Investigating Object Detection and Semantic Segmentation Using Preprocessed Radar Data

Albin Erlander, Felix Persson

Master's thesis
2024:E30

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Centrum Scientiarum Mathematicarum

Abstract

While cameras are the most prevalent devices used in physical surveillance and monitoring, there are situations where they are ineffective. In adverse weather conditions, darkness or privacy-sensitive contexts, there are excellent opportunities to replace or complement cameras with radar.

There are advanced and successful computer vision solutions for cameras, in areas such as object detection or semantic segmentation. However, the equivalent solutions are potentially underutilized for radar. As with cameras, computer vision applied on radar data could be potentially very useful and have a variety of applications. Of interest to this thesis specifically is the possibility of using computer vision techniques for optimizing radar signal processing. To this end, this thesis aims to investigate the potential of instantaneous object detection and semantic segmentation on preprocessed radar data.

A novel annotation framework, which is automated and camera-assisted, is developed to generate a custom data set. Three models are implemented and tested: AdaBoost (classifier), YOLOv8 (state-of-the-art object detection) and an adapted U-Net (semantic segmentation).

The results indicate that object detection and semantic segmentation based on single frames of radar data generated early in the signal processing chain is not only feasible, but promising.

Keywords

object detection, semantic segmentation, computer vision, U-Net, YOLO, AdaBoost, FMCW radar, range-Doppler, radar data annotation

Popular Science Summary

When thinking of physical security and surveillance, most people probably think of security cameras. However, there are situations where cameras do not work well, as anyone who has tried photographing in darkness or heavy rain knows. There are also situations where cameras can be inappropriate because of privacy reasons, but where there is still a need for security and monitoring.

In such cases, radar is a good alternative or complement to security cameras. Radars work by transmitting radio waves out into the environment, where they reflect off objects and then return to be received by the radar. How far away an object is can then be determined from how long it takes a signal to return, and where it is can be determined from the direction the signal returns from. This makes radars very good at finding where objects are located. In addition, if multiple signals are sent and received, the velocity of the object can be determined, which can be a big advantage over cameras.

There have been great advances made in machine learning and artificial intelligence over the past couple of years. One specific area of machine learning with a lot of applications is the field of computer vision. Put simply, this means teaching a machine learning model to look at images and recognize objects. This can for example be used to look at traffic camera photos and read license plates, or look at X-rays and help with diagnostics. While computer vision has been used for radar images as well, this field has not received as much attention. The aim of this thesis is to investigate some possibilities with using computer vision techniques for looking at radar images.

A radar typically receives a lot of data, but not all of it is useful. The data is processed and transformed in consecutive steps, called the 'signal processing chain'. For a very simplified example, imagine an empty field with only one radar and a single person standing ten meters in front of it. The radar transmits radio waves in many directions, some reflect off the person and return to the radar, which gets to work. The raw returning signal is itself hard to interpret, so the radar starts to transform it into something that is easier to work with. It performs a lot of calculations to determine how long it took the signal to return, from what angle it arrived, whether the object it bounced off is moving, and more. In a more realistic scenario, the radar is continuously transmitting signals and receiving signals which has to be interpreted, and the amount of processing required adds up quickly.

As mentioned, not all the information that is received and processed is useful. This is part of the problem this thesis wants to address. What if, in the example presented above, a machine learning model could tell the radar: 'Hey, the object ten meters in front of you seems to be a person'. If the radar knew what type of object was being detected early on, it could make it more precise and efficient by tailoring the processing of the signals. However, it creates some challenges for the machine learning model. First off, this means it has to be relatively quick itself. It also means the model has to act upon the data before it is fully processed, which means the available information is somewhat limited. Secondly, it has to be relatively light-weight so that it can be run on a radar unit, as opposed to a large and powerful computer. In this thesis, we implement, train and test three models, to investigate the possibilities and limitations for solving this problem.

Acknowledgements

We want to thank our industry supervisors Anders Skoog and Joel Nilsson for their time and support throughout all parts of this project. We also want to thank Santhosh Nadig for his valuable inputs.

Finally, we would like to thank our supervisor from Lund University Anders Heyden for his advice and help, as well as our examiner Carl Olsson.

Albin Erlander and Felix Persson

Contents

List of abbreviations

Abbreviation	Definition
AdaBoost	Adaptive Boosting
Adam	Adaptive moment estimation
ANN	Artificial Neural Network
C2f	Cross-stage partial bottleneck with 2 convolutions
CAE	Convolutional Autoencoder
CNN	Convolutional Neural Network
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DSP	Digital Signal Processing
FCN	Fully Convolutional Network
FMCW	Frequency Modulated Continuous Wave
IoU	Intersection over Union
MLP	Multilayer Perceptron
RAD	Range-Azimuth-Doppler
ReLU	Rectified Linear Unit
RCS	Radar Cross Section
SAMME	Stagewise Additive Modeling using a Multi-class Exponential loss function
SiLU	Sigmoid Linear Unit
SPPF	Spatial Pyramid Pooling Fast
SSD	Single-Shot Detector
YOLO	You Only Look Once
YOLOv8n	YOLO version 8 nano

1 Introduction

1.1 Background

Cameras are probably the technology most associated with physical surveillance and monitoring. However, there are situations such as in darkness or adverse weather conditions where cameras are ineffective. There can also be privacy considerations, and situations where recording with a camera is inappropriate. Under these and other conditions, there are good opportunities to replace or complement camera systems with radar. Radar technology generally provides good positional and radial velocity estimates, and the recorded data is less sensitive in terms of privacy. The data also retains its accuracy under some conditions where a camera might underperform, such as in darkness, fog, or rain.

There have been significant advances made in computer vision using cameras and machine learning, such as facial recognition and object detection. The success can be linked to the development of more sophisticated models and more computationally capable GPUs, as well as the increase of publicly accessible large scale data sets [34]. However, similar methods for radar data are potentially underutilized.

Computer vision using radar data could complement camera detection and classification when it fails, but also has other applications. For example, if classification could be performed early in the digital signal processing (DSP) chain, later stages could be focused on and optimized for the detected object's class. This would however present some significant challenges. The radar output data must be formatted and annotated to work with a suitable machine learning model, which has to be trained without the benefit of more sophisticated methods performed later on in the DSP. If the detection and classification is done in a surveillance setting, especially if it is to be done to benefit the DSP, the model also has to be lightweight and able to run on an embedded system.

1.2 Purpose

The purpose of this thesis is to investigate the potential of instantaneous object detection and segmentation based on preprocessed frequency modulated continuous wave (FMCW) radar data. Instantaneous here refers to operating frame-by-frame, and not with a temporal dimension. Preprocessed data refers to radar data that is partially processed, but not fully refined. The motivation is that such a computer vision framework, that is instantaneous and based on data from early stages in the DSP chain, can benefit the subsequent DSP. While that is the primary motivation, and hence an important point for the reader to be aware of, actual embedded implementation and testing is beyond the scope of this thesis.

To achieve this goal, a secondary one arises: The creation of a custom data set and annotation framework. Therefore, this study has produced its own data set, where the data is in the form of range-Doppler maps. It is automatically annotated by clustering and projecting radar targets onto synchronized camera images and mapping these clusters to predictions made by YOLO, a state-of-the-art camera image object detection model. In the context of this thesis, the primary classes of interest are: person, bicycle, and car.

Finally, the selection and testing of models naturally becomes another central area of this study. Three models are tested in this thesis: a multiclass AdaBoost for classification and feature extraction, YOLO for object detection, and a U-Net based model for semantic segmentation.

1.3 Previous research

A comparatively small amount of research is directed towards deep learning models and their application within the domain of radar. This stems from the deficit of publicly available data sets and the representations of input and output data varying within the field [1, 5, 44]. FMCW radars also have relatively high levels of noise and provide low angle resolution as well as less intuitive output data than those of other sensors, all of which are reasons behind other sensors being favored historically [24]. There are however some inherent benefits with FMCW radars in comparison to the other sensors, where their low cost, robustness towards weather and lighting conditions, and ability to capture radial velocity stand out, and they have proven adequate for several tasks, such as object detection [5, 24].

1.3.1 Radar data as input to neural networks

According to a 2021 review of deep learning applications on millimiter-wave radar signals, one of the most challenging tasks on the topic is representing the radar signals to fit as inputs to the variety of algorithms used [1]. This section brings up some ways of representing radar data, and how these have been used as inputs to neural networks in the previous research. Note that a more detailed explanation of radar signal processing and output data from an FMCW radar is provided in Section 2.8. For the purposes of this project, the data representations of most interest are range-azimuth-Doppler (RAD) mappings (analogous to range, angle and velocity).

A complete representation of the RAD data comes in the form of a three-dimensional cuboid, with discrete bins of range, azimuth and Doppler along each respective axis. Extracting a two-dimensional matrix from this three-dimensional cuboid is sometimes referred to as 'collapsing' one of the dimensions, which can be done by for example summing all elements along the axis to be collapsed. Using two-dimensional 'slices' of the RAD cuboid yields two-dimensional matrices of spatial data, in some respects more resemblant to the outputs from imaging sensors, on which many computer vision models are based. This, in turn, implies that these data representations allow for greater use of, or greater inspiration to be taken from, these models and other image processing methods.

The review in [1] includes 13 studies that trained models with only radar data as input. The main radar signal representations were: Micro-Doppler signatures, point clouds (detections in three-dimensional cartesian space), top-down representations (bird-eye views), and maps of combinations of range, azimuth, Doppler and elevation data. Micro-Doppler signatures are high resolution, detailed movements signatures, for example how a car wheel moves distinctly from the rest of the car.

There are several examples of studies training computer vision models with different profiles from the RAD data cuboid. The choice of representation varies from intuitive range-azimuth representations for example [25, 33] to using all pairwise combinations of the three parameters, such as [5, 21]. Further, the authors of [25] point out that dynamic objects with distinct Doppler spectra are easier to classify because of their micro-Doppler signatures. They also observed that range-azimuth spectra were particularly beneficial for classification in their study. Specifically they used the two-dimensional range-azimuth slices, extracted from where the Doppler intensity was at maximum for each frame [25].

As stated, there are examples of studies utilizing all the data from the RAD cuboid. One example is the input data used to train the RAMP-CNN (short for 'radar multiple-perspectives convolutional neural network') model, which uses three different two-dimensional maps combining range, azimuth, and Doppler values [5]. Another similar approach was taken by the authors of [21]. Their study includes two separate models, one trained only on range-azimuth data (summed over Doppler dimension), and one trained on all two-dimensional combinations of range-azimuth-Doppler. They also tried different models, and different coordinate systems (cartesian, polar) for input and output data. The hypothesis was that since their model worked by dividing the input image into uniform, cartesian grid cells, a transformation from polar coordniates (range-azimuth) to cartesian would be beneficial. Their results indictate that the hypotehsis was correct for their model. While the results varied, they also concluded that including the Doppler dimension helped with detection performance in the simpler models.

1.3.2 Annotation of radar data

Another major problem for deep learning applications on radar data is the lack of publicly accessible and annotated data sets [1, 5, 44]. Gao et al. also bring forward how manual labelling is more difficult for radar data than for camera images [5]. This implies that the design of a model would benefit from either adapting to a public data set or a robust annotation methodology. The previously discussed review [1] includes a summary of some publicly available data sets with varying methods for data annotation. One important aspect that varies between methodologies is the degree of automation, here referred to as manual, semi- and fully automatic. This section will focus on semi- and fully automatic methods aided by camera images.

First off, there are examples of semi-automatically annotated data sets, such as [22]. This study automatically labels the radar data through an object detection model, that uses both camera and radar data. After this, a fixed number of frames are manually corrected prioritizing the ones with the highest uncertainty, before the next round of training and prelabelling takes place. Another research paper [24] provides a semi-automatic annotation algorithm, were the authors use a stationary setup containing a FMCW radar together with a synchronized camera. Here, annotation is performed through usage of a Mask R-CNN in combination with a tracking algorithm, thus using information from consecutive frames.

Secondly, there are the fully automatic annotation methods (allowing for minor manual corrections). Of particular relevance to this study are camera-assisted automatic annotations, such as the data set used to train the RAMP-CNN model [5]. For this data set, binocular cameras were used alongside camera image detection and depth estimation

algorithms to provide the ground truths for the radar data. The annotations were provided as midpoints (range and azimuth), class and frame number, to account for time sequences of data. The choice to use points rather than bounding boxes, segmentation masks or other labels was made to ease the training burden. Another example of automatic camera annotation is performed by the RODnet family of models (three variations) [42]. The annotation first uses a monocular camera and a 3D object detection model to establish objects and classes present in the camera's field of view. These predictions are then projected onto a range-azimuth map, which is to be used as input to the model.

1.3.3 Computer vision models for radar data

The previously mentioned RAMP-CNN model [5] processes all pairwise combinations of RAD separately. By having three parallel processing stages consisting of convolutional autoencoders (CAEs), each input slice results in a corresponding feature map. The CAEs include batch normalization and uses ReLU activation functions.

The output from the CAEs are used as input in a fusion module, which compresses all feature maps into a single new range-azimuth feature map. Two convolutional layers take the fused features as inputs and perform RAMP-CNNs final recognition decisions, an inception layer followed by a normal 3D convolutional layer. The inception layer was originally proposed as part of GoogLeNet [35].

The three RODnet models [42] are structurally similar to CAEs. The main differences between the three are whether skip connections are implemented, and if inception layers are included in addition to normal convolutions. For the final detection decision, the authors derived a novel form of non-max suppression based on object overlap, to suit their model's output [42].

The model used in [21] is an adapted Single-shot detector for radar data. The backbone used for this model is inspired by, and uses the focal loss function from RetinaNet [18]. As mentioned in Section 1.3.1, they used two different models: one with range-azimuth input, and one with all three pairwise RAD combinations inputs. For the latter, feature maps are repeated along its missing dimension and then all feature maps are concatenated along the channel dimension. The features extracted by the backbone network go through additional convolutional layers, predicting confidence of detection for different classes with sizes close to a predefined size, at each feature location. Several of these pre-defined sizes can be used at each feature location. From the standard SSD, they also use bounding box regression to adapt the pre-defined sizes to better match the actual detections, and non-maximum suppression to remove detections deemed likely to be from the same object.

2 Theory

2.1 Machine learning and deep learning

The aim of this section is to introduce the reader to common machine learning and deep learning concepts and go into more depth in aspects which are of specific relevance to this thesis. The primary sources for this section are Géron's book [6] and Goodfellow et al.'s book [8].

2.1.1 The components of a neural network

Neural networks, or 'artificial neural networks' (ANNs), are learning models heavily inspired by the architecture of neurons. Here, one of the pioneering, and among the simplest, ANN architectures is the 'perceptron' which consists of a numeric input layer, a numeric output layer and weights connecting them [6, pp. 299–304]. For an example, see Figure 1. The perceptron will here be used to lay a foundation for the architectures and machine learning concepts presented in this report.

More specifically, the perceptron has an input layer consisting of a positive number of nodes, $n \in \mathbb{Z}^+$ and an output layer consisting of a positive number of nodes, $m \in \mathbb{Z}^+$. Moreover, each node holds a particular value, such that the input layer can be represented by a vector $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ and the output layer as a vector $\mathbf{h} = (h_1, \ldots, h_m) \in \mathbb{R}^m$. Each input node is then connected to all output nodes through real-valued weights $\mathbf{w}_i = (w_{1i}, \ldots, w_{ni})$, where $i \in \{1, \ldots, m\}$ denotes the index of the output node. The value of each output node, h_i can then be evaluated as a transformation of the result from the affine mapping $z_i = \mathbf{w}_i^\top \mathbf{x} + b_i$ such that

$$h_i = \phi(z_i) = \phi\big(\mathbf{w}_i^\top \mathbf{x} + b_i\big), \tag{1}$$

where $b_i \in \mathbb{R}$ denotes the node's bias and where $\phi : \mathbb{R} \to \mathbb{R}$ denotes the output layer's common activation function. In the case of the perceptron, the activation is traditionally some form of step function [6, pp. 299–304]. More details regarding activation functions are given in Section 2.1.2.

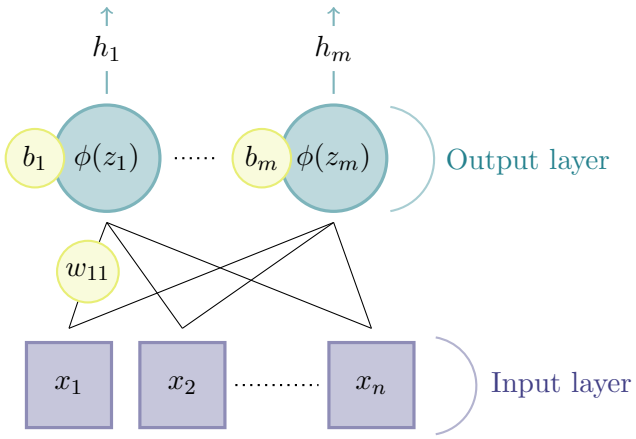

Figure 1: A perceptron consisting of n input nodes and m output nodes. Each node in the input layer is connected to each node in the output layer through weights indicated by the solid black lines. The figure is made with inspiration from [6, p. 304].

The notion of a perceptron can be extended, such that one or many layers are put between the input and output layer. These intermediate layers are often referred to as 'hidden layers', and whenever ANNs contain a large amount of hidden layers, the networks are referred to as 'deep neural networks' [6, p. 309].

2.1.2 The multilayer perceptron and activation functions

One common example of an ANN which contains one or more hidden layers is the 'multilayer perceptron' (MLP), where all layers are fully connected. If another layer is appended to the perceptron's output layer, such that its output layer becomes a hidden layer, the perceptron becomes an MLP, see Figure 2. In other words, the value of a single hidden node, given by (1), is now part of the input to the output layer. Let the new output layer consists of $k \in \mathbb{Z}^+$ nodes. Recall that the output layer in (1) consisted of m nodes, and by letting $j \in \{1, \ldots, k\}$ an element y_j in the output of the network $\mathbf{y} = (y_1, \ldots, y_k)$ can be computed as

$$y_j = \psi\big(\mathbf{w}_j^\top \mathbf{h} + b_j\big),$$

where $\mathbf{h} = (h_1, \ldots, h_m)^\top \in \mathbb{R}^m$ denotes a vector containing the outputs of the hidden layer. Moreover, $\mathbf{w}_j \in \mathbb{R}^m$ is the weights between output node j, and all nodes in the hidden layer and $b_j \in \mathbb{R}$ the bias of output node j. Lastly, $\psi : \mathbb{R} \to \mathbb{R}$ denotes the common activation function of the output layer.

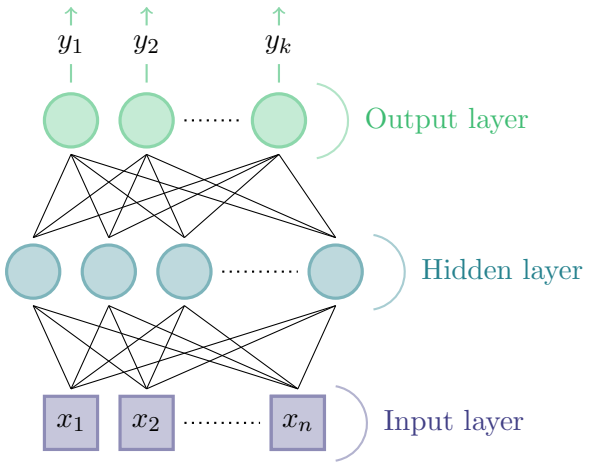

Figure 2: An example of an MLP consisting of three layers. The input layer here consists of n nodes, followed by a hidden layer consisting of m nodes. The hidden layer is, in turn, followed by an output layer consisting of k nodes.

Note that the MLPs (or single perceptrons) do not necessarily have to use step functions for activation. Here follows a few commonly adopted activation functions $f : \mathbb{R} \to \mathbb{R}$ in case of the MLP [6, pp. 312, 366],

$$\text{Recitified linear unit}: \quad \text{ReLU}(z) = \max(0, z) \tag{2}$$

$$\text{Sigmoid}: \quad \sigma(z) = \frac{1}{1 + \exp(-z)}$$

$$\text{Sigmoid linear unit (a.k.a. Swish)}: \quad \text{SiLU}(z) = z \cdot \sigma(z). \tag{3}$$

In the case of multiclass (or 'multinomial') classification, where the number of classes $C \in \mathbb{Z}^+$ is greater than two, it is common to adopt 'softmax regression'. This implies that the activation function for the output layer is given by the softmax function. To explain the function, let $\mathbf{s}(\mathbf{x}) = (\mathbf{s}_1(\mathbf{x}), \ldots, \mathbf{s}_m(\mathbf{x})) \in \mathbb{R}^m$ denote the model output given input $\mathbf{x} \in \mathbb{R}^n$, where the number of nodes in the output layer $m = C$. The softmax function is then given by

$$\text{Softmax}: \quad \sigma(\mathbf{s}(\mathbf{x}))_q = \frac{\exp(\mathbf{s}_q(\mathbf{x}))}{\sum_{j=1}^{m} \exp(\mathbf{s}_j(\mathbf{x}))}$$

for each output node index $q \in \{1, \ldots, m\}$. Consequently, the softmax function performs a normalization of the network's ouput vector $\mathbf{s}(\mathbf{x})$, and it is common to interpret the resulting vector as class probabilities [6, p. 170].

2.2 Training a deep neural network

Let $f_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^m$ denote a feed forward neural network with network parameters $\boldsymbol{\theta}$ such that $\boldsymbol{\theta}$ is a parameterization of the network's weights and biases. Moreover let $\mathbf{X} \in \mathbb{R}^{N \times n}$ be some training data set with targets $\mathbf{Y} \in \mathbb{R}^{N \times m}$, such that each input sample $\mathbf{x}_i \in \mathbf{X}$, $i \in \{1, \ldots, N\}$, has a measured target value associated with it $\mathbf{y}_i \in \mathbf{Y}$. The purpose of such a network is then to mimic the true underlying mapping $f^* : \mathbb{R}^n \to \mathbb{R}^m$ such that

$$f^*(\mathbf{x}_i) = \tilde{\mathbf{y}}_i \approx \mathbf{y}_i \quad \forall i \in \{1, \ldots, N\},$$

where $\tilde{\mathbf{y}}_i \in \mathbb{R}^m$ is the corresponding true value for the given sample [8, p. 164]. In other words, $\mathbf{y}_i$ constitutes a potentially noisy measurement of the actual truth $\tilde{\mathbf{y}}_i$, which is f^* evaluated at $\mathbf{x}_i$. If f consists of many layers, e.g., two hidden layers and one output layer with respective activations f_1, f_2, f_3, it can for a given sample $\mathbf{x}_i$ be viewed as the composite functional mapping

$$f_{\boldsymbol{\theta}}(\mathbf{x}_i) = \big(f_3 \circ f_2 \circ f_1\big)(\mathbf{x_i}).$$

Since the training data solely provides an approximate evaluation of f^* for the samples contained in the training data set, fitting f with respect to $\mathbf{X}$ and $\mathbf{Y}$ implies approximating the true underlying relationship f^* [8, pp. 164–165].

2.2.1 Cost functions and optimization

Now, given a model $f_{\boldsymbol{\theta}}$, there are many ways to tackle the problem of finding good values of $\boldsymbol{\theta}$ with respect to the training data set and the model itself. Typically, by letting $\boldsymbol{\Theta}$ denote the parameter space such that $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, one would define a real-valued cost function (or 'objective') $J : \boldsymbol{\Theta} \to \mathbb{R}$ and try to find its minimizing argument $\boldsymbol{\theta}^*$ [8, p. 151],

$$J(\boldsymbol{\theta}; \mathbf{X}, \mathbf{Y}) = \frac{1}{N} \sum_{i=1}^{N} L(\boldsymbol{\theta}; \mathbf{x}_i, \mathbf{y}_i), \tag{4}$$

where L constitutes a real-valued loss function applied on each sample.

In the case of softmax regression, it is often combined with the cross entropy cost function. To explain cross entropy error, let $\mathbf{X} \in \mathbb{R}^{N \times n}$ and $\mathbf{Y} \in \mathbb{R}^{N \times m}$ comprise a data set, and denote an input sample $\mathbf{x}^{(i)} \in \mathbf{X}$ for each $i \in \{1, \ldots, N\}$. Moreover, let $f_{\boldsymbol{\theta}} : \mathbb{R}^n \times \boldsymbol{\Theta} \to \mathbb{R}^m$ denote a feed forward neural network which uses softmax as the activation function on the output layer. One can then form a class probability prediction $\hat{\mathbf{p}}^{(i)} = (\hat{p}_1^{(i)}, \ldots, \hat{p}_m^{(i)})$, given a sample $\mathbf{x}^{(i)}$. Lastly, let $\mathbf{y}^{(i)} = (y_1^{(i)}, \ldots, y_m^{(i)}) \in \mathbf{Y}$ denote the corresponding target probability, such that $y_k^{(i)} \in \{0, 1\}$ for each $k \in \{1, \ldots, m\}$. The cross entropy error is then given by [6, p. 171]

$$\text{Cross entropy error :} \quad J(\boldsymbol{\theta}; \mathbf{X}, \mathbf{Y}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{m} y_k^{(i)} \log\big(\hat{p}_k^{(i)}\big).$$

Now, minimization of costs functions are typically done with help from an optimizer, e.g., Adam (short for 'Adaptive Moment Estimation'). Briefly, it can be said that Adam utilizes exponential moving averages of the first and second moment of historical gradients to update the network parameters. These moments are, in turn, refering to the estimated mean and uncentered variance of the gradient of the cost function [16]. The method has shown great success, but one of its drawbacks is that it can yield solutions which do not generalize as well as solutions obtained from other methods [14].

2.2.2 Weight initialization and batch normalization

To reduce risks related to vanishing gradients and training being slow, it is common to strategically initialize weights in a layer depending on the choice of activation function, such that the variance of the input to a layer is approximately of the same size as the variance of the layer's output [6, pp. 359–360]. In the case of softmax activation it is then common to adopt Glorot uniform weight intialization.

The method can be explained by letting n_j and n_{j+1} denote the number of nodes in the j:th layer and (j+1):th layer, respectively. Moreover, let $\mathbf{W}^{(j+1)} \in \mathbb{R}^{n_j \times n_{j+1}}$ denote the weight matrix containing all weights between the two layers, such that one of its elements $w_{ik}^{(j+1)}$ comprises the weight connecting node i in the n:th layer with the k:th node in the (n+1):th layer. Glorot uniform initialization then takes the form [7]

$$\mathbf{W}^{(j+1)} \sim \mathcal{U}\Big[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\Big],$$

where $\mathcal{U}$ denotes the uniform distribution. The corresponding bias vector $\mathbf{b}^{(j+1)} = (b_1^{(j+1)}, \dots b_{n+1}^{(n+1)})$ such that $\mathbf{b}^{(j+1)} = \mathbf{0}$.

In the case of ReLU it is common to use He weight initialization [6, p. 360]. The initialization then becomes

$$\mathbf{W}^{(j+1)} \sim \mathcal{N}\big[0, \sqrt{2/n_j}\big],$$

where $\mathcal{N}$ denotes the normal distribution and $\sqrt{2/n_j}$ its standard deviation [11]. In the case of He initialization the bias vector is set to $\mathbf{b}^{(j+1)} = \mathbf{0}$.

Even though clever choices of network initialization mitigate the risk of obtaining vanishing or exploding gradients, there is still a risk that the problem arises during the training phase. One way of further reducing the risk is to adopt a technique referred to as 'batch normalization', which implies zero centering and normalizing the input to a given layer [6, p. 367]. Here follows an explanation of the technique, in line with [6, pp. 367–368].

Let m_B denote the size of a subset B, often referred to as 'mini-batch', of some training data set $\mathbf{X} \in \mathbb{R}^{N \times n}$, such that $B \subseteq \mathbf{X}$. Moreover, let $\mathbf{x}^{(i)} \in \mathbb{R}^n$ denote a sample from the mini-batch which is fed as input to a layer adopting batch normalization. The algorithm then starts by evaluating the input mean vector for the current mini-batch according to

$$\boldsymbol{\mu}_B = \frac{1}{m_B} \sum_{i=1}^{m_b} \mathbf{x}^{(i)}$$

and the variance of the input according to

$$\boldsymbol{\sigma}_B^2 = \frac{1}{m_B} \sum_{i=1}^{m_b} \big(\mathbf{x}^{(i)} - \boldsymbol{\mu}_B\big)^2.$$

Thereafter the zero-centered and normalized input is computed as

$$\hat{\mathbf{x}}^{(i)} = \frac{\mathbf{x}^{(i)} - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2 + \varepsilon}},$$

where ε, normally set to 10^{-5}, is a smoothing term to avoid division by zero. Lastly, the output from the batch normalization is computed as

$$\mathbf{z}^{(i)} = \boldsymbol{\gamma} \otimes \hat{\mathbf{x}}^{(i)} + \boldsymbol{\beta},$$

where $\boldsymbol{\gamma} \in \mathbb{R}^n$ denotes an output scaling vector, $\otimes$ the element-wise product (or 'tensor' product), and $\boldsymbol{\beta} \in \mathbb{R}^n$ an output shift parameter. In some cases, by adding a batch normalization at the very start of the neural network, no data set normalization or rescaling may be needed beforehand [6, p. 367].

2.2.3 Evaluation

This section will present some metrics used to evaluate classifying machine learning models, starting with precision, recall, and F_1 score. These are all based on the true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN) observed in the results.

Precision (P) is defined as the fraction of positive predictions that were correct [6, p. 109],

$$P = \frac{TP}{TP + FP}.$$

Recall (R) is defined as the fraction of positives that were detected [6, p. 110],

$$R = \frac{TP}{TP + FN}.$$

The F_1 score is the harmonic mean of precision and recall, useful when both metrics are to be considered [6, p. 111],

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R}.$$

Mean precision is furthermore not to be confused with average precision. There is typically a trade-off between precision and recall in a machine learning model. Average precision refers to combined value of precision over different values for recall. More specifically, when plotting precision against recall in a 'precision-recall curve', average precision corresponds to the area under the curve. Average precision is calculated for each class and similarly to regular precision, when these values are combined, it is referred to as mean average precision (mAP) [6, p. 529].

Intersection over union (IoU) is a common metric for computer vision models. To explain the metric, let A and B denote two sets. The IoU can then be evaluated as [37],

$$\text{IoU} = \frac{A \cap B}{A \cup B}. \tag{5}$$

In the case of computer vision, the sets A and B in (5) above refers to the geometric area of predictions, for example bounding boxes or segmented areas.

When discussing object detection IoU is considered when determining mAP, and the IoU is typically considered in terms of thresholds. For example mAP50 refers to the mean average precision where IoU has to be at least 50%. Further, this can be calculated as averages for different IoU thresholds. For example, mAP50-95 refers to the average mAP at different thresholds, ranging from 50% to 95% at intervals of 5%.

When comparing models' proficiencies at detecting and classifying different classes another useful metric is the 'confusion matrix' [6, pp. 108–110]. A confusion matrix displays how often a particular class is classified correctly and how often it is classified as the other classes. One of the matrix axes displays the predicted labels and the other the true labels, while the cells hold the number of times a particular predicted-true label pair occurred during testing. Confusion matrices are often normalized over their rows or columns. If normalized over the true label axis, the diagonal where labels correspond to each other will show recall. If normalized over the predicted label axis, this diagonal will show precision.

2.3 DBSCAN Clustering

Clustering comprises unsupervised learning methods used for grouping more similar data points in a data set together. Thus, it concerns algorithms which establish linkage between more similar points and differentiates these points from the remainder of the data set [20].

One method commonly considered when the number of clusters can not be established beforehand is DBSCAN (short for 'Density-based spatial clustering of applications with noise'). The algorithm was presented in 1996 by Ester et al. in [4] and has the benefit of being able to distinguish clusters of any shape and is robust towards outliers. The clustering algorithm is in itself deterministic, but the data labels depend on the order of which data points are presented [15, 32]. DBSCAN, as with many other clustering methods, requires a metric for distance, such as Euclidian or Manhattan distance [4]. An example of DBSCAN clustering is shown in Figure 3.

DBSCAN requires two parameters, $minPts$ and ε. The former refers to the minimal total density in the neighbourhood of a point, for it to be considered a core-point. It is possible to weight each sample, but if no sample weights are specified, computing the neighbourhood density simply amounts to the total number of points. Each region which then contains one or more neighbouring core-points is deemed a 'cluster'. The second parameter, ε, refers to the distance within which two points are considered neighbouring points (referred to as 'directly reachable'). Points which are directly reachable from other core-points are called 'direct density reachable' and are part of the same cluster. If these points happen to not contain at least $minPts$ within the radius ε, they are referred to as 'border-points'. Lastly, points which are not core-points and not direct density reachable from any cluster, are considered noise [32].

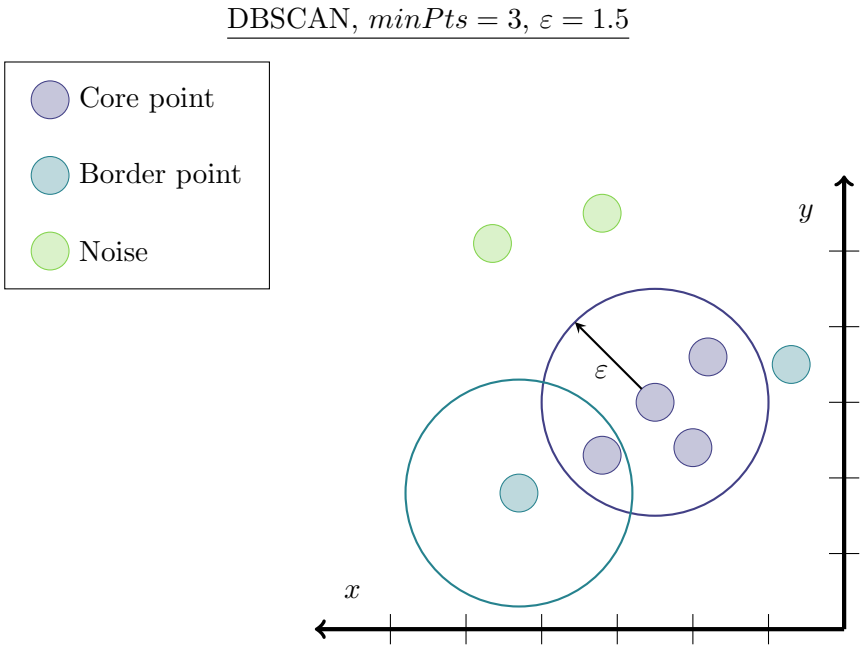

Figure 3: An illustration of the DBSCAN algorithm. In this particular example, the distance metric used is euclidean (as indicated by the circular radii), $minPts = 3$ and $\varepsilon = 1.5$. Two radii are illustrated in the figure, one for a purple core point, and a second for a blue border point. As can be seen from the figure, the established cluster contains four core points and two edge points. The core points are established through counting the number of direct neighbours within the radius of ε. If the number of neighbours is larger or equal to $minPts$, the point is deemed a core point.

2.4 Ensamble learning

Ensamble learning refers to methods which make use of the predictions from multiple predictors and then aggregates a single prediction by either averaging or counting the predictions (essentially voting) from the individual learners [6, p. 211]. As the methods are fairly easy to implement they are easy to try out without any extensive data set preprocessing [6, p. 235].

2.4.1 AdaBoost

AdaBoost (short for 'Adaptive boosting') is an ensamble learning algorithm, which utilizes information from many individual weak learners to make predictions [6, p. 222]. A 'weak learner' refers to an algorithm which achieves slightly better results than by predicting randomly, whereas a 'strong learner' is a high performing algorithm [6, p. 213]. A learner selecting the average of many weak learners can in practice be a strong learner and this is the desired result of training an AdaBoost model [6, pp. 213, 222].

In practice, AdaBoost follows an iterative scheme, where learners are incrementally added to the model. Here, the first step is to train a single learner and evaluate it on the training data. Thereafter, the sample weights are modified, such that missclassified samples contribute more to a consecutive learner. Thus, after the weights are updated, a second learner is trained and evaluated on the training data, after which the weights are updated again and a third learner is trained, and so on [6, p. 223]. The individual learners in AdaBoost can vary, but it is common to adopt so-called 'decision trees' [6, pp. 224–225]. Decision trees are, as the name suggests, learners which works by propagating a sample through a tree, starting at the top node iterating until it reaches a leaf node where a prediction is obtained. Each iteration is then analogous to deciding on which path to take next to propagate through the tree [6, pp. 195–196].

The AdaBoost algorithm can be explained by letting $\mathbf{X} \in \mathbb{R}^{N \times n}$ denote a data set with corresponding ground truth labels $\mathbf{y} \in \mathbb{R}^{N \times 1}$. Also, let $\mathbf{w} = (w^{(1)}, \ldots, w^{(N)}) \in \mathbb{R}^N$ denote a weight vector containing weights for each sample in the training data, and initialize this vector to $w^{(i)} = 1/N$ for each $i \in \{1, \ldots, N\}$. Lastly, let m denote the number of predictors. Then each predictor has a corresponding error rate r_j, $j \in \{1, \ldots, m\}$, which can be evaluted as

$$r_j = \sum_{\substack{i=1 \\ \hat{y}_j^{(i)} \neq y^{(i)}}}^{N} w^{(i)},$$

where $\hat{y}_j^{(i)}$ denotes the j:th predictor's inference on sample $\mathbf{x}^{(i)} \in \mathbf{X}$, and where $y^{(i)} \in \mathbf{y}$ is the corresponding ground truth. Each predictor is then assigned a weight α_j according to

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j}, \tag{6}$$

where $\eta \in \mathbb{R}_{>0}$ is the learning rate, typically set to $\eta = 1$. Next, before another predictor $j+1$ is added, the weights are adjusted according to

$$w^{(i)} = w^{(i)} \exp\big(\alpha_j\big), \quad \forall\, \hat{y}_j^{(i)} \neq y^{(i)}$$

and the weights remain unchanged for all samples where $\hat{y}_j^{(i)} = y^{(i)}$. To form a prediction for sample $\mathbf{x}^{(i)}$, the AdaBoost algorithm simply computes the weighted average of all predictors' outputs according to

$$\hat{y}^{(i)} = \sum_{j=1}^{m} \alpha_j \hat{y}_j^{(i)}.$$

2.4.2 AdaBoost classification and SAMME

To elaborate in the case of multiclass classification, let $K \in \mathbb{Z}^+$ such that $K \geq 2$ denote the number of classes. Moreover, let

$$\mathbf{p}^{(j)} = (p_1^{(j)}, \ldots, p_K^{(j)})$$

denote the normalized softmax output for a given sample and predictor $j \in \{1, \ldots, m\}$. The ensemble prediction $\hat{y}$ can then be computed as

$$\hat{y} = \underset{k \in \{1, \ldots, K\}}{\operatorname{argmax}} \sum_{j=1}^{m} \alpha_j p_k^{(j)}.$$

In the case where $K > 2$, it is common to adopt SAMME (short for 'Stagewise Additive Modeling using a Multi-class Exponential loss function'), which is a method proposed by Zhu et al. in [10], where a slight modification is made to (6) such that

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j} + \log(K - 1). \tag{7}$$

In the case where $K = 2$ the expression in (7) is equal to the one (6), and the algorithm reduces to AdaBoost.

2.5 Convolutional neural networks

Convolutional Neural Networks (CNNs) are commonly applied for image recognition tasks and are artificial neural networks based on the layout of the human visual cortex [6, pp. 479–480].

2.5.1 Convolutional layers

What mainly differentiates CNNs from other networks is that they contain so-called 'convolutional layers', which refer to a special type of layer related to the mathematical operation of convolution [8, p. 326]. An example of a convolutional layer is illustrated in Figure 4. In the discrete case, the convolution operator $*$ takes two functions, $x : \mathbb{Z} \to \mathbb{R}$ and $w : \mathbb{Z} \to \mathbb{R}$, and produces a third function $s : \mathbb{Z} \to \mathbb{R}$. By letting $t \in \mathbb{Z}$ one can formulate a discrete convolution as

$$s(t) = \big(x * w\big)(t) = \sum_{\tau=-\infty}^{\infty} x(\tau)w(t-\tau). \tag{8}$$

In relation to CNNs, x is generally referred to as 'input' and w the 'kernel'. The operation then produces output s, commonly referred to as 'feature map' [8, pp. 326–327].

An image channel comes in the form of a matrix, $I \in \mathbb{R}^{h \times w}$ such that each element $I(i,j) = I_{i,j} \in \mathbb{R}$ for $i \in \{1, \ldots, h\}$ and $j \in \{1, \ldots, w\}$. Since the input data is two dimensional, it makes sense to talk about a two dimensional convolution, where the kernel $K \in \mathbb{R}^{m \times n}$ also is of two dimensions. One possible representation of (8) then becomes [8, pp. 326–329]

$$S(i,j) = \big(K * I\big)(i,j) = \sum_{m} \sum_{n} I(i+m, j+n)K(m,n),$$

such that the value of each cell in the successive feature map then can be computed according to $Y_{i,j} = S(i,j) + b$, where $b \in \mathbb{R}$ denotes the kernel bias.

The kernel K determines the 'receptive field' of nodes in the output of the convolutional layer. The labelled nodes in the input image in Figure 4 constitute the receptive field of the labelled node in the feature map. Each kernel has the same number of trainable parameters as the size of the kernel plus bias [6, pp. 481–482].

CNN is used here to refer to all ANNs with so called convolutional layers as a major part of its architectures. Often these are combined with other modules such as an MLP tail for final decision making. However, it is possible to construct a model with only convolutional layers, called a fully convolutional network (FCN), for an example see U-Net described in Section 2.7.2.

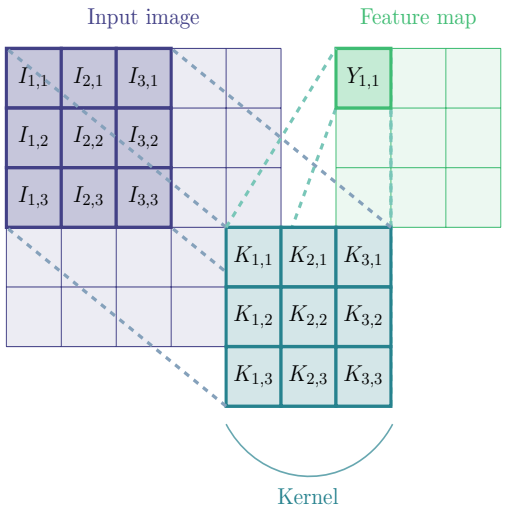

Figure 4: A forward pass through a convolutional layer in a CNN consisting of a single kernel $K \in \mathbb{R}^{3\times3}$. The number of kernels used will determine the feature depth of the output (i.e., the number of feature maps).

2.5.2 Padding and stride

In order to manipulate the sizes of input and output layers of a convolution 'zero padding' and 'stride' are often used. The former refers to adding zero-valued pixels around the input image, such that edge pixels can be treated as inner pixels. Stride, on the other hand, refers to the step size of the kernel before a forward pass of the cells in the receptive field is made, see Figure 5. In this way, the resulting feature map can be given a smaller dimensionality [6, p. 482].

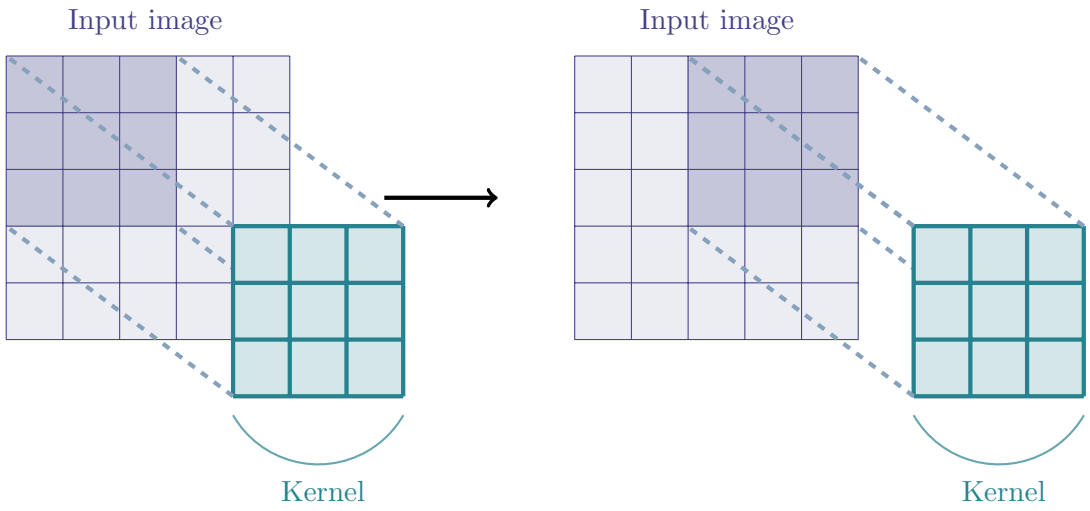

Figure 5: Illustration of stride of two. This implies that the kernel moves two cells before performing a forward pass of the cells in its receptive field. In this case, given that the both the horizontal and vertical stride is two and input is of size 5×5, the resulting feature map would be of size 2×2.

2.5.3 Pooling layers

Another layer type which is common in CNNs is the pooling layer. The aim of using a pooling layer is to downsample the input and make subsequent operations lighter. Like in convolutional layers, a neuron of a pooling layer is connected to a subset of the previous layer's output, within a receptive field. They are also similar to convolutional layers in that a pooling layer has a kernel size, a stride and potentially a padding. A common example of pooling is max pooling, which means that an output node simply assumes the max value of its receptive field. Another example of a pooling layer is average pooling, which as the name suggests averages the receptive field. The fact that the operation of a pooling layer is specified beforehand implies that such a layer does not contain any trainable parameters in contrast to regular convolutional layers. Apart from downsampling, max pooling can also give a CNN some degree of translational invariance, meaning it aids in keeping performance the same even with inputs shifted or translated in some way. However, like all pooling the downside is that a lot of data is neglected, which could potentially be useful in detection and classification tasks [6, pp. 491–493].

2.6 Regularization

Regularization refers to a set of methods which are used to avoid overfitting to a model's data set [6, pp. 392–393]. Overfitting, in turn, is when a model overgeneralizes based on its data set, such that the model performs well on its training data, but fails to generalize and perform when inferring on new data [6, pp. 30–31]. This is especially problematic for deep learning models since they are typically very large and contain many trainable parameters [6, p. 393], which makes them prone to overfitting. Simple solutions for the overfitting problem includes simplifying and shrinking the model, increasing the size of the training data set, or reducing the noise in the data [6, p. 31], but other methods will also be presented in this section.

2.6.1 Holdout and early stopping

One easy way to allow for regularization methods (and testing or validation more generally) to be implemented is to use data set holdout. This simply refers to splitting the data set into at least two, but often more, segments to be used for different purposes. For example, a common approach is splitting a data set into one training set containing about 80% of the data and a testing or validation set containing 20% [6, p. 34].

One way in which this can be utilized is through early stopping. This refers to training a model using a training set, evaluating it against a validation set, and deciding what model to keep based on the loss calculated from the validation set. Sometimes, early stopping also includes a maximum allowed number of epochs without any improvement. If a model for example does not improve its validation loss for five consecutive epochs, training is terminated and the last saved model is kept [6, p. 162].

2.6.2 Weight decay

One common approach for reducing overfitting is the incorporation of what is referred to as 'weight decay'. This simply implies that all weights are multiplied with a scalar $\alpha \in (0, 1)$ for each iteration, where e.g., $\alpha = 0.99$ can be used. In this way, the size of the network parameters can be regularized during training [6, p. 386].

2.6.3 Data augmentation

Data set augmentation refers to artificially increasing the size of the training data set by generating realistic variations on the existing training data. It helps avoid overfitting, such that if the training set is larger and more varied, it is more difficult to overgeneralize based on it. In the case of image classification or detection, common methods include randomly cropping and rotating the images used for training, to make the model better at detecting the objects in a bigger range of positions [6, p. 500].

2.7 Computer vision

Computer vision is a broad field of study within machine learning. For this study, three key ideas are of interest. The first is 'classification', meaning assigning a class to an object.

The second is 'object detection', the computer vision task of detecting instances of predefined classes of objects in digital images [45]. The goal of object detection is simply put to teach a model to answer the question of what objects are where. For the purposes of this report, object detection refers, as just suggested, to both object localization and classification, which can be done as individual processes or simultaneously. Localization can take different forms, such as placing bounding boxes around detections or marking midpoints.

Finally, there is 'semantic segmentation', which refers to classifying each individual pixel in an image. This can allow for more exact predictions, but does not equal object detection as described above, since two overlapping objects are not distinguished when classification is pixel-wise.

2.7.1 YOLO

The YOLO (short for 'You Only Look Once') suite of models was first introduced in 2016, when Redmon et al. tried to tackle the issue of other object detectors models being too slow to run on real-time systems. The model takes an entire image as input and produces bounding boxes and class predictions for objects it detects simultaneously, hence the name [28].

The original model consists of 24 convolutional layers and a two-layer MLP for final decisions [28]. Since then, the YOLO family of models have been through several iterations and improvements, and when selecting models for this study the latest iteration was YOLOv8 from Ultralytics [39]. It is a state-of-the-art model that builds and improves on

its predecessors. The YOLOv8 model itself consists of five variations with increasing size and this study utilized the smallest one, YOLOv8n [38]. It has 3.2 million parameters and require 8.7 billion floating point operations (FLOPs).

At the time of writing, no official paper has been published on YOLOv8. The authors of [2] included a figure of the YOLOv8n architecture in their study shown in Figure 6 below.

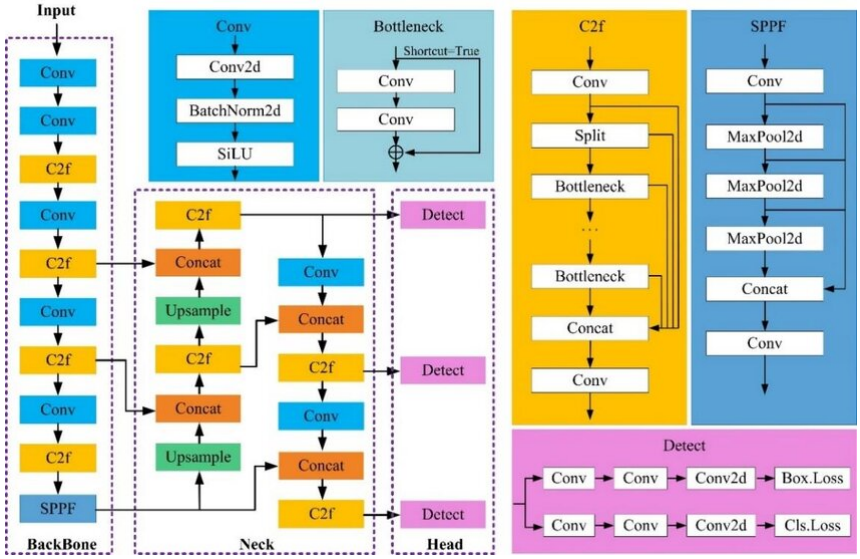

Figure 6: The YOLOv8n architecture, taken from [2].

The backbone (consisting of convolutional, C2f and SPPF modules) is used for feature extraction. Every convolutional layer is followed by a batch normalization layer and a SiLU activation function, see (3) [2]. The C2f module ('cross-stage partial bottleneck with two convolutions') combines high-level features with contextual information [37]. The SPPF module ('spatial pyramid pooling fast') lightens computational load by pooling features to a fixed-size map [2]. The neck of the model is mainly used to fuse differently dimensioned features. Finally, YOLOv8 uses three decoupled heads to predict objectness, classification, and bounding box placement tasks separately, allowing each branch to specialize. Each one uses a combination of CIoU (an IoU based loss function) and distributed focal loss for determining bounding box losses and binary cross entropy for classification loss [2].

2.7.2 U-Net

U-Net is a computer vision model that performs semantic segmentation, originally developed for medical imaging applications [30]. The model is described by the authors as consisting of two paths, a contracting path and an expansive path. The model first performs a series of 3×3 non-padded convolutional operations, all followed by ReLU activation functions, see (2), and every third feature map is followed by a 2×2 max pooling with a stride of two for downsampling. This contracting path corresponds to a quite typical module of convolutional architectures meant to extract features and their context from the image. A step in the expansive path starts with a convolution halving the number of feature channels (what the authors refer to as 'up-convolutions') and concatenating the

output of this operation with the feature map from the corresponding skip connection. A skip connection passes higher dimension, more shallow outputs to later parts of the network, illustrated by the gray arrows in Figure 7. Skip connections allow data to be passed between earlier convolutional layers and later transposed convolutional layers. For example, the output from the first encoding block is passed to both the second encoding block, and the final decoding block. Afterwards the operation is similar to the contracting path with 3×3 convolutions followed by ReLU activation functions. The final detection is done with a 1×1 convolutional layer, making the U-net a FCN [30].

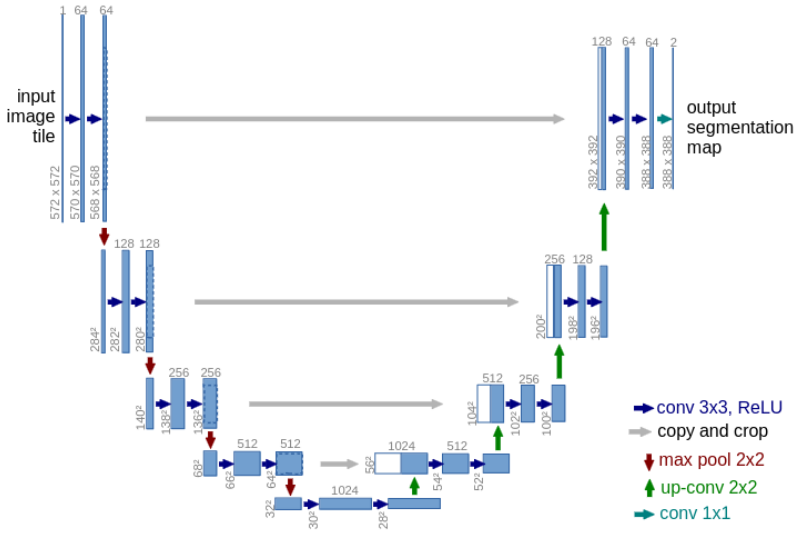

Figure 7: The original U-Net architecture from [30].

The U-Net architecture is shown in Figure 7. The blue boxes in the figure correspond to feature maps, with the number of channels written on top and the size in the lower left corner. White boxes represent feature maps passed through skip connections, and arrows the different operations performed by U-Net. U-Net adopts a pixel-wise softmax function for activation and a cross entropy cost function [30].

2.7.3 Pinhole Camera Projection Model

There are many ways of representing 3D world objects in a 2D image [36, pp. 51–52]. When objects are captured by a camera, the incoming rays are scaled and contingent on perspective distortion. This implies that some geometries in the real world may be subject to changes, e.g., parallel lines no longer being parallel, but rather converging towards each other in a vanishing point. The transformation of the 3D world onto a camera sensor is called 'perspective projection' [3, p. 500].

A commonly applied approximation of the perspective projection, is the 'pinhole camera projection model', which takes two factors into account, i.e., the intrinsic parameters of the sensor and the extrinsic parameters of the environment [3, 26, pp. 585–586, 515–516]. The former refers to the specifics related to the camera itself and includes parameters such as focal length, skew and image center location [36, p. 57]. The extrinsic parameters

instead refer to the camera sensor position and orientation in relation to the global frame, e.g., where the camera is located in a room [3, pp. 585–586]. The entire transformation can then be broken down into a combination of an external and an internal transform [26, pp. 518–519]. Here, the external transform can be viewed as a transformation of an arbitrary point in the global frame,

$$\mathbf{P}_W = \big(X_W, Y_W, Z_W\big)$$

to a point in the coordinate system of the camera,

$$\mathbf{P}_C = \big(X_C, Y_C, Z_C\big).$$

Note that this transformation can be decomposed into a rotational component $\mathbf{R}$, as well as a translational component $\mathbf{t}$, [26, p. 518]

$$\mathbf{P}_C = \mathbf{R}(\mathbf{P}_W - \mathbf{t})$$

By introducing homogeneous coordinates, the expression can be represented as a single matrix multiplication,

$$\mathbf{P}_C = \begin{bmatrix} \mathbf{R} & -\mathbf{R}\mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{P}_W.$$

Thereafter, the coordinates of the camera $\mathbf{P}_C$ can be converted to corresponding image plane coordinates, $\mathbf{x}_I = [u_i, v_i]$, where f denotes the focal length,

$$u_i = \frac{fX_C}{Z_C}, \quad v_i = \frac{fY_C}{Z_C}$$

which, in turn, can be expressed as pixels via knowledge of the sensor's principle point $\mathbf{x}_0 = [u_0, v_0]$ and the distance between two pixel centers as $\mu = [\mu_x, \mu_y]$,

$$u = \frac{f}{\mu_x} u_i + u_0, \quad v = \frac{f}{\mu_y} v_i + v_0.$$

This expression can be rewritten using homogeneous coordinates, according to

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f/\mu_x & \gamma & u_0 \\ 0 & f/\mu_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{x}_I \\ 1 \end{bmatrix}$$

where γ constitutes the skewness between x and y axis. The entire transform can then be defined as,

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R} & -\mathbf{Rt} \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{P}_W.$$

2.8 FMCW Radar

The foundational functionality of a radar is the transmission and reception of an electromagnetic signal which reflects off of the radar's surroundings, thereby providing information on the position and velocity of targets in the field of view. FMCW (frequency modulated continuous wave) radars use signals with frequencies increasing linearly with time, called chirps [23]. A chirp is defined by its starting frequency f_0, bandwidth B, and duration T_c, sometimes called modulation time. These parameters are illustrated in Figure 8 below. The slope

$$S = \frac{B}{T_c}$$

describes the rate of change of the frequency. Having a starting frequency linearly increasing with time over a certain bandwidth essentially gives the signal reference points with which a returning signal can be compared. The instantaneous frequency $f(t)$ for a chirp varies in time $t \geq t_0 \geq 0$ according to,

$$f(t) = f_0 + S(t - t_0).$$

The phase of the signal over time $\phi(t)$ for a chirp can then be calculated by integrating $\phi'(t) = 2\pi f(t)$ according to

$$\phi(t) = \phi_0 + 2\pi \int_{t_0}^{t} f(\tau)d\tau = \phi_0 + 2\pi\big((f_0 - St_0)(t - t_0) + \frac{S}{2}(t^2 - t_0^2)\big),$$

where ϕ_0 is the initial phase [23]. Finally, this allows the definition of the chirp transmission signal x_{TX}.

This signal depends on the phase according to $x_{TX} = A\sin(\phi(t))$, where A is an amplitude constant [23], such that

$$x_{TX}(t) = A\sin\Big(\phi_0 + 2\pi\big((f_0 - St_0)(t - t_0) + \frac{S}{2}(t^2 - t_0^2)\big)\Big).$$

The chirps are transmitted, reflected by surfaces in the radar's field of view, and then received again. Note that all objects detected by the FMCW radar will generate a received signal, so one transmission can have multiple receptions, and these received signals are delayed close replications of the transmitted signal [23]. Figure 9 below illustrates the frequencies of a transmitted and received chirp (one target). The approximately constant

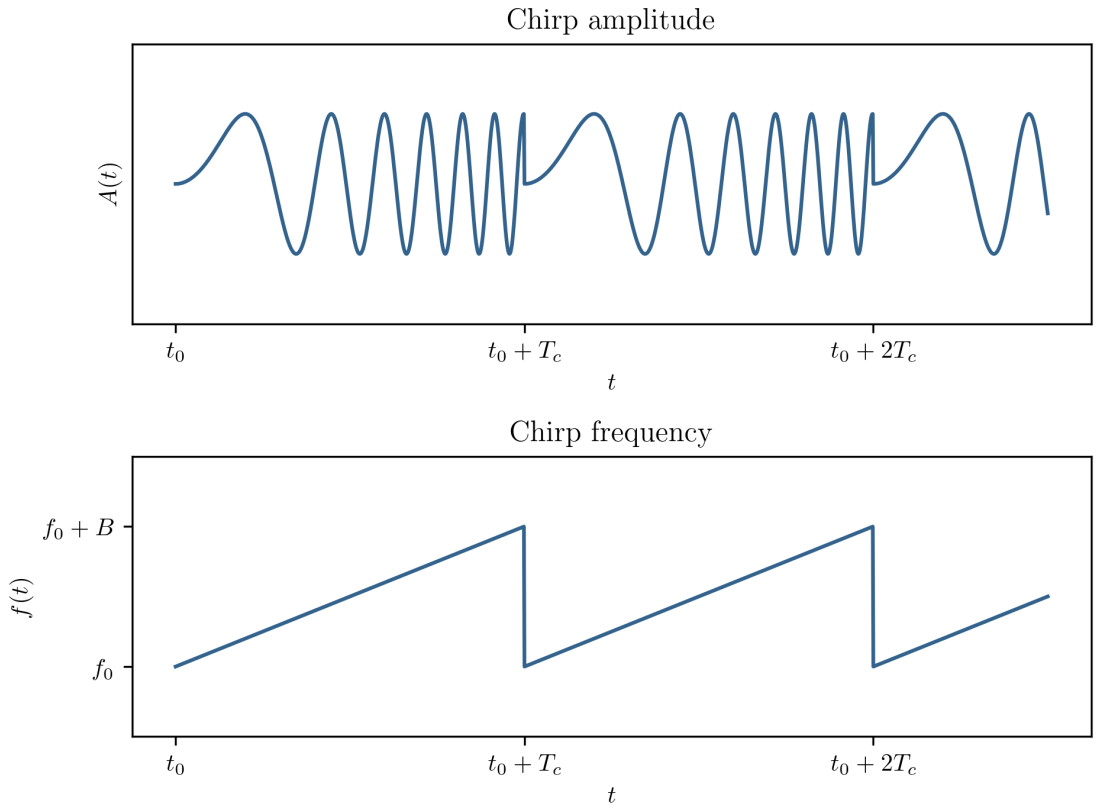

Figure 8: An example of FMCW chirps illustrating amplitude $A(t)$ and frequency $f(t)$ in relation to time t in the upper and lower subplots, respectively. The graphs also illustrates the relationships between the chirp parameters; starting frequency f_0, bandwidth B, and duration T_c. The slope of the frequency diagram during a chirp is S.

difference in frequency is called the 'beat frequency', denoted f_b, which also can be referred to as the 'intermittent frequency' (or IF-signal) [23].

Since the received chirp x_{RX} is a delayed replica of the transmitted chirp x_{TX}, it can be defined as $x_{RX}(t) = x_{TX}(t - \Delta t)$. Note that this is only an exact equality if the target is not moving, and is otherwise only approximately true. The transmitted and received signals then get combined into a mixed signal $x_m(t)$ according to

$$x_m(t) = x_{TX}(t)x_{RX}(t).$$

Because x_m is a product of two trigonometric functions, it will have a component based on the combined frequencies of the transmitted and received signals, and one based on their difference, according to

$$\sin(a)\sin(b) = \frac{1}{2}(\cos(a-b) - \cos(a+b)).$$

It is the latter that will have a constant frequency called the beat frequency f_b. The former

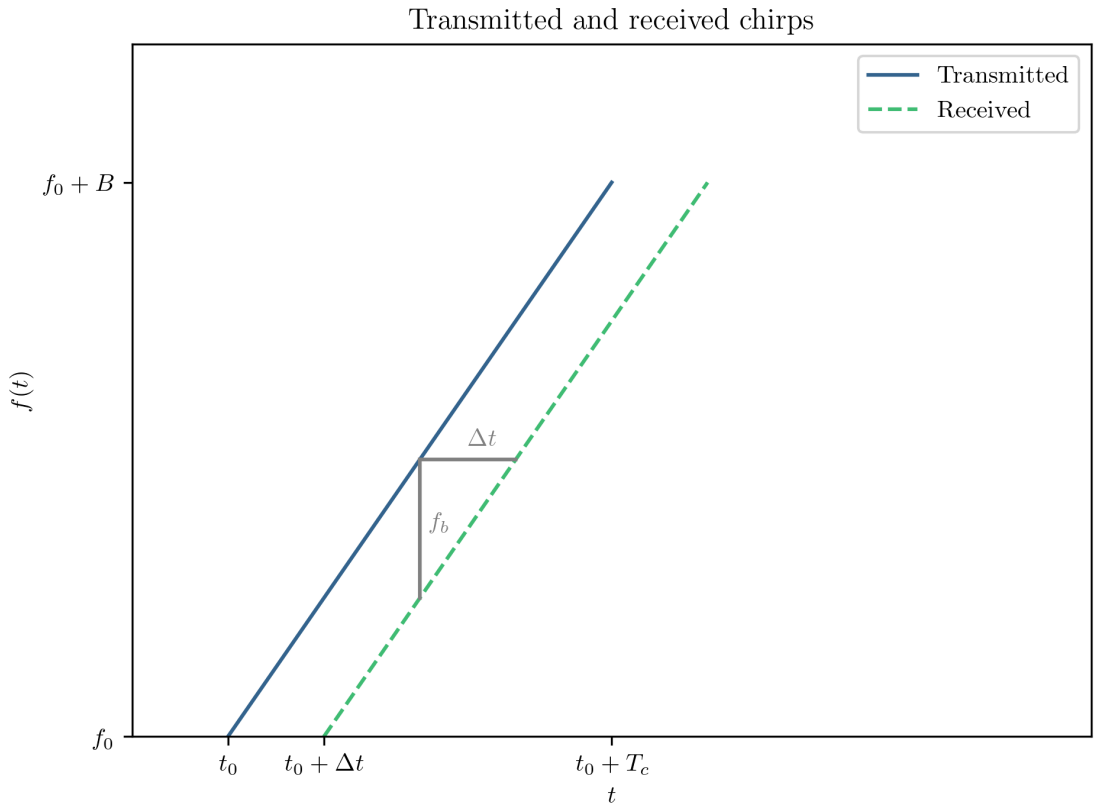

Figure 9: Instantaneous frequency $f(t)$ over time for a transmitted and received chirp with one target. Here, Δt denotes the time of travel and f_b the beat frequency.

is of very high frequency and can therefore be removed through filtering. Thus, it is not considered when referring to interpreting frequencies of x_m.

The mixed signal of multiple chirps are sampled using multiple antennae, resulting in a three-dimensional data structure, consisting of sample number, chirp number and antenna number, respectively [13]. See Figure 10 for reference.

Fourier transformations are used to convert functions in the time dimension to frequencies, and is therefore a popular method of processing the mixed signal x_m [23]. Since x_m is sampled and therefore discrete, the transformation used is a discrete Fourier transform (DFT), and more specifically, it is often a fast Fourier transform (FFT) [23]. The radar output data illustrated in Figure 10 is preprocessed into range, azimuth, and Doppler data, i.e., information on targets' distance to the radar, relative angular position, and radial velocity. How these are processed and calculated are the topics for the following sections.

2.8.1 Range estimation

Performing an FFT along the time sampling axis of the data will yield information about target distances [13]. The mixed signal x_m is as mentioned the difference between the frequencies of the transmitted and received chirps. Assuming the received signal x_{RX}

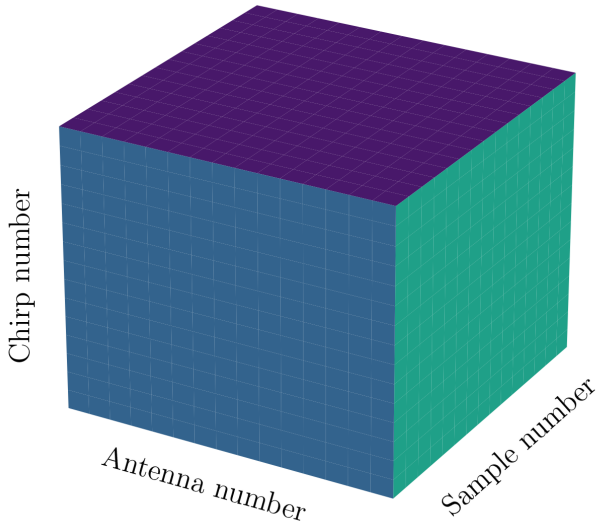

Figure 10: Three dimensional radar output data. The mixed signal is sampled by multiple antennae, multiple times per chirp to create measurements over all three axes.

is an exact, but time delayed, replica of the transmitted signal x_{TX}, the mixed signal's frequency or the beat frequency f_b will be directly proportional to this time delay [23]. The time delay Δt can be expressed as the time it takes a radar signal to travel the distance r twice, divided by the speed of light c, according to

$$\Delta t = \frac{2r}{c}.$$

As can be seen in Figure 9, the beat frequency f_b will depend not only on the time delay Δt, but the slope S, which gives the final formula [23],

$$f_b = \frac{2Sr}{c} \quad \Longleftrightarrow \quad r = \frac{f_b c}{2S}$$

To conclude, when sampling over a single chirp's duration T_c, the FFT along the sampling dimension will yield a spectrum whose main component is the beat frequency f_b, from which the range r can be calculated.

2.8.2 Azimuth estimation

Azimuth is the horizontal angular position of a target relative to radar's boresight, meaning the center of its field of view, see Figure 11 a). To estimate the azimuth angle, at least

two receiving antennas are required [27]. The angular position of the target is dependent on the distances between it and different antennae, and the distance between antennae. [12]. See Figure 11 b) for an illustration of these concepts.

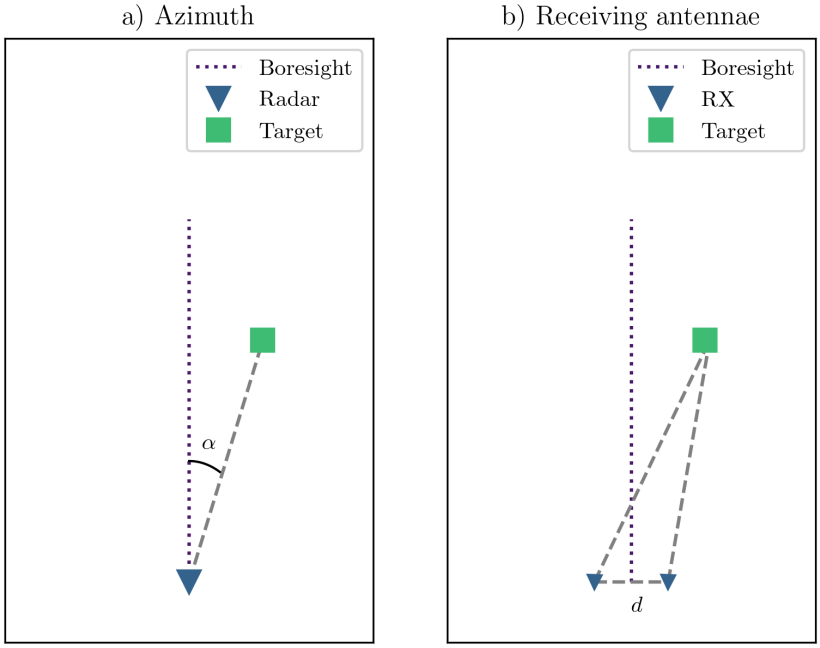

Figure 11: a) Top-down view of a radar with boresight marked, a single target and its azimuth angle α. b) Two receiving antennae (RX) and the distance between them d.

Similarly to determining f_b from a FFT along the sampling axis, the phase difference $\Delta\Phi$ between antennae is the basis for azimuth estimations, determined by performing the FFT along the antenna number axis of the three-dimensional radar output. Since the returning signal has to travel further to reach the left antenna in Figure 11 b) and therefore has a slightly shifted phase, the azimuth angle α can be obtained from [27],

$$\Delta\Phi = \frac{2\pi d \sin\alpha}{\lambda} \quad \Longleftrightarrow \quad \alpha = \sin^{-1}\left(\frac{\lambda\Delta\Phi}{2\pi d}\right)$$

where λ is the wave length and $d\sin\alpha$ describes the difference in range. Note that the term $d\sin\alpha$ equals the additional distance a signal has to travel to reach the further of the two antennas, separated by the distance d. The more antennae that are available, the better the angular resolution will be [13]. If the antennae are separated vertically, elevation angles of targets can be estimated in the same way, i.e., the vertical angular positions.

2.8.3 Velocity and Doppler estimation

Finally, from the FMCW radar data targets' radial velocity can be estimated. Sometimes these calculations are referred to as describing a target's 'Doppler', which will at times be used in this report as well. In this context, Doppler refers to the phase shift caused by a target's movements, which as will be shown is directly related to their radial velocity.

To determine radial velocity, at least two chirps separated by Δt have to be measured [12]. If the object is moving towards or away from the radar, this will cause the phase of the received signal to shift slightly between chirps. The phase difference corresponds to a motion in the target of $v \cdot \Delta t$. Each chirp is processed through a range-FFT according to Section 2.8.1 to determine its range. The range-FFTs for different chirps will have peaks in the same locations, but with shifted phases corresponding to their movements according to,

$$\Delta\Phi = \frac{4\pi v \Delta t}{\lambda} \quad \Longleftrightarrow \quad v = \frac{\lambda \Delta\Phi}{4\pi \Delta t}$$

where v is the radial velocity [12].

2.8.4 Radar cross section (RCS)

A measurement that will be of interest in this study is referred to as radar cross section (RCS). RCS is meant to describe reflectivity and not take into account the power of the transmitter, sensitivity of the receiver, or distance between the radar or the target. In other words, it is a measure of the ineherent reflective strength of a target. RCS can be defined as [29],

$$\text{RCS} = \lim_{r \to \infty} 4\pi r^2 \frac{P^{scat}}{P^{inc}}$$

where P^{inc} and P^{scat} are the power densities of the radar signal before and after reflecting the target (incident and scatter, respectively) and r is the range.

3 Method

3.1 Data set

The data for this study consists of radar data in the form of range-Doppler images, and synchronized camera images, together referred to as a frame (or a radar frame and camera frame, respectively). A range-Doppler image will at times be referred to as a range-Doppler map, if a mathematical matrix representation is emphasized. Finally, a frame (whether camera or range-Doppler) refers to an image and the corresponding metadata, such as timestamps. The frames are synchronized simply by comparing timestamps in the metadata of both the camera images and radar images. The radar data is what is used to train the models, while the corresponding camera images are only used in the annotation process, explained further in Section 3.2. There is another important distinction to make between the data used for the annotation process and the training process. The radar used in this thesis outputs data that is more processed than needed for the purpose, including range-Doppler data, angular positions (both azimuth and elevation), and more that is not considered simply preprocessed within the context of this thesis. During the annotation process, the fully refined radar data is used, but when training the models much of this data is withheld, to mimic preprocessed radar data.

The range-Doppler maps in this study are $n \times n \times 3$ tensors with discrete range and Doppler bins along the two first axes, with the last axis of size three corresponding to channel. The three channels are signal strength, RCS and detection mask. A detection mask is simply an $n \times n$ range-Doppler image with only 0's indicating background and 1's indicating detections. The detection mask is created from the signal strength channel of the range-Doppler map, which is processed such so that values below a certain threshold are reduced to 0, and all values above it are set to 1. An example of a range-Doppler image showing only signal strength (the first channel) can be found in Figure 17.

Using signal strength and RCS was motivated by a feature importance analysis which will be presented in Section 3.4. Including the detection mask as a channel was motivated by a speculation that this could aid in the models learning the unique shapes, or micro-Doppler signatures, of the classes. No formal results comparing the usage of different channels and representations will be presented in this thesis, but all three data channels seems to have improved the results.

The data is labelled differently depending on the model. The different models used either expect bounding boxes (in terms of pixel coordinates) or segmentations. When training YOLO, the only model used that expects bounding boxes, the range-Doppler maps are saved as three channel PNG-files ($n \times n \times 3$), rather than two-dimensional arrays, and the annotations are saved as separate text files containing bounding box classes and coordinates. This is the data set form expected by Ultralytics when training YOLOv8 [41].

U-Net which performs semantic segmentation and AdaBoost which classifies clusters of detections, requires a different labelling. For both a down-sampled and one-hot encoded range-Doppler map is provided as ground truth with the dimensions $m \times m \times c$, where $m < n$ and $c = 4$ is the number of classes including background.

Two different data sets were collected for this study, one for training and one for testing, consisting of 5097 and 855 frames, respectively. These data set sizes refers to the annotated and filtered but not augmented data sets. 20% (1019 frames) of the training data was withheld as a validation data set. After augmentation, described later in Section 3.3, the training split consisted of a total of 12234 frames. The class distributions, both instance- and pixel-wise, are displayed in Figure 12.

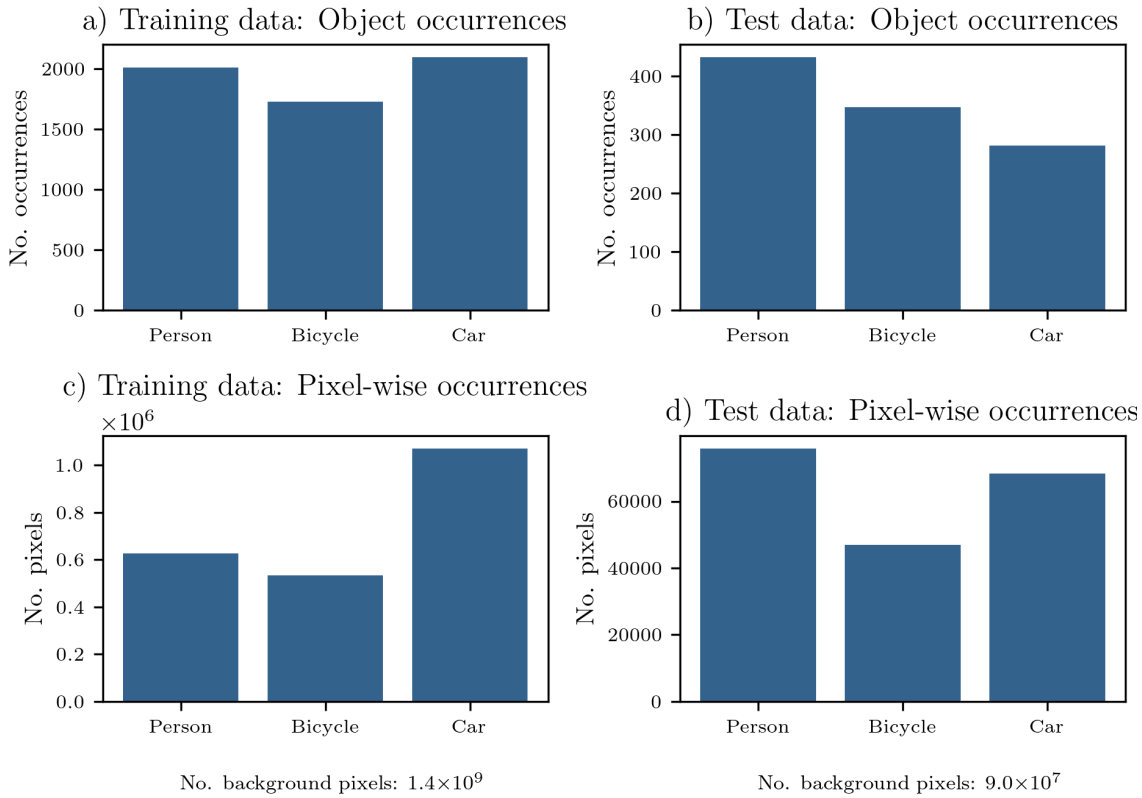

Figure 12: The class distributions of the data sets. Note that these data sets refer to the data after annotation (including filtering), but before augmentation. a) and b) shows the object occurrences in the training and test data sets. These distributions show the number of object instances, i.e., the number of bounding boxes in terms of ground truth labels. c) and d) shows the corresponding pixel-wise occurrences, i.e., the number of labelled pixels in the $m \times m$ ground truth masks. Underneath both c) and d) the number of background pixels are also stated for each data set.

3.1.1 Data collection

When collecting the data, the radar and camera were both mounted on a height of approximately two meters. Both were mounted with a very slight downward tilt. Some specifications of the radar used for recording are shown in Table 1.

The data for this study was recorded at three different locations, and several scenes were recorded at each location. Scene here refers to the specific position and field of view of the recording. In total, there were 38 recordings for the training data set, and 8 recordings for the testing data set. The recordings varied in length, but were generally around 1000

frames before annotation and filtering. As mentioned in the previous section, the final data sets contained 5097 and 855 frames of training and testing data, respectively. This means each recording contributed on average approximately 130 frames to the data sets.

During the recordings, data of the various classes were recorded at different ranges and velocities. The person class was recorded and annotated up to 43 m range and with varying states such as: standing almost still, walking slowly, walking faster and running. The bicycle class was recorded and annotated up to 44 m, and this class also at varying paces, e.g., biking slowly (almost not moving) and biking at a high speed. Finally, the car class was recorded and annotated up to 62 meters range, with a maximum velocity around 40 km/h.

Table 1: The specifications of the FMCW radar used.

Parameter	**Value**
Range resolution	0.841 [m]
Maximum range	121.153 [m]
Velocity resolution	0.098 [m/s]
Maximum velocity	16.084 [m/s]

3.2 Data annotation

This section will outline and describe the annotation method used for this study. The aim was to develop an annotation framework that is automated to the highest degree possible. The framework is outlined in Figure 13 below.

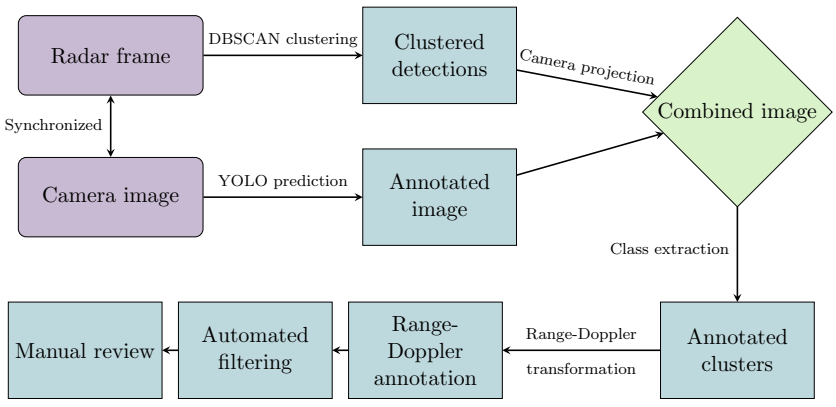

Figure 13: Overview of the annotation framework. The first step involves collecting synchronized camera and radar data. The camera image then goes through a YOLO prediction stage, where objects contained in the image are labeled. Separately, the obtained radar detections are clustered in three-dimensional cartesian space. After this stage, the radar clusters are projected onto the annotated camera image and mapped to YOLO's predictions, such that they either obtain a class or not. After this step, all radar clusters are projected onto the range-Doppler space. Thereafter, a filtering algorithm determines if the frame should be kept or not. Finally, all frames from a recording go through a manual review.

3.2.1 Automatic annotation

The annotation starts with letting a pretrained YOLOv8 model make predictions on all the camera images in the data set, see Figure 14. For a more detailed description of YOLO, see Section 2.7.1. The relevant classes are all available for YOLO predictions, and all other classes predicted by the model are ignored. One complication arose in that YOLO predicts bicycles as objects separate from the people riding them, while the class 'bicycle' for the purposes of this study represents a moving cyclist. Therefore, YOLO predicts both a person and a bicycle overlapping with each other, these predictions are merged, and labelled as a single bicycle. At certain ranges, YOLO quite consistently detects the person riding the bicycle and not the bicycle itself. For these cases, all detected persons were labelled as bicycles, and it was ensured that no pedestrians were present in the same recording.

Since the models used in this study are based on range-Doppler images, completely stationary objects were not of interest. However, YOLO still detects them. For this reason, a blurring function was developed to mask for example stationary cars, that would be detected by YOLO, but not be represented in a range-Doppler image the same scene. While blurring an object makes YOLO unable to detect objects moving directly in front of them, these exceptions are handled during the automated filtering, described later in this section.

Figure 14: An example of a camera image from the data set, before and after YOLO-predictions are applied.

When the images are annotated by YOLO, the next step is clustering the radar detections. For this step the radar detections are represented in three-dimensional cartesian space. The clustering is done by the DBSCAN algorithm, described in Section 2.3. The distance metric used was euclidean, with $minPts = 15$ and $\varepsilon = 0.75$. Whenever cars were present in the recording these values were adjusted to $minPts = 20$ and $\varepsilon = 0.6$, to adapt to the larger number of detections. These particular values were chosen through manual evaluation. A two-dimensional example of DBSCAN clustering is shown in Figure 15.

By clustering, whether radar detections were considered part of the same object (or no object) are determined. In the second plot of Figure 15 two clusters are highlighted in individual colors, and the rest of the detections remain gray, illustrating that they are categorized as noise.

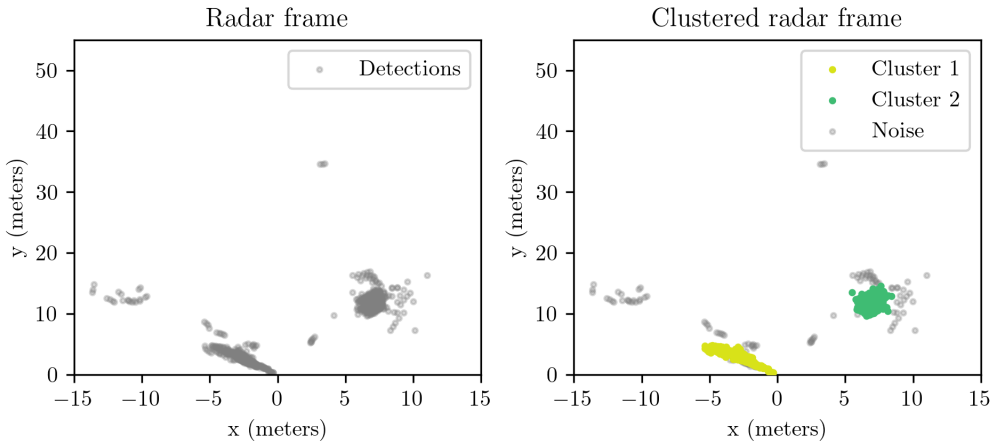

Figure 15: Radar detections in two-dimensional cartesian coordinates, top-down view. The figures show the same scene and frame as in Figure 14, before and after DBSCAN clustering. The radar is positioned at (0, 0).

So far, it has been described how the annotation framework detects objects in view of the camera, as well as what radar detections are considered to belong to the same object. The next step is projecting the clustered radar detections onto the camera image, as shown in Figure 16. See Section 2.7.3 for a more detailed description of how these projections were made. Now, clustered radar detections and YOLO detections are present in the same space, and they can be mapped to each other. The mapping is based on the shortest distance between the center of a cluster and the midpoint of a YOLO bounding-box, simply measured in the euclidean pixel distance. To remove the most extreme outliers immediately, a threshold of 200 pixels was set for the mapping, meaning clusters and bounding boxes further apart are not mapped. In the example in Figure 16, it is quite clear that the green points (cluster 2) corresponds to the bicycle, and that the yellow points (cluster 1) corresponds to the person. However, the mapping is not always this straight forward, which will be discussed more in the next section. The important thing is that the clusters, which are considered as part of the same object, are mapped to a YOLO detection, whose class can be extracted.

Figure 16: Combined image consisting of the YOLO annotation in Figure 14 and the projected clusters from Figure 15.

Now that a class for each radar cluster has been extracted, the final step is transforming these radar detections from being represented in three-dimensional space to a range-Doppler data representation. If the annotation is being performed for a bounding boxing model such as YOLO, a bounding box is determined from the minima and maxima of the clusters' range and Doppler values.

When the annotation is for a semantic segmentation model like U-Net, the individual range-Doppler bins are labelled with their corresponding class. In this case, a 'flood-filling' algorithm is also used. Flood-filling starts by comparing the range-Doppler bins labelled by a cluster with a detection mask. If there exists bins that are not labelled, included in the detection mask, and adjacent to labelled bins, they are also labelled as part of the same class. This process is repeated until no such bins are present. This process ensures that the full detection masks are labelled, and acts as a complement to the clustering.

Figure 17 shows the regular range-Doppler image, the bounding boxed range-Doppler image, and the segmented range-Doppler image (including YOLO's confidence score for the two latter) for the same frame as the previous few figures.

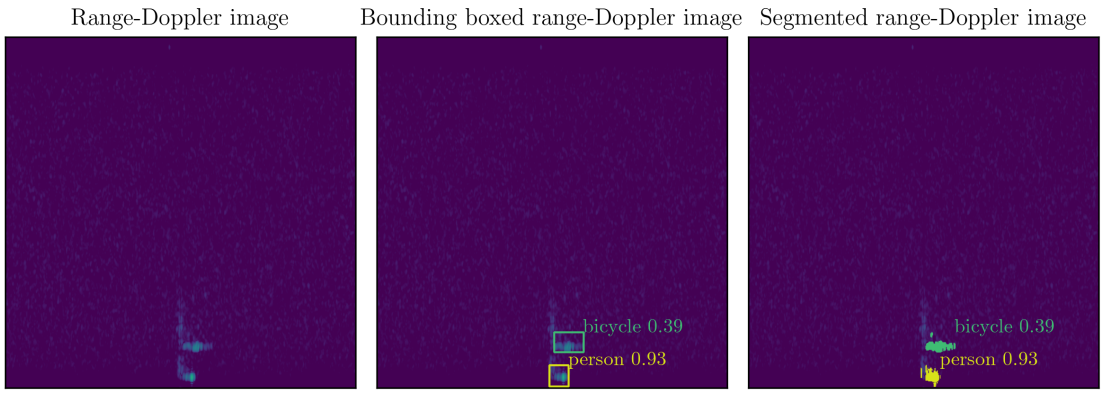

Figure 17: A pre-annotation range-Doppler image, a bounding boxed range-Doppler image, and a segmented range-Doppler image, all from the same frame as Figure 14 - 16. The annotations in this figure have been extracted from the YOLO predictions according to the methodology described in this section. The vertical axis corresponds to range, and the horizontal to Doppler. The further up in the image, the further away a target is. The center of the image horizontally corresponds to a radial velocity of zero, and detections to the right and left correspond to targets moving away from or towards the radar, respectively.

3.2.2 Filtering and reviewing

When all frames are annotated, an automatic filtering function is used. During the previous annotation steps, there are a couple of common pitfalls, only some of which (and how they are handled) have been discussed so far. For the rest, no simple work-around could be implemented, and these frames are marked for deletion and then removed during the filtering stage.

During the mapping between projected clusters and YOLO detections, two projected clusters can be mapped to the same bounding box. In this case, the distance between their midpoints in cartesian space is calculated and compared to a threshold of two meters. If they are further apart, it is deemed highly unlikely that both represent the same object, and it becomes impossible to automatically determine which is correctly mapped to the YOLO prediction. Put differently, two clusters are then simultaneously considered as different objects in the radar frame but are mapped to the same object in the camera frame. While potential solutions to this conflict exist, it significantly raises the risk for errors, and in such cases the whole frame gets marked for deletion.

The radar and the camera used in the recording setup have different fields of view. This has two implications for filtering data. Firstly, it is possible that the radar detects objects which are not contained in the camera image. In this case, their class cannot be extracted, and hence the frame is removed. Secondly, it is possible that projected radar clusters end up outside the field of view of the camera, even when projected to the same plane. In this case, the possible mappings where deemed to have a higher risk of error, and these frames where also omitted.

The previously two mentioned edge cases are connected to a broader one, the case where the number of projected clusters in the camera image does not match the number of objects detected by YOLO. This can however also have other causes, such as YOLO failing to detect an object that was detected and clustered by the radar, or mistaking some stationary part of the image as an object of interest.

As mentioned earlier in this section, a distance threshold in pixels was used to exclude more extreme edge cases early in the mapping stage. This means that it is possible for the radar to detect objects that can not be mapped, and therefore not classified and annotated. Effectively this means that the number of projected clusters are also compared to the number of clusters that are actually mapped. If they differ, the frame is also marked for deletion.

Finally, when all the frames are annotated and filtered, a final manual review was always performed of the recordings, to ensure that no faulty annotations not caught by the defined edge cases were undetected. During this process a small amount of frames where filtered out manually.

3.3 Data augmentation

As mentioned in Section 2.6.3, data augmentation is a regularization technique where realistic variations of the training data is generated. The following sections will introduce the augmentation methods developed for this study: Horizontal flipping and mixing. YOLOs configuration will be presented in Section 3.5, but briefly YOLOv8 has a wide selection of augmentation methods built-in. A full list of these is available in the Ultralytics documentation [40].

3.3.1 Horizontal flipping

Horizontal flipping, as the name suggests, refers to simply flipping the training data and annotation along the range axis. This implies that the objects in the frame are still present at the same range in the augmented frame, but all velocities change sign (e.g., a radial velocity of 10 m/s is transformed to -10 m/s). Horizontal flipping is used in other radar-based computer vision research, such as by [5].

3.3.2 Mixing

Mixing refers to augmenting a frame by adding in the objects of another. Again, this method was used by [5] among others. For mixing to create a realistic augmented frame, the detections of the two frames being mixed must not overlap. The method therefore compares the detection masks of two frames. If the frames are compatible, the detections from the range-Doppler image of the second frame are added into the range-Doppler image of the first frame. See Figure 18 for an example of mixing frames.

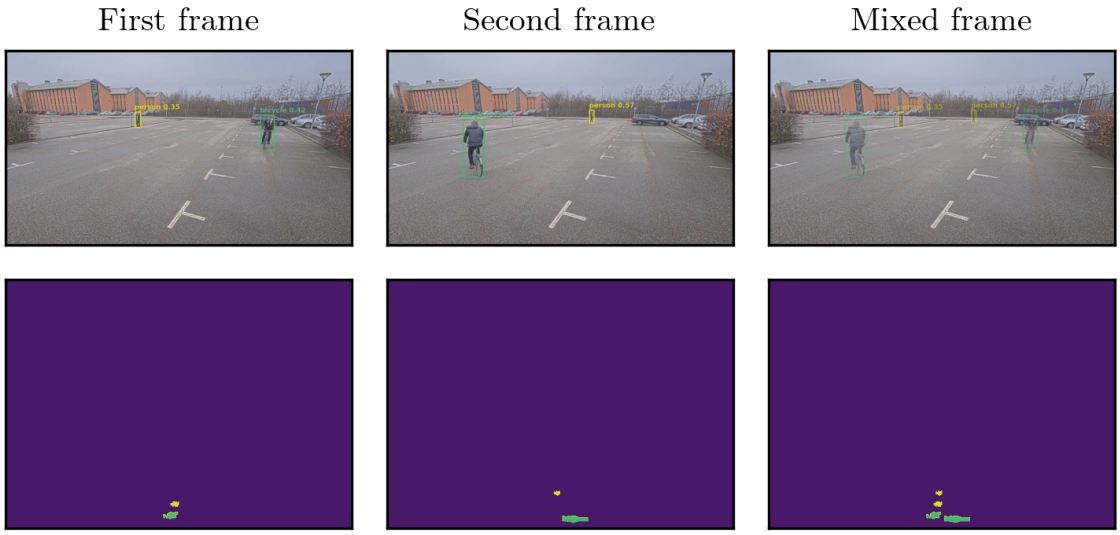

Figure 18: Example of mixing two frames for data augmentation. Note that the images are merely meant to illustrate the operation, in practice no mixing of camera images are performed.

3.4 AdaBoost (SAMME)

Due to their comparatively simple implementation, see Section 2.4, a SAMME classifier was implemented. One benefit of some ensemble learning methods (e.g., AdaBoost, gradient boosting and random forests) is that they can be used for feature selection. There are in practice many ways of performing feature selection, where the aim generally is to collect a subset of features which to a large extent explain the variance in the targets.

Before the classifier performs classification, the range-Doppler image is processed through additional steps, illustrated in Figure 19. When viewed as one cohesive process, the model used for prediction is a semantic classifier. However, rather than taking the entire image as input, the classifier takes a cluster of points and classifies it, which is then projected back onto the range-Doppler image. Thus, every pixel in the image is classified, either by the AdaBoost model, or classified as background through the threshold and clustering. Since objects tend to be more spread out in the Doppler dimension than in the range dimension, the former was divided by 5 before clustering. This value was determined through trial and error, and the clustering used was DBSCAN.

The number of estimators used were 140, which was determined through usage of grid search from `scikit-learn`. These estimators are by default decision tree classifiers. See Table 2 for the complete AdaBoost configuration.

Table 2: AdaBoost training parameters.

Parameter	**Value**
Estimator	Decision tree classifier
No. estimators	140
Learning rate	1
Boosting algorithm	SAMME

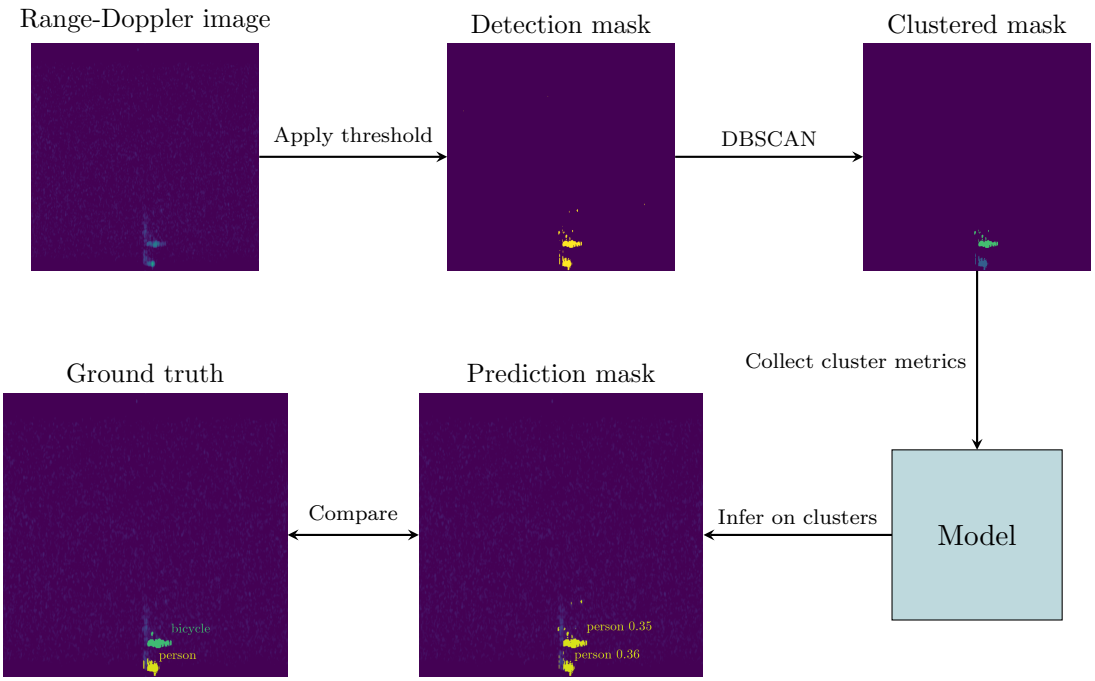

Figure 19: The steps included in the SAMME classifier. Firstly, a threshold is applied to the range-Doppler image, such that only sufficiently large values are kept and set to one. Contrary, all filtered bins are set to 0. This implies that the detection mask (top-middle picture) merely contains zeros and ones. After this step, DBSCAN performs clustering on the detection mask, based on Manhattan distance, with $minPts = 10$ and $\varepsilon = 8$. Finally, each cluster is passed through the model, which based on the values of the metrics make a prediction. In this very example, we can see that the model correctly labels the person closest to the radar, but missclassifies the bicycle which is present at a longer range.

With the implementation used, feature selection is based on something referred to as 'feature importance'. Since the base classifier used is a decision tree, the measure is computed as a mean of the provided feature importance from each individual classifier. For each classifier and input variable, this value is derived from the decrease in Gini index, when an input variable is introduced [9]. It ranges from zero to one and when the index is zero a class is entirely separable from the other classes in the training data [17].

For the purposes of this thesis, the features selected for analysis were based on the distribution of RCS, velocity and range in the different classes. For each of these, different features were calculated with respect to clusters in the data, see Table 3. More particularly, for each cluster in the range-Doppler image, the minimum, median, mean, maximum and span (i.e., difference between minimum and maximum) were taken into account.

Table 3: Feature importance for the various features. The columns show the metric being calculated for each attribute in the rows. Note that these were calculated per cluster found in the data. 'Span' here refers to the maximum minus the minimum value of the measure for a given cluster. The importance values written in bold are the most significant features for classification for each attribute.

Attribute	**Minimum**	**Mean**	**Median**	**Maximum**	**Span**
RCS	0.000	0.028	0.030	**0.213**	0.000
Range	0.000	0.023	0.000	0.128	**0.200**
Velocity	0.097	0.019	**0.210**	0.031	0.021

As can be seen from Table 3, one feature per attribute of particular importance were found. These are,

- Maximum RCS
- Range span
- Velocity median

3.5 YOLO

Since YOLOv8 was already implemented and used for the annotation, it was natural to use it again as a model for this project. As mentioned earlier in Section 2.7.1, YOLOv8 has five variations of different sizes, of which the smallest one called YOLOv8n (nano) was used. The architecture of this model is shown in Figure 6.

Apart from different sizes, YOLOv8 comes ready with several other configuration options and some relevant parameters are presented in Table 4. As mentioned in Section 3.3, YOLOv8 has a range of built-in data augmentation methods (full list available at [40]), which were left as default.

Table 4: YOLO training parameters

Parameter	**Value**
Optimizer	Adam
Learning rate	0.01
Weight decay	0.0005
Loss function(s)	CIoU, distributed focal loss, binary cross entropy
Epochs	100

3.6 U-Net

The original version of U-Net described in Section 2.7.2 was not directly implemented for this study. The model used in this study was built from the ground up, but was heavily inspired by U-Net. The general structure, with convolutions and skip connections followed by transposed convolutions is the same.

It is a smaller version, with fewer parameters and less feature depth, and has been trained directly on the study's radar data set. This study's version replaced the cropping layers with resizing layers, since they were more effective. Batch normalization layers has also been added to the model. The final convolutional layer has a softmax activation function and Glorot weight initialization, and all previous convolutional layers have ReLU activation functions and He weight initialization. Finally, this version of U-Net is a multi-class classifier, supporting a total of four classes.

Figure 20 shows the architecture of the adapted U-Net model. The full architecture is also outlined in appendix A.1.

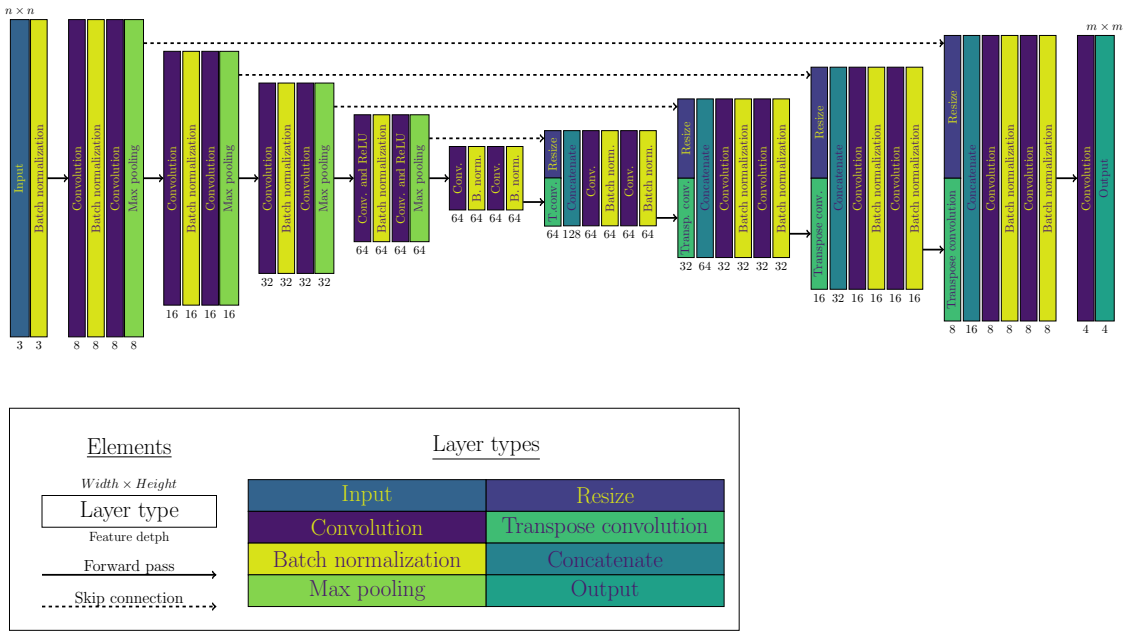

Figure 20: Illustration of the adapted U-Net Architecture. Forward passes are of course performed between all layers, but only illustrated between blocks here. Note that layer height is meant to illustrate layer dimensionality, but is approximate, and in practice changes slightly between blocks as well. Layers of the same color are of the same type, but the full names do not fit for all layers.

Table 5 below shows the training parameters used when training U-Net.

Table 5: U-Net training parameters.

Parameter	**Value**
Optimizer	Adam
Learning rate	0.001
Weight decay	0.0003
Loss function	Categorical cross entropy
Epochs	100

4 Results

4.1 AdaBoost

Below follows the performance of the AdaBoost classifier. Unless otherwise stated, the results are generated based on the testing data. Figures 21 and 22 shows AdaBoost confusion matrices on training and testing data, respectively.

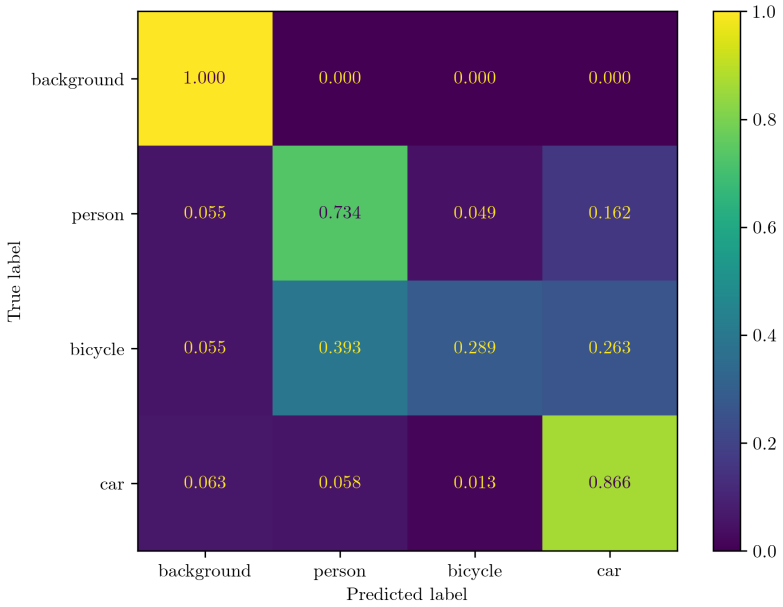

Figure 21: AdaBoost confusion matrix on training data.

AdaBoost scored a mAP of 0.708. Table 6 shows the other precision, recall and F_1 results.

Table 6: AdaBoost precision, recall and F_1 score on the test data set. Mean_{object} here refers to the mean excluding background.

Class	**Precision**	**Recall**	**F_1**
Background	1.000	1.000	1.000
Person	0.814	0.752	0.782
Bicycle	0.706	0.800	0.750
Car	0.839	0.774	0.810
Mean	0.840	0.832	0.834
Mean_{object}	0.786	0.775	0.781

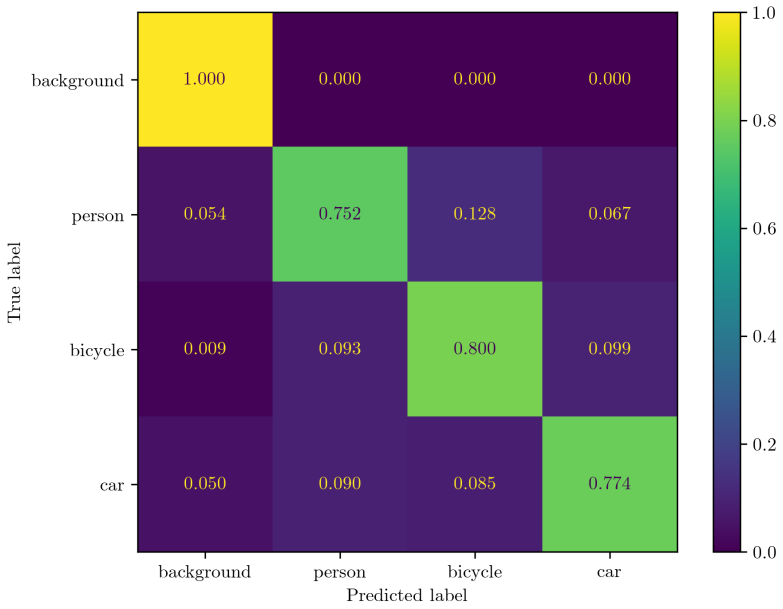

Figure 22: AdaBoost confusion matrix on testing data.

4.2 YOLO

This section will present the results training YOLOv8n on the range-Doppler data. As with the previous section, results are from test data if nothing else is indicated. Figure 23 displays YOLO's logarithmic losses during training.

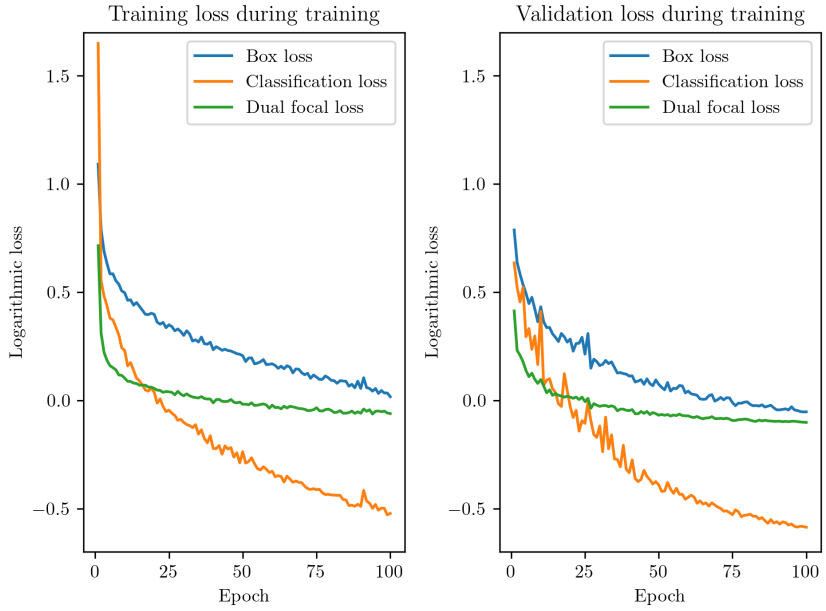

Figure 23: Logarithms of YOLO training and validation losses during training.

Figures 24 and 25 shows YOLO confusion matrices on training and testing data, respectively. Note that YOLO calculates these results based on a confidence threshold for detections of 0.25 and an IoU threshold of 0.45. The rows and columns for background should be considered with care, as YOLO does not predict background as a class. Instead, these indicate if an object was predicted were one did not exist, or if an existing object went undetected. This means that the normalization can be a bit misleading at first glance as well, especially on the row corresponding to true background.

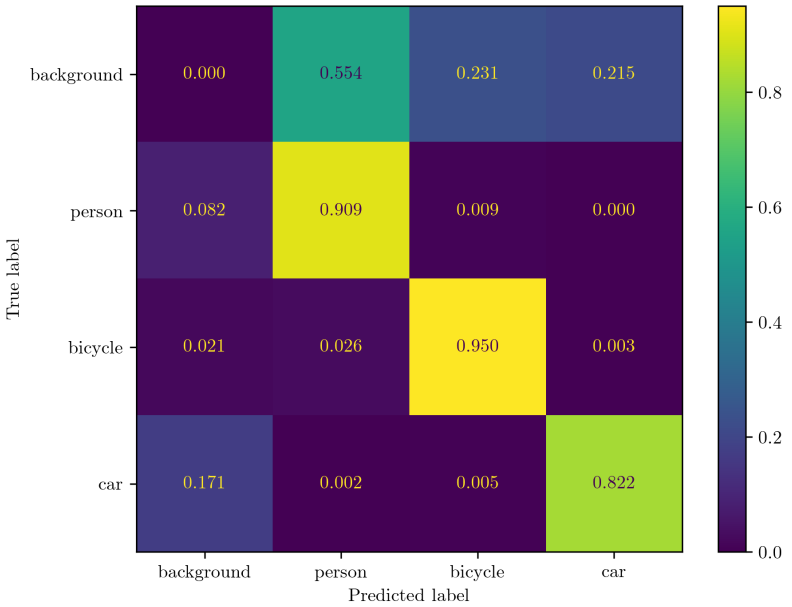

Figure 24: Normalized YOLO confusion matrix training data.

YOLO scored a mAP50 of 0.61 and a mAP50-95 of 0.271. Table 7 displays the precision, recall and F_1 results of YOLO.

Table 7: YOLO precision, recall and F_1 score.

Class	**Precision**	**Recall**	**F_1**
Person	0.596	0.553	0.574
Bicycle	0.546	0.87	0.671
Car	0.719	0.537	0.615
Mean	0.62	0.653	0.636

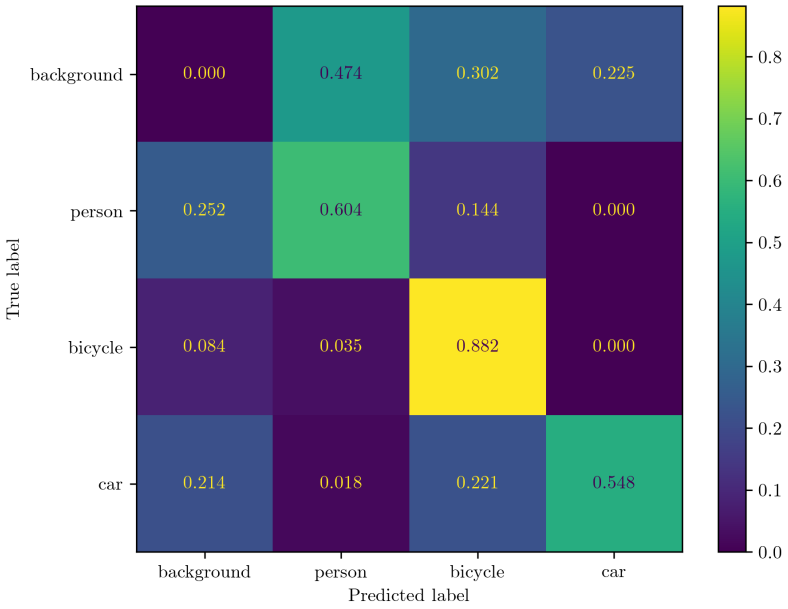

Figure 25: Normalized YOLO confusion matrix on testing data.

4.3 U-Net

Finally, this section will present the results of the adapted U-net. Again, the results are generated from test data, unless generation from training data is explicitly stated. Figure 26 below shows the logarithmic training and validation losses for U-Net during training. The final model represents the training when validation loss is at its lowest.

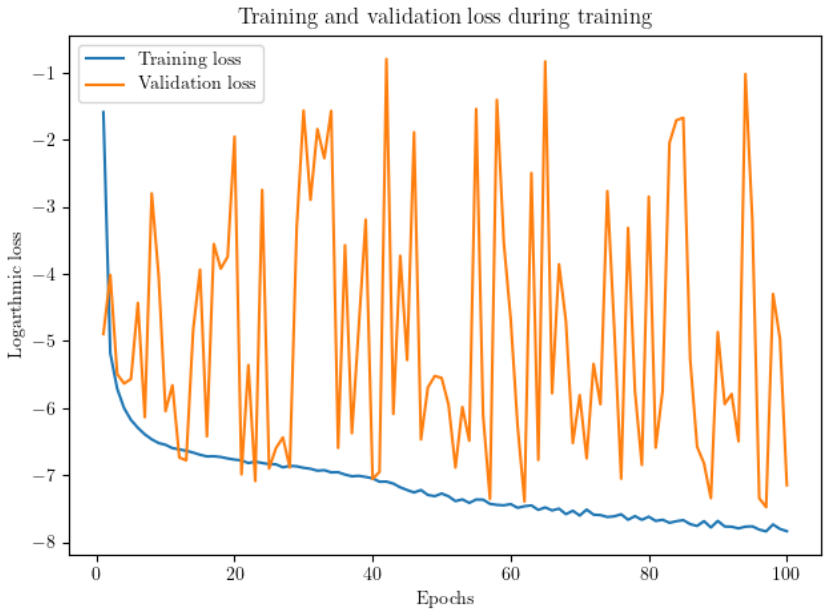

Figure 26: Logarithms of U-Net training and validation losses during training.

Figure 27 illustrates U-Net's confusion matrix from the training data, and Figure 28 shows U-Net's confusion matrix from the test data set.

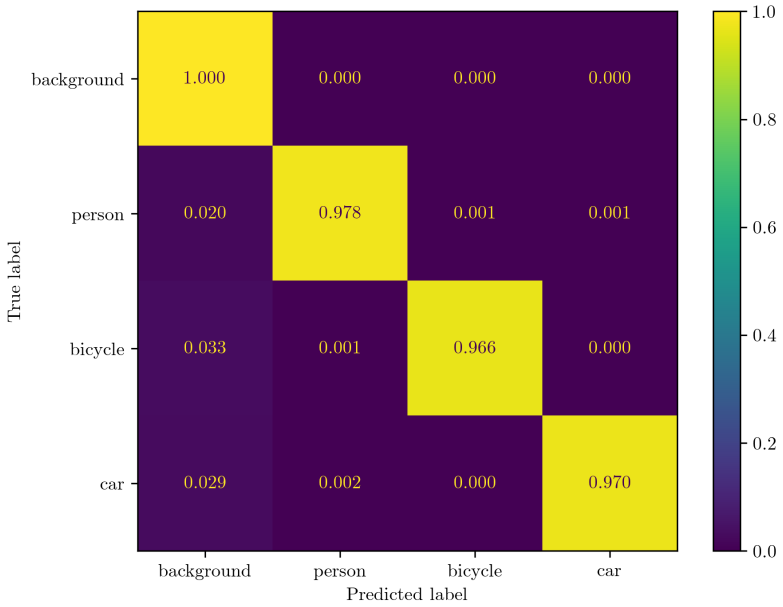

Figure 27: Normalized U-Net confusion matrix on training data.

U-Net scored a mAP of 0.860. Table 8 shows the U-Net precision, recall and F_1 results.

Table 8: U-Net precision, recall and F_1 score on the test data set. Mean_{object} here refers to the mean excluding background.

Class	**Precision**	**Recall**	$\mathbf{F_1}$
Background	1.000	1.000	1.000
Person	0.918	0.815	0.863
Bicycle	0.650	0.765	0.703
Car	0.865	0.741	0.798
Mean	0.858	0.830	0.841
$\mathbf{Mean}_{object}$	0.811	0.773	0.788

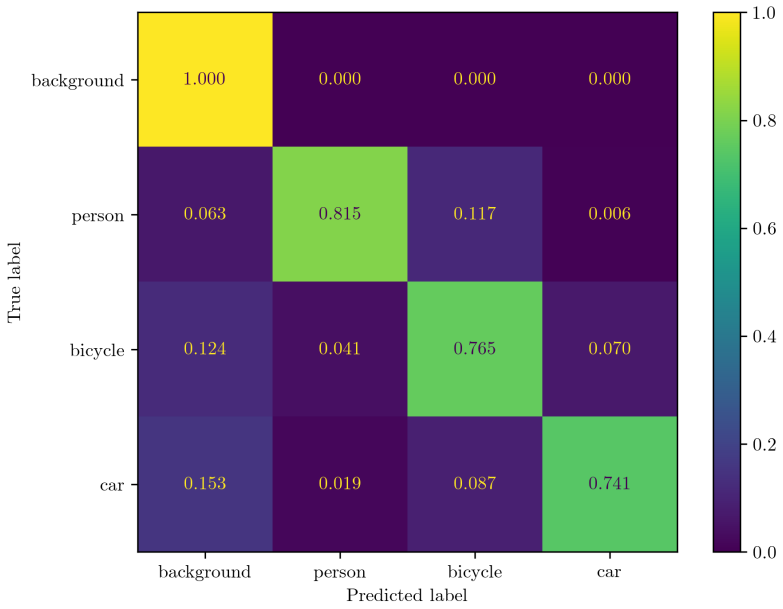

Figure 28: Normalized U-Net confusion matrix on test data.

5 Discussion

5.1 Model selection and performance

The model selection performed for this thesis was partially based on wanting to test different types of models, starting of with a simple classifier, AdaBoost. Firstly, by selecting an ensemble learning method, feature importance could be investigated early on. By analyzing these features, the process of finding appropriate input data representations for YOLOv8n and U-Net could be narrowed down. Keep in mind that this process was also inherently limited in the first place, since the purpose was to utilize preprocessed data present early in the DSP chain. Another benefit of using the AdaBoost classifier is its comparatively simple integration, and the fact that it performed well can motivate the use of relatively simple solutions to the problem. AdaBoost interestingly performed significantly better on the test data than on the training data, which is not very typical for a machine learning model. Specifically, the model mistakes bicycles for people and cars in the training data, but more successfully distinguishes the class in the test data.

It is possible that this is caused by class imbalance, since the inconsistency in performance regards primarily classifying bicycles. While it is true that bicycle is the least pixel-wise occurring class, the difference to the other classes, person in particular, does not seem significant enough to explain this result independently. Another possible explanation of this inconsistency concerns the noise. If the training data is noisier than the test data, but AdaBoost does not fit to the extra noise, it could cause this difference in performance. In that case the deficiency would be caused by the dataset, rather than by some flaw in the model.

A potential downside of using AdaBoost in a practical and embedded setting is the fact that it needs to process the data (clustering, calculating features, etc.) before inferring on it. To this end, a more unified approach was investigated.

Since the data is spatial and the tasks align well with computer vision tasks in cameras, CNNs were a natural choice. Furthermore, since YOLOv8n was already implemented as part of the annotation framework, it was an easy option to test as a next model. YOLOv8n was used out-of-the-box, not pre-trained but not modified either. This choice was made as a trade-off between YOLOv8n's performance and allowing greater focus on a third model. YOLOv8n's results were not as good as hoped and seems to indicate that the data was learned, but that it overfitted to the data set. This does not imply that models based on the YOLO architecture are unsuitable for such purposes, but rather that they would require major adaptations and changes, or a significantly larger training data set. Still, testing YOLOv8n yielded two important conclusions. The first is that a CNN-based and single-stage approach is feasible, and secondly that a smaller model is likely better.

This leads to the final model, U-Net, which performs semantic segmentation. This model was implemented from the ground up, meaning it is a lot more controllable and flexible than the previous models. It should however be stressed that this model is very much an adaptation of the original U-Net architecture from [30] to this thesis's specific goals, not a truly novel approach. U-Net was chosen as a basis because of its track record, relative simplicity and flexibility. This approach also allowed the lessons learned from AdaBoost and YOLOv8n to be fully utilized.

Even though the performance on the test data is promising, the highly variable validation loss, illustrated in Figure 26, combined with the high performance on the training data, illustrated in Figure 27, indicates that the model is overfitted. During development smaller architectures, e.g., 80,000 parameters and fewer, and stricter regularization were tested such as higher weight decay or adding dropout layers. In addition, various loss functions were tested as well. While some of these measures seemed to reduce the overfitting, they simultaneously reduced the performance of the model, and no good balance was found. Since these regularization efforts were not of primary importance to this thesis, these results were not presented, and only the best performing U-net is included in the results. While AdaBoost performs similarly or better in certain respects, U-Net is the model that scored the best in terms of mAP, and will therefore form the basis of comparison with a small selection of models from the previous research.

To anchor our results, and U-Net's mAP of 0.860 in particular, some results found in the previous research will be presented here. Note that these models were all discussed in Section 1.3.3. RAMP-CNN [5] reported an average precision of 0.792 for moving objects, and 0.812 overall. Note that this model used the full RAD cuboid and a temporal dimension, and the model was significantly larger, with over 100 million parameters. RAMP-CNN also had the same classes as this thesis, excluding background, since it is not a segmentation model. RODnet [42] which has a more similar architecture to our version of U-Net, but works on range-azimuth data, reported an average precision of 0.837. Similarly to RAMP-CNN, RODnet has the same classes as this study, excluding background since it performs object detection [42]. Finally, the SSD from [21] reported a mAP of between 0.862 and 0.88, and used either range-azimuth data or the full RAD data. The classes used are not explicitly given, but the study seems to be focused on cars, with some human detection. Without going into a very close comparison in the models, we think it is fair to conclude that our version of U-Net performs at a respectable level of precision considering the data used, the model size and the classes used.

5.2 Limitations

5.2.1 Data set

One possible limitation placed on our results, which is a common limitation within the field of deep learning, is related to the data set. There are a number of factors that influence how well a data set supports the training of a model, and hence its trained variants' generalization abilities. Firstly and perhaps most obviously is the size of the data set. As mentioned earlier in Section 3.1, the data sets in this study comprised a total of scarcely 6000 frames of data. While the amount of data was deemed sufficient to establish a proof-of-concept, it is by no means an exhaustive data set for covering the movements and ranges of people, bicycles and cars. In comparisons to much, but not all, of the related previous research, this data set is on the smaller side. CRUW, the data set created as part of the research for RODnet, contans 396,000 frames. The data set in [21] contained 106,000 of training data and 5200 frames of testing data. However, not all studies rely on such extensive data sets. The data set used to train RAMP-CNN [5] (called UWCR and in essence a predecessor to CRUW) contained two subsets of data for different scenarios consisting of 3600 and 7200 frames, respectively. To conclude, while the data used in this study seems to be sufficient to train the models to a point, the amount of

training data is definitely a possible limitation to better results and less overfitted models.

Another potential factor making the data set a limiting factor is the variation in the data. All the data was recorded with the same hardware, using only one car, two different bicycles, and the same handful of people. While separate test recordings were used, that recorded unique scenes, these were in the same general locations and recorded the same targets as the training data. This could not be easily addressed within the scope of the project, and creates some risk of too high correlation within and between the data sets. It would also have been beneficial to use completely separate recordings for the validation data.

There are aspects of the annotation framework that limits this projects ability to generate an effective data set, which by extension becomes limitations for the full project. When the data was being recorded, all classes where dedicated the same number of recordings (sometimes combined) and recordings were of similar lengths. However, as could be seen in Figure 12, some class imbalances arose. It is likely that some of this imbalance is random, but there are possible systematic factors that might have contributed to this as well. A majority of the recorded data was removed as part of the annotation process (as a rough estimate, somewhere in the neighborhood of 90%), typically because of a failure in the annotation process. It is possible that the different classes are inherently easier or more difficult for the annotation framework to handle, contributing to the observed class imbalance. Further, a potential difference in how difficult an object is to annotate could be caused by YOLO's performance with different classes, or how easy they are to detect and cluster as radar targets. As mentioned in Section 3.2, it was quite easily observed that YOLO performed worse at greater ranges, while radars generally can detect object at significantly larger ranges than this. This implies that while radar data can be captured at longer ranges, the models might perform significantly worse in these cases. The camera unit itself as well as image compression and format can also be relevant factors here. It is possible that a camera with higher resolution could have aided YOLO detections at greater ranges, and a binocular camera could have aided in depth perception, which in turn would have aided the mapping between camera and radar detections. The fact that a lot of data is lost during the annotation process is inherently limiting, since a larger data set would be beneficial. However, it is also possible that it skews the final data set, since there is a possibility that 'harder' frames are being filtered out to a greater extent. This speculation is based on the assumption that objects that are close in the range-Doppler image are harder to detect and classify for both the models and for the annotation framework. However, this is hard to substantiate without a more thorough investigation.

While the use of a flood filling algorithm presumably helped the annotation over all, it is possible that it contributes to some erroneous annotations when targets are close to each other in the range-Doppler map. However, even for manual annotators these cases can contribute with ambiguous class assignment to clusters. While many of instances of overlapping objects were likely filtered out, it is not an unrealistic phenomenon, and such cases were preserved as much as possible.

In summary, while the annotation framework seems to have produced quite good and usable data, it is not particularly efficient, and a lot of potentially viable data was deleted. An improved and tuned annotation framework would definitely have aided the results. It is worth mentioning that this was in part a deliberate choice. Better labelling was deemed

a good enough trade-off for losing a lot of data, because this could be counteracted by simply spending more time recording. However, this placed a major limitation on the amount of annotated data that could be produced within the scope of this project.

A final factor of interest to the limitations in the data sets is the augmentations. Following the discussion of class imbalance earlier, augmentation could have been used more selectively to decrease the imbalance in the data set. However, if augmentation was used in inverse proportion to a class's prevalence in the data set, less augmentation would have been performed in total. Instead, the total amount of additional frames generated was prioritized. In addition, for the two of the three models that perform semantic segmentation, which includes background, significant class imbalance was unavoidable.

Further, there exists a plethora of image augmentation techniques in the broader field of computer vision. However, many of these techniques, for example rotating an image, does not produce realistic variations when applied to the range-Doppler domain. The built-in, standard augmentation techniques used by YOLO contains such techniques, that should seemingly create range-Doppler images that are impossible to observe in reality. However, when YOLO was trained using only the augmentation developed for this study (flipping and mixing), it performed significantly worse. Note that these results were not of interest in any other respect, and were therefore not presented in Section 4. The question then arises, if YOLO performs augmentation that for many of the techniques produces variations that have no similarities with the real training data, why do they increase performance? One possible answer is that it is a large model, that inherently needs a much larger data set than provided in this study. It is also possible that such augmentations still aids in teaching the model the micro-Doppler signatures of the different classes. The shape of the cluster in the radar data remains the same, even if it is presented in an orientation that could never be observed in reality, which possibly helps the training more than it hurts it. In extension, it is possible that further augmentation, even if it is deemed to create unrealistic data, could aid in training the other models as well.

5.3 Future work

Perhaps the most obvious avenue for further research is simply increasing the scope. The selection of models or the types of models in this thesis is by no means exhaustive, and neither is the selection of classes.

A bigger data set would also be beneficial. As suggested by the comparison in Section 5.2.1, the data set developed for this project is on the smaller side. The annotation framework could also be furthered developed to be more efficient. In particular, there is a lot of potential for reducing the amount of data that is discarded in the process.

A final avenue of possible development is the radar data used and its representation. It is possible to extract more data from radar signals than used in the data sets generated for this study. The most obvious example that comes to mind is the azimuth angle of targets, which has been used for computer vision tasks in previous research (see Section 1.3.1). However, if future work wants to keep the emphasis on preprocessed data, what data is used and how much processing it requires must be considered carefully. If other data is incorporated, it is possible that other representations than range-Doppler images makes sense or perform better, which is another aspect that can be explored.

5.3.1 Possibilities for embedded use-cases

If object detection or semantic segmentation is performed to optimize radar signal processing, it has to not only work with preprocessed data, but be small and efficient enough to run on an embedded system. This discussion will be grounded in the size of models in terms of number of parameters. While this is not the only aspect of interest when considering how light-weight a model is or can be, it serves as a good starting point for future research and discussion.

The first point of interest is that a massive model is likely not needed to get adequate performance. The version of YOLO used has about three million parameters, and performed worse than the adapted U-Net model, which has about 300,000 parameters. Keep in mind that the fact that YOLO seems to overfit more is likely aided by the fact that the data set is quite small. Still, this is an indication that a smaller model is not only feasible, but preferable, which is favorable for possible embedded implementations.

To give some perspective on these sizes, some quick examples of models intended for embedded use will be presented here. An adapted YOLO model for detecting trees was published in 2022 which contains 1.78 million parameters [19]. While this is a significant improvement to official YOLO models in terms of size, it is still a large model in comparison to the U-Net model suggested in this thesis. Tiny SSD, a SSD (single shot detector) performing object detection through the placing of bounding boxes, intended for embedded systems, contained 1.13 million parameters [43]. Finally, as an example for embedded semantic segmentation, another SSD called FASSD-Net contains (in its smallest version) 1.9 million parameters [31].

In terms of number of parameters, the adapted U-Net model proposed in this thesis seems adequately small to run on embedded systems, and further testing and research into these possibilities looks promising.

6 Conclusion

As stated in Section 1.2, the purpose of this thesis is to investigate the potential of instantaneous object detection and semantic segmentation on preprocessed FMCW radar data. To achieve this, two secondary goals arose: Creating a custom annotation framework and data sets, as well as selecting and testing models using these data sets.

A custom annotation framework was developed, based on the projection of clustered radar detections onto camera images annotated by YOLOv8n, where they can be mapped to detected objects. The annotation framework generates high quality labels, and requires only minor manual filtering. However, a lot of recorded data is lost in the process.

Three models were selected and tested: AdaBoost, YOLOv8n, and an adapted U-Net. AdaBoost, being a quite simple classifier, required the development of an extended pipeline to be used in the same manner as the other models. The result was a semantic classifier, combining AdaBoost's cluster-wise classification with DBSCAN clustering. The development of AdaBoost also allowed for feature extraction, and the conclusion that the data used is in fact pertinent to the classification of FMCW data. The expanded AdaBoost model performs well, with a mean F_1 score of 0.834 and a mAP of 0.708.

Since YOLOv8n was implemented as part of the annotation framework, it was natural to retrain and test it on FMCW data. While YOLOv8n did work in this application, it is not optimal for this particular data set. Even though the smallest variant of YOLOv8 was used, it is a comparatively large model, and seems to have overfitted to the data set. It is likely that the general YOLO architecture could be much better suited to the task, if it was further scaled down and/or was trained using a much larger data set. YOLOv8n scored a mean F_1 score of 0.636, a mAP50 of 0.61, and a mAP50-95 of 0.271.

Finally, a model heavily inspired by U-Net was developed. It was also implemented from the ground up, giving us much more control and flexibility. For one, this allowed for broader experimentation with different parameters such as loss functions and optimizers. Perhaps more importantly, it gives direct control over the size of the model, which helped reduce the risk of overfitting. U-Net scored a mean F_1 score of 0.841 and a mAP of 0.860. While both CNNs seems to have overfitted, the results quite clearly indicate that U-Net is the better performing CNN in this thesis. U-Net has a slightly better precision in general than AdaBoost, but performs similarly over all metrics. The higher precision and comparable recall result in a slightly higher F_1 score than AdaBoost, and U-Net is the best performing model in terms of mAP.

To conclude on the primary purpose of this thesis, instantaneous object detection and semantic segmentation on preprocessed FMCW data is not only feasible, but quite promising. We have demonstrated that there are sufficiently accurate models, that can certainly be improved further by for example reducing overfitting. We have also demonstrated some possibilities of automatic data annotation, which can be used to generate larger data sets. However, the ultimate motivation of this thesis was the possibility of using models such as these to optimize the DSP of the radar. If the models used are sufficiently efficient and light-weight to run embedded in a radar unit and directly influence the processing is still uncertain.

A Appendix

A.1 Full adapted U-Net architecture

Layer name	Feature depth	No. parameters	Input(s)
input_2	3	0	
conv2d	8	224	input_2
activation	8	0	conv2d
batch_normalization_2	8	32	activation
conv2d_1	8	584	batch_normalization_2
activation_1	8	0	conv2d_1
max_pooling2d	8	0	activation_1
conv2d_2	16	1168	max_pooling2d
activation_2	16	0	conv2d_2
batch_normalization_3	16	64	activation_2
conv2d_3	16	2320	batch_normalization_3
activation_3	16	0	conv2d_3
max_pooling2d_1	16	0	activation_3
conv2d_4	32	4640	max_pooling2d_1
activation_4	32	0	conv2d_4
batch_normalization_4	32	128	activation_4
conv2d_5	32	9248	batch_normalization_4
activation_5	32	0	conv2d_5
max_pooling2d_2	32	0	activation_5
conv2d_6	64	18496	max_pooling2d_2
activation_6	64	0	conv2d_6
batch_normalization_5	64	256	activation_6
conv2d_7	64	36928	batch_normalization_5
activation_7	64	0	conv2d_7
max_pooling2d_3	64	0	activation_7
conv2d_8	64	36928	max_pooling2d_3
activation_8	64	0	conv2d_8
batch_normalization_6	64	256	activation_8
conv2d_9	64	36928	batch_normalization_6
activation_9	64	0	conv2d_9
batch_normalization_7	64	256	activation_9
conv2d_transpose	64	16448	batch_normalization_7
tf.image.resize	64	0	activation_7
concatenate	128	0	conv2d_transpose, tf.image.resize
conv2d_10	64	73792	concatenate
activation_10	64	0	conv2d_10
batch_normalization_8	64	256	activation_10
conv2d_11	64	36928	batch_normalization_8
activation_11	64	0	conv2d_11
batch_normalization_9	64	256	activation_11
conv2d_transpose_1	32	8224	batch_normalization_9
tf.image.resize_1	32	0	activation_5

concatenate_1	64	0	conv2d_transpose_1, tf.image.resize_1
conv2d_12	32	18464	concatenate_1
activation_12	32	0	conv2d_12
batch_normalization_10	32	128	activation_12
conv2d_13	32	9248	batch_normalization_10
activation_13	32	0	conv2d_13
batch_normalization_11	32	128	activation_13
conv2d_transpose_2	16	2064	batch_normalization_11
tf.image.resize_2	16	0	activation_3
concatenate_2	32	0	conv2d_transpose_2, tf.image.resize_2
conv2d_14	16	4624	concatenate_2
activation_14	16	0	conv2d_14
batch_normalization_12	16	64	activation_14
conv2d_15	16	2320	batch_normalization_12
activation_15	16	0	conv2d_15
batch_normalization_13	16	64	activation_15
conv2d_transpose_3	8	520	batch_normalization_13
tf.image.resize_3	8	0	activation_1
concatenate_3	16	0	conv2d_transpose_3, tf.image.resize_3
conv2d_16	8	1160	concatenate_3
activation_16	8	0	conv2d_16
batch_normalization_14	8	32	activation_16
conv2d_17	8	584	batch_normalization_14
activation_17	8	0	conv2d_17
batch_normalization_15	8	32	activation_17
conv2d_18	4	36	batch_normalization_15
Total params:	323828 (1.24 MB)		
Trainable params:	322852 (1.23 MB)		
Non-trainable params:	976 (3.81 KB)		

References

[1] Fahad Jibrin Abdu et al. "Application of deep learning on millimeter-wave radar signals: A review". In: *Sensors* 21.6 (2021), p. 1951.

[2] Tingting Chen and Qingzhu Zeng. "Research on Bubble Detection Based on Improved YOLOv8n". In: *IEEE Access* (2024).

[3] E Roy Davies. *Computer vision: principles, algorithms, applications, learning*. Academic Press, 2017.

[4] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.

[5] Xiangyu Gao et al. "RAMP-CNN: A Novel Neural Network for Enhanced Automotive Radar Object Recognition". In: *IEEE Sensors Journal* 21.4 (2021), pp. 5119–5132. DOI: 10.1109/JSEN.2020.3036047.

[6] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow.* " O'Reilly Media, Inc.", 2022.

[7] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[9] Hong Han, Xiaoling Guo, and Hua Yu. "Variable selection using mean decrease accuracy and mean decrease gini based on random forest". In: *2016 7th ieee international conference on software engineering and service science (icsess)*. IEEE. 2016, pp. 219–224.

[10] Trevor Hastie et al. "Multi-class adaboost". In: *Statistics and its Interface* 2.3 (2009), pp. 349–360.

[11] Kaiming He et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[12] Cesar Iovescu and Sandeep Rao. "The fundamentals of millimeter wave sensors". In: *Texas Instruments* (2017), pp. 1–8.

[13] Sung-wook Kang, Min-ho Jang, and Seongwook Lee. "Autoencoder-Based Target Detection in Automotive MIMO FMCW Radar System". In: *Sensors* 22 (July 2022), p. 5552. DOI: 10.3390/s22155552.

[14] Nitish Shirish Keskar and Richard Socher. "Improving generalization performance by switching from adam to sgd". In: *arXiv preprint arXiv:1712.07628* (2017).

[15] Kamran Khan et al. "DBSCAN: Past, present and future". In: *The fifth international conference on the applications of digital information and web technologies (ICADIWT 2014)*. IEEE. 2014, pp. 232–238.

[16] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[17] Carl Kingsford and Steven L Salzberg. "What are decision trees?" In: *Nature biotechnology* 26.9 (2008), pp. 1011–1013.

[18] Tsung-Yi Lin et al. "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.

[19] Feng Lü et al. "Tree Detection Algorithm Based on Embedded YOLO Lightweight Network". In: *Journal of Shanghai Jiaotong University (Science)* (2022), pp. 1–10.

[20] T. Soni Madhulatha. *An Overview on Clustering Methods*. 2012. arXiv: `1205.1117 [cs.DS]`.

[21] Bence Major et al. "Vehicle detection with automotive radar using deep learning on range-azimuth-doppler tensors". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*. 2019, pp. 0–0.

[22] Michael Meyer and Georg Kuschk. "Automotive radar dataset for deep learning based 3d object detection". In: *2019 16th european radar conference (EuRAD)*. IEEE. 2019, pp. 129–132.

[23] Vladimir Milovanović. "On fundamental operating principles and range-doppler estimation in monolithic frequency-modulated continuous-wave radar sensors". In: *Facta Universitatis, Series: Electronics and Energetics* 31.4 (2018), pp. 547–570.

[24] Arthur Ouaknine et al. "Carrada dataset: Camera and automotive radar with range-angle-doppler annotations". In: *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE. 2021, pp. 5068–5075.

[25] Kanil Patel et al. "Deep learning-based object classification on automotive radar spectra". In: *2019 IEEE Radar Conference (RadarConf)*. IEEE. 2019, pp. 1–6.

[26] Vincenzo Pesce, Andrea Colagrossi, and Stefano Silvestrini. *Modern Spacecraft Guidance, Navigation, and Control: From System Modeling to AI and Innovative Applications*. Elsevier, 2022.

[27] Sandeep Rao. *White paper: MIMO radar*. 2017. URL: `https://www.ti.com/lit/an/swra554a/swra554a.pdf`.

[28] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: `1506.02640 [cs.CV]`.

[29] Mark A Richards et al. *Principles of modern radar*. Citeseer, 2010.

[30] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer. 2015, pp. 234–241.

[31] Leonel Rosas-Arias et al. "FASSD-Net: Fast and accurate real-time semantic segmentation for embedded systems". In: *IEEE Transactions on Intelligent Transportation Systems* 23.9 (2021), pp. 14349–14360.

[32] Erich Schubert et al. "DBSCAN revisited, revisited: why and how you should (still) use DBSCAN". In: *ACM Transactions on Database Systems (TODS)* 42.3 (2017), pp. 1–21.

[33] Marcel Sheeny, Andrew Wallace, and Sen Wang. "300 GHz radar object recognition based on deep neural networks and transfer learning". In: *IET Radar, Sonar & Navigation* 14.10 (2020), pp. 1483–1493.

[34] Chen Sun et al. "Revisiting unreasonable effectiveness of data in deep learning era". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 843–852.

[35] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[36] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[37] Juan Terven, Diana-Margarita Córdova-Esparza, and Julio-Alejandro Romero-González. “A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas”. In: *Machine Learning and Knowledge Extraction* 5.4 (2023), pp. 1680–1716.

[38] Ultralytics. *Detect - Ultralytics YOLOv8 Docs*. URL: `https://docs.ultralytics.com/tasks/detect/`.

[39] Ultralytics. *Home - Ultralytics YOLOv8 Docs*. URL: `https://docs.ultralytics.com/`.

[40] Ultralytics. *Model Training with Ultralytics YOLO- Augmentation Settings and Hyperparameters*. URL: `https://docs.ultralytics.com/modes/train/#augmentation-settings-and-hyperparameters`.

[41] Ultralytics. *Object Detection Datasets Overview*. URL: `https://docs.ultralytics.com/datasets/detect/`.

[42] Yizhou Wang et al. “Rodnet: Radar object detection using cross-modal supervision”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2021, pp. 504–513.

[43] Alexander Womg et al. “Tiny SSD: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection”. In: *2018 15th Conference on computer and robot vision (CRV)*. IEEE. 2018, pp. 95–101.

[44] Ao Zhang, Farzan Erlik Nowruzi, and Robert Laganiere. “Raddet: Range-azimuth-doppler based radar object detection for dynamic road users”. In: *2021 18th Conference on Robots and Vision (CRV)*. IEEE. 2021, pp. 95–102.

[45] Zhengxia Zou et al. “Object detection in 20 years: A survey”. In: *Proceedings of the IEEE* (2023).

Master's Theses in Mathematical Sciences 2024:E30
ISSN 1404-6342

LUTFMA-3538-2024

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/