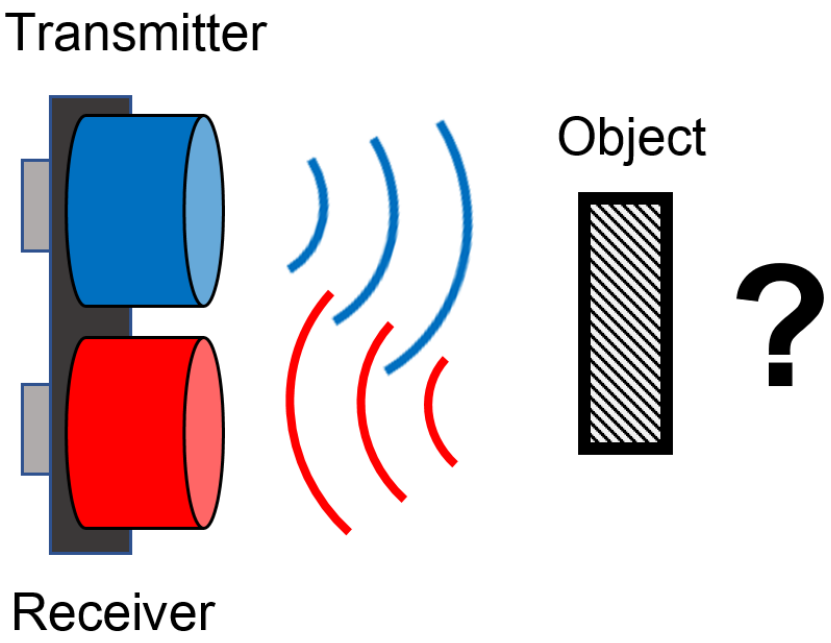
CHALMERS
UNIVERSITY OF TECHNOLOGY



Machine learning classification based on ultrasonic analog data

Evaluation of using raw ultrasonic sensor data for classifying various objects.

Master's thesis in Systems, Control & Mechatronics

RUBEN HWASSER

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

Machine learning classification based on ultrasonic analog data

Evaluation of using raw ultrasonic sensor data for classifying various objects.

RUBEN HWASSER

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Communication, Antennas, and Optical Networks
Chalmers University of Technology
Gothenburg, Sweden 2020

Machine learning classification based on ultrasonic analog data
Evaluation of using raw ultrasonic sensor data for classifying various objects.
RUBEN HWASSER

Supervisors: Jesper Pedersen, Chalmers University of Technology
Andreas Abramsson, Volvo Cars
Examiner: Erik Ström, Chalmers University of Technology

Master's Thesis 2020
Department of Electrical Engineering
Division of Communication, Antennas, and Optical Networks
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization of the basic principal behind using ultrasonics for detecting and potentially classifying an object.

Typeset in LaTeX
Gothenburg, Sweden 2020

Machine learning classification based on ultrasonic analog data
Evaluation of using raw ultrasonic sensor data for classifying various objects.
RUBEN HWASSER
Department of Electrical Engineering
Chalmers University of Technology

Abstract

As the automotive industry gravitates towards electromobility and autonomous drive, there is a huge demand for reliable safety and detection systems to both protect the driver and objects surrounding the vehicle. Current systems utilize ultrasonic sensors to detect such objects and this thesis investigates if it is possible to utilize machine learning algorithms to classify objects using only the raw output from ultrasonic sensors. It also explores various input data structures and evaluates the performances of two different neural network architectures. The first is based on the traditional and proven convolutional neural network (CNN) and the other is based on a recent network called CapsNet. Data is collected within a controlled environment meant to be reproducable if desired. On average, the best performance is shown by CNN and achieves an accuracy of 94% on classification of six different classes.

Keywords: object classification, machine learning, ultrasonics, signal processing, active safety

Acknowledgements

I would like to thank everyone who helped during this thesis and made it possible. Thank you Elpis Arapantonis, first and foremost for giving me the opportunity to work on this thesis topic at Volvo Cars and for taking the time to discuss the topic. Thank you Andreas Abramsson for helping me withe the everyday things in the office and helping me settle in quickly for a smooth start. Thank you Stefan Persson for answering all of my questions and for guiding me in the technical areas of ultrasonics and data collection. Thank you to Patrik Erneman and the rest of the team at Volvo for actively giving feedback and support when needed and for your hospitality. A huge thank you to Jesper Pedersen for your consistent availability throughout the thesis and helping to exchange ideas and to give feedback on my progress. Lastly, thank you Erik Ström for taking the time and responsibility of being the examiner of this thesis.

RUBEN HWASSER, Gothenburg, June 2020

Contents

List of Figures

List of Tables

1

Introduction

Electromobility and autonomous driving are rapidly increasing in popularity within the automotive industry. There is more emphasis on software and autonomous driving functions than ever before. All the software and autonomous driving features that are being integrated into the vehicles today ultimately aim to ease the drivers' experience and provide them with as much information of their surrounding as possible in order to make smart driving decisions.

Records of car accidents involving vehicles developed by Tesla show that their vehicles have been involved in accidents causing 117 total fatalities due to malfunctions in their autonomous driving systems since 2013, of which, 82 the victim was either a pedestrian or a cyclist [1]. As more and more automotive vehicles approach level 4 (human has option to control) and level 5 (full automation) of automation defined by the US National Highway Traffic Safety Administration [2], it is clear that automotive companies must ensure the autonomous drive systems can be trusted and reliable. One way to prevent these kind of accidents from happening would be for the cars to have a reliable classification system for objects such as pedestrians, cyclists, and other objects. A reliable object classification system will help autonomous drive system avoid potentially dangerous events and allow for early detection and avoidance. Today pedestrian classification is done with computer vision or machine learning methods. Fusing sensor information from various sensor types such as radar and LIDAR could allow for more accurate classification predictions.

1.1 Purpose

This thesis aims to explore if raw analog ultrasonic data can be used to train a neural network to perform object classification. Adding additional sources of object classification to a vehicle's active safety systems will help cross-verify classification from other sensors mounted on the vehicle. Ultrasonics are predominately utilized during low speed movement such as parking sequences and are mounted in prime locations to collect data from its immediate surroundings both in the front and rear of the vehicle.

The results from this project will help give insight on the potential for larger scale

research and application.

1.2 Objective

This thesis will investigate the possibility of object classification using only raw ultrasonic data. The main effort of this project will be placed on collecting the raw analog signal output to train two neural networks: one traditional convolutional neural network (CNN) and one based on the CapsNet architecture [22]. The two networks will be compared and evaluated to conclude which one is the most beneficial to classify simple objects. This thesis is done at Volvo Cars in Torslanda, Sweden within the active safety team.

The collected raw analog data requires various pre-processing methods and data reorganization in order to prepare it for input to the neural networks, two of which inputs will be compared as well and are denoted in this paper as: complex baseband and spectrogram representations. Two separate sets of data, T_1 and T_2, are collected to use for testing the performance and robustness of the two networks.

A key issue with using high frequency signal data for training neural networks is the high dimensionality of the input data. High frequency signals contain a large amount of data points per unit time and therefore this project must find a solution to reduce the input size without losing information and will be vital to reducing computation time.

Both CNN and CapsNet require identical training data which allows for direct comparisons to be made between the performance on specific datasets. Both networks are trained on the same complex baseband and spectrogram inputs. The networks will also test their performance on varying input size dependant on the sample size of each ultrasonic measurement, which will give insight on how sample size affects performance. Each input type is also prepared in 3 separate forms known as construction methods: image, matrix, and stacked matrix. Hence, the evaluation is a comparison between CNN and CapsNet and an investigation of how feasible each network is given the various input types and construction methods.

1.3 Scope

This thesis is limited to only perform classifications based on the raw, real-valued ultrasonic signal captured by the receivers. None of the networks are trained to output any information regarding the object's position. Instead, the networks are tasked with solely determining whether the ultrasonic data contains 1 of 6 object classes: noObject, circle, square, rectangle, large rectangle, and triangle.

The evaluation is based solely on data collected during the duration of the project and not any data resourced externally. The evaluation is a proof of concept and not a direct evaluation of the feasibility of a direct implementation of the results from

this study in real world scenarios or vehicle-mounted scenarios.

1.4 Scientific contribution

The main contributions this thesis will give in the area of ultrasonic based object classification using machine learning are:

- A comparison between the CNN-based architecture and the CapsNet inspired architecture and their respective performance based on ultrasonic data input.
- An comparison between the complex baseband and spectrogram representation and how the two different input types affect the networks performance.
- An evaluation of how increasing the number of samples within a measurement affects networks performance.
- An investigation of how filtered ultrasonic data affects the networks performance.

1.5 Outline of thesis

This thesis consists of five main sections.

Section 2: Background acts as a foundation for the reader and provides the reader with the basic technical and theoretical knowledge of the main concepts covered in this thesis. It mainly consists of sections describing the key concepts of ultrasonics and an overview of how signals are commonly processed. Furthermore, it includes an introductory section covering the building blocks of neural networks and an in depth description of to the two network architectures used in this thesis. This section also describes the evaluation metrics used to analyze performance. Lastly, this section includes an overview of related work within ultrasonics and classification.

Section 3: Methods details the procedures used in this thesis. It explains the data collection process used to build the training and test datasets, as well as how the data was used to train and validate the networks. The section also describes the exact network parameters used in this thesis for each network.

Section 4: Results consists of the results produced from the methods section. The results are presented as a comparison between CNN and CapsNet. Subsequently, the results are discussed in detail in **Section 5: Discussion** and the final conclusion of the thesis can be read in **Section 6: Conclusion**.

2

Background

This section explains the theory behind the key concepts utilized in this thesis. It begins by detailing the basics of ultrasonic sensors and standard signal processing methods followed by an introduction to the theory behind neural networks and their architectures. This section also details how the training is done to make sure that the results are portrayed fairly using certain evaluation metrics. Lastly, related work is discussed.

2.1 Ultrasonic sensors

The primary use of ultrasonic transducers or ultrasonic sensors is to detect and determine the distance to an object in close proximity based on how a transmitted sonic wave is reflected back [4]. A basic ultrasonic setup is composed of two components: a transmitter and a receiver, both of which are housed either together or separately depending on the application. A signal that is reflected back to the receiver is registered as a detection. Figures 2.1 and 2.2 illustrated the difference between the transmitter and receiver when housed together or separately. A transmitted signal can cause many echos which makes ultrasonic sensors very sensitive but optimal for close range purposes. These echos/reflections can be caused by both the ground and surrounding objects. In ultrasonics, an object of interest is often denoted as a target. A target can yield several ultrasonic detections as is illustrated in Figure 2.1. An ultrasonic sensor can determine the distance to a detection, and hence also a target, by using the received arrival time of the transmitted signal. The distance, d, to a detection can then be computed by the equation

$$d = \frac{c\Delta t}{2} \ , \tag{2.1}$$

where Δt corresponds to the time it takes for the signal to travel to and from the object, c corresponds to the velocity of waves in the medium, which in this case is the speed of sound in air, 343 m/s [5].

Ultrasonic transducers are also known as piezoelectric transducers which describes how ultrasonic sensors are able to estimate distance using sound waves. Ultrasonic sensors utilize whats known as piezoelectricity. Piezoelectricity is the electric charge that accumulates in certain materials when under mechanical stress. This is also

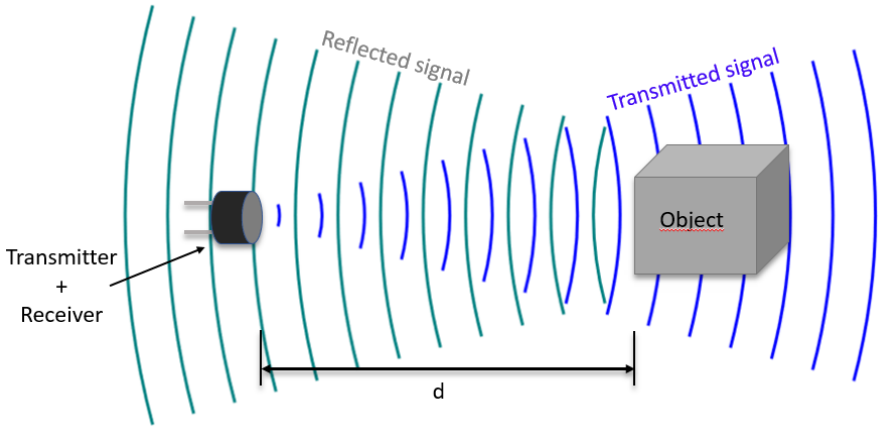


Figure 2.1: A transmitter and receiver housed together allows an ultrasonic signal to be transmitted and received from the same point.

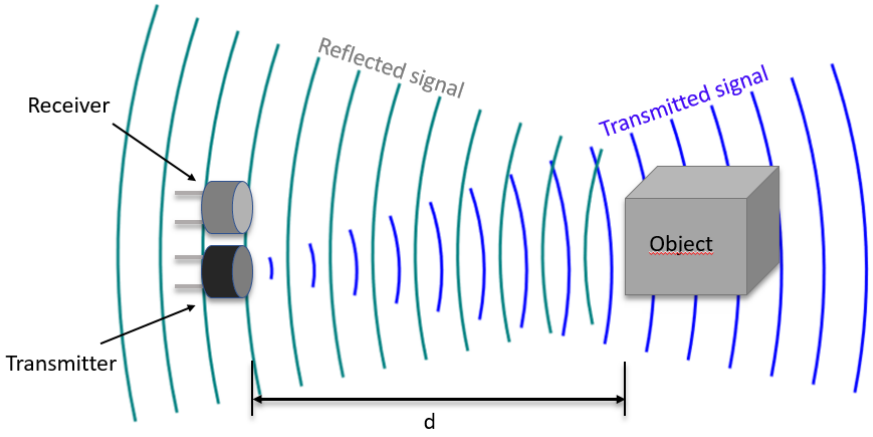
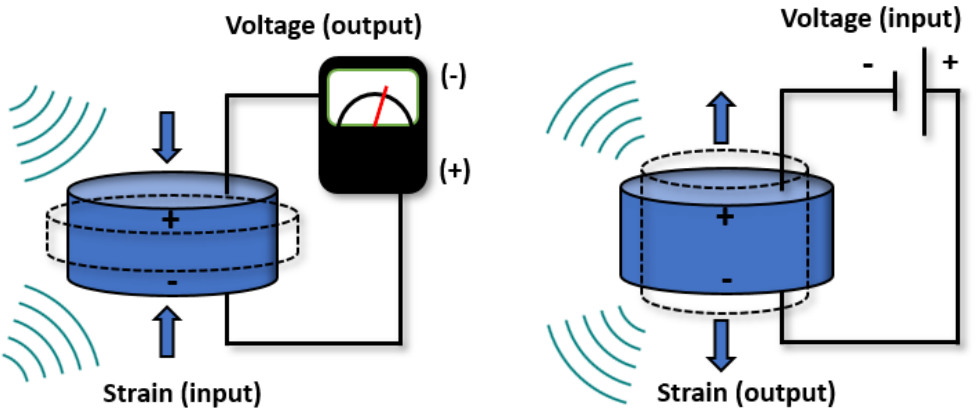


Figure 2.2: A separated transmitter and receiver act in the same way as when they are housed together but allows the receiver to be placed in a completely different point of interest.

referred to as the direct piezoelectric effect. The inverse is also true: applying an electric field generates a deformation. This is known as the converse piezoelectric effect [6].



(a) (b)

Figure 2.3: An illustration of the piezoelectric effect.

Piezoelectricity is found in a variety of electrically insulating materials, including inorganic and organic crystals, ceramics, and polymers, and also in biological materials such as bone and wood. Piezoelectrics find wide application in electrical engineering for use in electromechanical transducers, ultrasonic generators, radio frequency (RF) and microwave filters, sensors, and actuators [6].

2.2 Signal processing

Signal processing plays a vital role in extracting useful and manageable information from sensors. Sensor data can be very noisy and therefore require a series of filtering or reorganization in order to extract the necessary information to accurately perform tasks such as detection, classification, localization, or tracking. There are two main branches of signal processing known as analog processing and digital processing [7][8]. This thesis will focus entirely on digital signal processing. An example of a raw signal before and after digital signal processing is shown in Figure 2.4.

During signal processing, a signal can be visualized in two distinct representations: time domain and frequency domain. Each domain brings with it specific information regarding the measured signal. The time domain is the most traditional representation which shows a signals amplitude over time. The amplitude is expressed as a physical quantity - for example, voltage, sound pressure, etc. The second representation is the frequency domain which also describes the amplitude but represents it over a frequency spectrum instead of time. This allows for detailed analysis of which frequencies compose the signal within any given window of time.

2.2.1 Analog to digital converter

Nearly every environmentally measurable parameter is in analog form, that is, a continuous representation of a time-varying quantity. Theoretically, the analog sig-

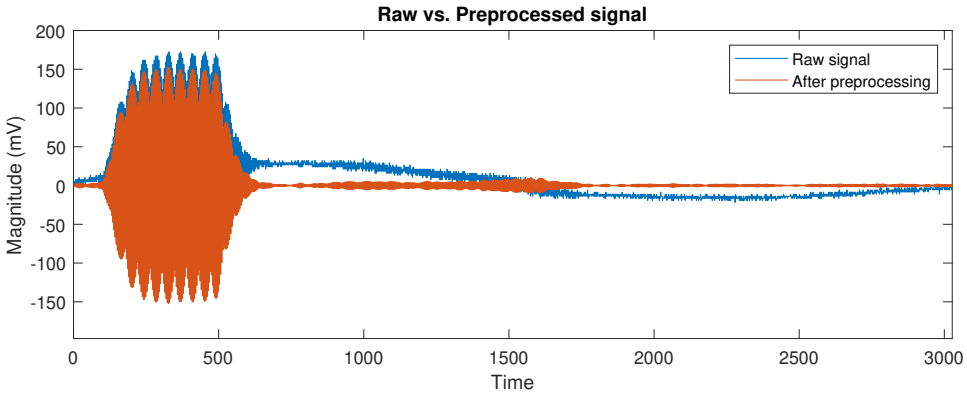


Figure 2.4: An example of the difference between the raw signal and the signal after processing. The raw signal consists of both high and low frequency noise which can be removed through basic signal processing and produce a less noisy signal as shown.

nal could be represented as a continuous time model and used for testing, however, this is not practical since the signals may compose many frequencies some of which may be high or low frequency noise. Therefore, it is easier to sample or measure the signal at a fixed sampling frequency, F_s. If sampled frequently, the analog signal can be represented digitally. This process is handled conveniently by the analog to digital converter (ADC).

The analog to digital converter (ADC) is an electronic integrated circuit used to convert the analog signals such as voltages to digital or binary form consisting of 1s and 0s. Most of the ADCs take a voltage input as 0 to 10V, -5V to +5V, etc. and correspondingly produces digital output as some sort of a binary number. The ADC samples the analog signal on each falling or rising edge of sample clock. In each cycle, the ADC measures and converts it into a digital value. The ADC converts the output data into a series of digital values by approximating the signal with fixed precision [9].

The most important aspects of the ADC is the sampling rate and number of bits per sample. In order to achieve accurate representations of an analog signal and avoid aliasing, the rate at which the ADC is converting a sample to binary must be frequent enough as to not alter the physical properties of the signal. Typically, an ADC begins by filtering the continuous valued analog signal with a analog low-pass filter that will remove any high frequency noise and interference. Following the analog low-pass filter, the ADC will then sample the analog signal based on a chosen sampling rate. This sampling rate is very important because it must be chosen to be high enough as to not distort or destroy the important frequency information in the signal. The Nyquist theorem states that a signal should be sampled with a sampling rate of at least twice the signal bandwidth in order to preserve the quality and information stored within in the signal. The Nyquist rate, which is defined as half the sampling rate, is typically chosen to be the maximum frequency in the input signal, which will ensure the frequencies below it will not suffer from aliasing

or distortion [10]. In some cases, the Nyquist rate is chosen at a higher rate 3-5 times the maximum frequency of the input signal. This would ensure frequencies and information within the signal are not lost due to potentially being affected by the transition band of the analog low-pass filter, for example [11]. Lastly, the ADC quantizes the output from the sampling process based on a specified bits/sample. This quantization maps the sampled signal to a set of discrete finite values. The higher the number of bits per sample, the larger the set of discrete finite values there are, which in turn increases the resolution but will require more powerful hardware to process the data during later digital processing steps.

2.2.2 Fast Fourier transform

The transformation of a time domain signal to the frequency domain can be estimated by an operation known as the Fourier transform. The output from an ADC is discrete therefore it is necessary to use the discrete time Fourier transform (DTFT). Since it is nearly impossible to determine the exact DTFT of a discrete signal $x(n)$, it is more practical to analyze a finite number of samples and then interpret the results in comparison with the DTFT. This is known as the discrete Fourier transform (DFT). The DFT is defined as,

$$X(k) = \mathrm{DFT}\left[x(k)\right] \triangleq \sum_{n=0}^{N-1} x(n)\mathrm{e}^{\frac{-j2\pi kn}{N}} \tag{2.2}$$

where N is the number of samples [12]. Note that $X(k)$ is N-periodic, i.e. $X(k + N) = X(k)$. In practice, a computationally efficient way of calculating the DFT is to use the Fast Fourier Transform (FFT) which is a computational scheme which requires less operations and can be utilized in programs such as Matlab. A detailed description of the operations made in FFT can be found in [12]. The DTFT is defined as,

$$X(\omega) = \mathrm{DTFT}\left[x(n)\right] \triangleq \sum_{n=-\infty}^{\infty} x(n)\mathrm{e}^{-j\omega n\Delta t} \tag{2.3}$$

where ω has unit radians per second [12].

2.2.3 Passband & baseband signals

In general, communication systems can be classified into two main groups depending on the range of frequencies they encapsulate. These groups are known as baseband and passband systems. Baseband signals are as is without modulation while passband signals are shifted to be transmitted at a higher frequency and are subsequently demodulated on the receiving side [13]. An example of the difference between baseband and passband signals can be seen in Figure 2.5.

Nearly all sources of information generate baseband signals which include any signals that have frequencies close to zero such as the human voice which is bandlimited to 20 kHz and video signals from a television which are bandlimited to 5.5 MHz [13].

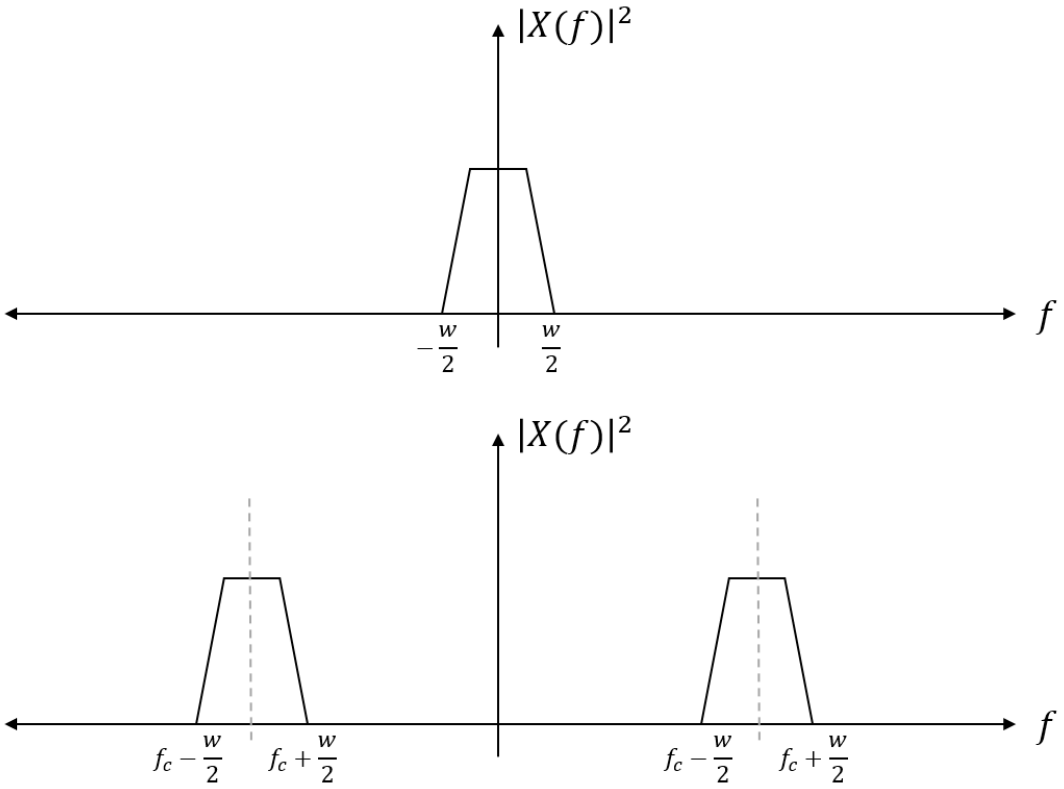


Figure 2.5: The upper graph illustrates a baseband signal centered around 0 kHz. The lower graph illustrates a passband signal operating around its center frequency, f_c.

2.2.4 FIR filter design

A common step in digital signal processing is to filter a sampled signal to remove unwanted noise or interference to yield a cleaner and more desirable signal for testing. Filter design is a vast subject and depends heavily on the intended application and constraints such as performance specifications and other constraints such as complexity in terms of memory or computational speed.

A finite impulse response (FIR) filter is defined by an impulse response of a finite duration and can easily designed to have linear phase. The output of the filter can be described as the result from a convolution between the input signal and the filter impulse response

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k) \ , \tag{2.4}$$

where the impulse response of the filter is $h(k)$ and the coefficients of the filter are defined by $k = 0, ..., M-1$ [12]. M denotes the length of the filter, i.e. the length of the impulse response. The frequency function is given by the DTFT as

$$H(\omega) = \sum_{n=0}^{M-1} h(n)\mathrm{e}^{-j\omega n} \tag{2.5}$$

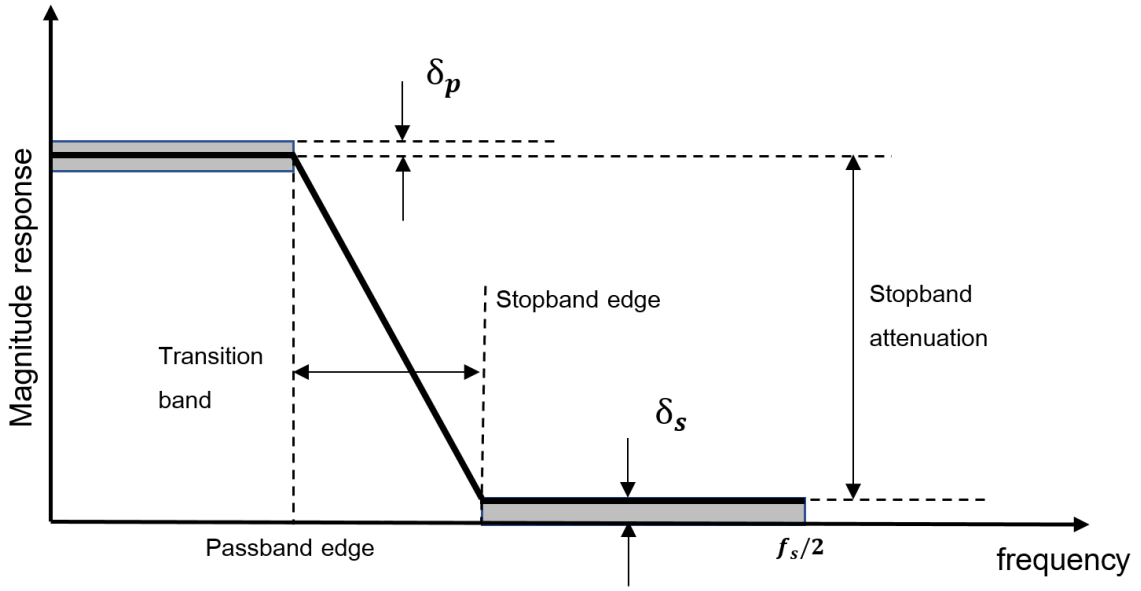


Figure 2.6: Basic properties of a FIR filter design where δ is the ripple within each band and f_s is the sampling frequency.

The two main categories considered within FIR filter design are frequency selective filters and time domain filters. The type used in this project is frequency selective filters. The objective of frequency selective filters is to enhance or suppress certain bands of the frequency spectrum in a given signal. Common types of frequency selective filters are

- Low-pass filter (LP)
- High-pass filter (HP)
- Band-bass filter (BP)
- Band-stop filter (BS)

Frequency selective filters are distinguished by these types based on their amplitude function. The amplitude function can be described by 3 regions: the pass-band region where the amplitude is close to the amplification A, a stop-band region where the amplitude function is small to attenuate well, and finally a transition region where the amplitude function moves from the pass-band to the stop-band [12]. These regions can also be seen in Figure 2.6. There are several window-based types of FIR filters used for real-valued signals. A common and widely used window-based FIR filter is known as Hamming. The Hamming filter falls under the category of cosine-sum windows whose general function is described as

$$w[n] = \mathrm{a}_o - (1 - \mathrm{a}_o) \cdot \cos\left(\frac{2\pi n}{M}\right), \quad 0 \leq n \leq M \tag{2.6}$$

Hamming windows are defined with $\mathrm{a}_o = 0.54$ which puts the end points of the

window at values slightly above zero amplitude [14]. The Hamming window is a widely used filter design due to its simplicity and for this application it is sufficient to use such a filter.

2.2.5 Multirate digital signal processing

In certain cases of digital signal processing, it could be advantageous to change the sampling rate of a signal. In this case, there are two options: up-sampling and down-sampling. These processes are known as multirate signal processing and are also known as expansion and decimation, respectively [15].

Up-sampling refers to increasing the sampling frequency of a signal, for example, by inserting zeros between the original signal values. This technique is popularly used in audio CDs, where the sampling frequency 44.1 kHz is increased to 176.4 kHz before digital to analog conversion in order to provide higher resolution and yield less distortions introduced by editing [15].

Down-sampling refers to decreasing the sampling frequency of a signal by extracting only a subset of the original signal values. This has a certain drawback in that some information may be lost in the process if the new sampling rate fails to satisfy the Nyquist theorem. Failing to uphold the Nyquist theorem will produce moderate to severe loss in information and aliasing.

2.2.6 Spectrogram

One way of representing a signal is graphing the frequency spectrum over time. This representation is known as a spectrogram. In practice, a spectrogram is represented by 2 dimensions. The first dimension is the x-axis which depicts time. The second dimension is the y-axis which depicts frequency. The magnitude of a signal at a given time and frequency is represented in a color spectrum. An example is shown in Figure 2.7 where the magnitude is represented by the yellow-blue color scale.

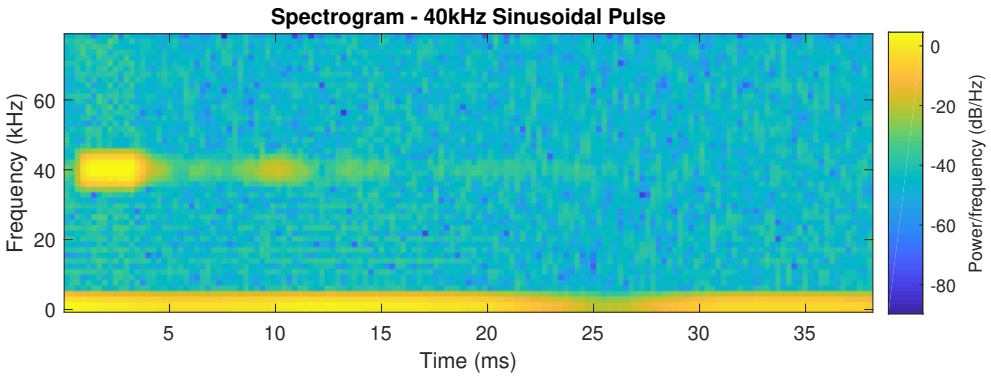


Figure 2.7: An example of a 40 kHz burst is illustrated within this spectrogram showing its frequency behavior over time, as well as frequency magnitude.

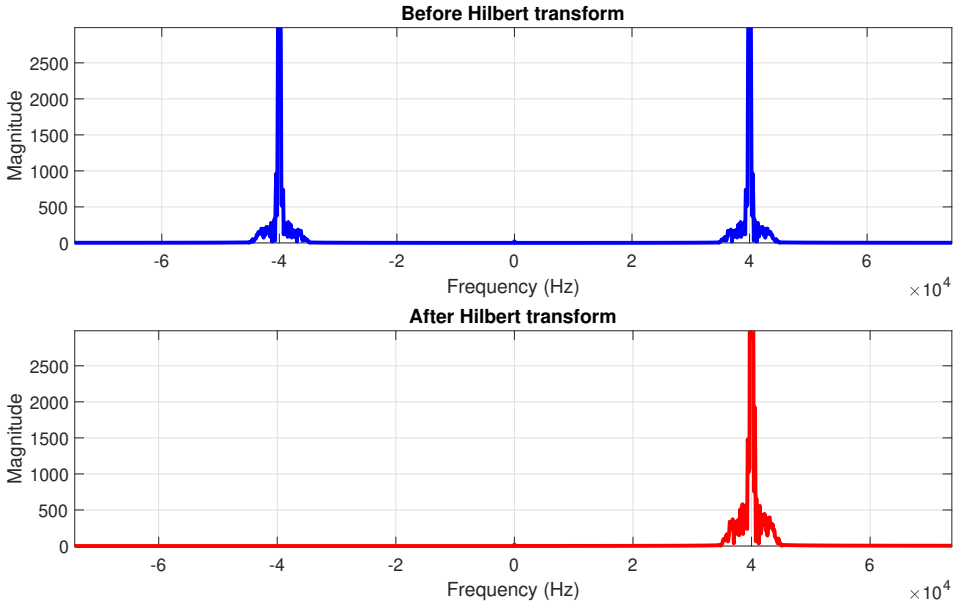


Figure 2.8: The top of the figure shows the frequency domain of an example real-valued signal. The bottom half shows the analytic signal through the process of the Hilbert transform. The resulting frequency domain of the analytic signal removes negative frequencies and doubles the magnitude of the positive frequencies.

2.2.7 Analytic signal

The Hilbert transform is a method of generating the phase portion of a real-valued signal. This can be used to yield the analytic signal which is a complex-valued representation of the real-valued signal. The analytic signal is described by

$$\hat{\boldsymbol{x}}(n) = \boldsymbol{x}(n) + j\boldsymbol{h}(n) \tag{2.7}$$

where $\mathbf{x}(n)$ represents the real valued signal and $\boldsymbol{h}(n)$ represents the Hilbert transform of $\mathbf{x}(n)$. The Hilbert transform essentially creates an imaginary part by forcing a -90 degree phase shift onto the real-valued part. In the frequency domain, the analytic signal contains only positive frequencies creating a single sided frequency domain. Figure 2.8 illustrates an example measurement showing the frequency domain of the real-valued signal before and after conversion to an analytic signal and the Hilbert transform. More can be read about the details of the Hilbert transform and the analytic signal representation in [17]. The analytic signal allows for more mathematical manipulations of the signal without loss of information because it does not alter the real-valued signal but rather adds an imaginary part [16].

In many cases the analytical representation of the passband signal is shifted or translated close to 0 Hz. This representation is referred to as the complex envelope.

2.3 Neural networks

Neural networks (NNs) are the basis for most of the machine learning algorithms. The name is often referred to as being inspired by the neurons in the human brain. NNs play a key role in many applications, from image detection to automatic language translation. Deep neural networks (DNNs) build upon NNs and are based on the fully connected layer. The fully connected layer contains neurons of which are not directly connected to each other but rather connected to neurons in the subsequent layer. A neuron in a neural network simply takes inputs from a previous layer and computes a new value, of which is passed on to the neurons in the next layer. The goal of NNs and DNNs is to approximate the true model $f^*(\boldsymbol{x})$, by defining the network model $g(\boldsymbol{x})$, based on the input $\boldsymbol{x}$. An illustration of a simple network can be seen in Figure 2.9.

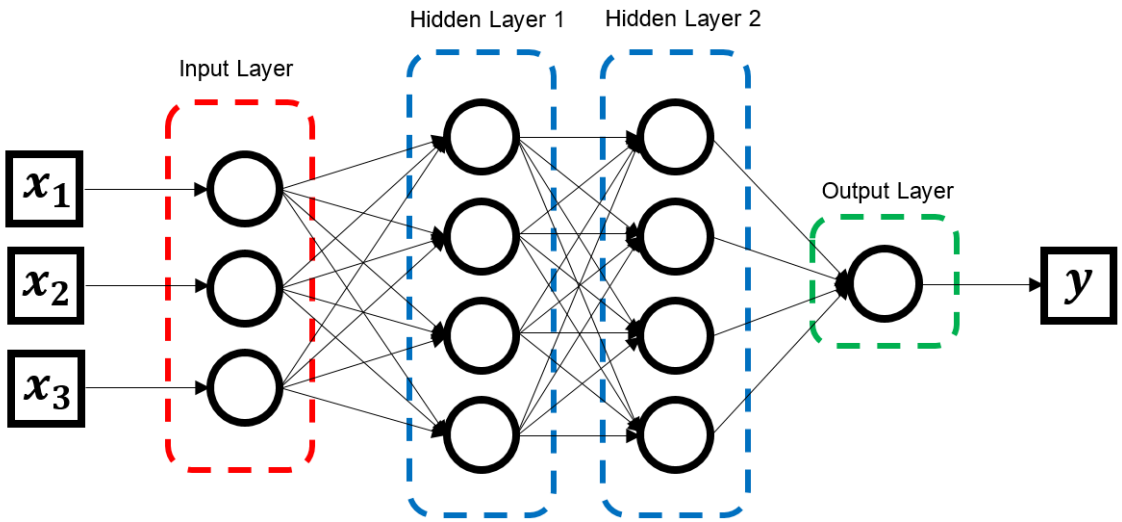


Figure 2.9: A fully connected neural network with an input layer with 3 inputs, 2 hidden layers, and 1 output.

2.3.1 Activation function

The activation function, ξ, converts the output values of a neuron to a more suitable value for input to the next layer of neurons. The activation function can either be linear or non-linear. In many applications, non-linear activation functions are used in order to solve more nontrivial problems using only a small number of neurons. The most common non-linear activation functions are the sigmoid function, $\sigma(\boldsymbol{x})$,

$$\sigma(\boldsymbol{x}) = \frac{1}{1 + \mathrm{e}^{-x}} , \tag{2.8}$$

and the rectified linear unit function,

$$\mathrm{ReLU(x)} = \max(0, x) . \tag{2.9}$$

When dealing with a classification problem, activation functions are important in achieving good prediction results. However, more importantly, the final layer of a classification network must be followed by a non-linear activation function such as the softmax function,

$$\text{Softmax}(x_i) = \frac{\mathrm{e}^{x_i}}{\sum_j \mathrm{e}^{x_j}} \,, \tag{2.10}$$

or sigmoid function (see (2.8)) in order to yield a number between 0 and 1, i.e. a probabilistic output which is desirable for classification problems. A probabilistic output yields a vector with probabilities of the input being a certain class. Softmax is most useful when there are multiple classes but there is only one right answer since softmax produces an output vector that sums to 1 of which the largest value is the predicted class. Sigmoid is necessary for multi-label classification where there may be multiple classes present within the same input image, therefore an output vector that sums to 1 is not ideal. For example, in an image classification problem, an image may contain multiple different classes, therefore the output vector should consist of the probabilities of each class being present in the image, i.e. each vector position is a value between 0 and 1.

2.3.2 Forwardpropogation, backpropogation and loss

Since the goal of a neural network is to accurately estimate the true model $f^*(\boldsymbol{x})$ then the network needs a tool to effectively set its parameters to values that closely mimic the true model. Forward- and backpropogation are processes that allows neural networks to learn and assess how much tuning a specific parameter affects the end result. Each neuron has learnable weights, w_i, and a learnable bias term, b. The input to each neuron, x_i, is multiplied with the corresponding weight, w_i. The sum of all these products for each neuron is then added with the bias, b. This process is applied to every input to the neuron and then summed together. As discussed previously, this summation is passed through an activation function, ξ, that introduces some non-linearity and provides more suitable values for input to the next layer of neurons. The output from each neuron can then be expressed as $y = \xi(w^\top \boldsymbol{x} + b)$. If there are more than one layer in the network, the output y becomes the new input, $\boldsymbol{x}$, to the next layer in the network and the process continues. This is known as forward propagation - the forward pass of the input data through the network. After forward propagation, the network produces an output y. A loss function is defined to give an estimate of how well the network has performed. An example of a loss function is the mean square error (MSE)

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 \,, \tag{2.11}$$

where $\hat{y}_i$ is the target variable, y_i is the predicted variable, and N is the number of samples being predicted. There are several other loss functions one can use to estimate the performance of a network. However, MSE is a widely used loss function that penalizes outliers and is therefore good for input data where the feature values are similar and close to normally distributed. This is the case because the difference between the true and predicted values is squared which causes larger errors to yield

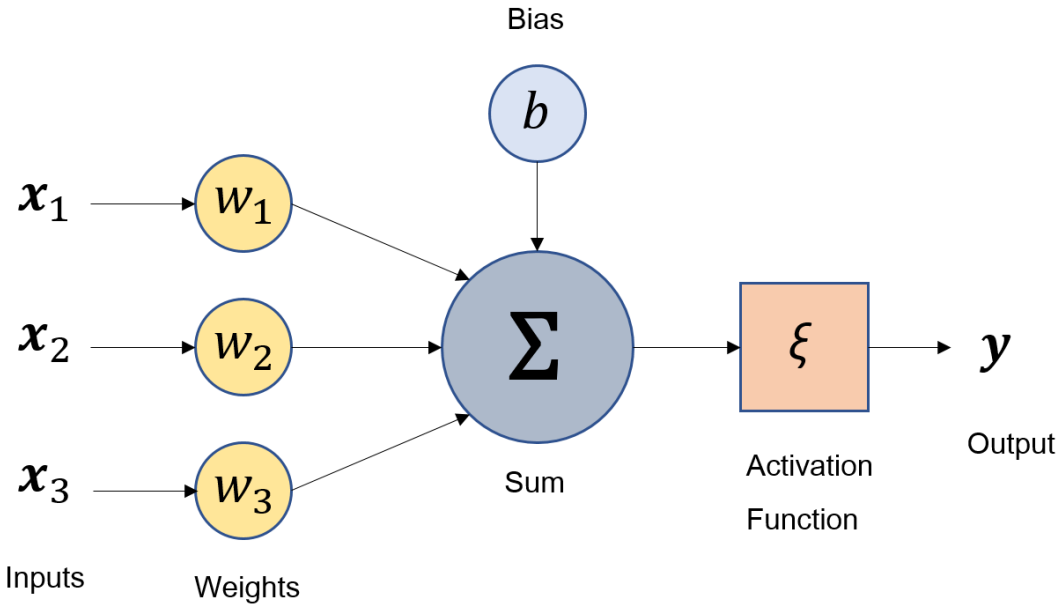


Figure 2.10: Computational flow of a neuron.

significantly higher loss, but if the input data is not normally distributed and has lots of outliers this loss function will gives lots of high loss values. Various other loss functions are tailored for specific problems, such as multi-class classification. In this case, the categorical cross entropy loss function,

$$L(y, \hat{y}) = -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij} \times \log(\hat{y}_{ij})) \ , \tag{2.12}$$

is much more advantageous, where $\hat{y}$ is the true one-hot encoded vector, y is the predicted probability vector, and M is the number of classes. In classification problems it is typical to label the training data with a one-hot encoded vector that has the same length as the number of classes and places a value of 1 at the vector position of the correct class and 0 at the others. Therefore, it is common to use the categorical cross entropy loss function which will compare this true one-hot encoded vector with the predicted probability distribution vector from the network.

In order to update the values of the learnable parameters in the network, denoted by Γ, backpropagation, which is the most common algorithm for training networks, is done. Backpropagation refers to the process of computing the gradient of the loss function with respect to the weights, or parameters, $\nabla_\Gamma L(\Gamma)$ of the network. In other words, the weights and biases will be updated to produce a lower loss in the next forward propagation.

Dividing the dataset into batches can further improve performance of training a network by grouping training data together in different batches. Each data point in the batch is inputted to the network one after the other where backpropagation

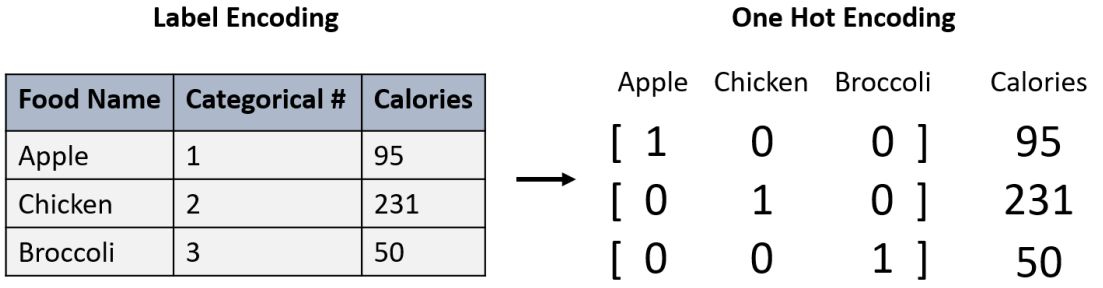


Figure 2.11: Example of one hot encoding for classification of three food types based on calories.

is only done at the end of each batch. Larger batch sizes allows for faster training but may not result in better model accuracy. Smaller batch sizes are preferred but slower. Smaller batch sizes create models that are more generalized and better at predicting test data outside the training set. The effects of large and small batch sizes was studied in [18] and confirmed that larger batch sizes are less able to generalize compared to smaller batch sizes.

2.3.3 Optimization algorithm

The loss function determines a difference in the true and predicted values of a network but does not explicitly determine how the weights should be updated. Various optimization algorithms are used to calculate how the parameters of the solution, Γ, will be updated. The majority of optimization algorithms take a step of length α into the direction of the gradient determined by the type of optimization algorithm, referred to as gradient descent. A very popular and proven algorithm is adaptive moment estimation optimizer (Adam) which utilizes parts of two other popular optimizers RMSprop and Adagrad [19]. From Adagrad, Adam takes the idea of momentum which updates the solution based on previous gradient direction in order preserve trajectory. From RMSprop, Adam utilizes the method of taking smaller steps in steep gradient descents and large steps in shallower descents in order to avoid overstepping and stepping to0 slowly. Adam is widely used because of its computational efficiency and is suitable for larger training datasets. The algorthim can be described as follows

2.3.4 Convolutional filter

A convolutional neural network (CNN) is a unique type of neural network that utilizes convolutional filters to extract features or patterns in an array of input values. Convolutional filters are proven to be extremely efficient for image recognition and classification. A 2D convolutional filter at its core can be described by its kernel size. The kernel size determines how large the filter is. In Figure 2.12, a 2×2 convolutional filter is applied to the left image. Convolutional filters can be different sizes and always consist of learnable weights. The convolutional filter is slid across the input with a step size, also known as the stride, which is typically set

Algorithm 1: The adaptive moment estimation, *Adam*, algorithm is generally initialized with the following parameters: $\alpha = 0.001$, $\epsilon = 10^{-8}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. [19]

Input : α: Stepsize
Input : β_1, $\beta_2 \in [0, 1)$: Exponential decay rates for moment estimates
Input : $f(\Gamma)$: Stochastic objective function with parameter Γ
Input : Γ_0: Initial parameter vector
Output: Γ_t: Resulting parameters
$m_o \leftarrow 0$ (Initialize 1st moment vector)
$v_o \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while Γ_t *not converged* **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\Gamma f_t(\Gamma_t - 1)$ (Get gradients w.r.t. stochastic objective at t)
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
  $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
  $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
  $\Gamma_t \leftarrow \Gamma_{t-1} - \alpha \cdot \hat{m}_t / \sqrt{\hat{v}_t} + \epsilon$ (Update parameters)
end

to 1. Each time the filter moves across the image, a dot product is performed on the values covered by the filter. The output of the filter is known as an activation map. Padding is a feature of convolutional filters that determines if the output will retain the input dimension size and how the filter will handle the edges of the input. For example, Figure 2.13 shows whats known as "same" padding which conserves the input dimension size in the output by zero-padding the input image to allow convolution to work properly at the edges of the input. In realistic applications of CNN, the input data is RGB images which contain a third dimension containing three color channels. In this case, 3D convolutional filters are used. The same principal as 2D filters is used but is now applied in 3D. Figure 2.14 illustrates how a 3D convolution on a 3-channel input may look. Typically, a layer in a CNN is composed of several convolutional filters resulting in several activation maps. The output of a CNN layer with multiple convolutional filters will have a depth corresponding to the number of filters used. The output in Figure 2.14 represents the result of four convolutional filters being applied.

2.3.5 Overfitting

The goal of designing neural networks is to create a trained model based on a subset of data that in theory can perform well on test data unseen by the network. This idea is known as generalization. The opposite of generalization is overfitting. Overfitting the model to the training data does not achieve good results on test data and essentially means the model is not learning to generalize but rather explicitly fit the model to the training data. A common solution to avoid overfitting is to

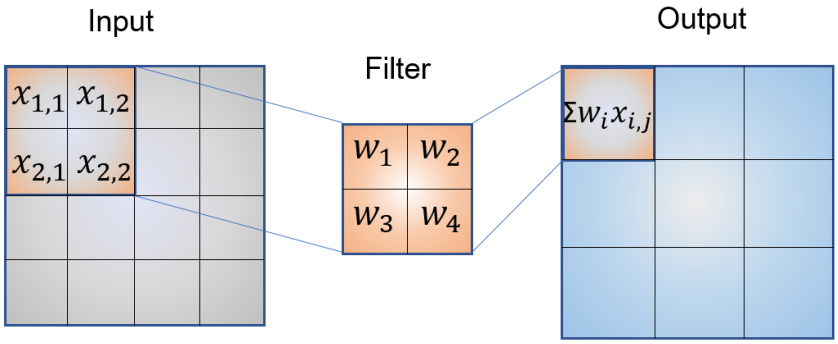


Figure 2.12: Example of a 2D convolutional filter.

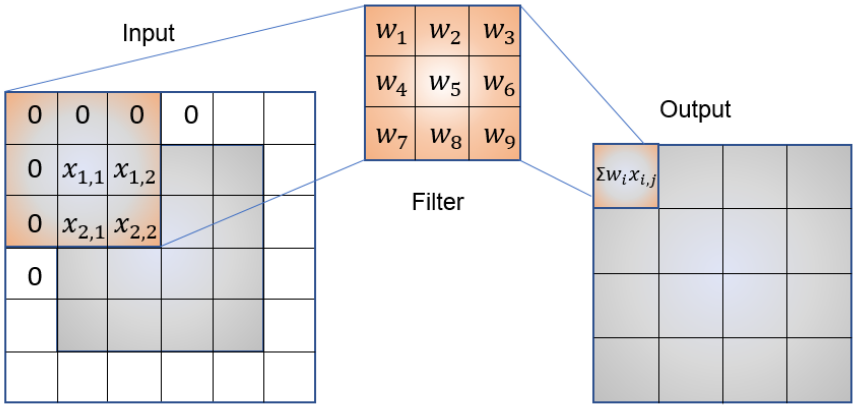


Figure 2.13: Example of zero-padding with 'same' dimensions.

expand the training dataset. This includes adding more data but also incorporating more challenging and varied data. Another solution may be to decrease the number of learnable parameters accessable to the model. This may seem counter-intuitive, however, less parameters will force the network to generalize the training data and therefore more likely to succeed on previously unseen test data. An illustration of this can be seen in Figure 2.15 where less parameters are used in the optimal model but produces a better model to predict future values. There are several other methods of reducing the risk of overfitting. A common and simple method is known as dropout. Dropout can be applied after the activation function of a layer and replaces a random percentage of the output values to zero. A case where the dropout is set to 0.5, half of the output of the activation functions will be set to zero therefore nullifying any dependency on those neurons and forcing the network to utilize other neurons to apply proper classification. This method can significantly reduce overfitting if utilized properly.

2.3.6 Batch normalization

As discussed previously, as each batch of training data is processed through the network the weights are updated through backprogagation via an optimization al-

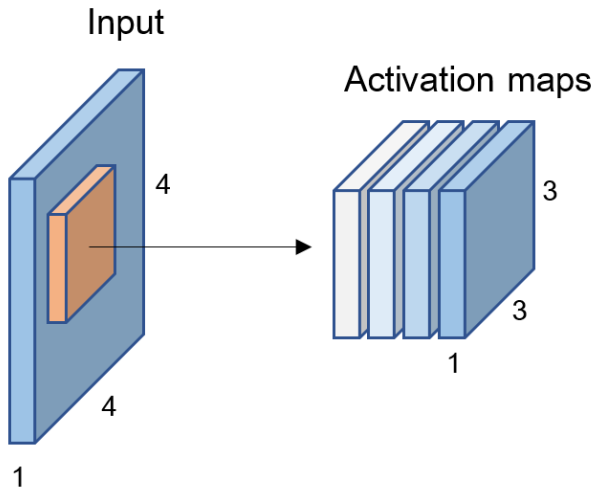


Figure 2.14: Example of a 3D convolutional filter.

gorithm. Each layer of the network contains a set of weights which are then applied to the inputs of the layer whose product is then carried into the next layer of the network. In many cases, the distribution of the output values of each layer is different and may result in a phenomenon known as internal covarial shift [20]. This phenomenon describes the change in means and variances of the inputs to the internal layers of the network during training. Although the benefits of batch normalization are not yet completely understood, the machine learning community have generally agreed that normalizing inputs before each layer greatly improves the speed, performance, and stability of neural networks, therefore reducing this internal covarial shift.

The normalization is done for every batch, $X = \{x_i, \ldots, x_{n_B}\}$, where n_B denotes batch size. The normalization procedure is as follows

$$\bar{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{2.13}$$

$$y_i = \gamma_1 \bar{x}_i + \gamma_2 \tag{2.14}$$

where μ_B and σ_B^2 are the mean and variance of the current batch, γ_1 is a learnable scaling factor, γ_2 is a learnable shift parameter, and ϵ is a very small value to avoid dividing by zero. The input, x_i, of every neuron is normalized to $\bar{x}$, and the output, y, is the result of the batch normalization. In tandem to overfitting, batch normalization has also shown to increase generalization in some cases [21].

2.3.7 Convolutional neural network

Utilizing the fundamental properties of convolutional filters, one can create a convolutional neural network (CNN) which consists of convolutional layers and fully connected layers. The advantage of using convolutional neural networks over a multi-level perceptron is the ability to successfully capture spatial and temporal

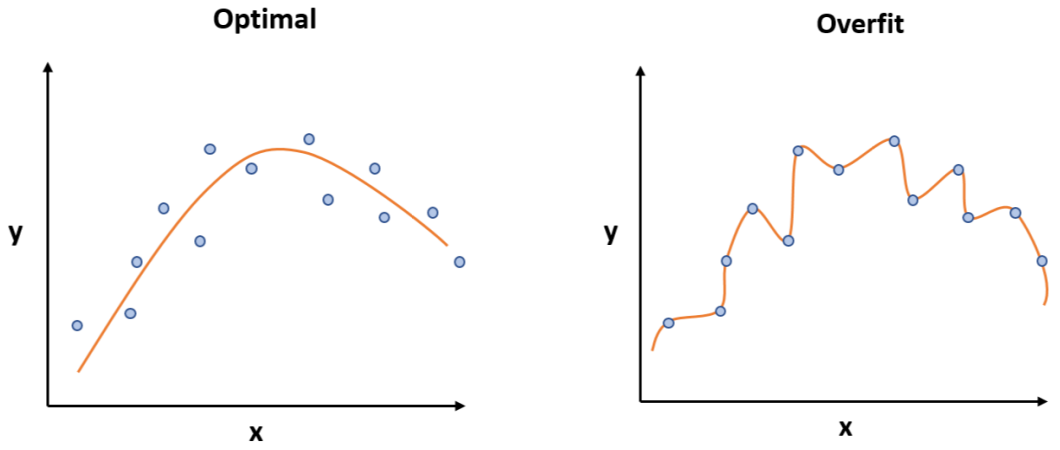


Figure 2.15: The optimal model utilizes less parameters to fit itself to the data points, therefore it is more likely to fit well if additional data points are added. The overfit model is perfect at representing the data points utilizing more parameters, however is very likely to perform poorly if additional points are added.

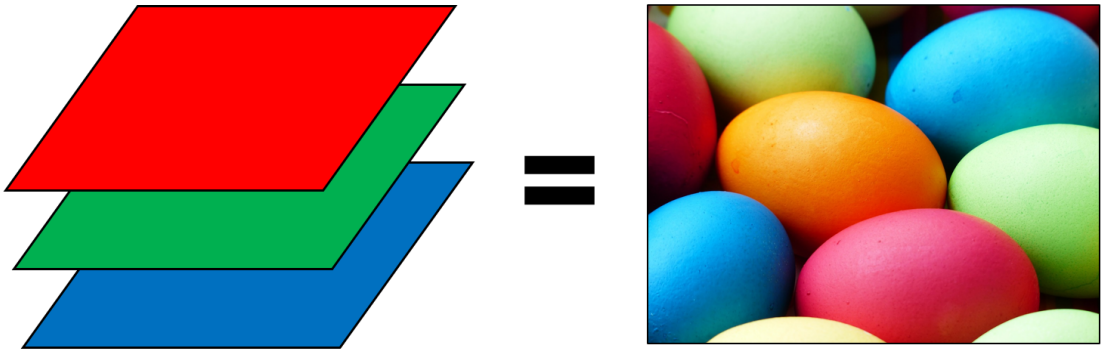


Figure 2.16: An example of an RGB image split into three channel, one for each color.

dependencies within an image. An example of an input image to a CNN can be seen in the RGB image in Figure 2.16. Each channel of the image is seperated into individual matrices and inputted into the CNN as a matrix of size $W \times H \times 3$, where W represents the width of the image and H represents the height. The goal of a CNN is to the reduce the input image into a form which is easier to process without losing features which can be critical for good predictions. When dealing with images of high dimensions such as 8K ($7680 \times 4320 \times 3$), it is critical to process the images with effective convolutional layers. Typically, an image is passed through several convolutional layers before passing through a max pooling layer and possibily several more convolutional layers. Max pooling refers to a process of reducing the spatial size of the output from a convolutional layer in order to reduce computational power required to process the data. Max pooling works similarly to a convolutional filter in the sense that it slides a square-sized kernal across the input, however, max pooling only returns the maximum value of the kernal to the output. An illustration

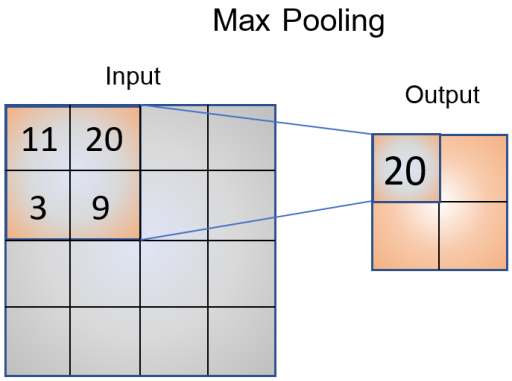


Figure 2.17: An example of a 2×2 size max pooling.

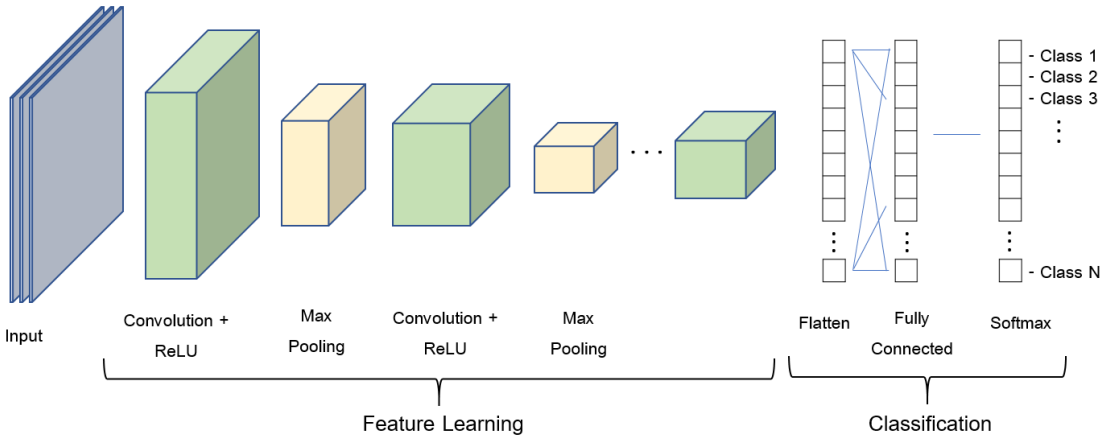


Figure 2.18: An example of a CNN architecture.

of max pooling can be seen in Figure 2.17. Once sufficient processing has been made from the several convolutional and max pooling layers that may make up the CNN, the output is flattened into a column vector and passed through one or several fully connected layers and backpropagation is applied to every iteration of training. The fully connected layers are where the network learns how to interpret the high-level features extracted by the convolutional layers. The output of the fully connected layers is finally passed though a softmax layer with the same number of neurons as there are classes to classify. The output vector from the softmax layer consists of a probability distribution. Each value of the probability vector represents a class and the highest or max value of this vector is the predicted class of the originally input image.

An illustration of a fully designed CNN can be seen in Figure 2.18

2.3.8 CapsNet

In 2017, the founding father of deep learning and well-respected inventor of novel machine learning algorithms, Geoffrey Hinton, released a series of two papers intro-

ducing a new neural network based on so-called capsules. This new network is based on an algorithm called *dynamic routing between capsules* that allows to train such a capsule-based network, or CapsNet [22]. This section summarizes the key aspects of CapsNet and how it differs from the traditional CNN.

2.3.8.1 Drawbacks of CNN

In general, CNNs are proven methods of successfully carrying out object classification with remarkable results. Despite their convenience and vast success, there are some fundamental limits and drawbacks.

Consider a very simple example of an image with two distinct features. For a CNN, the presence of these features can be a strong indicator to consider that there is a specific object in the image. Orientational and relative spatial relationships between these components are not very important to a CNN and therefore technically these features could be any where in the image, as long as both are present the CNN with read them as strong indicators. As previously discussed, CNNs are composed of convolutional layers whose job it is to detect important features. Lower level layers, i.e. closer to the input image, learn to detect features such as edges or color gradients, while higher layers will learn to combine simple features into more complex features. The highest fully connected layers will then combine these high level features and output a classification prediction. In CNN, pose (translational and rotational) is never taken into consideration between different features that make up the higher level features. In an attempt to solve this issue, max pooling is used to reduce the spatial size of the data and therefore increasing the "field of view" of higher layer neurons. This, in theory, allows higher level neurons to detect features in a larger region of the image. As stated before, max pooling results in fundamental data loss and therefore is a big gaping issue with this method for CNNs. Hinton, the inventor of the CapsNet, stated himself that max pooling working so well is a big mistake: *"The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster."* [23]. Max pooling does not need to be used in CNN to get good results but removing max pooling still does not solve the main problem which is that a CNN does not take into account the pose between two simple features in an image.

2.3.8.2 Capsules

A capsule in CapsNet is meant to replace the traditional neuron in a normal neural network. As stated in Hinton's paper, "Capsules encapsulate all important information about the state of the feature they are detecting in vector form." [22]. In other words, the length of a capsules output vector represents the probability of detection of a feature. The direction of which the vector points encodes the state of the detected feature. If a feature changes pose in an image, the length of the output vector of the capsule will not change, but the direction will. This type of invariance is not available to CNNs and max pooling which makes CapsNet an interesting alternative. A comparison between a capsule and a neuron is in Figure 2.19.

Capsule vs. Traditional Neuron			
Input from low-level capsule/neuron		vector($\mathbf{u}_i$)	scalar(x_i)
Operation	Linear Transform	$\hat{\mathbf{u}}_{j\|i} = \mathbf{W}_{ij}\mathbf{u}_i$	-
	Weighted Sum	$\mathbf{s}_j = \sum_i c_{ij}\hat{\mathbf{u}}_{j\|i}$	$a_j = \sum_i w_i x_j + b$
	Nonlinear Activation	$\mathbf{v}_j = \frac{\\|\mathbf{s}_j\\|^2}{1+\\|\mathbf{s}_j\\|^2}\frac{\mathbf{s}_j}{\\|\mathbf{s}_j\\|}$	$h_j = \xi(a_j)$
Output		vector($\mathbf{v}_j$)	scalar(h_j)

Figure 2.19: Difference between the computational process of a capsule and traditional neuron.

2.3.8.3 Dynamic routing between capsules

When a lower-level capsule outputs a vector representing the probability of a feature, it must pass this information to the next higher layer of capsules. This decision process is known as dynamic routing between capsules . As Hinton states, "Lower level capsules will send its input to the higher level capsule that 'agrees' with its input. This is the essence of the dynamic routing algorithm." [22]. With this in mind, the routing algorithm is presented in Algorithm 2.

Algorithm 2: Routing algorithm. The suggested number of iterations, r, is 2-3, more could cause overfitting [22].

Input : $\hat{\mathbf{u}}_{j|i}$: Output vector of capsules in layer l
Input : r: Number of routing iterations
Input : l: Current layer of capsules
Output: $\mathbf{v}_j$: Input to capsules in layer $l+1$
for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$
for r *iterations* **do**
  for all capsule i in layer l: $\mathbf{c}_i \leftarrow softmax(\mathbf{b}_i)$
  for all capsule j in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}$
  for all capsule j in layer $(l+1)$: $\mathbf{v}_j \leftarrow squash(\mathbf{s}_j)$
  for all capsule i in layer l and capsule j in layer $(l+1)$:
    $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i} \cdot \mathbf{v}_j$
end

The key element in this algorithm is the vector $\mathbf{c}_i$ which is essentially a probability vector that decides for each lower-level capsule which higher level capsule its output will go to. Softmax is used ensure the weights $\mathbf{c}_{j|i}$ are non-negative and whose sum

is equal to 1. The squash function

$$\mathbf{v}_j = \underbrace{\frac{\|\mathbf{s}_j\|^2}{1+\|\mathbf{s}_j\|^2}}_{extra"squashing"} \cdot \underbrace{\frac{\mathbf{s}_j}{\|\mathbf{s}_j\|^2}}_{unitscaling} , \tag{2.15}$$

is a novel nonlinear activation function that takes a vector and squashes its values between 0 and 1, but does not change its direction. This squashing function is described in [22] and presented in equation (2.15).

2.3.8.4 Architecture

The officially presented CapsNet from Hinton's papers consists of roughly 11.36 million trainable parameters. There is some scepticism on the exact number of parameters since the architecture is not very detailed in the original papers but, nonetheless, these parameters can be scaled to fit different applications. The CapsNet consists of two main parts: encoder and decoder. This thesis utilizes only the encoder and does not attempt to use a decoder to reproduce input data. The encoder part consists of three main layers: convolutional layer, PrimaryCaps layer, and ObjectCaps layer.

The first layer consists of a convolutional layer of 256 kernels with size $9 \times 9 \times 1$ and stride 1, followed by a ReLU nonlinear activation function. This layer is tasked with detected basic features in the input image.

The second layer consists of 32 primary capsules, with the same capsule properties discussed previously. This first layer of capsules is tasked with taking the basic features from the convolutional layer and produce combinations of the features. Each capsule applies eight $9 \times 9 \times 256$ kernels with stride 2. Since a stride of 2 is used, the output produces a size of $6 \times 6 \times 8$. The third dimension of the output represents the length of the output vector of the capsules, in this case 8.

The third layer consists of N object capsules, one for each class where N is the number of classes. The task of this layer is to take the input from each primary capsule and map them to a $16 \times N$ output space via an 8×16 weight matrix for each capsule.

The final step is calculating the loss for each of the object capsules which is presented as the CapsNet loss function [22]

$$L_c = T_c \cdot \max\left(0, m^+ - \|\boldsymbol{v}_c\|\right)^2 + \lambda(1-T_c) \cdot \max\left(0, \|\boldsymbol{v}_c\| - m^-\right)^2 , \tag{2.16}$$

where T_c is the true label, m^+ is a constant set to 0.9, m^- is a constant set to 0.1, $\boldsymbol{v}_c$ is output vector of capsule c, and λ is a constant set to 0.5 for numerical stability.

2.4 Evaluation metrics

2.4.1 Classification Accuracy

A common and straight forward method for evaluating a network's performance is simply calculating the classification accuracy on the test data. Classification accuracy is the ratio of number of correct predictions over the total number of predictions made.

$$\text{Classification accuracy} = \frac{\#\ of\ Correct\ predictions}{\#\ of\ Predictions\ made} \tag{2.17}$$

This method works best when the number of test samples is evenly spread across all classes. A big drawback to this method is that it does not provide any information regarding how well the network predicted certain class compared to another class. If the test dataset is composed 90% of one class then the accuracy can also yield a false positive accuracy if the other 10% is poorly predicted. Poor results on test datasets where the classification accuracy is high but 10% of the samples are poorly predicted compared to the rest, in certain applications, could cause harm if applied in real-life scenarios such as vehicle safety functions or medical procedures and diagnosis.

2.4.2 Confusion Matrix

A more in-depth evaluation can be done through a confusion matrix [24]. As the name suggests, the confusion matrix yields a matrix and describes the complete performance of the model. Assume a classification problem involving two different classes and a trained model is used to make predictions on a test dataset of 100 samples. An example output from a confusion matrix given this situation could yield the following result. Given the matrix above, there are four important aspects:

- True Class 1 - The cases where predicted is Class 1 and the actual is Class 1.
- True Class 2 - The cases where predicted is Class 2 and the actual is Class 2.
- False Class 1 - The cases where predicted is Class 1 and the actual is Class 2.
- False Class 2 - The cases where predicted is Class 2 and the actual is Class 1.

The classification accuracy is also the trace of the confusion matrix divided by the total number of samples.

Table 2.1: An example of a confusion matrix given 100 test samples and two different classes.

n=100	Predicted: Class 1	Predicted: Class 2
Actual: Class 1	50	10
Actual: Class 1	5	100

2.4.3 F1 Score

From the confusion matrix, one can quantify the model's performance on each class through the F1 Score

$$F1 = 2 \cdot \frac{1}{\frac{1}{precision} + \frac{1}{recall}} \ . \tag{2.18}$$

This metric is defined as the harmonic mean between precision and recall, which are quantitative values that combine to give the F1-score. Precision

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \ , \tag{2.19}$$

is the number of correct positive results divided by the number of positive results predicted by the model. Recall

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \ , \tag{2.20}$$

is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive). The range for the F1 Score is [0, 1] and details how precise a model is (i.e number of instances it classifies correctly), as well as how robust it is (i.e does not miss a significant number of instances) [24]. High precision and lower recall yields an extremely accurate model but it fails on a large number of samples that are difficult to classify. The greater the F1 Score, the better the performance of the model is on the test set. The key to a high F1 Score is finding the balance between precision and recall. Both precision and recall are available in the columns and rows of the confusion matrix.

2.5 Related work

Object classification via machine learning methods has proven effective when using sonar and audio signals, however there is much more research and experimenting to be done with ultrasonic sensors especially in the automotive industry. One very recent project conducted within the automotive industry was done by the Valeo Switches and Sensors department in Kronach, Germany which showed that it was possible to classify the height of certain low lying object/obstacles on the ground in front and behind a vehicle with accurate results up to 99.6% [29].

An important part of this thesis is how the ultrasonic data can be processed and visualized. Since CNN and CapsNet require images as input, it is important to visualize the data in such a way that includes as much detail as possible. How to visualize signal data has been researched quite a bit in previous projects such as in [29] with ultrasonic data and in [31] with audio signals. In the Valeo project, the ultrasonic signal data is organized into a 2-channel (real and imaginary) 14x14 image as input to the neural network [29] where as in [31] the audio data is visualized in a spectrogram and inputted into the network. Both yield promising results of which the spectrogram input method attempts to classify spoken numbers from 0-9 and achieved an accuracy of 97% [31].

The most common approach to processing signal data is to begin by sampling the signal at a sufficient sampling rate. As [31] states, using high frequency data such as ultrasonic for machine learning presents a huge problem - high dimensionality. One second of sound captured at 14.4kHz contains 14,400 data points, and the information it contains is more than just a statistical time series of pressure readings. It's a physical wave, with all of the properties that come along with physical waves, including oscillations, envelopes, phase, jitter, transients, and so on [31]. Furthermore, as shown in [29], filtering and downsampling must be applied to reduce noise and scale the dimentionality of the problem down enough to efficiently train the neural network in a reasonable time. In this project, a sampling rate of at least 80kHz will be needed in order to meet Nyquist theorem and various filtering methods will be tested and implemented.

The architecture of several major CNN networks has been studied and will inspire the construction of the CNN in this thesis. The CapsNet algorithm is formally presented in [22]. This method utilizes so-called capsules. A Capsule is a nested set of neural layers. So in a regular neural network you keep on adding more layers, but in CapsNet you would add more layers inside a single layer. Or in other words nest a neural layer inside another. In [22], CapsNet proved to yield better results than traditional CNNs on the MNIST dataset [32]. More in detail about CapsNet can also be read in [33] and the Github repository of the implementation of Capsnet used in [22] can be found at [34].

The main feature of CapsNet is its ability to use dynamic routing between capsules which claims to be superior to max pooling in [22]. Because of this feature, these capsules are good at handling different types of visual stimulus and encoding things like pose (position, size, orientation), deformation, hue, texture, etc [33].

3

Method

This section explains how data was collected and processed, and how they are reorganized and fed into the machine learning algorithms. The classification problem is defined in detail along with which scenarios were considered for testing. The section ends in an explanation of how the networks are defined and trained, and which metrics are used to evaluate them.

3.1 Equipment and experimental setup

The measurements collected to produce the results of this thesis are based on the following equipment and experimental setup to allow for easy reproduction of the results. Section 3.1.1 introduces the hardware used such as the ultrasonic sensors, function generator, and oscilloscope. Section 3.1.2 details the exact setup used to perform measurements and tests.

3.1.1 Equipment

Measurements are collected using the function generator and oscilloscope. The sensors used in this thesis are not the same type installed in current Volvo vehicles. There are two types of sensors used, transmitter and receiver.

3.1.1.1 Ultrasonic sensors

The two types of ultrasonic sensors used to collect measurements are listed below:

- KPUS-40T-16T is an ultrasonic transmitter whose center frequency is at 40 ± 1 kHz. The maximum input voltage is 20V. More detailed information on the KPUS-40T-16T transmitter be found within the technical data sheet [25].

- KPUS-40T-16R is an ultrasonic receiver whose center frequency is at 40 ± 1 kHz. More detailed information on the KPUS-40T-16R receiver be found within the technical data sheet [26].

In total, 1 transmitter and 3 receivers are used for this project.

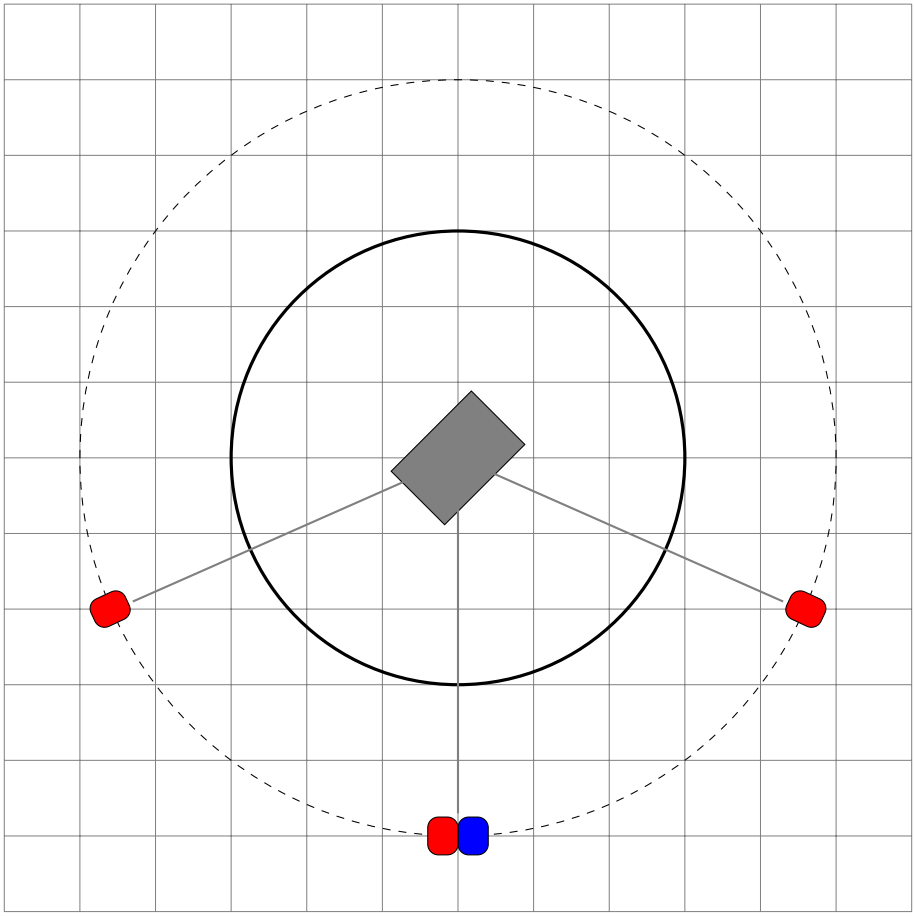


Figure 3.1: Top-down view of the experimental setup. Receivers are label with a red color, transmitter with blue, and object with gray.

3.1.1.2 Oscilloscope

The PicoScope 3000 Series 3425 Differential PC Oscilloscope is a 4 channel USB 2.0 oscilloscope that is connected to a computer and includes built-in Matlab functions for data collection. With four channels enabled, the oscilloscope can reach a maximum sampling rate of 5 MS/s per channel with a resolution of 12-bits. The bandwidth is 5MHz. More details can be found in the user manual [27].

3.1.1.3 Function Generator

The KEYSIGHT 33500B Series Waveform Generator is used to generate a 10V, 40kHz 100 cycle sinusoidal burst with a 25ms burst period. With USB compatibility, the generator can be controlled from a computer via Matlab functions. The built-in trigger function also allows a signal to be transmitted at the start of each new measurement event initialized by the oscilloscope. This helps to ensure consistency when taking a large number of measurements in sequence. More details can be found in the user manual [28].

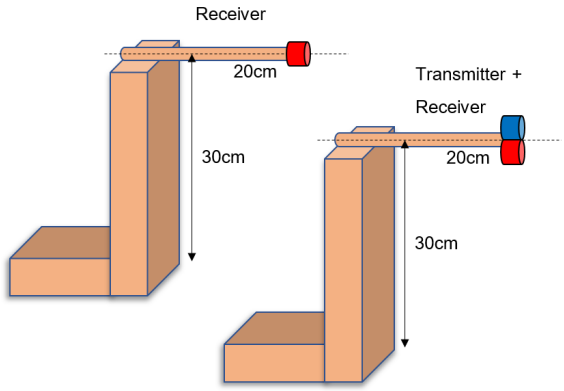


Figure 3.2: Visual illustration of the rigs used to mount the sensors.

3.1.2 Experimental Setup

In order to ensure reproducability and reduce complexity to properly analyze the possibility of object classification using ultrasonics, the setup consists of a simple and easy to understand design. The main setup consists of three sensor mounts, one rotating platform, and the object. Figure 3.1 shows the location of each sensor, rotating platform, and object.

Each sensor, transmitter and receivers, shown in Figure 3.1 is mounted on a small rig in order to ensure stability and allow for the sensors to be quickly maneuvered to a different location if desired. Figure 3.2 shows an example of how one of the mounts look like. The transmitter and a receiver are mounted together on one mount positioned at 0 degrees relative to the object as depicted in Figure 3.1. The other two receivers are placed on separate mounts at ± 65 degrees relative to the object as shown in Figure 3.1 in red. This setup was placed in an enclosed room at room temperature with all other obstacles, including walls, persons, and other miscellaneous objects, at least 1 meter away from the experimental setup to avoid any significant reflections from them.

Each sensor (1 transmitter and 3 receivers) is connected to a separate test probe which is connected to a separate port in the 4-channel PicoScope oscilloscope. The oscilloscope is connected and powered by USB through a PC which hosts Matlab and stores the raw signal. The function generator sends the pulse signal to both the transmitter and Channel A of the oscilloscope. The pulse from the function generator is sent to the oscilloscope channel A in order to analyze the transmitted signal as reference. Channels B, C, and D host the 3 receivers. The entire measuring cycle/circuit can be visualized in Figure 3.3.

Five different objects were used to train and test classification. Each object is distinguished by their geometric property and size. A sixth class is added named "noObject" which is included to ensure the networks would also learn to detect whether an object is present or not. Therefore, in total, there are six classes.

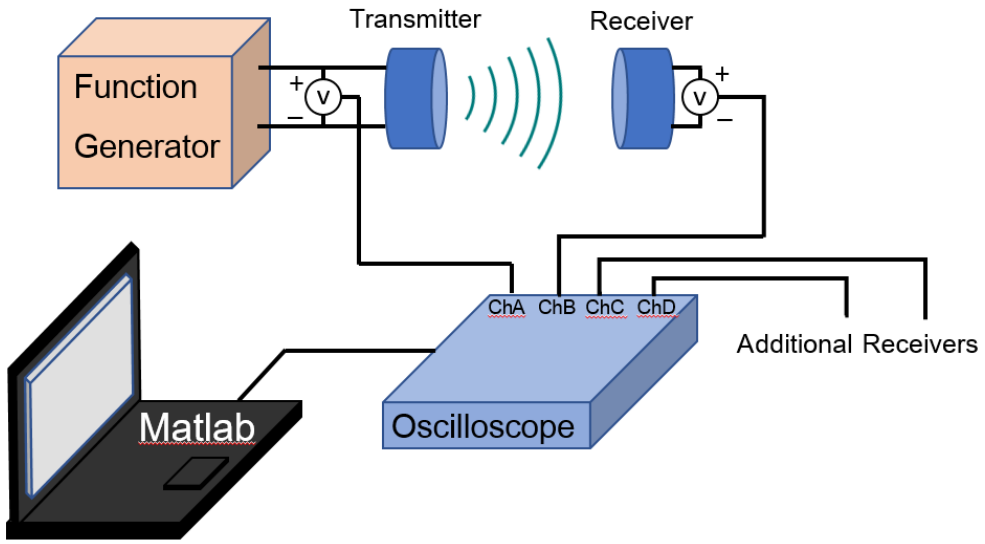


Figure 3.3: Pseudo-circuit showing the high-level measurement setup.

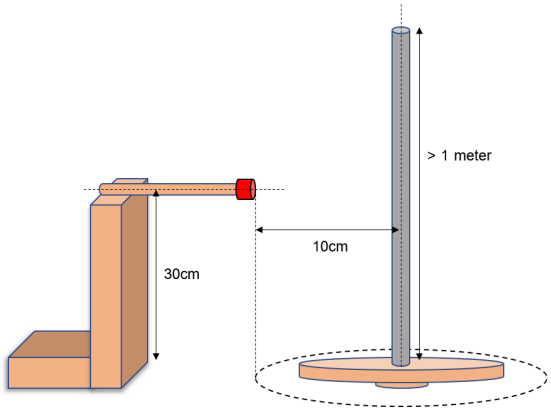


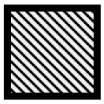
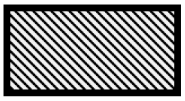
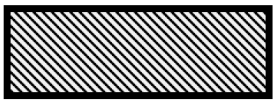
Figure 3.4: Illustration of how considerably large the height of the objects are compared to the distance to the sensors.

Classification is based on classifying the objects cross-sectional geometrical structure, in other words, the height of the objects in the vertical direction are significantly large. The geometric structure of each object is also uniform in the vertical direction, i.e. the geometry is consistent. A better illustration is made in Figure 3.4 where the circular shaped object is depicted as an example. The objects and their dimensions can be seen in Table 3.1.

3.2 Data collection

In order to achieve a wide variety of data and later be able to evaluate several aspects of the networks performance and robustness, the data collection process is divided into three main categories. Collecting various measurements in different scenarios

Table 3.1: The objects used for classification. All objects are made from ordinary wood-building material.

Object	Width	Height	
Circle	5cm	5cm	
Square	5cm	5cm	
Rectangle	10cm	5cm	
Large rectangle	20cm	5cm	
Triangle	5cm	5cm	

will help analyze how the networks' performances are affected. The more robust networks should perform well across all training datasets. The three categories by sample size are: 1600, 3300, and 6000 samples. Each of the three categories contain the exact same number of measurements and produce the same size training dataset after preprocessing, the only difference being the number of samples per measurement. As depicted in Figure 3.4, each object is set to a constant rotation on the rotating platform during data collection. The objects rotate on a platform at a constant rate 3 revolutions per minute. Each data category is further broken down into subcategories of measurements. The first subcategory is the input type: complex baseband and spectrogram. These are the two fundamental types of training data that will be compared by how well they perform on the CNN and CapsNet networks. The next subcategory is the construction method, i.e. how each of the two input types are created using the preprocessed data. The three construction methods are images in PNG format, matrices of size $N \times N \times 2$ where the two channels represent the real and imaginary parts of the signal, and lastly a stacked matrix form of size $N \times N \times 6$ where the data from all three receivers are stacked with their respective real and imaginary channels to create a single matrix.

Figure 3.5 breaks down all the previous statements in an illustrative diagram. A measurement is defined as the collected data from all three receivers. A measurement sequence is initialized with two main parameters. The first is the number of measurements to take, M. Note, measurements are taken in series in order to make the measurement process more efficient, but this will not affect the network training since the training data is randomized. The second parameter is the number of samples per measurement, nS.

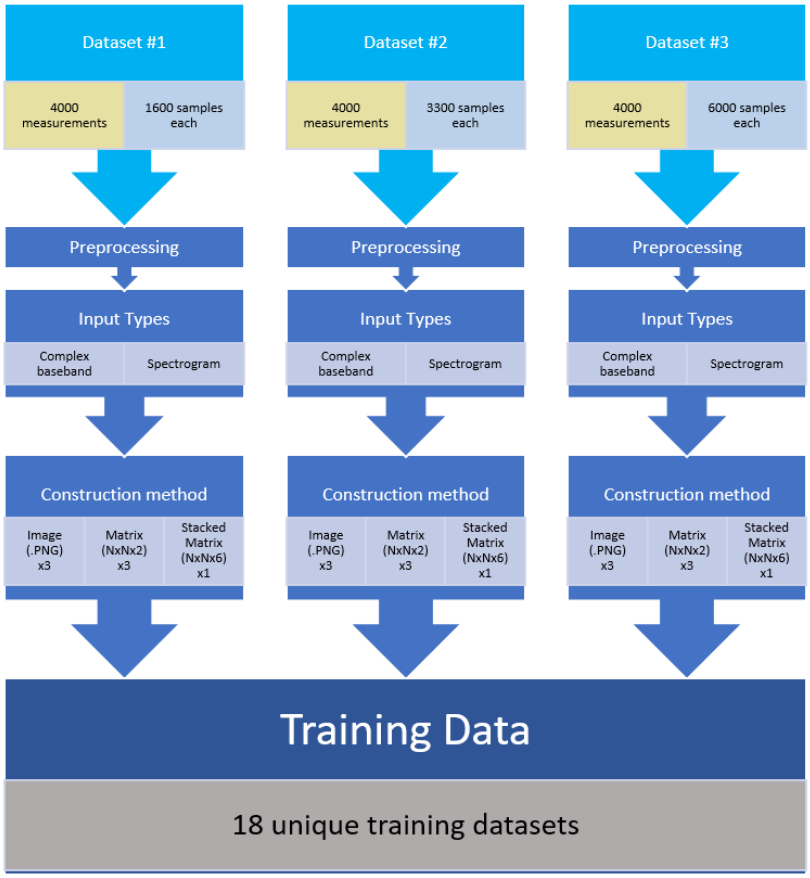


Figure 3.5: An illustrative diagram of the entire data collection process. Note: Each one of the 4000 measurements also contains the data from all three receivers, hence, there are 4000×3 measurements per dataset. Each network CNN and CapsNet will be trained on the 18 resulting training datasets.

At the start of each new measurement, the function generator triggers a single 10V, 100 cycle 40kHz sinusoidal burst allowing each sensor to capture the data from a single pulse. Each measurement, as its collected, is encoded to a digital signal via the 12-bit ADC and stored within the local memory of the oscilloscope. The memory size of the oscilloscope allows it to store a large number of measurements in series allowing the user to collect measurements in rapid series. Once all measurements have been collected, the oscilloscope sends the data to the computer. Each measurement, $\boldsymbol{x}_{raw}(n)$, (containing the raw output from each of the 3 receivers) is stored separately within Matlab.

3.3 Preprocessing of data

After a series of measurements have been taken, each measurement, $\boldsymbol{x}_{raw}(n)$, is passed through a series of preprocessing procedures to clean up each signal. Due to the quality of the sensors used and possible unwanted external interference, it is expected to see some low and high frequency noise disrupting the signal of interest. An example of an output from a receiver in a random measurement taken

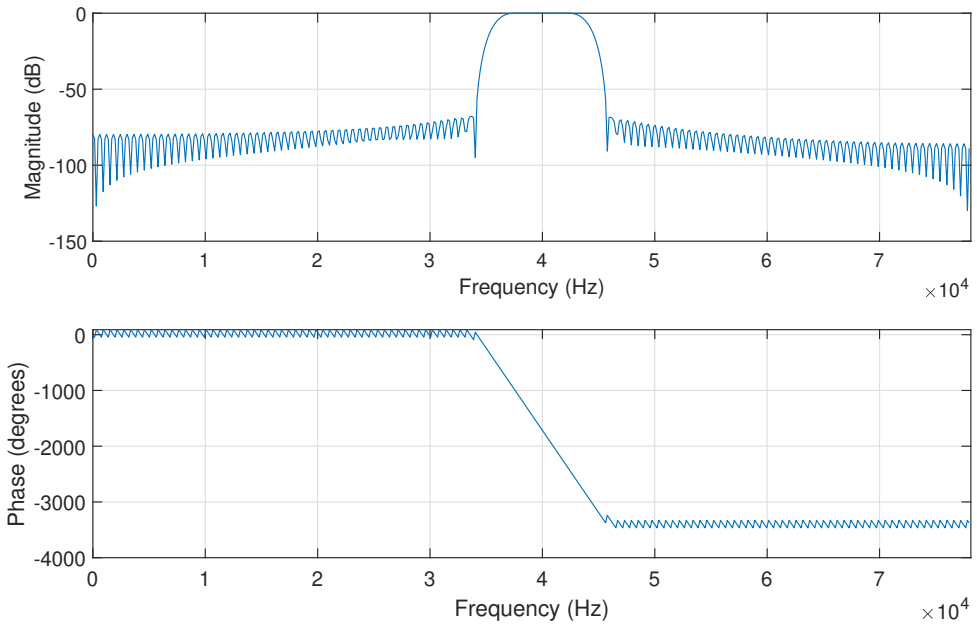


Figure 3.6: Frequency response of the designed Hamming window-based FIR filter.

with the current setup shown in Figure 3.7 clearly shows a distinct low frequency disturbance as well as some potential high frequency noise potentially coming from other surrounding electronics.

To remove these unwanted low and high frequencies, a window-based bandpass FIR filter is used. In this case, the Hamming window is used to filter the raw signal, $\boldsymbol{x}_{raw}(n)$, to a filtered signal, $\boldsymbol{x}_f(n)$. The filter is designed with stopband frequencies at $f_{S_1} = 35$ kHz and $f_{S_2} = 45$ kHz, and with passband frequencies at $f_{P_1} = 37$ kHz and $f_{P_2} = 43$ kHz. The frequency response is illustrated in Figure 3.6.

After the bandpass filter, the Hilbert transform is applied to $\boldsymbol{x}_f(n)$ as described in Section 2 and produces the analytical signal, $\boldsymbol{x}_a(n)$.

The next preprocessing step involves shifting the signal from passband to baseband. Converting a passband signal to baseband can be done with a simple frequency shift in the frequency domain based on Euler's formula. The passband signal is shifted by $f_c - \frac{f_{bw}}{2}$, where f_c is the center frequency and f_{bw} is the bandwidth of the bandpass filter. The shift to baseband is described as

$$\boldsymbol{x}_b(n) = \boldsymbol{x}_a(n)\mathrm{e}^{-j2\pi(f_c - \frac{f_{bw}}{2})n/F_s} \tag{3.1}$$

where F_s is the sampling rate. The reason for shifting is to allow for the important data in the frequency domain to lie in within the first segment of values and makes it easy to extract later. This shift also allows for the potential in future experiments to resample the signal at a much lower sampling rate if needed, i.e. at a minimum of 2 kHz following the Nyquist theorem.

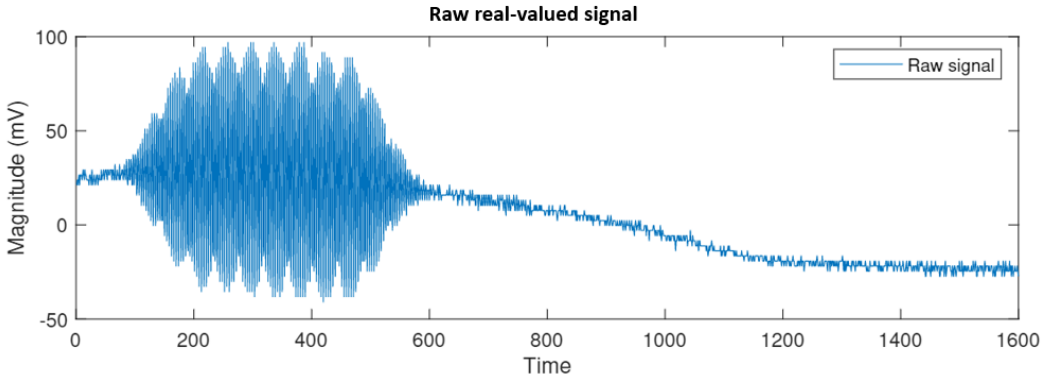


Figure 3.7: An example raw signal from a measurement of 1600 samples.

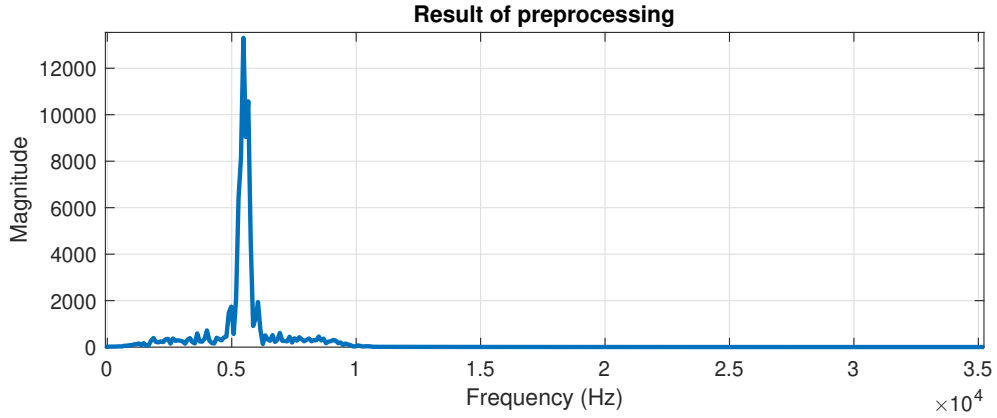


Figure 3.8: Frequency response, $\boldsymbol{X}_b(k)$, of the final preprocessed signal.

The final step is to convert the signal, $\boldsymbol{x}_b(n)$, to the frequency domain, $\boldsymbol{X}_b(k)$. Therefore, the result of preprocessing produces a signal, $\boldsymbol{X}_b(k)$, visualized in Figure 3.8. For a clearer picture, an overview of the processing steps can be also seen Figure 3.9

3.4 Data reorganization

After the preprocessing, each measurement is reorganized into two different formats used to create the training data of the CNN and Capsnet algorithms. Both data sets stem from the the same data and carry the same information but are visualized differently in an attempt to positively affect performance and draw conclusions from their comparisons. These two different data sets will be referred to as: the complex baseband and the spectrogram.

3.4.1 Complex baseband

What will be referred to as the complex baseband in this report, is a reorganized projection of the signal, $\boldsymbol{X}_b(k)$, from Figure 3.8. The signal, $\boldsymbol{X}_b(k)$, after preprocessing consists of a one dimensional complex-valued vector of size nS, where nS is the

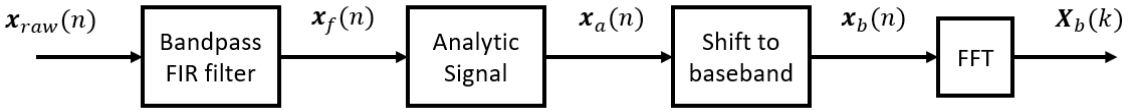


Figure 3.9: Overview of the preprocessing procedures.

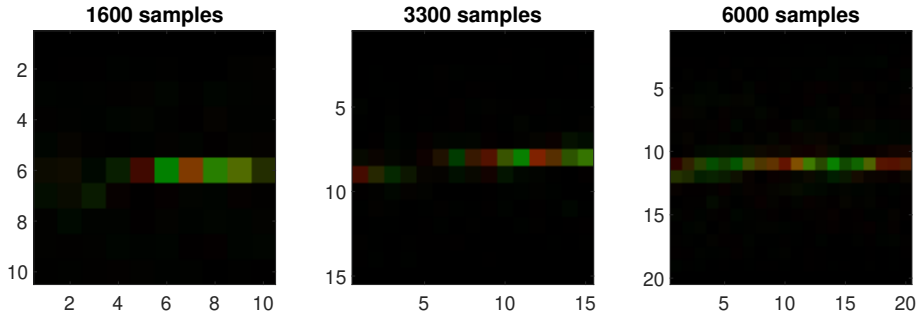


Figure 3.10: Example of complex envelope images for the three sample sizes 1600, 3300, and 6000.

number of samples in the signal. The baseband signal consists of the frequencies 0 kHz − 10 kHz, which for a sample size of 1600, is equivalent to the first 100 samples in $\boldsymbol{X}_b(k)$, which will produce a vector $\boldsymbol{X}_c$ with a length of 100. This new vector, $\boldsymbol{X}_c$, contains essentially only the frequency information within the bandwidth of the bandpass filter and therefore the only data interesting to train on for the neural networks. As discussed previously, the entire data collection set includes three subsets of data differentiated by the number of samples taken. For example, a measurement consisting of 6000 samples will result in a complex-valued frequency-domain vector of length 400 after preprocessing, measurement of 3300 samples will result in a complex-valued frequency-domain vector of length 225, and measurement of 1600 samples will result in a complex-valued frequency-domain vector of length 100.

Using Matlab's *reshape*() function, this one dimensional vector of size nS can therefore be reshaped into a matrix of size $N \times N$ without an loss of information since this step is merely a reshaping. From this complex-valued matrix, the three different construction methods can be created from Figure 3.5 for the complex baseband input type.

The first construction method is the image. An image requires 3 channels: red, green, and blue. The red channel is created from the real part of the complex-valued matrix. The green channel is created from the imaginary part, and the blue channel is simply filled with zeros. Combining the three channels creates an image. Examples can be seen in Figure 3.10 for each sample size. 1600 samples yields a 10×10 image, 3300 yields 15×15, and 6000 yields 20×20. Remember, each measurement contains the output from each of the 3 receiver. This implies that 3 images will be created from this construction method.

The second construction method described in Figure 3.5 is the $N \times N \times 2$ matrix

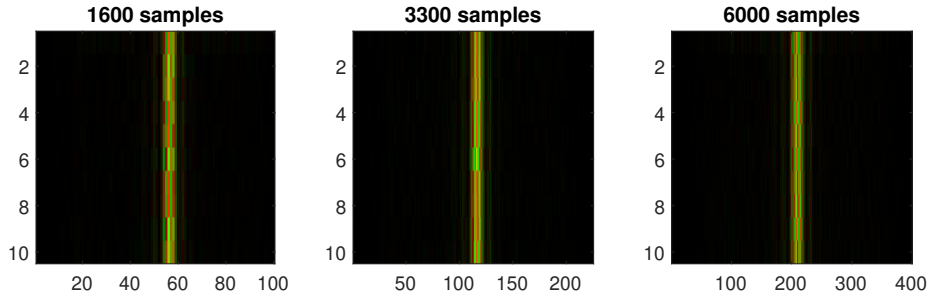


Figure 3.11: Example of spectrogram images for the three sample sizes 1600, 3300, and 6000.

form. This method is similar to the image except it does not convert the real and imaginary channels to an image, instead, leaves the matrix in its 2-channel form. Note: this creates a $N \times N \times 2$ matrix for each of the three receivers.

The last construction method described in Figure 3.5 is the $N \times N \times 6$ stacked matrix form. This method essentially takes the three $N \times N \times 2$ matrices from the second method and stacks them into a single 6-channel matrix.

As a result, three different training datasets are created for the complex baseband input. Each training dataset consists of 4000 training samples per class resulting in a total training size of 24,000.

3.4.2 Spectrogram

The second type of input used for training the networks is the spectrogram. As stated previously in the background section, spectrograms visualize signals by essentially plotting frequency spectrum over time with an overlayed heatmap showing magnitude values at each frequency in time. The method for constructing the spectrogram in this project is by combining the frequency spectrum of a series of 10 measurements. Ten consecutive measurements make up a single spectrogram, hence, the size of the spectrogram training dataset would be considerably smaller than the complex baseband since 10 measurements are required to make 1 spectrogram. To combat this issue, a type of "window" is slid across the series of measurements to create more spectrograms for training. As a result, the training data for the spectrogram input consists of 3865 measurements per class totaling to a training datset size of 23,190 spectrograms. An example image of a spectrogram created from this process is visualized in Figure 3.11. The other two construction methods are built in the exact same way as the complex baseband. In this case, 1600 measurement samples produces spectrogram input size of 10×100, 3300 produces a 10×225, and 6000 produces 10×400.

3.5 Convolutional neural network

Given that one of the construction methods, the stacked matrix of size $N \times N \times 6$, only yields 1 input, two versions of the CNN architecture need to be created: a multi-input CNN and a single-input CNN.

The relationship between the three sensors is very important and this cannot be lost or disconnected when designing the architecture of the CNN. Each sensor provides a different perspective of the object which is important when training the networks on classification. In short, additional sensors give the networks more data and allows the networks to make connections between them. Two CNN architectures are explored in this section. The first utilizes a parallel structure where the measurement from each of the three receivers is inputted into its separate CNN before being combined and output a single prediction. The second is a traditional CNN of 1 input, 1 output but the measurement inputted includes all three receiver outputs stacked on top of each other, as mentioned in Section 3.4, to create a 6-channel input representing the real and imaginary channels for each of the three receiver measurements. All networks utilize Adam optimization with default initialization parameters specified in [19] and use a learning rate of 0.001. The loss function for both CNNs is categorical cross entropy loss.

The first CNN creates three parallel layers that are later connected by a fully connected layer that has the sole task of making connections between what each sensor is observing and ultimately determining the class of the object. Figure A.2 visualizes and details the parameters used for each layer and activation function. This parallel CNN structure is used for both the image and $N \times N \times 2$ matrix construction methods.

The second CNN explored takes the shape of a more traditional CNN and begins with a single input layer followed by a series of convolutional filter layers, max pooling, and finally a fully connected layer and softmax, shown in detail in Table A.1.

In summary, two version of CNN created in order to handle the type of input data being used during training. Also, both networks utilize batch normalization and dropout to reduce the chance of overfitting and to encourage generalization.

3.6 CapsNet

The CapsNet architecture is based on the architecture introduced in Section 2.3.8 and realized in [29]. The architecture is implemented using Tensorflow 1.15. Similar to the CNN, two versions of the CapsNet architecture are created. One consists of the exact same structure as the one presented in Hinton's paper, and the other is constructed similarly to the parallel structure of the CNN architecture utilizing three separate networks combined into one.

The only change made to the proposed CapsNet architecture from Hinton is the size of the convolutional filter kernel size. Hinton presents the use of a kernel size of 9×9, however, in this project the input dimensions are considerably smaller so a kernel size of 5×5 is used instead. The CapsNet structure based on Hinton's paper including this minor adjustment can be visualized in Figure A.3.

The second version of CapsNet explored creates three parallel CapsNet structures followed by a series of fully connected layers but with scaled down number of learnable parameters for each of the three parallel structures. Scaling down the parameter size produces a total parameter size close to the traditional CapsNet architecture presented by Hinton. The same type of logic is applied to this second version of CapsNet as in the parallel structure of the CNN used for this project. Three inputs, one from each receiver, are used and passed through a separate set of layers before combining into a fully connected layer to make a single prediction. This CapsNet structure can be visualized in Figure A.4.

3.7 Labeling and Performance Evaluation

During the process of creating the training sets, each training sample is given a label determined by a one hot encoded vector. In the case of 6 different classes, this vector is given a length of 6 and a 1 is placed at the index which represents what object the training sample represents.

The evaluation of both the CNN and CapsNet networks are done in several steps. The first step is to test both networks on their ability to do object detection. In order to verify the capability of the networks to properly use the data collected for classification before introducing them to the six-object classification problem, each network is tested to classify between two classes: "noObject" and "Object". "noObject" represents there not being an object present, whilst "Object" represents any type of object present. Both the classification accuracy and confusion matrix will be evaluated to judge the performance of each network.

The second step of the evaluation will consist of testing both CNN and CapsNet on two different test datasets, explained in more depth in the Section 4. Each dataset will test the effects of increasing measurement sample size and compare the performance of the complex baseband and spectrogram input types for each network. Classification accuracy will be used when computing performance averages and overall performance, however, the detail lies in the confusion matrices which will also be presented.

Both networks CNN and CapsNet are trained for 50 epochs during object detection training and 100 epochs during the object classification training with a batch size of 100 used in all cases. Again, Adam optimizer is used for all trained models and initialized with default parameters described in [19] with a learning rate of 0.001 for all networks.

3.8 Computer hardware

All networks are trained using a Graphics Processing Unit, GPU, to make training as efficient as possible. For this thesis, a Geforce GTX 1060 with 6GB GDDR5 is used to train both CNN and CapsNet networks. GPUs are usually preferable when training deep neural networks due to their capacity to more efficiently process large amounts of data compared to a CPU. Both CPU and GPU are capable of training all networks presented in this thesis.

4

Results

In total, 36 networks were trained (including all combinations of CNN/CapsNet, 1600/3300/6000 sample size, complex baseband/spectrogram, and image/matrix/6-channel input types). All 36 networks were tested and results summarized in this section. The performance of CNN and CapsNet is evaluated on two sets of data, T_1 and T_2, detailed in Table A.1. The first, T_1, is a subset of data collected in the same instance as the training data, however this is still data the networks have never seen before. The second dataset, T_2 represents data collected at a completely separate instance that aims to simulate the act of reproducing measurements at a different time and day and to test the robustness of both CNN and CapsNet. A summary of the result comparison between CNN and CapsNet can be visualized in Table 4.1

Table 4.1: Brief summary of classification accuracy comparing CNN and CapsNet given the test datasets T_1 and T_2 based on averages across all input types and contruction methods.

	CNN	CapsNet
T_1	81.91%	72.96%
T_2	55.68%	53.73%

4.1 Object detection

Object detection was done to validate the potential of the networks to make classifications based on the created training data. Object detection was defined as classifying between two different classes: 'noObject' and 'object'. The 'object' class is composed of all other classes combined, hence, this class had predominately more training data compared to the class 'noObject'. The mean accuracy for CNN and CapsNet is 99.866% and 99.813% respectively on dataset T_1 and 91.163% and 87.721% respectively on dataset T_2. CNN performed better with 0.053 higher percentage points on T_1 and 3.442 percentage points better on T_2 than CapsNet. The average accuracy across all measurement sample sizes for each input type is portrayed in Table 4.2. Figure 4.1 also portrays the effect of changing the number of samples per measurement.

Table 4.2: Average classification accuracies on datasets T_1 and T_2 for object detection. Each accuracy is the average accuracy across the three sample sizes: 1600, 3300, 6000.

Input	T_1	T_2
CNN - complex baseband	99.736%	89.477%
CNN - spectrogram	99.995%	92.852%
CapsNet - complex baseband	99.769%	87.984%
CapsNet - spectrogram	99.857%	87.236%

4.2 Object classification

To give an overview, the average classification accuracies on the test datasets T_1 and T_2 can be seen in Table 4.3. The table gives only an overview. There are several outliers within the data where the network performed exceedingly well with one type of input compared to another. More will be explained when looking at some of the confusion matrices.

Table 4.3: Average classification accuracies on datasets T_1 and T_2 for object classification on the five objects described in Table 3.1 of Section 3.1.2.1 including the sixth class for "noObject". Each accuracy is the average accuracy across the three sample sizes: 1600, 3300, 6000.

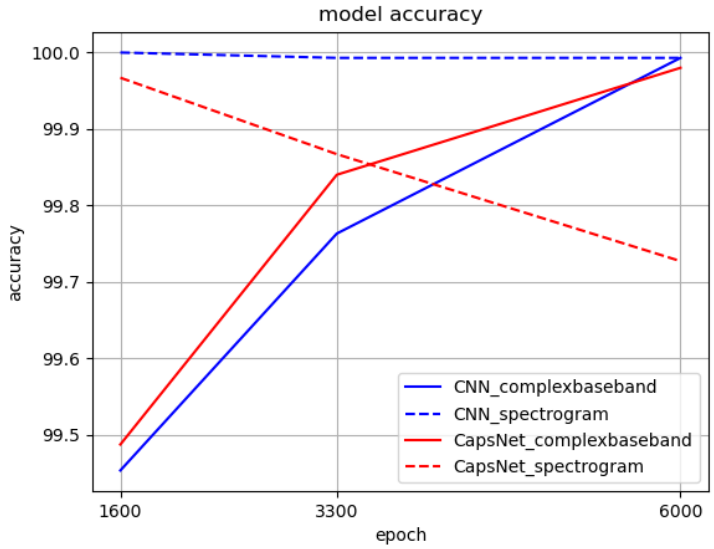
Input	T_1	T_2
CNN - complex baseband	85.34%	80.76%
CNN - spectrogram	78.48%	65.15%
CapsNet - complex baseband	67.26%	60.80%
CapsNet - spectrogram	44.10%	46.65%

Table 4.4: Complex baseband classification accuracies on dataset T_1.

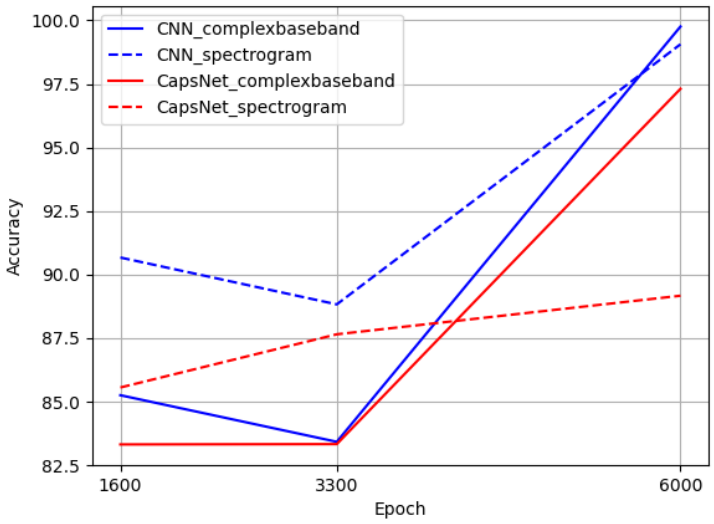
Complex Baseband	1600		3300		6000	
	CNN	CapsNet	CNN	CapsNet	CNN	CapsNet
Image	82.13%	1.805%	83.00%	64.47%	90.05%	77.09
Matrix	87.29%	80.30%	85.68%	74.06%	90.70%	79.81%
Stacked Matrix	81.84%	72.86%	82.49%	72.42%	84.90%	82.53%

Table 4.5: Complex baseband classification accuracies on dataset T_2.

Complex Baseband	1600		3300		6000	
	CNN	CapsNet	CNN	CapsNet	CNN	CapsNet
Image	75.20%	16.98%	71.47%	54.70%	92.30%	66.95%
Matrix	80.33%	69.61%	72.87%	65.02%	92.63%	68.07%
Stacked Matrix	77.00%	62.60%	74.57%	64.67%	90.50%	78.56%



(a) T_1



(b) T_2

Figure 4.1: Classification accuracy given both test datasets and the affect of changing measurement sample size.

Table 4.6: Spectrogram classification accuracies on dataset T_1.

Spectrogram	1600		3300		6000	
	CNN	CapsNet	CNN	CapsNet	CNN	CapsNet
Image	83.29%	76.00%	71.28%	65.15%	72.79%	40.62%
Matrix	79.65%	75.57%	73.18%	64.89%	73.95%	71.32%
Stacked Matrix	84.78%	1.093%	82.49%	1.134%	84.90%	1.093%

Table 4.7: Spectrogram classification accuracies on dataset T_2.

Spectrogram	1600		3300		6000	
	CNN	CapsNet	CNN	CapsNet	CNN	CapsNet
Image	69.96%	68.23%	65.13%	58.83%	68.64%	61.75%
Matrix	74.43%	62.14%	66.34%	56.67%	67.18%	61.14%
Stacked Matrix	85.57%	16.89%	85.13%	17.35%	85.09%	16.89%

4.2.1 Confusion Matrix and F1-score

The confusion matrices and F1-scores of the most successful and significant findings from testing will be presented in this section. The classes are one hot encoded with the following values:

- 0 - noObject
- 1 - Circle
- 2 - Square
- 3 - Rectangle
- 4 - Large Rectangle
- 5 - Triangle

The absolute highest performing network is the CNN network with the complex baseband input type based on the $N \times N \times 2$ matrix construction method with 6000 measurement samples. On test set T_1 and T_2 it achieved classification accuracies of 90.70% and 92.63% respectively. The confusion matrix and F1-scores in Tables 4.8 and 4.9 give more detail to these results. It is already clear that object detection is a trivial task and therefore one can see that the network correctly classified the class "noObject" 491 times and only miss-classified it 9 times, mistaking it for the "Large rectangle" 3 times and the "triangle" 6 times. The least successful classifications for this network is the "square" and "rectangle". When evaluating the F1-scores, one can refer to the confusion matrix, like in Table 4.8 where the columns determine the precision and the rows determine the recall. For example, looking at the model predictions for the object "square" in Table 4.8, one can see that when it predicted "square", it was correct 412 times but wrongly predicted "square" a total of 55 times yielding a precision of roughly $412/467 = 0.88$, which is the same value under precision for class "square" in Table 4.9. Similarly, the recall can be found in the row of Table 4.8 where the actual class is "square" and the corresponding model predictions. The model predicted the actual class "square" correctly 412 times but incorrectly 88 times which yields a recall value of roughly $412/500 = 0.82$, which also reflects the findings in 4.9 under recall for class 2. The highest performing CapsNet network is the network based on complexbaseband input type based on the

Table 4.8: Confusion matrix for the best performing CNN network based on complexbaseband input type based on the $N \times N \times 2$ matrix construction method with 6000 measurement samples on T_2. Columns: Model prediction. Rows: Actual class.

Class	0	1	2	3	4	5
0	491	0	0	0	3	6
1	0	470	3	2	4	21
2	0	0	412	46	6	36
3	0	0	49	435	8	8
4	0	0	2	14	480	4
5	0	0	1	2	6	491

Table 4.9: F1-scores for the best performing CNN network based on complexbaseband input type based on the $N \times N \times 2$ matrix construction method with 6000 measurement samples on T_2.

Class	precision	recall	F1-score
noObject	1.00	0.98	0.99
Circle	1.00	0.94	0.97
Square	0.88	0.82	0.85
Rectangle	0.87	0.87	0.87
Large Rectangle	0.95	0.96	0.95
Triangle	0.87	0.98	0.92

$N \times N \times 6$ stacked matrix construction method with 6000 measurement samples. On test set T_1 and T_2 it achieved classification accuracies of 82.53% and 78.56% respectively. The confusion matrix and F1-scores below in Tables 4.10 and 4.11 gives more detail to these results.

4.2.2 Effects of input type and sample size

Utilizing the highest performing network, the affects of changing the input type and the sample size can be analyzed. The following results stem from the best performing network introduced in Section 4.2.1 of the CNN with complexbaseband input type based on the $N \times N \times 2$ matrix construction method with 6000 measurement samples.

Decreasing the measurement sample size results in a decrease in performance on both test datasets T_1 and T_2, however, the smallest measurement sample size has slightly higher accuracy compared to 3300.

Although the construction method of using the $N \times N \times 2$ matrices proved most effective, Table 4.13 shows promising results from the other methods as well. Both the image and 2-channel matrix construction methods performed nearly identically, while the 6-channel method performed significantly worse by comparison.

Table 4.10: Confusion matrix for the best performing CapsNet network based on complexbaseband input type based on the $N \times N \times 6$ stacked matrix construction method with 6000 measurement samples on T_2. Columns: Model prediction. Rows: Actual class.

Class	0	1	2	3	4	5
0	189	0	0	34	36	222
1	0	459	2	0	2	10
2	0	0	386	38	4	53
3	0	0	104	337	15	15
4	0	0	6	15	437	9
5	0	1	46	1	8	421

Table 4.11: F1-scores for the best performing CapsNet network based on complexbaseband input type based on the $N \times N \times 6$ stacked matrix construction method with 6000 measurement samples on T_2.

Class	precision	recall	F1-score
noObject	1.00	0.39	0.56
Circle	1.00	0.97	0.98
Square	0.71	0.80	0.75
Rectangle	0.79	0.72	0.75
Large Rectangle	0.87	0.94	0.90
Triangle	0.58	0.88	0.70

Table 4.12: Effects of changing the measurement sample size on test datasets T_1 and T_2 for the best performing CNN.

Measurement Samples	T_1	T_2
1600	87.29%	80.33%
3300	85.68%	72.87%
6000	90.70%	92.63%

Table 4.13: Effects of changing the construction method on test datasets T_1 and T_2 for the best performing CNN. Image has dimension $N \times N \times 3$, 2-channel has dimension $N \times N \times 2$ and 6-channel has dimension $N \times N \times 6$.

Construction Method	T_1	T_2
Image	90.05%	92.30%
Matrix	90.70%	92.63%
Stacked Matrix	84.90%	90.50%

5

Discussion

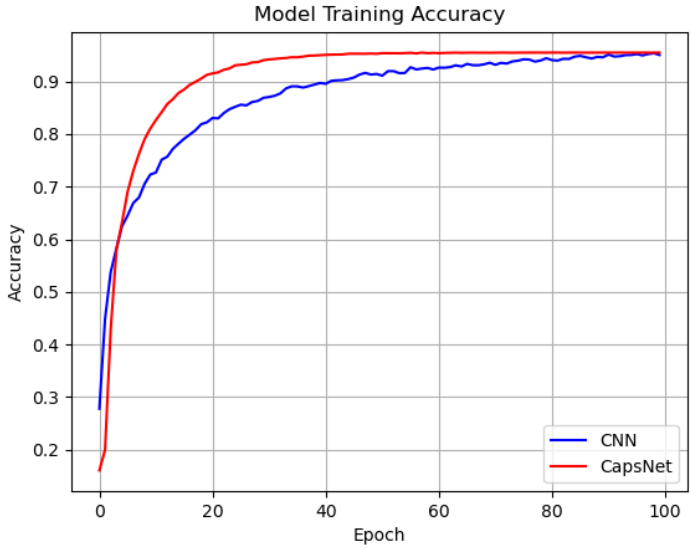
The following section presents a study of the methods and results reflecting on the performance of the networks. A comparison between the performance of CNN and CapsNet given the data collected is made. The discussion continues to address the effects of using a complex baseband input versus a spectrogram input detailing the pros and cons of both in regards to classification based on raw ultrasonic signals. The effects of filtering the data and increasing/decreasing sample size and how it may have affected the results. Lastly, a different approach to classification based on ultrasonics is suggested as potential future work in this field of research.

5.1 Network performance and comparison

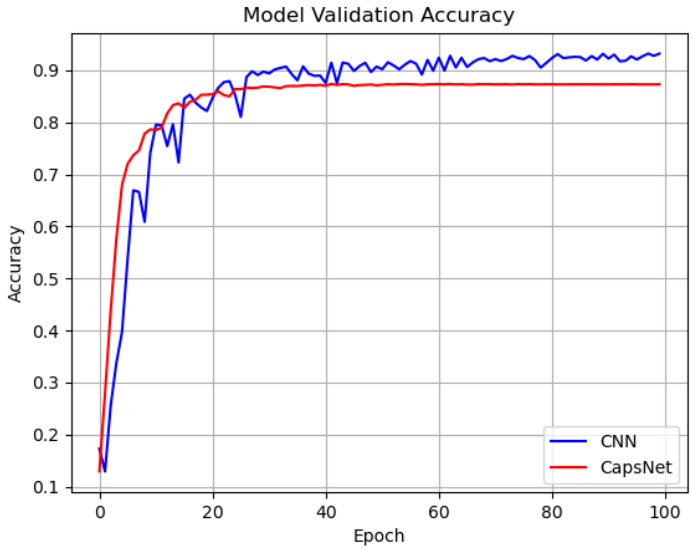
The results prove that there is strong potential in object classification based only on raw ultrasonic signals. The best networks from both CNN and CapsNet give results of 92.63% and 78.56% respectively based on test set T_2 which represents never-before seen data. The CNN network marginally outperforms the CapsNet in regards to object detection with just two classes. Given six classes, the CNN network generally performed better overall. In the cases where performance was not good, the indication is that the network is simply unable to properly correlate features within the input data and may require re-evaluating the contents of the training data, or simply collecting more training data.

In general, both CNN and CapsNet performed better on test dataset T_1 compared to T_2 which may not come as a surprise since the data in T_2 is much more susceptible to being slightly different from the training data therefore giving a good idea on the networks' robustness and ability to generalize. Another reason the networks perform better on T_1 is due to the fact that the data in T_1 is collected in the exact same setup and day as the training data, therefore more representative of the training data, but not actually apart of the training data. The networks may be approaching the entire classification problem the wrong way and are learning to classify the object and its current environment rather than the object itself. This also factors in the reason for lower accuracy on test set T_2 and potentially the reason for the networks struggle to generalize.

The nature of how CNN and CapsNet converge on their training and validation accuracies are significantly different. Given the results, CNN tends to be less consistent or smooth in regards to its convergence in both training and validation accuracy as shown in Figure 5.1. On the other hand, CapsNet clearly has a much smoother convergence and hints towards a sense of confidence in its classification and avoids premature convergence when given six classes. This can also be seen when comparing the training and validation loss, like in Figure 5.2, however, it is important to note that CNN uses categorical cross entropy loss whilst CapsNet uses MSE, therefore, the values are different but the nature of their convergence can still be compared.

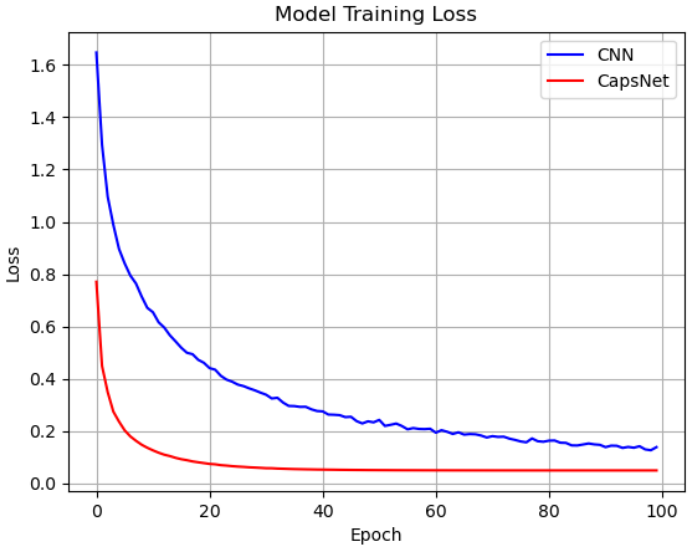


(a)

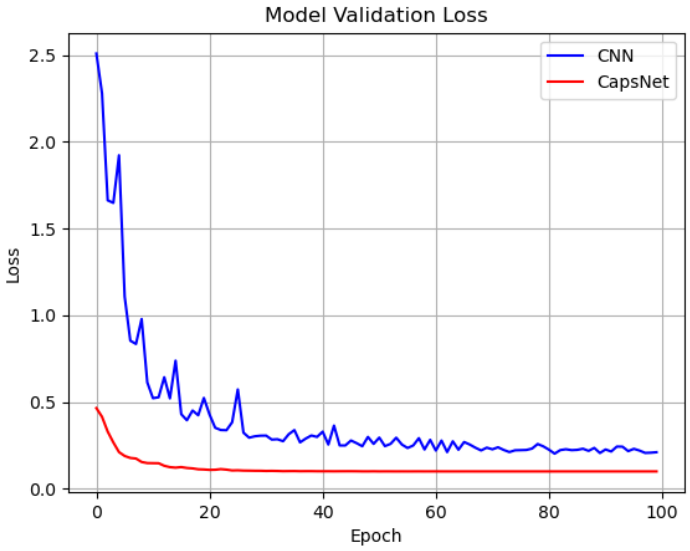


(b)

Figure 5.1: Training and validation accuracy of CNN and CapsNet using complex baseband input type, 6-channel construction method, and 1600 measurement samples.



(a)



(b)

Figure 5.2: Training and validation loss of CNN and CapsNet using complex baseband input type, 6-channel construction method, and 1600 measurement samples.

5.2 Complex baseband vs. spectrogram input

The general hypothesis in regards to the performance of the complex baseband versus the spectrogram type of input was that the spectrogram input is more ideal since it is essentially composed of more information. On top of this, the spectrogram has information in regards to the change in frequency over time which may give insight towards the objects geometry. As the results show, this is not the case and therefore

the complex baseband input type performed better. Overall, the better performing input type is complex baseband which performed on average 15.01 percentage points higher than spectrogram on T_1 and 10.93 points higher on T_2. The issue with the spectrogram input type may stem from several different issues. Due to the higher dimensionality of the spectrogram input, the networks have considerably more parameters and may require more training data compared to the complex baseband input whose input size is significantly smaller. On the other hand, there may be a fundamental issue with the creation of the spectrogram input since combining a series of 10 measurements in the way described in Section 3.4.2 may not prove as effective as the complex baseband input type where each training sample is represented by only a single measurement.

In general, for both the complex baseband and spectrogram input types, it is hypothesized that the networks utilized mostly the magnitude of the frequencies present in the input data. This can give a direct understanding of how close the current face of the object is and some insight on the objects surface geometry at any given instance. It can also be seen in the results that images performed generally poorly compared to just using matrix input. This could be due to the fact that the images were created with a blue channel consisting of zeros which essentially added dimensionality but no extra information.

5.3 Data collection and filtering

Much of the data collection and filtering processes are heavily influenced from the procedures in [29]. The project in [29] is a very similar project and attempts at collecting ultrasonic data and fitting it into a very manageable size. Convolutional neural networks are a fundamental type of network that has proven successful in a vast amount of projects and therefore was an attractive solution to this classification problem. The data collection process was done with network training time in mind. A smaller input size would allow for much more iterations of network training and evaluation compared to using much larger input sizes which carry many more learnable parameters. A major weakness of the measurement process is the quality of the ultrasonic sensors. The sensors used have very limited detection range and are much more exposed to noise compared to the ultrasonics used in the production of the Volvo cars. A higher quality ultrasonic with a combined transmitter, receiver could greatly increase performance. This project is aimed to be reproducible with easily accessible tools and parts, therefore, budget sensors were used.

5.4 Future work

An interesting project to build upon this project is to explore the potential of classifying material types using solely ultrasonic sensor data. Is it possible to classify the difference between fabric and a concrete wall? or a bush using similar methods as this project? If classification can be successful is this front, then it could help create a path towards scaling the project up towards real world scenarios and testing on

the vehicle.

In regards to data collection, an entire investigation on the most effective and efficient ways of creating training data should be done. The most important aspect of a successful neural network is the quality of the training data. This project only collected data from a fixed distance to the object, however, it would be interesting to see how the results change given a further distance to the objects. Also, this project examines the potential of one method of creating training data through images and 3D matrices, however, there are methods such as traditional multi-layer perceptrons and recurrent neural networks, RNN [30], which could outperform the results given in this project. RNNs take advantage of an internal state (memory) to process input with variable length sequences of inputs. Ultimately, these functions will need to run in real-time, therefore it is important to keep previous frames in memory to help strengthen classification predictions. More information on RNNs and its potential can be read in [30]

Lastly, exploring the potential of combining ultrasonic data with another type of sensor, such as radar, could yield improved classification results and increase robustness. The weaknesses of ultrasonics could be counter by the strengths of another sensor to allow for a much better classification of objects in close proximity to the vehicle at low speeds. In reality, it may be more effective to use a different type of sensor such as radar to produce better results, but more research and testing must be done with radar to support these claims.

6

Conclusion

In summary, there is strong evidence that machine learning algorithms can successfully perform object classification when only given raw ultrasonic sensor data. It is shown that performance is strong for CNN, but lower for CapsNet. Out of all the 36 trained networks, the best is CNN which produced a classification accuracy of 93% on T_2. The advantages that capsules have compared to max pooling do not appear to give CapsNet better results over CNN and is most likely due to nature of the data collected not consisting features of significant pose changes which may be prevalent in other applications.

The spectrogram input type, given its larger dimension and information, does not perform as well as the smaller complex baseband input type, most likely, due to how the spectrogram data was created and possibly the need for more training data. More training data for the spectrogram input type may bolster its performance, but it may be more effective to create a separate spectrogram for each measurement rather than creating a spectrogram with ten consecutive measurements. Doing this may provide more information since the spectrogram will only contain one measurement at a single time instance.

Of the three construction methods, both the image and matrix methods yielded similar results and are both viable options to use for future experimentation. The stacked matrix form containing six channels seemed promising and simplified the network architecture but in the end did not perform as well as the image and matrix methods.

This study also shows that providing more samples per measurement tends to produce better results, however, the 1600 measurement sample size has potential to produce relatively good results given low dimension size which is advantageous for fast training time, but the overall recommendation is to use at least 6000 samples per measurement. The filtering process does not seem to have a significant effect on the performance of the networks, besides basic bandpass filtering, when considering the other factors that effect performance, such as training dataset size, input type, and construction method. In future work, more emphasis should be put on collecting more data and building upon CNN and the matrix/image construction methods.

Using effective evaluation metrics is vital. Classification accuracy is a good high-level evaluation of a networks performance, however, there lies more information in the corresponding confusion matrices and F1-scores. It is strongly recommended to test future trained networks on data unseen by the networks that do not include data collected in the same setting as the training data in order to get a good indication of the network's robustness to slight changes in the input.

Bibliography

[1] E. Bachman, "Tesla Deaths," TeslaDeaths.com, 23-Dec-2019. [Online]. Available: https://www.tesladeaths.com/. [Accessed: 17-Jan-2020].

[2] M. Lynberg, "Automated Vehicles for Safety," NHTSA, 13-Apr-2020. [Online]. Available: https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety. [Accessed: 18-Jan-2020].

[3] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic Routing Between Capsules," arXiv.org, 07-Nov-2017. [Online]. Available: https://arxiv.org/abs/1710.09829. [Accessed: 17-Jan-2020].

[4] P. Pickering, "Radar and Ultrasonic Sensors Strengthen ADAS Object Detection," ElectronicDesign, 22-Aug-2017. [Online]. Available: https://www.electronicdesign.com/markets/automotive/article/21805470/radar-and-ultrasonic-sensors-strengthen-adas-object-detection. [Accessed: 17-Jan-2020].

[5] X. E. Gros, "Non-destructive Testing Techniques," in NDT Data Fusion, Elsevier, 1997, pp. 43–81.

[6] "Stress Coupled Phenomena: Piezoelectric Effect," in Encyclopedia of Materials, 2nd ed., Elsevier, 2001, pp. 8894–8897.

[7] S. N. Ahmed, "Signal processing," in Physics and Engineering of Radiation Detection, 2nd ed., Elsevier, 2015, pp. 477–540.

[8] "Signal Processing," in Applied Underwater Acoustics, Elsevier, 2017, pp. 743–807.

[9] "How to Convert the Analog Signal to Digital Signal by ADC Converter," ElProCus, 08-Oct-2014. [Online]. Available: https://www.elprocus.com/analog-to-digital-adc-converter/. [Accessed: 18-Mar-2020].

[10] "Capturing images," University of California, Berkeley. [Online]. Available: http://microscopy.berkeley.edu/courses/dib/sections/02Images/sampling.html. [Accessed: 18-Mar-2020].

[11] H. Austerlitz, "Analog/Digital Conversions," in Data Acquisition Techniques Using PCs, 2nd ed., Elsevier, 2003, pp. 51–77.

[12] S. Arar, "FIR Filter Design by Windowing: Concepts and the Rectangular Window," All About Circuits, 12-May-2016. [Online]. Available: https://www.allaboutcircuits.com/technical-articles/finite-impulse-response-filter-design-by-windowing-part-i-concepts-and-rect/. [Accessed: 17-Feb-2020].

[13] W. Abu-Al-Saud, "Baseband vs. Passband Communication Systems," Baseband vs. Passband Communication Systems, 2007. [Online]. Available: https://faculty.kfupm.edu.sa/EE/wajih/files/EE 370/EE 370, Lecture 07.pdf. [Accessed: 18-Mar-2020].

[14] "hamming," MathWorks Nordic. [Online]. Available: https://se.mathworks.com/help/signal/ref/hamming.html. [Accessed: 25-Mar-2020].

[15] H. T. Toivonen, "Multirate digital signal processing," Åbo Akademi University, 20-Jan-2009. [Online]. Available: http://users.abo.fi/htoivone/courses/sbappl/asp_chapter2.pdf. [Accessed: 26-Mar-2020].

[16] "Baseband Equivalent of Passband Signals," ECEN 661 at Texas AM Univ, 22-Jan-2011. [Online]. Available: http://ecen661tamu.wikidot.com/basebandequivalent. [Accessed: 02-Apr-2020].

[17] F. R. Kschischang, "The Hilbert Transform," Uni-

versity of Toronto, 10-Mar-2015. [Online]. Available: https://www.comm.utoronto.ca/frank/notes/hilbert.pdf. [Accessed: 14-Mar-2020].

[18] N. S. Keskar, "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima," Cornell University, 15-Sep-2016. [Online]. Available: https://arxiv.org/abs/1609.04836. [Accessed: 24-Apr-2020].

[19] D. P. Kingma and J. Ba, "Cornell University," Cornell University, 22-Dec-2014. [Online]. Available: https://arxiv.org/abs/1412.6980. [Accessed: 20-Mar-2020].

[20] Ioffe, S., Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shif. Google Inc. [Online] Available: https://arxiv.org/pdf/1502.03167.pdf. [Accessed: 26-Mar-2020]

[21] Seo, J. D. (2018, November 17). Batch Normalization and Internal Covariate Shift. [Online] Available: https://medium.com/@SeoJaeDuk/archive-post-batch-normalization-and-internal-covariate-shift-1d47661d236f [Accessed: 10-Apr-2020]

[22] Hinton, G. E., Sabour, S., Frosst, N. (2017). Dynamic Routing Between Capsules. Cornell University. [Online] Available: https://arxiv.org/abs/1710.09829 [Accessed: 14-Feb-2020]

[23] Bi, R. (2014, December). Geoffrey Hinton talks about Deep Learning, Google and Everything. [Online] Available: https://www.kdnuggets.com/2014/12/geoffrey-hinton-talks-deep-learning-google-everything.html [Accessed: 10-Mar-2020]

[24] Mishra, A. (2018, November 1). Metrics to Evaluate your Machine Learning Algorithm. [Online] Available: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234 [Accessed: 30-Apr-2020]

[25] Technical datasheet: KPUS-40T-16T-K768. (n.d.). [Online] Available: https://cdn.sos.sk/productdata/e0/32/1c2b05f6/t4016-ust-40-t.pdf [Accessed: 20-Jan-2020]

[26] Technical datasheet: KPUS-40T-16R-K769. (n.d.). [Online] Available: https://cdn.sos.sk/productdata/0f/2b/d03deb63/r4016-ust-40-r.pdf [Accessed: 20-Jan-2020]

[27] House, J. (2010, September 28). PicoScope 3425 PC Oscilloscope: User's Guide. [Online] Available: https://www.picotech.com/download/manuals/ps3425-en-3.pdf [Accessed: 25-Jan-2020]

[28] 33500B and 33600A Series Trueform Waveform Generators. (2020, April 28). [Online] Available: https://www.keysight.com/us/en/assets/7018-05928/data-sheets/5992-2572.pdf [Accessed: 15-May-2020]

[29] Valeo Schalter und Sensoren GmbH. (2019). Capsule Neural Network based Height Classification using Low-Cost Automotive Ultrasonic Sensors. [Online] Available: https://arxiv.org/pdf/1902.09839.pdf [Accessed: 28-Jan-2020]

[30] "Recurrent Neural Networks cheatsheet," Recurrent Neural Networks. [Online]. Available: https://stanford.edu/ shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks. [Accessed: 10-May-2020].

[31] Admin, AI Graduate. "Audio Classification Using CNN-Coding Example." Medium, Medium, 22 Mar. 2019, [Online] Available: medium.com/x8-the-ai-community/audioclassification-using-cnn-coding-example-f9cbd272269e. [Accessed: 3-Feb-2020]

[32] LeCun, Yan, et al. "THE MNIST DATABASE." Yann.lecun.com, [Online] Available: yann.lecun.com/exdb/mnist/. [Accessed: 4-Mar-2020]

[33] "What Is a CapsNet or Capsule Network?" Hackernoon, 31 Oct. 2017, [Online] Available: hackernoon.com/what-is-a-capsnet-or-capsule-network-2bfbe48769cc. [Accessed: 5-Feb-2020]

[34] Debarko. "A Tensorflow Implementation of CapsNet (Capsules Net) in Hinton's Paper Dynamic Routing Between Capsules ." GitHub, 1 Nov. 2017, [Online] Available: github.com/debarko/CapsNet-Tensorflow. [Accessed: 8-Mar-2020]

[35] Frisk, D. (2018) A Chalmers University of Technology Master's thesis template

for LaTeX. Unpublished.

A

Appendix 1

A.1 Datasets

	noObject	circle	square	rectangle	large rectangle	triangle
T_1	50 (1.67%)	590 (19.66%)	590 (19.66%)	590 (19.66%)	590 (19.66%)	590 (19.66%)
T_2	500 (16.67%)	500 (16.67%)	500 (16.67%)	500 (16.67%)	500 (16.67%)	500 (16.67%)

Table A.1: Distribution of test data per class of each test set T_1 and T_2.

A.2 Network architectures

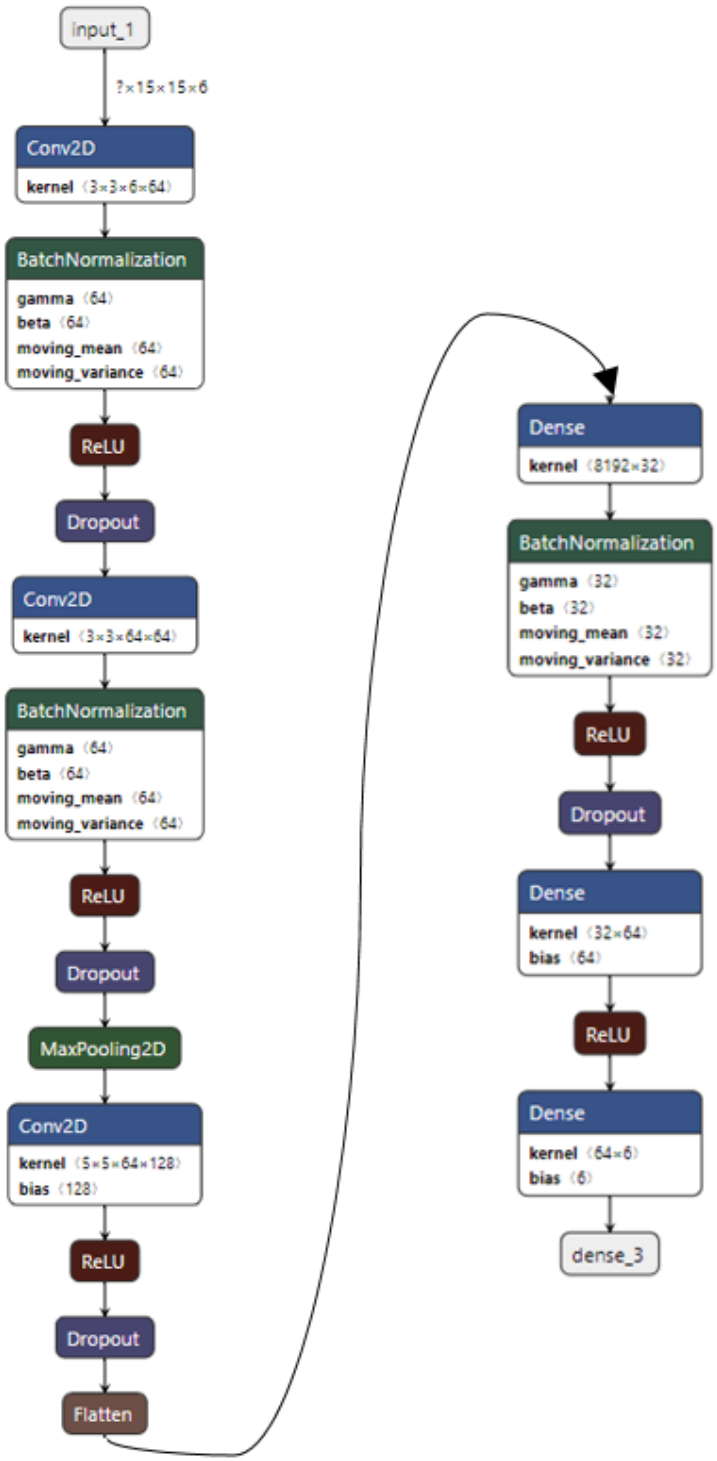


Figure A.1: The CNN architecture with the parallel structure used for the image and matrix input data.

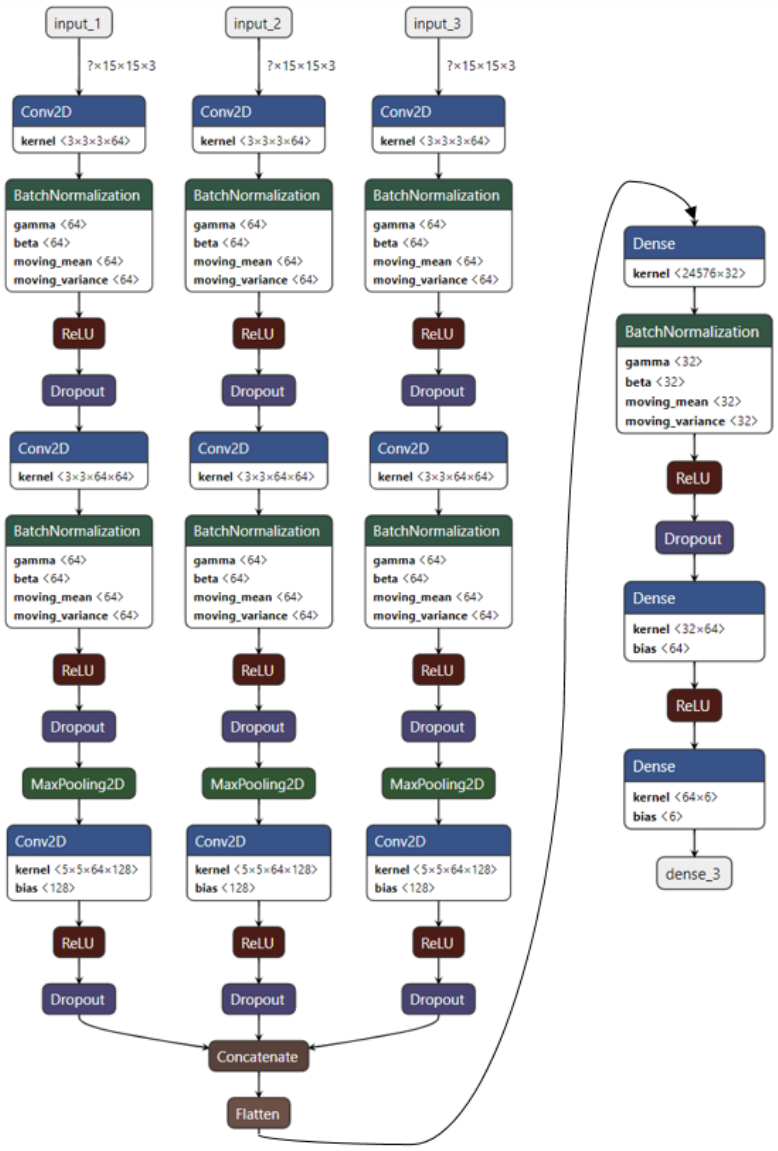


Figure A.2: The traditional CNN architecture without the parallel structure used for the 6-channel input.

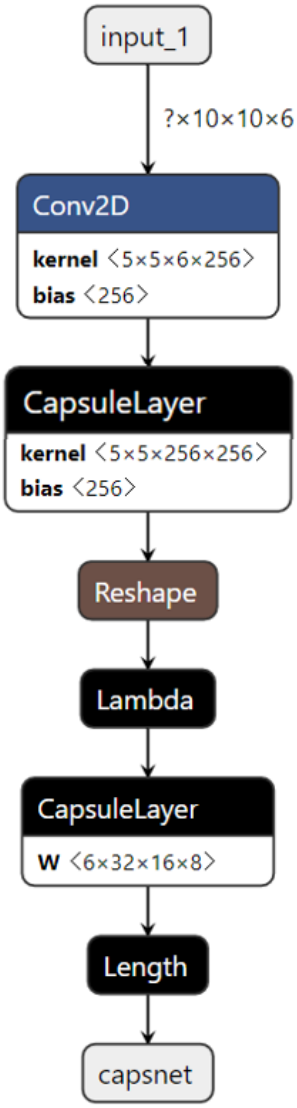


Figure A.3: The standard CapsNet architecture based on Hinton's paper in [22] with the reduced kernal size of $5x5$. Used for the 6-channel input.

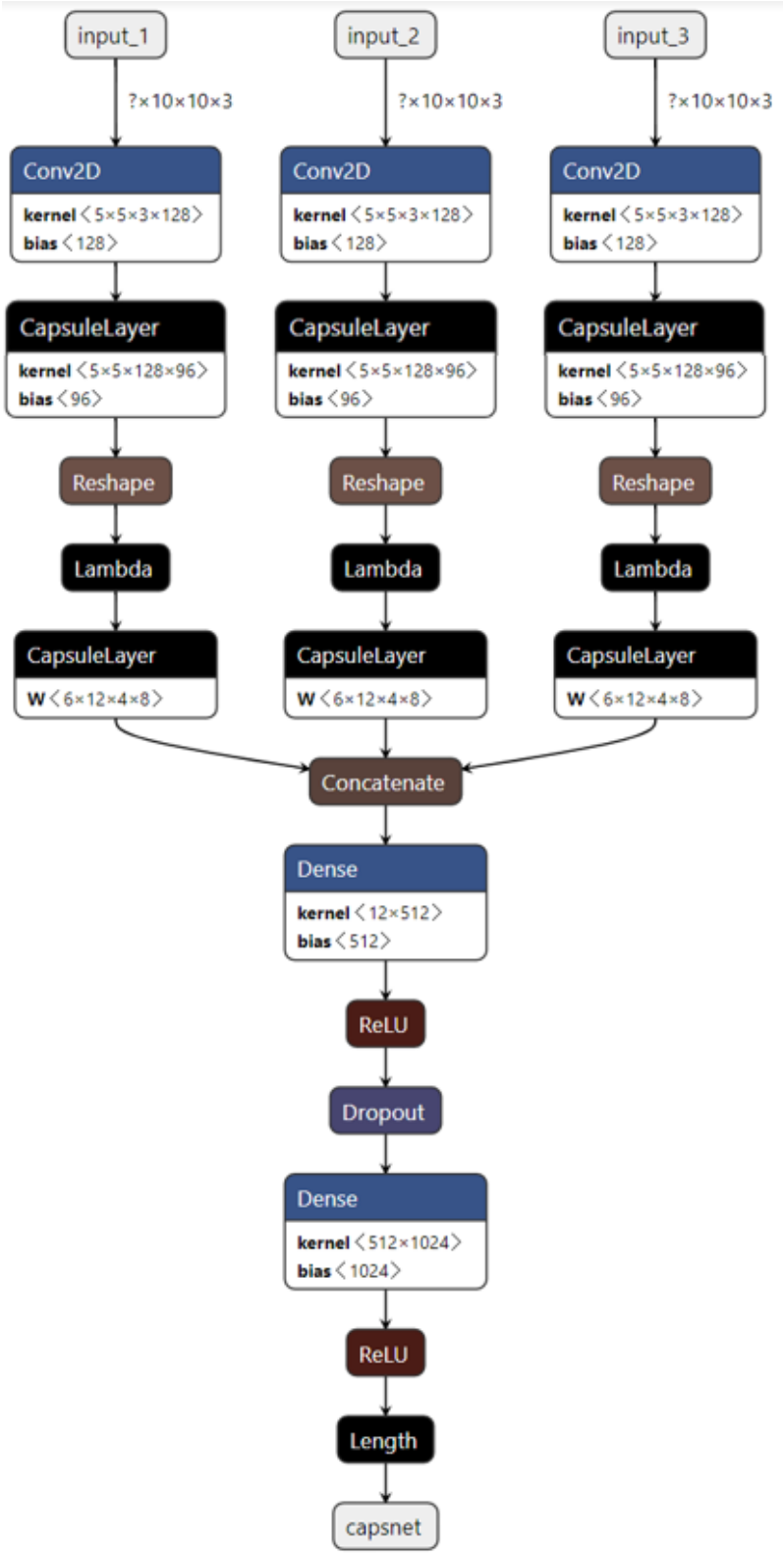


Figure A.4: The CapsNet architecture with the parallel structure used for the image and matrix inputs.