Identifying Piggybacking with Radar and Neural Networks

Hannes Olsson, Joel Sigurdsson

Master's thesis
2022:E67

Lund Institute of Technology
Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Centrum Scientiarum Mathematicarum

Abstract

A common problem in access control is piggybacking. This is when a person without authorized access sneaks closely behind another with access through a door. This thesis seeks to answer whether using radar is a viable solution when attempting to detect piggybacking. Detection will be made by classifying sequences of point clouds generated by the radar, using neural networks. The thesis compares two different placements of the radar, at the side of- and above a door, with an existing camera based piggybacking detection solution.

In addition to comparing the results, the development of the model will be described in detail. This includes exploring different architectures for the neural network(s). Moreover, strengths and weaknesses of radar technology, compared to camera technology will be discussed.

The results show that all three solutions perform well, with accuracy above 99% when one or two people are walking normally in frame. When comparing the solutions on more challenging scenarios such as one person carrying a big box or two people hugging while walking, both radar based solutions outperform the camera based solution.

In general, slightly better separation between people can be seen in the point clouds generated by the radar placed above the door. This resulted in slightly better performance compared to the placement at the side of the door in certain scenarios.

In a world where privacy and integrity is more valued than ever, radar has a big role to play in modern access control solutions. The results from this thesis show that a radar can perform at the same level, and sometimes better than a camera for detecting piggybacking.

Contents

1 Introduction

1.1 Background

This is a project idea that has been developed by the New Business Access Control division at Axis Communications, together with the authors. One major security threat in access control is piggybacking. Piggybacking is when a person with authorized access unlocks a security door, and someone without access walks in closely behind them. There are existing solutions for identifying this scenario using camera technology, but Axis would like research regarding the viability and effectiveness of using radar technology. Recording an area with cameras is sometimes restricted by law, while doing so with radar might not be. For this reason, usage of radar can be of interest even if it does not outperform the camera solution.

1.1.1 Piggybacking

In certain access control situations it can be of interest to monitor whether multiple people are attempting to pass through a security door in tandem. By walking very close to each other, a malicious employee can make it difficult for surveillance systems to identify the unauthorized second person. Axis' existing surveillance systems for this type of detection is based on camera technology. It has been determined to be of interest to explore whether a high-resolution FMCW radar can be used for the same purposes.

1.1.2 Radar versus Camera

For the purpose of detecting piggybacking, the point clouds from a radar have a few major advantages over images from a camera that are purely technical. They contain data in three dimensions and they contain velocity data. The trade off for this is a loss of detail. See Figure 1 for an example of how the same scene looks when represented in both camera image and radar point cloud. All that can be seen is the shape of the objects in the radar's field of view. The radar is also not affected by lighting conditions and changes in the environment, such as whether it is raining or not in a scene. When considering non-technical aspects, radar has advantages in aspects related to economy and integrity. From an economical point of view, a radar is generally cheaper to produce than a video camera. From an integrity point of view, a radar is a lot less intrusive since it only sees shapes, and can not be used to identify a person. In some places, camera surveillance is restricted by law, while surveillance using radar might not be.

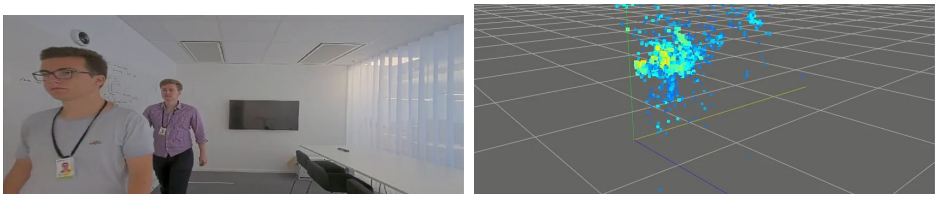

Figure 1: View from the camera and the radar at the reader position

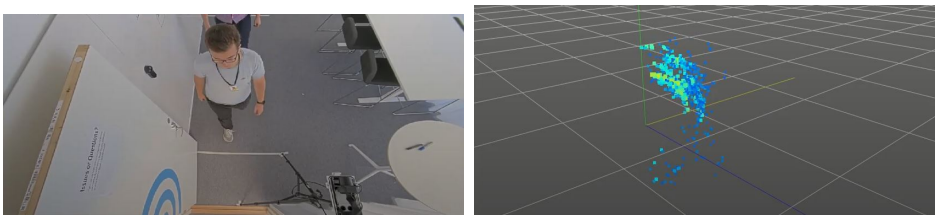

Figure 2: View from the camera and the radar at the roof position

1.2 Purpose

The purpose of this thesis is to design a viable model for pre-processing raw radar-data as well as classifying a set of scenarios related to piggybacking. There is no specified number for what level of accuracy would considered 'viable'. Two important thresholds would be if it performs better than a guess, and if it performs as well as the camera solution. Furthermore, two radar units will be used. One will be located above the door, recording at a suitable downwards angle. The other will be placed at the side of the door, roughly at chest height, angled horizontally. The views from both radars can be seen in Figure 1 and Figure 2. A sketch showing the two different placements of the radars can be seen in Figure 3. Data will be recorded from both points of view simultaneously, with the intent to compare the two positions in terms of accuracy.

The research questions that this thesis seeks to answer is:

- (RQ1): Is radar at all a viable solution for this type of classification problem?
- (RQ2): How does the developed radar solution perform in comparison to an existing camera solution on the same test data?
- (RQ3): What is the preferred placement of the radar: Above the door, or at the side of the door?

On top of these research questions, two additional purposes are formulated as:

- (AP1): Develop a model with the best possible accuracy given the project's time constraints.
- (AP2): Provide insights that can aid development of similar projects.

Figure 3: Drawing showing the two positions of the radar

1.3 Prior Research

1.3.1 Classification on high resolution 3D models

A resource frequently used (e.g. [14], [17], [18], [22]) in research regarding classification on high resolution 3D models is ShapeNet [3]. With high resolution means that the shapes are detailed enough that a human can easily tell what it is. ShapeNet is a database that contains two different datasets that are used to evaluate models on classification of 3D models. These datasets are called ShapeNet10 and ShapeNet40

and include 10 and 40 classes respectively. Some examples of classes are airplanes, cars and cups. As of September 2022, the state of the art model on ShapeNet is PIG-Net [18]. Another good performer is PointNet [22], which will be central within this thesis.

1.3.2 Pose classification of people

In a previous thesis at Axis, the same radar hardware was used to classify humans as either standing, sitting or lying down. Important take-aways from this thesis is how proficient PointNet is at classifying peoples poses [1]. Given this proficiency at a fairly similar problem, with the same radar, PointNet would be expected to perform well on the piggybacking problem.

1.3.3 Classification on point clouds

Point clouds are challenging to classify. One reason for this is that they are unordered. Therefore, for a model to be successful at classifying point clouds, it needs to be invariant to input permutation [22]. According to R.Qi et al. there are three ways of handling this. The first solution is to order the input canonically. The second would be to treat the input as a sequence and augment the data by all possible permutations, and train a RNN on the permutations. Thirdly, one could use a symmetric function to aggregate the information from each point [22]. The creators of PointNet argue that using a symmetric function to aggregate the information is the most effective way.

1.4 Statement of contribution

Throughout the thesis work, Hannes Olsson and Joel Sigurdsson have been working closely together with most aspects of the project. Some division of labor during the development phase was done, where Joel had more responsibilities regarding the implementation of software suited to his relative expertise in computer science. Hannes focused more on experimenting with the networks, making changes to architecture and hyperparameters to find improvements in performance. This was suited to his relative expertise in mathematics. Every chapter of the thesis has been worked on together, although some division of labor for subsections has been done. The contributions to both the development phase and the writing phase have been equal.

2 Theory

In this chapter, the knowledge applied within the thesis will be explained. The chapter will briefly cover FMCW radar and the camera solution. It will also provide an overview on neural networks, as well as explanation of techniques that are implemented during the project.

2.1 Radar

Radar is a technology that can measure distance to an object using radio-waves. In addition to measuring the distance to an object, it can also be used to measure the angle to the object, the object's velocity and direction of the velocity. A radio wave is emitted in pulses or continuously by the radar, the wave is then reflected by objects in its field of view back to the radar. From now on, a radar that emits its signal in pulses will be referred to as a pulsed radar, and a radar that emits its signal continuously will be referred to as a continuous radar.

2.1.1 FMCW Radar

The radar modules used in this project are Frequency-Modulated Continuous Wave radars or FMCW for short. The FMCW radar is a variation of the Continous Wave radar, CW for short. One major difference between pulsed radar and Continuous Wave radar is that instead of sending out pulses, the CW radar radiates a continuous signal. This gives it different properties compared to pulsed radar. The CW radar is generally simpler and uses less power, it can detect objects that are very close to the radar and it has a higher resolution than the pulsed radar. It also has a lower maximum range than a pulsed radar. A pulsed radar can operate on ranges in the kilometers, while a CW radar is effective up to about 100 meters. This range varies depending on the radar in question. Being able to detect objects very close to the radar makes it ideal for the use case within this thesis. One disadvantage of the CW radar is that it cannot tell the distance to an object. This is why an FMCW radar is used instead of a regular CW radar. The FMCW radar maintains positives of the CW radar, such as having high resolution and being able to detect objects very close to the radar, with the added property that it can measure distance to an object. It does this by modulating the frequency of the transmitted signal. Modulation of the frequency during transmission is also known as transmitting in chirps. The distance to the object is then calculated by considering both the difference in phase and frequency, between the transmitted signal and the received signal. This is referred to as the Doppler shift [2]. A complete FMCW radar system consists of atleast two antennas, one or more TX antennas for transmitting and one or more RX antennas for receiving. It also includes a synthesizer that generates the chirps and a mixer combining the transmitted and received signals. The mixer combines the signal generated by the synthesizer that is currently being transmitted, and the signal received by the RX

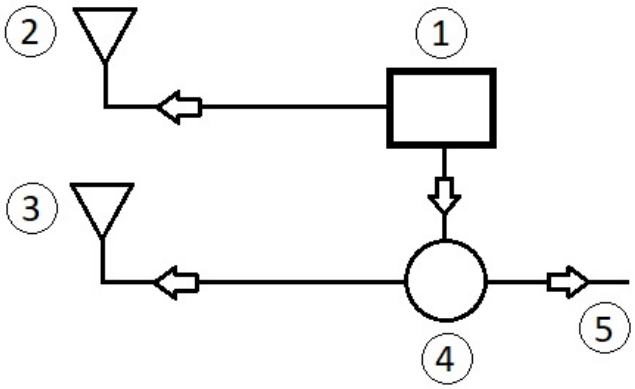

Figure 4: (1) the synthesizer, (2) one or many TX antennas, (3) one or many RX antennas, (4) the mixer, (5) the IF signal. The arrows show where the radio signal is sent.

antenna. This combined signal is called the IF signal and it contains important properties that will be used to calculate distance, velocity and angle to an object. Figure 4 shows a complete FMCW radar system, in a block diagram. The FMCW radar module used in this thesis has a high resolution. For two points to be seen as two separate points, they need to be at least 5.8 cm apart.

Range measurement As previously mentioned, FMCW radar uses frequency modulation of the transmitted signal to determine distance to a target. The transmission is done in chirps that have a start frequency f_c, a bandwidth B, a duration T_c and a frequency shift S [12]. Figure 5 shows how the chirps look with regard to amplitude over time and frequency over time.

Once a reflected signal is received, the mixer combines the two sinusoidal waves into a single sinusoid. The frequency of the IF signal (the combined sinusoid) is equal to the difference between the frequencies of the transmitted signal and the received signal and the phase of the IF signal is equal to the difference in phase between the transmitted signal and the received signal. The distance d to an object is calculated as

$$d = \frac{c|\Delta t|}{2} = \frac{c|\Delta f_r|}{2S}, \tag{1}$$

where c is the speed of light, Δt is the time delay, Δf_r is the frequency difference and S is the frequency shift per time unit [12].

Velocity measurement Using the phase difference from the sinusoid produced by the mixer, the velocity v of an object can be calculated as

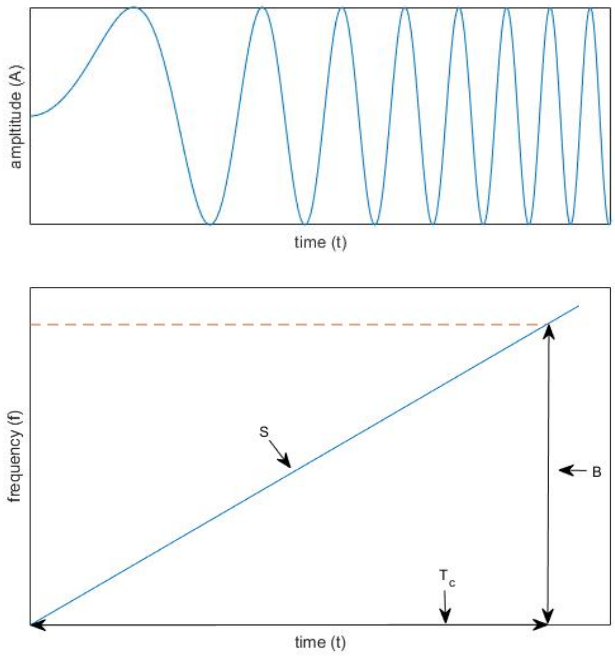

Figure 5: Plots showing the chirps with amplitude and frequency over time.

$$v = \frac{\lambda \Delta\omega}{4\pi T_c}, \tag{2}$$

where $\Delta\omega$ is the phase difference between the transmitted signal and the received signal, λ is the wavelength and T_c is the time between chirps [12].

Angle measurement If more than one RX antenna is used, the angle to an object can be measured. This is the so-called angle of arrival (AoA). With two RX antennas at a distance Δd from each other, the angle of a reflected signal can be calculated by analyzing the phase difference at the two antennas.

The phase difference $\Delta\phi$ between the RX antennas is calculated as

$$\Delta\phi = \frac{2\pi\Delta d}{\lambda}. \tag{3}$$

Using basic geometry one can show that $\Delta d = l\sin(\theta)$, where l is the distance between the two RX antennas and θ is the angle of arrival. For the example setup in Figure 6, θ can be calculated as

$$\theta = \sin^{-1}(\frac{\lambda\Delta\phi}{2\pi l}). \tag{4}$$

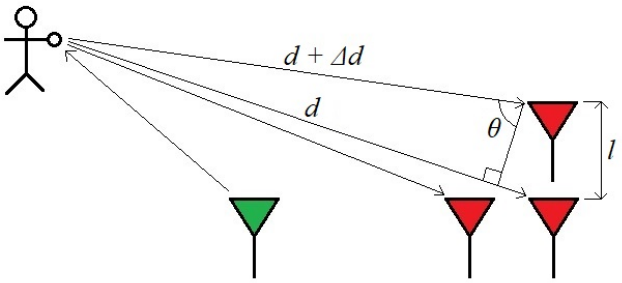

Figure 6: Example setup of TX antenna (green) and RX antennas (red) for AoA calculation.

Determining (X, Y, Z) coordinates To be able to determine the position of a point in the room, at least three RX antennas positioned in different planes are needed. With more than three antenna, both the vertical angle θ and the azimuth angle σ can be calculated. The azimuth angle is an angle in a spherical coordinate system, more precisely it is the horizontal angle from a cardinal direction. Once you have the distance d, the vertical angle θ and the azimuth angle σ, it is fairly straightforward to determine the (X, Y, Z)-coordinates of the point, which are calculated as

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = d \begin{pmatrix} \cos(\theta) * \sin(\sigma) \\ \cos(\theta)) \\ \cos(\theta) * \cos(\sigma) \end{pmatrix}, \tag{5}$$

where θ is the vertical angle, σ is the azimuth angle and d is the distance to the radar.

2.2 Artificial Neural Networks

Artificial neural networks (ANNs), also referred to simply as neural networks, are an attempt at creating a mathematical reproduction of the structures found in the human brain [9]. This is achieved by creating an assembly of simple processing elements, with a functionality loosely based on the neurons found in humans and other animals. Neurons are nerve cells that communicate via impulses of voltage in the cell wall. A neuron typically receives input from many thousands of connections, which are, in simple terms, summed together in the cell body to result in an output resulting in a voltage impulse. Synapses mediate the connections by adjusting the strength of the signal, and are modeled as weights in the artificial neuron, which similarly considers a sum of all the weighted inputs before calculating an output.

Artificial neural networks can be used for a variety of purposes. In this thesis the application will be a *classification problem*. This is essentially the task of guessing the

correct *label* for an image, based on a predetermined set of *classes*. A label is a pointer that contains information regarding which class a piece of data belongs to, but the terms are practically interchangeable. An example of this task is classification of handwritten digits. A resource that can be used for such research is the MNIST dataset, which contains 70,000 images [15]. Classification of MNIST involves assigning one of ten labels corresponding to the digits 0-9 to a large set of images with a resolution of 28x28. See Figure 7 for an example of labeling. While typical 2D images is a more common subject for image recognition, in this thesis the images will be in the form of 3D point clouds, and the classes used will be either 'Piggybacking' or 'Not Piggybacking'.

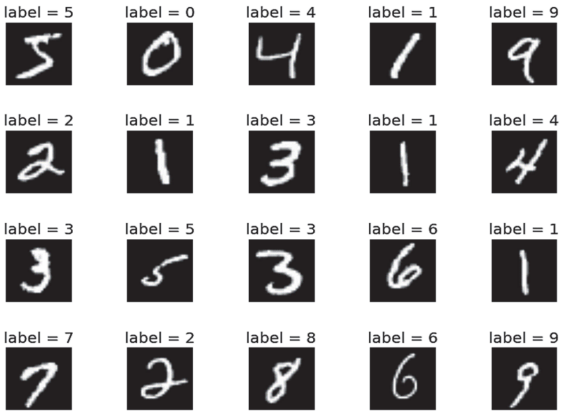

Figure 7: Samples from the MNIST dataset and their respective labels [24, p.3].

Artificial neural networks (ANN) consist of five main components: *artificial neurons*, *connections*, *activation functions* and *biases* living on the artificial neurons, as well as *weights* living on the connections. A column of artificial neurons is referred to as a *layer*. Artificial neurons are hereafter referred to simply as *neurons* or *nodes*. These terms will be used interchangeably. ANNs are created by connecting layers of artificial neurons. See Figure 8 below for a simple example network. The size of a layer refers to how many neurons it contains. Each neuron in the network has a single output value, also referred to as an activation. This output depends on the connections from the previous layer, their related weights, the neuron's bias, and finally an activation function. How these activations are calculated will be further explained in Section 2.2.1.

There are multiple types of neural networks. Two examples which will be used in this thesis are *feedforward networks* and *recurrent networks*. The network shown in Figure 8 is a fully connected feedforward network. In a fully connected feedforward network, each neuron's output value is passed on to every neuron in the next layer, until the output is reached. The two *hidden layers* in Figure 8 are *dense* layers, meaning that they are fully connected to the previous layer. Hidden layers simply

refers to any layers between the input- and output layers. The output layer is the final layer in the network, in this case a dense layer of size 1. Arrows between neurons represent the connections, each of which is associated with a unique weight. Recurrent neural networks will be covered in Section 2.2.5.

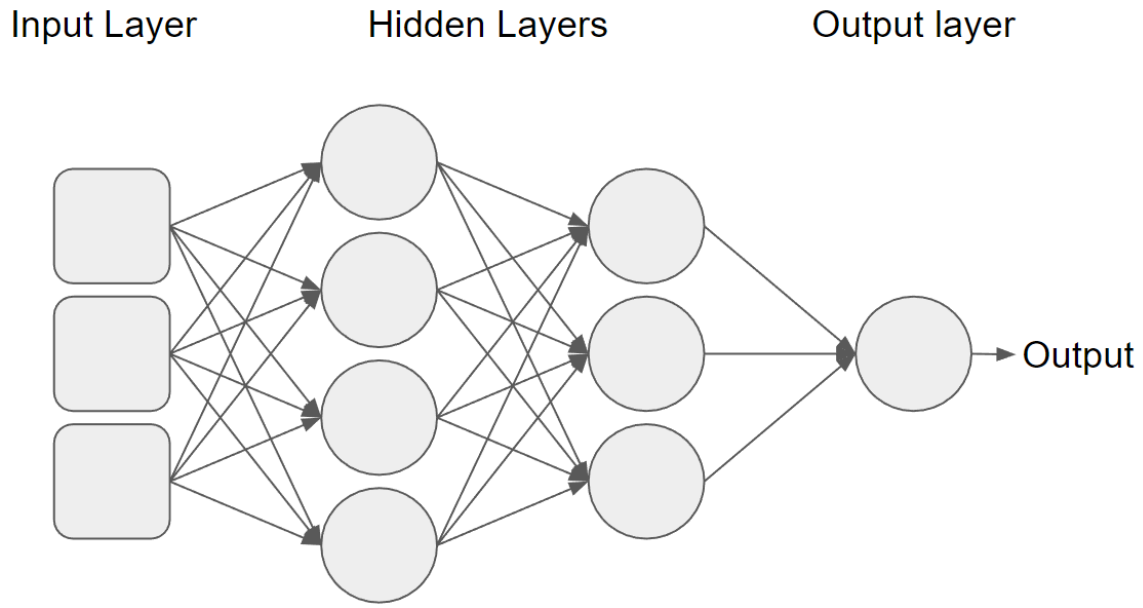

Figure 8: Example ANN with three inputs (squares), two hidden layers of neurons (circles), and a single output neuron (circle).

2.2.1 Activation of Neurons

To clarify how information travels through the network, it is crucial to explain how the neurons are activated. As an example, the sum of the weighted inputs and the bias, z_1^3, for the single output neuron n_1^3 in layer 3 of Figure 8 can be calculated according to

$$z_1^3 = w_{1,1}^3 a_1^2 + w_{2,1}^3 a_2^2 + w_{3,1}^3 a_3^2 + b_1^3. \tag{6}$$

Note that the input layer is counted as layer 0. The notation $w_{i,j}^k$ refers to one specific weight. The superscript refers to the layer the weight is received by, currently $k = 3$. The subscript i, j refers to the position of the relevant neurons in their respective layers: i for the sending neuron and j for the receiving neuron. For example, the weight associated with the highlighted connection in Figure 9 would be referred to as $w_{3,1}^3$. Note that weights are assigned to the receiving layer. Similar notation applies for the output value of each neuron, denoted as a_i^k, but since this value is the same for all connections originating in the neuron there is no need to specify which neuron it is pointing to in the receiving layer. The bias of a neuron n_j^k is denoted as b_j^k.

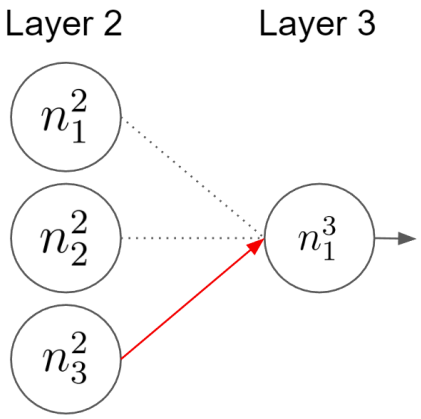

Figure 9: Highlight of the connection related to the weight $w_{3,1}^3$ and the output a_3^2.

In more general terms, the activation $a_j^k \in \mathbb{R}$ of a neuron n_j^k with position j in the layer k is calculated using the outputs from the entire previous layer $a_i^{k-1} \in \mathbb{R}$, $i \in \{1, 2, .., n\}$, where $n \in \mathbb{N}$ is the size of the layer $k-1$. The calculation also involves a bias b_j^k, and a set of weights $w_{i,j}^k \in \mathbb{R}$. These values, denoted z_j^k, are then fed through a selected activation function $f : \mathbb{R} \to \mathbb{R}$ according to

$$a_j^k = f(z_j^k) = f(\sum_{i=1}^{n}(w_{i,j}^k a_i^{k-1}) + b_j^k). \tag{7}$$

Within this thesis, each layer uses a specified activation function to calculate the output for each neuron. The selection may vary between layers. The networks used in this thesis utilize the following two activation functions: the sigmoid function, which can be seen in Figure 10, and the rectified linear unit (ReLU) function, which can be seen in Figure 11. There are no well defined rules regarding optimal choices of activation functions for each specific application of ANNs [19]. These were selected as they are commonly used in practice, and showed good results for this application.

Since layers are structured as columns, it stands to reason that activations for an entire layer can be efficiently described using column vectors. The activations for the layer k can be calculated by replacing the sums with matrix operations:

$$\boldsymbol{a}^k = \boldsymbol{f}(\boldsymbol{W}^k \boldsymbol{a}^{k-1} + \boldsymbol{b}^k) \tag{8}$$

Here, each row of the matrix $\boldsymbol{W}^k \in \mathbb{R}^{m \times n}$ corresponds to the weight vector $\boldsymbol{w}_j^k \in \mathbb{R}^n$ for each neuron $j \in \{1, 2, ..., m\}$ in the receiving layer k. The vector $\boldsymbol{a}^{k-1} \in \mathbb{R}^n$ is a column vector containing all the outputs of the previous layer $k-1$. The activation function $\boldsymbol{f}$ is applied element wise, e.g for an example vector v, see (9).

$$\text{The sigmoid function: } \sigma(z) := \frac{1}{1+e^{-z}}$$

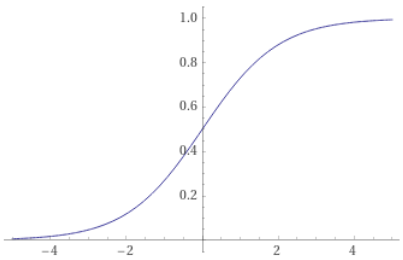

Figure 10: The sigmoid function visualized. The horizontal axis represents z, the vertical axis represents the activation.

$$\text{The ReLU function: } R(z) := \max(0, z)$$

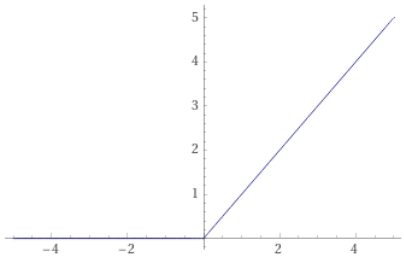

Figure 11: The ReLU function visualized. The horizontal axis represents z, the vertical axis represents the activation.

$$\boldsymbol{f}\left(\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}\right) = \begin{bmatrix} f(v_1) \\ f(v_2) \end{bmatrix} \tag{9}$$

By repeating (8), a network with $K \in \mathbb{N}$ layers can be represented in a single function $\boldsymbol{y}$, where $\boldsymbol{x}$ is the input vector and $\boldsymbol{\theta}$ represents the configuration of weights and biases for the entire network, known as the parameter vector. The output of the neural network is then the output of a function $\boldsymbol{y} : \mathbb{R}^m \times \mathbb{R}^M \to \mathbb{R}^d$,

$$\boldsymbol{y}(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{f}^K(\boldsymbol{W}^K \boldsymbol{f}^{K-1}(...\boldsymbol{f}^2(\boldsymbol{W}^2 \boldsymbol{f}^1(\boldsymbol{W}^1 \boldsymbol{x} + \boldsymbol{b}^1) + \boldsymbol{b}^2)...) + \boldsymbol{b}^K). \tag{10}$$

To break this function down, consider first the activation functions $\boldsymbol{f}^1, \boldsymbol{f}^2, ..., \boldsymbol{f}^K$. These are the selected activation functions for each layer. The parameter vector $\boldsymbol{\theta}$ contains the weight matrices $\boldsymbol{W}^1, \boldsymbol{W}^2, ..., \boldsymbol{W}^K$ and bias vectors $\boldsymbol{b}^1, \boldsymbol{b}^2, ..., \boldsymbol{b}^K$ for each layer, and is of length M. The property of each layer being dependent on all previous

layers is captured using functions within functions. The deepest parenthesis in the function, $\boldsymbol{f}^1(\boldsymbol{W}^1\boldsymbol{x}+\boldsymbol{b}^1)$, contains the input vector $\boldsymbol{x}$ as well as the weights and biases from the first layer. These values are put through the activation function $\boldsymbol{f}^1$, resulting in the output vector $\boldsymbol{a}^1$ from the first layer. This output vector is then used to make the same calculation for the second layer via $\boldsymbol{f}^2$. By repeating this for all layers, the final output vector $\boldsymbol{y}(\boldsymbol{x}, \boldsymbol{\theta})$ can be calculated. This output vector will have length d, which corresponds to the amount of output neurons in the network.

Interpreting the output vector can be done in different ways. For example, when classifying the aforementioned MNIST dataset of digits with ten classes, a reasonable choice would to have an output vector of length 10, one for every digit. The activation of every neuron in the output layer would correspond to how similar the network considers the input image to each class. The specific answer could then be selected as the neuron with the strongest activation. In this thesis, another approach called *binary classification* will be used, as only two classes [Not piggybacking, Piggybacking] will be considered. The networks will use structures with a single output neuron using sigmoid activation: $\boldsymbol{y}(\boldsymbol{x}, \boldsymbol{\theta}) =: \sigma_{output}$, a scalar with range $(0, 1)$. The classification is then made according to

$$\text{Classification}(\sigma_{output}) = \begin{cases} \text{Not piggybacking}, & \text{if } \sigma_{output} < 0.5 \\ \text{Piggybacking}, & \text{if } \sigma_{output} > 0.5 \end{cases}. \tag{11}$$

2.2.2 Optimization

So far Section 2.2 has covered how a neural network makes a decision. But the question remains how these decisions can be more than a wild guess. The set of weights and biases $\boldsymbol{\theta}$ is at the core of this functionality. By selecting these parameters in an appropriate manner, the decisions made by the neural network can become increasingly accurate. This iterative selection process of parameters, typically referred to as *training* the network, will be covered within this section. The parameters $\boldsymbol{\theta}$ will be referred to as *trainable parameters*.

Cost functions To enable quantitative evaluation of a network's decision making, a *cost function* will be utilized. The cost function will be defined in such a way that a lowering in the cost is equivalent to an increase in accuracy. The purpose of optimization will then be to minimize the cost function. Cost is also typically referred to as *loss*, and will be used interchangeably.

A basic example of a cost function is the *quadratic cost function*, defined as

$$C(\boldsymbol{X}, \boldsymbol{\theta}) := \frac{1}{2n} \sum_{i=1}^{n} ||\hat{\boldsymbol{y}}(\boldsymbol{x}_i) - \boldsymbol{y}(\boldsymbol{x}_i, \boldsymbol{\theta})||^2. \tag{12}$$

The notation $||.||$ refers to the L_2 norm, $\boldsymbol{X} \in \mathbb{R}^{n \times m}$ is the full dataset of inputs, specifically a matrix where each row corresponds to an input vector. Every input vector $\boldsymbol{x}_i$ has length $m \in \mathbb{N}$, and there is a total of $n \in \mathbb{N}$ inputs. The function $\hat{\boldsymbol{y}} : \mathbb{R}^m \to \mathbb{R}^d$ outputs the label vector for every input vector. The cost function calculates the square of the length of a vector with the difference between the label vector $\hat{\boldsymbol{y}}(\boldsymbol{x}_i)$ and the output vector $\boldsymbol{y}(\boldsymbol{x}_i, \boldsymbol{\theta})$. For example, when using one output neuron i.e. $d = 1$ with sigmoid activation, $\hat{\boldsymbol{y}}(\boldsymbol{x}_i)$ will be a scalar with value 0 or 1 for each $\boldsymbol{x}_i$. An uncertain answer close to the breakpoint 0.5, such as $\boldsymbol{y} = 0.44$, might result in a correct classification if the answer is $\hat{\boldsymbol{y}} = 0$, 'Not Piggybacking', but it will still incur a cost. The quadratic nature of the cost function is to further increase the penalty of classifications which are far off the mark, relative to small errors. The costs for all n inputs in the set $\boldsymbol{X}$ are summed together and normalized, resulting in the final cost C.

Another example of a cost function, specifically the function that will be used within this thesis, is the binary cross-entropy function. This is defined as

$$C(\boldsymbol{X}, \boldsymbol{\theta}) = -\frac{1}{n} \sum_{i=1}^{n} \left(\hat{y}(\boldsymbol{x}_i) \cdot log(y(\boldsymbol{x}_i, \boldsymbol{\theta})) + (1 - \hat{y}(\boldsymbol{x}_i)) \cdot log(1 - y(\boldsymbol{x}_i, \boldsymbol{\theta})) \right). \quad (13)$$

Here, the function $y : \mathbb{R}^m \times \mathbb{R}^M \to \mathbb{R}$ is similar to $\boldsymbol{y}$, except it requires $d = 1$. Similarly, $\hat{y} : \mathbb{R}^m \to \mathbb{R}$, will again output the correct label for a sample $\boldsymbol{x}_i$, but in this case the output is always a scalar: $\hat{y}(\boldsymbol{x}_i) \in \{0, 1\}$. Because of this property, one of the terms $\hat{y}(\boldsymbol{x}_i)$ and $(1 - \hat{y}(\boldsymbol{x}_i))$ will always be 0, while the other will be 1. The individual terms in the sum that runs over the dataset of size n will therefore either be $log(y(\boldsymbol{x_i}))$ if the label is 1, and $log(1 - y(\boldsymbol{x_i}))$ if the label is 0.

These terms are essentially the logarithm of the probabilities of each entry $\boldsymbol{x}_i$ belonging to its class $\hat{y}(\boldsymbol{x}_i)$. Because the logarithm is an exponential curve that is large close to zero, it penalizes low probabilities. By minimizing the binary cross-entropy, the model will become increasingly certain of which category a new sample belongs.

Backpropagation After defining a cost function to evaluate performance, the following challenge is to select a set of weights and biases such that (13) is minimized. The explanation presented in this subsection will be a summary of the method explained in Chapter 2 of [19], an example based on simple feedforward networks. The essence of this process is to calculate the gradient of the cost function with regard to $\boldsymbol{\theta}$, $\nabla_{\boldsymbol{\theta}} C$, which is a vector containing the partial derivatives for the cost function with regard to every weight and bias in the network, $\frac{\partial C}{\partial w^k_{i,j}}$ and $\frac{\partial C}{\partial b^k_j}$. Due to the large amount of connections in a typical network, the gradient will be a vector of considerable dimensionality. For example, the networks used during this thesis had a parameter

count ranging from tens of thousands to millions. But the core of the computational complexity is the fact that a weight in a given layer will be impacted by the configuration of every trainable parameter in previous layers. Calculating $\nabla_{\boldsymbol{\theta}} C$ is at the essence of machine learning, and is typically done using the *backpropagation algorithm* or a variation thereof such as *Adam*, which will be explained in the paragraph titled Adam. For readers unfamiliar with the backpropagation process, the authors would like to suggest [26] for a brief introduction, or [19] for a more in-depth yet pedagogical explanation.

The first step of explaining the backpropagation algorithm is defining the error δ_j^k of a neuron j in layer k according to

$$\delta_j^k := \frac{\partial C}{\partial z_j^k}. \tag{14}$$

Again, z refers to the sum of weighted inputs and bias before an activation function is applied. The following method will allow for calculation of the error for every neuron in the system, as well as using those values to calculate $\frac{\partial C}{\partial w_{i,j}^k}$ and $\frac{\partial C}{\partial b_j^k}$ which will be contained within the gradient. The first step is to calculate the error for each neuron in the final layer K according to

$$\delta_j^K = \frac{\partial C}{\partial a_j^K} f'(z_j^K). \tag{15}$$

The right side of (15) contains the effect of the neuron's activation on the cost function multiplied by the derivative of the activation function for that specific value of z. Both of these terms can be computed after feeding forward through the network to calculate the output vector. The second step is to define an equation that allows for a layer's error vector $\boldsymbol{\delta}^k$ to be calculated given $\boldsymbol{\delta}^{k+1}$,

$$\boldsymbol{\delta}^k = ((\boldsymbol{w}^{k+1})^T \boldsymbol{\delta}^{k+1}) \odot \boldsymbol{f}'(\boldsymbol{z}^k). \tag{16}$$

Since the error for the final layer, $\boldsymbol{\delta}^K$ can be calculated using (15) above, (16) can be used to calculate the errors backwards through the entire network. This approach is the origin of the backpropagation algorithm's name. The operator $\odot$ refers to the Hadamard product. Once this action is performed, the partial derivatives for the gradient can now be calculated according to the following equations (17) and (18).

$$\frac{\partial C}{\partial b_j^k} = \delta_j^k \tag{17}$$

$$\frac{\partial C}{\partial w_{i,j}^k} = a_i^{k-1}\delta_j^k \tag{18}$$

Gradient Descent After calculating the gradient, the set of weights and biases will be adjusted. The gradient is a vector of partial derivatives, in other words a measurement of how much every parameter impacts the cost, and in which direction. By taking a step in the direction of the negative gradient, the cost is reduced. Iterating this process to gradually reduce the cost function is referred to as *gradient descent* [27]. Parameters with a large partial derivative get a relatively larger nudge, while those with a smaller impact receive less adjustment. The overall size of adjustment is determined by η, referred to as the *learning rate*.

For an arbitrary trainable parameter (any weight or bias) p the adjustment Δp is calculated as

$$\Delta p = \eta \cdot \frac{\partial C}{\partial p}. \tag{19}$$

For the entire configuration of the network, the parameters will be updated according to

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \cdot \nabla_{\boldsymbol{\theta}} C(\boldsymbol{X}, \boldsymbol{\theta}_t). \tag{20}$$

The learning rate is a *hyperparameter*, one of the overall settings configured prior to any training. Other examples of hyperparameters are the number of *epochs* and the *batch size*. An epoch is a training performed over the entire set of training data. The batch size is the amount of input vectors processed before a parameter adjustment is performed.

Gradient descent is performed in the case where the gradient is calculated using the entire dataset, i.e. when the batch size is equal to the amount of input vectors. This can be problematic when the dataset is very large, for various reasons. The most apparent reason experienced during this thesis was limitations in GPU memory, since larger batches require more memory to process, but configuring this hyperparameter is also the focus of much research, for example [6], [16], [24]. When using a batch size of less than the total number of samples, steps will be performed using a gradient calculated for a single batch. Thus multiple timesteps will be made during each epoch, according to

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \left(\frac{1}{B} \sum_{i=1}^{B} \nabla_{\boldsymbol{\theta}} C(\boldsymbol{x}_i, \boldsymbol{\theta}_t) \right), \tag{21}$$

where $\boldsymbol{b} \ni \boldsymbol{x}_i$ is a batch sampled from $\boldsymbol{X}$, and B is the batch size. This variation of gradient descent is referred to as *mini-batch gradient descent*. The reader might be familiar with the term *stochastic gradient descent*, which refers to the special case $B = 1$.

Adam The choice of optimizer used for training the networks within this thesis is Adaptive Moment Estimation, or *Adam*. It is an optimizer that shows robustness and performance in several case scenarios, and is preferred in all types of applications by some neural network researchers [5]. Adam was designed to combine the advantage of two, at the time, popular methods [13]. The first is AdaGrad, which is designed to function well on sparse gradients. The second is RMSProp, which performs well when working with non-stationary settings, i.e. when the theoretical optimal function the network is trying to learn changes over time.

A key feature of Adam is the inclusion of the exponential moving averages of the gradient ($\boldsymbol{m}_t$), and a moving average of the squared gradient ($\boldsymbol{v}_t$) in the update rule [13]. The required inputs are the learning rate η, the exponential decay rates $\beta_1, \beta_2 \in [0, 1)$, the cost function C, and the initial parameter vector $\boldsymbol{\theta}_0$. The parameters $\boldsymbol{m}_0, \boldsymbol{v}_0,$ and the timestep t are initialized as vectors of zeros. These moving averages decay at rates β_1, β_2 respectively, and are updated for each timestep according to (23) and (24). All operations in the following equations are performed element wise.

$$\boldsymbol{g}_{t+1} \leftarrow \nabla_{\boldsymbol{\theta}} C(\boldsymbol{X}, \boldsymbol{\theta}_{t+1}) \tag{22}$$

$$\boldsymbol{m}_{t+1} \leftarrow \beta_1 \cdot \boldsymbol{m}_t + (1 - \beta_1) \cdot \boldsymbol{g}_{t+1} \tag{23}$$

$$\boldsymbol{v}_{t+1} \leftarrow \beta_2 \cdot \boldsymbol{v}_t + (1 - \beta_2) \cdot (\boldsymbol{g}_{t+1})^2 \tag{24}$$

Because the moving averages are initialized as 0, they will have an initialization bias. This is counteracted by initialization bias correction which will not be covered in detail, but can be read at [13, p.3]. The bias-corrected moving averages are $\hat{\boldsymbol{m}}_t = \boldsymbol{m}_t/(1 - \beta_1^t)$, $\hat{\boldsymbol{v}}_t = \boldsymbol{v}_t/(1 - \beta_2^t)$, and will be used in Adam's update rule:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \cdot \frac{\hat{\boldsymbol{m}}_t}{\sqrt{\hat{\boldsymbol{v}}_t + \epsilon}}. \tag{25}$$

Here, ϵ is a scalar typically set to $\epsilon = 10^{-8}$ used to avoid division by zero. Note that (22) corresponds to gradient descent. If mini-batch gradient descent is desired, (22) should be replaced with

$$\boldsymbol{g}_{t+1} \leftarrow \left(\frac{1}{B} \sum_{i=1}^{B} \nabla_{\boldsymbol{\theta}} C(\boldsymbol{x}_i, \boldsymbol{\theta}_t) \right), \tag{26}$$

similar to (21).

Adam was selected as the optimizer of choice due to several factors. The primary factor was due to familiarity, as it's a common occurrence in machine learning courses experienced by one of the authors. This decision was reinforced by considering comparisons of algorithms such as [25] where Adam is described as possibly the best choice, and [5] where its performance is among the top three for all tested datasets.

2.2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are some of the most commonly used neural networks in the world. They are a type of feedforward network. Their primary use is for image classification. The way it uses convolutions makes it efficient at extracting shapes and features from images.

The typical structure of a CNN can be seen in Figure 12. Like all neural networks, it starts with an input layer. Following the input layer comes a structure containing a convolutional layer followed by a pooling layer. This structure can then be repeated until the results are satisfactory. This sequence of layers split the input image into smaller images called filters. The job of the filters is to capture small shapes and features of the original image, that might repeat in other images of the same class. For example if the input image is of a bird, one or many of the filters might try to capture the beak while some other filters might try to capture the feet. Once the image has been split into small enough filters that capture all interesting features, the output of the final pooling layer is flattened and fed into one or many dense layers. Eventually you reach the output layer that has one or many output neurons.

The Convolutional Layer When people think of a convolutional layer they usually think of the two dimensional kind, but CNNs also exist for one dimension and three dimensions. A convolutional layer uses a kernel of a fixed size, the kernel moves along all dimensions of the input and performs element wise multiplication between the kernel and the current kernel sized window of the input. In the example in Figure 13, a two dimensional convolution is shown. It uses a $(3, 3)$ kernel with different weights. It is currently on the fourth convolution, moving from left to right and top to bottom.

The Pooling Layer The pooling layer's job is to reduce the dimensionality of the network. This is done by aggregating the output of the convolutional layer using some operator or function. The most common one, and the one used in this thesis is Max Pooling. It uses the max operator [7]. In Figure 14 another method called Average Pooling is also included to highlight the differences between pooling methods.

3D CNN and Voxelization for Point clouds Convolutional networks usually require highly regular formats for the input data [22]. A direct way of using CNNs with point clouds is to utilize three dimensional convolution kernels. These work

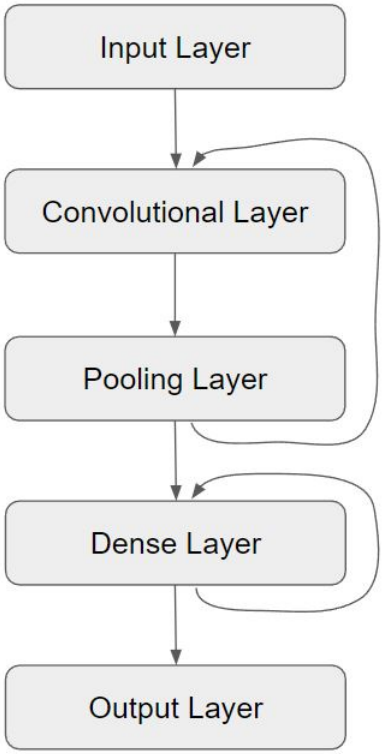

Figure 12: Typical structure of a CNN

similarly to convolutional kernels in one or two dimensions, but instead of moving the kernel along one or two axes the kernel moves along three axes. For this to work, every input needs to have the same dimensions. A common strategy to ensure this is voxelization [22]. A voxel is the three dimensional equivalent of a pixel. Voxelization means a room is defined with a fixed resolution of (X, Y, Z) voxels. This room contains voxels of size (x, y, z) in some unit (e.g. metres), depending on the amount of voxels and the size of the modeled area. For example, if the modeled room is a cubic meter $(1, 1, 1)$ m and the resolution is $(10, 10, 10)$ totaling a thousand voxels, each voxel will correspond to one cubic decimeter $(0.1, 0.1, 0.1)$ m. In this example as well as during the applications in this thesis the voxels are cubes, but they can be selected as any rectangular shape.

Once a room has been defined, every relevant point in the input point cloud is assigned to a voxel. This strategy has a lot of disadvantages. One disadvantage is that voxelization results in sparse data. As an example of this phenomenon, consider a corridor with a door. The door is a 1 meters wide, two meters tall, and the goal is to monitor the three meter area in front of the door with one of the radar modules described in Section 2.1. Assuming that people will walk completely straight in front of the door, the room can be defined as $(3, 1, 2)$ meters. To fully utilize the 5.8cm

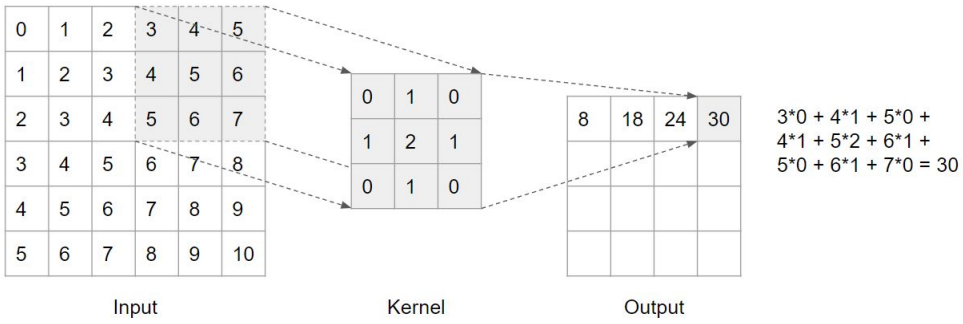

Figure 13: Example of a convolution in a convolutional layer

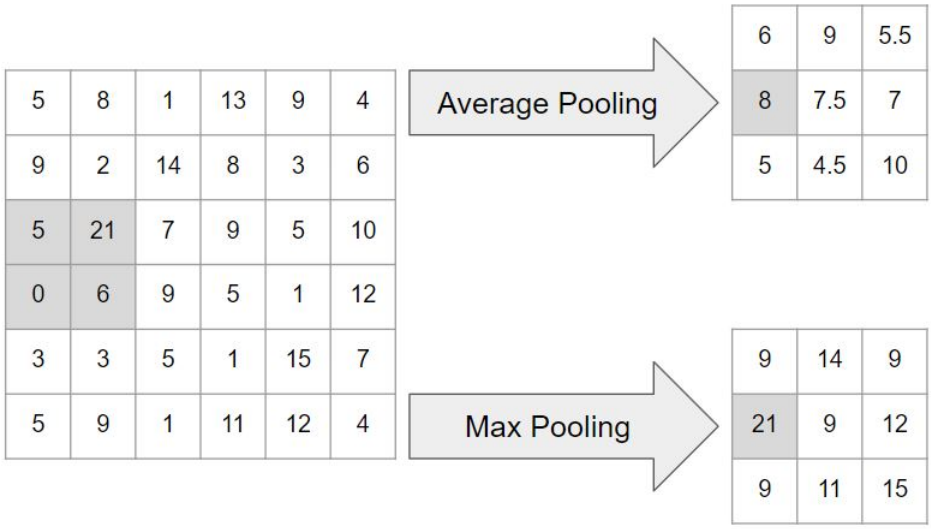

Figure 14: Differences between Average Pooling and Max Pooling

resolution of the radar, desired voxel size is selected as 5cm. This results in a resolution of $(60, 20, 40)$, totaling 48 thousand voxels. If the point clouds collected contain a median of 300 points, more than 99% of voxels will usually be empty in each input frame.

On top of making the data very sparse, this leads to large datasets, requiring significant amounts of memory and time for training.

2.2.4 PointNet

PointNet is a neural network design published in 2016, proposing a new way of using neural networks with point clouds. One significant contribution was a change in the input representation [22]. PointNet is designed for both classification and segmentation of high resolution point clouds, albeit in separate parts of the network structure.

The authors describe the architecture as "surprisingly simple", and summarize the key to their approach as "the use of a single symmetric function, max pooling" [22, p.1].

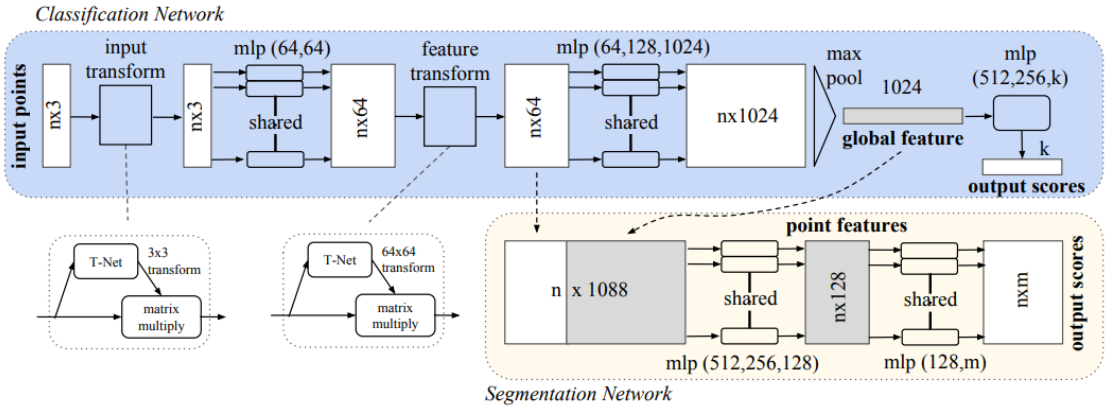

Figure 15: PointNet architecture [22, p.3]. The right half of the Classification Network is the key part for classifying point clouds, and was the basis for the PointNet implementation within this thesis. The notation mlp refers to multilayer perceptrons, which means a set of dense layers. The associated numbers refer to the layer sizes. Additionally, n is the number of points in each input frame, and k is the number of classes.

The purpose of the first part of the network's architecture is to make it invariant towards geometric transformations of the input [21]. This is performed by the Transformer Network (*T-Net*), which will not be explained in detail as its use within this thesis is very limited. This corresponds to the left half of the *Classification Network*, highlighted in blue in Figure 15. The same reasoning is applied for the Segmentation Network, which is not used within this thesis.

The approach used for handling PointNet's input data is to make it two dimensional. The dimensions of the input data to PointNet is $X \in \mathbb{R}^{n,\tau}$, where $n \in \mathbb{N}$ is the number of points, and $\tau \in \mathbb{N}$ the number of features. The features are typically the points' x-, y-, and z-coordinates, but it can be more than three.

Since point clouds are unordered datasets, an area of focus for the development of PointNet was to achieve invariance to reordering of the input points. When viewing the neural network as a function, this property can be viewed as the function being symmetric. A symmetric function is a function which has the same output regardless of the order of the inputs. An example of this would be a function that satisfies $f(x_1, x_2, ..., x_n) = f(x_2, x_1, ..., x_n) = f(x_1, x_n, ..., x_2)$ for all sets of inputs within the domain of f. Another example is max pooling, described in Section 2.2.3.

The approach was to implement a symmetric function that aggregates information from all n points into a single vector which is invariant to the input order. Their

idea was to "approximate a general function defined on a point set by applying a symmetric function on transformed elements in the set" [22, p.4], according to

$$f(\{x_1, x_2, ...x_n\}) \approx g(h(x_1), h(x_2), ..., h(x_n)). \tag{27}$$

In this equation, $f : 2^{\mathbb{R}^N} \to \mathbb{R}$, $h : \mathbb{R}^N \to \mathbb{R}^K$, and $g : \underbrace{\mathbb{R}^K \times ... \times \mathbb{R}^K}_{n} \to \mathbb{R}$ are all symmetric functions. The function h is approximated by three dense layers of sizes $(64, 128, 1024)$, while g is approximated using a max pooling layer, and n is the amount of input points. The process up to this point results in a *global feature vector* of size 1024 which is highlighted in the top right of Figure 15.

This resulting global feature vector is then fed through another set of three dense layers of sizes $(512, 256, k)$. It takes the global feature vector as input, and results in an output vector of size k. These six dense layers as well as the max pooling layer are the parts of the PointNet architecture that will be mainly utilized within this thesis.

2.2.5 Recurrent Neural Networks (RNN) and LSTM

A recurrent neural network differs from a feedforward network by having feedback connections, in addition to feedforward connections. This essentially means that the connections in the network are allowed to form cycles. This allows neurons in the network to take previous input and output into account during activation. In Figure 16, the feedback connections are the arrows pointing from the output layer to the first hidden layer. The feedback connections essentially allows the RNN to have a memory, storing information for use at a later time. Because of this, a recurrent neural network can process sequential data. This makes them very useful for certain tasks that feedforward networks find tough. An example of such a task is natural language processing. The memory of an RNN allows it to not only take one word or letter into account at a time, but to look at an entire sequence of words (a sentence) to get a better understanding of the context. The property that recurrent neural networks can consider sequential data leads to them also being referred to as *temporal*, as they can process data in the time dimension.

Traditional RNNs suffer from a problem called *the vanishing gradient problem*. This problem leads to RNNs having short memory [20]. When doing backpropagation through an RNN, the cycles in the network influences the weights in a way that leads to some gradients either diverging to infinity or becoming infinitesimal. This leads to neurons early in the network learning slowly, and sometimes not learning at all. To combat the vanishing gradient problem, variations of RNN that allows the neural network to remember for longer has been developed. Two commonly used variations of RNN are *Long Short-Term Memory* networks (*LSTM*), first suggested by Hochreiter and Schmidhuber in 1997 [11], and Gated Recurrent Units (GRU) first suggested

by Cho et. al in 2014 [4]. The biggest difference between these implementations, and the traditional RNN are how the feedback connections are handled. In this project, the LSTM network is used and will be described in greater detail in the next paragraph.

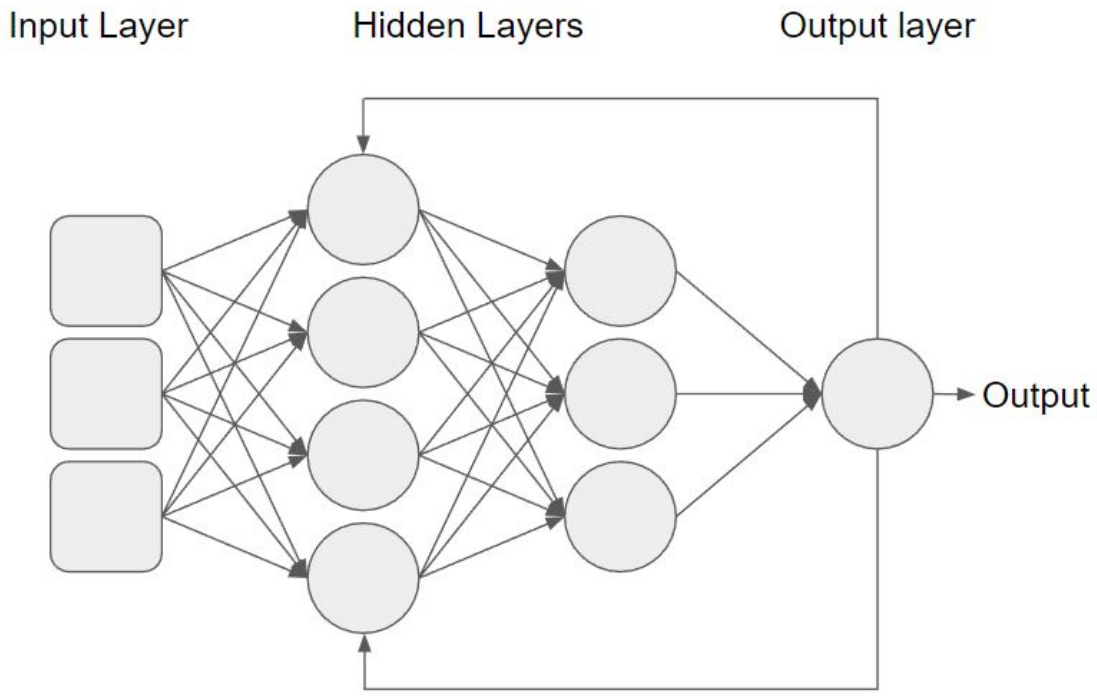

Figure 16: Example RNN with three inputs (squares), two hidden layers of neurons (circles), and a single output neuron with recurrent connections to the hidden layers (circle).

LSTM An LSTM (Long Short-Term Memory) is a type of RNN (Recurrent Neural Network). Just like an RNN, an LSTM has feedback connections. Therefore an LSTM looks a lot like an RNN but with other ways of handling memory. LSTMs learn to keep only the important information in memory, allowing them to remember for longer periods of time, hence the name Long Short-Term Memory.

Memory LSTM layers contain memory cells. This is how the network remembers things, and takes past output into account. There are several different structures of memory cells, but all include an input gate and an output gate [30]. The variant used in this project also includes a forget gate.

In the memory cell in Figure 17 c(t) is the cell state at point t, this is where the memory is stored. The output is denoted as h(t) and is also called the hidden state. The input data is denoted as x(t). Small t denotes one step in whatever is the

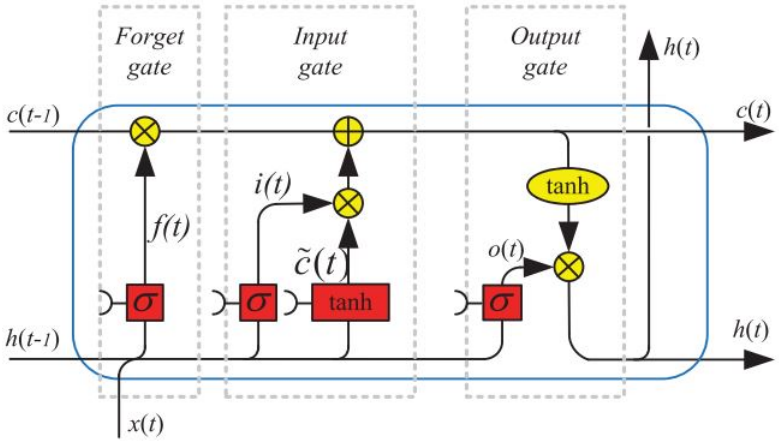

Figure 17: Depiction of an LSTM Memory cell with a forget gate [30].

sequential domain. It could for example be a second if the network operates in the time domain. As can be seen in the left part of Figure 17, the previous cell state and hidden state is part of the cells activation. The length of the LSTM layers memory is essentially how many memory cells are stacked in succession.

Bidirectional LSTM The bidirectional variant of LSTM was first suggested by Graves and Schmidhuber in 2005 [8]. It combines BRNN (Bidirectional Recurrent Neural Network) with LSTM. The bidirectional variant essentially allows the network to consider the sequence in both directions. In a time-context this would be past to future and future to past. It accomplishes this simply by doubling the size of the LSTM layer. One half of the layer sees the data in the original order, and the other half of the layer sees the data in reverse order. In the end, the output of the layer takes both orders into account.

2.2.6 Overfitting

Overfitting is a problem in machine learning. It means that a model has been too well adjusted to the training data, making it perform worse on unseen data [10]. Overfitting is related to how much training data you have and the complexity of the network. In general more training data and fewer weights and biases, leads to less overfitting. A neural network can be seen as a function. An overly complicated function can be too adjusted to the data. Figure 18 shows how an overly complicated function might look, compared to a simpler, more well fitted function. This overly complicated function will most likely generalize worse to unseen data. Like the function in Figure 18, an overly complicated neural network with too many weights and biases can be too well adjusted to the data. If a dataset used for training a neural network is too small, it is more likely that it will be hard to tell the signal from the

noise. This essentially means that in a smaller dataset, the neural network will learn a pattern on both the good data and the noise. While in a larger dataset it will be easier for the neural network to learn what is information and what is noise, leading to the network filtering out the noise.

There are ways of preventing overfitting from happening, two of those are *Regularization* and *Dropout*. Regularization and dropout will be discussed in greater detail in this section. An overview of more approaches to preventing overfitting can be read about in [29].

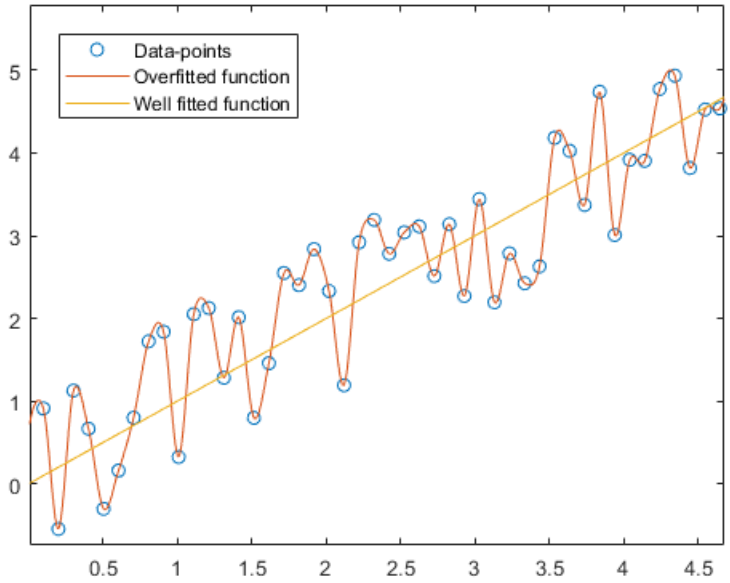

Figure 18: Example of one overfitted function and one well fitted function.

Regularization Regularization is a way of penalizing less important features. Less important features means that they have less influence on the final classification. When a network has many features it can be hard to tell which ones are important. The regularizer attempts to minimize the weights of the less important features. This is done by adding a regularization or penalty term to the cost function, according to

$$C_r(\boldsymbol{X}, \boldsymbol{\theta}) = C(\boldsymbol{X}, \boldsymbol{\theta}) + \alpha\Omega(\boldsymbol{\theta}). \tag{28}$$

In the equation above, C_r is the regularized cost function, C is the cost function, $\boldsymbol{\theta}$ is the set of weights and biases, $\alpha > 0$ is a constant used to limit how aggressive the regularization is and finally Ω is the regularizer. In this project L_2 regularization is used. When doing L_2 regularization, the regularizer is the Euclidean distance over the weights. Inserting this into (27), gives us

$$C_r(\boldsymbol{X}, \boldsymbol{\theta}) = C(\boldsymbol{X}, \boldsymbol{\theta}) + \alpha||\boldsymbol{\theta}||_2. \tag{29}$$

Dropout Dropout means that during training, neurons are dropped with probability p. The optimal value for p varies, but is usually around 0.5 for hidden layers and 0.2 for input layers. Figure 19 shows an example network where dropout has been applied. If a layer in a network is very simple, e.g. consisting of only one neuron, dropout should not be used. It is also not recommended to use dropout in the output layer.

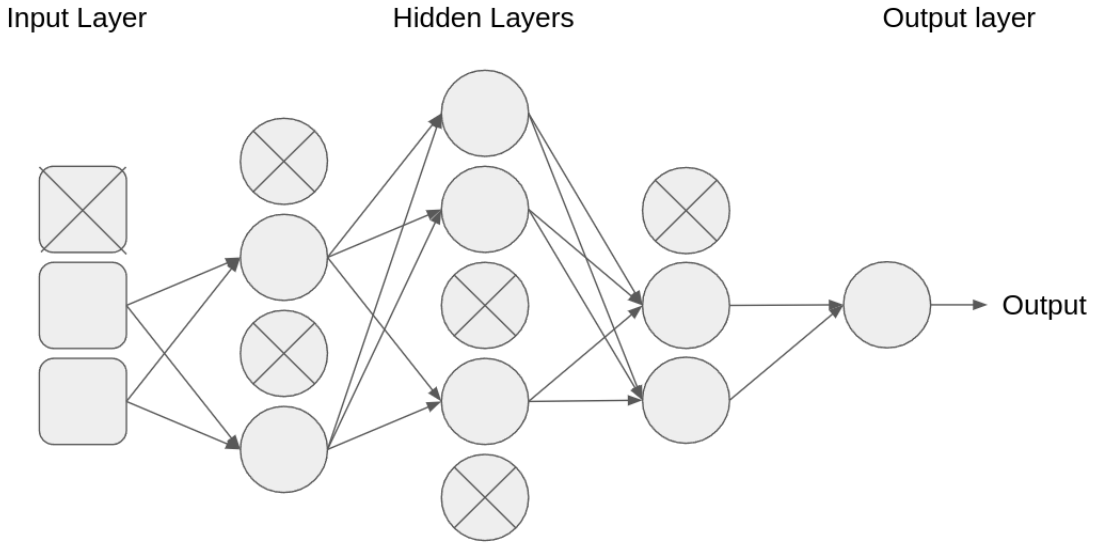

Figure 19: Example network where dropout has been applied to the input layer and the hidden layers.

Dropout ensures that neurons don't become too dependent on specific connections. When predicting, each weight is multiplied by the probability p of that neuron being dropped during training. It is a method that has been empirically proven to make neural networks less subject to overfitting. Dropout makes training a network slower, but it allows it to be trained for a larger amount of epochs.

2.2.7 F-score

The *F-score* is used to evaluate performance of neural networks. In this thesis it will be used to compare different network structures, and different combinations of features. The F-score combines the two metrics *precision* and *recall* [28]. Precision is a metric that shows what percentage of the positives are *true positives (tp)*, and recall is a metric that shows what percentage of all *true positives (tp)* and *false negatives (fn)* are actually *true positives (tp)*. Because of how precision and recall utilizes

positives and negatives, this can be calculated for every class in a neural network. One can also calculate a F-score for the entire dataset by averaging the F-scores of each class. Precision and recall are calculated according to (30) and (31) respectively. Figure 20 is a helpful tool when trying to understand precision and recall.

$$\text{precision} = \frac{tp}{tp + fp} \tag{30}$$

$$\text{recall} = \frac{tp}{tp + fn} \tag{31}$$

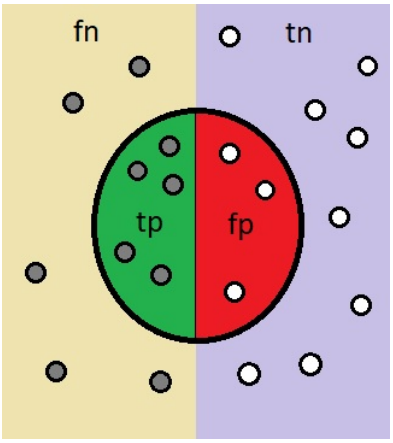

Figure 20: Helpful figure for understanding precision and recall. tp are true postives, fp are false positives, fn are false negatives and tn are true negatives.

In this thesis, F_1-*score* is used. The F_1-score is defined as the harmonic mean of the precision and the recall, i.e.

$$F_1 = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{2tp}{2tp + fp + fn}. \tag{32}$$

2.3 Camera based Piggybacking detection

The camera based piggybacking detection solution that will be used as a comparison to the radar based solutions has been developed by Axis. Therefore all details can't be described due to confidentiality.

The camera solution starts by doing segmentation on the camera feed, in order to attempt to separate the foreground from the background. After separating the background and foreground, it uses Hidden Markov Models, or HMM for short to track the foreground subjects. A HMM is a statistical model that tries to model events and event transitions that are observable, but which depends on events that are not observable [23].

To tailor the camera solution to a specific problem in a specific environment, a couple of settings can be tweaked. One of these settings is the sensitivity of the camera solution. When setting the sensitivity too low, it sometimes doesn't register when a person walks past. When it is set too high, it sometimes registers 2 or 3 people when actually only one walks past. After trying different sensitivity levels, the default values appeared to be optimal for the piggybacking problem. With the default values the camera solution almost never misses registering a person walking past, and rarely registers too many people.

3 Method

The main part of the project was the development phase, entailing weeks of trial and error of various network structures, seeking continuous improvement in terms of accuracy and robustness. There were no specific quantified goals in terms of accuracy for (AP1). Instead, this phase of the project would have its scope determined by a running evaluation of how much time was remaining for the thesis, combined with how well the current network was performing relative to the expectations of the authors. In other words, it was an iterative process with an arbitrary goal. Much of this chapter is not required for defining how the research questions were answered, its purpose is to highlight some key learnings that might be of interest for further development in the case where this proof-of-concept is acted upon. Note that the terminology used in this chapter will include *networks*, referring to a single neural network, and *models* which refers to an entire processing pipeline. This is a distinction which becomes important after models including multiple networks are introduced.

3.1 Generation and preprocessing of data

3.1.1 Hardware setup

The capturing devices used consisted of two radars developed by Axis and one camera developed by Axis. The room used for recording was a conference room in the office building, with the left wall covered in whiteboard, while the others were made of glass. The height of the roof was 2.6 m. One radar was attached to the ceiling above the door, directed in an angle of 66° below horizontal. This radar will be referred to as some variant of *roof-radar*. The other was placed at the side of the door at a height of 1.4 m, angled horizontally and perpendicular to the wall. This radar will be referred to as the *reader-radar*. A sketch of how the radars were placed can be seen in Figure 3. The dome camera unit was placed along the roof roughly 1.5 m in front of the door, facing vertically downwards.

The radar units output 10 frames per second, each frame consisting of a point cloud. Each point contains a variety of data, not all of which is used within this thesis. At most, five pieces of information is used as input to the networks from each point: X-coordinate, Y-coordinate, Z-coordinate, Radar Cross-Section (RCS), and radial velocity. RCS is a measure of how detectable an object is by the radar. A higher RCS value means that the point represents something that is seen more clearly. The radial velocity is the speed at which a point is moving towards- or away from the radar. It can therefore be both positive and negative.

3.1.2 Categorization of data

To test the adaptability and robustness of the models, multiple categories of scenarios were introduced. Some scenarios were designed to resemble situations that a model

would be faced with in a real-life application, both normal activities such as a person carrying a large object, and malicious activity such as two people trying to walk in a way that one person hides behind the other. Other scenarios were simply introduced after performing random tests to evaluate a model's robustness and having it fail, such as one person walking sideways. The categories abbreviated as (NP) and (P) will be referred to as the *main categories*, while the other will be referred to as *subcategories*. (NP) along with its subcategories include one person and are labeled as 'Not Piggybacking', whereas (P) and its subcategories include two and are labeled as 'Piggybacking'.

- (NP): Not piggybacking: One person walking normally.
- (NP_s): Not piggybacking, sideways: One person walking in a sideways fashion with arms in a broad position by the hips.
- (NP_c): Not piggybacking, cardboard box: One person walking while carrying a large box in front of the body.
- (NP_b): Not piggybacking, backpack: One person walking while wearing a stuffed 60 liter hiking-backpack.
- (NP_f): Not piggybacking, fast: One person walking at twice the normal speed.
- (P): Piggybacking: Two persons walking normally, with roughly half a meter of distance between them.
- (P_h): Piggybacking, hugging: Two persons walking with upper bodies in contact during the entire sequence.
- (P_d): Piggybacking, diagonal: Two persons walking in a way such that the person in the back is covered from the reader-radar's point of view.
- (P_f): Piggybacking, fast: Two persons walking at twice the normal speed.

The development process of this list was iterative, with categories being added and addressed one at a time. If a model could learn to correctly classify a new category simply by including a few minutes worth of training data, it would indicate that the model is adaptable. One way robustness was tested was to include a category in tests without including it in the training data, which was done for (NP_b). Robustness is used as a broad term to signify that a model could correctly classify scenarios similar to, but not exactly like what is contained in the training data. This is closely tied to overfitting, and how well the network generalizes.

Note that the initial list of categories simply consisted of the main categories, (NP) and (P). These were the only categories used for training and testing while experimenting with network structures. Therefore, the subcategories will not be utilized until Section 3.5.

3.1.3 Data recording

All training data has been recorded by the authors of this project. This was done by starting a recording on both radars, then walking back and forward inside a specified area until a satisfactory amount of data has been recorded. The main benefit of this method is that it greatly reduces the time needed to gather training data, as it eliminates the need for editing the recordings. A major part of editing would otherwise be handling the part of a piggybacking sequence where only one of the people is within the boundaries. Once a recording is finished, the first and last 150 frames are deleted to make sure that no frames with incorrect data are included in the training data, e.g. frames with only one person. In every recording, only one category is represented. In most of the training data, one or both of the authors are present. There is however a small amount of training and testing data that include different people. The total amount of training data for (NP) and (P) can be seen in Table 1.

	Training data		Testing data	
	(NP)	(P)	(NP)	(P)
Frames	36589	25866	10203	8733

Table 1: Table showing the amount of train- and test-data for (NP) and (P).

When starting a training, the training set is divided into a training set consisting of 75% of the data, as well as a validation set consisting of 25% of the data. This split is randomized and will be different for every instance of training, but does not change between epochs. If desired, the random seed can be specified to enable reproduction of results. The best model was determined by calculating the loss function for the network's classifications on the validation set and choosing the epoch with the lowest value.

Automated Testing Data The testing datasets are recorded using the same method as the training data. But these datasets are stored separately, sorted by category. The testing datasets are smaller, as their main purpose was to act as validation during the development phase, as well as being able to measure performance for a specific category. The automated tests are limited in their usefulness, as the black-box tests for the camera solution can not be performed using pre-recorded datasets. Therefore, any conclusions of this report will be drawn solely from manual tests, described in Section 3.7. However, after developing a satisfactory model, a final set of automated testis were performed. This set was run on a variety of configurations to showcase how each architecture performed compared to the others. The purpose of presenting this information is to aid in potential future research on the subject. The networks were trained on sets of data recorded by the roof-radar representing (NP) and (P). The training was performed over 50 epochs. The testing dataset also

consists of data from those two categories only. The total size of this testing dataset after being filtered was 9725 frames.

Note that the datasets were grown in increments during the development phase, and early decisions may have been made based on instances of training performed on smaller sets of training data. However all results presented in the thesis, including this section, are gathered from networks trained on the final datasets.

3.1.4 Point filtering

To remove uninteresting points created by noise or other disturbances, boundaries were implemented. These boundaries has dimensions $(3, 1.2, 2.4)$ meters. The dimensions are in (X, Y, Z), where X is the axis along which people will be walking (the length of the boundaries), the Y-axis is the width of the boundaries and the Z-axis is the height of the boundaries. Since there are two different positions for the radar modules, two different filtering functions were implemented to ensure that the boundaries correspond to the same space in the recording room. The bottom of the boundaries is aligned with the floor, and the center of the boundaries is aligned with the center of the doorway. Think of it as the corridor a person approaching the door perpendicularly would walk through.

Only the points inside the boundaries are kept. Figure 21 illustrates the filtering process. The point clouds presented are from the same frame, recorded in the reader position. The left plot shows the point cloud before filtering, along with the boundaries. The limits for the axes were automatically generated so that every point would fit within the figure. Note for example that there are points appearing up to 8m away from the radar in the X-direction. This occurs despite the room being roughly 4m long in that direction, and must therefore be reflections. The right plot shows the points that remain after filtering. In this plot all of the remaining points exist within the specified boundaries, which should significantly reduce the amount of noise before the points are used as input. The limits for the axes in this plot were manually selected to be half as long as the left plot to preserve the shape of the boundaries and allow for visibility around it.

If after filtering, less than a specified minimum number of points remain, the frame is considered empty. An empty frame will not be fed into the network. The minimum number of points ended up being set to 125 fairly early in the development. This number was not decided upon by any quantifiable means, since the accuracy generally went up the higher this number was set. The author's reasoning regarding this was that the frames with very few points were of such low quality that classification would be difficult regardless of the network's accuracy. The issue with setting the limit too high was that few frames remained after processing, which was predicted to be a possible issue for real-time access-control applications.

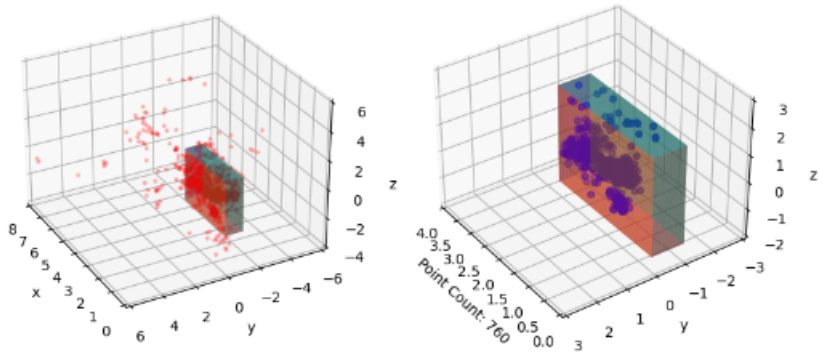

Figure 21: Illustration of the boundaries and point filtration.

The variation in points per frame is partially due to randomness. Based on the authors' observations, two frames captured 100 ms apart of virtually the same scene can contain significantly different amounts of points. But the main reason for variations is simply what happens inside the boundaries. The most straightforward reason for a drop in point count is simply that the person(s) are leaving the boundaries. However, there was also a systematic drop occurring within the boundaries due to geometrical limitations with the roof position. This is due to the angle of the radar, exacerbated by the fact that its signal is weaker near the edges of the radars field of view. The field of view is a cone, with an angle of about 120 degrees. As a person moves further away from the roof radar, the edge of its field of view will move down along the persons body. It turns out that the point filtering boundaries are larger than the 'optimal zone' for the roof radar, wherein a persons body can be captured in its entirety, resulting in a significant amount of frames being discarded. This was determined not to be a critical issue, instead viewed as an inherent weakness for the placement, to be considered in the comparison.

The numbers for the training data in Table 1 represent the total number of frames before any post-processing was done, and are roughly equal for both radar positions. The difference per recording is roughly 10-20 frames depending on which order the recordings were stopped in, and can be disregarded. However, due to the 125-point limit for acceptable frames to be put through the network, the actual numbers used for training are lower. Due to the aforementioned geometrical limitations of the roof radar position, significantly more frames are filtered out for the roof position than for the reader position. This can be seen in Table 2 and Table 3.

3.1.5 Velocity augmentation

Since the radial velocity was introduced as a feature, the models appeared more sensitive to the pace at which the people were walking. While including it increased

	Training data		Testing data	
	(NP)	(P)	(NP)	(P)
Raw frames	36589	25866	10203	8733
Filtered	30915	24970	8893	8384
Difference	-15.5%	-3.5%	-13%	-4%

Table 2: Table showcasing the effects of filtering on the reader position.

	Training data		Testing data	
	(NP)	(P)	(NP)	(P)
Raw frames	36589	25866	10203	8733
Filtered	19920	15053	4657	5068
Difference	-45.6%	-41.8%	-54%	-42%

Table 3: Table showcasing the effects of filtering on the roof position.

the overall accuracy on test sets similar to the training data, it generalized poorly for scenarios where people are walking very fast. At around twice the normal speed, none of the isolated attempts of detecting piggybacking were successful. Since recording data with the back-and-forth method used for this research would be difficult at such high speeds, particularly when involving two people, an attempt was made to implement data augmentation for the same purpose. This was achieved by multiplying the velocity vector with an augmentation function. Specifically, a step function $\boldsymbol{f}_n$ was used. The function takes the radial velocity vector $\boldsymbol{v} \in \mathbb{R}^m$ as input and multiplies it with a sinusoid according to

$$\boldsymbol{f}_n(\boldsymbol{v}) = (a \cdot \sin(\frac{[n/s]}{\omega}) + \delta) \cdot \boldsymbol{v}. \tag{33}$$

Here, n is the frame number, m is the amount of points in a frame, s determines how many frames should pass before a step is taken, ω determines how big each step should be along the sinusoid. Meanwhile, a and δ determine the amplitude and location of the curve. $[n/s]$ is a floor division operator, which gives the function its step property. The final values used for training were set to ($a = 0.75$, $\delta = 1.75$, $s = 500$, $\omega = 8$). The behavior this results in is that the function multiplies v by factors varying from 1 to 2.5. A step of size $\frac{\pi}{8}$ radians is taken along the sinusoid every 500 frames. The sinusoid has been visualized in Figure 22.

3.1.6 The final datasets

The size of the complete training dataset can be viewed in Table 4, and the testing dataset in Table 5. Note that training data for (P_d) is not used for the roof-radar. The

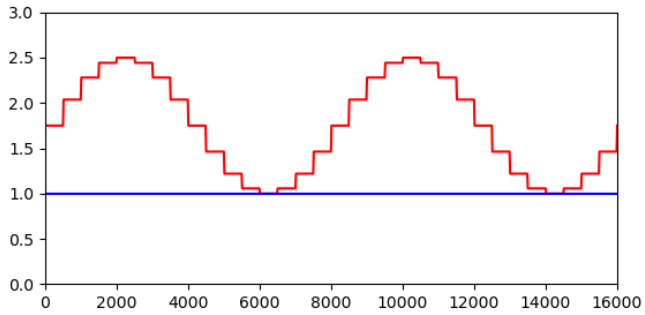

Figure 22: The sinusoid used for velocity augmentation, with frame number on the x-axis.

reason is that this category was never an issue for the roof-position, which was sufficiently robust to function for this category in all preliminary testing. This category is more challenging for the reader-radar due to geometrical aspects, an intentional part of the scenario's design. Note also that (NP_b) was not used in the training or testing data at all. This category will be relevant during the manual tests, explained in Section 3.7. The purpose of such a category was to test the robustness of the model. Finally, the categories including fast movement, (NP_f) and (P_f) do not have their own datasets. They are instead learned by applying the velocity augmentation. Actual fast walking is performed during the manual tests.

	(NP)	(NP_s)	(NP_c)	(P)	(P_h)	(P_d)
Raw frames	36589	13601	12653	25866	20628	6030
Reader filtered	30915	12305	9443	24970	18333	5376
Roof filtered	19920	8387	5904	15053	10335	not used

Table 4: The training datasets in terms of raw frames, as well after filtration for both radar positions.

	(NP)	(NP_s)	(NP_c)	(P)	(P_h)	(P_d)
Raw frames	10203	3001	2198	8733	3381	2471
Reader filtered	8893	2657	1669	8384	3064	2148
Roof filtered	4657	1362	926	5068	1836	925

Table 5: The testing datasets in terms of raw frames, as well after filtration for both radar positions.

3.2 Model 1: 3D CNN

For implementation of all neural networks in this thesis, the python library Keras was used. Keras is an API that is built upon Tensorflow. Tensorflow is a machine

learning platform developed by Google. Keras was picked due to familiarity of the authors and its widespread use in the machine learning community.

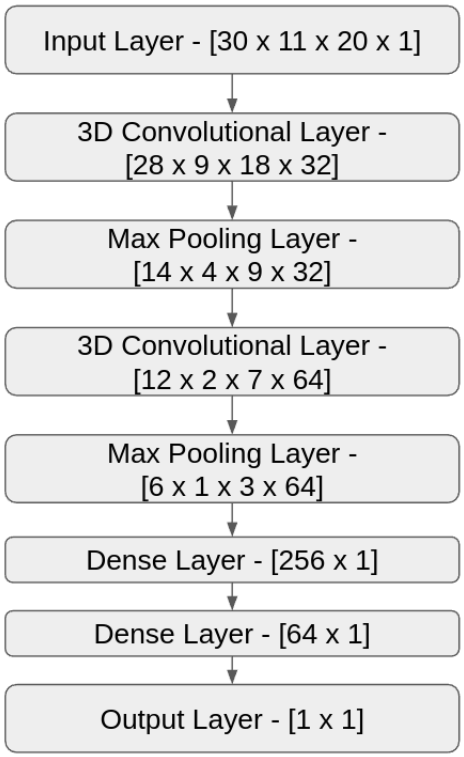

Figure 23: Structure of the 3D CNN, including dimensions in brackets.

The first approach involved using a rather simple 3D CNN with two convolutional layers, two max pooling layers and three dense layers. The full structure of the network, including the dimensions of each layer can be seen in Figure 23. One of the first problems encountered was that due to the randomness of the point clouds, every frame generated by the radar had varying amounts of points. Because of the requirement of having a fixed input size in a neural network, passing the raw data into the neural network was impossible. The preferred approach to solving this was to implement a voxelization process. This was done by defining a room with a 3D grid of voxels. The dimensions of the room was selected to roughly correspond to the boundaries used for point filtration. Experiments with different voxel sizes were conducted and an appropriate voxel size appeared to be $(10, 10, 10)$ cm. All voxels are initialized as 0. For each point in a cloud that falls inside a voxel, the value of that voxel is incremented by 1. While this did solve the dimensionality problem, it introduced a new problem. With a voxel size of $(10, 10, 10)$ cm, and a room corresponding to the boundaries used for point filtration, the total amount of voxels is $30 \cdot 12 \cdot 24 = 8640$. A swift study of point counts for filtered frames at the time resulted in a rough estimate of 300 points per frame, albeit with a large variance. This resulted in sparse data,

meaning that a lot of the voxels were empty.

Network structure	t/epoch	Test Loss	Test Acc.	F1 (NP)	F1 (P)
3D CNN w/ XYZ	13 sec	0.588	0.684	0.703	0.662

Table 6: Table showing performance on different measures of the 3D CNN. From left to right, training time, test loss, test accuracy, F1-score on (NP) and F1-score on (P).

The 3D CNN network structure was the first point of reference for evaluating the performance of a network in terms of training times and accuracy. The results from testing it are presented in Table 6. The results were a success in terms of being able to predict with higher accuracy than a guess, but the authors believed there was much room for improvement. A decision was made to experiment with other network structures before spending more time on tuning the architecture and configuration of the 3D CNN network.

3.3 Model 2: PointNet

In the previous work conducted by Anton Almqvist and Anton Kuusela [1], summarized in Section 1.3.2, the effectiveness of a network structure based on PointNet for a similar application was shown. As they were working with point clouds generated by the same radar, the likelihood of PointNet being effective for piggybacking detection appeared high. The structure of the implementation of a PointNet based network can be seen in Figure 24.

When experimenting with PointNet, the idea of using voxels was abandoned. Instead all points that remain in a frame after filtering are inserted into a matrix of size (Max number of points, Number of features). The first configuration used three features: the X, Y, and Z coordinates. The max number of points is configurable. It was set to 512, which was slightly higher than the perceived median amount of points in a frame after filtering. The points are sorted by RCS, in descending order. A frame is then handled according to (34) depending on how many points it contains after filtering. Padding means that the empty rows of the matrix, more precisely the rows from the final row with points, to the very final row is filled with zeros. Truncating means that the matrix is filled with the 512 points that have the highest RCS value, while any remaining points remain unused. This new structuring of the training data resulted in the dimensions being much smaller than when using voxelization. Because of this, the number of trainable parameters was also reduced.

$$\begin{cases} \text{Frame considered empty,} & \text{if Number of points} < 125 \\ \text{Frame padded to 512 points,} & \text{if } 125 \leq \text{Number of points} < 512 \\ \text{Frame truncated at 512 points,} & \text{if } 512 \leq \text{Number of points} \end{cases} \tag{34}$$

A possibility that became apparent when moving away from the 3D convolutional layer was that more than three features could be used. It was determined that the measurements of RCS and radial velocity for each point could be of interest. The idea behind including velocity was that it might help to separate people whose limbs are moving out of sync, and at different speeds.

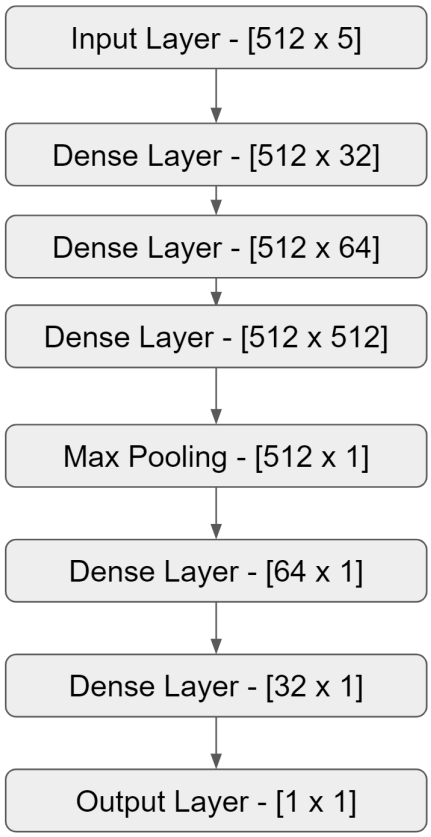

Figure 24: Structure of the PointNet based network, including dimensions in brackets. The main differences from the PointNet structure in Figure 15 is the exclusion of the initial part of the network pertaining to T-Net, as well as the segmentation network. Furthermore, the size of all layers has been halved.

The results using the PointNet based architecture are presented in Table 7. The very left column of Table 7 describes the different network configurations tested. For these tests, all network structures are identical aside from the features used. The feature combinations used were, from top to bottom, {X, Y, Z}, {X, Y, Z, RCS}, {X, Y, Z, RCS, Velocity}, and {X, Y, Z, RCS, Velocity} with velocity augmentation applied to the training datasets. Significant improvements in accuracy was observed compared to the 3D CNN. Due to the promise this architecture showed, any attempts at improving the 3D CNN were abandoned after testing the PointNet-based architecture. But at 90% accuracy, it was determined that there was still room for improvement.

Network structure	t/epoch	Test Loss	Test Acc.	F1 ((NP))	F1 ((P))
PointNet w/ XYZ	25	0.227	0.900	0.900	0.899
PointNet w/ RCS	25	0.225	0.896	0.896	0.897
PointNet w/ VEL	26	0.193	0.917	0.915	0.919
PointNet w/ AUG	26	0.218	0.903	0.898	0.907

Table 7: Comparison between the PointNet based network structure with different features.

T-Net, the left part of the PointNet structure seen in Figure 15, was implemented and put through brief trial runs. The results gathered from this was that it didn't improve the performance of the network, paired with a significant increase in training time. The purpose of T-Net is to make the predictions invariant to certain rigid transformations of the input point clouds, such as mirroring and rotations. Therefore it was determined to be unnecessary for the testing purposes within this thesis, with its very limited spatial constraints for input data. Instead, the faster performance of a network structure without T-Net was prioritized.

3.4 Model 3: LSTM and PointNet

In an attempt to preserve information contained in sequences of data, temporal layers were introduced. An LSTM network was deemed most appropriate for this problem. The performance of Model 2 (presented in Section 3.3), which was based on PointNet was promising, therefore an attempt of combining PointNet with an LSTM network was made. The LSTM network was designed in a way that the input was structured into short sequences of a fixed length. This length will be referred to as *memory size* in the future. Picking a suitable memory size ended up being an important parameter for the performance of the network.

A higher memory size usually resulted in better performance when testing the network on the test sets. However, it was theorized that a higher memory size would reduce responsiveness when shifting to a sequence of a different class. The training and testing data were structured in a way that the network was first fed all (NP) data, then all (P) data, meaning there was only one shift between the two classes in the entire run. This made any potential issue negligible for such tests. However, for more realistic applications such as in Section 3.6 below, where a real-time classification software is implemented, these effects become noticeable. Different memory sizes were tried before finding that the optimal value was somewhere between 5 and 20.

The input dimensions of this network is (Memory size, Max number of points, Number of features). The Keras wrapper TimeDistributed was used to pass this input into the PointNet based network. This wrapper feeds the frames in each sequence, one by one into the PointNet based network, and inserts the outputs into a vector of size (Memory size, 1). In this network, the PointNet based network can be seen as a

sub-network of a larger network. The output vector of the TimeDistributed PointNet sub-network is then fed into an LSTM layer. Finally the outputs of the LSTM layer is fed into two TimeDistributed Dense-layers. The output from these layer will be a vector of size (Memory size, 1), and will contain a prediction for each frame in the input sequence. The full structure of the network can be seen in Figure 25.

A variant of this network where the LSTM layer was replaced by a Bidirectional LSTM layer was also tested. As can be seen in Table 8 this showed better performance, at the cost of a slight increase in network complexity. This was deemed to be a worthwhile trade-off. The left column of Table 8 contains the differences in network structure and features. From top to bottom, Regular LSTM with {X, Y, Z, RCS, Velocity} with velocity augmentation, Bidirectional LSTM with {X, Y, Z}, Bidirectional LSTM with {X, Y, Z, RCS, Velocity} and Bidirectional LSTM with {X, Y, Z, RCS, Velocity} with velocity augmentation. With an accuracy of 99% it was deemed that there was little room for improvement, and the performance was satisfactory in terms of achieving (AP1). Therefore, the Bidirectional LSTM network combined with the PointNet based network will be the focus of the rest of the thesis.

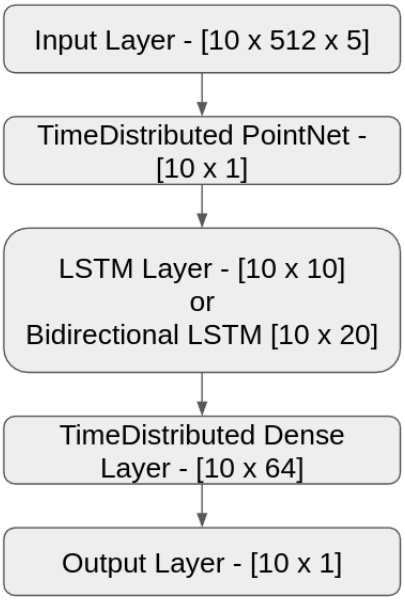

Figure 25: Structure of the LSTM and PointNet based network, including dimensions in brackets.

3.5 Implementation of subcategories

After reaching a level of performance of 99% accuracy when classifying (NP) and (P), a decision was made to branch out into other scenarios rather than trying to

Network structure	t/epoch	Test Loss	Test Acc.	F1 ((NP))	F1 ((P))
LSTM w/ AUG	27 sec	0.148	0.953	0.950	0.956
Bi-D LSTM w/ XYZ	26 sec	0.043	0.990	0.989	0.990
Bi-D LSTM w/ VEL	27 sec	0.051	0.987	0.991	0.983
Bi-D LSTM w/ AUG	27 sec	0.053	0.987	0.986	0.988

Table 8: Comparison between the LSTM and PointNet based network structure with different features and different LSTM layers.

increase the accuracy further. At this point in time, the subcategories presented in Section 3.1.2 were implemented.

As previously mentioned, the categories were introduced and implemented one at a time. The first category considered was (P_h). Recording a small testing dataset and running it through the latest version of the Bidirectional LSTM model showed poor results. The model mostly classified these new frames as containing only one person. The working hypothesis regarding this behavior was that the two people in (P_h) were represented in the point clouds as one large blob of points. This pattern was more similar to the single blob seen in (NP) than the two, separate blobs in (P). Introducing (P_h) training data to the network resulted in significant improvement. A reproduction of this was constructed by performing a brief training of two networks for the roof-position, as seen in Table 9.

Training-categories	(NP) Acc.	(P) Acc.	(P_h) Acc.
(NP), (P)	98.3%	98.4%	18.2%
(NP), (P), (P_h)	93.8% (-4.5%)	97.1% (-1.3%)	86.4% (+68.2%)

Table 9: The accuracy of the network on the three test sets related to (NP), (P), and (P_h) when adding more categories to training data.

The results gathered from this reproduction is similar to what was observed during development. The good news was that the model showed adaptability, with great improvement in accuracy for (P_h). The bad news was that it seemed to have a negative effect on the other two categories. The similarity between (P_h) and (NP) resulted in reduced interclass variation, while the dissimilarity between (P_h) and (P) resulted in increased intraclass variation. Both factors made the classification more difficult, as the categories were no longer as well separated. This pattern was unfortunately not unique to (P_h), and was the main challenge related to introducing new categories. The model was able to learn to classify new scenarios after being presented with just a couple of minutes of training data from the relevant category, but doing so would often reduce the accuracy for (NP) and (P), which were considered the most important categories by the authors.

3.5.1 Introducing Multiple Networks

A system that makes predictions can include more than one *network*. To combat the issue of the reduced accuracy when implementing more categories, an attempt was made to use a model containing multiple networks. Initially, a model was built using two networks: one for separating (NP) and (P), the other for separating (NP) and (P_h). Each was trained only using the data from the relevant categories. This proved very effective, and the concept was further researched by implementing more categories.

Since the network trained for separating (NP) and (P_h) is specialized towards identifying frames that were incorrectly classified as 'Not piggybacking' by the first network, it will be referred to as the False Not Piggybacking-, or *FNP network*. The training data for (P_d) was also added to the FNP training data (for the reader-radar). This resulted in a that network was very accurate at separating the included categories: (NP), (P_h), and (P_d). The same was true for its counterpart, the False Piggy-, or *FP network*. This network was very accurate when separating (P), (NP_s), and (NP_c). The network trained on (NP) and (P) will be referred to as the *P/NP network*. The networks were, of course, not very accurate on categories not included in the training data.

The problem was that these networks would only be presented with the intended subcategories in the case where the P/NP network classified incorrectly. Basically, the model relied on the P/NP network being correct for the categories (NP) and (P), while being incorrect on the subcategories. This is problematic, as the P/NP network will sometimes get the subcategories right. As seen in Table 9, a network trained on (P) and (NP) can be correct 18.2% of the time for (P_h). Three potential solutions were identified by the authors:

- Solution 1: Increase the probability of the P/NP network being wrong on the subcategories by including them in the training set, but with inverted labels. For example, the category (P_h) would be included in the training data for the P/NP network, but it would be "incorrectly" labeled as 'Not piggybacking'.
- Solution 2: Include the training data from every subcategory in both the FNP and FP networks. This would increase the chance of the model being right, even if the P/NP network wasn't incorrect as intended.
- Solution 3: Return to a model using only a single network, trained on every category.

The first solution was discarded due to it being perceived as unconventional and convoluted. The second and third solutions were considered. An automated test was performed to evaluate the solutions, the results of which can be found in Table 10. The results for FP and FNP are acceptable across the board. Note that results for (NP) from the FP network, as well as results for (P) from the FNP network can be

disregarded. This is because the P/NP network is expected to be correct on those categories 99% of the time. The network trained on all categories performs very poorly for (NP), but otherwise well. Based on these results, Solution 2 was selected.

Network	FP		FNP		All categories	
Excl. data	(NP)		(P)		None	
Category	Test Acc.	Test Loss	Test Acc.	Test Loss	Test Acc.	Test Loss
(NP)	0.643	1.428	0.994	0.038	0.362	0.829
(P)	0.984	0.048	0.684	1.249	0.965	0.186
(P_d)	0.94	0.289	0.779	0.932	0.929	0.169
(P_h)	1	0.015	0.978	0.096	0.992	0.136
(NP_s)	1	0.015	1	0.015	0.986	0.105
(NP_c)	1	0.018	1	0.016	1	0.185

Table 10: Table showcasing the loss of performance when introducing data from different categories. The columns are three different networks trained on [(P), (P_h), (NP_s), (NP_c)], [(NP), (P_h), (NP_s), (NP_c)] and [(NP), (P), (P_h), (NP_s), (NP_c)]. Note that (P_d) training data is not included due to this test being performed on the roof-position.

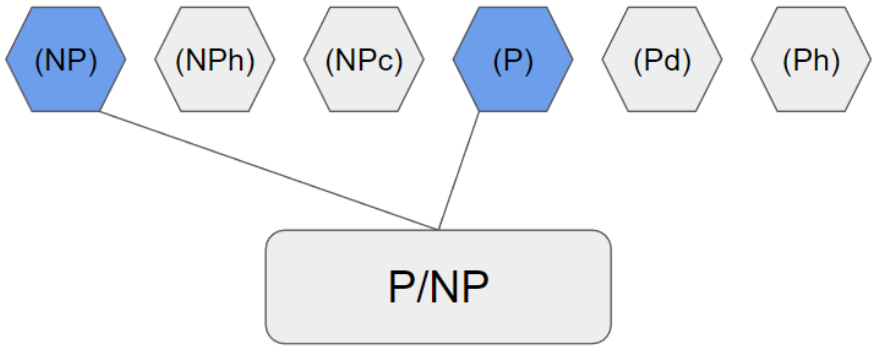

Figure 26: These are the training-sets included in the P/NP network.

As such, the final model ended up consisting of a total of three networks. The main network, P/NP, is still focused on separating (NP) and (P), trained only on the datasets from those two categories, which can be seen in Figure 26. The other two networks are trained on each of the included subcategories, sans one of the main categories. The FNP network is trained on all training datasets except that of (P), which can be seen in Figure 27. Conversely, the FP network is trained on every dataset except that of (NP), which can be seen in Figure 28.

The model considers the networks according to the flowchart presented in Figure 29. If the P/NP network indicates 'Piggybacking', i.e. $y > 0.5$, the output of the False P network is considered when making a classification. Conversely, if the P/NP network

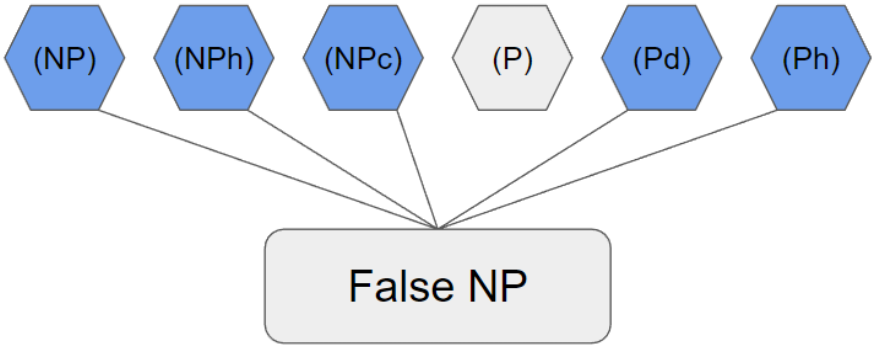

Figure 27: These are the training-sets included in the FNP network.

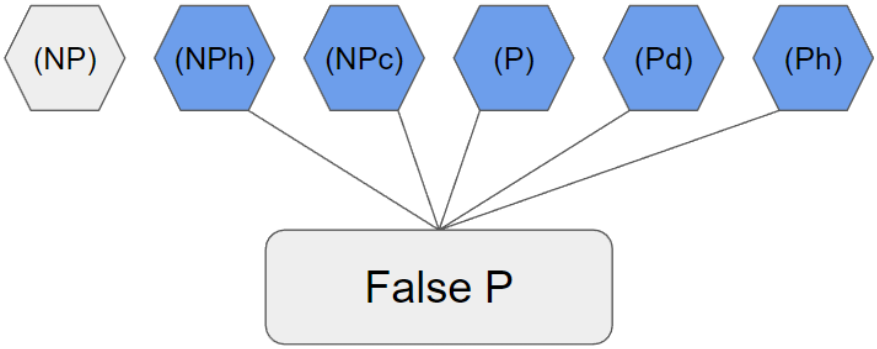

Figure 28: These are the training-sets included in the FP network.

indicates 'Not piggybacking', i.e. $y < 0.5$, the False NP network will be considered. The thresholds for the specialized network are set differently than the P/NP threshold of 0.5. The reasoning was that, since the P/NP network already made a decision, the specialized network should have to be extra certain to overturn it. This setup of three networks working together will be referred to as the *triple-network model*.

3.6 Implementing a real-time classification model

To enable comparison between the models built in this thesis and the existing camera solution it was necessary to perform real-time tests. The camera solution software was available only as a finished product, operating in real-time on the feed of an Axis-camera. The determined approach for the test design was to perform tests on a full sequence of one or two people entering the testing zone, and passing through it completely.

Since all training data has been recorded in long sequences of walking back and forth within the boundaries, multiple issues had to be addressed:

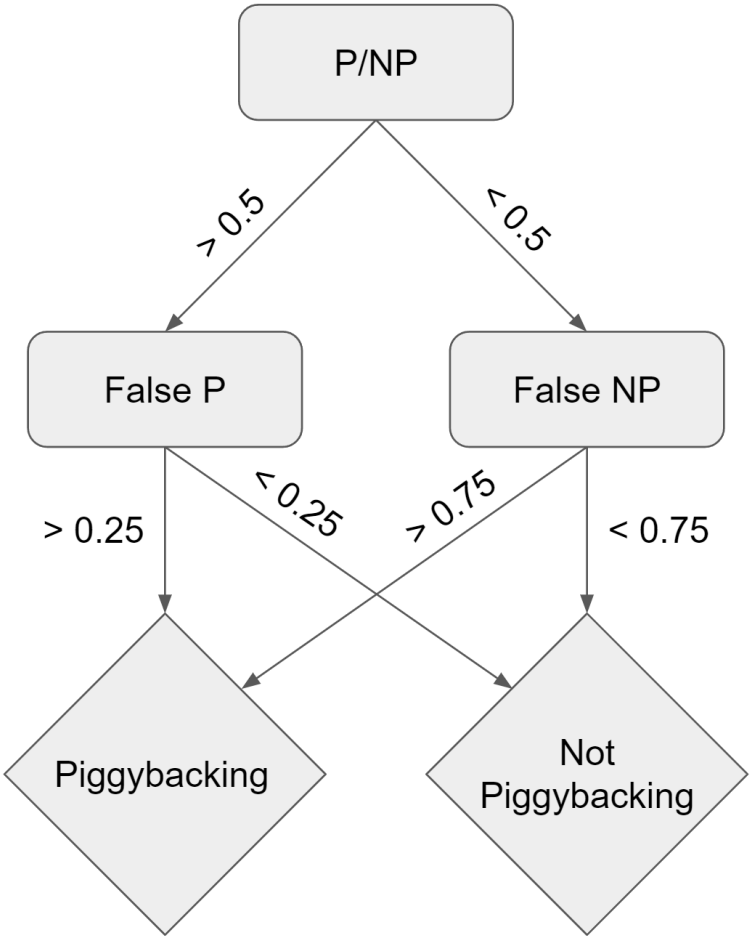

Figure 29: Flowchart depicting the decision making process of the triple-network model.

- In a realistic scenario, a majority of frames will contain no people.
- The beginning and end of a sequence must be determined.
- In a sequence of two people passing, there will be frames near the beginning and end of the sequence where only one of the two remain within the boundaries.

The initial approach was to diverge from binary classification and instead experiment with having three output nodes i.e. $d = 3$. These output nodes would be denoted O_0, O_1, O_2, where the index refers to the amount of people predicted in the frame. While experimentation on this architecture was not extensive, the results were quite clear. From previously having around 99% test accuracy on (NP) and (P), the model now averaged around 90%. While this was not deemed to be an insurmountable issue, another method showed more promise. The final solution ended up being an extension of the previously used data processing feature which filtered the frames with too few

points. Specifically, if a frame does not meet the requirements for the minimum amount points, it will be thrown away. If ten frames are tossed in a row it will be considered the end of the sequence, at which point the LSTM-memory is cleared to avoid any residual effects on the next scenario. A new sequence will then begin as soon as a frame with enough points is recorded.

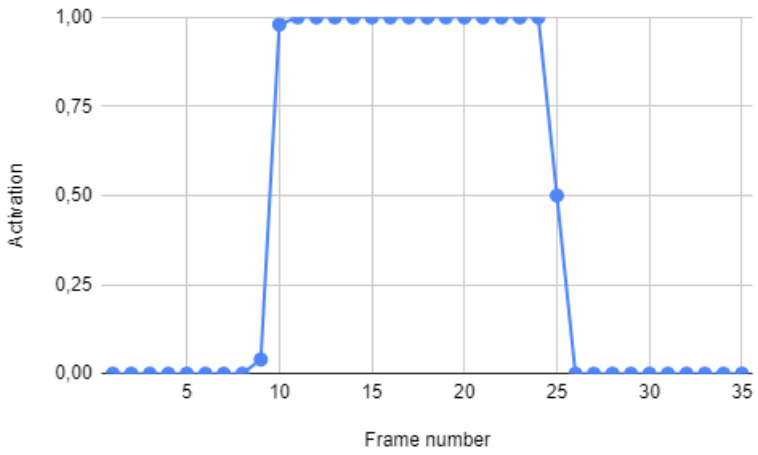

Figure 30: The output of one sequence of 'Piggybacking' from the real-time classification system, with frame number on the x-axis and the value of the output neuron on the y-axis.

The third issue was that a sequence of piggybacking will, even with a perfect network, contain several frames classified as not piggybacking. This can phenomenon be seen in Figure 30. The predictions at, or close to, zero occur in the beginning and end of a sequence. This is because the person in front will appear within the boundaries first. They will remain alone for a brief instance of time, before the second person also enters the boundaries. This will cause the activation to shoot up to one, as seen in the figure around frame 10. Conversely, the second person will remain alone within the boundaries for a similar amount of time after the first person exits the boundaries. This happens in the figure around frame 25.

If the selected approach for making a classification for an entire sequence is to calculate the mean of the activations, the final output is given by $y_{final} : \mathbb{R}^m \times \mathbb{R}^M \to \mathbb{R}$, according to

$$y_{final}(\boldsymbol{x}_i, \boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} y(\boldsymbol{x}_i, \boldsymbol{\theta}). \tag{35}$$

Here, the function $y : \mathbb{R}^m \times \mathbb{R}^M \to \mathbb{R}$ represents the output of a network for one input vector, similar to (10) with $d = 1$, while n is the number of frames within the sequence.

If (35) is calculated for the sequence pictured in Figure 30 the mean activation would be 0.44, below the typical cutoff at 0.50. While the threshold for what is to be classified as 'Piggybacking' could be tweaked to a number which would result in correct classifications most of the time, it was conjectured that a more robust approach would be to calculate the mean activation for each set of n successive predictions, and selecting the set with the highest value. The kernel size $n = 5$ was determined based on the amount of ones observed in a typical activation sequence for the roof radar, since its sequences are typically shorter. Using this method, the final prediction is calculated as

$$y_{final}(\boldsymbol{x}_i, \boldsymbol{\theta}) = max(\sum_{i=1}^{5} \frac{y(\boldsymbol{x}_i, \boldsymbol{\theta})}{5}, \sum_{i=2}^{6} \frac{y(\boldsymbol{x}_i, \boldsymbol{\theta})}{5}, ..., \sum_{i=n-5}^{n} \frac{y(\boldsymbol{x}_i, \boldsymbol{\theta})}{5}). \tag{36}$$

When applying this method to the sequence pictured in Figure 27, $y_{final}(\boldsymbol{x}_i, \boldsymbol{\theta}) = 1$.

The methods described in this section was used when building a real-time classification model which contained the triple-network model. When running this model, the three networks in the triple-network model perform a prediction on each sequence in parallel. The outputs of each network is calculated according to (36). These outputs are then used to make a classification, according to the flowchart in Figure 29. The real-time classification model involves multiple components, all of which can be viewed in Figure 31.

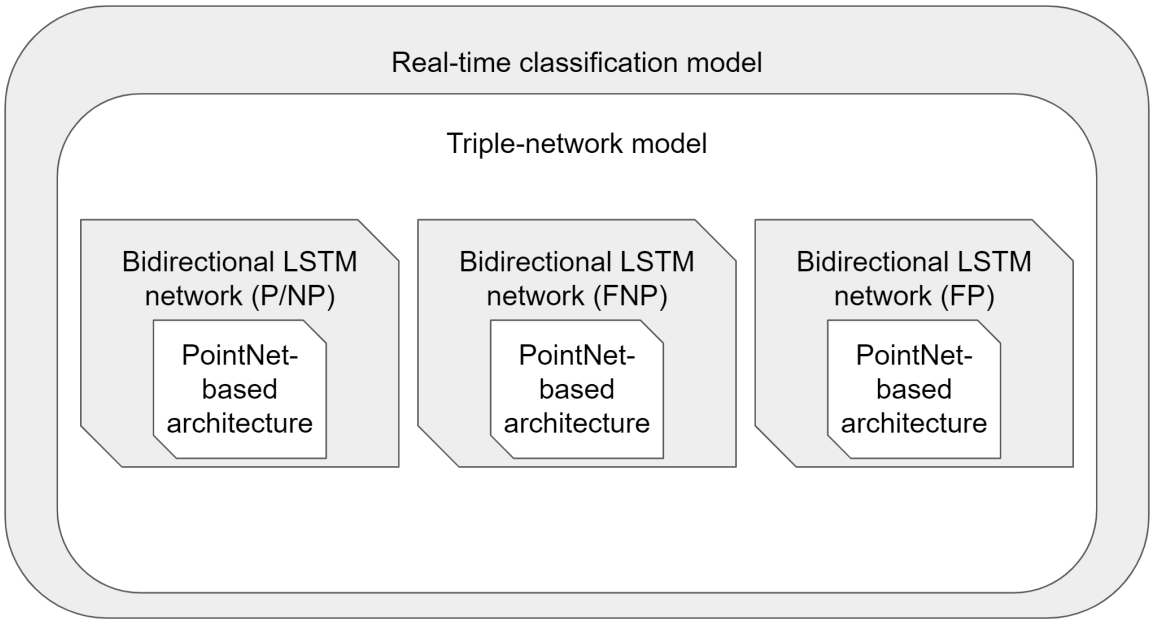

Figure 31: Depiction of the real-time classification model and its components.

3.7 Manual Test Design and Execution

To enable parallel testing for the radars and the camera solution, manual tests had to be implemented. These were done using the real-time classification model, and as such involved full sequences of with or without piggybacking. A flowchart of the testing process is presented in Figure 32.

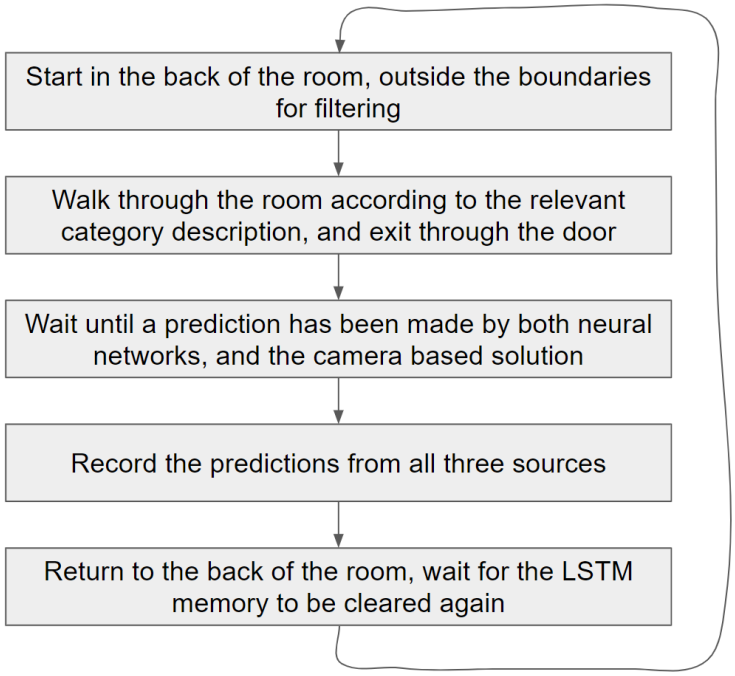

Figure 32: Flowchart depicting the process when doing manual tests.

Due to the time consuming and tedious nature of this testing process, the amount of samples recorded had to be somewhat limited. The amount of samples for each category is presented in Table 11.

Category	(NP)	(NP_s)	(NP_c)	(NP_b)	(NP_f)	(P)	(P_h)	(P_d)	(P_f)
Samples	400	100	100	100	100	400	100	100	100

Table 11: Sample sizes for each category of manual testing.

The models used for manual testing contained the triple-network Bidirectional LSTM models, for both radar positions. This selection was made based on the results of the automated tests, as such no significant preliminary testing was performed using other designs within the real-time system. Some preliminary testing was performed to determine the amount of features to be included. As seen in Table 8, the performance for (NP) and (P) was similar regardless of the number of features. There was, however, a noticeable difference in performance for (P_f) and (NP_f), which ended up

being the decisive factor. This resulted in Bidirectional LSTM with all five features and velocity augmentation being selected for the manual testing. The parameters used when training the final networks are found in Table 12.

	Reader	Roof
Epochs	500	500
Memory size	5	10
No. Features	5	5
Base learn rate	0.001	0.001
Learning decay	0.1	0.1
Batch size	32	16

Table 12: Hyperparameter configuration for the networks used in manual testing.

The memory size was selected based on performance in automated tests. The base learning rate was set to $\eta_0 = 0.001$, decaying as the amount of epochs t increases according to

$$\eta_t = \eta_0 \cdot e^{-\lambda t}. \tag{37}$$

The decay rate λ was set to 0.1. Both learningrate hyperparameters are selections which showed good results in preliminary testing. Finally, the batch sizes were set given constraints in GPU memory, 32 and 16 respectively. The difference is due to the memory sizes, a larger memory for the LSTM requires more GPU memory. The training data used are the sets described in Table 4.

4 Results

4.1 Manual tests

Table 13 shows the accuracy for every category for the reader placement, the roof placement and the camera solution. It also shows how many manual tests have been made for each category.

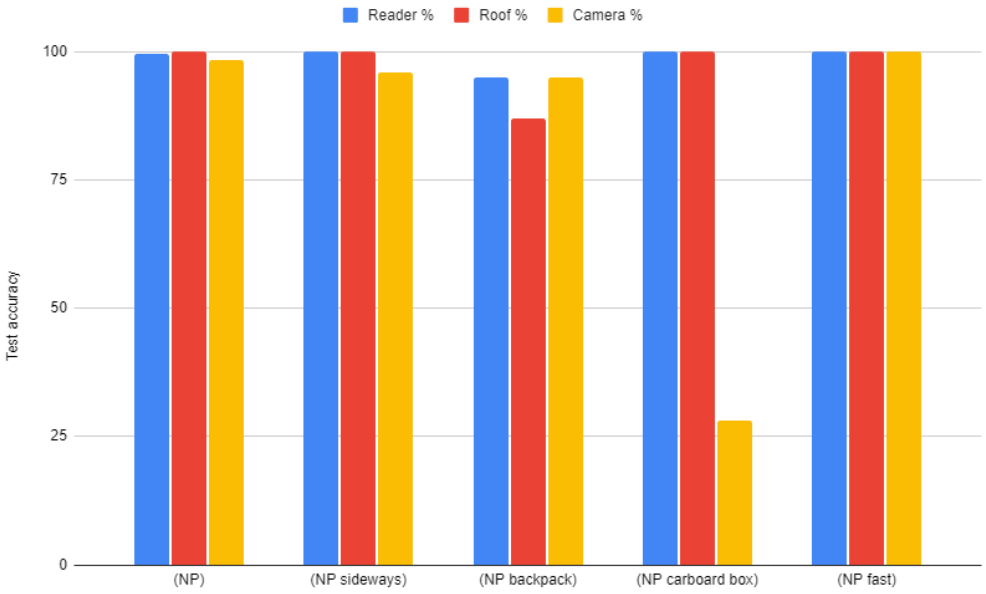

Figure 33: Bar-chart showcasing the results on (NP) and (NP)s subcategories.

On top of the 1500 tests presented in Table 13 as well as in Figure 33 and Figure 34, 120 extra tests for category (P) were done. In these different people than the authors appeared. All three solutions managed an accuracy of 100% across the 120 tests.

As shown by the results in Table 13, Figure 33 and Figure 34, radar is indeed a viable solution for detecting piggybacking. When looking at the two most common, and most important categories (NP) and (P) the radars perform better or on the same level as the camera.

For (NP) the radars are slightly better with 99.5% accuracy for the reader placement and 100% accuracy for the roof placement. This is in comparison with the 98.25% accuracy of the camera solution. The performance of the camera solution is influenced by the sensitivity setting described in Section 2.3. The default value for sensitivity was used since this appeared to slightly more often see one person more than it should, compared to how often it saw one less person than it should. This was considered a good trade off since the authors think that missing an instance of piggybacking

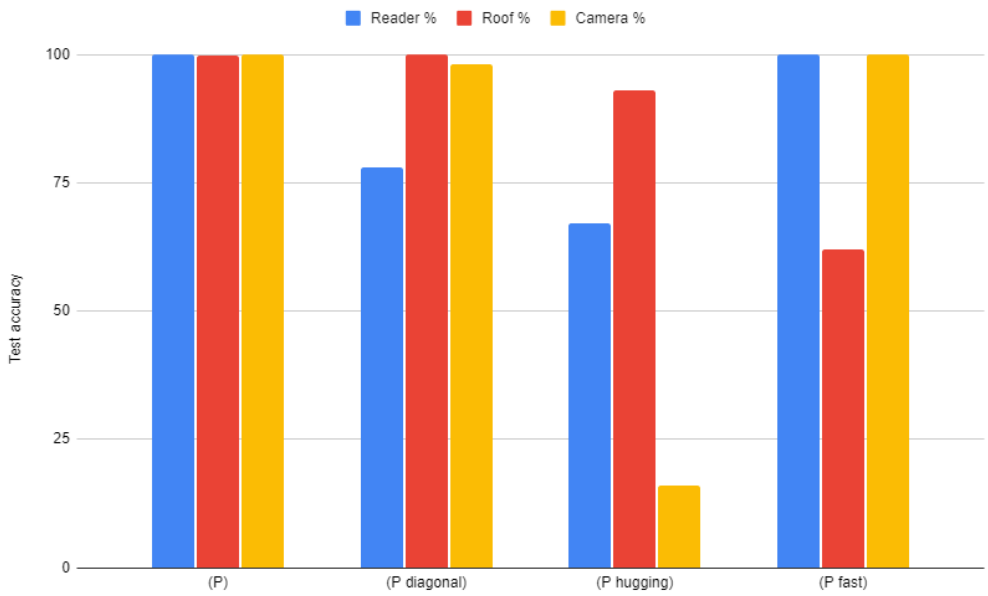

Figure 34: Bar-chart showcasing the results on (P) and (P)s subcategories.

is worse than flagging for it when it is not happening. The results of using the default sensitivity setting is that the camera solution should generally be better at the 'Piggybacking' categories than at the 'Not piggybacking' categories.

On (P) all three solutions perform very well. Both the reader placement and the camera solution predicted correctly in 400/400 attempts leading to an accuracy of 100%. The roof placement was wrong once in 400 attempts and had an accuracy of 99.75%. An interesting note is that the camera solution often saw three or even four people exiting the room. This is likely due to the sensitivity setting used, described in Section 2.3. In 18% of the attempts, the camera solution saw more than two people walking. This have however been disregarded for the sake of our testing, and counted as correct predictions.

The hypothesis for (P_d) was that the reader placement would struggle, but the roof placement and the camera solution would perform similarly to their performance on (P). This category was designed to be challenging for the reader placement as from the readers point of view, the goal is to completely cover the person walking behind. This was the hardest data to generate since the exact movement was hard to reproduce. Some test scenarios are likely to have been easier for the reader if for example, the shoulder of the person walking behind was showing for a period of time. The hypothesis ended up being accurate as the reader placement showed significantly worse performance than the roof placement and the camera solution. The reader placement had an accuracy of 78%, the roof placement had an accuracy of 100% and the camera

Category	Reader %	Roof %	Camera %	Samples
(NP)	99.5	100	98.25	400
(NP_s)	100	100	96	100
(NP_c)	100	100	28	100
(NP_b)	95	87	95	100
(NP_f)	100	100	100	100
(P)	100	99.75	100	400
(P_h)	67	93	16	100
(P_d)	78	100	98	100
(P_f)	100	62	100	100
avg %	93.28	93.53	81.25	

Table 13: Table comparing the results of the three-network models and the camera solution on all categories.

solution had an accuracy of 98%. An interesting note regarding the reader placements performance is that it had an 86% accuracy when one of the authors walked in front and a 70% accuracy when the other author walked in front. This was not expected since the reader placement performed better with the taller author walking in front. A possible cause is the aforementioned variety in test execution.

(P_h) was expected to be the toughest category across the board, which ended up being true. In this category the people walking were hugging while walking. This meant that their upper bodies were touching, leaving minimal separation between the persons. Out of 100 samples the reader placement predicted 67% correctly, the roof placement predicted 93% correctly and the camera solution only predicted 16% correctly. The difference in performance between the reader placement and the roof placement is most likely down to the roof placement seeing more separation from its point of view. A big factor in this category might be the persons' heads, since they are still well separated. Perhaps the roof placement is better at picking up this separation. The drop-off in performance of the camera solution in comparison to the radars is significant. More often than not, the camera only saw one person exiting.

(NP_s) is not a common scenario in a realistic access control perspective, the category was introduced due to the challenge it proved for earlier iterations of the network. After training the specialized networks on (NP_s) data they both performed well however. Both radar placements ended up having an accuracy of 100% and the camera solution had an accuracy of 96%.

The networks showed great performance for (NP_b) in early stages of development, therefore no (NP_b) data was included in the training sets. This performance dropped significantly after adding all classes except $(NP)/(P)$ to the False P and False NP

models. A decision was made to continue without adding new training data, and use this category as a test of robustness. The reader placement achieved an accuracy of 95%, the roof placement achieved an accuracy of 87% and the camera solution achieved an accuracy of 95%. Both authors performed 50 tests each. These tests were performed on different days. Because of this, the content of the backpack was different, which resulted in slight differences in size of the backpack. This was likely reflected in the point clouds, which could be expected to be slightly larger when the backpack was fuller. This difference is believed to have made a large impact on the results. On the 50 attempts with a slightly fuller backpack the accuracy was 92%, 74% and 90% for the reader placement, the roof placement and the camera solution. On the 50 attempts with a less full backpack the accuracy was 98%, 100% and 100% for the different solutions. A takeaway from this category is that just because something worked well in a model trained with fewer categories, it does not mean that it works as well in a model trained on more categories. If data from a (NP_b) training set was included, the results would most likely be better in this category. Regardless, the accuracy for this category is a positive indication for the robustness of the model.

Both radar placements had an accuracy of 100% on (NP_c), while the camera solution only had an accuracy of 28%. The camera solution even detected three persons on a few occasions. The camera solution would achieve better results, if a lower sensitivity setting was used. This would however impact all other categories, and most likely lead to worse results for the camera solution overall.

(NP_f) is basically the same as (NP), with the difference that the person walks at about double the speed. All three solutions managed an accuracy of 100% across 100 samples on this category.

The results on (P_f), which is similar to (P) but with double speed, gave useful insights about memory size for the roof placement. Both the reader placement and the camera solution achieved an accuracy of 100% while the roof placement only achieved 62%. This is most likely due to a combination of the limited field of view of the roof placement, and the larger memory size of 10 frames. When walking faster, the part of a sequence when both persons are within the roof radar's boundaries is not long enough to reliably fill the buffer for the LSTM layer with frames containing both persons. This leads to a way lower accuracy than the two other solutions, in what should be a relatively easy category. Another way to impact this category could be a decrease of the kernel size for the real-time classification model.

5 Discussion

5.1 Comparison between network-structures

Network structure	Aug.	Test Loss	Test Acc.	F1 ((NP))	F1 ((P))
3D CNN w/ XYZ		0.588	0.684	0.703	0.662
PointNet w/ XYZ		0.227	0.900	0.900	0.899
PointNet w/ RCS		0.225	0.896	0.896	0.897
PointNet w/ VEL		0.193	0.917	0.915	0.919
PointNet w/ VEL	X	0.218	0.903	0.898	0.907
LSTM w/ VEL	X	0.148	0.953	0.950	0.956
Bidirectional w/ XYZ		0.043	0.990	0.989	0.990
Bidirectional w/ VEL		0.051	0.987	0.991	0.983
Bidirectional w/ VEL	X	0.053	0.987	0.986	0.988

Table 14: Compiled table of the results from evaluating network structures in Chapter 3. Table includes the important network structures with different combinations of features. Whatever follows w/ are the features used for that specific network. XYZ means that only X, Y, Z was used, RCS means that X, Y, Z, RCS was used and VEL means that X, Y, Z, RCS, Velocity was used.

The results in Table 14 are from the automated tests comparing the important network structures with different combinations of features. All of these networks are of the P/NP kind, therefore all of these tests were done on only (NP) and (P). A trend can be seen, that velocity augmentation generally lowers the performance slightly on these categories. The performance drop is very small however. Because the drop in performance on (NP) and (P) is so small, the final networks ended up using velocity augmentation anyway. The reason being that it increased performance significantly on (NP_f) and (P_f) during preliminary real-time tests, that are not presented. The performance increase on (NP_f) and (P_f) can only be observed in the manual tests, since generating automated test-data for those categories was deemed very difficult to physically perform.

It is clear that the structure of the network makes more difference than the inclusion of RCS and velocity. A large jump in performance can be seen when moving from a 3D CNN to a PointNet based architecture. Yet another large jump can be seen when LSTM is introduced. A final jump in performance can be seen when moving from a standard LSTM layer to a bidirectional LSTM layer. This is interesting since the model is operating in the time domain, and looking at sequences from future to past doesn't make a lot of sense. The performance increase with the bidirectional variant is however quite significant when compared to the standard LSTM.

5.2 Limitations

Differences from realistic access-control scenarios Due to practical reasons, neither the recorded training and test data, nor the real-time testing scenarios involve interacting with a card reader and/or number pad which are common occurrences in an access control environment. Doing so would significantly alter the sequences. While classifying such scenarios would likely be possible with the current hardware based in the findings in this thesis, it would involve more complex recording of training data.

Additionally, it would likely have a greater impact on the reader-position than the roof-position. If the radar is positioned near the card reader, its view can be significantly obscured depending on how the person interacts with the card reader. If the person wears their card as a badge around the neck, they might stand completely in front of the reader to scan their card. This could potentially block the radar's view entirely, thus making the roof position strongly preferable to ensure continuous monitoring.

Comparison with the camera solution It is important to keep in mind that the results in this thesis should not be considered a comparison between radar and camera in terms of hardware capabilities. Such a comparison would be very unfair, as the radar models have been specifically trained to handle most of the scenarios. The purpose of (RQ2) was to make a comparison with this specific combination of camera hardware and software, and the results should be viewed as such.

5.3 Computation time

While computation time was not considered in attempts to answer the research questions, the authors would like to briefly touch on the subject with (AP2) in mind. Computational time can be an important metric for practical applications. Generally every increase in computational speed is positive, but in this thesis the accuracy was the focus. The reason is that the computation time never became a bottleneck during the project. With a triple-network model run within the real-time model, a prediction is made every time a new point cloud is generated by the radar. This happens every 100 ms. The hardware used to run the real time classification system was a desktop computer with an Intel i7 9700K CPU. On this computer, each network of the triple-network model needed 12 ms to make a prediction. This resulted in each classification taking on average $12 \cdot 3 = 36$ ms. Even though 36 ms is within a good margin from the 100 ms between readings from the radar, this raised some warning flags regarding the time it takes to make a prediction. In a realistic access-control application, one might want to run this software on much slower hardware. The prediction time could then become a problem.

Two methods of lowering the prediction time were very briefly experimented with.

The first method was going back to a model using a single network, as seen in Table 10. In this network, data from all categories ((NP), (P), (P_d), (P_h), (NP_s), (NP_c)) was included. This resulted in the accuracy dropping significantly on the main categories, and another structure was selected for continued experiments. Even so, the authors believe that such a solution could show promise if more time was spent on recording training data and tuning the network. The second method was to lower the complexity of the networks. More precisely, the number of neurons in the layers of the PointNet-based part of the networks was halved. No notable drop in performance was observed in preliminary testing of such networks. It did however impact the training time and the time to predict significantly. The training time was cut from 27 s/epoch to 13 s/epoch, and the time to predict was cut from 12 ms to 6 ms for each network.

5.4 Decision thresholds

When moving from a model containing one network to several networks, the overrulement thresholds < 0.25 for the False Piggy network and > 0.75 for the False Not Piggy network were used. These thresholds were chosen with the motivation that for a more specialized network to overturn the decision of the P/NP network, it has to be very certain. Not a lot of time was spent on trying different thresholds. When performing the manual tests, the FP and FNP networks very rarely overturned a correct decision made by the P/NP network. The manual tests did however show that the activation of the specialized networks often came close to their thresholds, without quite reaching them when they were supposed to overturn the decision of the P/NP network. For example False Piggy would quite often have activations around 0.3 when it needed < 0.25 to overturn the decision of P/NP. The fact that the specialized networks are less certain than the P/NP network can be due to several reasons. One potential reason for this could be that the training data for these networks the categories is more difficult to separate. Another reason could be the fact that the datasets for the subcategories used for training these networks are smaller than the training sets for (P) and (NP).

5.5 Future Work

5.5.1 Experimentation with decision thresholds

As seen in Section 5.4, there was likely some room for improvement with the decision thresholds. In retrospect, the authors feel that more research regarding the optimal thresholds should have been done. With more generous thresholds of 0.5 for both specialized networks, or < 0.4 for the False Piggy network and > 0.6 for the False Not Piggy network, the results would certainly be different. More often than with < 0.25 and > 0.75, the specialized networks would overturn correct decisions, but they would be better at overturning incorrect decisions. A change in the thresholds for the specialized models could potentially yield better results in terms of accuracy of

the model. More experiments would however be needed to confirm this theory.

5.5.2 Different environments and movement patterns

In this thesis all generation of data was done in the same room. When briefly testing the model in another room it performed worse but still acceptably when using the same boundaries as was defined in Section 3.1.4, when walking in a similar fashion as in the training data. The model did however perform significantly worse when the boundaries were altered to allow moving in different patterns, for example from the very left of the room, to the very right. One approach that was considered but never implemented in this project was using *clustering*. The hypothesis is that if clustering is used, the classifier could be better adapted to a wider range of environments and movement patterns. Instead of looking at the entire point cloud, the clustering would attempt to find smaller point clouds representing the people walking. After the clustering algorithm has been applied, a network could look at the amount of clusters found, the size of the clusters, the distance between the clusters, the velocity and direction the clusters are moving in, and so on to try and detect piggybacking.

5.5.3 Variety of scenarios

Another interesting point for future research would be to further explore the relationship between the amount of categories and the overall accuracy. If the indications of more scenarios in the training data resulting in generally lower accuracy holds true, it might be of interest to train a network with specific categories for each implementation. Such an endeavor would be aided by the relatively short training times for the PointNet based architecture.

Additionally, while a small variety of scenarios were included in the tests performed on the model, achieving sufficient reliability for a general use case would likely involve training for many more scenarios. The set of scenarios may also vary depending on the environment of a prospective use case. Creating a tailored model could be an interesting solution if the potential relationship described in the previous paragraph is proven. For example, there might be environments where some sort of cart is frequently used for transporting goods and must be included in training data. Those datasets need not be included in an environment such as a typical office building, and could then be excluded for improved performance.

5.5.4 Parameters for the real-time classification

The design of (36) for calculating the final classification score of a sequence was chosen somewhat arbitrarily, and could be partially blamed for the reduced accuracy in (P_f). Based on a handful of observations made during testing, reducing the kernel size used for calculating the highest mean activation would've resulted in more correct classifications. Such a change would of course also affect the other categories, making

the system more biased towards piggybacking across the board. Even if (P_f) is considered not to be of interest, it is possible that the tweaking of this parameter could offer significant improvement for relatively little effort.

Further testing could be performed by keeping full records of the sequences of activations for a set of real-time tests. These sequences could then be run through a program which performs the final prediction using various kernel sizes when calculating the highest mean activation, to observe which configuration performed best overall. These sequences are printed to the terminal in the existing software, but complete records were unfortunately not kept while performing the manual tests.

6 Conclusion

With the results discussed previously the first two research questions, (RQ1) and (RQ2) can be answered confidently. Radar is a viable solution for detecting piggybacking and the radar solutions perform better overall than the existing camera solution. Answering (RQ3), whether the roof placement or the reader placement is better is less straightforward. On average they perform almost equally with an average accuracy across the 9 classes of around 93%. It is, however, most likely easier to increase this accuracy for the roof position. Its average across the classes is lowered drastically by its score on (P_f) which could be solved by lowering the memory size or altering the height or angle of the radar module.

Category	**Reader %**	**Roof %**	**Camera %**	**Samples**
(NP)	99.5	100	98.25	400
(NP_c)	100	100	28	100
(P)	100	99.75	100	400
(P_h)	67	93	16	100
(P_d)	78	100	98	100
avg %	88.9	98.55	68.05	

Table 15: Table comparing the results of the three-network models and the camera solution on five chosen categories.

One can also argue about which categories are more important than others. Without doubt, (NP) and (P) are the most important since they will make up the majority of real world situations. (P_d) and (P_h) could be considered the second most important categories, since these could be ways that mischievous people try to trick the system. (NP_c) could also be considered an important category, since this is something that happens often in an office environment, especially when including tangential scenarios such as a person carrying a desktop computer or similar objects. The results when only considering these 5 categories can be seen in Table 15, they paint a different story. Across the five categories, the reader placement achieves an average accuracy of 88.9, the roof placement achieves an average accuracy of 98.55 and the camera solution achieves an average accuracy of 68.05. Considering only these five categories make the roof placement look like the ideal placement. If considering other properties of the placements than their technical prowess, arguments can be made that the reader placement is better. For example if it is possible to incorporate the radar module into a card reader, that eliminates the need for an additional device above the door. This has several advantages such as easier installation and maintenance, less wiring required and possibly being cheaper to produce. Since (AP1) defines a purpose of this project as attempting to develop a neural network that is as effective as possible at detecting piggybacking, the economical and practical arguments are disregarded, and the roof placement is considered to be better.

References

[1] Anton Almqvist and Anton Kuusela, "Pose classification of people using high resolution radar indoor," M.S. thesis, Lunds Tekniska Högskola, 2022.

[2] E.J. Barlow, "Doppler radar," *Proceedings of the IRE*, vol. 37, no. 4, pp. 340–355, 1949. DOI: 10.1109/JRPROC.1949.231638.

[3] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu, *Shapenet: An information-rich 3d model repository*, 2015. DOI: 10.48550/ARXIV.1512.03012. [Online]. Available: https://arxiv.org/abs/1512.03012.

[4] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. arXiv: 1406.1078. [Online]. Available: http://arxiv.org/abs/1406.1078.

[5] E. M. Dogo, O. J. Afolabi, N. I. Nwulu, B. Twala, and C. O. Aigbavboa, "A comparative analysis of gradient descent-based optimization algorithms on convolutional neural networks," in *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)*, 2018, pp. 92–99. DOI: 10.1109/CTEMS.2018.8769211.

[6] Fengli Gao and Huicai Zhong, "Study on the large batch size training of neural networks based on the second order gradient," *CoRR*, vol. abs/2012.08795, 2020. arXiv: 2012.08795. [Online]. Available: https://arxiv.org/abs/2012.08795.

[7] Hossein Gholamalinezhad and Hossein Khosravi, *Pooling methods in deep neural networks, a review*, 2020. DOI: 10.48550/ARXIV.2009.07485. [Online]. Available: https://arxiv.org/abs/2009.07485.

[8] Alex Graves and Jürgen Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural Networks*, vol. 18, no. 5, pp. 602–610, 2005, IJCNN 2005, ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2005.06.042. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608005001206.

[9] Kevin Gurney, *An introduction to neural networks*. UCL Press, 1997.

[10] Douglas M. Hawkins, "The problem of overfitting," *Journal of Chemical Information and Computer Sciences*, vol. 44, no. 1, pp. 1–12, 2004, PMID: 14741005. DOI: 10.1021/ci0342472. eprint: https://doi.org/10.1021/ci0342472. [Online]. Available: https://doi.org/10.1021/ci0342472.

[11] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[12] Cesar Iovescu and Sandeep Rao, "The fundamentals of millimeter wave radar sensors," 2021.

[13] Diederik P. Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, 2014. DOI: 10.48550/ARXIV.1412.6980. [Online]. Available: https://arxiv.org/abs/1412.6980.

[14] Davi Lazzarotto and Touradj Ebrahimi, *Sampling color and geometry point clouds from shapenet dataset*, 2022. DOI: 10.48550/ARXIV.2201.06935. [Online]. Available: https://arxiv.org/abs/2201.06935.

[15] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. "The mnist database of handwritten digits." (), [Online]. Available: http://yann.lecun.com/exdb/mnist/ (visited on 10/17/2022).

[16] Dominic Masters and Carlo Luschi, *Revisiting small batch training for deep neural networks*, 2018. DOI: 10.48550/ARXIV.1804.07612. [Online]. Available: https://arxiv.org/abs/1804.07612.

[17] Farid Ghareh Mohammadi, Cheng Chen, Farzan Shenavarmasouleh, M. Hadi Amini, Beshoy Morkos, and Hamid R. Arabnia, *3d-model shapenet core classification using meta-semantic learning*, 2022. DOI: 10.48550/ARXIV.2205.15869. [Online]. Available: https://arxiv.org/abs/2205.15869.

[18] Seokhyun Moon, Wonho Zhung, Soojung Yang, Jaechang Lim, and Woo Youn Kim, "Pignet: A physics-informed deep learning model toward generalized drug target interaction predictions," *Chem. Sci.*, vol. 13, pp. 3661–3673, 13 2022. DOI: 10.1039/D1SC06946B. [Online]. Available: http://dx.doi.org/10.1039/D1SC06946B.

[19] Michael A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[20] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of the 30th International Conference on Machine Learning*, Sanjoy Dasgupta and David McAllester, Eds., ser. Proceedings of Machine Learning Research, vol. 28, Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1310–1318.

[21] Charles R. Qi, "Cvpr17 machine learning 1 - pointnet: Deep learning on point sets for 3d classification and segmentation," 2017. [Online]. Available: https://www.youtube.com/watch?v=Cge-hot0Oc0 (visited on 10/17/2022).

[22] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas, *Pointnet: Deep learning on point sets for 3d classification and segmentation*, 2016. DOI: 10.48550/ARXIV.1612.00593. [Online]. Available: https://arxiv.org/abs/1612.00593.

[23] L. Rabiner and B. Juang, "An introduction to hidden markov models," *IEEE ASSP Magazine*, vol. 3, no. 1, pp. 4–16, 1986. DOI: 10.1109/MASSP.1986.1165342.

[24] Pavlo Radiuk, "Impact of training set batch size on the performance of convolutional neural networks for diverse datasets," *Information Technology and Management Science*, vol. 20, pp. 20–24, Dec. 2017. DOI: 10.1515/itms-2017-0003.

[25] Sebastian Ruder, *An overview of gradient descent optimization algorithms*, 2016. DOI: 10.48550/ARXIV.1609.04747. [Online]. Available: https://arxiv.org/abs/1609.04747.

[26] Grant Sanderson. "Playlist: Neural networks." (), [Online]. Available: https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi (visited on 10/17/2022).

[27] Shai Shalev-Shwartz and Shai Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[28] Marina Sokolova, Nathalie Japkowicz, and Stan Szpakowicz, "Beyond accuracy, f-score and roc: A family of discriminant measures for performance evaluation," vol. Vol. 4304, Jan. 2006, pp. 1015–1021, ISBN: 978-3-540-49787-5. DOI: 10.1007/11941439_114.

[29] Xue Ying, "An Overview of Overfitting and its Solutions," in *Journal of Physics Conference Series*, ser. Journal of Physics Conference Series, vol. 1168, Feb. 2019, 022022, p. 022 022. DOI: 10.1088/1742-6596/1168/2/022022.

[30] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang, "A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures," *Neural Computation*, vol. 31, no. 7, pp. 1235–1270, Jul. 2019, ISSN: 0899-7667. DOI: 10.1162/neco_a_01199. eprint: https://direct.mit.edu/neco/article-pdf/31/7/1235/1053200/neco_a_01199.pdf. [Online]. Available: https://doi.org/10.1162/neco%5C_a%5C_01199.

Master's Theses in Mathematical Sciences 2022:E67
ISSN 1404-6342

LUTFMA-3489-2022

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

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