From Tab. 3.1, the best-performing "DSTN+SG+PA" is used for the comparison, noted as "Ours". However, the computational complexity stays the same in different DSTN training settings.

Tab. 3.2 shows a comparison of the number of parameters ("# params"), number of operations ("# ops") and the error rate reported in the respective papers. The number of operations is counted on the network processing one input image. The "Hardware" shows the theoretical hardware limitation according to the description of the hardware in the beginning of this section. The table shows the significant complexity advantage of the presented approach compared to other networks, which are hardly applicable to low cost embedded hardware. Of course, the runtime gain

[1] https://tinyurl.com/yc8jydfv

comes at the cost of some performance loss compared to the costly top-performing approaches. However, compared to approaches that require a factor of 15 to 350 more operations, the performance loss of the proposed network stays in a range of 2–2.5%, while it even outperforms an approach that requires more than 13 times more operations.

Benefiting from the additional complexity of the localization network, "DSTN-CL" reaches better performance than the other DSTN networks. Of course, this comes with a slightly larger number of parameters and operations. The decision of which structure to use for a potential application will eventually be based on the actual available resources on the utilized embedded hardware.

Tab. 3.2.: Computational complexity of different networks on SVHN with the proportion to the hardware baseline and the classification error rate on the test set.

Network	**# params**		**# ops**		**Error**
Baseline	34.2K	0.4%	5.8M	11.7%	4.59%
Same-Consume	36.4K	0.5%	6.3M	12.6%	4.48%
Ours	34.6K	0.4%	6.1M	12.2%	**4.25%**
DSTN-CL	36.7K	0.5%	6.2M	12.4%	**3.95%**
Hardware	8M	100%	50M	100%	N/A
Sermanet [Ser+12]	0.7M	8.75%	83.6M	167.2%	4.9%
Goodfellow [Goo+13]	11M	137.5%	96.2M	192.4%	2.47%
Liao [LC15]	4.4M	55%	2.2B	4400%	**1.76%**

3.3.2 German Traffic Sign Recognition Benchmark Database

In ADAS, TSR is one of the basic tasks that plays an important role in assisting the driver. German traffic sign recognition benchmark (GTSRB) is one of the public datasets addressing this task (see Sec. A.1.1). This dataset contains 51,839 real-world images of German traffic signs (39,209 for training and 12,630 for testing), having 43 classes in total. All the images are in color. The images were manually cropped and centered to the sign with a small background. In general, this dataset mostly fits the real ADAS requirement.

Nevertheless, the dataset does not perfectly match the objective, as it contains only German traffic signs. In most cases in this dataset, the signs from the same class look exactly the same. Still, some other differences due to, for example, environment change, are expected to be reduced by the DSTN as well. To train the network on this dataset, all images are resized to 48×48 pixels. The baseline network has three 3×3

convolutional layers with 16, 32, and 64 output channels. It uses a 2×2 max-pooling layer after each convolutional layer, followed by two fully connected layers of 128 and 43 neurons. The DSTN unit with a bi-linear sampler is inserted between the first max-pooling layer and the second convolutional layer. The "Same-Consume" network raises the output channel size of the first convolutional layer to 19. Also, a more complex localization network is utilized by adding one more convolutional layer to the DSTN. This is noted as "DSTN-CL".

The performance of different networks is shown in Tab. 3.3. The error rate shows the network performance on the test set. The networks are denoted by the same abbreviations as in Sec. 3.3.1. From this table, the DSTN outperforms the baseline network, with a 20% reduction in testing errors. Also, the complexity-comparable baseline network shows only a minor improvement in performance due to the limited addition to the network capacity. The "Stop Gradient" component proved to be very valuable on this dataset. The performance is even decreased compared to baseline when Stop Gradient is switched off. Also, "DSTN-CL" with a more complex localization network produces even better results, though the computational cost is higher as well.

Tab. 3.3.: Classification error rates of different networks on the German traffic sign recognition benchmark (GTSRB) and the improvement to the baseline.

Network	Error	Improvement
Baseline	1.78%	0%
Same-Consume	1.74%	2.2%
DSTN	1.98%	-11.2%
DSTN+SG	**1.40%**	**21.3%**
DSTN+PA	1.97%	-10.7%
DSTN+SG+PA	1.54%	13.5%
DSTN-CL	**1.18%**	**33.7%**

Similar to the evaluation in Sec. 3.3.1, the proposed network is compared with two other networks from the GTSRB result list. One is better than human performance (1.16% in error), and the other is worse. They both have a limited number of customized layers for a comprehensible and fair estimation of their computational complexity. They are noted as "Sermanet" [SL11] and "Ciresan" [Cir+12]. The other notations remain as presented in Tab. 3.2. The number of operations is counted on the network processing one input image resized to the network setting, as stated in the respective papers. As shown in Tab. 3.3, the best-performing "DSTN+SG" used in the comparison is noted as "Ours." The computational complexity remains among different DSTN training settings. The comparison is shown in Tab. 3.4.

Tab. 3.4.: Computational complexity of different networks on GTSRB with the proportion to the hardware baseline and the classification error rate on the test set.

Network	# params		# ops		Error
Baseline	0.3M	3.8%	13.7M	27.4%	1.78%
Same-Consume	0.3M	3.8%	14.7M	29.4%	1.74%
Ours	0.3M	3.8%	14.1M	28.2%	**1.40%**
DSTN-CL	0.3M	3.8%	15.4M	30.8%	**1.18%**
Hardware	8M	100%	50M	100%	N/A
Sermanet [SL11]	1.7M	21.3%	0.2B	400%	1.69%
Ciresan [Cir+12]	38.6M	482.5%	8.5B	17000%	**0.54%**

From the comparison, the huge gap of required operations between the compared networks and the presented approach stays again. Light-weighted networks with DSTN still outperform one of the two networks. Again, “Ours” is preferred according to the very limited computational resources and still benefits from “DSTN-CL” when more resources are available.

3.3.3 Private Traffic Sign Recognition Database

To evaluate the performance of DSTN on a real ADAS, the DSTN unit was tested on a private traffic sign recognition dataset from Aptiv plc. This dataset contains real image patches acquired by a sign detection algorithm. Detection is not manually performed, so the signs are not always in the center of the patch. These images are taken in multiple countries, where the signs have different designs and appearances. Due to the limited capacity of the networks, such differences may not be easily solved by the network. Using the DSTN is expected to reduce such intra-class differences, thus increasing the performance. The original images are captured on the road with a test vehicle. They were cropped by an industrial object detection algorithm developed by Aptiv. These cropped images were resized to the same size before feeding to the networks. Each cropped image is manually labeled by a team of experts, and these labels are used in supervised training of the networks.

The classification error rate for TSR on this private dataset is shown in Tab. 3.5. The error rate shows the network performance on the test set. The networks are denoted by the same abbreviation as in Sec. 3.3.1. As shown in the table, the DSTN network structure outperforms the baseline networks. In addition, the post-activation variant increases the final performance of the DSTN. Given that this dataset contains traffic signs from multiple countries, the small intra-class differences are much more

obvious. As a result, the error is decreased by approximately 25% compared to the complexity-comparable baseline. One remarkable finding is that the network can hardly converge without Stop Gradient if no fine-tuning of the training hyper-parameters is conducted. All experiments with non-converging trainings are marked with "*" in the table. This was also the reason that Stop Gradient was found useful during the start of this research.

Tab. 3.5.: Classification error rates of different networks on private TSR dataset and improvement to the baseline

Network	Error	Improvement
Baseline	2.33%	0%
Same-Consume	2.11%	9.4%
DSTN	N/A*	N/A*
DSTN+SG	2.26%	3.0%
DSTN+PA	N/A*	N/A*
DSTN+SG+PA	**1.73%**	**25.8%**

The runtime of the network is tested on embedded hardware with the same specifications[2] as stated at the very beginning of this section. The network takes only 0.53 milliseconds to classify a single traffic sign. Considering the low computational capability of the hardware, and the complete DSTN software running on it, this runtime is acceptable.

Fig. 3.7 shows four examples (in columns a–d) of the sampling grid on this TSR dataset. The first row is the input traffic sign image for the whole network, an end of speed limit of 20km/h sign with intra-class variations. The second row shows one channel of the input feature map to the DSTN unit, together with the produced sampling positions as red dots. The third row shows the sampling positions only. The sign in (a) has a very narrow termination line in comparison to others, which is finally extended wider by DSTN via sampling to decrease intra-class variances. For the signs in (b) & (c), the DSTN tried to change the font of the numbers, while the sampling grid varies greatly in the region of the numbers. For the sign in (d), the network may consider it a different sub-class. The DSTN is not distinctly changing it, as the sampling grid is considered pretty much the same as identity sampling.

[2] 2x750MHz DSP cores, sharing a 96KB L2 cache and 32MB RAM.

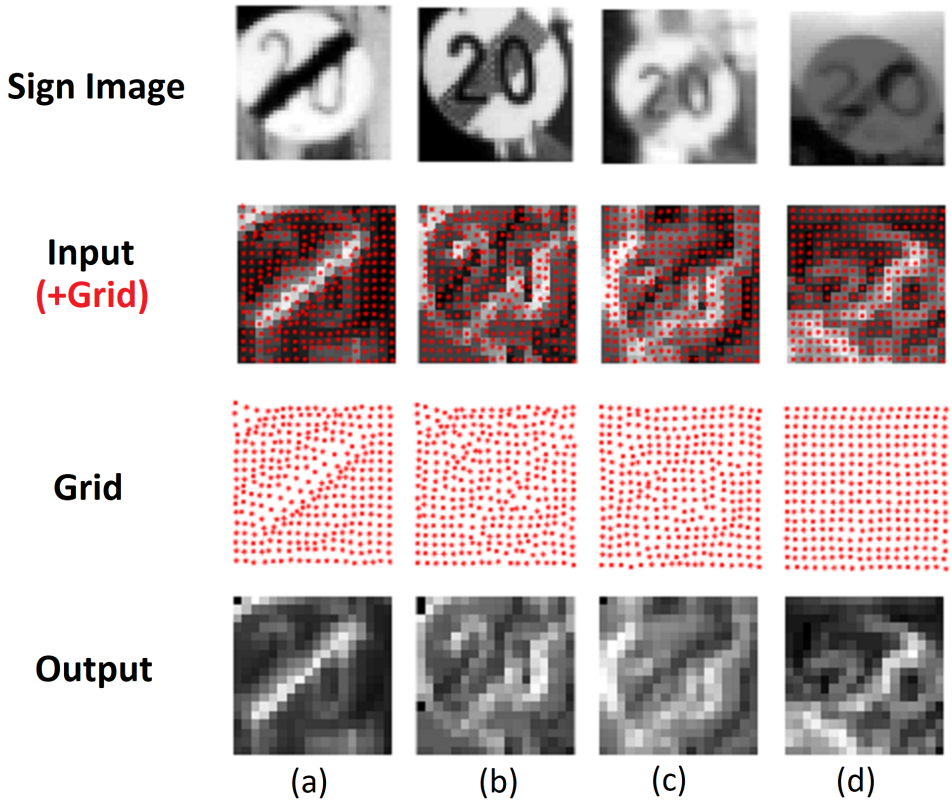

Fig. 3.7.: An example of a DSTN sampling grid on DSTN data, with the end of speed limit of 20km/h signs. First row: input images of the whole network. Second row: input feature maps to DSTN covered with produced sampling positions in red. Third row: produced sampling positions. Fourth row: output of the DSTN layer. The difference may not be significantly visible due to the gray-scale averaging of feature channels and printing compression.

3.4 Summary

In this chapter, a novel convolutional neural network unit that can perform sampling based on pixel-wise translations is introduced. This unit, the DSTN, can reduce small spatial differences in input or in feature matrices of the same class. It increases the capacity of a neural network without consuming much more computational resources. This is explicitly important on an embedded hardware for ADAS software. DSTN can help a small neural network perform better when the network itself has difficulties dealing with small in-class differences. In the evaluation, networks with DSTN always outperform networks without this unit.

The DSTN is designed to be trained in an end-to-end manner. It can be easily integrated into any existing network structure. Especially in ADAS software, when

more training data is introduced, a light-weighted network may lack the capacity to deal with more intra-class variances. Given the fact that the DSTN is simple to integrate, it will not require many modifications to any existing network.

In the following chapters, several further extensions of the DSTN will be presented. The sampling method in DSTN has shown its advantage in modeling complex local variation and inspired the following research and contributions in this dissertation.

4 Radar Data Recording, Annotation, Description and Sensor Alignment

This chapter will present the data used in the research on radar perception tasks. Radar perception is a very important task in ADAS and AD. Radars have great advantages in long-range distance measuring and velocity measuring compared to cameras, ultrasonics or even LiDAR. Compared to cameras and LiDARs, radars are lower in cost; hence, they are preferred in manufactures of vehicles, and even multi-sensor settings can be applied.

In order to perform supervised learning with neural networks, a large amount of data and annotations are necessary and important to achieve reasonable performance. Unlike cameras or LiDARs, radars are mostly very specific to different manufacturers; hence, it is hard to transfer the data or data processing to radars produced by other manufacturers. Moreover, compared to the long history of CV, ML is a very new research topic in the radar field, especially when combined with ANN. Although many public databases are available online with a large amount of data and annotations for various tasks, there are very few in the field of radar.

When starting research on radar perception tasks in this dissertation, there was no public database online for radar perception research. For this reason, one of the research topics and the precondition for the research is data collection.

To collect enough data and annotations for neural network training, there are two major methods: simulation and real data collection. For simulation, it normally uses a set of algorithms to generate the data with a defined environment. It has the benefit of generating unlimited amounts of data with free-given perfect annotations. Unfortunately, the quality of the algorithms used in a simulator also highly affects the quality and capability of the ML algorithm trained with these simulated data.

On the other hand, real data collection is a common method of data collection. Most of the public databases use this method, especially on CV and LiDAR tasks. Other than collecting a large amount of data from real-world recordings, there is also a need for manual annotation. Because one cannot control the environment

in the data recordings (though a controlled environment can be an option), the environment has to be manually annotated based on the collected data. For most CV tasks, the story is short: label what you see in the image. For LiDAR, it can be a little bit complex, as LiDAR produces 3D point clouds; however, labeling is still quite straightforward. As for radar, things are no longer simple.

Unlike cameras and LiDARs, radar data are very hard to label, even for radar experts. The noise and instability of radar data acquisition makes it difficult for human beings to understand the data. A common way is to use a reference sensor to annotate the data; cameras, LiDAR, or even a global positioning system (GPS) can be used. Then, the labels are transferred to the radar domain and supervised learning is performed on these transferred labels.

The following sections first discuss the coordinate systems used when defining and processing data. Then, the simulator and simulated data used in the research are presented. Finally, the dataset composed from from real-world recordings is introduced.

Most of the work introduced in this chapter is based on others' contribution and support, primarily from Aptiv engineer teams. Other than these external supports, Sec. 4.3.3 presents the work by the author, which solves the mismatch issue of multiple sensors in the recorded data. In this chapter, these detailed introductions of the radar data are mainly for completeness and serve as the basis for the following chapter. Apart from the above-mentioned section, the credits for external contributions are also specified in the following sections.

4.1 Coordinate Systems

4.1.0.1. Sensor Coordinate System

When using radar data, it starts from a range-Doppler-angle cube (Sec. 2.3.1.2). The cube contains information of radar detections with attributes of range, Doppler velocity, angle, and magnitude (corresponding to RCS). To project each detection onto a spatial coordinate system, a polar coordinate system is normally used. The range and angle are directly projected into a location in a polar coordinate system, and the Doppler velocity and RCS are understood as additional attributes of each detection. This polar coordinate system is a polar radar sensor coordinate system (SCS).

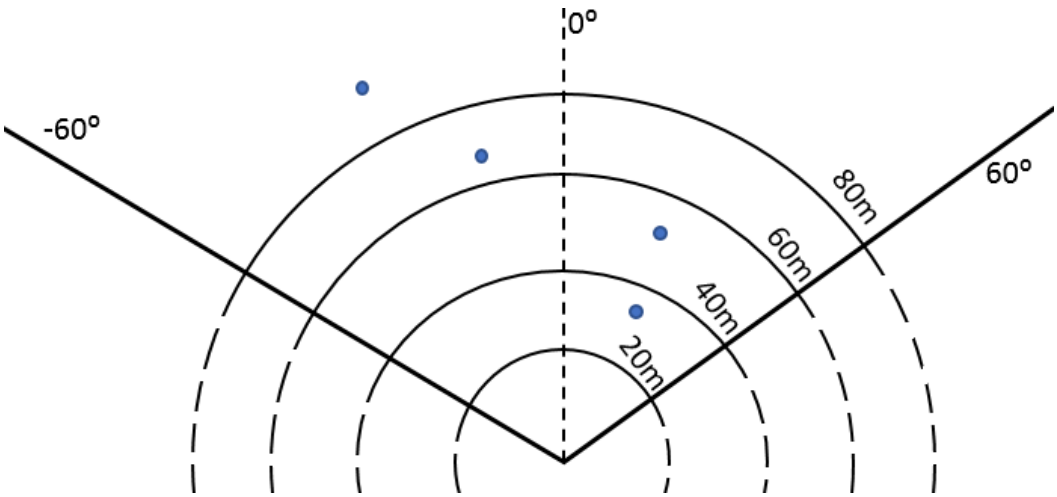

Fig. 4.1.: An example of a polar radar sensor coordinate system.

Fig. 4.1 shows an example of a polar radar SCS with several radar detections (blue dots). The origin of radar SCS is at the center of the radar antennas array. The polarity of radar mounting is how the signal sweeps. Assuming the radar sweeps from left to right, when it is mounted upside-down, the sweep is then from the right to left. In this way, the angle calculated from the signal is negated. This polarity of radar mounting is compensated for when calculating the angle. Based on the range and angle, each radar detection is placed in a polar coordinate. The zero-angle is at the boresight of the radar. In the figure, the zero-angle is shown as a dashed line. Due to the limited FOV of radar, the angles of detections are limited within the FOV, which is shown as solid lines ($\pm 60°$).

Based on the polar radar SCS, the coordinate can easily be converted into a Cartesian coordinate system, which is a Cartesian radar SCS. Fig. 4.2 shows the conversion of these two radar SCSs. The axes of the Cartesian coordinate system are shown in red arrows. The origin is the same as in the polar system, and the x-axis points to the boresight angle, and the y-axis points to the right, accordingly. Here, a right-handed Cartesian coordinate system is used.

The coordinates are converted by

$$\begin{aligned} x_{vcs} &= r cos(\theta) \\ y_{vcs} &= r sin(\theta), \end{aligned} \tag{4.1}$$

where r is the range in polar and θ is the angle in polar.

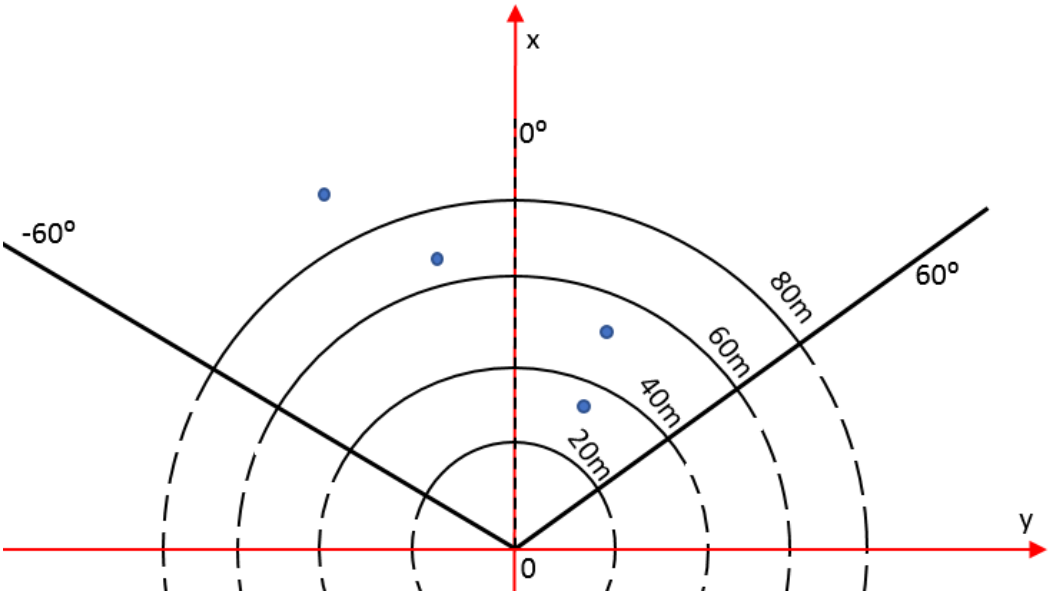

Fig. 4.2.: An example of a polar radar sensor coordinate system with Cartesian radar sensor coordinate system overlaid.

Similarly, for other sensors, the SCS is defined in a similar way. LiDAR, as it is also a laser-sweep manner in measurement, first puts the data in a 3-D spherical system, where each detection has the azimuth angle and elevation (zenith) angle as two angular axes and a distance (radius) axis. Then it converts the coordinate into a 3-D Cartesian system, which defines the x-axis as the zero-azimuth-zero-elevation, y-axis to the right and z-axis to the downward accordingly. To make it simple, a right-handed Cartesian coordinate system is used.

4.1.0.2. Vehicle Coordinate System

When putting all sensors on a vehicle, it needs to define a common coordinate system incorporating all data from different sensors. This is a vehicle coordinate system (VCS). The origin of VCS is defined at the center of the rear axle of the vehicle on the ground, and the x-axis to the forward direction, y-axis to the right, and z-axis to the ground, according to ISO8855 [Sta11]. The VCS is also a right-handed Cartesian coordinate system.

Each data sample from a different sensor is converted from its own Cartesian SCS into this common VCS. For simpleness, the conversion is performed in Cartesian systems, where a common way is by matrix multiplication on given data sample coordinates

$$S_{\text{VCS}} = T S_{\text{SCS}}, \tag{4.2}$$

where S_{SCS} contains the coordinates s_x, s_y, s_z of a data sample in SCS

$$S_{\text{SCS}} = \begin{pmatrix} s_x \\ s_y \\ s_z \\ 1 \end{pmatrix}, \tag{4.3}$$

and transformation matrix T is defined as

$$T = \begin{pmatrix} & & & d_x \\ & R & & d_y \\ & & & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{4.4}$$

where d_x, d_y, d_z are the translation offsets of the respective sensor in VCS and the rotation matrix R is the product of the rotation matrices of each axis

$$R = R_z(\alpha) R_y(\beta) R_x(\gamma) \tag{4.5}$$

with

$$\begin{aligned} R_x(\gamma) &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\gamma & -sin\gamma \\ 0 & sin\gamma & cos\gamma \end{pmatrix} \\ R_y(\beta) &= \begin{pmatrix} cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -sin\beta & 0 & cos\beta \end{pmatrix} \\ R_z(\alpha) &= \begin{pmatrix} cos\alpha & -sin\alpha & 0 \\ sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}, \end{aligned} \tag{4.6}$$

where α is the yaw angle along the z-axis, β is the pitch angle along y-axis, and γ is the roll angle along the x-axis.

The conversion takes into account the mounting of the sensor in VCS. The mounting of radar is defined based on the placement of boresight and taking the origin of radar SCS as the reference point. After conversion, all data samples from each sensor are in a common VCS. In this manner, the data can be spatially referred to each other.

4.1.0.3. World Coordinate System

Based on GPS or an even more accurate differential global positioning system (dGPS), the location of a vehicle can be understood in a global coordinate system because the recordings of real data or even the simulation data are relatively short. The movement of the ego vehicle within the recording time of each sequence is trivial compared to the GPS scale. To reduce complexity and for easier conversion, normally take the VCS of the first sample in each sequence as the definition of the world coordinate system (WCS).

Along with the movement of the ego vehicle, the origin of VCS is moving accordingly. Based on the movement, it is possible to calculate the difference of the current VCS compared to the WCS and convert the data samples in each VCS to the same WCS. In this manner, the data samples can be used and understood within the same spatial coordinate system.

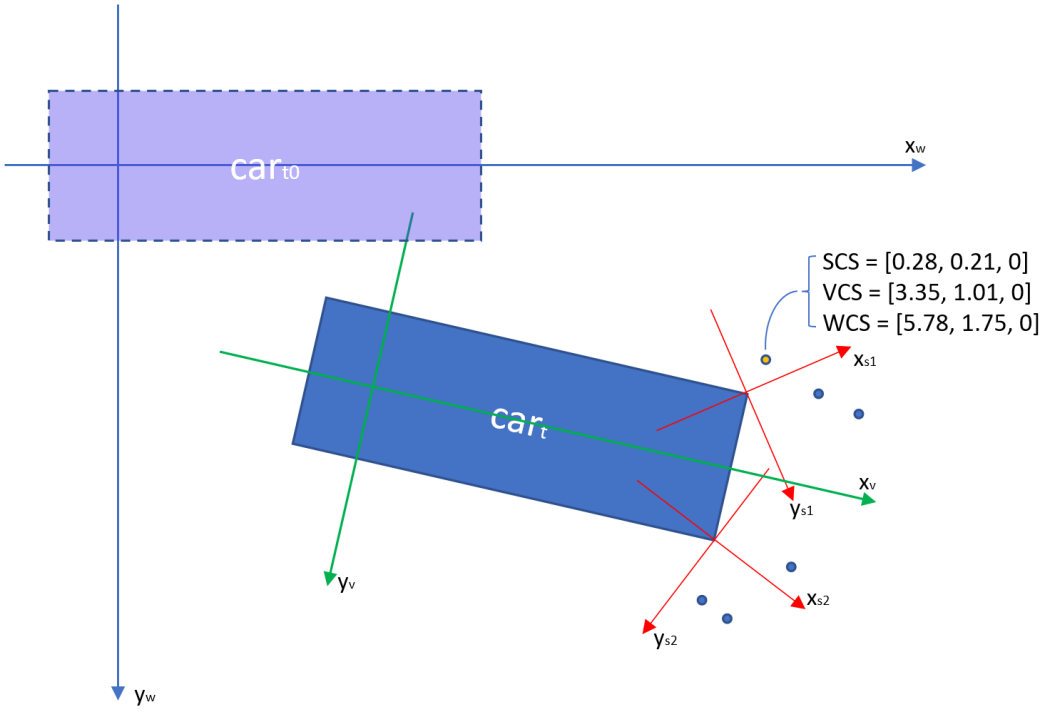

Fig. 4.3.: An example of different coordinate systems applied to data samples.

Fig. 4.3 shows the relations between Cartesian SCS, VCS, and WCS. The red axes are the SCSs of two sensors, the green axes are the VCS of current time t, and the blue axes are the WCS, which is the VCS of the first sample at $t0$. The z-axis of each coordinate system points to the ground; hence, it is not shown in the figure. The coordinates of the orange sample point in these coordinate systems are also shown in the figure (in $[x, y, z]$). With these coordinate systems and conversions, the samples can easily be placed into a common spatial space for the need.

4.2 Radar Data Simulator

In order to get a large amount of annotated data, a simulator is the first choice in most cases. Thanks to the engineering team at Aptiv, they developed a good radar data simulator for data generation purposes. With the help of this simulator, it can easily obtain a lot of radar data with perfect annotation. Moreover, it is even possible to extend the definition of the environment to add more features or noises to the data.

4.2.1 Design and Methods

The simulator is a ray-tracing-based algorithm. The simulated sensor shoots several rays into the environment and receives energy feedback. The feedback is processed in a similar way to conventional radars with FFT. Then the processed data is presented with attributes of range, angle, Doppler velocity, and magnitude.

Fig. 4.4 shows an example frame of a radar simulator. In (e), the ground truth of the environment and the ego vehicle are shown. The ego vehicle is shown as a magenta box at the center of the bottom. Each white box is the target vehicle. The red lines on each vehicle and the ego vehicle indicate the movement vectors. The long white curves indicate guardrails or walls.

The radar sensor is mounted at the front center of the ego vehicle. (a) shows how the rays are cast and the reflections of rays. The responses of each ray are finally processed by FFT and producing a range-angle map as (b). Moreover, the magnitude response and Doppler velocity response on each reflected ray are also presented in (c) and (d).

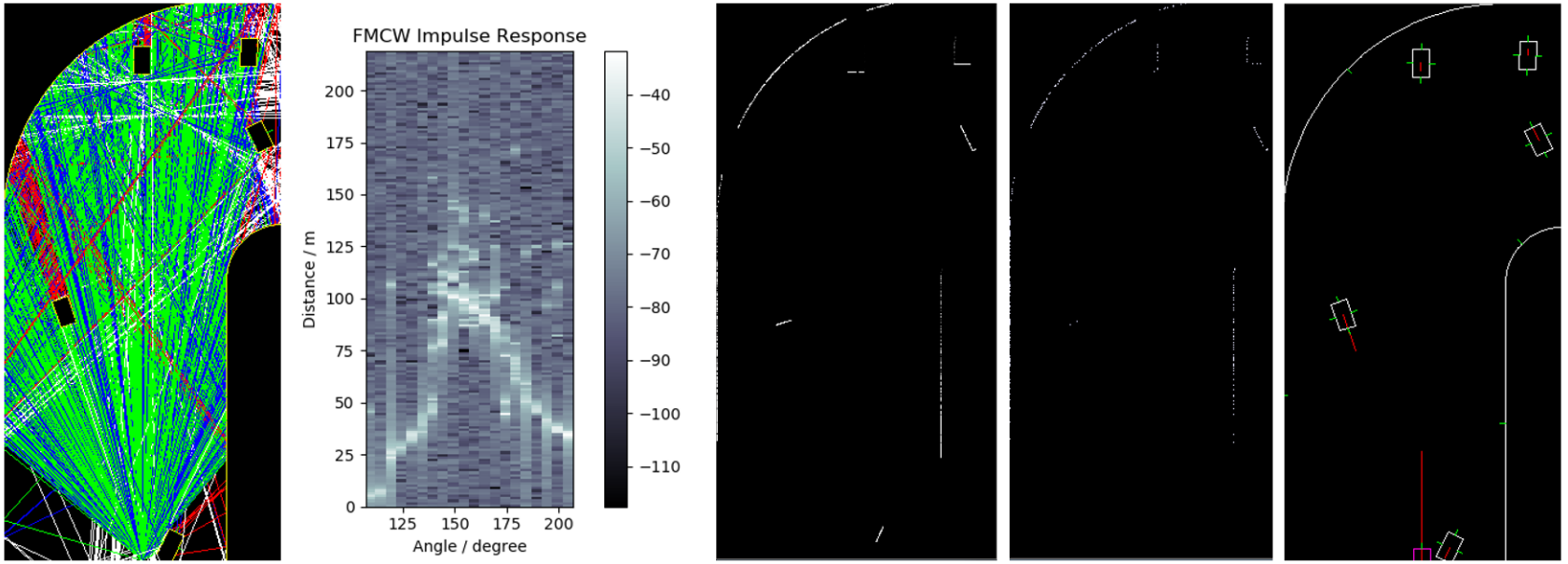

Fig. 4.4.: An example of radar simulator. (a) shows the rays cast by the sensor; (b) is the range-angle map after FFT processing; (c) is the reflection response from rays; (d) is the Doppler velocity response; (e) is the ground-truth.

4.2.2 Data Description

After running the simulator, several data are provided. The range-angle map is provided as image-like data. The velocity and magnitude responses are provided in a point cloud manner, where the position of each point is in Cartesian radar SCS and the magnitude and velocity are the attributes of each point. The ground truth is provided as bounding boxes with the center position, the orientation, the velocity, and also the class labels. The guardrails are not considered as boxes; hence, they are only used in environment generation as noise producers.

This dataset has several advantages. First, the data is still very clean compared to real-world scenarios. Although including a guardrail-like environment to produce a multi-path reflection effect, the reflections are still under control. In the real world, the reflection of the energy can come from various sources; reflections from the uneven ground surface or even the bumper of the ego car in front of the radars can contribute to noise in the energy.

Second, the data had perfect annotations. Labeling radar data is very difficult. Visible in Fig. 4.4, the data in (b), (c), and (d) are not easily labeled by human beings, even though they are clean enough. The importance of having correct labels for supervised learning makes it require even higher quality data annotations. In this view, simulated data is perfect.

Third, the data is under complex processing, similar to conventional radars; hence, the ANN and other supervised learning methods cannot easily overfit to the processing model. This is normally one of the problems of simulated data, that the high capability ML model can easily learn to fit to the data model of simulation.

However, there are also disadvantages of this simulated data. It does not contain any real-world noise sources. The noise patterns from various real-world noise producers cannot be easily simulated. Although signal loss and environmental noise producers (e.g., guardrails) are added, it is still not as comparable to a common real-world scenario. Also, the magnitude is calculated only by a simple RCS formula, where the various materials or complex shapes of reflectors are not considered. This may make the magnitude much cleaner than real-world recordings.

Nevertheless, with the use of simulation data, several studies on radar perception tasks have been conducted. During that period, real-world data collection and annotation were started in parallel. Although the simulated data has quite some issues with the quality and the limited resemblance to real-world data, findings from these studies are still valuable and help to move forward.

4.3 Real-World Dataset

Due to several limitations of simulated data, it is still necessary to make real-world data recordings and use these data for research and development on a productive level. The engineering teams at Aptiv supported the setup of a vehicle equipped with several sensors for radar ML tasks.

The data are recorded from a real-world environment without any control. It simply runs over daily roads and records the environment using equipped sensors. The data is then manually annotated by experts and used in supervised neural network training.

This section will briefly introduce the recording vehicle, data description, and several challenges faced during the data processing.

4.3.1 Sensor Setup and Recording

The recording vehicle was a Volvo XC90 (see Fig. 4.5). It was equipped with six experimental short-range radars produced by Aptiv. The radars were mounted at the four corners of the vehicle and two under the B-pillars. This gives an overall FOV of $360°$, which will cover most of the surrounding areas of the ego vehicle. A high-resolution LiDAR+Camera system from Hesai was mounted on the roof of the vehicle. It will provide a good reference of the environment, as camera images and LiDAR point cloud are much easier for human beings to understand than pure radar detections. Moreover, the vehicle also had a high-quality inertial measurement unit (IMU) for ego-motion measuring and a dGPS system for high precision position measurement. All of these sensors were calibrated by their mounting positions. The calibration was performed by an engineer team in Aptiv.

The recordings are mostly done near the Aptiv site and in downtown Wuppertal. It covers various scenarios, including urban roads, suburban roads, autobahns, and even tunnels and bridges. Most daily scenarios are covered by these recordings. After recording, the scenarios were cleaned up, and approximately one hour of logs were selected for manual annotation. The annotated logs were then split into training set and evaluation set, where the evaluation set was manually selected based on the coverage of different scenarios.

Fig. 4.5.: A picture of recording vehicle Volvo XC90.

4.3.2 Labeling

The manual annotation was performed by experts from an internal team of Aptiv together with external teams from other companies. The annotation is done on the LiDAR point cloud, with the camera images as references.

The most valuable road user objects were labeled as bounding boxes. They cover more than 30 classes, with sub-classes of vehicles, pedestrians, and several stationary objects on the road. The vehicles include small cars, vans, and even large trucks, or bendy buses. The pedestrians included adults, children, scooters, and strollers. The annotated objects also have coverage of two-wheeler vehicles, e.g., bike and motorcycles. To further analyze the environmental effects of radars, even traffic barriers, trash cans, poles, and bridge pillars are labeled. These objects may provide reflections of radar; hence, they can affect the objects nearby or objects occluded by them.

Fig. 4.6 shows an example of a labeled frame by the internal labeling team of Aptiv. The scene is labeled based on the LiDAR point cloud, with cameras as reference to identify object types, especially the sub-classes. The labels are in 3-D bounding boxes with attributes of its class and also with other attributes, such as movement status. With these additional attributes, it is not only possible to understand the object but also to see if the object is in a moving or standing status. Such movement status is very important for radar perception, as the Doppler velocity will be highly affected by such status.

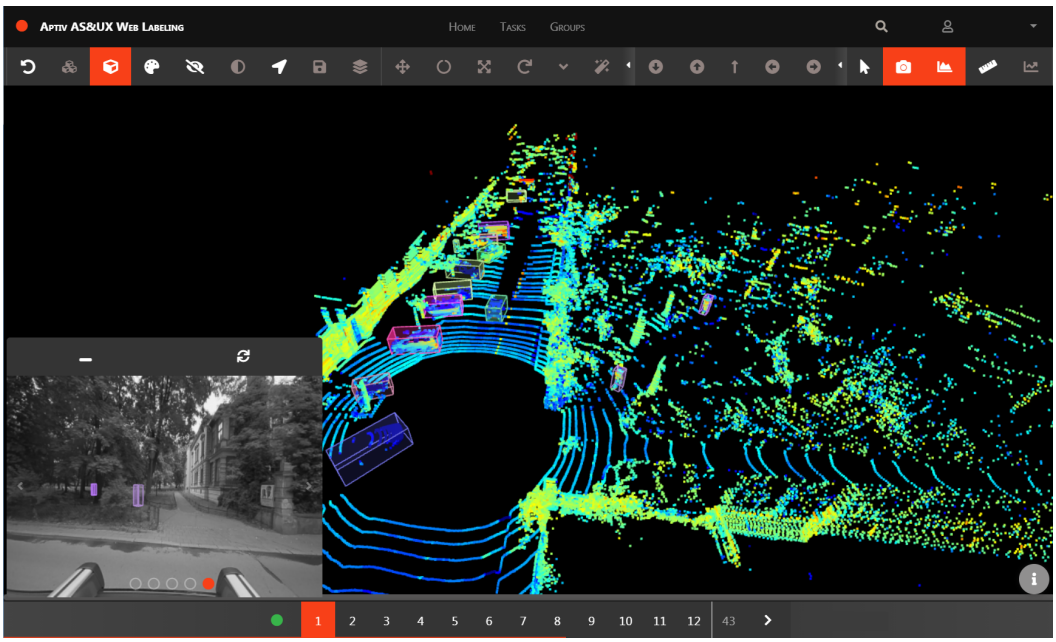

Fig. 4.6.: An example of a labeling tool used by the internal labeling team in Aptiv.

4.3.3 Sensor Alignment

Due to the differences in data measurement of different sensors, challenges occur in collecting the sensor data into a data frame. The detections from different radars have to be aligned and collected into a data package, and they also need to align with the annotation reference sensor (i.e., LiDAR).

Only in such a way can one perform multi-radar perception tasks with supervised learning. This is much more complex than with single-sensor annotated data, where the data measurement and annotation are on the same data source. In this section, the design of the sensor alignment is introduced.

4.3.3.1. Radar Alignment

In total, six radars were used, each making a measurement at a different timestamp. The detections were collected from each radar and performed a packaged data frame. Each data frame shall also be tagged with a common timestamp.

The main problem comes from the random starting time of each radar. Although each radar triggers its own measurement on a regular basis (20Hz), but the initial measurements are not synchronized between radars. To collect the radar measurements and minimize the time differences between the measurements with respect to

the real-world scenario, the system first defines a master radar and takes it as the master timestamp.

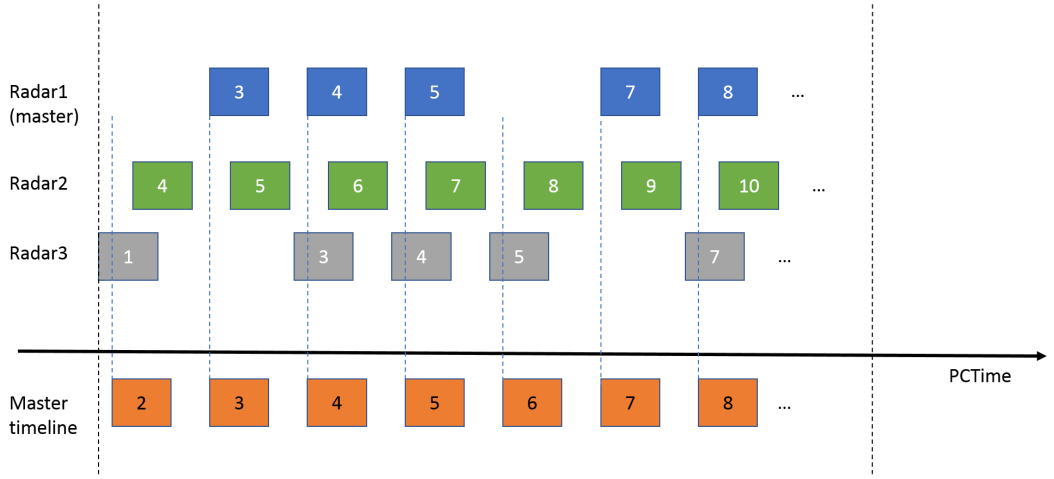

Fig. 4.7.: Master timestamp definition and illustration of measurements from different radars.

Fig. 4.7 shows the illustration of three radars and their own measurements together with their own frame indices. Taking radar 1 as the master radar, the master timestamps for each frame are defined by its measurements. For frames without measurement from this master radar (due to data loss or corruption), the timestamps are inter-/extrapolated. The blue dashed lines indicate the master timestamps. The blue boxes are the data frames from radar 1, where the numbers are frame indices. The same applies to green boxes for radar 2 and gray boxes for radar 3. The orange boxes are master data frames, and the numbers are master frame indices. The axis ("PCtime") indicates the clock used for the timestamps.

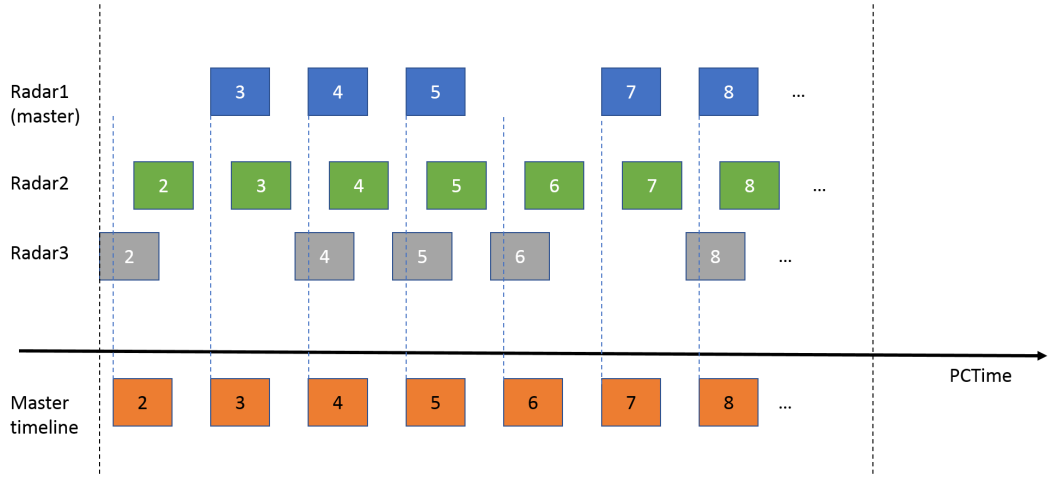

Fig. 4.8.: Realignment of radar frames to the master frame.

After defining the master frames, the measurements from each radar are then aligned to the master, where the frame indices are reset based on the master frame indices. Fig. 4.8 shows how the realignment is performed. The radar measurements within a master frame are assigned to this frame. The assignment is done in the nearest manner, where the start timestamp of each radar frame is assigned to a master frame where the start timestamp of this master frame is the nearest to the assigned one. The assignment is one-to-one, where each master frame can accept only one data frame from one radar. If multiple data frames exist that can be assigned to the same master frame, the nearest will be taken and the others are discarded.

After the above alignment, all radar data frames are within the same frame definition and are used to describe the same scenario. If there are objects detected by multiple radars, the data from the same frame can be used to increase the quality of detection of this object.

4.3.3.2. Radar-LiDAR Alignment

When aligning the radar detections with manual annotations, another alignment issue arises: the annotation reference sensor needs to be aligned with the radar detections. As annotations are performed in LiDAR, the LiDAR and radars have to be aligned.

There are two ways to achieve such alignment. Since the same clock for all sensors is used, the LiDAR annotations are interpolated to each master radar frame with respect to the master radar frame timestamp. Interpolation requires the objects from different frames to be labeled under the same unique ID. When initiating the annotation tasks for the experts, such features were requested and included in the labels. Although there still exists an issue with objects visible only once in the whole recordings or that are occluded for a very long time, most of the objects can be interpolated based on their locations in WCS.

Another way of alignment is to define data frames only by those with labels. After radar alignment, the LiDAR frames are assigned to the master radar frames. Fig. 4.9 shows such a situation. Other than the aligned radar data frames, the cyan boxes indicate the LiDAR data frames. Due to the difference in triggering of data measurement, LiDAR has a lower measurement frequency. Due to time and cost reasons, the annotation has an even lower frequency. The number in each cyan box indicates the annotated LiDAR frame index.

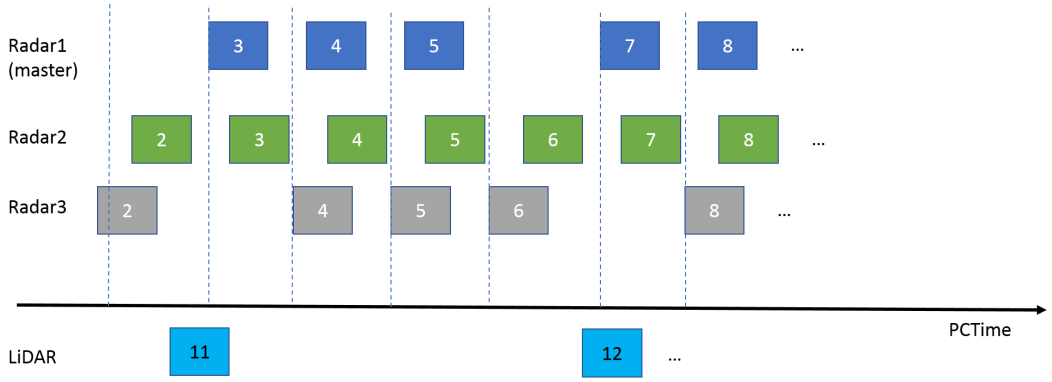

Fig. 4.9.: An illustration of master radar frames and LiDAR frames.

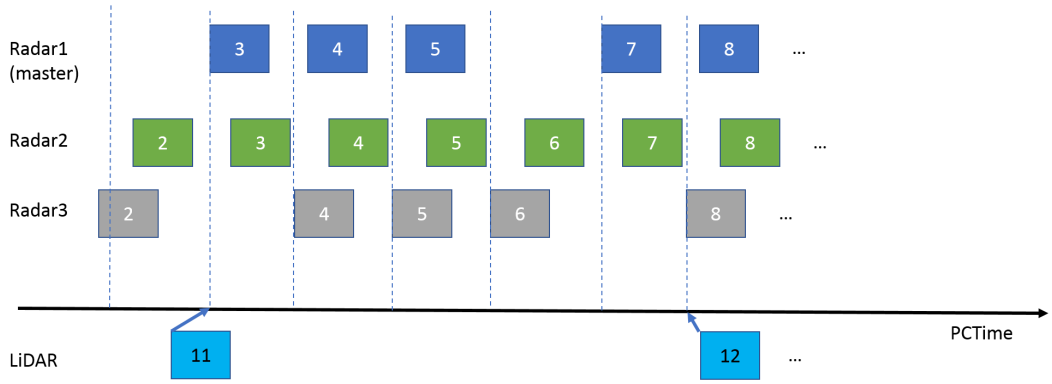

Fig. 4.10.: The assignment of annotated LiDAR frames to master radar frames.

The alignment between labeled LiDAR frames and master radar frames is based on the nearest manner. Fig. 4.10 shows the assignment between the annotated LiDAR frames and the master radar frames. The assignment takes the start timestamp of each annotated LiDAR frame and assigns it to the nearest master radar frame timestamp.

After the assignment, only the data frame with annotated LiDAR data was used for the network training. Fig. 4.11 shows the cleaned-up data frames after assignment. In each of these frames, the measurements from different sensors are considered to represent the same scenario.

4.3.4 Ego-Motion Compensation for Radar

After data assignment, all data were transformed into VCS so that they were in the same spatial space. To make better use of the Doppler velocity in each radar

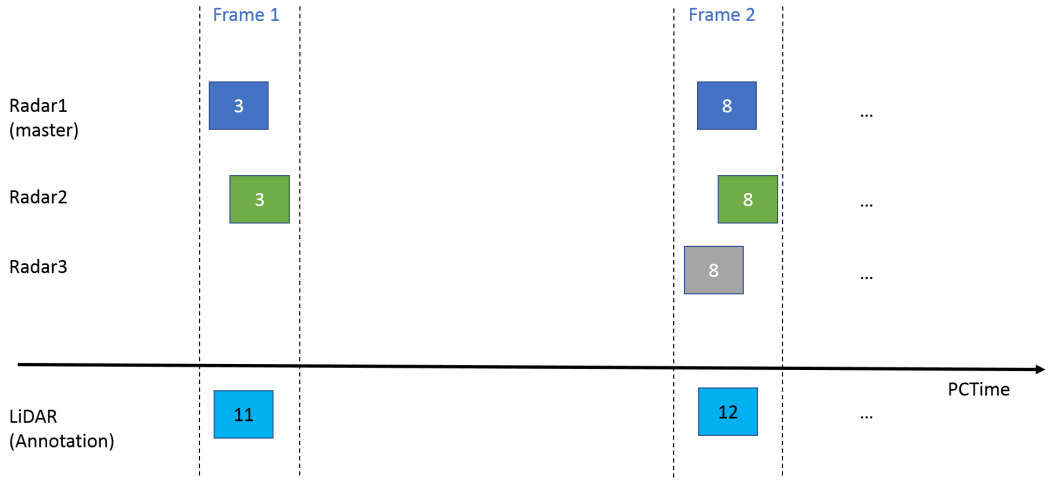

Fig. 4.11.: The cleaned-up data frames with annotation.

detection, the relative Doppler velocity was further compensated with ego-motion to obtain the ground velocity in VCS for each detection.

The compensation was done by first calculating the ground velocity at the sensor mounting position under

$$\begin{aligned} v_{lon} &= v_{ego} cos(\theta_{slip}) - \gamma y_{sen} \\ v_{lat} &= v_{ego} sin(\theta_{slip}) + \gamma x_{sen}, \end{aligned} \tag{4.7}$$

where v_{ego} is the ego speed, θ_{slip} is the ego side-slip angle, y_{sen} and x_{sen} are the mounting position. After calculating the sensor velocity, the compensation of the Doppler velocity is

$$v_{comp} = v_{Dop} + v_{lon} cos(\theta_{det} + \theta_{sen}) + v_{lat} sin(\theta_{det} + \theta_{sen}), \tag{4.8}$$

where the v_{Dop} is the measured Doppler velocity, θ_{det} is the detection angle, and θ_{sen} is the sensor mounting angle.

The compensated detection velocity can largely help the algorithms relate different detections to the same object. This will also help to relate detections from multiple sensors, as the compensated velocity can reduce the relation to the detection angle and mounting angle of the objects.

The only issue is that the measured Doppler velocity is only in the perpendicular direction to the sensors, which does not contain any information about the velocity

in the tangential direction. The algorithm needs to solve the loss of such information by using either detections from multiple sensors, or from multiple timestamps.

4.3.5 Data Description

This dataset includes data from six radars, one LiDAR, and five cameras, as well as manual annotations on the LiDAR domain. Moreover, it has meta information of ego-motion from IMU and ego-position from dGPS.

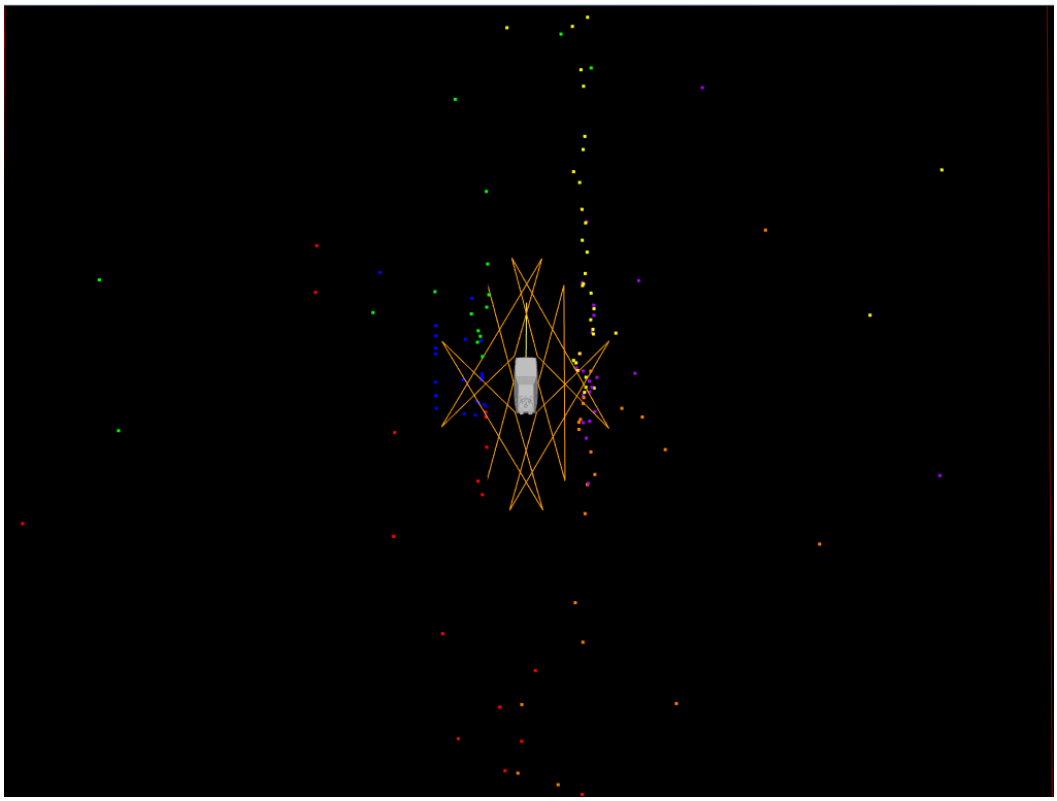

Fig. 4.12.: An example data frame of radar detections, color-coded by radar sensors.

In radar data, it has radar detections containing position in VCS, with the Doppler velocity, compensated velocity, and magnitude in RCS. Fig. 4.12 shows the detections for all six radars in the surrounding region. The different colors indicate different radars, where green for the front-left radar, yellow for the front-right radar, blue for the left radar, magenta for the right radar, red for the rear-left radar, and orange for the rear-right radar. The orange triangles present the FOVs of each radar.

LiDAR data includes LiDAR detections containing positions in VCS with the reflection intensity of each detection. Fig. 4.13 shows the LiDAR detections in the surrounding region. The color indicates the intensity of the reflections.

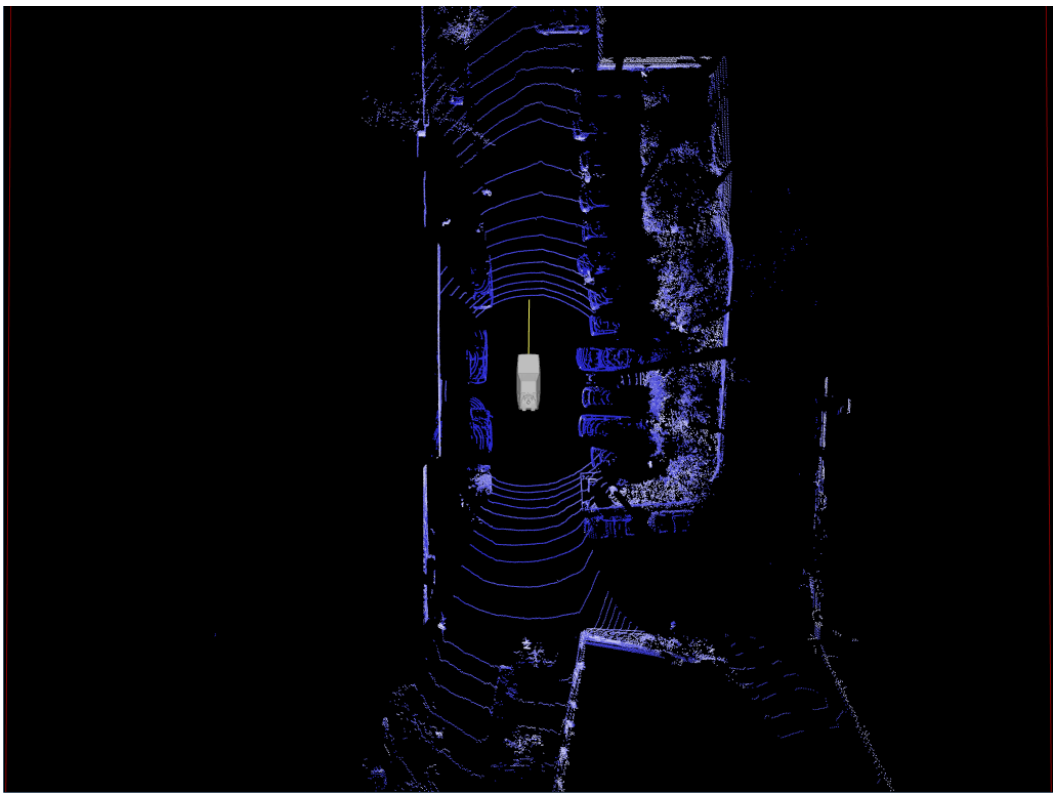

Fig. 4.13.: An example data frame of LiDAR detections, color-coded by reflection intensity.

The images from the cameras are shown in Fig. 4.14. Annotated bounding boxes were also drawn on the camera images. The boxes are projected from VCS into a camera view. The camera data contains four gray-scale fish-eye cameras for the front, left, right, and rear views, and one colored telephoto camera for the front view in further range. The green boxes are vehicle classes, red cylinders are pedestrian classes, and blue boxes are two-wheeler classes.

Fig. 4.15 shows an overlay of all sensors with annotated boxes in a surrounding region. The data is in sequences of recordings, and the sequences are split into training or evaluation sets. The LiDAR are in gray for a better view. The green boxes are vehicle classes, red cylinders are pedestrian classes, and blue boxes are two-wheeler classes.

In the following chapters, experiments are conducted with the simulation data and the real-world dataset described in this chapter. Data is one of the most important topics in supervised learning, especially neural network training. Only with enough annotated radar data is it possible to start research work in radar perception tasks with ML and DL.

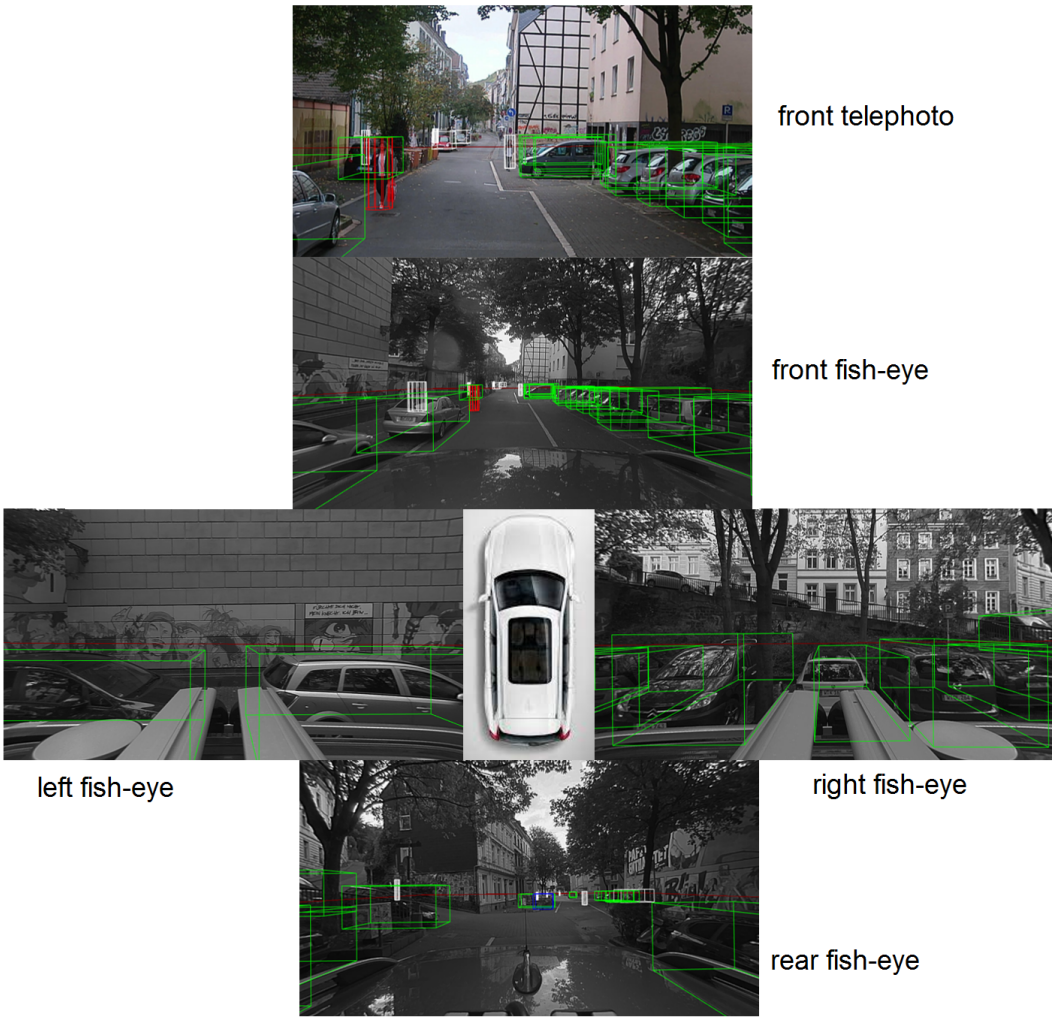

Fig. 4.14.: An example data frame of cameras with annotated bounding boxes.

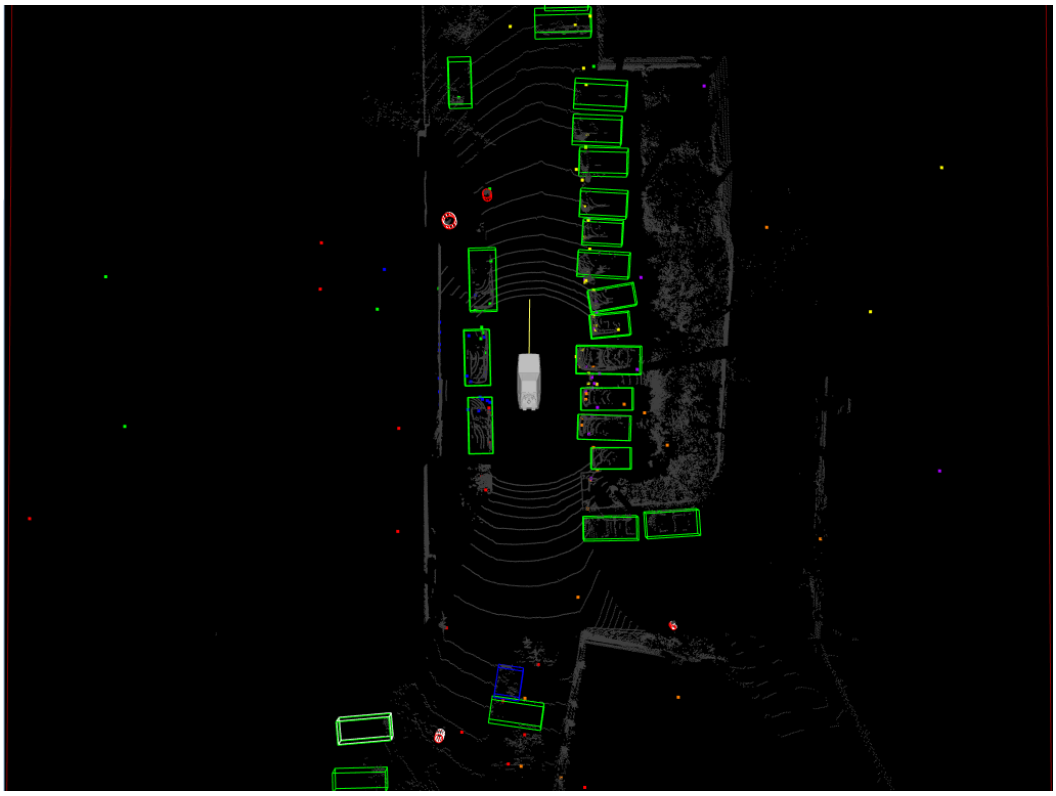

Fig. 4.15.: An example data frame with an overlay of all sensor detections and annotated bounding boxes.

5 A Deep Neural Network for Automotive Radar Perception

In recent years, along with the intensive development in ML, DL, and CV, research in ADAS and AD has been moving forward significantly [BT19]. Several sensors are widely used in driving assistant systems, including cameras, ultrasonic sensors, radars, and even the most recent LiDARs and vehicle-to-everything (V2X). Among all of these sensors, radars have their advantages in low-cost and high-accuracy distance and speed measurements. Such abilities are very useful in object detection in the far range and, most importantly, in accurate moving object detection. In common sense, the moving objects in a road environment around vehicles could include vehicles and vulnerable road users (VRUs). The latter include pedestrians, cyclists, and motorcyclists, whose lives are easily threatened in accidents or dangerous cases.

Detecting these moving objects, especially VRUs, is a very important task in the perception modules of ADASs. Seeing the enormous amount of research into CV, many researchers are also migrating and transferring their knowledge to the radar perception field. Due to the large flexibility of ANN and DL, the newly developed techniques in CV could easily migrate into DL on radar perception tasks. The greatest challenges lie in signal processing and feature embedding other than the high-level perception tasks.

Given the fact that images are normally rigid and dense, while the radar detections are non-rigid and sparse, the feature embedding is conducted differently from the methods applied mostly in CV tasks. The CNN is very hard to apply to such non-rigid and sparse data than images. The story does not end but starts from here. Many researchers have proposed multiple ways to deal with such challenges in the past few years. When starting research on radar perception by DL in this dissertation, similar challenges were presented.

This chapter first covers several state-of-the-art studies on radar perception by ANN or DL. Then it introduces research on the feature embedding of radar signals. It also discusses several developments in radar perception systems with the techniques of DL.

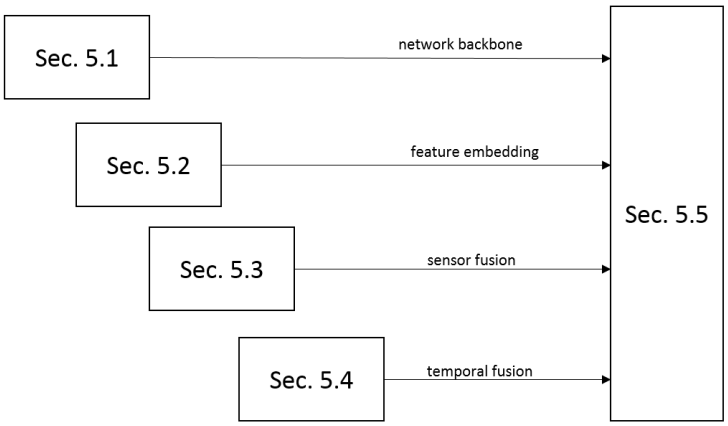

Fig. 5.1.: Relations between sections in this chapter; details are given in each respective section.

Sec. 5.1 introduces a few state-of-the-art neural network object detectors, one of which will be the backbone of the proposed radar object detector. Then, in Sec. 5.2, several radar feature embedding methods are proposed. Sec. 5.3 presents a few methods of sensor fusion in neural networks within a multi-sensor vehicle setting environment. In Sec. 5.4, a novel sampling-based temporal fusion recurrent network module is proposed. Finally, Sec. 5.5 shows a combined network of the mentioned proposals for radar object detection. Each section provides a building block for this deep neural network radar object detector. The relations between each section and the contribution of each section to this final object detector are shown in Fig. 5.1.

5.1 State of the Art

At the time when research interests moved from CV to radar perceptions, researchers noted a major issue they have not endured for a long time: no publicly available dataset. Especially for ML and DL research, data is one of the essential and most important components in the research. Due to business and intellectual property protection reasons, radar data is normally unavailable in a public space. More than that, the design of radar systems varies much across different manufactures, so that the data looks much more different than similar situations among camera images.

However, the story goes a bit differently in LiDAR. LiDAR shares several data properties as radars (e.g., sparsity and non-rigidity). Although they are still denser than radars in most cases, they are still far from comparable to camera images in density. Moreover, LiDAR has even higher accuracy in distance measurements,

though it suffers from a shorter maximal range due to safety reasons. Furthermore, there are several publicly available LiDAR content datasets for DL purposes. For these reasons, there have been many studies working on LiDAR data, other than radar data.

The story began with PointNet [Qi+17a]. Other than solving the tasks in grid-based data, the authors directly deal with points in a point cloud. Although this work does not deal with tasks based on LiDAR data, it shows several areas of potential for dealing directly with point-cloud data using DL methods and solving issues from sparsity. Fig. 5.2 shows the network structure proposed in their work. The design looks similar to an MLP but has several differences, such as point-wise feature embedding and feature transformation. With these techniques, the network does not suffer from issues due to the order of points of the input point cloud, and the processing runtime could also be optimized, as they are point-wise and shared weights.

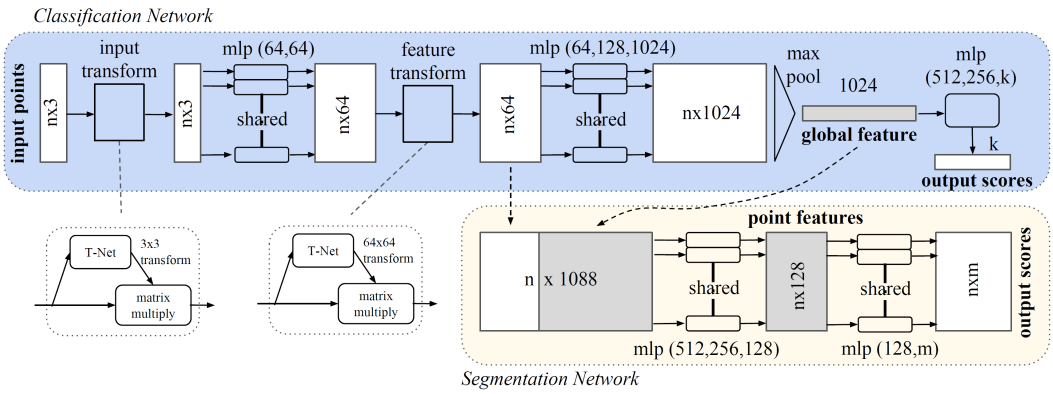

Fig. 5.2.: The proposed network structure of PointNet [Qi+17a].

To further improve PointNet, an advanced version called PointNet++ was proposed by the same research team [Qi+17b]. They improved the design by a new hierarchical structure on a sparse point cloud, which looks similar in concept to the CNN on image. Fig. 5.3 shows the proposed hierarchical structure of their network. The PointNet layers are doing feature embedding, and the grouping & sampling layer is comparable to convolution and pooling layers. Such a design is very impressive in showing how to transfer network design from CNN-based CV research to point-cloud data. For example, the semantic segmentation network design by the authors looks like a similar structure to U-Net for semantic segmentation on images [RFB15].

Another proposal by VoxelNet shows different possibilities [ZT18]. Other than directly processing and embedding features on points or hierarchical point clouds, the authors proposed a way to embed features from point-cloud into rigid grids. The

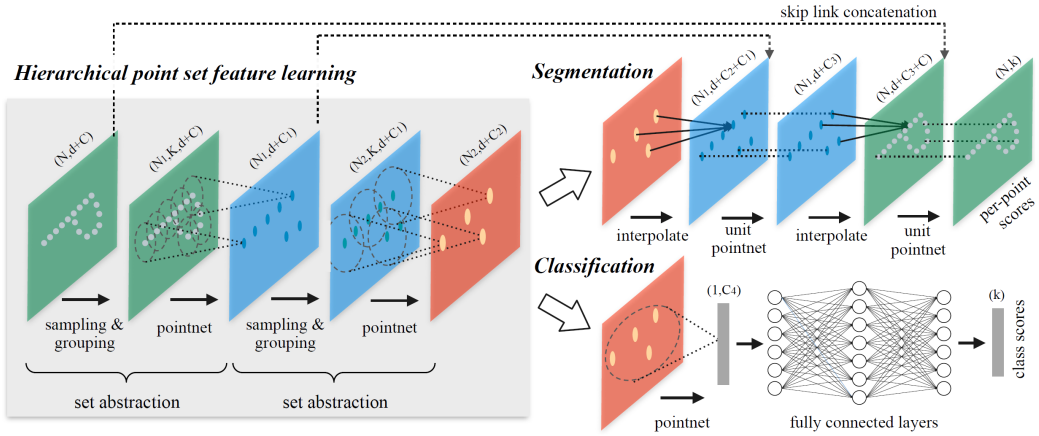

Fig. 5.3.: The proposed hierarchical network structure of PointNet++ [Qi+17b].

authors designed a voxel-feature-embedding network to embed features from points in a voxel to a grid-cell of a 3D matrix. In this way, the point cloud is embedded into a 3D dense image. After this embedding, the networks and techniques from the CV field can easily be applied. Fig. 5.4 shows the proposed network structure of VoxelNet. After the feature embedding network, the remaining modules are simply convolution layers and object detection techniques from image processing. Through VoxelNet, the authors show another encouraging possibility of processing point-cloud data using DL. In particular, they present one method to easily migrate techniques from previous CV research.

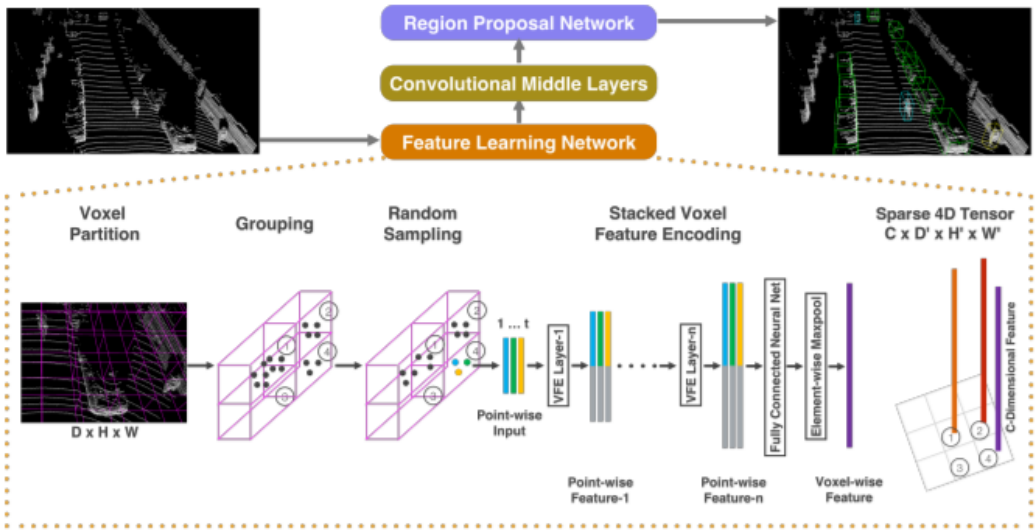

Fig. 5.4.: The proposed network structure of VoxelNet [ZT18].

When the story moves to radar perception, the researchers show a good example of applying PointNet++ to radar detections for semantic segmentation [Sch+18]. The

authors presented their experiments on the use of radar detection properties and showed the importance of velocity in object classification on radar data. Fig. 5.5 shows an example of a predicted frame from their network. It is clear that the important objects are detected in the semantic segmentation without much of the noise or false positives.

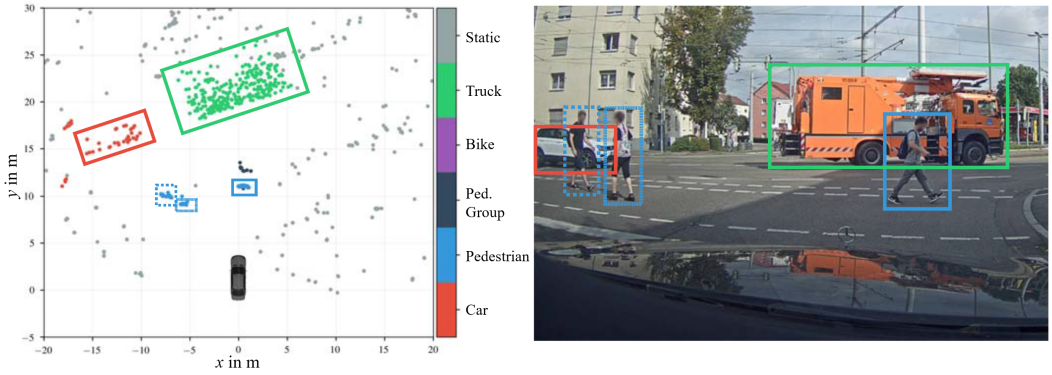

Fig. 5.5.: An example of predicted segmentation labels on a scene, rearranged from [Sch+18].

There are also many other works showing object detection networks with a fusion of sensor settings [NQ19][Nob+19]. Most of these works concentrate on using radar detections as object proposals or object localization and then use camera images for object classification. The fusion of different sensors inside a deep neural network shows more possibilities for DL for driving assistant systems.

In the most recent years, with the publication of several publicly available annotated datasets with multiple sensors (e.g., camera, radar, and LiDAR), many new studies and proposals have been presented on automotive perception tasks. The following sections will present the contributions primarily to radar perception tasks within the automotive field. Some of the research contributions were in a very early stage when there was not much research interest in this field or any public dataset available. The designs or architectures may not be optimal in the latest proposed approaches but were impressive at the time they were proposed. Previous research and contributing publications from the author, which the contents of following sections are mainly based on, will be referred respectively.

5.2 Radar Signal Embedding with Deep Neural Network

Deep neural networks are very strong and robust in feature embedding in various applications, including image processing or speech processing. In radar signal processing, the first challenge is to produce feature embedding from radar signals.

In order to take advantage of previous research on image processing, it is also important to put the radar signals into an image-like format. In this manner, various image processing techniques can be easily applied. Not similar to image processing or speech processing where unsupervised learning can be applied to produce image or speech feature embedding (e.g., autoencoder [BKG20]), radar signals cannot be easily reproduced and recovered by such autoencoders due to the high signal-to-noise ratio (SNR). Instead of reproducing the original input signals, networks are designed to perform perception tasks. In this way, a part of the trained deep network is dedicated to performing feature embedding.

This section will present several methods used to represent radar signals in image-like formats for various perception tasks. These methods are proposed in different phases of the research and on different datasets. The methodologies applied in early-phase research may look out-of-date from a latter point of view, but they were cutting-edge and efficient at the time when working on that data.

5.2.1 Radar Data Representation

According to different data sources, different representations of radar signals are fed into the network. The data is processed in different manners based on the source and the network structure or design.

5.2.1.1. Magnitude Map from Simulator

The first representation comes from the simulated data. In this data, only one radar is mounted at the front of the ego-car. The simulator performs ray-tracing on the environment and generates a reflection magnitude map in a polar space of SCS. This means that the data is in the form of a grid where each cell is in a polar coordinate and has a value of reflection magnitude.

To match the target space in a Cartesian VCS, it first transforms this magnitude map from polar space to a Cartesian space and translates it into VCS origin. Fig. 5.6 shows two example frames of this data. The two data frames (a) and (b) come

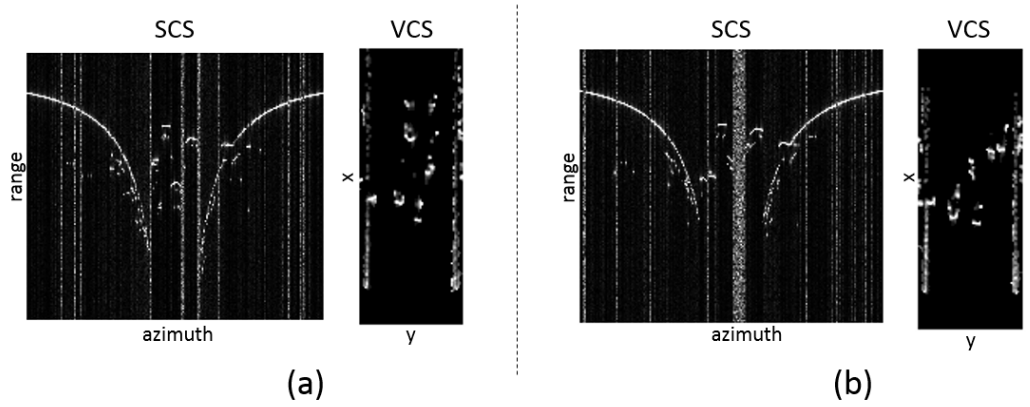

Fig. 5.6.: Example frames of magnitude map from simulator. (a) and (b) come from different frames in the same simulated data sequence.

from two different frames of the same data sequence. The environment is similarly defined only with movements of the objects. The magnitude map in SCS is with axes of range and azimuth angle, which are in a polar space. After coordinate transformation, the data are presented in Cartesian space in VCS. The simulator generates not only the pure cleaned radar response but also some white noises to simulate the SNR.

The data is with several objects (eight cars and two guardrails in Fig. 5.6) in the environment. As shown in the example frames, it is quite hard to identify all objects and also difficult to get the exact location and shape within a frame. To make use of temporal fusion (to be explained in Sec. 5.4), the data are generated in continuous sequences of frames. Each sequence contains 100 frames to ensure enough examples for temporal fusion.

Each frame is presented in an image-like format with x and y axes of VCS, and each cell is filled with the radar response magnitude as values. The network can easily handle such a format in a similar way to gray-scale images. When transforming from polar space to Cartesian space, it first calculates the coordinate of each cell in polar space. The coordinates of each polar cell are taken from the center of the cell in the map. When multiple transformed cells exist in one single Cartesian cell, it averages the response magnitude of all transformed polar cells. It is obvious that not every VCS cell can obtain a matched polar cell. In this case, the cells are filled with zero magnitude. Due to the non-linearity of the transformation, the data are denser in the near range and sparser in the far range in the VCS map. This can make it harder to detect objects in farther ranges than nearer ranges.

5.2.1.2. Property Map from Real-World Data

Instead of grid maps, the radar signals from real-world recordings are in the form of radar detection point clouds. Each point has properties of polar coordinate in SCS, Doppler velocity, and RCS magnitude. As mentioned in Sec. 4.3.5, these properties are further transformed into VCS Cartesian coordinates and compensated velocity.

To present these data in an image-like format, a similar method is used to put them in a Cartesian grid map in VCS. Since each radar detection is already transformed in VCS coordinates, simply place them in a cell in the grid map. Once the cells contain the relevant detections, it takes the maximum of compensated velocity and the maximum of the RCS values as two properties of each cell. Moreover, it also takes the logarithm of the number of detections $log(n + 1)$ as the third property of each cell. For empty cells, it simply sets all properties to zero. In this way, the network is able to see not only the radar signal properties, but also the density of the radar detections.

Fig. 5.7 shows two example frames of the property map in VCS. Each map contains three data channels, namely the "Doppler" (compensated velocity), the RCS and the "#Detections" (number of detections). Each property is further scaled to reduce the inner variance of the data. The grid cells are a half meter large, and the VCS is defined as 40 meters in each direction; hence, it is 160 pixels in height and width. Due to the high sparsity of radar detections, only very few cells in the map are filled with values. Compared to the simulated data, it is even harder to identify objects within a single frame.

After preprocessing, each frame is an image-like format with three property channels. The network can easily handle such a format in a similar way of RGB-colored images. To deal with the high sparsity data, temporal fusion will play an important role in network design. It collects several sequences of recordings, and each sequence contains several frames. With this data, the network will be able to perform temporal fusion to achieve better performance.

5.2.1.3. Gridized Point Cloud from Real-World Data

With the development of the research, it has been found that the hand-crafted properties for each grid cell have their limitations. For example, the maximum velocity cannot separate multiple nearby objects when they fall into the same cell. The maximum RCS cannot separate objects near a large reflection surface (e.g., a pedestrian next to a wall).

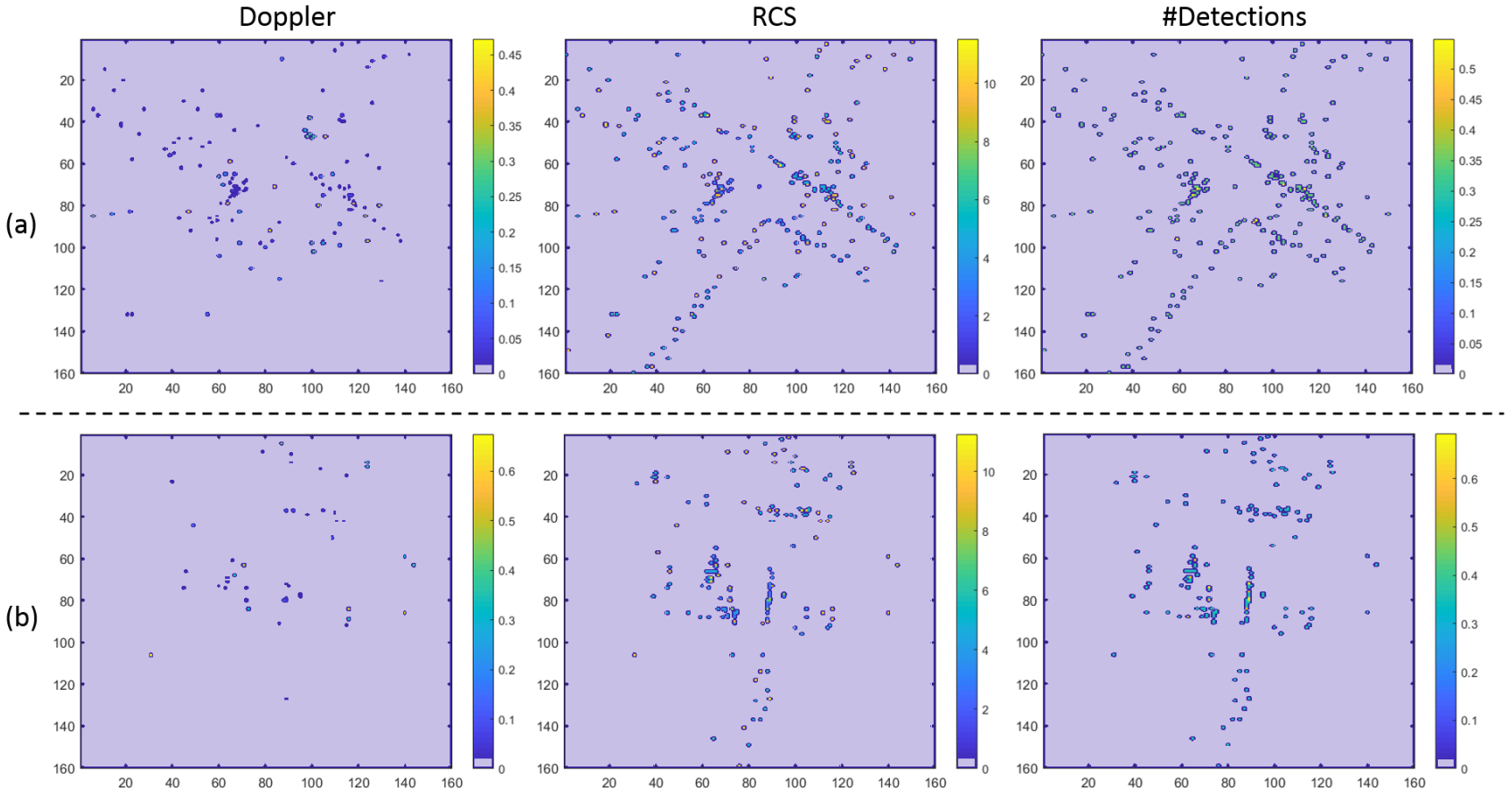

Fig. 5.7.: Example frames of property map from real-world data. (a) and (b) comes from different frames in the same recording sequence. Colors for near zero values are modified for better visualization.

To increase the capability of the network, another data format is designed, which is also an image-like format, but with much richer features. Instead of calculating the properties of the radar detections within a grid cell, it puts the radar detections as is. To make this possible as a fixed data size for the network, a fixed number of detections is defined for each cell. If the number of radar detections is higher than this number, the preprocessor randomly downsamples to the desired number. Otherwise, it randomly upsamples the detections and adds small noise to the location features. When no detection exists in the cell, it fills all features as zero.

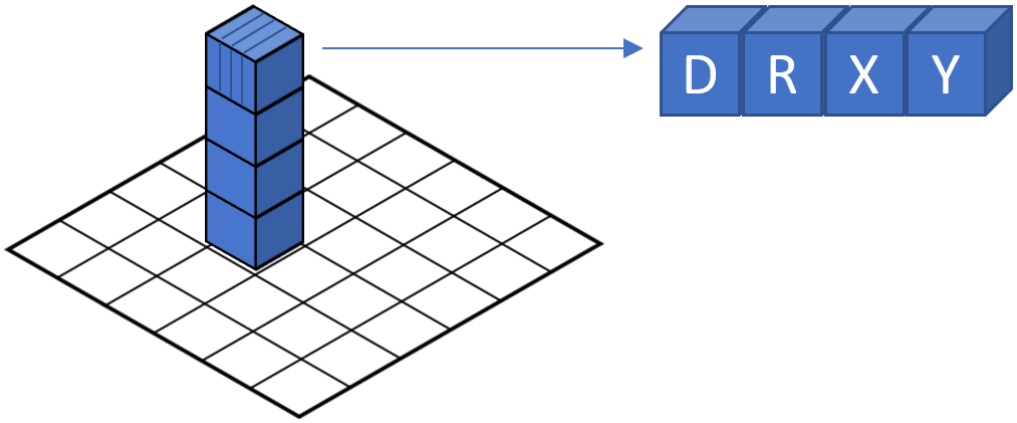

Fig. 5.8.: An illustration of gridized point cloud. Each cell contains four detections, and each detection has four properties.

Fig. 5.8 shows an illustration of the gridized point-cloud format. Each grid cell contains four detections. Each detection contains four features, namely "D" for compensated velocity, "R" for RCS, "X" and "Y" for the location coordinates. With this data, the network is able to see radar detections from different objects and also has richer features for object detection.

Instead of three dimensions for images, namely height, width, and colors, this new image-like format has four dimensions. The traditional image processing network is no longer able to handle this format. Hence, a new network structure is needed. The next section will introduce the design of a new network structure to deal with such a format.

5.2.2 Deep Neural Network for Feature Embedding

Deep neural networks are very effective in feature extraction and feature embedding. On image-like maps, CNN shows robustness in both local and global feature

extraction. With the small kernel and shared weights, CNN can perform well on a spatially homogeneous feature space.

When processing the radar signals described in the previous section, primarily CNN-based deep neural networks are utilized. As they are all in an image-like format, multiple CNN-based methodologies can be easily applied. Parts of the following content are based on our previous publications [NZS19a][NZS19b][NZS19c].

5.2.2.1. Feature Embedding on Grid Image

To perform object detection on the magnitude map from the simulator, an image segmentation task is conducted to segment the object pixels. In this manner, if the segmentation predictions are well-performed, the network is capable of separating radar signals from objects and objects from the background.

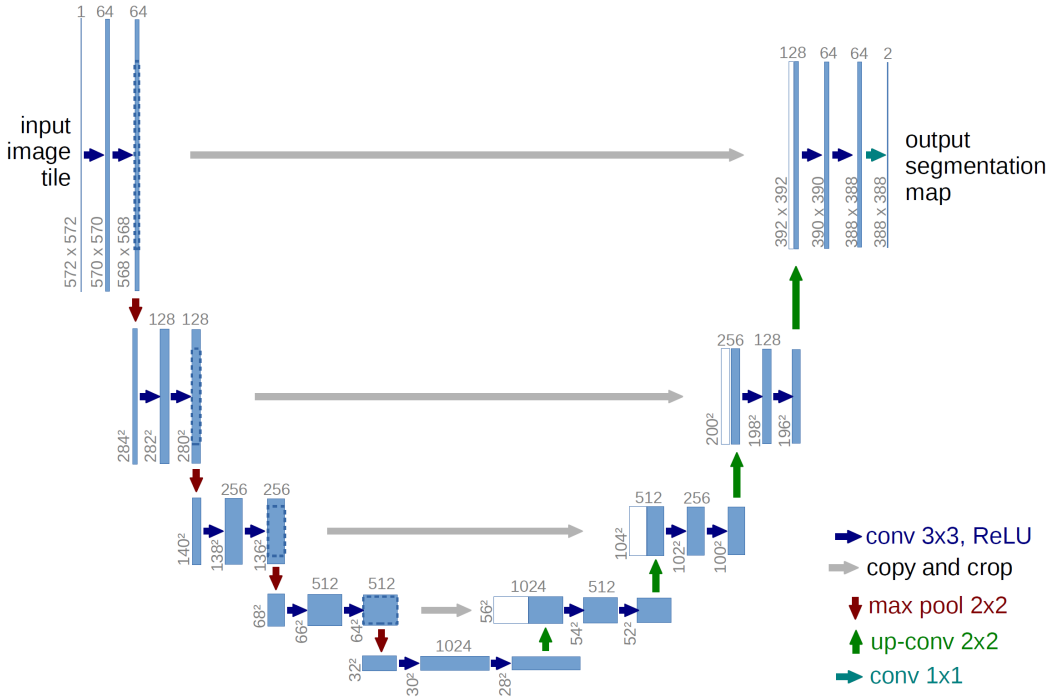

Fig. 5.9.: The network structure in U-Net [RFB15].

U-Net has a very good design for performing image segmentation [RFB15]. It can extract higher-level features by multiple downsampling and upsampling, but it can also keep lower-level features by copying and concatenating the inner layer outputs to the upsampled features. Fig. 5.9 shows the original network design proposed by the authors.

With the connection from layers before downsampling to the layers after upsampling, the combination of low-level features (e.g., shapes, edges) and high-level features (e.g., semantics, objects) can show good object segmentation while keeping good shape predictions. Without these connections, the network tends to output blurry boundaries of the segmentation due to the upsampling operations.

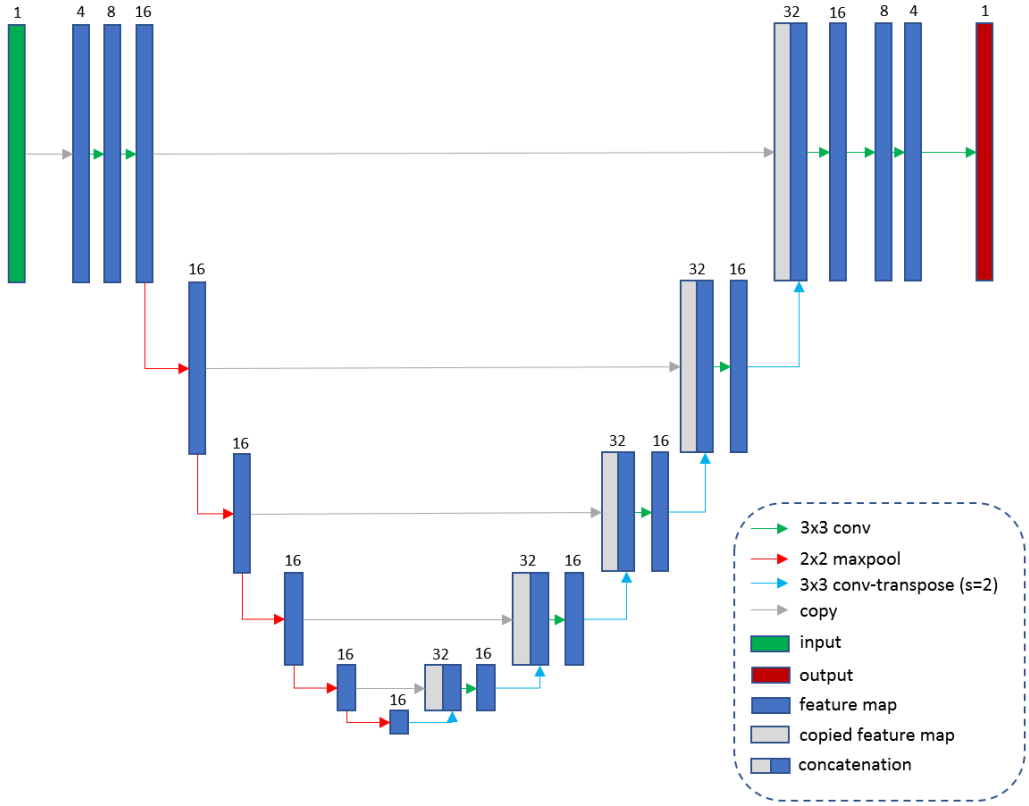

Fig. 5.10.: The network structure for radar signal segmentation.

Fig. 5.10 shows the network structure used for the radar segmentation task. The network has six levels of features. The downsampling is done by max-pooling, and upsampling by transposed convolution with stride of two. In each level except the deepest one, it adds connection (copy and concatenation) between the features before downsampling and after upsampling. The numbers indicate the channels of each feature map. Very large numbers of channels are not used in each feature map due to the limited channels from the input, and also to reduce unnecessary computational complexity.

Fig. 5.11 is one example of this network performing on radar signals. Although the input is very noisy, the network is still capable of outputting good shapes of the objects. The only missing object (middle-left) is occluded by the other objects; hence, it is hardly possible to be found by the network. The network also shows the

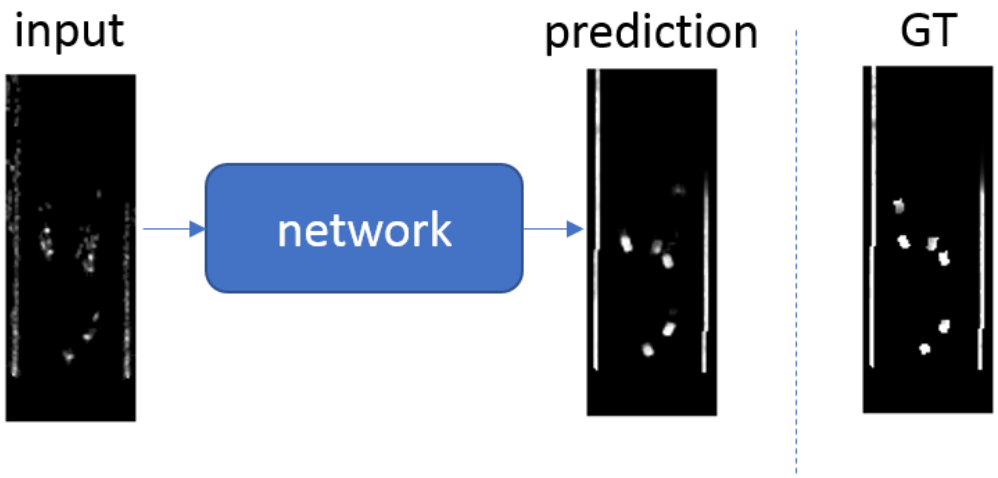

Fig. 5.11.: An example of the input, ground truth, and network prediction of semantic segmentation on simulated radar signals.

capability to reduce noise when seeing the noise in front of the bottom-right object in the input map, but cleaned prediction in that area. Unfortunately, the network is not perfect, as there are some low-valued predictions in the top-right area where there is only noise in the input, but no object in ground-truth.

Based on these findings, the neural network is capable of finding objects in a very noisy input map from radar signals. Moreover, the network is also capable of reducing background noises to some extent.

Similar to the magnitude map, a similar network can be used to extract features from the property map input. In general, any image-like map with multiple channels can use a similar network to extract features for the objective task. The U-Net structure is very useful in extracting local and global information within an image. For radar object detection use case, it is good in finding local information of object shapes and edges but also has good performance in finding global information like semantics and distinguishing between objects and background noise.

5.2.2.2. Feature Embedding on a Gridized Point Cloud

In order to handle a gridized point-cloud map in a neural network, a special network structure is necessary. VoxelNet has shown one good design in processing a voxelized LiDAR point cloud [ZT18]. In a similar manner, the following network structure is proposed to process a gridized radar point cloud.

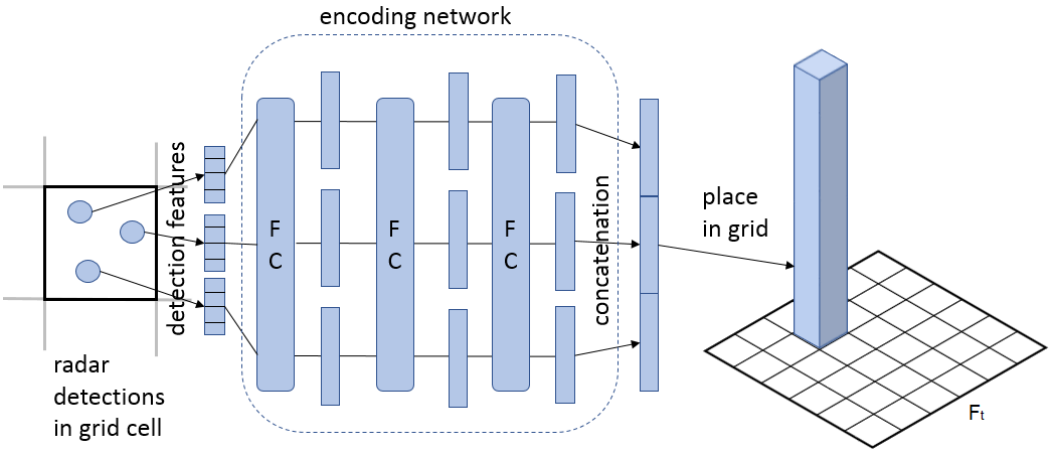

Fig. 5.12.: An illustration of a gridized point-cloud encoding network.

Fig. 5.12 shows the encoding network structure. Each grid cell has either a specific number of points or zero points. Each point has four features ("detection features" in Fig. 5.12). For each point, it runs a number of fully connected layers (three layers in the figure) for feature encoding. Each layer uses shared weights; hence, encoding is common for all points. In the case of an empty cell, it uses zeros as features in the meaning of empty to the network.

After the encoding network for each point, it finally concatenates the feature vector into a long vector and places the vector in the output tensor at the position of the cell. To reduce overfitting, the order of the concatenation is random. Once the feature encoding network runs on each cell, the final output feature map ("Ft" in Fig. 5.12) should have a shape of $(H, W, n \times C)$, where H and W are grid size dimensions, C is the length of the output feature vector of the encoding network for each point, and n is the specific number of points set to each grid (in the example, $n = 3$).

The output feature map after this point-cloud encoding is in an image-like format with multiple channels. With this feature map, a common feature extraction network can be applied (e.g., U-Net).

5.3 Sensor Fusion by Deep Neural Network

In ADAS and AD sensor settings, it is very common to use more than one sensor. In some settings, multiple sensors of the same type are mounted on the surroundings of the ego vehicle to give a 360° view of the environment. Only in this way are the assistant systems able to reduce the limitation of blind spots of sensors and of the

driver. In more general settings, multiple sensors of different types are used. In this case, due to the different strengths and capabilities of different sensors, the assistant system is able to use different sensors to deal with different tasks. These include cameras for traffic sign and traffic light recognition, radars for far-range vehicle detection, and LiDAR for near-range pedestrian detection.

One challenge in multi-sensor settings is how to fuse the perception outputs from different sensors. One common method is to use the Kalman filter [SH00]. In these methods, the observations are taken from the perception results from different sensors. As they are processed by different observation models, the differences in sensor behavior or capability are considered and resolved.

This section will introduce the different sensor fusion methods used and proposed during the research. The first method is on homogeneous sensors (sensors of similar type or with similar properties). This method takes a straightforward approach to fusing sensor observations. Then, a more general approach will be introduced that can deal with sensors of different types within a neural network. This method is more flexible in fusing information, as it does not require homogeneous sensor settings. Part of the following content is based on our previous publications [SZM20a][SZM20b][SZM20c].

5.3.1 Sensor Fusion by Data Preprocessing

When fusing homogeneous sensors (e.g., multiple side radars on a car), the easiest approach is to put the radar detections from each sensor into a common space. In a previous example (Fig. 5.7), it first transforms the coordinates of radar detections from six side radars around ego vehicle into VCS. Then, it preprocesses these detections purely in VCS, so that all of them are in the same spatial space.

The main reason for this method is because these radar detections share the same property list, which means that other than physical differences, the sources of detections are not distinguishable. In such a manner, when a network is processing the point cloud with fused radar detections, it is possible to be trained to model the physical behavior of the object observed by different sensors. For example, due to the mounting position difference, the object observed by different radars can have different relative velocities. Even after compensation, the velocity still has only the normal component of the real ground velocity vector. In this case, the network will be able to resolve the ground velocity.

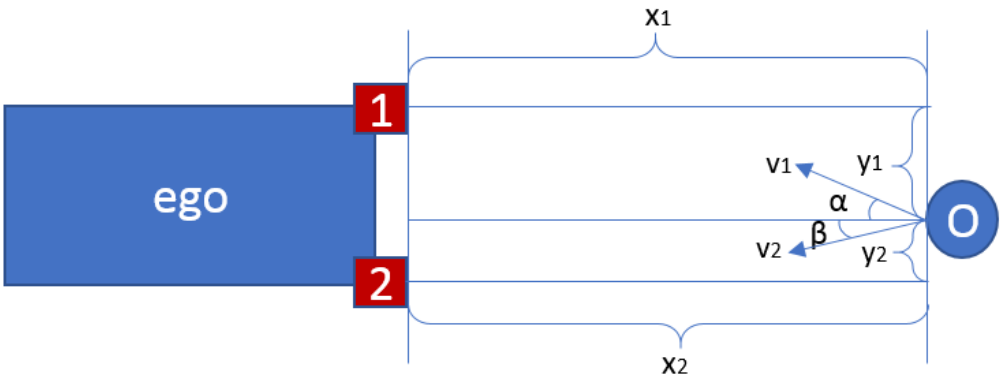

Fig. 5.13.: An illustration of solving the real ground velocity vector of an object by two radar detections.

Fig. 5.13 presents an object (circle "O") detected by two radars (red rectangles "1" and "2"). Each detection has properties of spatial locations (x_1 y_1 and x_2 y_2) and compensated velocity (v_1 and v_2). The ground velocity vector of the object can be estimated by solving vector components v_x and v_y in equations

$$\begin{aligned} v_1 &= v_x cos(\alpha) + v_y sin(\alpha) \\ v_2 &= v_x cos(\beta) + v_y sin(\beta), \end{aligned} \tag{5.1}$$

where the trigonometric functions can be easily calculated by the position measurements as

$$\begin{aligned} cos(\alpha) &= \frac{x_1}{\sqrt{x_1^2 + y_1^2}} \\ sin(\alpha) &= \frac{y_1}{\sqrt{x_1^2 + y_1^2}} \\ cos(\beta) &= \frac{x_2}{\sqrt{x_2^2 + y_2^2}} \\ sin(\beta) &= \frac{y_2}{\sqrt{x_2^2 + y_2^2}}. \end{aligned} \tag{5.2}$$

This is only an educational example. In practice, to reduce the effect of measurement noises, the more detections of the same object can be measured, the higher-quality velocity vector estimation it is possible to achieve.

5.3.2 Sensor Fusion by Neural Network

When handling sensor fusions of different types, putting the detection or measurements in the same space will not work. Either the properties are different from different sensors (e.g., radar has location, Doppler velocity, and RCS, while LiDAR has location and reflection intensity). Although RCS is also related to reflection intensity, the definition and physical behavior are different from those in LiDARs. Even more complex, the camera images are not in a point-cloud format at all, which makes it difficult to place them in a common space such as VCS.

To solve these issues of different sensor types, a new method is proposed using a deep neural network to fuse sensor measurements. Inspired by the feature embedding methods in Sec. 5.2.1, it can use a neural network to encode sensor measurements into an image-like format. Then it can further process and extract features within each sensor, specifically using neural networks. These neural networks can work purely on specific sensor data, without interaction with other sensors. When this image-like format indicates a common space (e.g., VCS), the network is able to fuse and process the embedded features from different sensors in the same space. For the non-geometry data (e.g., camera images), the feature encoding network can be designed to embed the features into a geometry space [Rod21].

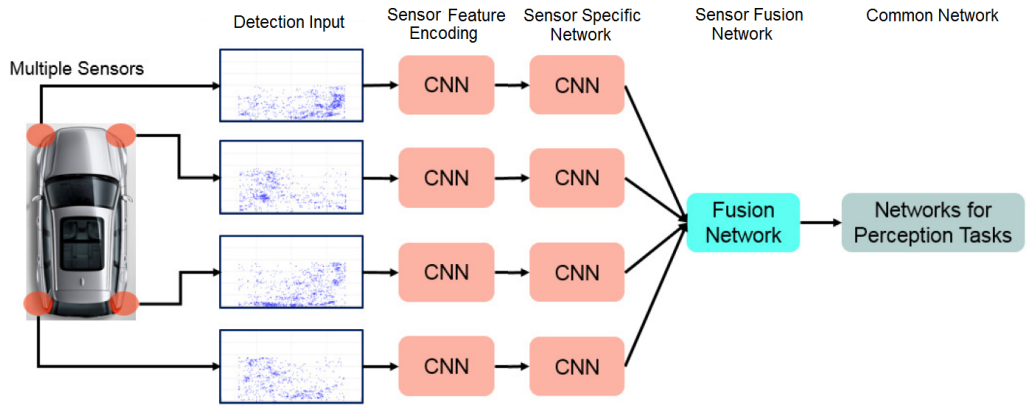

Fig. 5.14.: An illustration of the design of sensor fusion in a deep neural network.

Fig. 5.14 shows an illustration of sensor fusion in a deep neural network. Instead of putting the raw sensor measurements directly in a common space, it first performs feature encoding by neural network (e.g., CNN) using measurements from each sensor specifically. After feature encoding, the feature maps of each sensor are further processed by a network dedicated to the current specific sensor. In such a manner, these networks can be placed and executed on sensor hardware before transferring to

central processors. After the sensor-specific feature extraction, the feature maps are combined and fused together by a fusion network. Several methods can be applied when fusing different feature maps together (e.g., max-pooling, concatenation, or weighted sum). Once the feature maps are fused in VCS, a general perception network can be applied to this fused feature map to perform perception tasks.

One advantage of this fusion method is that the computation load can be spread partially on each sensor. In this way, not only can the data transferring from sensors to central processors be optimized via feature embedding, but also the computational effort can be further balanced. Especially when the central processor is heavily loaded due to multiple tasks, the perception task load can be reduced by putting a sensor-specific network on the sensor processor.

A second advantage is the end-to-end training strategy. When all feature embedding and perception tasks are connected within a large deep neural network, it is possible to train the network in an end-to-end fashion. The feature extraction and embedding in each sensor-specific network can be further improved by internal learning during back propagation. The feature may be focused on the strength of the specific sensor type, while the weakness is already resolved by another sensor type. It is hardly possible to design such focus behavior manually.

5.4 A New Sampling-Based Recurrent Neural Network Module

Sec. 5.2.1 discussed the sparsity of radar signals in data representations. In the example frames, it is clear that the actual radar measurements are too sparse to easily detect objects. Even with an effective deep neural network, the performance is still far from satisfactory.

To make use of sequential data that can acquire multiple observations along with the elapsing of time, RNN is a good choice in temporal fusion. This section first introduces a few advanced RNN designs, which are the fundamentals of the research. Then it briefly recaps previous contributions of sampling methods in neural networks. Finally, it presents a proposed sampling-based RNN design that is specifically intended to improve ADAS and AD tasks.

The majority of the content in this section is based on our previous publications [NZS19d][NZS19e][NZS19f].

5.4.1 Recurrent Neural Network

Sec. 2.1.1.3 briefly introduces the most fundamental concept of RNN. It contains a recurrent connection over time that can keep memory and pass to future time (and in bidirectional cases, also to the past time [SP97]).

5.4.1.1. Long Short-Term Memory

To reduce gradient vanishing and explosion issues, several advanced designs of RNN were proposed. One of the most famous design is long short-term memory (LSTM) [HS97]. Instead of directly passing the hidden state over time, LSTM introduced several filter gates that can let the network control the data flow and the amount of information to be passed and stored in memory. Moreover, it also has two memory states instead of one, one of which is for short-term memory, and the other is for long-term memory.

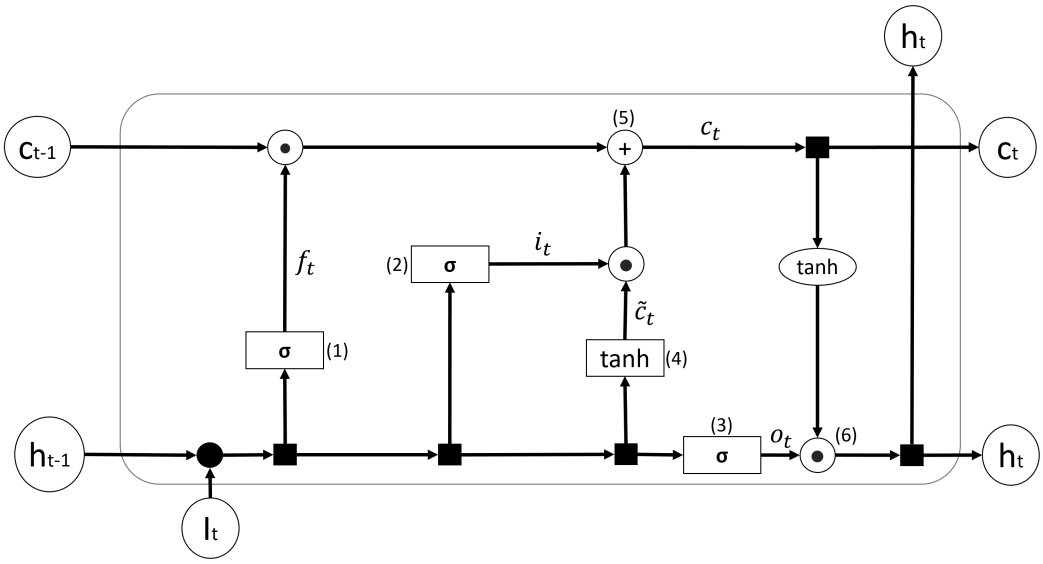

Fig. 5.15.: An illustration of the design of LSTM. The corresponding equations are Eq. 5.3 (1), Eq. 5.4 (2), Eq. 5.5 (3), Eq. 5.6 (4), Eq. 5.7 (5), and Eq. 5.8 (6).

Fig. 5.15 shows a brief illustration of the LSTM structure. The legend of the illustration follows Fig. 5.16. All network design illustrations in this section follow the same legend definition. In LSTM, the first component is the forget gate,

$$f_t = \sigma(W_f I_t + U_f h_{t-1} + b_f), \tag{5.3}$$

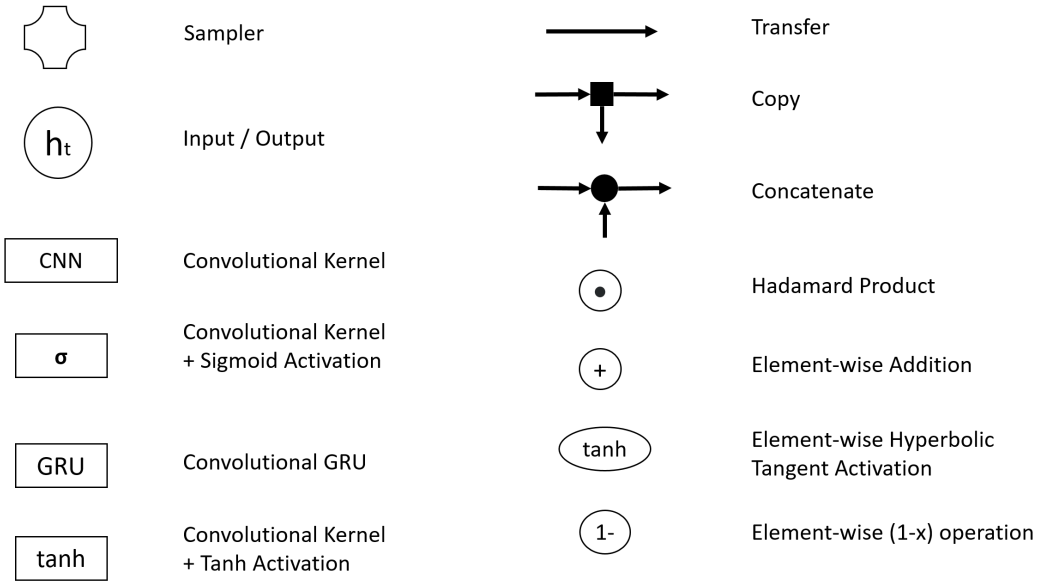

Fig. 5.16.: The legend of all network design illustrations in Sec. 5.4

where sigmoid activation (σ) is applied to a fully connected layer on concatenation of input signal of current time I_t and the short-term memory (hidden state) from the last time h_{t-1}. This forget gate is designed to control the data flow of long-term memory (cell state) from the last time c_{t-1}. This reduces the amount of long-term memory to be used; hence, it is called a forget gate.

The second component is the input gate,

$$i_t = \sigma(W_i I_t + U_i h_{t-1} + b_i), \tag{5.4}$$

which controls the data flow of the current input data. This reduces the amount of current input data to be used; hence, it is called an input gate.

The third component is the output gate,

$$o_t = \sigma(W_o I_t + U_o h_{t-1} + b_o), \tag{5.5}$$

which controls the data flow of the output data. This reduces the amount of output data to be used; hence, it is called an output gate.

The fourth component is input processing, that is,

$$\tilde{c}_t = tanh(W_c I_t + U_c h_{t-1} + b_c), \tag{5.6}$$

which transforms the raw input signal into cell state space and also considers the short-term memory from the last time. This signal is considered to be processed input data in the same feature space of the cell state. Instead of sigmoid activation, hyperbolic tangent activation is applied.

Then the new cell state of current time is calculated by applying the gates as

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \tag{5.7}$$

where the forget gate is applied to the cell memory and the input gate is applied to the processed input data. The gates are performed by a Hadamard product, as their value range after sigmoid activation is (0,1). After adding up, the output signal is considered the new cell memory of the current time.

Finally, the output and the new short-term memory to be passed to the future is calculated by

$$h_t = o_t \odot tanh(c_t), \tag{5.8}$$

where a hyperbolic tangent activation is applied to the cell memory and the output gate is applied to control the output data flow.

It can be seen that the gates are important in this design, while the data flow can be controlled by dynamic signals based on the current input and the short-term memory. In this way, the network gains a much larger capability to deal with the complex control of the data within the recurrent connections. As a drawback, computational complexity has been largely increased. LSTM has four fully connected layers inside, together with a few more element-wise operations. It has about four times the computational cost of a conventional RNN unit.

The original LSTM is designed for fully connected layers. In a modern use case, such as video processing, convolutional LSTM was proposed [Xin+15]. Instead of the dot product of vectors, the operations are replaced with convolutional kernels. Moreover, the authors also proposed a peephole connection design that is different from the original peephole LSTM in [GS01].

The peephole convolutional LSTM is with

$$
\begin{aligned}
f_t &= \sigma(W_f * I_t + U_f * h_{t-1} + V_f * c_{t-1} + b_f), \\
i_t &= \sigma(W_i * I_t + U_i * h_{t-1} + V_i * c_{t-1} + b_i), \\
o_t &= \sigma(W_o * I_t + U_o * h_{t-1} + V_o * c_{t-1} + b_o), \\
\tilde{c}_t &= tanh(W_c * I_t + U_c * h_{t-1} + b_c), \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \\
h_t &= o_t \odot tanh(c_t),
\end{aligned}
\tag{5.9}
$$

where $*$ denotes the convolutional operator. The main difference is that the gates also consider long-term memory. In this way, the short-term noise can be reduced.

5.4.1.2. Gated Recurrent Unit

To further reduce the high computational complexity in LSTM, a simplified recurrent unit, the gated recurrent unit (GRU), was proposed [Cho+14]. In GRU, the output gate is removed, and instead, one single gate is used to balance the output signals.

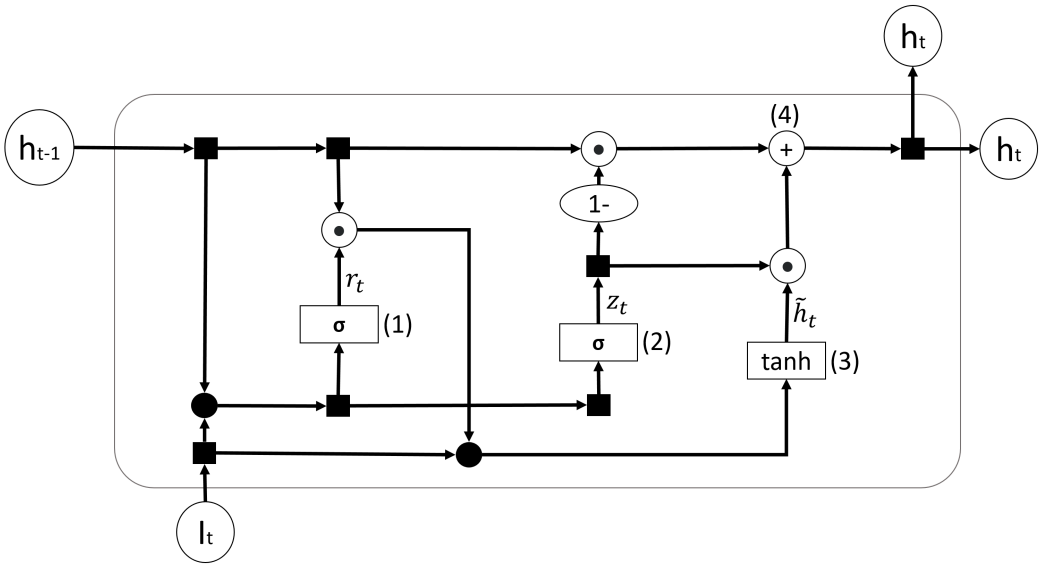

Fig. 5.17.: An illustration of the design of GRU. The corresponding equations are Eq. 5.10 (1), Eq. 5.11 (2), Eq. 5.12 (3), and Eq. 5.13 (4).

Fig. 5.17 shows an illustration of the structure of the GRU. Instead of two memories (cell memory and hidden state), this unit has only one memory (hidden state). The first component is reset gate,

$$
r_t = \sigma(W_r I_t + U_r h_{t-1} + b_r), \tag{5.10}
$$

which controls the data flow of the memory used. This gate is similar to the forget gate in LSTM.

The second component is the update gate,

$$z_t = \sigma(W_z I_t + U_z h_{t-1} + b_z), \tag{5.11}$$

which controls the balance between the old memory and new input data.

The third component is the input data processing, that is,

$$\tilde{h}_t = tanh(W_h I_t + U_h(r_t \odot h_{t-1}) + b_h), \tag{5.12}$$

where the new input is combined with the old memory but filtered by a reset gate. The processed input data contains not only the observation from current time but also an appropriate amount of memory from the past.

Finally, the output and the new memory of the current time are calculated by

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t, \tag{5.13}$$

where the output and new memory are a balance of the processed input data and the old memory via an update gate.

The advantage of GRU is that it has one less layer than LSTM. Also, it reduces the possibility that when all the gates are zero, the memory will be cleared in LSTM. In GRU, when all gates are zero, the output is identical to the old memory.

Similar to convolutional LSTM, all fully connected layers can be replaced by convolutional layers, which will be usable in video processing [Bal+15].

With gate design and memory handling, the capability of recurrent control of RNN has been largely increased. In addition, the conventional issues during training have been largely reduced.

Fig. 5.18.: An example of the movement flow of a vehicle in the environment.

5.4.2 Sampling in Neural Network

Chap. 3 introduced a sampling method inside a neural network. Instead of normal convolutional kernels or fully connected layers, sampling can largely increase the flexibility of neural networks in feature extraction and feature refinement.

When the feature map is in a bird-eye view of VCS, the feature vectors of the objects are normally located near the actual location in VCS on the feature map. In this case, if the object is moving in the environment, the related feature vectors are also moving in the feature map accordingly. Fig. 5.18 shows an example of an object moving in the environment. The red arrows indicate the movement of the component. In a similar concept, if the feature vectors are moved in the feature map relative to the object movement, the network will be able to keep the same information as the object without efforts to recalculate the features from scratch.

To move the feature vectors in a feature map, sampling is a very useful technique. As was discussed, the sampling method supports backpropagation during training; hence, any network with sampling method can still be trained in an end-to-end fashion.

The objects are moving along with time. Temporal fusion can improve the performance of object detection as the information is gathered from multiple frames in a sequence. If the feature vectors of memory can move according to the object movement, the network can directly reuse the memory through direct concatenation to the current observation at the same location on a feature map. Based on these inspirations, the following recurrent network unit is proposed.

5.4.3 Gated Recurrent Sampler

To make better use of RNN memories, gates are very useful techniques in dynamically balancing the data flow over time. To better fit the automotive use cases, sampling on a VCS feature map is a good method to reproduce and match the object movements in the local environment. With all of these considerations, a novel convolutional recurrent network unit, gated recurrent sampler (GRS) is designed and proposed.

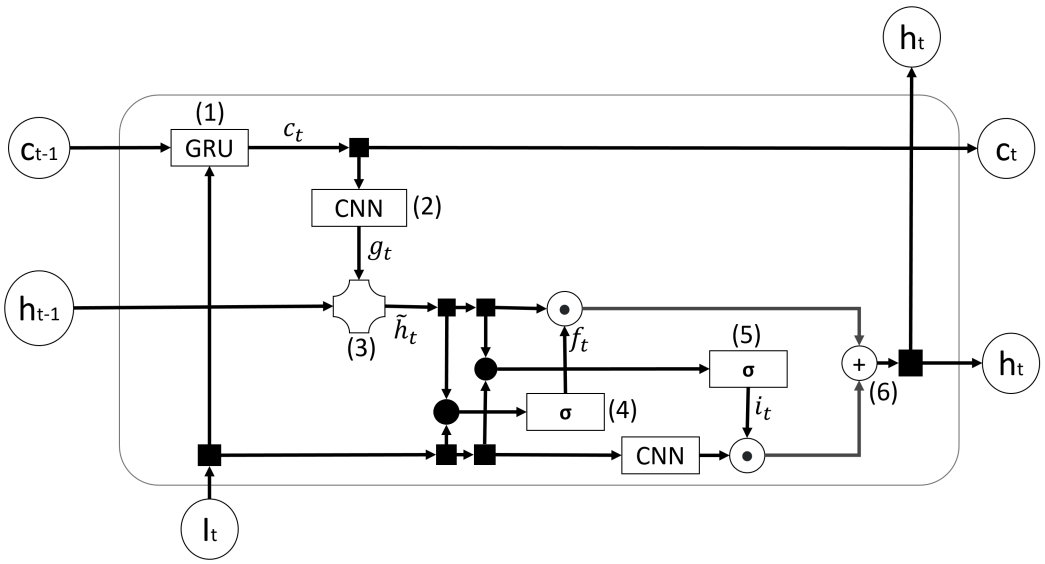

Fig. 5.19.: An illustration of the first version of GRS structure. The corresponding equations are Eq. 5.14 (1), Eq. 5.15 (2), Eq. 5.16 (3), Eq. 5.17 (4), Eq. 5.18 (5), and Eq. 5.19 (6).

GRS is a convolutional recurrent network unit that contains gates in the structure and a sampler that can move feature vectors in the feature map of memories. Since it works on a grid feature map, all operations are convolutional kernels instead of fully connected layers. Fig. 5.19 shows the first version of the proposed GRS. The structure is designed according to a concept similar to LSTM, but each component is based on different objectives.

The first component is cell memory,

$$c_t = GRU(SG(I_t), c_{t-1}). \tag{5.14}$$

Unlike the cell memory in LSTM, the cell memory in GRS is designed to remember and keep the movement of objects. It uses a convolutional GRU with the past cell memory (as GRU states) and the current input features to refresh the movement features of the environment. This cell memory is isolated from the feature map for

the main stream task of the neural network; hence, it is designed to focus only on object movements. For better isolation, a stop-gradient operation ("SG") can be applied to the input features, so that the previous feature extractions for the main stream task are not affected by this movement extraction.

The second component is to calculate the sampling grid of movement, that is

$$g_t = \theta(W_g^{\times 2} * c_t + b_g^{\times 2}), \tag{5.15}$$

where a CNN with an output of two channels for a 2D feature map is applied to the current cell memory. θ indicates an activation function for this CNN layer. This activation function can be a linear function if the movement of objects can happen in all cases. Each channel indicates the movement of one dimension of each pixel. This sampling grid is used in the third component, the sampler,

$$\tilde{h}_t = Sampler(h_{t-1}, g_t), \tag{5.16}$$

where the memory from the last frame is refined by the object movements based on the grid via a sampler. The design of this sampler is identical to the sampler in Sec. 3.2.3. The resampled memory should match the current locations of objects in the feature map.

After the sampling, it calculates the forget gate,

$$f_t = \sigma(W_f * I_t + U_f * \tilde{h}_t + b_f), \tag{5.17}$$

and the input gate,

$$i_t = \sigma(W_i * I_t + U_i * \tilde{h}_t + b_i). \tag{5.18}$$

These gates will be used to control the data flow of resampled memory and the current input feature.

In the end, the new memory and the output is

$$h_t = f_t \odot \tilde{h}_t + i_t \odot \phi(W_e * I_t + b_e), \tag{5.19}$$

where it first transforms the input features into an output space by a CNN layer, then applies the two gates to the memory and the transformed input. ϕ indicates an activation function of this CNN layer. The combined features are used as both the new memory for the future and the output of this recurrent unit for the current frame.

Although the structure looks like an LSTM, it does not use an output gate to reduce the computational complexity. However, the overall computational complexity is still very high due to the grid processing and the GRU on cell memory.

5.4.3.1. Simplified GRS with Cell Memory

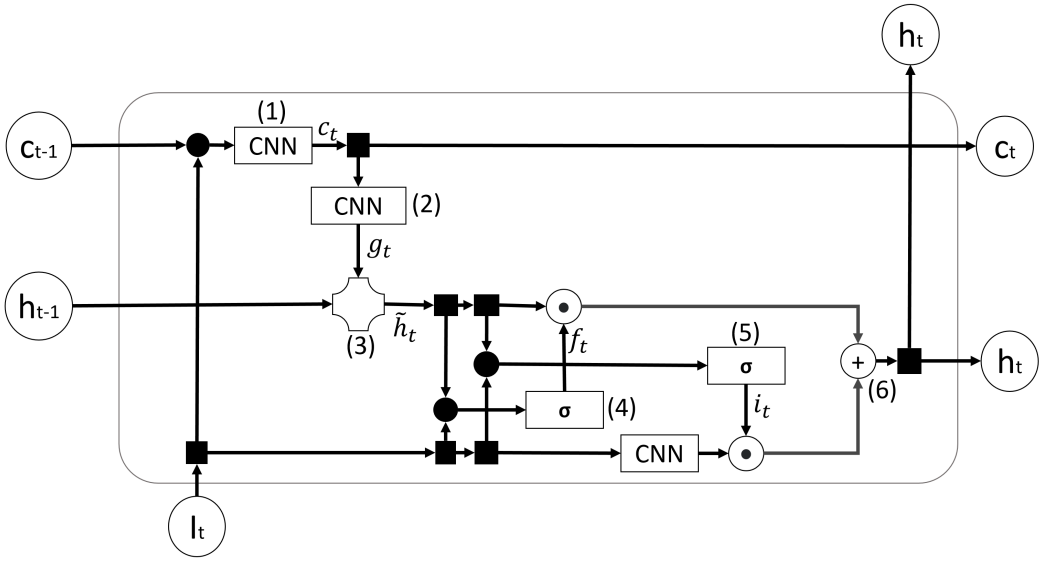

Fig. 5.20.: An illustration of the simplified version of the GRS structure. The corresponding equations are Eq. 5.20 (1), Eq. 5.15 (2), Eq. 5.16 (3), Eq. 5.17 (4), Eq. 5.18 (5), and Eq. 5.19 (6).

To reduce computational complexity, a simplified version of GRS is proposed. Fig. 5.20 shows an illustration of this simplified version. Instead of a complex GRU, it uses a simple CNN to combine the cell memory and the current input with stop-gradient isolation ("SG") as

$$c_t = \phi(W_c * SG(I_t) + U_c * c_{t-1} + b_c). \tag{5.20}$$

This change can largely reduce the computational cost but still keeps the original concept to combine the old cell memory with the current input. The remaining components in the simplified GRS remain the same.

5.4.3.2. Simplified GRS with Cell Memory and Peephole Connection

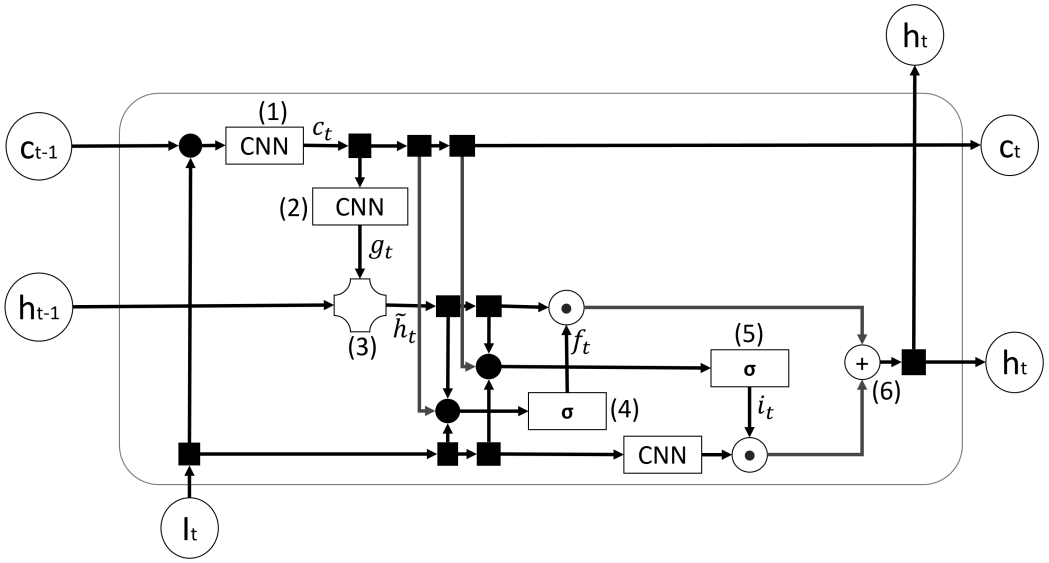

Fig. 5.21.: An illustration of the simplified version of the GRS structure with peephole connections. The corresponding equations are Eq. 5.21 (1), Eq. 5.15 (2), Eq. 5.16 (3), Eq. 5.22a (4), Eq. 5.22b (5), and Eq. 5.19 (6).

To further increase the network capability, a peephole version of the GRS is designed, which is shown in an illustration in Fig. 5.21. Here, it uses not only the normal memory but also the cell memory to calculate each gate. The stop-gradient on input is no longer needed as the isolation is broken with a peephole connection. The new cell memory is calculated simply as

$$c_t = \phi(W_c * I_t + U_c * c_{t-1} + b_c). \tag{5.21}$$

The two gates are calculated as

$$f_t = \sigma(W_f * I_t + U_f * \tilde{h}_t + V_f * c_t + b_f), \tag{5.22a}$$
$$i_t = \sigma(W_i * I_t + U_i * \tilde{h}_t + V_i * c_t + b_i). \tag{5.22b}$$

In this design, the gates can also be affected by cell memory. In the case of unreasonable movements or large noise, the data flow over the gates can be re-balanced.

5.4.3.3. Simplified GRS with Cell Memory on Gate Control

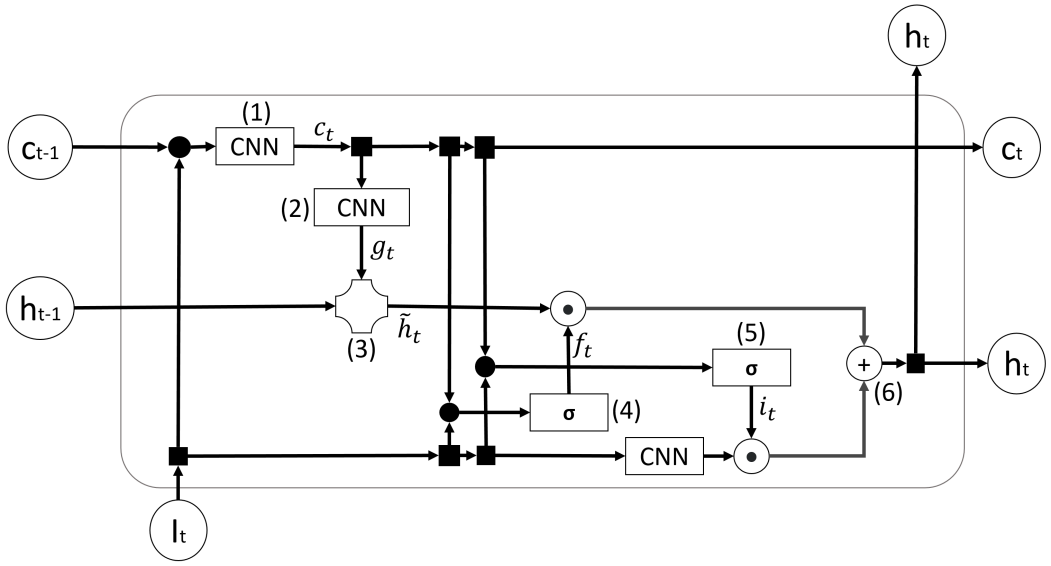

Fig. 5.22.: An illustration of the second simplified version of the GRS structure. The corresponding equations are Eq. 5.20 (1), Eq. 5.15 (2), Eq. 5.16 (3), Eq. 5.23a (4), Eq. 5.23b (5), and Eq. 5.19 (6).

Then, to further modify the structure of GRS under several different considerations, a second implementation is using cell memory to calculate the gates. An illustration is shown in Fig. 5.22.

The gates are calculated as

$$f_t = \sigma(W_f * I_t + U_f * c_t + b_f), \tag{5.23a}$$

$$i_t = \sigma(W_i * I_t + U_i * c_t + b_i). \tag{5.23b}$$

This change makes the balancing of the data flow focused on the movement instead of main stream features.

5.4.3.4. Simplified GRS with Short-Term Cell Memory on Gate Control

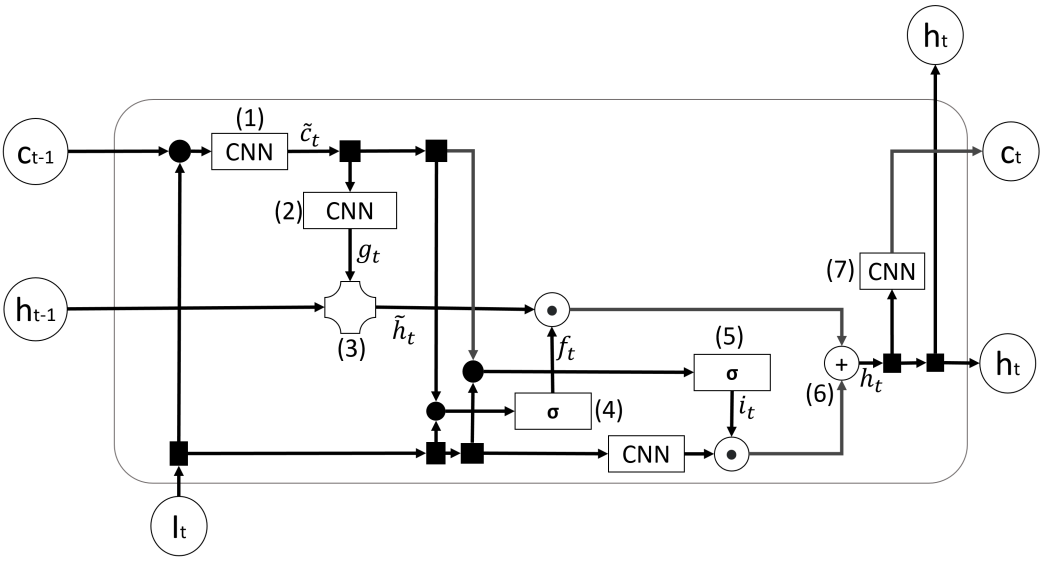

Fig. 5.23.: An illustration of the third simplified version of the GRS structure. The corresponding equations are Eq. 5.24a (1), Eq. 5.24b (2), Eq. 5.24c (3), Eq. 5.24d (4), Eq. 5.24e (5), Eq. 5.24f (6), and Eq. 5.24g (7).

The third implementation is a short-term memory design where the cell memory comes from the main stream features. An illustration is shown in Fig. 5.23. The structure is based on the second version in Sec. 5.4.3.3, where the gates are controlled by cell memory. Instead of directly passing the isolated cell memory, the memory is calculated from the hidden state. The components of this version are as

$$\tilde{c}_t = \phi(W_c * I_t + U_c * c_{t-1} + b_c), \tag{5.24a}$$

$$g_t = \theta(W_g^{\times 2} * \tilde{c}_t + b_g^{\times 2}), \tag{5.24b}$$

$$\tilde{h}_t = Sampler(h_{t-1}, g_t), \tag{5.24c}$$

$$f_t = \sigma(W_f * I_t + U_f * \tilde{c}_t + b_f), \tag{5.24d}$$

$$i_t = \sigma(W_i * I_t + U_i * \tilde{c}_t + b_i), \tag{5.24e}$$

$$h_t = f_t \odot \tilde{h}_t + i_t \odot \phi(W_e * I_t + b_e), \tag{5.24f}$$

$$c_t = \phi(W_h * h_t + b_h). \tag{5.24g}$$

In this version, the cell memory is only short-term memory that is mainly based on current changes. This should be an advantage in very dynamic scenes but a disadvantage in long-term stable movements.

5.4.3.5. Simplified GRS with Past Memory on Gate Control

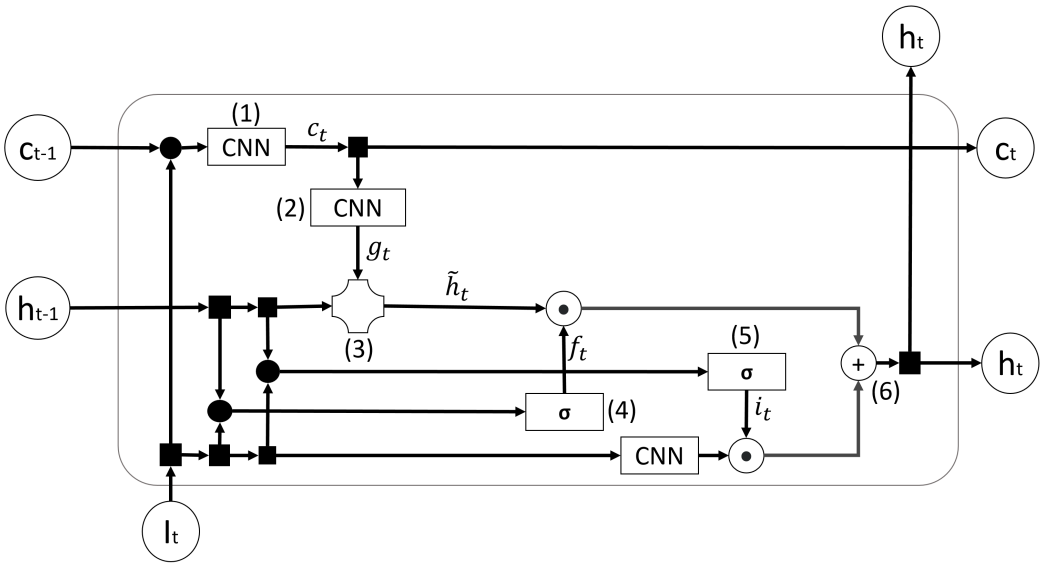

Fig. 5.24.: An illustration of the fourth simplified version of the GRS structure. The corresponding equations are Eq. 5.20 (1), Eq. 5.15 (2), Eq. 5.16 (3), Eq. 5.25a (4), Eq. 5.25b (5), and Eq. 5.19 (6).

Instead of using the re-sampled memory, a fourth version is designed that uses the hidden state directly from the last frame to calculate the gates. Fig. 5.24 shows an illustration of this version. The major concept is based on the first version, but the gates are calculated by

$$f_t = \sigma(W_f * I_t + U_f * h_{t-1} + b_f), \tag{5.25a}$$

$$i_t = \sigma(W_i * I_t + U_i * h_{t-1} + b_i). \tag{5.25b}$$

Thus, the memory will directly affect the data flow balancing of the current frame. In this way, it reduces the dynamic errors due to sampling when balancing the data flow.

5.4.3.6. Simplified GRS with Past Cell Memory on Gate Control

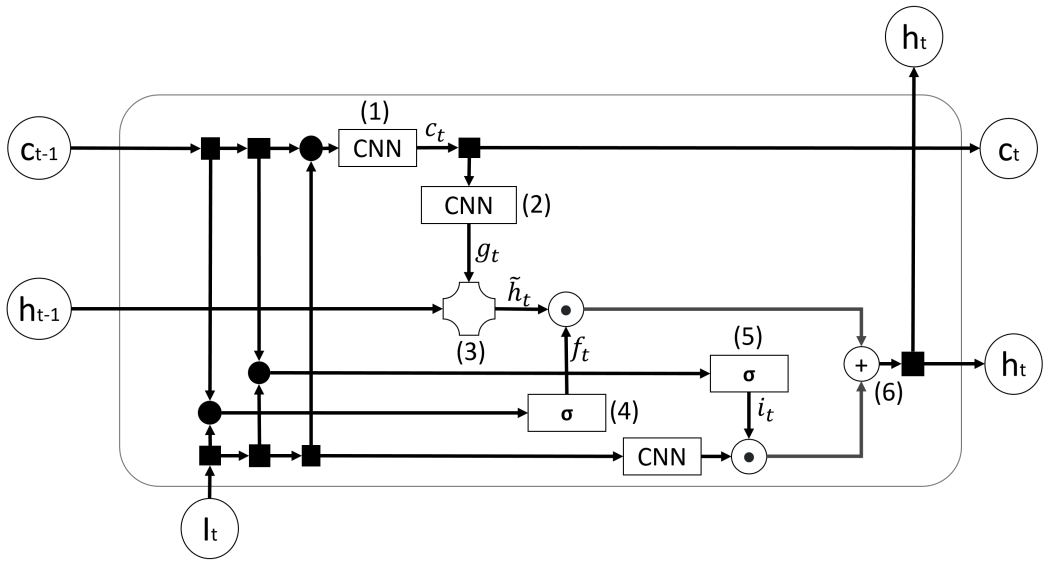

Fig. 5.25.: An illustration of the fifth simplified version of the GRS structure. The corresponding equations are Eq. 5.20 (1), Eq. 5.15 (2), Eq. 5.16 (3), Eq. 5.26a (4), Eq. 5.26b (5), and Eq. 5.19 (6).

Based on a similar concept as in Sec. 5.4.3.5, the fifth version is proposed that uses the cell memory from the last to directly calculate the gates. Fig. 5.25 shows an illustration of this. The gates are then calculated via

$$f_t = \sigma(W_f * I_t + U_f * c_{t-1} + b_f), \tag{5.26a}$$

$$i_t = \sigma(W_i * I_t + U_i * c_{t-1} + b_i), \tag{5.26b}$$

where the past movement memory will affect the data flow. In this way, the long-term stable movements are focused on and preferred in this version.

5.4.3.7. Simplified GRS with Past Short-Term Cell Memory on Gate Control

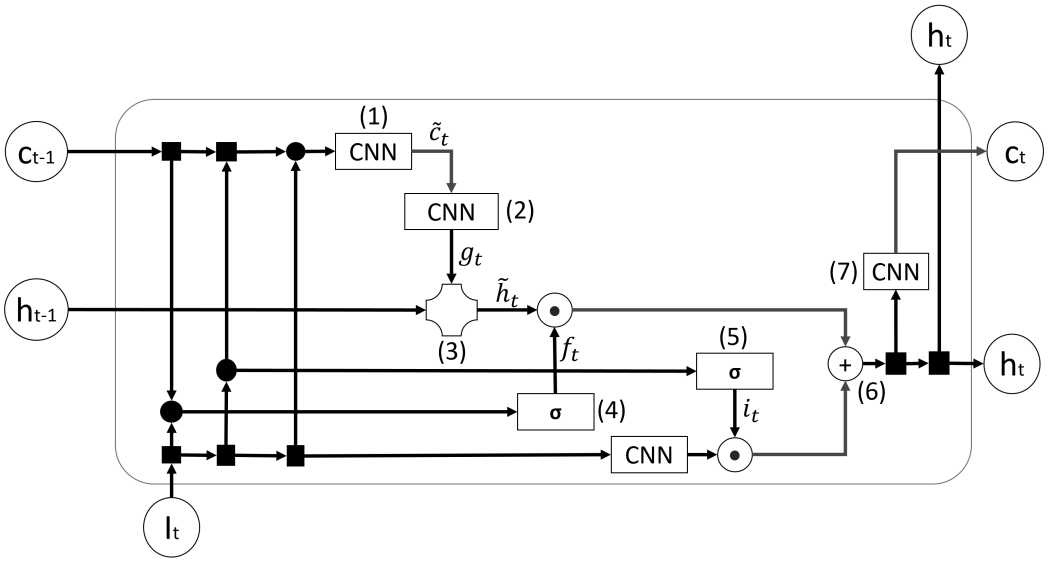

Fig. 5.26.: An illustration of the sixth simplified version of the GRS structure. The corresponding equations are Eq. 5.27a (1), Eq. 5.27b (2), Eq. 5.27c (3), Eq. 5.27d (4), Eq. 5.27e (5), Eq. 5.27f (6), and Eq. 5.27g (7).

Finally, a short-term memory version is introduced that also uses the cell memory from the last frame to calculate the cell memory. An illustration of this sixth version is shown in Fig. 5.26. The components are defined as

$$\tilde{c}_t = \phi(W_c * I_t + U_c * c_{t-1} + b_c), \tag{5.27a}$$
$$g_t = \theta(W_g^{\times 2} * \tilde{c}_t + b_g^{\times 2}), \tag{5.27b}$$
$$\tilde{h}_t = Sampler(h_{t-1}, g_t), \tag{5.27c}$$
$$f_t = \sigma(W_f * I_t + U_f * c_{t-1} + b_f), \tag{5.27d}$$
$$i_t = \sigma(W_i * I_t + U_i * c_{t-1} + b_i), \tag{5.27e}$$
$$h_t = f_t \odot \tilde{h}_t + i_t \odot \phi(W_e * I_t + b_e), \tag{5.27f}$$
$$c_t = \phi(W_h * h_t + b_h). \tag{5.27g}$$

This version is mainly focused on short-term memory and the main stream features. The movement features are kept implicitly within the main stream.

5.4.3.8. GRU Style GRS

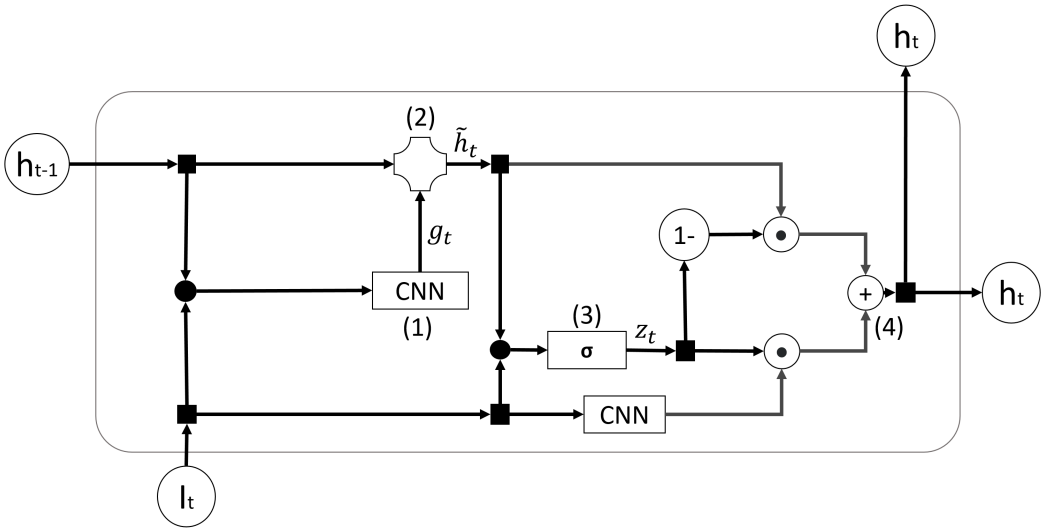

Fig. 5.27.: An illustration of the first version of the GRS structure under GRU style. The corresponding equations are Eq. 5.28 (1), Eq. 5.29 (2), Eq. 5.30 (3), and Eq. 5.31 (4).

To further reduce the computational complexity, two versions of GRS inspired by the use of an update gate in GRU (Eq. 5.11 and Eq. 5.13) are proposed. These two versions do not isolate the cell memory, but use the hidden state to calculate the movement sampling grid. The first version is shown in Fig. 5.27.

The first component is still used to calculate the sampling grid as

$$g_t = \theta(W_g^{\times 2} * I_t + U_g^{\times 2} * h_{t-1} + b_g^{\times 2}), \tag{5.28}$$

where the grid is calculated directly by the current input and the hidden state from the last frame. Then, the hidden state is resampled by this grid via

$$\tilde{h}_t = Sampler(h_{t-1}, g_t). \tag{5.29}$$

After resampling, the hidden state should be refined to the current object locations based on the movements.

The third component is the update gate, which is calculated by

$$z_t = \sigma(W_z * I_t + U_z * \tilde{h}_t + b_z). \tag{5.30}$$

This gate will balance the use of the hidden state and the current input. The output of the balanced features is calculated by

$$h_t = (1 - z_t) \odot \tilde{h}_t + z_t \odot \phi(W_e * I_t + b_e). \tag{5.31}$$

The balancing concept is inspired by the use of an update gate in GRS. Since the gate z_t is in a value range of (0,1) after sigmoid activation, the new hidden state and output should be a weighted balancing between the resampled hidden state and the processed input.

5.4.3.9. GRU Style GRS with Past Memory on Gate Control

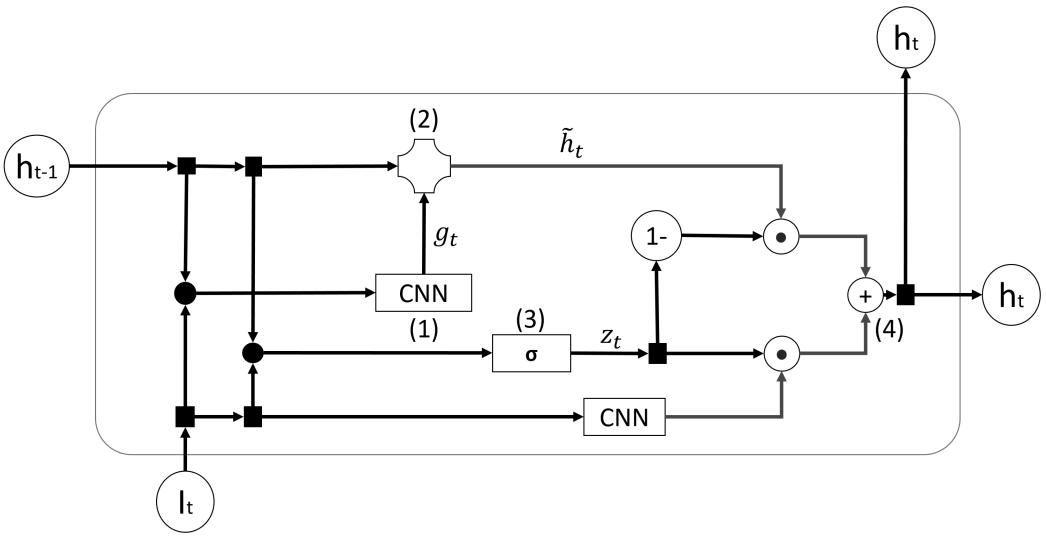

Fig. 5.28.: An illustration of the second version of the GRS structure under GRU style. The corresponding equations are Eq. 5.28 (1), Eq. 5.29 (2), Eq. 5.32 (3), and Eq. 5.31 (4).

Similar to the fourth design of GRS in Sec. 5.4.3.5, a second version of GRU style GRS is designed, which directly uses the hidden state from the last frame to calculate the update gate. Fig. 5.28 shows an illustration of this version. The update gate in this version is calculated as

$$z_t = \sigma(W_z * I_t + U_z * h_{t-1} + b_z), \tag{5.32}$$

where the movement is focused on the feature from the last frame. This change should make the unit focus more on long-term stable movements.

5.4.4 Behavior Study

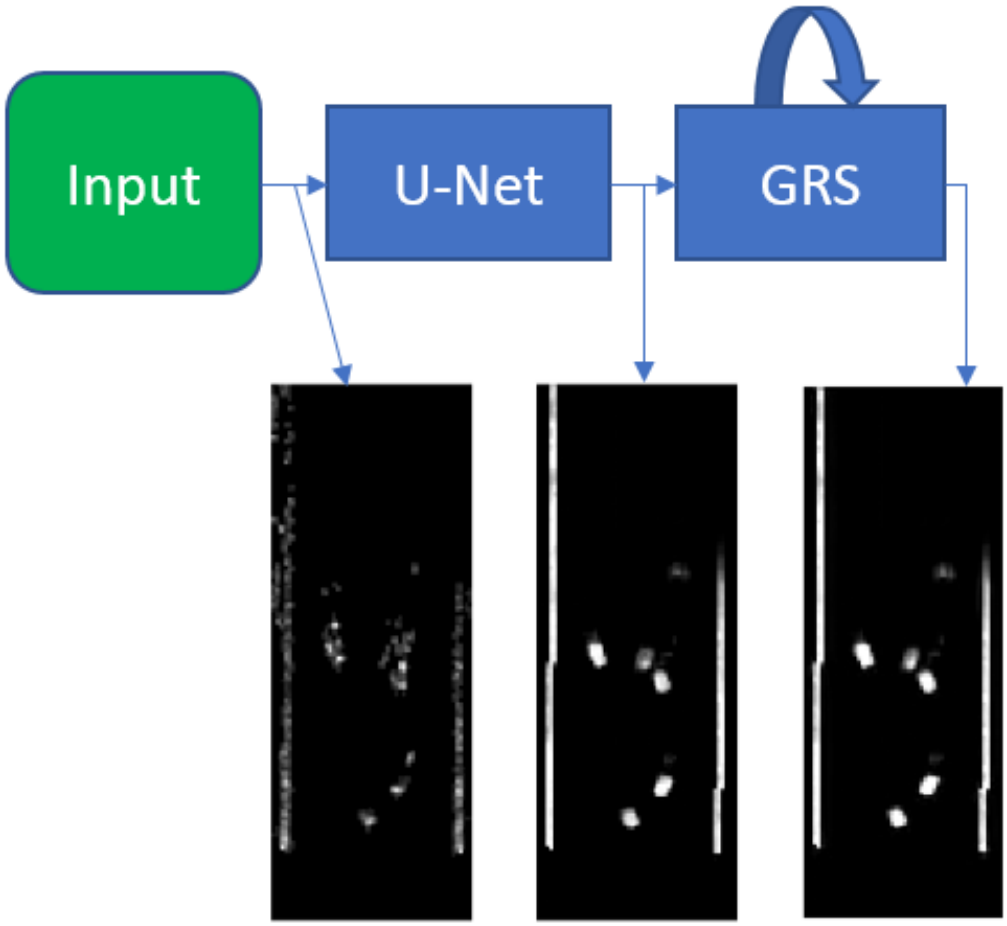

Fig. 5.29.: The network structure applying GRS on the magnitude map embedding.

To understand the behavior of GRS, a network taking the feature embedding from Sec. 5.2.1.1 and performing temporal fusion via GRS is tested. The network structure is shown in Fig. 5.29. The temporal fusion via GRS is attached after the feature is embedded via U-Net. The segmentation output should be further refined during temporal fusion.

Fig. 5.30 presents the network input signal, the segmentation embedding, and the refined prediction after temporal fusion. These three frames come from the same sequence, and the ground-truth is also shown as a reference. In the first frame, since there is no past memory, the output after temporal fusion is almost identical to the embedding output. However, after a few more frames, the object segmentation after temporal fusion has a much better boundary and higher confidence. Moreover, the noise prediction in the first frame is further reduced in the fifth frame after the GRS.

In the tenth frame, even though the embedding output has very low confidence on one object (at center), the GRS is still able to keep the object with relatively high confidence at the correct location.

To understand how the sampling method is performing in GRS, Fig. 5.31 shows an example of a sampling grid inside GRS overlaid on the predicted segmentation. When looking into the details on the yellow circled two objects, the sampling grid tends to move (red arrows in the figure) the feature vectors of the boundary toward the new location. Interestingly, the center area is not moved greatly, as the center pixels of the object do not change much with respect to the object segmentation.

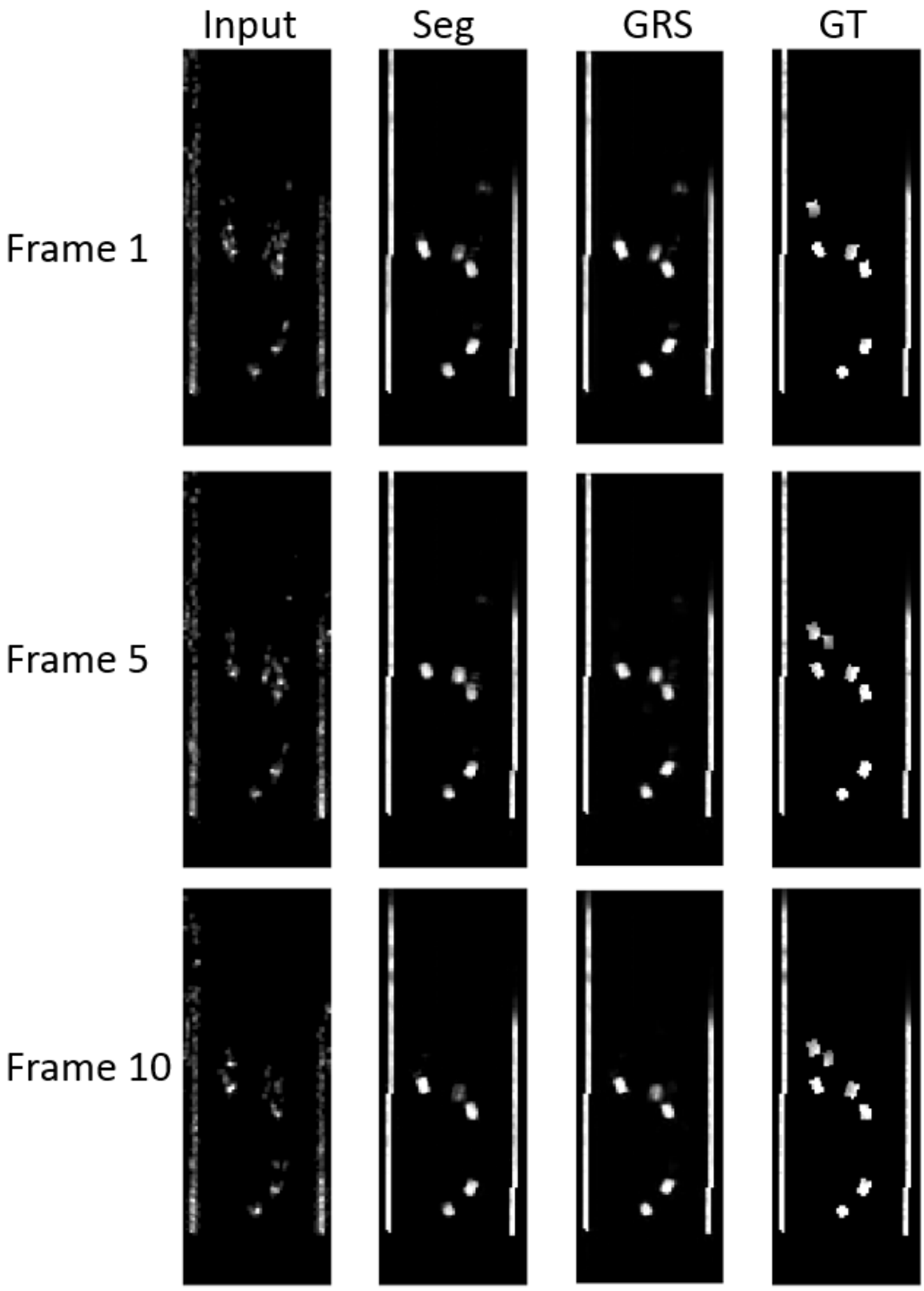

Fig. 5.30.: The network input and outputs of three frames over time in the same sequence. "Seg" is the single-frame segmentation output from U-Net. "GRS" is the temporal-fused segmentation output after applying GRS. "GT" is the ground-truth segmentation.

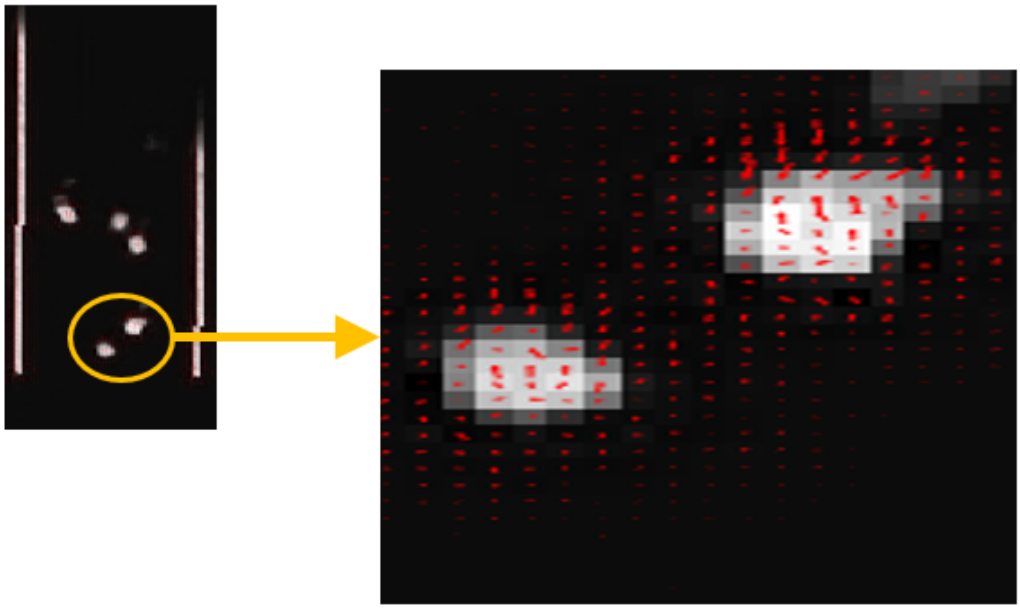

Fig. 5.31.: The sampling grid visualization of GRS on magnitude map data.

5.5 A Deep Neural Network Radar Perception System

In the previous sections, they have shown several designs and proposals on using deep neural networks to extract features from radar signals and applying temporal fusion with regard to the automotive use cases. In the end, a radar perception system is proposed that can perform perception tasks for automotive needs.

This section first presents trials on object detection using neural networks and post-processing techniques. Then, it briefly introduces a few previous research works on neural network object detectors. Lastly, it discusses the design of a combined neural network for object detection with input of radar signals.

5.5.1 Object Detection by Segmentation

On the simulated magnitude map radar data, networks performing object segmentation tasks were used. Sec. 5.2.2 presents a U-Net structure that can generate good object segmentation results on this data, and Sec. 5.4.4 shows that temporal fusion can further improve this result.

The question is how to apply this segmentation result in automotive use cases. In general, automotive applications require objects instead of pixel-wise segmentation maps to identify and predict objects and movements in a local environment. Tracking an object using the Kalman filter can be achieved with motion models, while tracking objects in a segmentation map is much more complex.

For these reasons, a post-processing method is proposed to generate object detection bounding boxes based on an object segmentation map. Once the network has predicted the segmentation result, it applies the DBSCAN algorithm to the segmentation image to cluster pixels belonging to the same objects. After clustering, it calculates a minimal rectangle-shaped convex hull for each cluster to get a rectangle bounding box for each object.

Fig. 5.32 shows one example of this post-processing algorithm. The colors indicate clusters after the DBSCAN algorithm. The cyan-colored guardrails are not considered objects, as the needle-like shapes are unreasonable for vehicles. This algorithm can obtain a list of bounding boxes out of the segmentation result. Then, the Kalman filter can easily be applied to track each object in the scenario.

Due to the limitation of the clustering algorithm, Fig. 5.33 shows a failure example of this post-processing algorithm. The red cluster should contain two objects, but

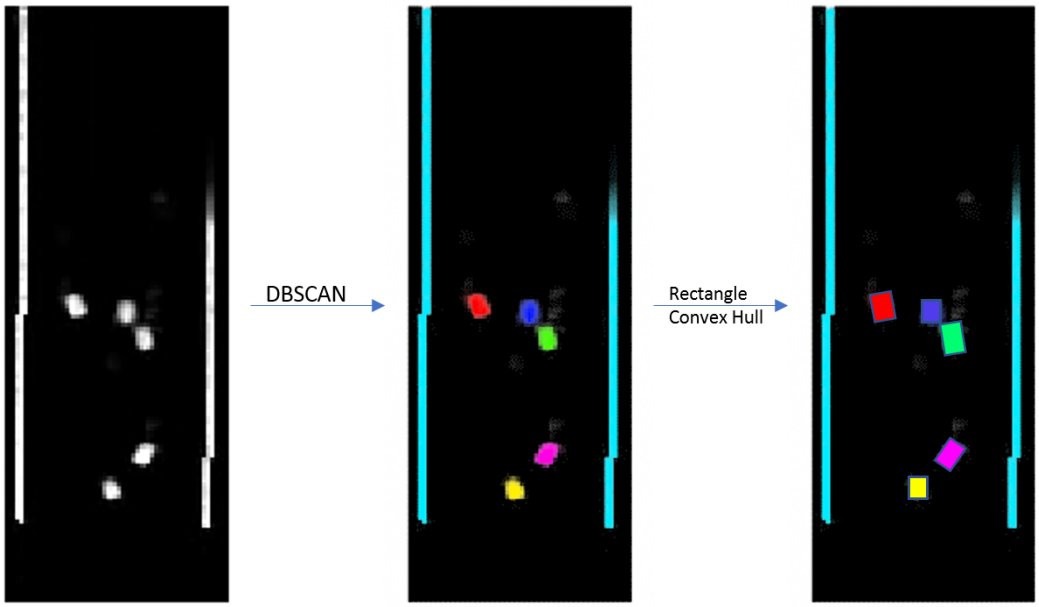

Fig. 5.32.: An example of DBSCAN clustering results (in color) and bounding boxes for each object.

these two objects are very near in distance. The gap between these two objects is not clear enough in the segmentation map. In this case, the DBSCAN will cluster the pixels of these two objects as one single cluster; hence, it will generate only one bounding box after convex hull.

However, in most cases, this post-processing algorithm is good at identifying object bounding boxes in a segmentation map. Due to the limitation of the clustering algorithm, there still exist several failure cases, especially when objects are too close; hence, they are considered connected in the DBSCAN algorithm.

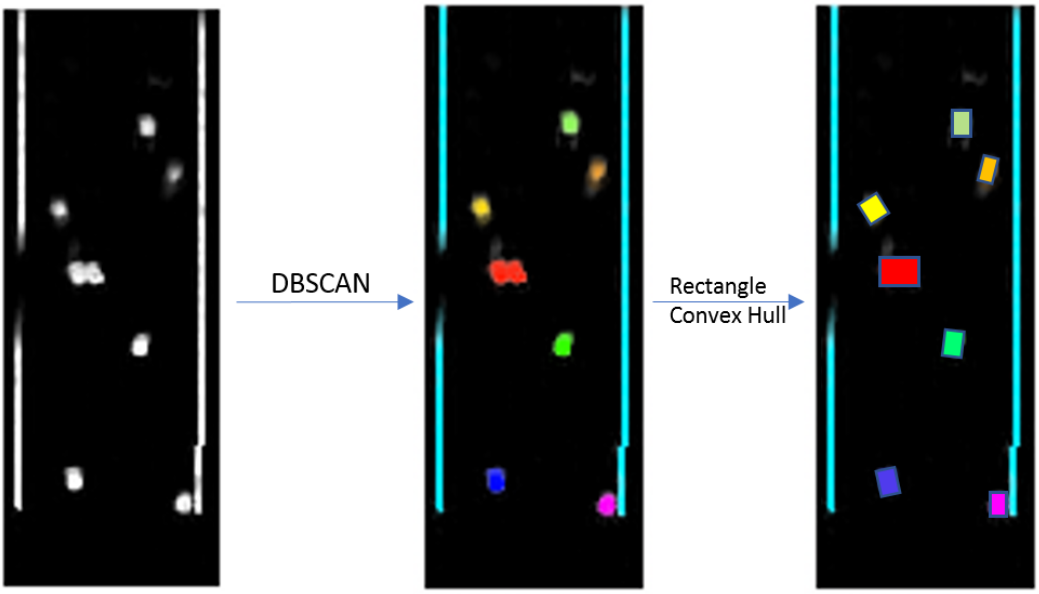

Fig. 5.33.: A failure example of the DBSCAN clustering result (in color) and bounding boxes for each object. The red object is a failure case in this scenario.

5.5.2 State-of-the-Art Deep Object Detectors

In the object detection research field on image data, many researchers have contributed multiple good neural network designs for object detectors. They are shown to be very effective and robust in object detection tasks on image data.

Among all different network detectors, there are two main concepts in the design: one-shot detectors and two-shot detectors. One-shot detectors are designed to work in an end-to-end fashion, where the network will directly output a list of bounding boxes and their classification as detection result with input of image. Two-shot detectors have two stages: the detection phase and the object phase. In the detection phase, the network will only output the estimated area of the object as proposals. In the object phase, the proposed areas will be further processed by the network to generate bounding box and object classification results.

5.5.2.1. One-Shot Detectors

The one-shot detectors are based on the concept of directly assuming and predicting objects in each segment of an image. The objects predicted in each segment are then post-processed based on the network confidence output to clean up the unwanted predictions.

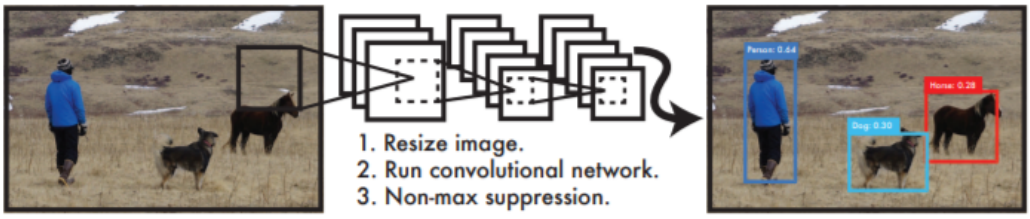

Fig. 5.34.: An illustration of the basic concept of the you only look once (YOLO) detector [Red+16].

YOLO is one of the one-shot detector networks [Red+16]. Fig. 5.34 shows the basic concept in its design. It uses one CNN network to extract and embed features. The embedded features are in a smaller grid after downsampling layers. Each cell in this grid is considered a segment in the image. If a segment contains a part of a real object, this segment is assigned object properties, such as object class and bounding box definitions. During training, the network should train to predict the bounding box and the object class for each segment relevant to the target object. In the inference phase, the bounding boxes are filtered by confidence thresholds and combined by a non-maximal suppression (NMS) algorithm.

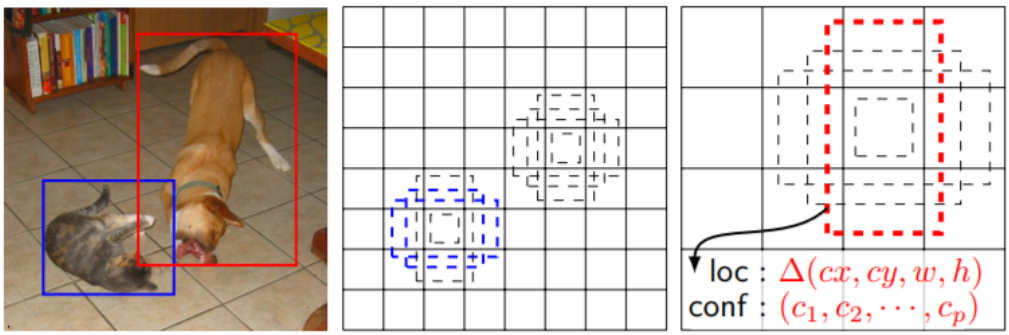

Fig. 5.35.: An illustration of the basic concept of the single shot multibox detector (SSD) [Liu+16].

SSD is another one-shot detector network under a different design [Liu+16]. They introduce anchors as templates for object proposals. The network should predict an object on each anchor and define its bounding box. Instead of one single size, the SSD predicts objects over different sizes of images; hence, it can support various objects using different sizes of the anchors. Fig.5.35 shows a basic concept of SSD design. (a) is the input image and the ground truth bounding boxes of the two objects. (b) shows the anchor defined on the center pixel after downsampling to the blue object, and (c) shows the anchor with different downsampling level matches

the red object. Not all anchors at the same center are assigned to the same object, so that the objects overlapping or nearby can be detected via different anchors at the same location. Moreover, the anchors have different aspect ratios to match the object bounding-box shapes.

Similar to YOLO, the bounding boxes predicted by the network will be filtered by confidence thresholds and the NMS algorithm to resolve issues when an object is detected by multiple bounding boxes.

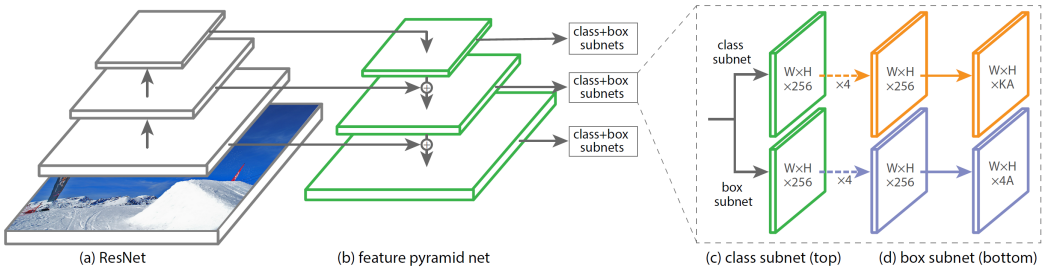

Fig. 5.36.: An illustration of the design of the RetinaNet detector [Lin+17].

At a later time, RetinaNet was proposed with several improvements on the SSD detector [Lin+17]. First, it uses pyramid feature extraction (like U-Net design) instead of downsampling-only extraction. In this way, the lower pyramid levels can obtain richer semantic information from higher levels. Also, the authors proposed a new loss function, focal loss, to reduce the imbalance issue between the number of object anchors and the background anchors. Fig. 5.36 shows the design of the RetinaNet network. The object detection head of this network is very much similar to SSD, where the network predicts object class and bounding box regression on anchors of different pyramid levels. Hence, the network can show good performance on both large and small objects in the image.

5.5.2.2. Two-Shot Detectors

Unlike one-shot detectors, two-shot detectors are based on the concept of running object classification and bounding box estimation only on proposed areas of an image. Fast RCNN is one of the two-shot detectors [Gir15]. Instead of only an image, the network requires an image and a list of object proposals as input. The network does feature extraction and also re-samples the features based on the proposed regions. Fig. 5.37 shows the network structure of their proposal. The network uses a special ROI-pooling layer to resample the variant sizes of object regions into a fixed size of feature map; then, the classification and bounding box regression will be performed on each fixed feature map of the proposed ROIs.

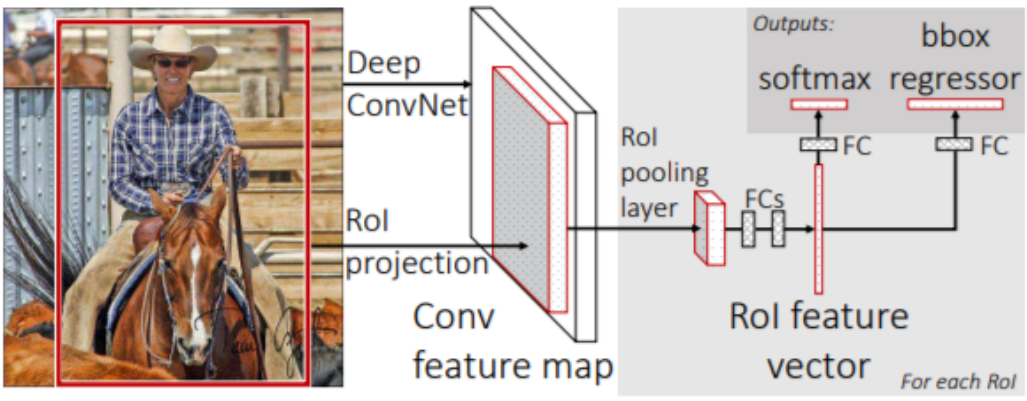

Fig. 5.37.: An illustration of the structure of a Fast RCNN detector [Gir15].

To use this network, object proposals need to be done using another method. Due to this limitation, the performance is also affected by the proposal method, and the proposed method cannot be refined during the training of Fast RCNN.

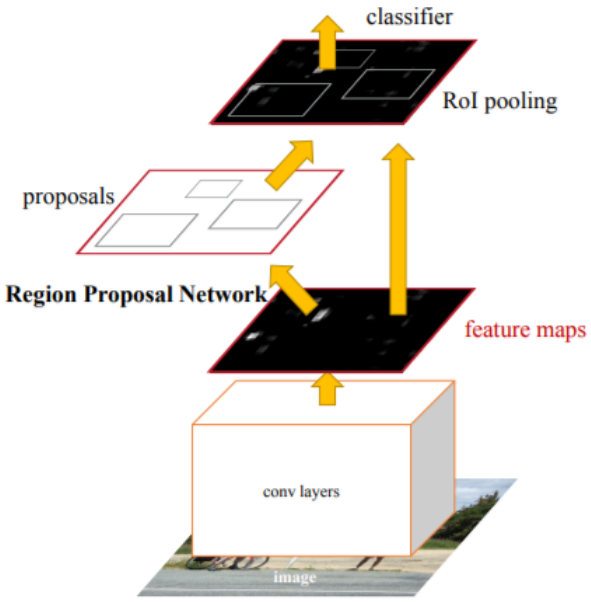

Fig. 5.38.: An illustration of the structure of a Faster RCNN detector [Ren+15].

To further solve the limitations and issues of Fast RCNN, another two-shot detector called Faster RCNN was proposed as an improved version of Fast RCNN [Ren+15]. The object proposals are performed by a region proposal network in Faster RCNN. It no longer requires a separate object proposal method. Moreover, the authors also introduce anchors into their object detection heads. The network structure is presented in Fig. 5.38. With these improvements, although it is still under the design

of a two-shot, the network can be trained in an end-to-end fashion and have a much faster runtime than Fast RCNN.

5.5.3 Deep Radar Object Detector

To get better performance on object detection using radar signals as input, a combined network is designed with several components that can benefit object detection. Sec. 5.2.2.2 presents a neural network structure for gridized point cloud encoding. Sec. 5.4 introduces a new sampling recurrent network unit GRS. Sec. 5.5.2 recaps a few state-of-the-art designs of neural network object detectors.

This section will show a proposed network with a combination of the introduced components that can show good object detection performance on radar signals. Part of the content in this section is based on our previous publications [Su+21a][Su+21b][Su+21c].

5.5.3.1. Modified RetinaNet for Radar

The designed object detector is intended to detect moving and stationary vehicles in the local environment of the ego vehicle. Fig. 5.39 shows the structure of the proposed radar object detector network. The input radar detections are first processed and encoded by the gridized feature encoding network. After this encoding, the scenario is presented in a high-dimension 2D feature map. The encoding network has a similar structure as in Fig. 5.12. It increases the number of neurons in each fully connected layer by 16, 32 and 64. After processing, it replaces the concatenation layer in the encoding network with a max-pooling layer over each point feature vector, resulting in a 2D feature map with 64 feature channels.

Then, a pyramid feature extraction network (inspired by U-Net and RetinaNet) is applied. To obtain better performance via temporal fusion, this network applies GRS in each feature pyramid level during downsampling (purple layers in the figure). To balance the network complexity and the task, the GRS with the GRU style of the first version (Sec. 5.4.3.8) is used. This version of GRS has less computational complexity but can still maintain good performance on movement catching. Moreover, GRS is utilized on each pyramid level under the consideration that object moving can be different on each downsampled level.

In the end, it uses a similar design of anchor-based object detection head as RetinaNet. But, instead of running this head on all five pyramid levels, it runs only on

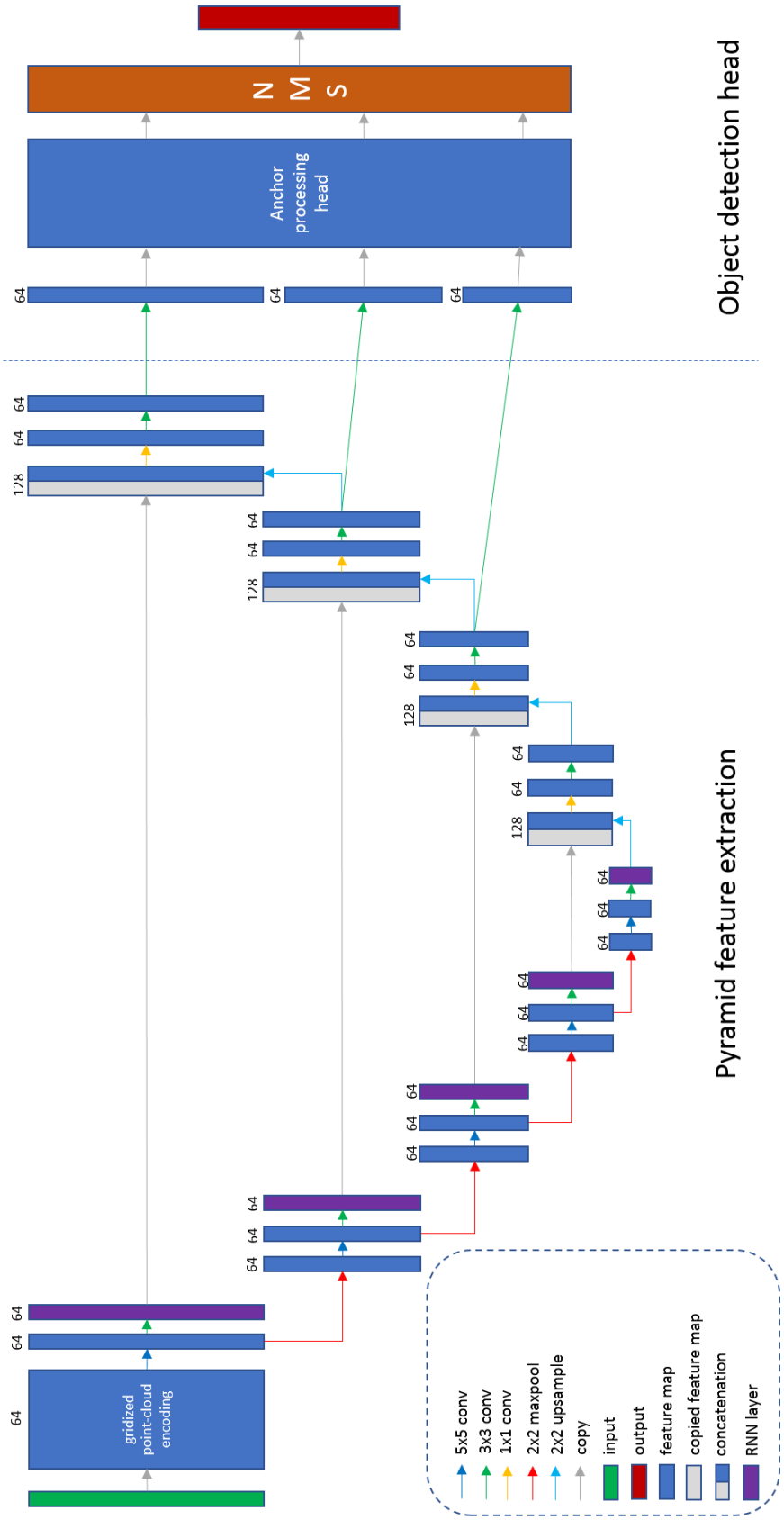

Fig. 5.39.: An illustration of network structure of proposed radar object detector.

the first three low levels because the grid has a defined cell size of 0.5m $\times$ 0.5m. After two downsampling steps, each cell is already 2m $\times$ 2m large. With respect to an ordinary vehicle size of 4m $\times$ 2m, the cell at this level is already large enough to represent a car.

With respect to these sizes, it is not necessary to use anchors in various sizes; they can be limited to a list of sizes matching real-world objects. For example, on the third level, the anchors can be relatively small. Anchors with 10 pixels are not needed because that would mean 20m in the real world.

In the last step, it applies a post-processing NMS on the list of detected object bounding boxes to remove the duplicated prediction from different anchors or different pyramid levels. Because objects hardly overlap in the bird's-eye view; hence, the network can further reduce the anchors to a much smaller amount (one or two anchors with different aspect ratios). With all of these modifications, it can reduce unnecessary computational costs in the object detection head.

5.5.3.2. Experiment Results

The network is trained and evaluated on real-world data with manual annotations. The performance is evaluated by F1 score, which is defined as

$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{tp}}{\mathrm{tp} + \frac{1}{2}(\mathrm{fp} + \mathrm{fn})}. \quad (5.33)$$

The true-positives (tp), false-positives (fp), and false-negatives (fn) are calculated by matching the predicted bounding boxes and the annotated bounding boxes with respect to their object classes and intersection-over-union (IoU).

Network	Class	Precision	Recall	F1 Score
Single Frame	Moving	0.4538	0.2805	0.3467
	Stationary	0.3762	0.1018	0.1603
Conv-LSTM	Moving	0.4632	**0.4471**	0.4550
	Stationary	0.3702	**0.2501**	**0.2985**
GRS	Moving	**0.6790**	0.4012	**0.5044**
	Stationary	**0.4415**	0.1948	0.2703

Tab. 5.1.: Performance of radar object detector and baselines on the evaluation set.

Tab. 5.1 shows the performance of object detector networks on the evaluation set. The network "Single Frame" replaces the RNN layers in the network with

convolutional layers. Hence, the network does not have any temporal fusion. The "Conv-LSTM" network uses convolutional LSTM as all RNN layers. The "GRS" network uses the first version of GRS with GRU style (Sec. 5.4.3.8) as RNN layers.

The performance on moving cars and stationary cars are reported. On moving cars, the single-frame network has a good performance comparable to that of the network with temporal fusion. This is mainly because the network can extract Doppler velocity features from the radar detections, which is a very important indicator of a moving car in the scenario. In this case, even without temporal fusion, the network is still able to identify and separate moving cars from the background. However, on stationary cars, the single-frame network no longer performs well due to the limited information of stationary cars from very sparse radar detections. Only with temporal fusion, where the network is able to collect enough features to identify the shapes of stationary cars, can the network detect stationary objects.

With respect to the sampling method, the network can benefit from GRS, which improves the capability of temporal fusion with movement features. The network shows very good precision on moving cars with a similar recall rate as the convolutional LSTM network. However, on stationary cars, the network has slightly worse performance than the convolutional LSTM. The reason is that the movement features of stationary objects are very limited. Since the GRS is trained to take advantage of movement features through learning on moving cars, the network tends to focus more on movements, which are hardly recognizable on stationary cars. Still, with the gates inside the GRS, the network is able to recover good attention on features without movements.

Overall, a deep neural network is proposed that can deal with pure radar detections and perform object detection tasks. The network is trained in an end-to-end fashion and shows good performance on the evaluation set.

6 Conclusion

This dissertation has presented several contributions and designs applying neural networks to various automotive perception tasks. Automotive use cases are of high relevance to academic research in CV, DL, and signal processing. Their contributions include the proposal of quite a few methods and the transfer of knowledge from other research fields to automotive perception tasks.

The Chap. 3 discussed the possibility and benefit of applying sampling methods in neural networks. The experimental results show that sampling can improve network generalization capability by refining the features to a similar data distribution on trained data. More than that, it has also shown a lot of potential with sampling methods in general neural network designs. These contributions also inspire the following and further research.

The Chap. 4 presents the difficulties and challenges faced when starting to work on radar data. The unavailability of publicly accessible datasets is a huge blocker in the research, especially when starting with deep learning. It took quite some effort to obtain annotated data that could be used for the research. This has changed quickly. For those beginning research now, many public datasets with radar signals are available online [Cae+20][Oua+21][Sch+21]. Seeing the challenges of lacking data, researchers who have just started their research can largely benefit from these datasets.

The Chap. 5 introduced several methodologies and techniques for processing radar signals by neural networks. Moreover, a novel recurrent unit design with sampling methods is proposed. In the end, the proposals are combined with one of the state-of-the-art object detector networks. The network with a few modifications for radar signals and automotive perception tasks has shown good performance results.

These findings are not the end of the research. A few further publications have started to dig deeper into the lower-level signals (e.g., CDC data) from radar measurements [Zhu+21a][Zhu+21b][Zhu+21c]. It is hoped that with the following and future research and contributions, the radar perception system can be further improved using neural networks. Ultimately, perception needs in AD can be fulfilled by radar-only sensor settings.

Appendix A

A.1 Computer Vision Tasks

This section presents details of each CV task mentioned in Sec. 2.2.2.

A.1.1 Image Classification

Image classification is the task of assigning a label from a fixed set of categories to an image, normally to the major object in the image. One major challenge in this task is how to efficiently extract features from an image represented in a large dense matrix. Image feature embedding techniques are normally used to compress the key information into a feature vector that is much smaller than the raw image matrix and also invariant to several sub-differences across images of the same category.

Image classification tasks normally have images that contain one major object as input. This major object defines the classification label of this image. For example, for an image of a dog lying on the road, the label would be "dog." The algorithm should be able to see this image and output "dog" as its label prediction. The image may be given in dynamic or fixed size, normally with grayscale or RGB colors. The possible classes are pre-defined and should have enough examples for each class in a dataset. The algorithm is correct when the predicted label matches the annotated label; otherwise, it is incorrect.

One of the famous datasets is the MNIST database [LC10]. The dataset contains a training set of 60,000 examples and a test set of 10,000 examples. Each example is a cropped and normalized image of a handwritten digit from 0 to 9. Each example comes with a label of the digit it presents. Fig. A.1 shows a few examples in this dataset. All examples in MNIST are in grayscale and have a black background.

A more practical but complex dataset for digit classification is SVHN [Net+11]. This dataset provides 73,257 examples for training, 26,032 examples for testing, and 531,131 additional examples as easy training data. Fig. A.2 shows a few examples of this dataset. Although all digits are cropped and scaled to the same size, it is much more complex than the MNIST dataset. It contains colored images and background

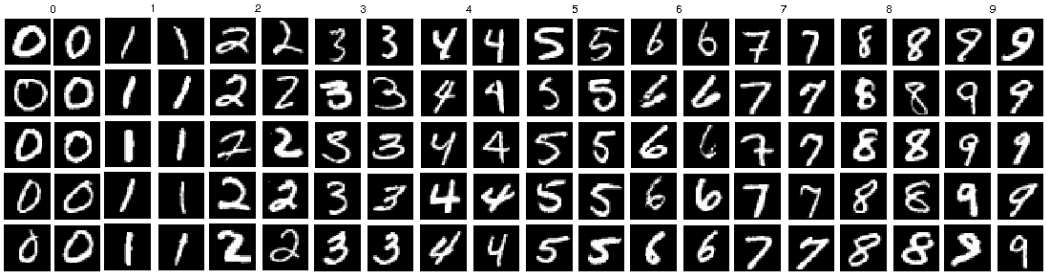

Fig. A.1.: Examples in the MNIST dataset [LYP16].

from real-world objects. This makes it much harder and more complex for the ML algorithms, as the color can be different for the same digit. Moreover, it can contain more than one digit in an image of which the labeled digit is centered. For some localization invariant algorithms, it may be hard to localize the target, which may lead to wrong predictions.

Fig. A.2.: Examples in SVHN dataset.

When talking about image classification in automotive use cases, traffic sign classification is one of the most important tasks. GTSRB provides a dataset with real-world images of different traffic signs together with their labels [Sta+12]. It contains more than 40 classes and more than 50,000 images in total. Fig. A.3 shows a few examples of this dataset. It is obvious that the images are cropped but with different lighting conditions and view angles. It is a very practical task for automotive perception systems.

Fig. A.3.: Examples in the GTSRB dataset.

ImageNet is a widely used image classification dataset [Den+09]. It contains 1,000 object classes within 1,281,167 training images, 50,000 validation images, and 100,000 test images. All images were cropped from real-world photos and labeled by human annotators. Fig. A.4 shows a few examples from this dataset. ImageNet

is widely used in image feature embedding networks and is popular for training pre-trained models for various CV tasks using DL [SZ14].

A.1.2 Image Object Detection

Image OD is the task of detecting objects and localizing them in an image. It is normally combined with an image classification task, as the feature for localization is class specific, and the classification algorithm can help reduce false detections. In common cases, the objects are labeled by bounding boxes, together with their class label. In one image, it can contain multiple objects with the same or different labels. Multi-object detection is more complex than single object detection, though it is more common in practice.

In common image OD tasks, the input is a grayscale or RGB image. The annotations are provided as bounding boxes of each object that cover their presence. The bounding boxes can be given in various forms but are easily converted to a rectangle box with coordinates on the image. The box may be defined as its four corners or its center and its length and width. In more complex cases, a rotated box may be given by providing the rotated angle of the rectangle. The pixels under each box are not necessarily the object underneath if the object is non-rigid or overlapping happens. The box is normally the minimal rectangle box covering all visible pixels of an object in a given image.

The performance is two-fold. One aspect is to consider the quality of bounding boxes predicted by the algorithm when matching them to the annotations. The errors are considered when, for example, the predicted box is slightly smaller than the desired label, or the predicted box is shifted in the center position. Another fold is the quality of classification. In a pure object detection task, the binary classification between object and background is examined. In multi-class detection, the classification performance is similar to image classification but on predicted boxes. The boxes are considered to be matched when the overlap area between the predicted box and annotated box is above a certain threshold. The overlap is normally calculated by the Jaccard index (or IoU) as

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}, \tag{A.1}$$

where the sets are defined by the pixels underlying box A and box B.

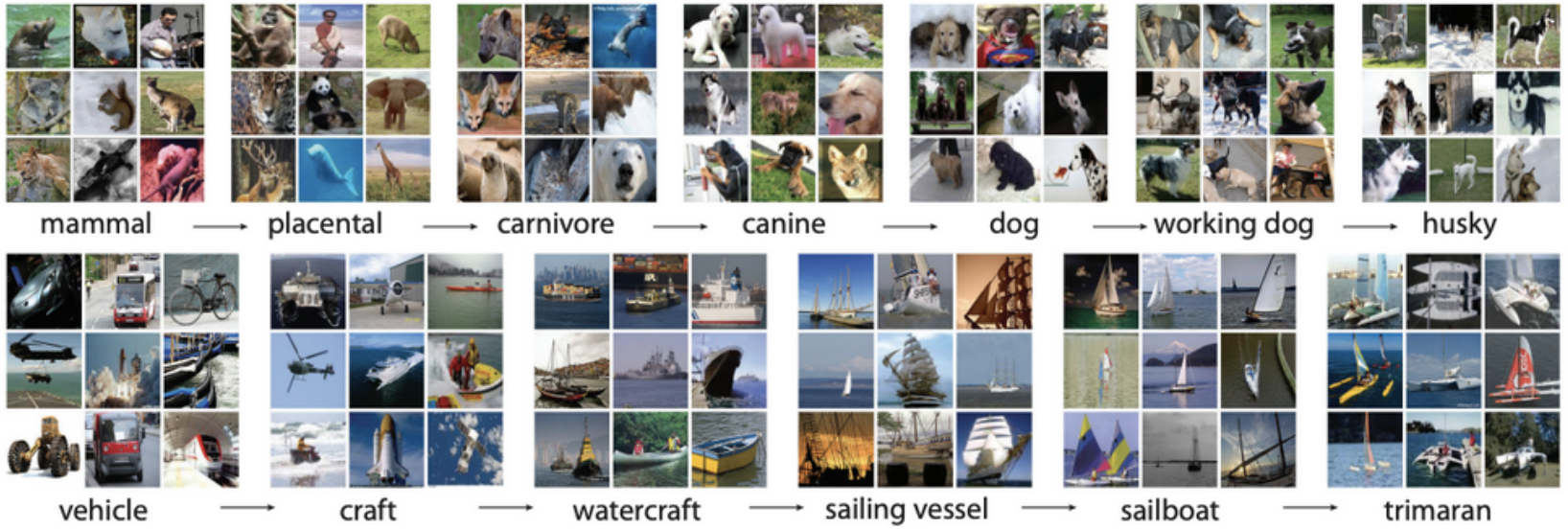

Fig. A.4.: Examples in ImageNet dataset.

In the case of two non-rotated rectangle boxes, given top-left and bottom-right corners of $Box_1 = \{(x^1_{tl}, y^1_{tl}), (x^1_{br}, y^1_{br})\}$ and $Box_2 = \{(x^2_{tl}, y^2_{tl}), (x^2_{br}, y^2_{br})\}$ (minimal coordinates $(0, 0)$ is at the top-left corner of the image), and the IoU can be calculated based on these four corners by

$$
\begin{aligned}
x_{Box_1 \cap Box_2} &= \left(min(x^1_{br}, x^2_{br}) - max(x^1_{tl}, x^2_{tl})\right), \\
y_{Box_1 \cap Box_2} &= \left(min(y^1_{br}, y^2_{br}) - max(y^1_{tl}, y^2_{tl})\right), \\
Area(Box_1) &= (x^1_{br} - x^1_{tl}) * (y^1_{br} - y^1_{tl}), \\
Area(Box_2) &= (x^2_{br} - x^2_{tl}) * (y^2_{br} - y^2_{tl}), \\
Area(Box_1 \cap Box_2) &= x_{Box_1 \cap Box_2} * y_{Box_1 \cap Box_2}, \\
Area(Box_1 \cup Box_2) &= Area(Box_1) + Area(Box_2) - Area(Box_1 \cap Box_2), \\
J(Box_1, Box_2) &= \frac{Area(Box_1 \cap Box_2)}{Area(Box_1 \cup Box_2)}.
\end{aligned}
\tag{A.2}
$$

If the IoU is above a certain threshold, the paired boxes are considered as a match.

Pascal VOC is a popular OD dataset [Eve+10]. It contains 20 classes and has 27,450 annotated objects in 11,530 images in the training set. Pascal VOC is a multi-object detection dataset in which more than one object can be labeled in an image. Fig. A.5 shows a few examples of the images and bounding-box annotations. The class of objects is specified at the top of each column.

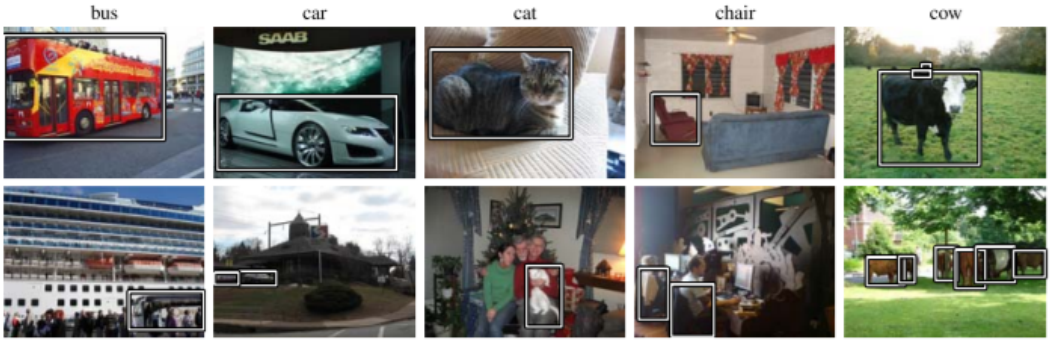

Fig. A.5.: Examples in Pascal VOC dataset.

A.1.3 Image Segmentation

Image segmentation is the task of partitioning an image into different segments. In this manner, it is a task to classify each pixel in an image, other than to classify

the whole image as one label. There are two typical types of segmentation tasks: semantic segmentation and instance segmentation.

Segmentation tasks involve having an image as input and a segmentation map as annotation. The segmentation map can have multiple channels for various semantic categories or object instances. The map normally has the same size as the input image; each pixel in this map is labeled by a desired class. The pixel of the same class (or instance) should show a segmented area on the image that covers the visible object or semantic region of a given class. The segmentation tasks do not need to consider non-rigid object shapes, as they are pixel-wise labels that can have arbitrary presented shapes. The performance is normally calculated pixel-wisely, similar to image classification, but at the pixel level. If a pixel is predicted with the correct class, it is a correct pixel, otherwise incorrect. The performances are normally averaged among all pixels.

Semantic segmentation identifies specific regions in an image belonging to a semantic category (e.g., a front-view image of a road from a car) (Fig. A.6). Semantic segmentation does not consider objects of the same semantic category. The objects overlapped in the image connect the semantic segmentation label regions.

Fig. A.6.: Example of semantic segmentation of road [FKG13].

Instance segmentation is a task that combines object detection and semantic segmentation. Instead of providing a bounding box of each object in an image, instance segmentation provides the area where the object is in the image. It is better at dealing with overlapped objects and non-rigid or curved object shapes than bounding box representation.

COCO dataset provides instance segmentation labels for images [Lin+14]. Fig. A.7 shows a few examples of labels overlying the images. Each object is labeled with its own segmentation region, and regions are not overlapped. It is a challenging task in

CV, as it not only deals with pixel-level classification but also needs to perform data association to cluster pixels of the same object.

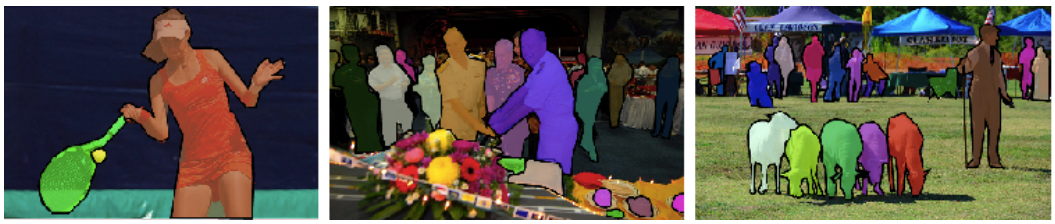

Fig. A.7.: Examples of instance segmentation labels in the COCO dataset.

This dissertation contributes to a few challenging tasks of image classification and tasks similar to image OD or image segmentation. It is understood that detecting objects and segmenting regions is not only limited to images but also includes videos or even image-like data. Any 2D or 3D matrix can be considered an image with single (2D case) or multiple (3D case) color channels. The latter chapters present radar representation in an image-like manner and the contribution of dealing with CV tasks on such image-like data.

Bibliography

[Aba+16] Martín Abadi, Ashish Agarwal, Paul Barham, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467* (2016) (cit. on p. 13).

[Abi+18] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, et al. "State-of-the-art in artificial neural network applications: A survey". In: *Heliyon* 4.11 (2018), e00938 (cit. on p. 12).

[Ali17] Haitham Ali. "Study of the Effect of RCS on Radar Detection". In: *European Scientific Journal* 13 (May 2017) (cit. on p. 23).

[Alo+09] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. "NP-hardness of Euclidean sum-of-squares clustering". In: *Machine learning* 75.2 (2009), pp. 245–248 (cit. on p. 17).

[Alp20] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020 (cit. on p. 6).

[AV06] David Arthur and Sergei Vassilvitskii. *k-means++: The advantages of careful seeding*. Tech. rep. Stanford, 2006 (cit. on p. 17).

[Ayo10] Taiwo Oladipupo Ayodele. "Introduction to Machine Learning". In: *New Advances in Machine Learning*. Ed. by Yagang Zhang. Rijeka: IntechOpen, 2010. Chap. 1 (cit. on p. 6).

[BT19] John E Ball and Bo Tang. *Machine learning and embedded computing in advanced driver assistance systems (ADAS)*. 2019 (cit. on p. 69).

[BB82] Dana H Ballard and Christopher M Brown. "Computer vision. englewood cliffs". In: *J: Prentice Hall* (1982) (cit. on p. 18).

[Bal+15] Nicolas Ballas, Li Yao, Chris Pal, and Aaron Courville. "Delving deeper into convolutional networks for learning video representations". In: *arXiv preprint arXiv:1511.06432* (2015) (cit. on p. 91).

[BKG20] Dor Bank, Noam Koenigstein, and Raja Giryes. "Autoencoders". In: *arXiv preprint arXiv:2003.05991* (2020) (cit. on p. 74).

[Bay+18] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. "Automatic differentiation in machine learning: a survey". In: *Journal of machine learning research* 18 (2018) (cit. on p. 13).

[Bez13] James C Bezdek. *Pattern recognition with fuzzy objective function algorithms*. Springer Science & Business Media, 2013 (cit. on p. 17).

[Bre01] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32 (cit. on p. 10).

[Bre+17] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 2017 (cit. on p. 10).

[Cae+20] Holger Caesar, Varun Bankiti, Alex H. Lang, et al. “nuScenes: A multimodal dataset for autonomous driving”. In: *CVPR*. 2020 (cit. on p. 119).

[Can86] John Canny. “A computational approach to edge detection”. In: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), pp. 679–698 (cit. on p. 19).

[Car+19] Alexandra Carlson, Katherine A Skinner, Ram Vasudevan, and Matthew Johnson-Roberson. “Sensor transfer: Learning optimal sensor effect image augmentation for Sim-to-Real domain adaptation”. In: *IEEE Robotics and Automation Letters* 4.3 (2019), pp. 2431–2438 (cit. on p. 18).

[Cho+14] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, et al. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *arXiv preprint arXiv:1406.1078* (2014) (cit. on pp. 16, 90).

[Cir+12] Dan C. Ciresan et al. “Multi-column deep neural network for traffic sign classification.” In: *Neural Networks* 32 (2012), pp. 333–338 (cit. on pp. 43, 44).

[Dai+16a] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. “R-fcn: Object detection via region-based fully convolutional networks”. In: *Advances in neural information processing systems*. 2016, pp. 379–387 (cit. on p. 15).

[Dai+17] Jifeng Dai et al. “Deformable Convolutional Networks”. In: *ICCV*. 2017 (cit. on pp. 32, 33).

[Dai+16b] Jifeng Dai et al. “R-FCN: Object Detection via Region-based Fully Convolutional Networks”. In: *NIPS*. 2016, pp. 379–387 (cit. on p. 30).

[DT05] Navneet Dalal and Bill Triggs. “Histograms of oriented gradients for human detection”. In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR’05)*. Vol. 1. Ieee. 2005, pp. 886–893 (cit. on pp. 19, 21).

[Den+09] J. Deng, W. Dong, R. Socher, et al. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *CVPR09*. 2009 (cit. on p. 122).

[DY14] Li Deng and Dong Yu. “Deep learning: methods and applications”. In: *Foundations and trends in signal processing* 7.3–4 (2014), pp. 197–387 (cit. on p. 16).

[DH20] Robert DiPietro and Gregory D Hager. “Deep learning: RNNs and LSTM”. In: *Handbook of medical image computing and computer assisted intervention*. Elsevier, 2020, pp. 503–519 (cit. on p. 16).

[Dun73] Joseph C Dunn. “A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters”. In: (1973) (cit. on p. 17).

[Est+96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. “A density-based algorithm for discovering clusters in large spatial databases with noise.” In: *kdd*. Vol. 96. 34. 1996, pp. 226–231 (cit. on p. 17).

[Eve+10] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. “The Pascal Visual Object Classes (VOC) Challenge”. In: *International Journal of Computer Vision* 88.2 (June 2010), pp. 303–338 (cit. on p. 125).

[Fan+08] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. “LIBLINEAR: A library for large linear classification”. In: *the Journal of machine Learning research* 9 (2008), pp. 1871–1874 (cit. on p. 10).

[FP11] David Forsyth and Jean Ponce. *Computer vision: A modern approach*. Prentice hall, 2011 (cit. on p. 22).

[FS+96] Yoav Freund, Robert E Schapire, et al. “Experiments with a new boosting algorithm”. In: *icml*. Vol. 96. Citeseer. 1996, pp. 148–156 (cit. on p. 10).

[Fri17] Jerome H Friedman. *The elements of statistical learning: Data mining, inference, and prediction*. springer open, 2017 (cit. on p. 9).

[FKG13] Jannik Fritsch, Tobias Kuehnl, and Andreas Geiger. “A new performance measure and evaluation benchmark for road detection algorithms”. In: *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. IEEE. 2013, pp. 1693–1700 (cit. on p. 126).

[GL21] Leonardo Galli and Chih-Jen Lin. “A Study on Truncated Newton Methods for Linear Classification”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2021) (cit. on p. 10).

[Gei+17] Robert Geirhos, David HJ Janssen, Heiko H Schütt, et al. “Comparing deep neural networks against humans: object recognition when the signal gets weaker”. In: *arXiv preprint arXiv:1706.06969* (2017) (cit. on p. 21).

[GS01] Felix A Gers and E Schmidhuber. “LSTM recurrent networks learn simple context-free and context-sensitive languages”. In: *IEEE Transactions on Neural Networks* 12.6 (2001), pp. 1333–1340 (cit. on p. 89).

[Gir15] Ross Girshick. “Fast r-cnn”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448 (cit. on pp. 30, 112, 113).

[Gir+14] Ross Girshick et al. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”. In: *CVPR*. CVPR ’14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 580–587 (cit. on p. 29).

[Goe13] Greg Goebel. *The Wizard War: WW2 & the origins of radar*. 2013 (cit. on p. 22).

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016 (cit. on p. 14).

[Goo+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, et al. “Generative adversarial nets”. In: *Advances in neural information processing systems* 27 (2014) (cit. on p. 13).

[Goo+13] Ian J. Goodfellow et al. “Maxout Networks”. In: *ICML*. ICML’13. Atlanta, GA, USA: JMLR.org, 2013, pp. III–1319–III–1327 (cit. on pp. 41, 42).

[Gru+17] Dominique Gruyer, Valentin Magnier, Karima Hamdi, et al. “Perception, information processing and modeling: Critical stages for autonomous driving applications”. In: *Annual Reviews in Control* 44 (2017), pp. 323–341 (cit. on p. 9).

[HS+88] Chris Harris, Mike Stephens, et al. “A combined corner and edge detector”. In: *Alvey vision conference*. Vol. 15. 50. Citeseer. 1988, pp. 10–5244 (cit. on p. 19).

[Hea21] National Cancer Institute. U. S. National Institutes of Health. *SEER Training Modules, Cancer Registration & Surveillance Modules.* 2021 (cit. on p. 12).

[Hel+20] Abdelhamid Helali, Haythem Ameur, JM Górriz, J Ramírez, and Hassen Maaref. “Hardware implementation of real-time pedestrian detection system”. In: *Neural Computing and Applications* (2020), pp. 1–13 (cit. on p. 21).

[HS+99] Geoffrey E Hinton, Terrence Joseph Sejnowski, et al. *Unsupervised learning: foundations of neural computation*. MIT press, 1999 (cit. on p. 6).

[Ho95] Tin Kam Ho. “Random decision forests”. In: *Proceedings of 3rd international conference on document analysis and recognition*. Vol. 1. IEEE. 1995, pp. 278–282 (cit. on p. 10).

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cit. on p. 87).

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366 (cit. on p. 14).

[Hsi+08] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S Sathiya Keerthi, and Sellamanickam Sundararajan. “A dual coordinate descent method for large-scale linear SVM”. In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 408–415 (cit. on p. 10).

[Hua+19] Cai Huaiyu, Chen Yanzhen, Zhuo Liran, and Chen Xiaodong. “LiDAR object detection based on optimized DBSCAN algorithm”. In: *Opto-Electronic Engineering* 46.7 (2019), pp. 180514–1 (cit. on p. 17).

[Hyö96] Heikki Hyötyniemi. “Turing machines are recurrent neural networks”. In: *Proceedings of step* 96 (1996) (cit. on p. 15).

[Inc21] The MathWorks Inc. *Radar Toolbox*. Natick, Massachusetts, USA, 2021 (cit. on p. 18).

[IIN17] Noman Islam, Zeeshan Islam, and Nazia Noor. “A survey on optical character recognition system”. In: *arXiv preprint arXiv:1710.05703* (2017) (cit. on p. 18).

[Jad+15] Max Jaderberg et al. “Spatial Transformer Networks”. In: *NIPS*. 2015, pp. 2017–2025 (cit. on pp. 31, 33–36).

[Jia+16] Xu Jia et al. “Dynamic Filter Networks”. In: *NIPS*. 2016, pp. 667–675 (cit. on p. 30).

[Kas80] Gordon V Kass. “An exploratory technique for investigating large quantities of categorical data”. In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 29.2 (1980), pp. 119–127 (cit. on p. 10).

[KR90] Leonard Kaufman and Peter J Rousseeuw. “Partitioning around medoids (program pam)”. In: *Finding groups in data: an introduction to cluster analysis* 344 (1990), pp. 68–125 (cit. on p. 17).

[KKD12] Dominik Kellner, Jens Klappstein, and Klaus Dietmayer. “Grid-based DBSCAN for clustering extended objects in radar data”. In: *2012 IEEE Intelligent Vehicles Symposium*. IEEE. 2012, pp. 365–370 (cit. on p. 17).

[Kha+20] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. “A survey of the recent architectures of deep convolutional neural networks”. In: *Artificial Intelligence Review* 53.8 (2020), pp. 5455–5516 (cit. on p. 21).

[KZS+15] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. “Siamese neural networks for one-shot image recognition”. In: *ICML deep learning workshop*. Vol. 2. Lille. 2015 (cit. on p. 21).

[Kra91] Mark A Kramer. “Nonlinear principal component analysis using autoassociative neural networks”. In: *AIChE journal* 37.2 (1991), pp. 233–243 (cit. on p. 13).

[Kri+12] Alex Krizhevsky et al. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *NIPS*. 2012, pp. 1106–1114 (cit. on p. 29).

[KB14] Gaurav Kumar and Pradeep Kumar Bhatia. “A detailed review of feature extraction in image processing systems”. In: *2014 Fourth international conference on advanced computing & communication technologies*. IEEE. 2014, pp. 5–12 (cit. on p. 18).

[LB98] Yann LeCun and Yoshua Bengio. “The Handbook of Brain Theory and Neural Networks”. In: ed. by Michael A. Arbib. Cambridge, MA, USA: MIT Press, 1998. Chap. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258 (cit. on p. 29).

[LeC+89] Yann LeCun, Bernhard Boser, John S Denker, et al. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551 (cit. on p. 14).

[LC10] Yann LeCun and Corinna Cortes. “MNIST handwritten digit database”. In: (2010) (cit. on p. 121).

[Li+18] Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. “Independently recurrent neural network (indrnn): Building a longer and deeper rnn”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 5457–5466 (cit. on p. 16).

[Li+16] Yuan Li et al. “Pushing the "Speed Limit": High-Accuracy US Traffic Sign Recognition With Convolutional Neural Networks”. In: *IEEE T-IV* 1.2 (2016), pp. 167–176 (cit. on p. 29).

[LC15] Zhibin Liao and Gustavo Carneiro. “Competitive Multi-scale Convolution”. In: *CoRR* abs/1511.05635 (2015) (cit. on pp. 41, 42).

[LYP16] Seung-Hwan Lim, Steven R Young, and Robert M Patton. “An analysis of image storage systems for scalable training of deep neural networks”. In: *system* 5.7 (2016), p. 11 (cit. on p. 122).

[LLK18] Sohee Lim, Seongwook Lee, and Seong-Cheol Kim. “Clustering of detected targets using DBSCAN in automotive radar systems”. In: *2018 19th international radar symposium (IRS)*. IEEE. 2018, pp. 1–7 (cit. on p. 17).

[Lin+17] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. “Focal loss for dense object detection”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988 (cit. on p. 112).

[Lin+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, et al. “Microsoft coco: Common objects in context”. In: *European conference on computer vision*. Springer. 2014, pp. 740–755 (cit. on p. 126).

[Liu+16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, et al. “Ssd: Single shot multibox detector”. In: *European conference on computer vision*. Springer. 2016, pp. 21–37 (cit. on p. 111).

[Llo82] Stuart Lloyd. “Least squares quantization in PCM”. In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137 (cit. on p. 16).

[LS97] Wei-Yin Loh and Yu-Shan Shih. “Split selection methods for classification trees”. In: *Statistica sinica* (1997), pp. 815–840 (cit. on p. 10).

[LG15] Bhagyashri P Lokhande and Sanjay S Gharde. “A Review on Large-scale Video Classification with Recurrent Neural Network (RNN)”. In: *International Journal of Computer Science and Information Technologies, Jalgaon, India* (2015) (cit. on p. 15).

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440 (cit. on pp. 15, 29).

[MP43] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133 (cit. on p. 11).

[Mer+01] I Skolnik Merrill et al. “Introduction to radar systems”. In: *Mc Grow-Hill* 7.10 (2001) (cit. on pp. 23, 25).

[MS10] Janardan Misra and Indranil Saha. “Artificial neural networks in hardware: A survey of two decades of progress”. In: *Neurocomputing* 74.1-3 (2010), pp. 239–255 (cit. on p. 12).

[Mit97] Tom M. Mitchell. *Machine Learning*. New York: McGraw-Hill, 1997 (cit. on pp. 6, 10).

[Miy+20] Tomo Miyata, Masahiro Yanagawa, Akinori Hata, et al. “Influence of field of view size on image quality: ultra-high-resolution CT vs. conventional high-resolution CT”. In: *European radiology* 30.6 (2020), p. 3324 (cit. on p. 18).

[Mou+18] Abdallah Moujahid, Mounir ElAraki Tantaoui, Manolo Dulva Hina, et al. “Machine learning techniques in ADAS: a review”. In: *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE)*. IEEE. 2018, pp. 235–242 (cit. on p. 9).

[NQ19] Ramin Nabati and Hairong Qi. “Rrpn: Radar region proposal network for object detection in autonomous vehicles”. In: *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2019, pp. 3093–3097 (cit. on p. 73).

[NH10] Vinod Nair and Geoffrey E Hinton. “Rectified linear units improve restricted boltzmann machines”. In: *Icml*. 2010 (cit. on p. 14).

[Net+11] Yuval Netzer, Tao Wang, Adam Coates, et al. “Reading digits in natural images with unsupervised feature learning”. In: (2011) (cit. on p. 121).

[NRU20] Eshaan Nichani, Adityanarayanan Radhakrishnan, and Caroline Uhler. “Do Deeper Convolutional Networks Perform Better?” In: *arXiv preprint arXiv:2010.09610* (2020) (cit. on p. 14).

[Nob+19] Felix Nobis, Maximilian Geisslinger, Markus Weber, Johannes Betz, and Markus Lienkamp. “A deep learning-based radar and camera sensor fusion architecture for object detection”. In: *2019 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*. IEEE. 2019, pp. 1–7 (cit. on p. 73).

[NZS19a] Christian Nunn, Weimeng Zhu, and Yu Su. *A device and a method for extracting dynamic information on a scene using a convolutional neural network*. Patent. EP3561727A1. Oct. 2019 (cit. on p. 79).

[NZS19b] Christian Nunn, Weimeng Zhu, and Yu Su. *A device and a method for extracting dynamic information on a scene using a convolutional neural network*. Patent. US20190325241A1. Oct. 2019 (cit. on p. 79).

[NZS19c] Christian Nunn, Weimeng Zhu, and Yu Su. *A device and a method for extracting dynamic information on a scene using a convolutional neural network*. Patent. CN110390249A. Oct. 2019 (cit. on p. 79).

[NZS19d] Christian Nunn, Weimeng Zhu, and Yu Su. *A device and a method for processing data sequences using a convolutional neural network*. Patent. EP3561726A1. Oct. 2019 (cit. on p. 86).

[NZS19e] Christian Nunn, Weimeng Zhu, and Yu Su. *A device and a method for processing data sequences using a convolutional neural network*. Patent. US20190325306A1. Oct. 2019 (cit. on p. 86).

[NZS19f] Christian Nunn, Weimeng Zhu, and Yu Su. *A device and a method for processing data sequences using a convolutional neural network*. Patent. CN110390381A. Oct. 2019 (cit. on p. 86).

[O'M+19] Niall O'Mahony, Sean Campbell, Anderson Carvalho, et al. "Deep learning vs. traditional computer vision". In: *Science and Information Conference*. Springer. 2019, pp. 128–144 (cit. on p. 19).

[Oua+21] Arthur Ouaknine, Alasdair Newson, Julien Rebut, Florence Tupin, and Patrick Pérez. "CARRADA Dataset: Camera and Automotive Radar with Range- Angle-Doppler Annotations". In: *2020 25th International Conference on Pattern Recognition (ICPR)*. 2021, pp. 5068–5075 (cit. on p. 119).

[PMB13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *International conference on machine learning*. PMLR. 2013, pp. 1310–1318 (cit. on p. 15).

[Pas+17] Adam Paszke, Sam Gross, Soumith Chintala, et al. "Automatic differentiation in pytorch". In: (2017) (cit. on p. 13).

[Pee+16] Maurice Peemen, Runbin Shi, Sohan Lal, et al. "The neuro vector engine: Flexibility to improve convolutional net efficiency for wearable vision". In: *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2016, pp. 1604–1609 (cit. on p. 14).

[Pen+17] Scott Drew Pendleton, Hans Andersen, Xinxin Du, et al. "Perception, planning, control, and coordination for autonomous vehicles". In: *Machines* 5.1 (2017), p. 6 (cit. on p. 8).

[Qi+17a] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660 (cit. on p. 71).

[Qi+17b] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space". In: *arXiv preprint arXiv:1706.02413* (2017) (cit. on pp. 71, 72).

[Qui93] JR Quinlan. "C4. 5: Programs for machine learning Morgan Kaufmann San Francisco". In: *CA, USA* (1993) (cit. on p. 10).

[Red+16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788 (cit. on p. 111).

[Ren+15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems* 28 (2015), pp. 91–99 (cit. on p. 113).

[Rod21] Thomas Roddick. "Learning Birds-Eye View Representations for Autonomous Driving". PhD thesis. University of Cambridge, 2021 (cit. on p. 85).

[Roh11] Hermann Rohling. "Ordered statistic CFAR technique-an overview". In: *2011 12th International Radar Symposium (IRS)*. IEEE. 2011, pp. 631–638 (cit. on p. 26).

[Roh83] Hermann Rohling. “Radar CFAR thresholding in clutter and multiple target situations”. In: *IEEE transactions on aerospace and electronic systems* 4 (1983), pp. 608–621 (cit. on p. 26).

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241 (cit. on pp. 71, 79).

[Ros58] Frank Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65.6 (1958), p. 386 (cit. on p. 12).

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536 (cit. on pp. 13, 37).

[Rus+15] Olga Russakovsky, Jia Deng, Hao Su, et al. “Imagenet large scale visual recognition challenge”. In: *International journal of computer vision* 115.3 (2015), pp. 211–252 (cit. on p. 21).

[RN02] Stuart Russell and Peter Norvig. “Artificial intelligence: a modern approach”. In: (2002) (cit. on p. 6).

[SH00] JZ Sasiadek and P Hartana. “Sensor data fusion using Kalman filter”. In: *Proceedings of the Third International Conference on Information Fusion*. Vol. 2. IEEE. 2000, WED5–19 (cit. on p. 83).

[Sch05] Martin Schneider. “Automotive radar-status and trends”. In: *German microwave conference*. 2005, pp. 144–147 (cit. on p. 24).

[Sch+17] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. “DBSCAN revisited, revisited: why and how you should (still) use DBSCAN”. In: *ACM Transactions on Database Systems (TODS)* 42.3 (2017), pp. 1–21 (cit. on p. 17).

[Sch+18] Ole Schumann, Markus Hahn, Jürgen Dickmann, and Christian Wöhler. “Semantic segmentation on radar point clouds”. In: *2018 21st International Conference on Information Fusion (FUSION)*. IEEE. 2018, pp. 2179–2186 (cit. on pp. 72, 73).

[Sch+21] Ole Schumann, Markus Hahn, Nicolas Scheiner, et al. “RadarScenes: A real-world radar point cloud data set for automotive applications”. In: *arXiv preprint arXiv:2104.02493* (2021) (cit. on p. 119).

[SP97] Mike Schuster and Kuldip K Paliwal. “Bidirectional recurrent neural networks”. In: *IEEE transactions on Signal Processing* 45.11 (1997), pp. 2673–2681 (cit. on p. 87).

[SL11] Pierre Sermanet and Yann LeCun. “Traffic sign recognition with multi-scale Convolutional Networks”. In: *IJCNN*. IEEE, 2011, pp. 2809–2813 (cit. on pp. 43, 44).

[Ser+12] Pierre Sermanet et al. “Convolutional Neural Networks Applied to House Numbers Digit Classification"”. In: *ICPR*. 2012 (cit. on pp. 41, 42).

[STS11] John Shawe-Taylor and Shiliang Sun. “A review of optimization methodologies in support vector machines”. In: *Neurocomputing* 74.17 (2011), pp. 3609–3618 (cit. on p. 10).

[SZ14] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014) (cit. on p. 123).

[Sin+17] Shashi Pal Singh, Ajai Kumar, Hemant Darbari, et al. “Machine translation using deep learning: An overview”. In: *2017 international conference on computer, communications and electronics (comptelix)*. IEEE. 2017, pp. 162–167 (cit. on p. 15).

[SD18] Narasimha Reddy Soora and Parag S Deshpande. “Review of feature extraction techniques for character recognition”. In: *IETE Journal of Research* 64.2 (2018), pp. 280–295 (cit. on p. 19).

[Sta+12] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. “Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition”. In: *Neural Networks* 0 (2012), pp. – (cit. on p. 122).

[Sta11] International Organization for Standardization. *ISO 8855 - Road Vehicles, Vehicle dynamic and road-holding ability*. 2011 (cit. on p. 52).

[Su+21a] Yu Su, Weimeng Zhu, Florian Kästner, and Adrian Becker. *Methods and systems for object detection*. Patent. EP3767332A1. Jan. 2021 (cit. on p. 114).

[Su+21b] Yu Su, Weimeng Zhu, Florian Kästner, and Adrian Becker. *Methods and systems for object detection*. Patent. US20210018615A1. Jan. 2021 (cit. on p. 114).

[Su+21c] Yu Su, Weimeng Zhu, Florian Kästner, and Adrian Becker. *Methods and systems for object detection*. Patent. CN112241008A. Jan. 2021 (cit. on p. 114).

[SZM20a] Yu Su, Weimeng Zhu, and Mirko Meuter. *Method of multi-sensor data fusion*. Patent. EP3702802A1. Sept. 2020 (cit. on p. 83).

[SZM20b] Yu Su, Weimeng Zhu, and Mirko Meuter. *Method of multi-sensor data fusion*. Patent. US20200280429A1. Sept. 2020 (cit. on p. 83).

[SZM20c] Yu Su, Weimeng Zhu, and Mirko Meuter. *Method of multi-sensor data fusion*. Patent. CN111639663A. Sept. 2020 (cit. on p. 83).

[Sul+17] Amr Suleiman, Yu-Hsin Chen, Joel Emer, and Vivienne Sze. “Towards closing the energy gap between HOG and CNN features for embedded vision”. In: *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2017, pp. 1–4 (cit. on p. 21).

[Sun+16] Shizhao Sun et al. “On the Depth of Deep Neural Networks: A Theoretical View”. In: *AAAI*. 2016, pp. 2066–2072 (cit. on p. 33).

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018 (cit. on p. 6).

[Vap99] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999 (cit. on p. 10).

[VKJ11] RLK Venkateswarlu, R Vasantha Kumari, and G Vani JayaSri. "Speech Recognition by Using Recurrent NeuralNetworks". In: *International Journal of Scientific & Engineering Research* 2.6 (2011), pp. 1–7 (cit. on p. 15).

[VSP13] Gaurav Vijayvargiya, Sanjay Silakari, and Rajeev Pandey. "A survey: various techniques of image compression". In: *arXiv preprint arXiv:1311.6877* (2013) (cit. on p. 18).

[Wer74] Paul Werbos. "Beyond regression:" new tools for prediction and analysis in the behavioral sciences". In: *Ph. D. dissertation, Harvard University* (1974) (cit. on p. 12).

[Win07] Volker Winkler. "Range Doppler detection for automotive FMCW radars". In: *2007 European Radar Conference*. IEEE. 2007, pp. 166–169 (cit. on p. 25).

[Xia+20] Yi Xiao, Felipe Codevilla, Akhil Gurram, Onay Urfalioglu, and Antonio M López. "Multimodal end-to-end autonomous driving". In: *IEEE Transactions on Intelligent Transportation Systems* 23.1 (2020), pp. 537–547 (cit. on p. 8).

[Xin+15] SHI Xingjian, Zhourong Chen, Hao Wang, et al. "Convolutional LSTM network: A machine learning approach for precipitation nowcasting". In: *Advances in neural information processing systems*. 2015, pp. 802–810 (cit. on p. 89).

[ZT00] Liang Zhao and Charles E. Thorpe. "Stereo- and neural network-based pedestrian detection". In: *IEEE T-ITS* 1.3 (2000), pp. 148–154 (cit. on p. 29).

[Zho20] Ding-Xuan Zhou. "Universality of deep convolutional neural networks". In: *Applied and computational harmonic analysis* 48.2 (2020), pp. 787–794 (cit. on p. 14).

[ZT18] Yin Zhou and Oncel Tuzel. "Voxelnet: End-to-end learning for point cloud based 3d object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4490–4499 (cit. on pp. 71, 72, 81).

[ZS19a] Weimeng Zhu and Jan Siegemund. *Method of processing image data in a connectionist network*. Patent. EP3495988A1. June 2019 (cit. on p. 29).

[ZS19b] Weimeng Zhu and Jan Siegemund. *Method of processing image data in a connectionist network*. Patent. US20190171939A1. June 2019 (cit. on p. 29).

[ZS19c] Weimeng Zhu and Jan Siegemund. *Method of processing image data in a connectionist network*. Patent. CN110033009A. July 2019 (cit. on p. 29).

[ZSK18] Weimeng Zhu, Jan Siegemund, and Anton Kummert. "Dense Spatial Translation Network". In: *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*. IEEE. 2018, pp. 1–7 (cit. on p. 29).

[Zhu+21a] Weimeng Zhu, Yu Su, Peet Cremer, et al. *Method and system for object detection*. Patent. EP3767324A1. Jan. 2021 (cit. on p. 119).

[Zhu+21b] Weimeng Zhu, Yu Su, Peet Cremer, et al. *Method and system for object detection*. Patent. US20210018609A1. Jan. 2021 (cit. on p. 119).

[Zhu+21c] Weimeng Zhu, Yu Su, Peet Cremer, et al. *Method and system for object detection*. Patent. CN112241003A. Jan. 2021 (cit. on p. 119).

List of Figures

List of Tables

List of Glossaries

Acronyms

Acronym	Description	Page(s)
AD	Autonomous Driving	8, 9, 29, 49, 69, 82, 86, 119
AdaBoost	Adaptive Boosting	10
ADAS	Advanced Driver-Assistance System	29, 33, 39, 42, 44, 46, 49, 69, 82, 86
AI	Artificial Intelligence	6
ANN	Artificial Neural Network	11–16, 29, 49, 57, 69, 141
CART	Classification And Regression Tree	10
CDC	Compressed Data Cube	27, 119
CFAR	Constant False Alarm Rate	26, 27, 141
CHAID	Chi-Squared Automatic Interaction Detection	10
CNN	Convolutional Neural Network	13–16, 19, 21, 29–31, 33, 36, 69, 71, 78, 79, 85, 94, 95, 111, 141
CUT	Cell Under Test	26
CV	Computer Vision	18, 19, 21, 22, 29, 49, 50, 69–72, 119, 121, 123, 127
CW	Continuous Wave	24

Acronym	Description	Page(s)
DBSCAN	Density-Based Spatial Clustering of Applications with Noise	17, 18, 108–110, 144
DCNN	Deformable Convolutional Network	32, 33, 35, 142
dGPS	Differential Global Positioning System	54, 58, 65
DL	Deep Learning	16, 19, 21, 29, 66, 69–73, 119, 123
DOA	Direction of Arrival	25, 27
DSP	Digital Signal Processor	39, 45
DSTN	Dense Spatial Translation Network	29, 32–47, 142
FCN	Fully Convolutional Network	15
FFT	Fast Fourier Transform	25, 27, 55, 56, 141, 142
FMCW	Frequency Modulated Continuous Wave	24, 25, 141
FOV	Field of View	18, 51, 58, 65
GAN	Generative Adversarial Network	13
GPS	Global Positioning System	50, 54
GPU	Graphics Processing Unit	12, 19
GRS	Gated Recurrent Sampler	93, 95–107, 114, 116, 117, 143, 144
GRU	Gated Recurrent Unit	90, 91, 93, 95, 102, 103, 114, 117, 143, 144
GTSRB	German Traffic Sign Recognition Benchmark	42–44, 122, 145, 147
HOG	Histogram of Oriented Gradients	19–21, 141
IMU	Inertial Measurement Unit	58, 65
IoU	Intersection-over-Union	116, 123, 125
LSTM	Long Short-Term Memory	87, 89–91, 93, 95, 116, 117, 143
MIMO	Multiple Input Multiple Output	25

Acronym	Description	Page(s)
ML	Machine Learning	1, 2, 5–7, 9, 16, 19, 39, 49, 57, 58, 66, 69, 70, 122
MLP	Multi-Layer Perceptron	12–14, 16, 71
MNIST	Modified National Institute of Standards And Technology	39, 40, 121, 122, 145
NMS	Non-Maximal Suppression	111, 112, 116
OCR	Optical Character Recognition	18
OD	Object Detection	22, 29, 123, 125, 127
QUEST	Quick, Unbiased, Efficient, Statistical Tree	10
RADAR	Radio Detection and Ranging	22
RAM	Random-Access Memory	39, 45
RCS	Radar Cross Section	23, 27, 50, 57, 65, 76, 78, 85
RDA	Range-Doppler-Angle	25, 26
ReLU	Rectified Linear Unit	14
RNN	Recurrent Neural Network	15, 16, 86, 87, 89, 91, 93, 116, 117, 141
ROI	Region of Interest	30, 32, 112
SCS	Sensor Coordinate System	50–55, 57, 74–76
SNR	Signal-to-Noise Ratio	74, 75
SSD	Single Shot Multibox Detector	111, 112, 144
STN	Spatial Transformer Network	31–33, 141
SVHN	Street View House Number	31, 39–42, 121, 122, 145, 147
SVM	Support Vector Machine	10, 11, 21, 141
TSR	Traffic Sign Recognition	29, 31, 42, 44, 45, 147
V2X	Vehicle-to-Everything	69

Acronym	Description	Page(s)
VCS	Vehicle Coordinate System	52–55, 63–66, 74–76, 83, 85, 86, 92, 93
VRU	Vulnerable Road User	69
WCS	World Coordinate System	54, 55, 62
YOLO	You Only Look Once	111, 112, 144