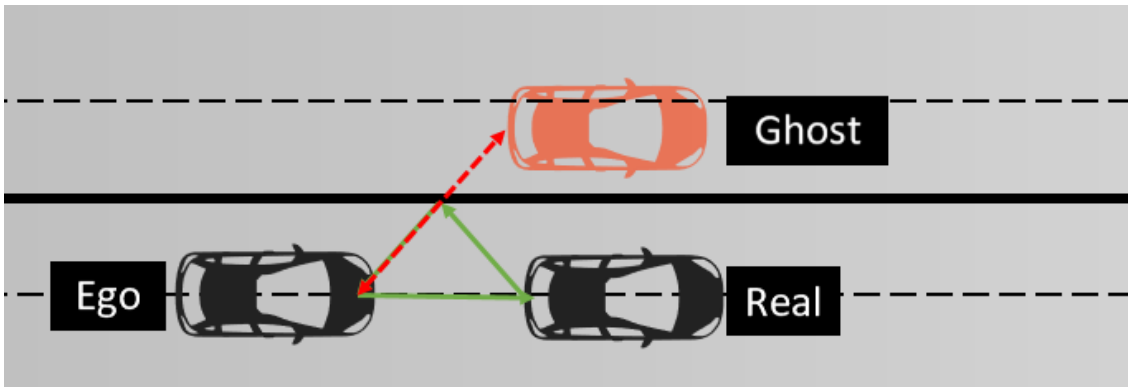


Radar Ghost Detection and Elimination

Classification of Radar Ghost Objects in Highway Traffic using CART and Neural Network Classifiers.

Master's Thesis in Complex Adaptive Systems & Systems, Control and Mechatronics

Foad Alhayek
Svante Trelsmo

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Radar Ghost Detection and Elimination

Classification of Radar Ghost Objects in Highway Traffic using CART and Neural Network Classifiers.

FOAD ALHAYEK
SVANTE TRELSMO

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2022

Radar Ghost Detection and Elimination
Classification of Radar Ghost Objects in Highway Traffic using CART and Neural Network Classifiers.
FOAD ALHAYEK
SVANTE TRELSMO

Supervisor: Sebastian Oleszko, Aptiv
Examiner: Jonas Sjöberg, Mechatronics, Electrical engineering

Master's Thesis 2022
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Ghost object appearance scenario where a real car and guardrail cause the radar signal to bounce in a way that cause a ghost to appear on the other side of the guardrail from the ego vehicle.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Radar Ghost Detection and Elimination
Classification of Radar Ghost Objects in Highway Traffic using CART and Neural Network Classifiers.
FOAD ALHAYEK, SVANTE TRELSMO
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Advanced Driver Assistance Systems rely on sensors, which provides information about the environment, to make safe decisions. One commonly used sensor is radar. Radars are great at measuring distance and velocity of objects. However, one drawback is that they are prone to trigger false alarms and indicate objects exist when there are none. These false alarms are called ghosts and arise mainly from multi-path propagation phenomena.

To eliminate ghosts, a common method is to use sensor fusion with multiple sensors, which is both expensive and complex. In this thesis a method is proposed for eliminating ghost objects solely based on radar data. The proposed method utilizes classifiers, such as Classification and Regression Trees and Neural Network based models. All models are trained on a highway traffic dataset, with LiDAR data as ground truth.

The results indicates that two classifiers, Fully Connected (FC) Network and Random Forest (RF), performed the best, with scores of around 96% & 95% in accuracy, recall and precision respectively. Both classifiers outperform the more advanced Attention based network by around 15 percentage points. With fewer features used, the RF classifier outperforms the FC network. Which indicates that the RF requires less information than the FC to distinguish between ghost and real objects. Moreover, the objects historical behaviour did not have a positive impact when classifying objects.

Keywords: Radar Ghosts, LiDAR, Attention, CART, Machine Learning, Random Forest, ADAS.

Acknowledgements

We would like to take the opportunity to thank everyone involved with the project at Aptiv for their support. In particular we want to thank our supervisor Sebastian Oleszko for the valuable input and feedback as well as the practical support he has given us. Finally we want to send our gratitude to our examiner Jonas Sjöberg for enabling us to do this project and for the guidance.

Foad Alhayek, Svante Trelsmo, Gothenburg, June 2022

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AEB	Automatic Emergency Breaking
AD	Autonomous Driving
ADAM	Adaptive Moment Estimation
ADAS	Advanced driver-assistance systems
BEV	Birds Eye View
FN	False Negative
FP	False Positive
GD	Gradient Descent
GT	Ground Truth
LiDAR	Light Detection and Ranging
NaN	Not-a-Number
NN	Neural Network
Radar	Radio Detection and Ranging
RMSE	Root Mean Square Error
RMSP	Root Mean Square Propagation
SGDM	Stochastic Gradient Descent
SNR	Signal to Noise Ratio
TN	True Negative
TP	True Positive

Contents

List of Figures

List of Tables

1

Introduction

This chapter presents an overview of the thesis. It includes a background to the problem, why it is important to investigate, what the end goal is, and the proposed solution.

1.1 Background

Radar sensors are cost-effective and provide decent measurements of objects' sequential behavior in the operating environment. The sequential data is then used in areas of autonomous driving and advanced driver assistance systems (AD/ADAS)[1]. However, some flaws originate from the way a radar works. One of these is the creation of "ghost" objects. Ghosts appear when the radar clusters detections and create objects where there are no real objects. As seen in Figure 1.1, a ghost can appear when there is a vehicle in front and a guardrail on the side. Here the outgoing radar signal bounces on the vehicle and barrier, which causes the sensor to believe that there is another vehicle on the other side of the guardrail.

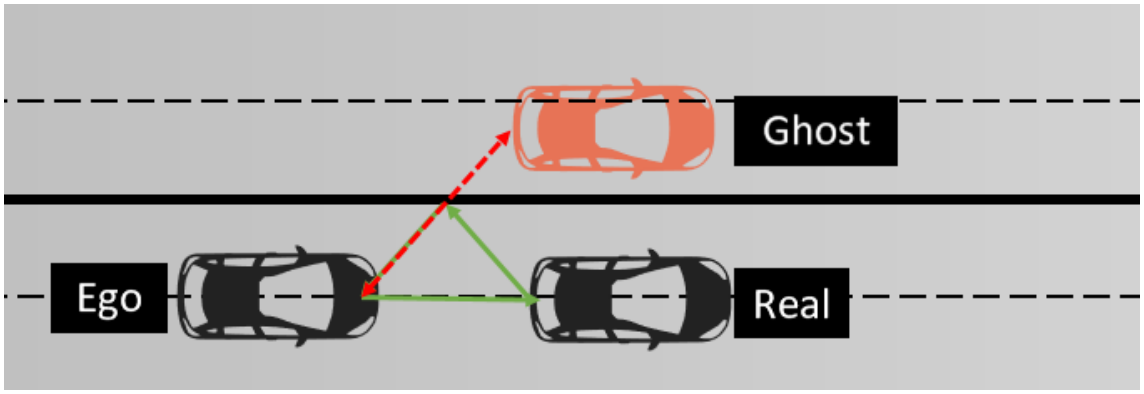


Figure 1.1: Example of a ghost object occurrence where the ghost appears on the adjacent lane behind a guardrail. The green arrows indicate the true path of the radar signal. The red striped arrow show the estimated path.

Some other examples where ghosts interfere with the AD/ADAS are when ghosts appear close to the ego vehicle, seen in Figure 1.2. In this scenario, the uneven surface of the real vehicle causes the radar signal to bounce on the ground in such a way that a ghost appear right in front of the ego vehicle. The ground bounce can also occur without another vehicle in the frame, as shown in Figure 1.3, where the road elevation ahead could cause false-positive detections.

In these cases, the ECU (Electronic Control Unit) can issue the AEB-system (Autonomous Emergency Braking) to trigger and thus abruptly apply full breaks to the ego vehicle. This can be an inconvenience and in some cases even dangerous, as vehicles traveling behind the ego vehicle might crash into the ego vehicle.

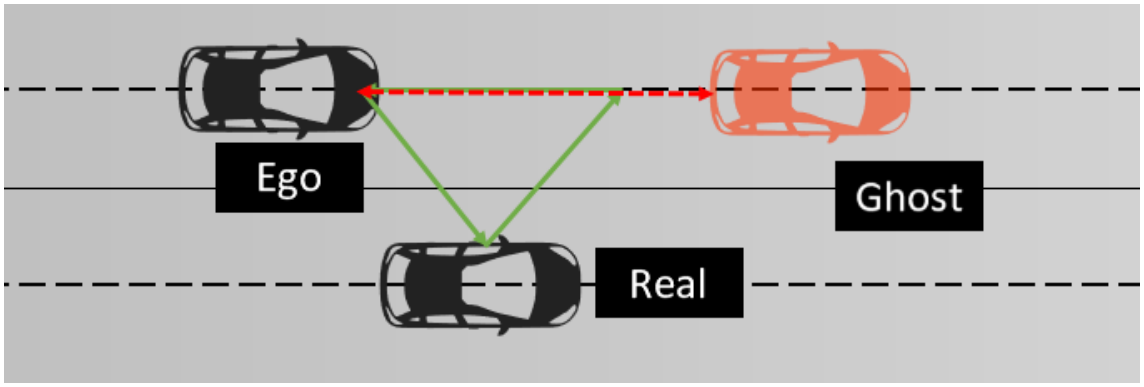


Figure 1.2: Example of a dangerous ghost object occurrence where the ghost appear straight ahead of the ego vehicle. The green arrows show the path of the radar signal and the red striped arrow show the estimated path.

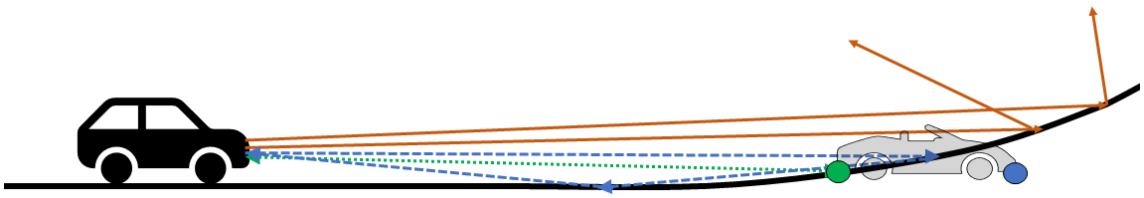


Figure 1.3: Side view of ground bounce which cause a ghost appearance in front of ego vehicle. The green and blue arrows/dots represent detections cause false positive detections. The brown arrows indicate signals that are scattered away from the radar sensor. In gray the generate ghost object is shown.

To reduce the number of ghost targets, one could utilize sensor fusion with e.g. radar and LiDAR (Light Detection and Ranging) [2]. The LiDAR has a much higher resolution than radar, increasing its accuracy and ability to eliminate dangerous ghosts. However, one drawback of LiDAR is its poor operation in adverse weather. Moreover, reducing the number of ghosts purely on radar inputs is beneficial, as this would reduce the cost of production and increase weather robustness.

1.2 Contributions

The thesis evaluates the performance of classifiers in eliminating ghost objects based solely on radar data while not removing real objects. Promising results are presented with a simple fully connected network and Random Forest. Furthermore, the thesis shows that "Longitudinal position variance" and "Lateral position" features had the largest importance, while "Relative time" and "Curvature" had the least importance and impact on the model's performance. Lastly, an interesting discovery indicates that the objects historical behaviour has negligible impact when classifying between ghost and real objects.

1.3 Related research

In recent years several methods for false-positive detection classification have been investigated. Most papers utilize machine learning methods to model the behavior of radar ghost detections [3, 4, 5]. These papers all use radar data that is initially not annotated and, as such cannot use supervised learning directly. In order to use existing networks such as PointNet and PointNet++, which are supervised learning networks, authors in [4, 5] utilized LiDAR and GNSS respectively as ground truth to label the data autonomously. On the other hand, authors in [3] built their own transformer based neural network in order to train the network to learn connections between detections and not just features, such as radar cross-section and Doppler velocity.

Another critical aspect is that [3, 4] uses a dataset called nuScenes to verify their algorithms. The nuScenes-dataset consists of data from the three most common sensors used in AD/ADAS applications; LiDAR, camera, and radar, as well as GPS and IMU [6]. This differentiates the nuScenes dataset from other publicly available datasets, which focus on cameras only.

Most previous research utilize detection-level data, i.e., before data has been clustered to objects, to eliminate radar point detections. In contrast, this thesis present classifiers which utilize objects already created by a tracker system to eliminate the ghosts in a later stage of the tracking process.

1.4 Proposed solution

The proposed system is based on data collection from radars and a LiDAR to generate a labeled dataset with environmental information. The environmental information is taken from the radar input and the label, indicating real or ghost, is generated utilizing LiDAR input. A classifier is then trained on the dataset to enable it to distinguish ghosts from real objects solely on radar data. An overview of the proposed system is shown in Figure 1.4. The blue boxes show the pipeline of the radar data, whereas the orange boxes show the LiDAR data. These are merged and the combination is shown in green.

First, information about the environment is collected from these separate sensors. This raw data is often noisy and needs to be processed to extract useful information. This process is executed in the ”Signal Processing” block. Here the signal is also converted from analog to digital in the processing block. The discrete signal is then passed to an algorithm tasked with creating objects, such as cars, trucks, and motorcycles. The radar objects are created based on traditional classification algorithms, whereas the LiDAR objects are generated with a neural network. The LiDAR objects are then set as ground truth and used to generate the ghost/real label, which is attached to the radar objects in the merger. Then the merged objects are forwarded to a classifier that utilizes supervised learning and tries to classify the radar objects as true or false positives. These predictions are then used to filter out ghosts and feed real objects to the next layer in the full AD/ADAS.

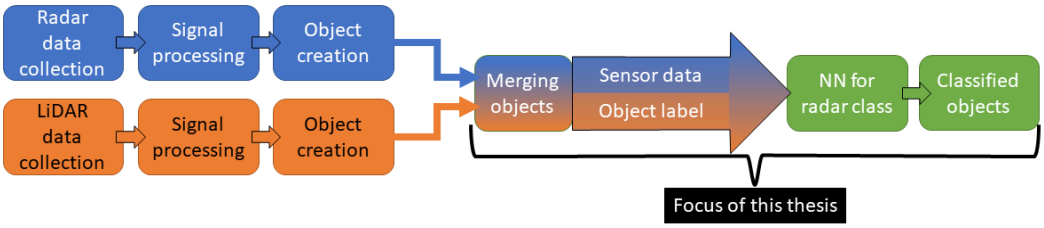


Figure 1.4: Overview of the pipeline with each step of the process, from data collection to classified objects illustrated. The blue boxes refer to radar related operations and information, orange refer to LiDAR, and the green correspond to classifiers.

1.5 Outline of Thesis

The thesis consists of five chapters, with the first chapter containing the introduction and background to the problem. Here an overview of the problem and proposed solution is also presented. The second chapter presents a technical background with the fundamentals of radar, LiDAR, object tracking, data processing, and machine learning. This chapter can be skipped if the reader has a background in these areas.

In the third chapter, both method and results of data collection and processing are presented. The results from the data analysis can be found in Section 3.6. The fourth chapter follows the same structure as the third. Here the reader will find the method and results for the classifiers used. In Section 4.5 are the results for all implemented classifiers. The thesis is concluded with a discussion and conclusion about the results and insights from the analysis.

2

Technical Background

This chapter covers some underlying theory which is utilized in the thesis. Here the fundamental theory and functionality of the radar sensor is covered, and a brief overview of the LiDAR sensor's functionality. The construction of objects, how they are tracked, and how ghost objects are created is described. The chapter ends with the underlying technical building blocks of machine learning and classification algorithms used.

Note that the technical background can be skipped if the reader is proficient in the mentioned areas. The chapter is referred back to in Chapters 3 and 4, and is available for those interested in learning more about the topics.

2.1 Functionality of Radar

A radar (RAdio-Detection-And-Ranging) system aims to detect objects in the environment. To achieve this, the radar utilizes electromagnetic waves emitted from a transmitter. These waves then hit objects and scatter in unknown directions, depending on the geometry and object material, with some of them bouncing back to the receiver. The signals that return are then processed in order to determine where they bounced from [7].

The process of a single antenna radar starts with a pulse generator which constructs the signal to be transmitted [1, 7]. This signal is then passed through an upconverter, which increases the signal's frequency, and an amplifier boosts the power of the signal. The duplexer, which is a switch, places the antenna in transmitting mode. As the radar utilizes the same antenna for transmission and receiving, the duplexer switch is required to disable the mode that is currently not in use and activate the one that is. After this step, the signal is passed onto the antenna and transmitted to generate information about the environment. As soon as the signal is sent the duplexer places the antenna in receiver mode to be ready to catch the bounced signal.

When the antenna obtains the signal, it is passed through a low-noise amplifier in order to increase the amplitude of the entire signal while keeping the SNR (Signal to Noise Ratio) as high as possible [8]. A downconverter is then utilized to reduce the frequency to match the frequency transmitted signal. At this stage, the signal

processing algorithms extract the distance to the target, angular position, and relative radial velocity. These values are then bundled together into a container called a detection. This process is illustrated in Figure 2.1.

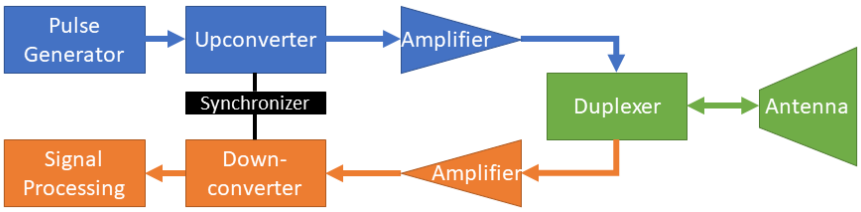


Figure 2.1: High-level schematic of a single antenna radar system. The blue boxes are the transmitter part, the orange boxes are the receiver, and green are the shared parts.

2.1.1 Estimation of Distance to Target

In order to measure the distance to target it is possible to utilize an equation called "The Radar Equation" [1]. This equation calculates the relation between transmitted and received powers, and is defined as

$$\frac{P_r}{P_t} = \frac{G_t G_r \lambda^2 \sigma_s}{(4\pi)^3 R^4} \tag{2.1}$$

where P_r and P_t are the power, and G_r and G_t are the amplification of the received and transmitted signal, respectively. λ is the wavelength, and σ_s is the radar cross-section. With both the transmitted and received powers known, it is possible to use this equation to calculate the distance which the signal has traveled, i.e. distance to target:

$$R = \left[\frac{G_t G_r \lambda^2 \sigma_s P_t}{(4\pi)^3 P_r} \right]^{\frac{1}{4}} \tag{2.2}$$

Another way to calculate the range to targets is to utilize "Time-of-Flight". This is computed by calculating the time difference from when the signal is transmitted to when it is received, multiplied with the speed.

2.1.2 Velocity Estimation

To measure the velocity of an object the Doppler effect is utilized where the relative radial velocity can be calculated from the difference in transmitted and received frequency [1, 7]. It is described as

$$f_r - f_t = 2v\frac{f_t}{c - v}, \quad if\ c \gg v \Rightarrow$$
$$v = \frac{(f_r - f_t) \cdot c}{2f_t} \tag{2.3}$$

where f_r refers to the received frequency, and f_t is the transmitted frequency.

2.1.3 Angular Estimation

The angle to an object is more complex to extract from the received signal than the previous features. Several methods can be applied to extract the information, J.Gamba [1] differentiates these into three categories; quadratic, linear prediction, and subspace. These methods will not be covered here as they are outside the scope of the thesis, but the reader can read more in [1, 9].

2.1.4 Automotive Radar Characteristics

In automotive applications, the frequency band utilized is between 24 and 100 GHz, with research currently active in the +100 GHz range [1]. This frequency band lies in the millimeter band and has the advantage of small antenna sizes, high resolution, and low interference between radars. All of these factors make the GHz-radar advantageous for automotive applications. However, a negative aspect of mm-waves is that they perform poorly at long distances. In automotive applications, long-range is around 250m, whereas long-range for radar applications, in general, is several 100 kilometers, e.g. Over-The-Horizon radar [1]. Thus this is not an issue for automotive purposes, and the band can efficiently be used.

2.2 Overview of LiDAR functionality

LiDAR (Light Detection and Ranging) is a sensor that operates similarly to the radar sensor. The main difference is that the LiDAR utilizes laser pulses to detect objects and higher frequencies in the transmitted signal. As stated in Section 2.1.4, the state-of-the-art highest resolution radar sensors used in automotive applications operate in around 24-70 GHz. In contrast the LiDAR utilizes waves with frequencies of around 200-300 THz at wavelengths, i.e. a factor of 10 000 higher [10]. Furthermore, an increased frequency improve the resolution in the environmental images gathered, and consequently increase the accuracy of object tracking. By utilizing the higher frequencies the LiDAR is not as sensitive to the multi-path problem as the radar, due to the signal strength reducing a lot more with each bounce.

The disadvantage of using a LiDAR is initially, the cost of the sensor. A high quality LiDAR costs from around 1000 USD and radar starts at around 50 USD [11, 12]. As such, due to the small margins in car production, a price reduction of around 900 USD is significant. The LiDARs accuracy also deteriorate in adverse weather,

e.g., heavy rain, fog, or snow. It becomes almost useless as the laser pulses bounce off the raindrops rather than objects of interest. This disruption is further shown and evaluated in [13].

2.3 Object Tracking Methodology

In automotive applications, it is essential to continuously track dynamic objects in the environment. The radar acts independently producing instantaneous images of the surroundings. However, to state anything about the objects, it has to be supplemented with an object tracker. The task of the object tracker is to associate the input data to an object, either new or previously existing, filter out faulty detections, and estimate object behavior.

To predict the movement of objects, the conventional method utilizes Kalman filters [1]. A Kalman filter is a recursive optimal estimator that uses measurements and prior knowledge to make predictions on an object's behavior. The process of the Kalman filter consists of a prediction and update. The goal of the prediction is to calculate the probability of state, x at time k, given all previous measurements, $y_{1:k-1}$. The update step then utilizes the current measurement, y_k, to give the probability of x_k given $y_{1:k}$. Although the Kalman filter is a highly efficient estimator its assumptions, that the noise is Gaussian and the dynamical system is linear, is a major drawback. To circumvent the assumptions, Bayesian filtering and the particle filter have gained popularity [14].

The particle filter uses a set of particles and weights that estimate the system's states and can estimate any distribution. The particle filter is initialized by randomly assigning values to particles and uniformly distributing the weights. The particles are then updated with importance sampling to give them the value corresponding to $p(x_k|x_{k-1})$, i.e. probability of state x_k given previous state x_{k-1}. The weights are updated to correspond to the probability of the measurement y_k given the state x_k, and then normalized. Finally, the estimation of the state is updated with both the particles and weights. A drawback of this method is the degradation of particles where the weights get concentrated around one particle resulting in poor estimation. Resampling can be added to reset both the particles and weights when one particle gets too large weight. More detail on both Kalman filters and the particle filter can be found in [15].

Objects contain different amounts of detections over time and, in some time instances, might be blocked entirely from the radars. The tracker thus has to keep track of objects in the absence of sensor input. The Kalman filter and particle filter are easily modified to enable this feature. Both algorithms update their estimates based on the previous measurement; if a measurement is missing, it is common to skip the update-step and purely use the predicted state for the next iteration. The task is then instead to determine when an object disappears. A simple method is to remove objects when they have no associated measurements for a set period of time.

2.3.1 Multi-object tracking

As cars operate in environments where it is crucial to track multiple objects at each instance, a method for keeping track of which object is which and what new sensor data should be assigned to pre-existing and new objects respectively is needed [16]. There are several methods for assigning detections to objects; Global Nearest Neighbors, Generalized Nearest Neighbors, Multiple Hypothesis Tracker, and Joint Probabilistic Data Association.

2.3.2 Multi-path propagation problem

When tracking objects, it is sometimes the case that the signal transmitted does not take a straight path back to the sensor. Objects and structures have shapes causing the signal to bounce in different directions. Most of these bounces do not make it back to the sensor, however, some make it back to the sensor. This phenomenon is called Multi-Path Propagation [17]

Multi-Path Propagation is the leading cause of ghost objects appearing. When the transmitted signal does not take a path straight back to the sensor but returns from another angle, it can fool the sensor into thinking there is an object at a location where there is none.

2.4 Pre-processing methods

Real-world data is often cluttered and chaotic, which makes it difficult for humans and computers to draw valuable insights from it. Some steps that facilitate analysis and increase usefulness include cleaning and feature selection, feature scaling, and smoothing.

Data cleaning is the process of cleaning the dataset from erroneous data, such as sensor errors, and filtering the dataset only to include useful features for the problem. The feature selection can be difficult as it might be almost impossible to determine which features distinguish the objects even with deep knowledge in the area. There are, however a few methods for selecting features, of which some are a byproduct of classifiers. The classifiers and how the feature importance is calculated are covered later in Section 2.8.

Feature scaling is a method of modifying the range between $[0, 1]$ or $[-1, 1]$. One way to scale a dataset is to divide the whole array, $\mathbf{x}$, of values by the absolute maximum value such that the new range of values is between negative one and one,

$$\mathbf{y} = \frac{\mathbf{x}}{|\max(\mathbf{x})|}. \tag{2.4}$$

In most cases however, some outliers affect the range such that the majority of values are close to zero. To combat this, it is advantageous to utilize percentiles and divide by a set percentile value,

$$\mathbf{y} = \frac{\mathbf{x}}{\mathrm{Q}(\mathbf{x},\, \mathrm{P})} \tag{2.5}$$

where P is the P-th percentile, and Q is the percentile function that yields the value. Important to note is that the values are no longer bounded between negative one and one.

2.4.1 Outlier removal with Isolation Forest

Isolation Forest is a method which removes outliers in a dataset [18]. It is an unsupervised learning method that focuses on anomaly isolation. Moreover, Isolation Forest is an accurate and efficient anomaly detector, especially for a large dataset. Furthermore, the method is based on decision trees, where the trees represent recursive partitioning. Partitions are generated by randomly selecting a feature and then randomly selecting a split value between the minimum and maximum values of the selected feature. A normal data point generally requires more partitions to be isolated versus an anomaly point. An anomaly score is used for decision making and is defined as

$$s(x, n) = 2^{-\left(\frac{E(h(x))}{c(n)}\right)}, \tag{2.6}$$

where s is the anomaly score, x is the data point, n is the number of data points, $h(x)$ is the path length of x, $E(h(x)$ is the average of $h(x)$ from a collection of isolation trees, and $c(n)$ is the average path length of unsuccessful search in Binary-Search-Tree and is defined as:

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n} \tag{2.7}$$

where $H(i)$ is the harmonic number and is estimated to be $ln(i) + e$ where e is the Euler's constant. Therefore, the outlier decision is based on when:

- s is very close to 1 which indicates an anomaly (outlier).
- s is much smaller than 0.5 which indicates a normal data point.
- all instances return $s \approx 0.5$ which indicates the entire dataset does not have any distinct anomaly.

2.5 History & time embedding of tracked objects

When dealing with sequential data, it is crucial to incorporate the time data in some way. Previously it was common to add this information as another feature in the input. This, however showed poor results as the network did not take advantage of

it properly. To combat this, Kazemi et al. [19] introduced a vector representation of time called "Time2Vec" (T2V),

$$\mathrm{T2V}(\tau)[i] = \begin{cases} \omega_i \tau + \phi_i & \text{if } i = 0 \\ F(\omega_i \tau + \phi_i) & \text{if } 1 \leq i \leq k \end{cases} \tag{2.8}$$

where τ is a scalar representation of time, ω and ϕ are tunable parameters, and F is a periodic function, e.g. sine or cosine. The two parts of the vector enable both non-periodic and periodic patterns to be embedded. T2V also has another advantage, which is that it is time-invariant. It can thus handle time inputs on different timescales.

To observe an object over time and possibly make a more accurate prediction of its class, the object's history should be embedded. A simple way of incorporating this information is to use a deque (double-ended queue)[20]. Deque is an abstract data type that enables storing and extracting elements from both sides of the queue. This enables new data to be pushed in from one side and removes the oldest data from the other side. The historical data about each object is either kept or expelled depending on how long the object has lived and if it is still alive.

2.6 Building blocks of Neural Networks

In the field of data science and machine learning, common methods to use are deep learning and neural networks (NN) [21]. The NNs are, as the name implies, built upon imitating neurons in our brains [22]. These neurons can either be active (1) or inactive (-1), and the decision to determine whether to activate the neuron or not is based on a weighted sum of input values which are passed through an activation function called signum. In Figure 2.2 the functionality of a single neuron is depicted and is mathematically defined as

$$z = \sum_{i=1}^{N} w_i x_i + b_i \tag{2.9}$$

where w is the weight, b is the bias, and z is the summed scalar before activation function is applied.

The signum function is defined as

$$\mathrm{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ +1 & \text{if } z > 0 \end{cases}. \tag{2.10}$$

However, this limit the application of the neuron, and thus other activation functions which give the neuron values within a set range are preferred [22]. Alternative activation functions are e.g. Logistic Sigmoid, ReLU, and Softmax, which are defined as

$$\sigma(z) = \frac{1}{1+e^{-z}}, \tag{2.11}$$

$$\text{ReLU}(z) = max(0, z), \tag{2.12}$$

$$\text{Softmax}(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \text{ for } j = 1, ..., K, \tag{2.13}$$

where K is the number of neurons in the layer [22, 23]. Worth noting is that Softmax is applied to all neurons at once to create a probability. It is thus commonly used in the last layer of networks.

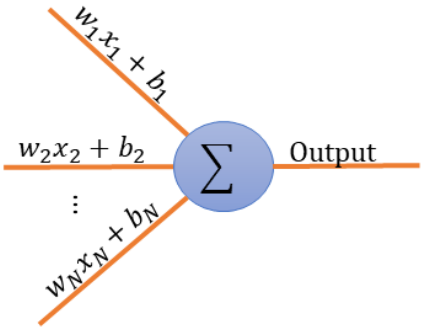


Figure 2.2: Function of a single neuron used for constructing neural networks. N inputs are passed to the neuron where they are summed, and an activation function is applied to transform the summed weights into an output from the node.

A neural network can be created by stacking multiple neurons and connecting them to each other. The most basic neural networks are called fully connected networks; an example of these types is shown in Figure 2.3. Here the blue dots represent neurons, and the orange lines are the connections between the neurons. As seen, each neuron in the previous layer is connected to each neuron in the next layer, and the network is thus fully connected.

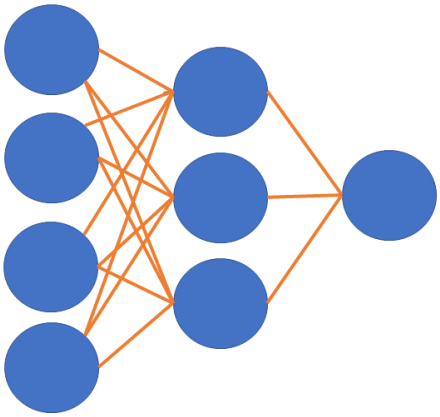


Figure 2.3: Example of a fully connected neural network. The blue dots represent the neurons which are connected, in orange, with the weights and biases in order to create a neural network.

2.6.1 Training neural networks

A standard practice when building networks is to randomly initialize the weights and biases; which are then tuned through the training process. During training, there are mainly two methods, unsupervised and supervised learning [21]. The fundamental difference between these two is that supervised learning utilizes a dataset where the target is known and labeled. Hence, the target can be seen as a ground truth, i.e. what the network's output should be. Unsupervised learning does not have this luxury. Thus, it is commonly used to either cluster data or to find similarities in the input data, i.e., high-level perspective and relationships between the features.

In supervised learning, there are several optimizers available to use [24]. These optimizers are based on Gradient descent (GD) which utilizes the Jacobian of a loss function, described later in Section 2.6.3, to update the weights and biases in the NN. This method however, is not optimal and thus other adaptations have to be created.

Some examples of other optimizers are Stochastic Gradient Descent (SGDM), Mini-Batch Gradient Descent, Momentum, Root Mean Square Propagation (RMSP), and Adaptive Moment Estimation (Adam) [21, 24, 25]. SGDM builds upon GD and includes randomness in the training method which can decrease training time and reduce the risk of getting stuck in local minima. Mini-Batch GD splits the dataset into batches to reduce memory usage and increase parameter update frequency. Momentum is useful in high variance scenarios where the loss function causes the update to oscillate instead of going towards the minimum. RMSP adapts the learning rate, described later in Section 2.6.2, to avoid exploding and vanishing step sizes. Finally, Adam combines some of the previous features to build an optimizer with all the advantages mentioned. However, the drawback of this is that it becomes computationally heavy and thus increases training time.

2.6.2 Hyperparameters

When training networks, there are multiple parameters that can be tuned to optimize the process. These parameters are commonly referred to as hyperparameters [26]. These differ from "normal" parameters in that they are separate from the model itself but rather modify external parts of the training process. Where "normal" parameters are, for example the weights and biases in the network, the hyperparameters are, e.g., learning rate and batch size.

The batch size is a chosen value that splits the dataset into evenly sized batches. If the splits does not evenly split the dataset the last uneven batch is removed. Utilizing a large batch size could reduce training time since the backpropagation is run once per batch. However, there are a few drawbacks from this, the first of which is memory size. If a too large batch size is used it is highly likely that hardware constraints will be a limiting factor. By reducing the batch size, the hardware constraints can be kept, and the weights and biases are updated in a more stochastic process. With smaller batch sizes there is a larger variance in the updating of weights and biases, and thus the risk of getting stuck in local minima or saddle points decreases [27].

Learning rate is directly related to updating weights and biases as it determines how long steps should be taken at each update. This value is multiplied with the gradient calculated from the loss function. It is also common to modify the learning rate throughout the training process.

2.6.3 Loss functions

There are several loss functions and they are all advantageous for different types of problems [28]. The problem of classifying ghosts or real objects is an example of a binary classification, where the network should distinguish between two classes. With binary classification problems, it is efficient to utilize binary cross-entropy loss (BCEL) [29]. BCEL is defined as

$$\text{Log loss} = \frac{-1}{N}\sum_{i=1}^{N}(y_i log(p_i) + (1 - y_i) log(1 - p_i)) \tag{2.14}$$

where y_i is the ground truth and holds the value one (TP) or zero (FP). p_i is the predicted probability of the object being TP or FP generated by the network, and N is the total number of objects to be classified. This way, if the true label is one, then the second part of the sum is zero and the network is punished for how "far" away it is from classifying the object in question as TP.

Another commonly used loss function is Root Mean Square Error (RMSE) [30]. RMSE is defined as follows

$$\text{RMSE} = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}} \tag{2.15}$$

where n is the number of observations, $\hat{y}_i$ are predicted values and y_i is the observed values. RMSE has a close resemblance to Euclidean distance, and thus it is common to use in regression problems.

2.7 Transformers & Attention mechanism

In order to process sequential datasets recurrent neural networks (RNNs) have been extensively used [31]. They enable sequenced data to be processed in order, with information about previous elements carried over to the next element. RNNs have one significant weakness, which is training time [32]. They take a sequence of data, such as a sentence of words, and pass one element at a time. This allows the network to use information about the previous element in the sequence when processing the current element. As each element has to be processed in order and one at a time there is no possibility of utilizing parallelization, which would decrease training time [33]. To combat this, Vaswani et al. [31] introduced the Transformer network.

Transformer networks are constructed with two major blocks, an encoder and a decoder block, as shown in Figure 2.4. Both blocks are composed of N identical layer stacks. These major blocks are based on one principal idea: the attention mechanism. The attention mechanism consists of mainly three vectors; queries, keys, and values. The attention function is computed on a set of queries packed into a matrix Q. Similarly, the keys and values are packed into matrices K and V. The attention function is then computed by taking the matrix multiplication of Q and K, divided by the square root of d_k, which is the dimension of queries and keys, applying a softmax function (see Equation (2.13)), and matrix multiply with V. This mathematical process is shown in Figure 2.5a and is defined as

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.16}$$

where the inverse square root of d_k is the scaling factor that prevents vanishing gradients.

The Multi-Head Attention consist of h number of attention functions (Equation (2.16)). Furthermore, queries, keys, and values are projected linearly with different, learned linear projections. Thus, the input to the attention function is the previously mentioned Q, K, and V but split evenly over h and are linearly projected. The output of the h attention functions are concatenated and once again linearly projected, resulting in the output of the Multi-Head Attention block. The whole process is shown in Figure 2.5b.

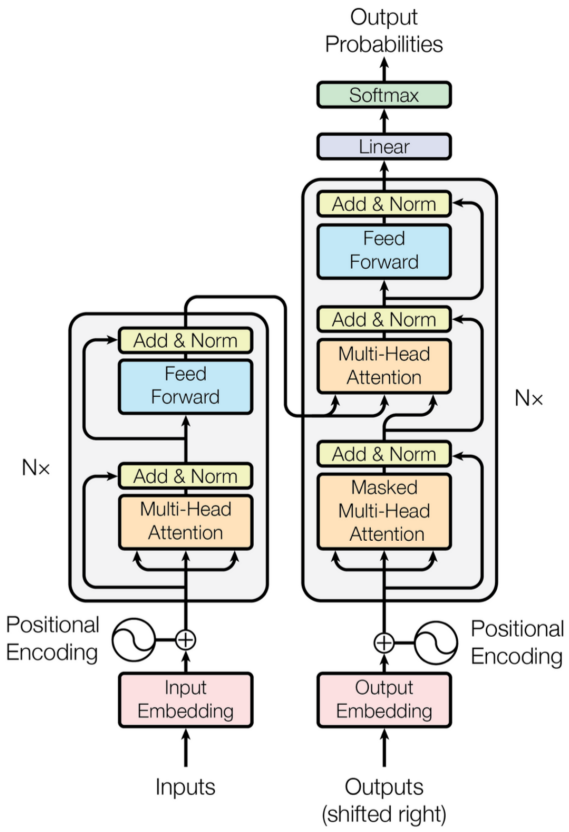
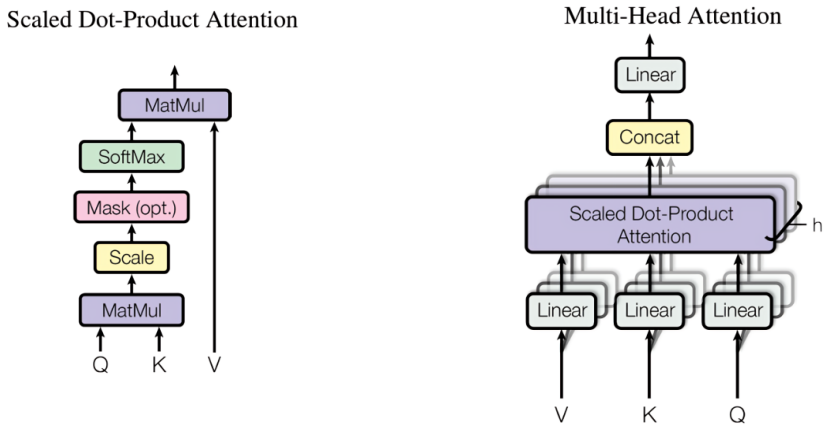


Figure 1: The Transformer - model architecture.

Figure 2.4: Block schematic of the transformer network. Figure taken from [31] under CC 4.0.



(a) Block schematic of the Scaled dot product.

(b) Block schematic of the Multi-Head Attention.

Figure 2.5: Figures taken from [31] under CC 4.0.

The Feed Forward block consists of two fully connected layers with a ReLU activation (see Equation (2.12)) in between. The dimensionality of input and output of the feed-forward network is set to be the same. At the same time, the inner-layer dimension is freely chosen but larger than the input/output layer.

A residual connection [34] and a normalization layer is employed around e.g. Multi-Head Attention and Feed Forward blocks (see Figure 2.4). Lastly, only the key and value in the final encoder stack are passed as an output.

Before the input is sent to the encoder and decoder blocks, a positional encoding is applied to the data. Positional encoding enables the encoder and decoder to process input sequentially correct. Hence, if the position is not encoded when parallelization is applied, the transformer will simply overlook any sequential position information. Thus, positions are embedded utilizing wave functions:

$$PE_{pos,2i} = \sin\left(\frac{\text{pos}}{10000^{2i/d_m}}\right) \tag{2.17}$$

$$PE_{pos,2i+1} = \cos\left(\frac{\text{pos}}{10000^{2i/d_m}}\right) \tag{2.18}$$

here the two functions are used for embedding positions at even and uneven rows in the input matrix. The variable "*pos*" refers to the column of the input, i to the row, and d_m is the total number of rows in the input. Vectors produced from the wave function (2.17) are added to each input column. Hence, giving a unique vector that helps the network to create attention between each element and its position in the input sequence.

A customized learning rate is used over the course of training, according to the formula:

$$l_{rate} = d_m^{-0.5} \min(t_s^{-0.5}, t_s w_s^{-1.5}) \tag{2.19}$$

where t_s is the training step and w_s is a constant predefined "warm-up step". Hence, Equation (2.19) corresponds to increasing the learning rate linearly for the first w_s and decreasing it thereafter proportionally to the inverse square root of t_s.

For more information regarding the decoder block and Transformer method as a whole, the reader can learn more at [31].

2.8 Classification and Regression Trees (CART)

CART is a collection of decision tree algorithms which, in turn are a series of if-else statements bundled together to make decisions based on tabular data [35]. An example of a straightforward binary decision tree is shown in Figure 2.6. Here the goal is to classify an object as a car or a truck based on its width and height. If the object is thinner than 7 meters, it will be classified as a car. If it is wider and if the height is lower than 2.5 meters, it is also classified as a car; otherwise, it is classified as a truck.

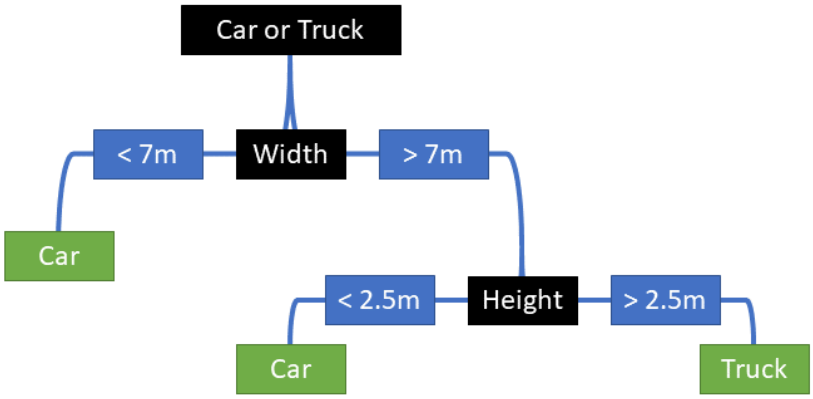


Figure 2.6: Simple example of a decision tree where the goal is to classify an object as car or truck based on width and height.

In the collection of CART algorithms two well-performing versions are Random Forest (RF) and Extreme Gradient Boosting (XGBoost). The concept of Random Forest builds upon a group of trees making individual predictions and picking the prediction with the most votes [36]. RF utilizes the power of groups; thus, if an individual tree or small subgroup can make erroneous predictions that will be countered by the large majority that makes an accurate prediction. To assume that the group prediction is better than a prediction from an individual tree, there should be a low correlation between the individual trees. If there is a high correlation, the trees will most likely make the same mistakes, and they will rather than help each other converge towards the same answer.

In order to mitigate the risk of the forests' correlation becoming too high, two methods can be implemented. The first one is called Bagging or bootstrap aggregation, and it takes advantage of the high volatility of decision trees from variations in training data. Bagging takes a randomly sampled slice of the input data and repeats the process until it has the same shape of input data. This is then done for all the trees in the forest and produces a low correlation forest. The other method is called feature randomness and limits the number of features that each tree can pick from the whole dataset. This further reduce the correlation between the trees, thus improving the final results.

XGBoost applies gradient boost to decision trees to improve model performance and reduce execution time [37]. Boosting is a technique similar to that of Random Forest, but instead of a fixed number of models helping each other achieve good results new ones are added until adding more do not improve the results. In gradient boosting, the new models generated predict the residuals of previous models. The loss function used is gradient descent, hence the name.

2.9 Confusion matrix & performance metrics

Commonly in binary classification problems, a confusion matrix is used to categorize the output from the classifier [38]. This matrix is shown in Figure 2.7 and contains four categories; True-Positive (TP), False-Positive (FP), False-Negative (FN), and True-Negative (TN).

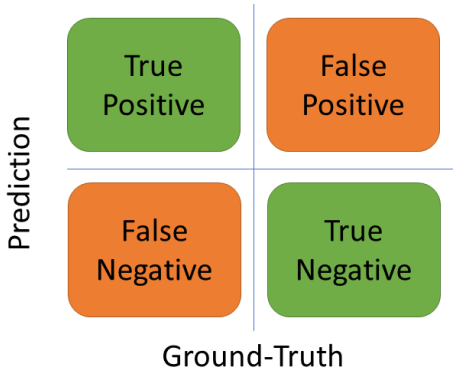


Figure 2.7: Confusion matrix of prediction vs ground-truth. The first diagonal contain those objects/detections which are predicted correctly and on the anti-diagonal are the containers for the faulty predictions.

These categories are filled with classified objects, depending on their ground-truth label and predicted label. The top row with ”Positive” categories contain objects predicted to be real, and the bottom row with ”Negative” categories contain objects predicted to be ghosts. Whether or not these objects are placed in the ”True” or ”False” columns depend on the ground-truth label. Thus, if a real object is passed to the classifier it is placed in either TP or FN depending on if the classifier correctly predict it to be real or not. For ghosts the same logic holds, were, if the classifier predicts that it is a ghost it is placed in the TN category, and in FP if the classification was wrong.

These categories can then be utilized to evaluate the performance of the classifier [39]. Three common metrics are; accuracy, precision, and recall. They all indicate certain aspects of the classifier and how performs in these. Accuracy, define the overall performance of the classifier, and is calculated as

$$\text{accuracy} = \frac{TP + TN}{TP + FP + TN + FN}. \tag{2.20}$$

Precision is defined as

$$\text{Precision} = \frac{TP}{TP + FP}, \tag{2.21}$$

and specifies the ratio of how many real object predictions were correct. Recall is calculated as

$$\text{Recall} = \frac{TP}{TP + FN}, \tag{2.22}$$

and return the ratio of how many real objects the tracker finds over how many it misses.

2.10 Verification of Data-Based Models

There are some inherent pitfalls when utilizing data-based models that can cause significant issues when applied in real scenarios; one of these pitfalls is overfitting. Overfitting is the problem where the model draws the decision boundaries too complex and cannot generalize well to new input. To counteract this, the dataset used is commonly split into three parts; training set, validation set, and test set. The training set is used for training and updating the model. The validation set is then used throughout the training process to verify that the accuracy and loss decrease for data that is not directly used in tuning the model parameters. Finally, when the training process is over, the test set is utilized. Here it is essential that the model cannot go back and be tuned based on the results from the test set. This would defeat the purpose of the test set, which is to introduce the model to previously unseen data.

2.11 Feature importance

An advantage of the CART methods is that they produce a byproduct which makes it is possible to determine feature importance from the results. For XGBoost, the importance score is calculated by utilizing a metric called ”gain”. Gain is defined as the improvement in accuracy generated by a feature. I.e., if the feature is used to split the data at a split point the result will improve after the split. The gain is calculated for all trees in the forest, and an average gain for each feature is generated. The feature with the highest average gain is thus the feature with the most significant importance. Random Forest utilize a method called Gini Impurity (GI) to determine the feature importance. GI-score is calculated as follows

$$\text{GI} = 1 - \sum_{i=1}^{K} P_i^2 \tag{2.23}$$

where K is the total number of classes and P_i is the probability of that class existing in the node. The GI-score ranges between 0 and 0.5, with 0 being the best score indicating a perfect split. Random Forest then calculates the GI-score for all trees and averages them to generate a GI-score for the entire forest. The features that give the lowest GI-score are then ranked as the most important feature.

Lastly, there is a method called "Permutation Importance". It is a model inspection technique and is computed after a model has been trained [40, 41]. The method takes data with N features, shuffles a single feature column, and computes the permutation importance score/prediction accuracy with the trained model. Finally, it returns to the unshuffled data, and repeats the method k times to get an average score. This process is executed for all N features. Therefore, the method is model agnostic, and the feature importance is an average score and thus does not sum up to 100% as the previous two methods do.

3

Data pipeline & analysis

In this chapter, the pipeline from data collection to the ready-to-use dataset is presented; i.e., how was the data collected, parsed, processed, labeled, and structured. Analysis of the dataset is also presented in this chapter.

3.1 Data collection & Sensor specifications

The data used to train and evaluate the networks was collected during the summer of 2020 and in clear weather, ensuring optimal conditions for both sets of sensors. The LiDAR sensor used is a Pandora LiDAR, and the radars are Aptiv proprietary short-range corner radars.

Table 3.1: Properties of the sensors used when collecting the data.

-	Radar	LiDAR
Number of sensors	4	1
Range interval	[0.3m, 100m]	[0.3m, 200m]
Range accuracy	± 15cm	± 2cm at >0.5m
Azimuth range	± 75°	360°
Azimuth resolution	± 1°	0.1°
Vertical range	± 6 °	[+7°, -16°]

3.2 Extracted radar data

The length of the logs selected is just over 2 hours, of which 90 minutes (75%) were assigned to training data, 18 minutes (15%) for validation, and 12 minutes (10%) for testing. The total length is equivalent to approximately 2 250 000 rows of data, with an update frequency of 10 Hz.

From the logs, radar object data was extracted to build the tabular dataset, i.e., a feature per column and each row representing an object. These objects are generated by Aptivs tracker system and are thus categorized into the categories mentioned in Section 2.9. Furthermore, the tracker only provides TP and FP objects as TN and FN lack radar data. The TN and FN objects can be constructed artificially, however this is outside the scope of the thesis.

The objects used contain multiple features, and those utilized are presented in Table 3.2. Here both distances (longitudinal and lateral) and velocity are relative to the ego vehicle, e.g. an oncoming object will have negative velocity. In contrast, "Speed" and "Curvature" are relative to the object's centroid, and a negative speed indicates that the object is reversing on its own track. "Relative time" contains the time since the last observation of the particular object and should be constant; however, due to errors or delays, some intervals are doubled.

Table 3.2: List of all the features used for the classifiers.

Speed	Longitudinal position	Longitudinal position variance
Curvature	Lateral position	Lateral position variance
Number of detections	Longitudinal velocity	Longitudinal velocity variance
Relative time	Lateral velocity	Lateral velocity variance
Bounding box orientation	Longitudinal bounding box dimension	Lateral bounding box dimension

3.3 Data processing approach

Targets are created after the data has been loaded into a Pandas DataFrame and are set to zero for ghost objects and one for real objects. The ratio of ghost and real objects are 64.5% and 35.5% respectively before any preprocessing has been applied. All rows containing at least one NaN value are removed, and relative time is added to the dataset. A new object is defined if either the object ID is different between each scan or the difference between two scan ID:s exceed one scan frame. The relative time is then computed by the time difference between the first and second scan. Moreover, all new objects are initialized with a zero in relative time, and the time is scaled up with a factor of ten to be as close to the interval of $[0, 1]$.

Every variance feature was first converted to the log scale

$$Y = \log_{10}(x + 1) \tag{3.1}$$

where x is an array of data points, Y is the resulting conversion and the addition of 1 is performed to prevent $\log_{10}(0)$; which is undefined. Scaling was then done with percentile (see Section 2.4)

$$\begin{aligned} \mathrm{K} &= \max(|Q(x, P)|, |Q(x, 100 - P)|) \\ y &= \frac{Y}{K} \end{aligned} \tag{3.2}$$

where K is the scaling factor, Q is the percentile function, y' is the scaled data, and P is the percentile; which was set to be the 99th percentile ($P = 99$).

The feature ”Curvature” was defined in radians and therefore converted to gradients and then scaled with Equation (3.2). All following features were all scaled using the percentile method described in Equation (3.2). After feature scaling, Isolation Forest (see Section 2.4.1) was used to remove outliers in the dataset.

For networks that utilized history, e.g. attention networks, a history embedding (see Section 2.5) was added to the whole dataset. The history embedding was initialized to a zero array of size 12 for each new unique object. The deque is then filled in a first-in-last-out manner. Lastly, the dataset is sorted by timestamp and the data structure is converted from Pandas DataFrame to NumPy. After all data processing, approximately 1 600 000 rows of data are left; a reduction of 28.9% from the original raw data. Furthermore, the ratio of ghost and real objects are 56.1% and 43.9%, respectively.

3.4 Data exploration of manually annotated data

Data exploration included manually annotating a small subset of the available data by hand. This was done by estimating whether objects are real or ghosts by observing the Birds-Eye-View (BEV) of objects in a frame, observing the front camera view and drawing conclusions if an object in the BEV could be an actual object or not.

In total, around 4 minutes were annotated, with around 350 objects annotated. The annotated objects were classified into five categories; real, ghost, likely real, likely ghost, and unknown. The first two categories contained only objects that were confirmed visually through camera feed or by deducing scenarios where an object could not exist, such as behind a barrier on a bridge. As the name suggests, unknown objects are unknown and cannot be categorized as any other class. The ”likely” classes are add-ons to the ”Unknown” class and contain objects that cannot be confirmed as ghosts or real through visual examination or common sense but are highly likely in either category due to their behavior. This addition was not added to all objects in the ”Unknown” as the majority of these objects were not possible to deduce what they most likely were.

3.5 Automatic ground truth with LiDAR

As mentioned in Section 2.2, the LiDAR achieves higher resolution than the radar, and thus it produces object classification with higher accuracy. This can be utilized to label the dataset with relatively high-quality labels. In order to generate the automatic ground truth, both a neural network and classical object tracking methods are utilized. Initially, the LiDAR-points are fed to a neural network which creates initial object guesses; these guesses are then sent to a non-causal filter. This filter is responsible for creating consistent object tracks, removing false detections, filling in missing detections, and adding sensor occlusion estimates. The objects that the NN classifies are; cars, trucks/buses, motorbikes, and pedestrians.

In some time frames, there might occur some error or delay in the system which causes detections to disappear or not ”arrive” in time. To reduce the risk of object tracks splitting, the tracker needs to fill in some information about the object even though there might not exist any actual data. The auto-gt algorithm works offline and thus has access to all information, even ”future” data; it can then utilize this ”future” data to extrapolate the probabilistic behavior of the object and generate data where it is missing.

As the sensors are located on different parts of the vehicle, it is critical to consider sensor occlusion. When collecting the data, four radar sensors located at each corner of the car at a height of around 50cm were used, and one rotating LiDAR was fixed on the car's roof.

In Figure 3.1 a simple scenario is depicted where some form of barrier occludes the radar while the LiDAR observes the vehicle behind as its vantage point is higher. Here the radar does not create an object whereas the LiDAR does, thus creating an erroneous ground-truth label. The opposite can also occur with the LiDAR-sensor being occluded and not radar. Both these situations, where one sensor is occluded, creates an imperfect dataset. Thus, directly affecting the classifiers negatively.

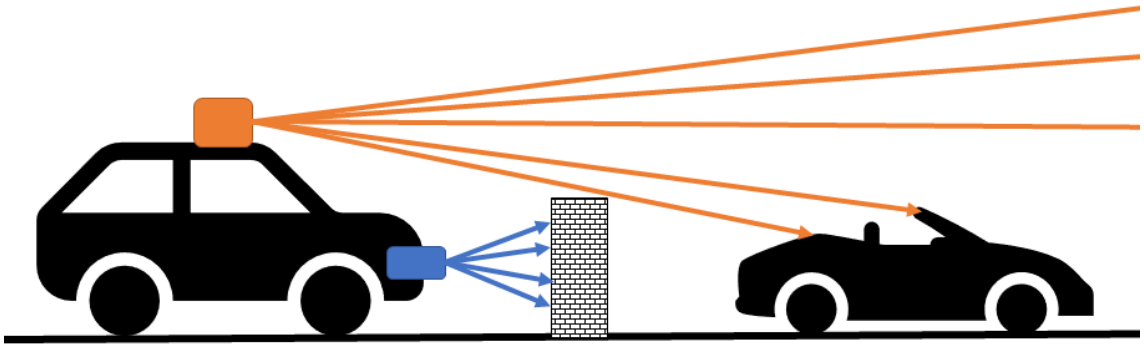


Figure 3.1: Simple illustration of occlusion where the placement of the radar (blue) sensors causes the wall to block its view of the vehicle in front while the LiDAR (orange) still is able to pick it up. Thus, creating an erroneous ground-truth label.

3.6 Statistical analysis of datasets

Here a statistical analysis is presented to ensure quality data is used to train the classifiers. It is shown in both the manually and auto-labeled datasets, there is a skew in regards to the number of ghost versus real objects. This is corrected by taking the lifespan into account which is further explained in the Subsections 3.6.1 and 3.6.2. Furthermore, there is a bias towards highway traffic scenarios which is not corrected for, as it is out of the thesis scope to consider other traffic scenarios.

The analysis consists of distribution analysis, feature importance, and imbalance evaluation. The distribution analysis and feature imbalance evaluation aid in evaluating the quality of the dataset. In cases with highly imbalanced datasets there is a considerable risk that it will be carried over to the results and causing the network to have an inherent bias.

3.6.1 Analysis of manually annotated dataset

The distribution of the manually annotated dataset is shown in Figure 3.2. Here it is clear that most objects were difficult to confidently classify with the information available and the ”unknown” category is hence the largest. The ratio between ghosts and real objects is also interesting to note as there are about three times as many ghost objects as there are real objects, not including the ”likely” classes. However, suppose lifespan (Figure 3.2) is taken into account, and each unique object is multiplied by the number of time instances it is alive. In that case, the dataset distribution approaches 46% real objects, 8% ghosts, and 46% unknown.

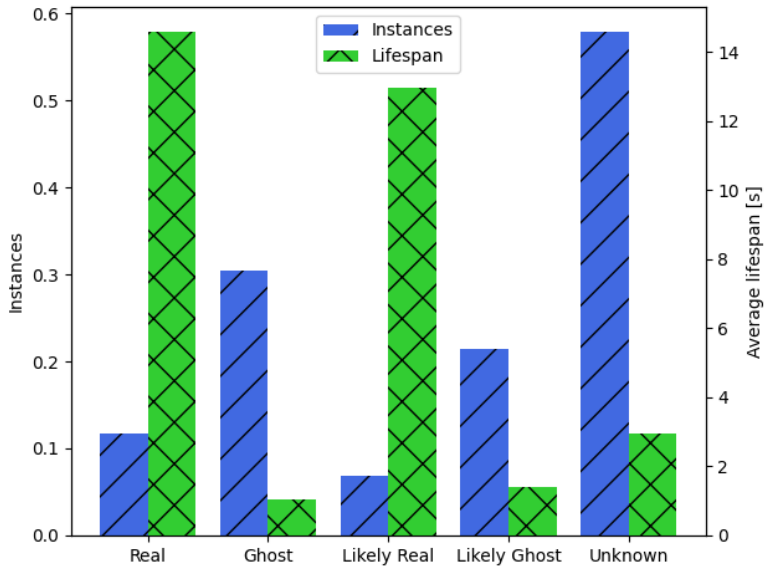
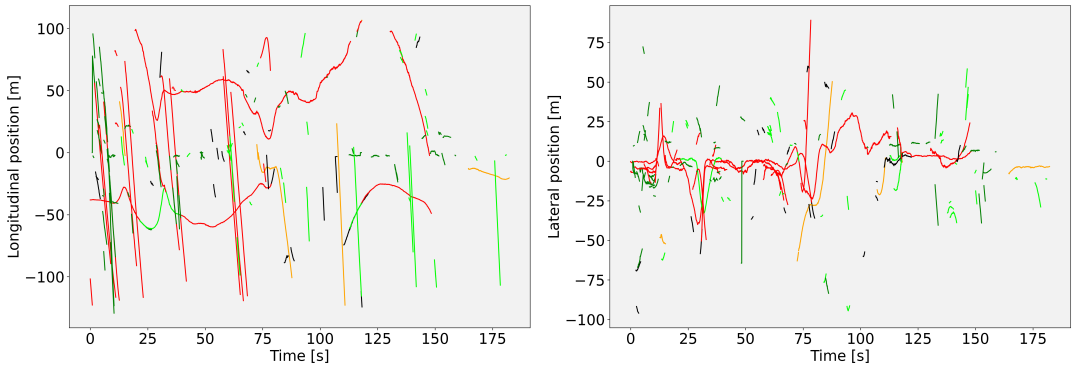
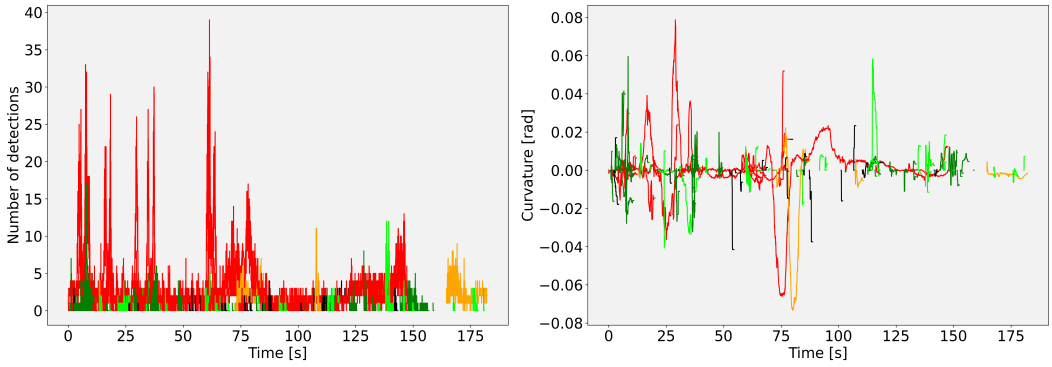


Figure 3.2: Categorical distribution of objects and average lifespan from the manually annotated set. The categories labeled ”likely” are a subset of the category ”unknown”. Note the double axis for distribution and average lifespan.

By observing the manually annotated objects over time, a simple behavior analysis was conducted. An insight into real and ghost objects was gained by looking at the pattern, anomalies in the graph, and the overall behavior. The graphs are presented in Figure 3.3. Figure 3.3a show the longitudinal and lateral position, curvature and number of detections are shown in Figure 3.3b. Here the colors represent the five classes; red: real, orange: likely real, green: ghost, lime green: likely ghost, and black: unknown.



(a) Graphs of longitudinal (left) and lateral position (right) over time. The position is relative to the ego vehicle, hence along the zero position.



(b) Graphs of number of detection (left) and curvature (right) over time. The curvature is relative to the ego vehicle, hence along the zero position.

Figure 3.3: Each line represent an object over time. The colors represent; red:real, orange:likely real, green:ghost, lime green:likely ghost, and black:unknown.

3.6.2 Statistical analysis of Auto-GT dataset

The automatically annotated dataset had around 21 000 false-positive and 7 500 true-positive unique objects. This results in a skewed dataset, with around 74% of the objects being false positive ghost objects. However, when observing the lifetime of the objects, the distribution becomes more evenly distributed. The average lifetime of a ghost object is around 1.8 seconds, whereas real objects live around 3.7 seconds on average. Thus, if the lifetime of the objects is taken into account the distribution is about 58% ghosts and 42% real objects.

In Figure 3.4 the distribution of each category, labeled by the auto-gt algorithm, is shown. The orange striped bars represent the false positive objects, and the green crossed bars represent the true-positive objects. It is clear that most objects in the dataset are cars with two-wheelers and trucks representing only around 10-15% combined and cars representing around 40% of all objects.

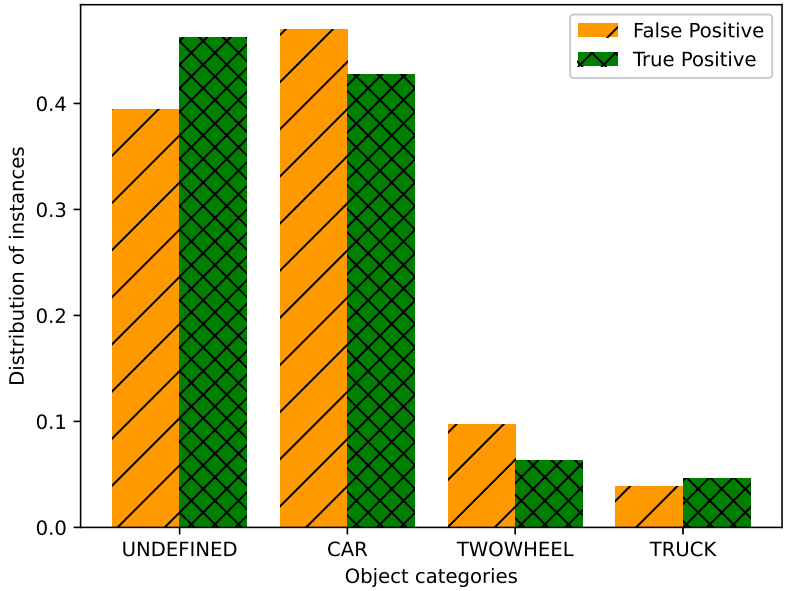


Figure 3.4: Categorical distribution based on LiDAR Auto classification.

3.6.2.1 Distribution analysis of specific features

A subset of the available features from the Auto-GT dataset is shown in Figures 3.6 and 3.5. In Figure 3.5 the distributions of lateral and longitudinal positions are shown. By visual examination, it is clear that there is a greater difference between FP- and TP-objects in lateral position than longitudinal. This is also clear by taking the ratio between the variances in both features. For the lateral position, the variance is around 10 times larger for FP than TP, and for the longitudinal position, the variance is 1.4 larger for FP than TP.

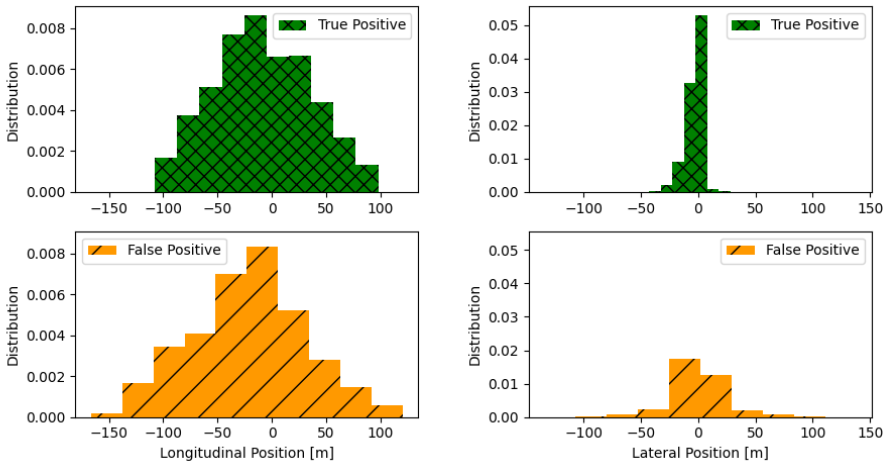


Figure 3.5: Distribution of longitudinal and lateral position for both FP and TP objects.

In Figure 3.6 the distribution of number of detections related to a specific object is shown. The objects with ten or more detections are clustered together into one bin. It is clear that TP-objects have, in general, more detections related to them than FP-objects. Another noteworthy point is that the majority of FP-objects have no detection related to them during their lifetime.

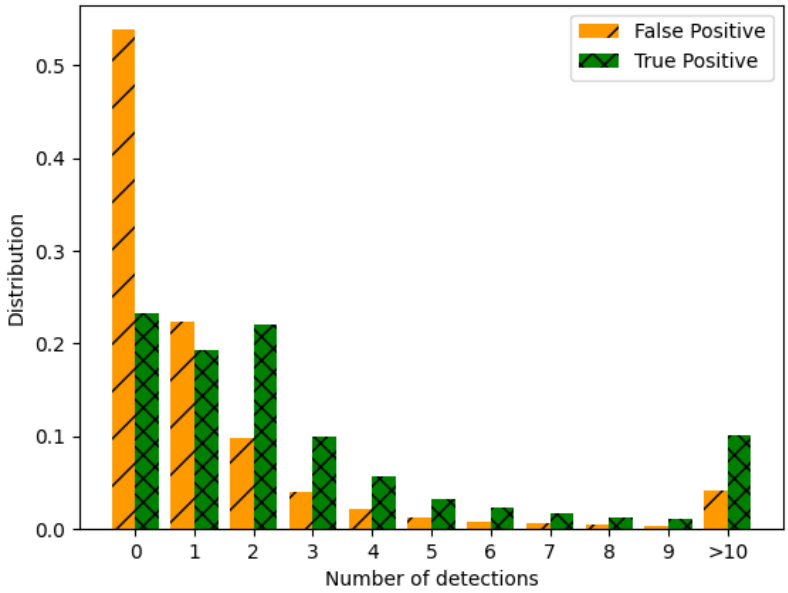


Figure 3.6: Distribution of number detections, related to a specific object, split between TP and FP objects. The objects with 10 or more associated detections are clustered together.

3.6.2.2 Feature importance

When dealing with large feature sets, it is possible to determine which features contain valuable information. These features are ranked through feature importance. To determine the importance of each feature used, the three methods mentioned in Section 2.11; Random Forest, XGBoost, and Permutation are used. The results are presented in Figures 3.7, 3.8, and 3.9. Moreover, the focus is not on the actual score, but rather on the ranking and relative score between the features.

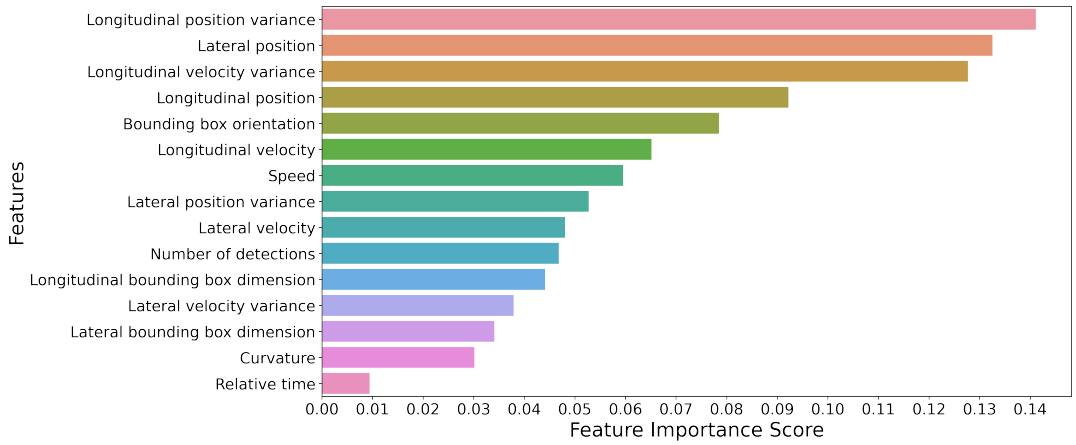


Figure 3.7: Feature importance ranking with random forest.

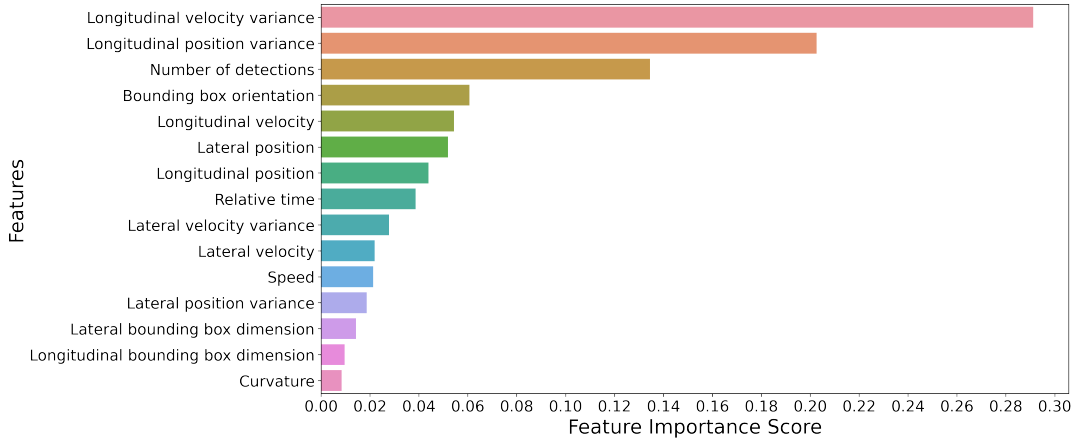


Figure 3.8: Feature importance ranking with XGBoost.

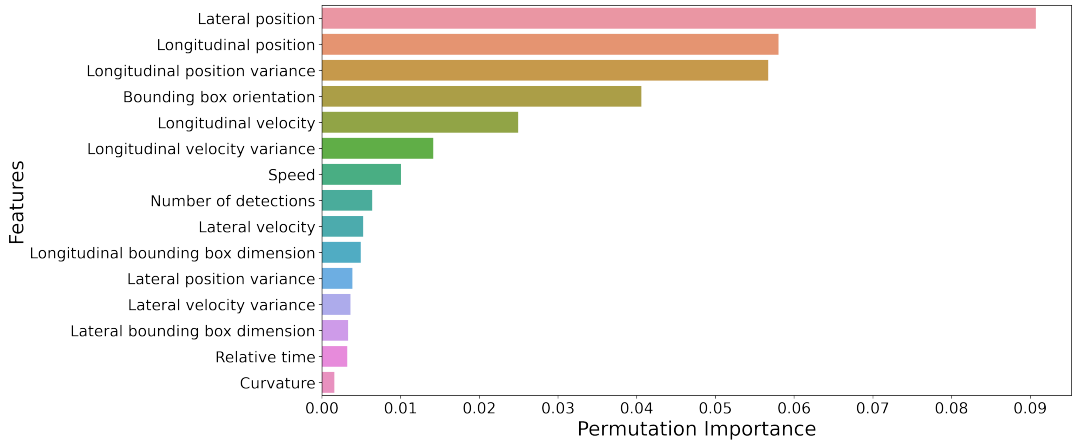


Figure 3.9: Feature importance ranking with permutation.

By assigning a score based on each feature's rank from all methods, all features' general importance can be estimated. The score is assigned two ways, with and without weights. The weights are given by the importance score and is divided by the rank (score = rank/importance) , i.e. for Random Forest the feature, "longitudinal position variance" is calculated as $1/0.14 = 7.14$ and for "relative time" it is $15/0.01 = 1500$. As such, a lower score is interpreted as a better score. A general importance ranking is generated by applying this method to the results from all three feature importance methods and taking the average rank. The results are shown in Table 3.3.

Table 3.3: General importance ranking for all features, with and without weights. Combined rank is the average rank between "with and without weights".

Feature	W weights	W/O weights	Combined rank
Long pos var	1	1	1
Lat position	2	2	2
Orientation BB	3	4	3
Long position	4	4	4
Long vel var	6	3	5
Long vel	5	6	6
# of detections	8	7	7
Speed	7	8	7
Lat vel	9	9	9
Lat pos var	10	10	10
Long BB-dim	11	12	11
Lat vel var	12	11	11
Lat BB-dim	13	14	13
Rel time	14	13	13
Curvature	15	15	15

4

Design & Performance of Classifiers

This chapter covers the performance of all classifiers used. All neural network classifiers were built using Python 3.9.10 and TensorFlow-GPU 2.5.0 with CUDA 11.6. Furthermore, all neural networks trained used a dropout rate of 0.1, output size of 1, ADAM optimizer, and binary cross-entropy as a loss function. While for CART, RMSE is inherently used as a loss function.

Four runs were executed for all classifiers; one with all 15 available features, one with top 13, top 8, and top 6 ranked features from Table 3.3. However, the difference in performance between 6 & 8 and 13 & 15 features was insignificant; thus, only 8 and 15 features are presented. Eight features were selected as it was a threshold at which fewer features worsened the results significantly. A decrease in the amount of features used is to reduce: noise in the dataset, unnecessary computations, and memory size.

Each classifier is then evaluated on the three performance metrics; accuracy, recall, and precision, described in Section 2.9. Here, the four categories, TP, FP, FN, and TN, used to compute respective metrics are defined as in Section 2.9, where

- TP: Real object correctly classified as real
- FP: Ghost object incorrectly classified as real
- FN: Real object incorrectly classified as ghost
- TN: Ghost object correctly classified as ghost

4.1 Logistic Regression Classifier

The most straightforward approach is to solve the problem on hand with a linear solution. Hence, logistic regression was used as it is a linear model with binary targets. The model was constructed with the help of a Python package scikit-learn [41, 42]. Results were identical both with and without applying L2 penalty. Thus, only results for L2 penalty applied are shown in Table 4.1.

4.2 Random Forest Classifiers

Random Forest works well on tabular data, thus it was expected to perform well with the problem at hand. Thus, four Random Forest classifiers with 20, 50, 100, and 150 estimators, respectively, were similarly constructed with the help of scikit-learn [41, 42]. The results for 150 number of estimators is presented in Table 4.1 and the full result table can be found in Appendix A.1, A.2.

4.3 Fully Connected Neural Networks

Fully connected networks can solve non-linearly separable problems, which the logistic regression can not [22]. Thus a simple, fully connected network was constructed to test the hypothesis that a neural network can find the difference between ghosts and real objects. The drawbacks of the network are that it is both computationally and memory expensive, which is why bigger networks only implement fully connected layers at the end of the network and due to their prediction properties. However, their inherent simplicity makes them an ideal starting point. Therefore three, two hidden layers deep, networks with varying sizes were constructed. First one with 32 and 16 layers (*FC32/16*), second one with both layers being 512 neurons (*FC512*) and the third one with both layers being 1024 neurons (*FC1024*). All three FC networks used a batch size of 512, ReLU as an activation function, and a sigmoid for the output layer. Figure 4.1 presents a general model-architecture. The result for *FC1024* can be seen in Table 4.1 and the full result table can be found in Appendix A.1, A.2.

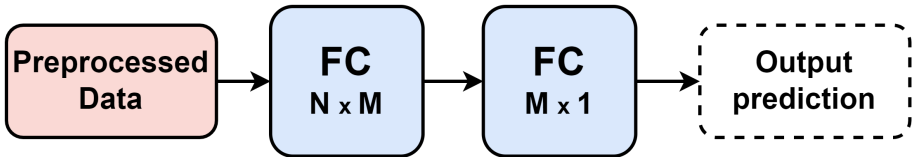


Figure 4.1: Block diagram of a fully connected network with an input layer, two hidden layers, and an output layer. The shapes represent ”number of neurons” x ”number of outgoing connections”.

4.4 Attention based network

Two attention based networks were constructed, one with and one without time embedding Time2Vec (see Section 2.5) integrated. Both networks used a batch size of 512, customized learning rate (see Equation (2.19)), six encoder layers, history size of 12, and sigmoid as the activation function for the output layer. Due to shape constraints within the network, 3 and 4 heads were used for 15 and 8 features, respectively. The results are presented in Table 4.1.

Figure 4.2 showcases the attention based network. The left block (encoder layer) is identical to the Transformer encoder, shown in Figure 2.4 and further explained in Section 2.7. However, the right block is a new "Prediction layer" with Time2Vec. In cases Time2Vec is not used, the concatenation block is omitted along with the following flatten and Time2Vec layer.

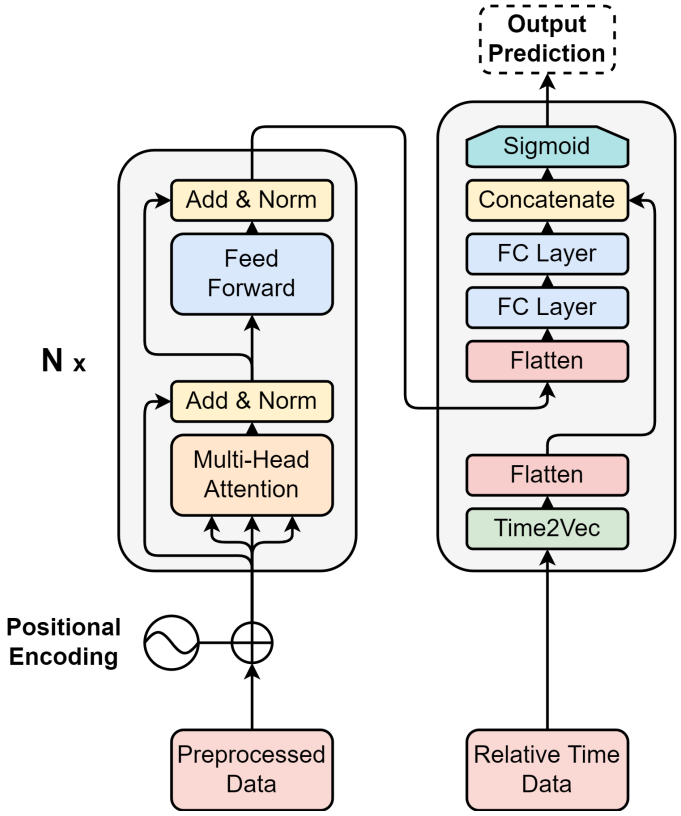


Figure 4.2: An abstract model-architecture over the attention based network with Time2Vec implemented. The left block is the encoder layer, which is identical to the Transformer encoder shown in Figure 2.4 and further discussed in detail in Section 2.7. The right block is a Prediction layer which outputs a prediction $[0, 1]$.

4.5 Model performance

Table 4.1 showcases results on test data for classifiers with 8 and 15 features. The full results are shown in the appendix A.1, A.2.

Table 4.1: Results and performance of the networks with both 8 and 15 features. Top values in each cell show the results from 8 features, and 15 is below. The complete tables can be found in Appendix A.1, A.2.

Network name	Loss	Accuracy	Recall	Precision
FC1024	0.1942 0.1114	0.9195 0.9622	0.9259 0.9656	0.9096 0.9553
Logistic regression	0.5100 0.4967	0.7399 0.7533	0.7694 0.7699	0.6902 0.6988
XGBoost	0.3873 0.3754	0.8500 0.8590	0.8383 0.8423	0.8310 0.8376
Random forest 150	0.2373 0.2023	0.9437 0.9591	0.9361 0.9519	0.9390 0.9547
Attention + T2V	0.3779 0.3550	0.7948 0.8140	0.7448 0.7983	0.7842 0.8085

5

Insights

The chapter contains insights and discussions of the results. A proposal for future work is also given to show where the proposed solutions can be further developed.

5.1 Dataset discussion

An initial insight of the dataset can be seen from both the distribution and behavior analysis results of the manually annotated data. (see Figure 3.2, 3.3). The distribution graph presents that ghost objects, on average, have a substantially lower lifespan than real objects. While looking at the number of instances, ghost objects appear three times more than real objects.

The behavior analysis shows no meaningful pattern which can distinguish real versus ghost objects from each other by a visual or reasonable examination. Figure 3.3 showcases that the problem of distinguishing between real and ghost objects is multi-dimensional and highly complex. However, some noteworthy observations can be seen in the number of detection graphs from Figure 3.3b. Here real objects are more likely to have a higher amount of detections, which is also supported by observing the distribution graph from the Auto-GT dataset in Figure 3.6. Another noteworthy observation is the lateral position graph from Figure 3.3a; where ghost objects with a low lifespan are detected more frequently in the peripherals of the ego vehicle, around ± 25 m. An explanation of this behavior could be due to the multi-path propagation problem explained in Section 2.3.2 and showcased in Figure 1.1. It could also originate from the occlusion problem described in Section 3.5.

5.1.1 Limitations of Auto-GT

Some limitations of utilizing LiDAR to generate the labels are mentioned in Section 3.5, in particular, the problem of occlusion. However, one more limitation originates from the fact that a neural network produces the labels. This NN is not perfect and will likely mislabel some objects even with LiDAR input. These errors are kept when the new network tries to classify only with radar data. As such, a result of 100% for the classifiers proposed in this thesis can only be said to have an accuracy at the same level as the Auto-GT classifier. However, due to the Auto-GT classifier's high accuracy, the results show a great promise that this method is viable to be developed further and be implemented into vehicles in the future.

5.2 Feature analysis

By analysing the results of feature importance in Table 3.3, it is clear that curvature is the least important feature. In the table, both relative time and lateral bounding box dimensions are ranked at the bottom. A possible explanation for these results could be the physical properties of moving vehicles. Regarding the lateral bounding box dimension, it is a reasonable assumption that the bounding box dimension is less important as there exist several vehicle shapes, e.g. Sedan, Motorcycle, Van, Bus, and Truck. This is supported as the longitudinal bounding box dimension is only two ranks ahead but still in the bottom five. Curvature being last could be explained that in highway scenarios, most objects have similar curvature close to zero. It could be argued that feature engineering the relative curvature would give it more value. A change rate could include information such as ghosts acting anomaly by having angular velocity not physically possible. Similarly, relative time is ranked higher than curvature and could be due to the previous explanation even though most of the time the relative time is constant. Furthermore, relative time could be ranked at the bottom due to time, in general, is irrelevant when classifying ghosts and real objects. This explanation is further discussed in Section 5.3.

Focusing on the most important features, one notable difference in rank is between longitudinal and later position variance. An explanation could be derived from Figure 3.5 which shows longitudinal position distribution has a high variance in comparison to lateral position. High variance usually contains more information, which a classifier can utilize to improve prediction.

Continuing, both longitudinal and lateral position is in the top four. Furthermore, bounding box orientation is an interesting feature highly ranked. It could be argued that the reason should be similar to the curvature explanation. However, a logical explanation could be that the dataset only contains highway traffic. Hence, the network has no experience with oddly orientated bounding boxes which could, in a different traffic scenario, be parked cars in a parking lot or cars in an intersection.

Nevertheless, it is not possible to confirm the hypothesis without rigorous testing. Regarding both longitudinal and lateral positions, an explanation for their high rank could be due to both features being extensively used in equations of motion. Another explanation could be due to a pattern of occurrence of a specific set of distances, similarly to what was discussed in Section 5.1, e.g. the network recognizes a pattern caused by multi-path propagation. Moreover, the lateral position being higher ranked than the longitudinal position could be explained with the same logic (e.g. visually compare both graphs in Figure 3.3a). Furthermore, it can also be a consequence of the fact that a common reason ghosts appear is caused by guardrails which are positioned in the lateral axis relative to the ego vehicle (see Figure 1.1).

Important to note in general about feature importance and their ranking is that they are highly dependent on the operating domain. The data used here was mainly highway traffic, and if the operating domain is changed, the feature importance ranking would most likely do so as well.

5.3 Analysis and comparison of classifiers

The ghost object ratio of 56.1% sets the baseline for all classifiers. A value under said ratio indicates a classifier fails to learn correctly and instead is worse than if a model guesses every object is a ghost. No classifier had a value at or under the baseline, indicating all models could successfully extract relevant information from the dataset and learn to differentiate between ghosts and real objects.

As expected, logistic regression performed worst in both cases (8 and 15 features). This could be explained as it is a linear model and has a hard time linearly separating between the two classes (real and ghost) in a multi-dimensional and highly complex problem.

The performance difference between 8 and 15 features is generally low, and one needs to consider if the improvements exceed the increased computation time and memory size. This is especially crucial if the model would be implemented in a real system with major constraints on computational power and memory size. However, both random forest and fully connected networks (except FC32/16) had significant improvements with more features. Between the two network types, random forest performed better with less data; while the fully connected networks outperformed the random forest models with almost half the loss, marginally higher accuracy ($\mp$0.3%), slightly higher recall ($\mp$1.4%), and marginally higher precision ($\mp$0.1%).

Both random forest and fully connected networks are non-linear classifiers, hence an explanation why they performed better than the other classifiers. However, what stands out in the results is that the XGBoost (which is also a CART and hence a non-linear classifier) is performing notably worse than random forest. An explanation of this behavior could not be said nor discussed with an intellectual height without further testing and thoroughly tuning. A hypothesis for the bad performance is that XGBoost has a lot of parameters and customizability, which need a lot of care and thought when chosen.

Another interesting result was the performance of FC32/16 in comparison to the other two fully connected networks. An explanation for the worse performance could be that the number of hidden neurons are too few. Thus the network is not fully able to capture the underlying patterns within the dataset (underfitting) and, in turn have a more challenging time distinguishing between real and ghost objects. Compared to the other two fully connected networks, overfitting and memoization is a risk that contains more neurons. However, on unseen test data, both networks with more neurons perform excellent, showcasing no signs of overfitting. One could argue that the network is overfitted to the type of dataset, highway traffic. Although, the exploration of other traffic scenarios is outside the scope of this thesis.

Lastly, the performance of the complex attention networks further emphasizes the alternated insight of which time is not as relevant as first thought. Instead, the performance worsened when time embedding was implemented with Time2Vec, which indicates noise is added to the network. Furthermore, by comparing against both random forest and fully connected networks, where both outperformed by a signifi-

cant margin, a noteworthy observation is that neither of the two network types uses any history or time embedding. Moreover, both network types were trained and evaluated on a shuffled dataset, hence giving a very interesting insight into how real and ghost objects are identified (time has little to no effect in classification). An explanation could be that in any given instance, a ghost object is already containing features of certain values which characterizes it as a ghost in the network. During the lifespan of the object, said specific values change relative to each other and thus retain the key characteristics which define the object as a ghost; vice versa could be said for real objects. Thus, from one instance to another the object type is not changing which nullifies the importance of time.

5.4 Future work

The results show great promise that the proposed solution can deliver high-quality filtration of ghost objects. It would therefore be advantageous to continue developing the solution. The operational domain of the proposed system is currently limited to highway traffic. It would thus be needed to develop the solution further to be useful in other traffic scenarios, such as inner cities, rural roads, and car parks.

Another possible development area for the future that is highly interesting to look further into is considering the environment in each instance. By embedding the environmental information, a relationship between all objects can be feature engineered, which might increase the classifier's performance.

Due to the scaling limit of the currently best networks. It would be interesting to further look into other more suitable complex networks, perhaps such as graph attention networks, residual attention networks, and echo state networks.

5.4.1 Edge case analysis

Most ghost objects generated are generally of no concern to the ego vehicle as they are often in areas where they do not affect the decision logic. It would therefore be interesting to test the algorithm on specific edge cases where a ghost has affected the decision logic to observe whether or not it improves the performance. An interesting approach could be to train a classifier on a manually annotated dataset with these edge cases specifically in mind. This new classifier could work as an appendage to the system as it is today or to the presented classifiers that are presented in this thesis.

6

Conclusion

A conclusion that the proposed solution is an adequate method within the operating domain of highways can be made. However, the results do have a couple of constraints, one of which is the operating domain. The classifiers are trained using highway data and will likely not achieve the same quality if applied to other environments. Furthermore, the dataset used to train the classifiers was automatically generated with a pre-trained NN to generate labels. This classification is not flawless both regarding misclassifications and physical limitations such as sensor placements. The former can be improved upon with more accurate sensors and software; the latter, however, is difficult to improve as changing the layout of the sensor placements does not necessarily improve the accuracy or performance of the system.

In conclusion, there is a clear advantage of using more features. The increase in overall performance justifies a dataset with higher complexity. Furthermore, it is demonstrated that the time aspect did not have a meaningful impact as initially thought and could be disregarded as a key characteristic of identifying ghosts versus real objects. Moreover, the proposed solution shows that relevant information can be extracted from the dataset to distinguish ghosts from real objects with considerably high accuracy.

Two models stand out from the rest, Random Forest 150 and FC1024. How the two models hold up when further scaling the number of features and at what ceiling before the results worsen; can not be concluded from the results. Furthermore, how the performance will be when more operating domains are included and the two models are retrained; can also not be concluded solely based on this thesis. However, what can be concluded is that both models are among the simpler models compared, especially the latter, which is a subpart of the more complex attention network.

Lastly, more testing on challenging scenarios must be done before implementing the solution in a real system. The proposed solution is intended to be implemented in safety-critical scenarios. Thus it is vital to test the model rigorously with challenging scenarios. It is of the utmost importance to identify in which scenarios it fails to remove actual objects.

Bibliography

[1] Jonah Gamba. *Radar Signal Processing for Autonomous Driving*. 1st. Springer Publishing Company, Incorporated, 2019.

[2] Bin Yang et al. "RadarNet: Exploiting Radar for Robust Perception of Dynamic Objects". In: (July 2020).

[3] LeiChen Wang et al. "Radar Ghost Target Detection via Multimodal Transformers". In: *IEEE Robotics and Automation Letters* 6.4 (Oct. 2021), pp. 7758–7765. ISSN: 2377-3766. DOI: 10.1109/LRA.2021.3100176. URL: https://ieeexplore.ieee.org/abstract/document/9497756.

[4] Florian Kraus et al. "Using Machine Learning to Detect Ghost Images in Automotive Radar". In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, Sept. 2020, pp. 1–7. ISBN: 978-1-7281-4149-7. DOI: 10.1109/ITSC45102.2020.9294631. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9294631.

[5] Mahdi Chamseddine et al. "Ghost Target Detection in 3D Radar Data using Point Cloud based Deep Neural Network". In: *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, Jan. 2021, pp. 10398–10403. ISBN: 978-1-7281-8808-9. DOI: 10.1109/ICPR48806.2021.9413247. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9413247.

[6] Motional. *nuScenes*. URL: https://www.nuscenes.org/nuscenes.

[7] M.I. Skolnik. *Radar Handbook, Second Edition*. 2nd. McGraw-Hill, 1990.

[8] William R. Deal et al. "Microwave Active Circuits and Integrated Antennas". In: *The Electrical Engineering Handbook*. Elsevier, 2005, pp. 691–706. DOI: 10.1016/B978-012170960-0/50048-7.

[9] Hamid Karim and Mats Viberg. "Two decades of array signal processing research". In: *IEEE Signal Processing Magazine* (July 1996).

[10] Bahman Hadji. *Demystifying LiDAR: An In-Depth Guide to the Great Wavelength Debate*. June 2021.

[11] Neuvition Inc. *LiDAR Price for Cars*. Feb. 2022. URL: https://www.neuvition.com/media/blog/lidar-price.html.

[12] Akshay Jawarkar, Akshay Jadhav, and Sonia Mutreja. *Automotive LiDAR Market by Application (Semi-Autonomous Vehicle and Autonomous Vehicle), Technology (Mechanical LiDAR and Solid-state LiDAR), Range (Short- & Mid-range and Long Range), and Vehicle Type (Internal Combustion Engine (ICE) and Electric & Hybrid): Global Opportunity Analysis and Industry Forecast, 2021–2028*. June 2021. URL: https://www.alliedmarketresearch.

com/automotive-lidar-market#:~:text=Using%20LiDAR%20for%20automotive%20applications,approximately%20between%20%2420%2C000%20to%20%2475%2C000.

[13] Marcel Sheeny et al. "RADIATE: A Radar Dataset for Automotive Perception in Bad Weather". In: (Oct. 2020).

[14] N.J. Gordon, D.J. Salmond, and A.F.M. Smith. "Novel approach to nonlinear/non-Gaussian Bayesian state estimation". In: *IEE Proceedings F Radar and Signal Processing* 140.2 (1993), p. 107. ISSN: 0956375X. DOI: 10.1049/ip-f-2.1993.0015.

[15] Simo Sarkka. *Bayesian Filtering and Smoothing*. Cambridge: Cambridge University Press, 2013. ISBN: 9781139344203. DOI: 10.1017/CBO9781139344203.

[16] Lennart Svensson, Yuxuan Xia, and Karl Granström. *Multi ObjectTracking-for Automotive Systems*. 2014. URL: https://www.edx.org/course/multi-object-tracking-for-automotive-systems?index=product&queryID=f26bf24d523a78697b3b328e9f52aab6&position=1&linked_from=autocomplete.

[17] Tristan Visentin, Jurgen Hasch, and Thomas Zwick. "Analysis of multipath and DOA detection using a fully polarimetric automotive radar". In: *2017 European Radar Conference (EURAD)*. IEEE, Oct. 2017, pp. 45–48. ISBN: 978-2-87487-049-1. DOI: 10.23919/EURAD.2017.8249143.

[18] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation Forest". In: *2008 Eighth IEEE International Conference on Data Mining*. IEEE, Dec. 2008, pp. 413–422. ISBN: 978-0-7695-3502-9. DOI: 10.1109/ICDM.2008.17.

[19] Seyed Mehran Kazemi et al. "Time2Vec: Learning a Vector Representation of Time". In: (July 2019).

[20] Python Software Foundation. *collections - Container datatypes*. May 2022. URL: https://docs.python.org/3/library/collections.html#collections.deque.

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

[22] B. Mehlig. *Machine learning with neural networks*. Jan. 2019. DOI: 10.1017/9781108860604. URL: https://arxiv.org/pdf/1901.05639.pdf.

[23] Siddharth Sharma, Simone Sharma, and Anidhya Athaiya. "Activation functions in neural networks". In: *International Journal of Engineering Applied Sciences and Technology* 4.12 (2020), pp. 310–316.

[24] Vitaly Bushaev. "Understanding RMSprop — faster neural network learning". In: *Towards Data Science* (Sept. 2018). URL: https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a.

[25] Sanket Doshi. "Various Optimization Algorithms For Training Neural Network". In: *Towards Data Science* (Jan. 2019). URL: Various%20Optimization%20Algorithms%20For%20Training%20Neural%20Network.

[26] Jason Brownlee. *What is the Difference Between a Parameter and a Hyperparameter?* July 2017. URL: https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/.

[27] Rong Ge et al. "Escaping From Saddle Points — Online Stochastic Gradient for Tensor Decomposition". In: *Proceedings of The 28th Conference on Learning Theory*. Ed. by Peter Grünwald, Elad Hazan, and Satyen Kale. Vol. 40.

Proceedings of Machine Learning Research. Paris, France: PMLR, May 2015, pp. 797–842. URL: https://proceedings.mlr.press/v40/Ge15.html.

[28] Ravindra Parmar. *Common Loss Functions in Machine Learning*. Sept. 2018. URL: https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23.

[29] Daniel Godoy. *Understanding Binary Cross Entropy/Log Loss: a Visual Explanation*. Nov. 2018. URL: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a.

[30] James Moody. "What does RMSE really mean?" In: *Towards Data Science* (Sept. 2019). URL: https://towardsdatascience.com/what-does-rmse-really-mean-806b65f2e48e.

[31] Ashish Vaswani et al. "Attention Is All You Need". In: (June 2017).

[32] Zachary C. Lipton, John Berkowitz, and Charles Elkan. "A Critical Review of Recurrent Neural Networks for Sequence Learning". In: (May 2015).

[33] Eduardo Muñoz. "Attention is all you need: Discovering the Transformer paper". In: *Towards Data Science* (Nov. 2020).

[34] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2016, pp. 770–778. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.90.

[35] Wei-Yin Loh. "Classification and regression trees". In: *WIREs Data Mining and Knowledge Discovery* 1.1 (Jan. 2011), pp. 14–23. ISSN: 1942-4787. DOI: 10.1002/widm.8. URL: https://doi.org/10.1002/widm.8.

[36] Tony Yiu. *Understanding Random Forest*. June 2019.

[37] Jason Brownlee. *Gentle Introduction XGBoost Applied Machine Learning*. Aug. 2016.

[38] Tom Fawcett. "An introduction to ROC analysis". In: *Pattern Recognition Letters* 27.8 (June 2006), pp. 861–874. ISSN: 01678655. DOI: 10.1016/j.patrec.2005.10.010.

[39] Marco Dozza. *Evaluation of Active Safety*. Oct. 2019.

[40] André Altmann et al. "Permutation importance: a corrected feature importance measure". In: *Bioinformatics* 26.10 (May 2010), pp. 1340–1347. ISSN: 1460-2059. DOI: 10.1093/bioinformatics/btq134.

[41] Fabian Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12.85 (2011), pp. 2825–2830. URL: http://jmlr.org/papers/v12/pedregosa11a.html.

[42] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: (Sept. 2013).

A

Full classifier results

Table A.1: Results and performance of the networks with 8 features. Best results are highlighted in bold.

Classifier	Loss	Accuracy	Recall	Precision
FC32/16	0.3534	0.8390	0.8450	0.8403
FC512	0.2145	0.9086	0.9215	0.8983
FC1024	**0.1942**	0.9195	0.9259	0.9096
Logistic regression	0.5100	0.7399	0.7694	0.6902
XGBoost	0.3873	0.8500	0.8383	0.8310
Random forest 20	0.2487	0.9381	0.9227	0.9392
Random forest 50	0.2402	0.9423	0.9331	0.9388
Random forest 100	0.2371	**0.9438**	0.9359	**0.9394**
Random forest 150	0.2373	0.9437	**0.9361**	0.9390
Attention	0.3455	0.8285	0.8148	0.8019
Attention + T2V	0.3779	0.7948	0.7448	0.7842

Table A.2: Results and performance of the networks with 15 features. Best results are highlighted in bold.

Network name	Loss	Accuracy	Recall	Precision
FC32/16	0.3302	0.8525	0.8580	0.8564
FC512	0.1353	0.9498	0.9593	0.9419
FC1024	**0.1114**	**0.9622**	**0.9656**	**0.9553**
Logistic regression	0.4967	0.7533	0.7699	0.6988
XGBoost	0.3754	0.8590	0.8423	0.8376
Random forest 20	0.2152	0.9537	0.9397	0.9542
Random forest 50	0.2071	0.9571	0.9473	0.9546
Random forest 100	0.2037	0.9585	0.9509	0.9544
Random forest 150	0.2023	0.9591	0.9519	0.9547
Attention	0.3279	0.8445	0.8424	0.8137
Attention + T2V	0.3550	0.8140	0.7983	0.8085

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY