Implementation of a neural network on a microcontroller for recognition of warning signals

Anders Liss
Abstract

Implementation of a neural network on a microcontroller for recognition of warning signals

Anders Liss

The development of neural networks is expanding onto platforms with a lesser computational power such as microcontrollers. As a result of this development, solutions from neural networks can be compressed and implemented on everyday products. The microcontrollers physical footprint is relatively small compared to computers can be used in order to create “smart” products, which take in and process sensor data from the surrounding, possible in combination with a neural network.

In this thesis, a deep neural network (DNN) was implemented on an STM32F746NG microcontroller unit in order to primarily recognize the Yaris car horn. The network classified the car horn with adequate accuracy and with a latency of 120ms. This was a result of experiments in order to evaluate four different neural networks (deep neural network, convolutional neural network, convolutional recurrent neural network, depthwise separable convolutional neural network). The networks were trained on a computer with a data set created during the project and implemented on the microcontroller unit.
Sammanfattning

Utvecklingen av neurala nätverk har utvidgats till hårdvaru-platformar med mindre beräkningskapacitet, t.ex. enkapseldatorer. Ett resultat av denna utveckling är att neurala nätverk kan reduceras och implementeras på vardagliga produkter. Enkapseldatorer, vars fysiska storlek är liten jämfört med datorer kan användas för att utveckla ”smarta” produkter som kan ta in och hantera sensordata från omgivningen, möjligtvis i kombination neurala nätverk.

Contents

1 Introduction 7
1.1 Project Purpose ........................................ 7
1.2 Project Goal ........................................... 7
1.3 Limitations ............................................ 8
1.4 Report overview ...................................... 8
1.5 Related Work .......................................... 9

2 Theory 10
2.1 Deep learning neural networks ....................... 10
  2.1.1 Convolutional layer ................................ 11
  2.1.2 Activation function ................................ 13
  2.1.3 Pooling layer ..................................... 13
  2.1.4 Dense layer ........................................ 15
  2.1.5 Neural networks implemented with limited processing power .................................... 15
  2.1.6 Measuring classification accuracy ............... 16
  2.2 Training the network .................................. 17
    2.2.1 Backpropagation .................................. 18
    2.2.2 Gradient descent .................................. 18
    2.2.3 Hyperparameters .................................. 19
  2.3 Signal processing .................................... 19
    2.3.1 Spectrogram ...................................... 20
    2.3.2 The mel scale ...................................... 21
  2.4 STM32F746g discovery board ......................... 22
  2.5 Active Noise Cancellation with AS3415 ............ 22

3 Implementation 24
3.1 Creating the data set ................................ 24
  3.1.1 Audio recording & editing ....................... 25
  3.1.2 Labeling the data set ............................ 26
  3.1.3 Spectrograms ..................................... 29
  3.1.4 Iterations in data sets ........................... 30
  3.2 Evaluated networks .................................. 31
  3.3 Recognition of multiple warning signals .......... 32
  3.4 Creation & training of the network ................. 34
  3.5 STM32 Architecture ................................. 35
    3.5.1 Board Specific Package (BSP) .................. 36
    3.5.2 STM32_AudioPreprocessing_Library ........... 36
3.5.3 CMSIS ........................................... 37
3.5.4 STM32Cube & X-Cube AI ....................... 37
3.6 Preprocessing audio input .......................... 37
3.7 Optimizing performance ............................ 38
  3.7.1 Runtime ........................................ 38
  3.7.2 Accuracy ........................................ 39

4 Experiment ........................................ 39

5 Results ............................................ 41
  5.1 Classification of a single warning signal ......... 41
  5.2 Classification of multiple warning signals ...... 43
    5.2.1 Multiple neural networks ..................... 43
    5.2.2 Single neural network classifying multiple warning signals ................. 45

6 Discussion ......................................... 47
  6.1 Relevance of experiment results .................. 47
  6.2 Experiment setup ................................. 49
  6.3 Incoherence between training data and actual data .. 49
  6.4 Performance of the system ........................ 50
  6.5 Classification of multiple warning signals and scaling the system ............ 50
  6.6 Choice of neural network ........................ 50

7 Conclusion ......................................... 51
List of Abbreviations

MCU - Microcontroller unit
STM32 - Microcontroller platform created by STMicroelectronics
STM32 MCU - Project-specific microcontroller: STM32F746NG
IOT - Internet of things
ReLU - Rectified linear unit
FC - Fully connected
Conv - Convolutional
DS-Conv - Depthwise separable convolutional
STFT - Short time Fourier transform
FPU - Floating point unit
MEMS - Microelectromechanical system
LCD - Liquid Crystal Display
SRAM - static RAM
ANC - Active noise cancellation
CNN - Convolutional neural network
GRU - Gated recurred unit
DNN - Deep neural network
LSTM - Long short-term memory
CRNN - Convolutional recurrent neural network
DS-CNN - Depthwise separable convolutional neural network
BSP - Board specific package
PDM - Pulse density modulation
PCM - Pulse code modulated
SAI - Serial audio interface
DMA - Direct memory access
CMSIS - Cortex Microcontroller Software Interface Standard
AI - Artificial Intelligence
1 Introduction

1.1 Project Purpose

The development of neural networks are expanding onto platforms with a lesser computational power such as microcontrollers [10]. As a result of this development, solutions from neural networks can be compressed and implemented on everyday products. These neural networks are able to analyse data and can be trained to recognize certain patterns. From this, the networks are able to make predictions. If the network is trained well and able recognize patterns correctly, the result is a product with the ability to react to its surrounding without the user having to actively provide input; a smart device. This thesis project is based on the implementation a neural network in order to construct a proof-of-concept for a pair of smart headphones. With noise cancelling headphones, the listener is at risk for being taken out of his or her surrounding by isolating all outside sounds. Implementing a neural network to recognize warning signals, for example a car horn, the headphones could react to the warning signal and keep the user aware of the surrounding.

Figure 1: Graphic overview of proof-of-concept. A pair of headphones with active noise cancellation of the surrounding. When recognizing a warning signal, in the form of a car horn, the headphone throws a bypass switch to the active noise cancelling filter, allowing the listener to be aware of his or her surrounding.

1.2 Project Goal

The goal of this project is to evaluate a variety of neural networks in order to find the most relevant network type to use in the classification of warning signals. A number of types of networks will be evaluated, such as deep neural networks and a number of convolutional networks. The network which is
most suited will then be implemented on a STM32F746NG microcontroller unit (MCU) for real time classification of audio input.

1.3 Limitations
As a proof of concept and a way of measuring performance the project will primarily focus on the successful classification of one type of warning signal: the car horn of a Toyota Yaris. Following this, the opportunity for scaling the solution to further warning signals will be evaluated. The key factors in the evaluation will be the performance of the system and the observed accuracy of the predictions of the network. As a way of reducing the latency of the system for use in real life, consideration has to be taken to limit the time from audio input to the reaction of the system to a warning signal. The goal of this thesis is to achieve a latency of 100ms from warning signal to the reaction of the system. This project goal raises questions aimed to be answered within this paper. Which type of neural network is best suited for systems with limited processing power while still resulting in an accepted level of predicted accuracy? What is a reasonable overhead of time from audio input to system response?

1.4 Report overview
The aim of the theory of this report is to give a high level description of the fields of study needed in creating this proof of concept. At first theory behind neural networks will be covered: the overall structure, individual layers, process of training, and some statistical theory behind understanding the predictions made by the network, and also the case of false positive predictions. Further, the report will describe the field of implementations of neural networks on microcontrollers in order to place the thesis project within its scope. Following this, some theory behind signal processing will be described, especially the fields of Fourier transform, spectrograms and frequency analysis of audio. Finally, the functionality of an ready-made solution for noise cancellation is presented. The reason is to present a way of creating a product containing the noise cancelling functionality in combination with the neural network classification.

The implementation section outlines the process from creating the data sets used in training the network to implementing the network on the STM32F746NG MCU. During the time of the project, many different forms of data sets and neural networks have been evaluated, and these are presented here. In order to gather data in evaluation the performance and accuracy of the networks,
an experiment is outlined. The results from experiment are then presented. The results are used to answer the questions and problems raised in the introduction. Finally, discussion regarding the outcome of the project, potential improvements and how one would build on this project will be presented before concluding the whole of the project.

1.5 Related Work

The cross-section of deep neural networks and embedded system and IOT solutions consists of many points of research. Amazon, Google and arm has released papers evaluating the implementation of neural networks on microcontrollers in order to perform keyword spotting \[15\] \[7\] \[2\]. One key paper for this thesis was "Hello Edge: Keyword spotting on Microcontrollers" [26] which provided the framework for the type of networks to be evaluated in this thesis.
2 Theory

2.1 Deep learning neural networks

Deep neural networks are computational models composed of multiple layers consisting of nodes [11]. The layers are connected to each other via individual connections between the nodes, each containing a weight: value which signifies the strength between the connection of the two nodes. These layers, nodes and connections construct models with multiple levels of abstraction [11]. The weights of the connections are tuned in a process called backpropagation. This process is done by minimizing the error of the network by slightly altering the value of the nodes in the models. This tuning of weights is done continually during the training of the network.

A convolutional neural network (CNN) is a type of neural network built with convolutional layers, and have been effectively used within image recognition [9]. An overview of a CNN can be seen in fig. 2. The input of the convolutional neural network consists of a grid of values [4, p. 326]. The input data is convoluted with different filters, relating to different weights as described in eq. (1), where \( \beta_i \) are the weights between the connection between the layers and the \( x \) value indicates the value within the node. Along with the node value and the weight, a bias \( \epsilon \) is added. The \( \sigma \) variable denotes the non-linear activation function, often sigmoid or ReLU, described in section 2.1.2.

\[
y = \sigma(\beta_1 x_1 + \beta_2 x_2 ... \beta_3 x_3 + \epsilon)
\]  

(1)

Figure 2: Overview of a neural network, where a vector of values are convoluted with the abstract layers. Each node represents the sum of every node on a previous layer, with a scalar weight \( a_i \), \( b_i \) multiplied. The final layer is the prediction layer, where in the case of this project, the prediction will be either "car horn" or "no car horn".
2.1.1 Convolutional layer

The convolutional layer has the capability of finding patterns, or "detecting local conjunctions of features from the previous layer" [11, p. 439]. The kind of pattern found depends on the filter used and is generally specified by the learning algorithm in order to optimize the result [4, p. 329]. The filter is also called the kernel, and is a smaller matrix. One example of a kernel can be seen in fig. 3.

\[
\begin{array}{cc}
    a_1 & a_2 \\
    a_3 & a_4 \\
\end{array}
\]

Figure 3: A kernel used in convolution of the ingoing data. The values of \(a_i\) are chosen in order to recognize different patterns.

The example in fig. 4 shows the process of the kernel sliding through an matrix with values detailing the darkness of the pixel. This is the summation of values of the pixels, multiplied by scalar weights with a bias referred to in 1. In this case, a higher value at \(x_i\) could for example translate to a darker pixel. Sliding across the matrix, the final result is another matrix with equal amounts of values. This is one layer of a convolutional filter.
Figure 4: The process of convolution. The kernel slides through all pixels, from top to bottom of the matrix and sums the values as described in eq. (1).

However, the kernel can differ in appearance and the values within the kernel decide what type of patterns the convolution picks up on. The values in the kernel are iteratively optimized in order to for the layers to provide accurate pattern recognition. In order to optimize the accuracy of the neural network, several filters can be used in convolution. This will create several layers, each recognizing different patterns, as shown in fig. 5.

Figure 5: Different layers of convolution, as a result of different filters processing the ingoing data. The difference of the kernels implies different types of pattern recognition.
2.1.2 Activation function

In order to efficiently train a network by reducing the cost function via backwards propagation, a linear model could be appealing [4, p. 165]. A linear model may provide efficient optimization for simpler problems, but for more complex problems the nonlinear functions can provide a better estimation. The linearity of the neural network can be attributed to the activation functions implemented within [4, p. 170]. The basic functionality of the activation function is to provide a universal way in the network of assessing the relative level of intensity. A key factor when training a network is choosing an activation function. Two common activation functions to use with networks such as the convolutional network is the Sigmoid function [13, p. 3] and the rectified linear activation function (ReLU). [4, p. 170]

\[ \text{Sigmoid}: \sigma(x) = \frac{1}{1 + e^{-x}} \quad (2) \]

\[ \text{ReLU}: \sigma(x) = \max(0, x) \quad (3) \]

\[ \text{Softmax}: \sigma(z) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad (4) \]

As of today, the ReLU is the activation function recommended to use within convolutional neural networks [4, p. 170]. The reason is the importance in the training of the network and the tuning of the weights [9, p. 3]. Outside of the hidden layers, a softmax function can be used in order to represent a probability distribution over a discrete variable with \( n \) possible values [4, p. 180]. This will range the summation of data from 0 to 1, and can be seen as the predicted output of the neural network, see eq. (4).

2.1.3 Pooling layer

Increasing performance without a reduction in accuracy is a key factor when designing an effective neural network. The so called pooling layer can be used in order to reduce the amount of parameters by introducing an invariance to the output of the convolutional layer [4, p. 335-339]. One example of an invariance through max pooling is invariance to translation or rotation [4, p. 338]. By choosing the largest value indicating a pattern recognition of the convolutional filter, the output will reflect a pattern recognition, irregardless of placement or rotation of the pattern recognized, see fig. 6.
Figure 6: *Description of rotational invariance, as described in [4, p. 338].*

With the pooling layer summarizing responses over many nodes, the sum of all nodes going into the max pooling layer can be summarized with one node. This, in combination with the length of the stride could ultimately improve computational efficiency of the neural network [4, p. 337]. Stride specifies the number of pixels the filter moves at a time when sliding over a grid of input values. When the stride length is 2, the filter moves two pixels at a time. The output of the layer is related to the number of computations. By specifying a larger stride length, fewer computations are made resulting in fewer output parameters and a smaller network size [12].

There are two prevalent ways of pooling: average pooling and max pooling. In fig 7 the max pooling process can is shown.

![Max Pooling Diagram](image)

Figure 7: *Max pooling of a matrix. The pooling is done to reduce the number of parameters by selecting the maximum value of a parameter in a fixed area of the matrix.*
2.1.4 Dense layer

A fully connected layer or dense layer connects the output from every node from the previous layer to the inputs of the nodes of the next layer. Each connection has a weight attached to it, the value of which is a result of training the neural network. A simple example of a dense layer is shown in fig. 8.

![Diagram of a dense layer]

Figure 8: A fully connected layer, connecting five nodes to the final two nodes, for example the prediction of the network.

2.1.5 Neural networks implemented with limited processing power

When reducing the amount of memory and computational power, as in the case of implementing a neural network on a MCU, it is important to evaluate the structure of the implemented neural network. Evaluation of performance of neural networks on microcontrollers have been made before [26], [15], [2]. In these papers, tests were done on Arm Cortex processors, one processor being the Arm Cortex M7, present within the STM32F746NGHx MCU used in this project. In [26], neural networks ranging from small (80kb in size) to medium (200kb in size) to large (500kb in size) were tested to find the level of accuracy and number of operations needed. Along with the test details, the hyperparameters for the different neural networks were detailed [26, p. 8]. In tab. 2, the neural networks evaluated are detailed.
Table 1: Neural networks implemented on microcontroller units by Zhang, Y et al [26].

| NN model | DNN | CNN | Basic LSTM | LSTM | GRU | CRNN | DS-CNN |

2.1.6 Measuring classification accuracy

Defining the accuracy of the system can be done by assessing a confusion matrix. The confusion matrix is used to summarize the classification performance of the neural network. It is a two-dimensional matrix indexing the true class of an object in relation to the prediction of the neural network [24]. The structure of the confusion matrix used to evaluate the performance of the neural network predicting the Yaris car horn is detailed in fig. 9.

![Confusion matrix for the four outcomes of the prediction of a car horn.](image)

Using the results from the confusion matrix, a measure of the system's accuracy called the F1 score will be calculated with equation 5. The F1 score is defined as the harmonic mean of a test's precision and recall [16].

\[
F = \frac{2PR}{P + R}, \tag{5}
\]

where \( P \) indicates precision eq. (6) and \( R \) indicates recall, see eq. (7).

\[
P = \frac{TP}{TP + FP} \tag{6}
\]
\[ R = \frac{TP}{TP + FN} \]  
\[(7)\]

2.2 Training the network

The training of the network is done through the optimization of the loss function, also called a cost function. The loss function is the error of the network, and the optimization is done by minimization of this error [13, p. 44]. This process is described further in 2.2.1.

Training a neural network is done through the correction of the value of the weights multiplied at summation of the nodal values, described in eq. (1). In order to understand whether the change of the weight is correct, one has to assess the gradient descent of the loss function [13, p. 17]. Correcting the value of the weight means moving either away from or towards the slope of the loss function, with the goal to reach the lowest point of the loss function, which relates to the lowest error.

During training, the network is being fed with data from a collection consisting of both the input data and the labelling of the data. This is called a data set. The input data is analyzed within the layers of the neural network and as output the network returns a prediction on the nature of the data. This prediction is compared to the actual label of the data set, meaning the "true" value.

This training process is done is divided into two stages: the training phase and the validation phase [4, p. 119]. The data set is therefore divided into two sets of data. The training data is used to learn the parameters and the validation data to estimate the error during training for tuning of hyperparameters. Hyperparameters are values not set by the learning algorithm itself, but are user specified [4, p. 118], detailed in section 2.2.3. The training data and validation data stem from the same original data set, and the ratio between the amount of training data in relation to the amount validation data is called the validation split. A common validation split is 80 percent training data and 20 percent validation data [4, p. 119].

Further, the training is divided into epochs. An epoch is the complete cycle of the training algorithm; when the network has been trained with the entire training set. At this point, the epoch ends and another epoch is started [13, p. 23]. Depending on the amount of epochs used in training (relating to the accuracy of the system), the number of epochs differ.
2.2.1 Backpropagation

The basis behind backpropagation is the iterative process of evaluating $\frac{\delta C}{\delta w}$. This relates the derivative of the cost function $\delta C$ to the change in the value of a weight $\delta w$ [13, p. 43]. A more general definition for backpropagation when training the network can be seen in eq. (8), where $J(\Theta)$ denotes the cost function, and $L(x_i, y_i, \Theta)$ is the loss function.

$$\hat{\Theta} = \arg \min_{\Theta} J(\Theta), \text{ where } J(\Theta) = \frac{1}{n} \sum_{i=1}^{n} L(x_i, y_i, \Theta)$$  \hspace{1cm} (8)

2.2.2 Gradient descent

This process of reduction of the cost function is done by stochastic gradient descent. The process can be thought of as the simulation of a ball rolling down to the bottom of a valley. It is done by numerically assessing each step in the descent and choosing the direction where derivative in respect to change of variables has the largest negative value, IE where the downward slope is as fast as possible. The bottom in this analogy represents a low point of the cost function [13, p. 17-18].

![Figure 10: The gradient ascent / descent following the slope of a three dimensional plane. Picture from Wikimedia Commons [25].](image)
2.2.3 Hyperparameters

One key aspect of training the network is to change relevant hyperparameters. One example of a hyperparameter is the complexity of the model [3]. A high level of complexity could result in overfitting and make the network poor at generalization, while on the other side of the spectrum, underfitting could result in the model not being able to assess the data in its entirety [3]. The training process is defined by the user but ultimately done by a computer. The user based optimization of the neural networks are more related to the hyperparameters: the structure of the neural network, kernel size used in convolutional layers, stride length etc. When implementing a neural network on a restricted amount of memory such as the case on a MCU, the tuning of the hyperparameters will be important. Table 2 contains examples of hyperparameters in the networks described by Zhang, Y et al in "Hello Edge: Keyword Spotting on Microcontrollers" [26].

Table 2: Hyperparameters of the neural networks implemented on microcontroller units by Zhang, Y et al [26].

<table>
<thead>
<tr>
<th>NN model</th>
<th>Model hyperparameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNN</td>
<td>Number of fully-connected (FC) layers and size of each FC layer</td>
</tr>
<tr>
<td>CNN</td>
<td>Number of Conv layers: features/kernel size/stride, linear layer dim., FC layer size</td>
</tr>
<tr>
<td>Basic LSTM</td>
<td>Number of memory cells</td>
</tr>
<tr>
<td>LSTM</td>
<td>Number of memory cells, projection layer size</td>
</tr>
<tr>
<td>GRU</td>
<td>Number of memory cells</td>
</tr>
<tr>
<td>CRNN</td>
<td>Conv features/kernel size/stride, Number of GRU and memory cells, FC layer size</td>
</tr>
<tr>
<td>DS-CNN</td>
<td>Number of DS-Conv layers, DS-Conv features/kernel size/stride</td>
</tr>
</tbody>
</table>

2.3 Signal processing

Implementations of neural networks in order to perform keyword spotting [15] or audio event detection [7] has been done by implementing spectrograms when preprocessing the input audio. Since convolutional networks are well suited for the classification of images [9], a visualization of the input audio is one approach in order to effectively implement classification of audio. This stems from the fact that audio recognition is based on the conversion of audio signals into a quantifiable format which the computer can understand. In this case, the conversion is from a continuous audio stream to discrete audio samples, transformed using short time Fourier transform (STFT) and displayed in a spectrogram.
2.3.1 Spectrogram

The spectrogram is utilized in order to visualize the length, frequency and intensity of an audio signal [18]. An example of which can be seen in fig. 11.

![Spectrogram Example](image)

**Figure 11:** The spectrogram of an signal with intensity at frequency 512Hz.

The spectrogram is calculated by computing the STFT on the length of the input audio vector [18]. The transform is done by dividing the input vector into segments and computing the Fourier transform on each segment. The result of this is a number of frequency spectrums. The combination of the spectrums in chronological order creates the spectrogram. The equation for calculating the discrete short time Fourier transform for each segment is detailed below in eq. (9).

\[
\text{STFT}\{x(n)\}(m, \omega) \equiv X(m, \omega) = \sum_{n=-\infty}^{\infty} x(n) w(n - m) e^{-j\omega n} \quad (9)
\]

The window function \( w \) is a Hann window or a standard normal distribution. The variable \( m \) is a discrete value, and with the case of a fast Fourier transform, the \( \omega \) is also discrete. The result is a representation of the individual frequencies, or sub frequencies, constructing the ingoing signal. Corresponding to each sub frequency is an amplitude of the frequency. Therefore, the amplitude and frequency of each time-step is calculated.
2.3.2 The mel scale

There are ways to further process the ingoing audio in order to make the readings more relevant for the case of this thesis. The mel scale is based on how the human ears recognize shifts in frequency, relative to the actual frequency [14]. By scaling the spectrogram using the mel scale, the final output is also in a linear scale more appropriate to the perceived scale of human hearing. The conversion between Hertz scale and mel scale can be seen in eq. (10).

\[ m = 2595 \log_{10} \left(1 + \frac{f}{700}\right) \]  

(10)

An example is that for a human, the scale between 500Hz and 1000Hz is perceived as larger than 2000Hz to 2500Hz. Since the end goal of the product is to recognize sounds of alarm centered around what humans recognize as warning signals, it would be reasonable to implement the mel scale in the spectrogram. In fig. 12 is the mel scale, relative to the Herz scale. When implementing the mel scale on the spectrogram, the frequency range is partitioned into mel bins. Mel bins are partitions in which intervals of frequency are allocated. 30 mel-bins denotes partitioning the frequency range into 30 intervals, or bins. The intensity of a frequency range contained within a bin dictates the value between 0 - 1: 1 being low amplitude and 0 being high amplitude.

![Figure 12: Mel scale in relation to Hertz scale [8] by Krishna Vedala. Used in accordance with free use under GFDL.](image)

Figure 12: *Mel scale in relation to Hertz scale [8] by Krishna Vedala. Used in accordance with free use under GFDL.*
2.4 STM32F746g discovery board

The microcontroller chosen to implement the neural network on is the STM32F746NG based on an Arm Cortex M7-processor. With a floating-point unit (FPU) enabling floating point operations and resources for signal processing [19], the STM32 family of MCUs are well suited for the many dot operations and multiplications used in neural networks [26, p. 3]. The STM32F746g-discovery board connected to the MCU brings the relevant periphery hardware needed to develop the application. Two microelectromechanical system (MEMS) omnidirectional microphones are connected to the discovery board for audio input [22]. An LCD touchscreen display can be implemented in order to display audio input, signal processing steps or other relevant information for visualization of data. Further, the STM32F746NG has an SRAM memory of 320 KB and 1MB flash memory, with an internal clock frequency of 216 MHz. The combination of relevant hardware with a large amount of resources and libraries along with computational power for the software makes the STM32F746g a good candidate for developing neural network applications. The STM32 platform can be used with the STM32CubeMX development tools [20]. Implementation of neural networks on the STM32 MCU can be done with the X-Cube-AI software expansion for STM32Cube [21].

2.5 Active Noise Cancellation with AS3415

The building blocks for the noise-cancelling headphones are the STM32F746NG MCU, detailed previously and a active noise cancelling (ANC) filter. The AS3415 is a solution-on-a-chip [1], meaning it is a product developed for direct implementation without needing in depth configuration. Choosing a solution for noise reduction is based on one major factor included in the AS3415. The integrated chip has a bypass switch, allowing for the microcontroller to control when the noise reduction is active [1]. If the ANC filter is deactivated and the bypass switch is active, the sound will pass through the filter without the added noise cancellation. In theory, this would mean that the headphones change from noise cancelling headphones to ordinary headphones.
The topology of the ANC circuit AS3415 is feedforward. An external microphone records the noise surrounding the headphone and mixes in a phase-shifted signal of the noise with the audio input. Figure 14 shows the block diagram of the process.
3 Implementation

Creating the proof-of-concept was done by first of all creating the data set used in training the network. A simple network model was then created in Keras and Tensorflow in order to be implemented with the STM32F746NG MCU. Before the final implementation of the neural network on the MCU, the preprocessing steps needed to be in place on the MCU.

3.1 Creating the data set

The neural network was trained on data from audio recordings. Processing the audio into a data set that was viable for training neural networks consisted of several steps. Creating the data set from scratch offered the opportunity to create a framework that could be used to create more data sets based on audio recordings. The functionality of the system to be created stems from proper implementation of the neural network on the microcontroller platform but also from the framework detailed below. The framework consists of the following steps:

- Recording audio.
- Creation of a large (two hours long) audio file consisting of samples to be classified mixed with ambient noises.
- Splitting the long audio file into samples of 4410 samples (relating to a 100ms snippet, since sampling frequency was 44100 Hz.
- Labeling contents of 100ms audio snippets (car horn vs. no car horn).
- Transforming the 100ms audio snippets with from audio files to spectrogram figures.
- Categorizing figures depending on the label.
- Reshaping figure size to 30x30.
- Create data set appending the individual figures into one long file containing input and one containing labels (output).

With this, the data set was created and was used to train networks in recognition of warning signals. Certain parts of the framework were more computational heavy than others, for example the splitting of one large
audio file into many smaller 100ms samples. With a two hour audio file, this creates 72000 individual audio samples.

![Conceptual overview of the large audio file with car horns dispersed throughout the file.](image)

**Figure 15:** Conceptual overview of the large audio file with car horns dispersed throughout the file.

Fig. 15 shows an overview of the audio file containing the ambient soundscape of a city combined with the car horns supplied within. In order to use it in training the network, the correct labelling of the data needed to be done.

The final task was to convert this large amount of audio into a shape that the network can recognize. As previously stated, the audio recognition was based upon image recognition from the creation of spectrograms.

### 3.1.1 Audio recording & editing

The data set was created by recording several audio samples of the car horn on a Toyota Yaris. The different samples varied as result of recording the car horn passing the microphone with different speeds, in addition to standing still. The maximum speed of the car passing was at 70km/h.

After recording a variety of honking sounds, an ambient sound file was found online [17]. This file contains the ambient noise of a small town, with traffic, some birds and other ambient sounds. The car horns were then distributed randomly along a 2-hour long audio file. Two hours of audio, divided into 100ms or 50ms samples to be analyzed with the spectrogram would result in 72000 or 144000 samples as a result of the sampling frequency of 44.1 kHz. This also implies that the length of each sample is 4410 in the case of 100ms and 2205 in the case of 50ms.
Figure 16: Audio files in editing software, the top layer shows the car horns randomly placed across the file. Below is the ambient sound file used as background noise.

The total amount of car horn samples placed in the sound file was 1992 samples, each 100ms long. Each sample consists of 4410 points of data. The next task was to combine sum the absolute value of the 4410 data points in each sample, in order to get 72000 samples describing if there was a car horn or not in the sample.

3.1.2 Labeling the data set

In order to train the neural network with the data set, the labelling of the input data had to be created. Finding where the car horns are located was done by evaluating the sound file consisting only of the isolated car horns. Each car horn-sample would be signaled by a non-zero value. Knowing that the total amount of car horn samples was 1992 samples, a threshold could be found. If a sample was above this threshold value, it follows that it was a car horn sample. Taking this into practice means assessing the absolute value of the car horn audio signal. If the absolute value is above a certain threshold, one registers a logical "on" value (one).
As shown in fig. 17, the value which correctly relates the number of non-zero samples of the car horns is at threshold $= 25.50$. If the sum of the absolute value in a sample (consisting of 4410 points of data) is above the threshold, this equates to a car horn being present in the sample. Using the threshold value, a vector was created consisting of ones where the car horn is active. This was the labels of the data set to be used in training the networks.
An overview of the algorithm used in registering the car horns can be seen below.

Figure 19: Algorithm used to assess whether a car horn was in the sample or not. \( i = 72000 \) samples

```plaintext
threshold = 25.50;
for i = 1:72000
    if sum(abs(soundfile(i,:))) > threshold
        car_horn(i) = 1;
    end
end
```
3.1.3 Spectrograms

Splitting the 2 hour audio file into samples was done in Python, using pydub. The audio segments were exported in 100ms pieces. Following this, the samples were transformed using Short-time Fourier transform (STFT) and then scaled to mel-scale using the librosa library in python. The choice of grey scale instead of colours stems from the fact that the intensity of the sound, and the placement of the sound in the spectrogram are together indicative of the type of sound. The resulting data set contained images turned into vectors with a scale between 0 - 1, depending on darkness level along with a label in the form

\[
\text{data set}[1] = (\text{image}, \text{label})
\]

Choosing the optimal size for the images is a balance between weighing the legibility of the image versus the performance of the network. The best case is a small picture were defining features can still be noticeable. This was measured by eye, with the assumption that if an observer can notice certain patterns, the size of the image is good enough. The size of the image was therefore chosen to be 30x30. This means that the image is represented by 30x30 values ranging from 0-1, describing the darkness of each point. A value close to 1 denotes a low amplitude with white, 0 denotes high amplitude with black. The compressed spectrograms can be seen below in fig. 20.

![Figure 20: Compressed spectrograms with an image size 30x30 used to train the neural network.](image)
3.1.4 Iterations in data sets

During the course of the project, many variants of data sets were created based on the process described above. The reason being to evaluate the effect on the accuracy of the system depending on the length of the sample and the structure of the sample. Below details the six different types of data sets created to be used in training the networks.

Table 3: Iterations of data sets used in evaluating the networks

<table>
<thead>
<tr>
<th>data set No</th>
<th>Sample length</th>
<th>Size of data set</th>
<th>F&lt;sub&gt;Max&lt;/sub&gt;</th>
<th>n_mels</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100ms</td>
<td>72000</td>
<td>F&lt;sub&gt;s&lt;/sub&gt;/2 = 22050 Hz</td>
<td>128</td>
<td>Only Yaris car horn</td>
</tr>
<tr>
<td>2</td>
<td>100ms</td>
<td>72000</td>
<td>20000</td>
<td>30</td>
<td>Only Yaris car horn</td>
</tr>
<tr>
<td>3</td>
<td>50ms</td>
<td>144000</td>
<td>F&lt;sub&gt;s&lt;/sub&gt;/2 = 22050</td>
<td>30</td>
<td>Only Yaris car horn</td>
</tr>
<tr>
<td>4</td>
<td>100ms</td>
<td>72000</td>
<td>20000</td>
<td>128</td>
<td>Multiple warning signals</td>
</tr>
<tr>
<td>5</td>
<td>100ms</td>
<td>72000</td>
<td>20000</td>
<td>30</td>
<td>Multiple warning signals</td>
</tr>
<tr>
<td>6</td>
<td>50ms</td>
<td>144000</td>
<td>20000</td>
<td>30</td>
<td>Multiple warning signals</td>
</tr>
</tbody>
</table>

In all these cases, the processed data set was spectrograms of size 30x30. The appearance of the spectrogram changed in relation to which settings are used. Fig 23 shows the same sample in time, processed with the differing spectrogram settings. While there were six data sets in total used, there were three settings for the appearance of the spectrogram used, but the data sets differed in that they contained either only Yaris car horns or multiple warning signals, see table 3 When evaluating the choice of spectrogram, the number of samples in the data set and the relative appearance of the spectrogram related to how the spectrogram on the STM32 MCU were considered. The number of samples could relate to how well the networks could be trained. The appearance of the spectrogram was evaluated in order to get a coherence between the images training the network and the images the network will classify.
3.2 Evaluated networks

The type of networks chosen to be used for the implementation of the system were chosen from the list of networks evaluated in [26], detailed in 2. There are a multitude of networks to be evaluated, but limiting to four of these
detailed in the 1, both based on size and functionality would facilitate a
more thorough evaluation of these networks. The four networks chosen to
be evaluated are shown in tab. 4. The topology of the networks are included
in the appendices. The DNN network

<table>
<thead>
<tr>
<th>Network type</th>
<th>No. of parameters</th>
<th>Flash memory req.</th>
<th>RAM memory req.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNN</td>
<td>2,842</td>
<td>11.37 kB</td>
<td>144kB</td>
</tr>
<tr>
<td>CNN</td>
<td>12,302</td>
<td>49.21 kB</td>
<td>16.01 kB</td>
</tr>
<tr>
<td>CRNN</td>
<td>32,018</td>
<td>125.68 kB</td>
<td>9.19 kB</td>
</tr>
<tr>
<td>DS-CNN</td>
<td>12,075</td>
<td>48.94 kB</td>
<td>57.60 kB</td>
</tr>
</tbody>
</table>

The first network is the deep neural network (DNN). It consists primarily of dense layers in combination with max pooling. A dropout layer is used to further remove parameters. Three of these networks (CNN, CRNN, DS-CNN) are convolutional networks with differing capabilities. The convolutional neural network (CNN) contains a convolutional layer with max pooling layer, stride length 2 for reduction of network size. Batch normalization was used after the convolutional layer, with the goal of improving training rate [6]. The convolutional recurrent neural network (CRNN) shares the convolutional layers of CNNs. However, after a phase of convolutional layers, the output is fed into two layers of bi-directional long short-term memory (LSTM) layers. It is able to detect patterns using the convolutional layers and extract temporal dependencies using the recurrent layers [26, p. 4]. The CRNN is therefore recognizing patterns and can evaluate them in relation to earlier input signals.

The depthwise separable convolutional neural network (DS-CNN) has been used to achieve compact network architectures in the area of computer vision [26]. By replacing 3D convolutional filters with two layers consisting of (I) a 2D convolutional filter followed by a (II) 1D convolutional filter, the result are small and efficient models [5]. A visual overview of the topology of the models can be seen in appendix ??.

3.3 Recognition of multiple warning signals

Evaluating the systems capability to recognize several different warning signals can give an indication of the scalability of the system. More outcomes results in a need for more complex networks to analyze and perform classification. The limited processing power and memory space of the micro-
controller again provides a limitation of the size of the network. The choice of additional warning sounds was a bicycle bell and another car horn. This means that the network has to accurately classify four different scenarios: ambient noise, Yaris car horn, a bicycle bell and another car horn.

The methodology of evaluating the scaling possibility of the neural network on the STM32F746g-disco followed the same steps as with the recognition of a single sound. Some decisions had to be made in order to scope the project boundaries. The first decision was to focus on the recognition of one warning signal at a time. This meant that the audio file and training data would never contain a sample with two warning signals at the same time. In order to assure that there was no overlap between warning signals, a hierarchy was created, using an audio effect called “audio gate”. The audio gate effectively silences one sound, in the case that another specified sound was being played at the same time. This hierarchy was structured so that the Yaris car horn was at the top followed by the bicycle bell, and lastly the second car horn was at the bottom of the hierarchy. While the Yaris car horn was sounding, none of the other samples was played. If the bicycle bell was sounding, any eventual sounds from car horn 2 was silenced. If none of the other two were active, car horn 2 could be heard.

Following the creation of the audio file, a labelling file was constructed. In contrast with the Yaris car horn, which had many different samples recorded, the bicycle bell and car horn was only one sample. Constructing a label with both the Yaris horn, the bell and the second car horn was done as in the previous case with one warning signal. However, since there were different outcomes, the Yaris car horn was labeled as "1", the bell as "2" and the second car horn as "3".

With an audio file and correct labels, spectrograms could be constructed and used in training the neural networks for recognizing the multiple sounds. At this point, assessment was made regarding the most relevant way of implementing neural networks to classify different signals. One straightforward way would be to use the data set with multiple labelled warning signals and train a network to recognize them all. Another method would be to train three individual networks to specialize in recognizing one of the three warning signals. Figure 24 shows a overview of the different alternatives. One important point was that while it appears that the three networks are working in parallel, that is not the case. With the single core capacity of the STM32F746NG MCU, the process would be to go through one network at a time.
3.4 Creation & training of the network

The training of the network was done in batches containing 10 photos, with Adam optimization and binary crossentropy as loss function metric in the case of the single warning signal. For multiple warning signals, categorical crossentropy was used. The training was done with a validation split of 0.2, where the number of epochs chosen to achieve the highest validation accuracy depending on model. The result the training of networks can be seen in fig. 25.

![Figure 25: Training of the four models using the data set containing 100ms-long samples with 30 mel bins. The models training accuracy converges to values above 99 percent.](image)

The input data $X$ consisted of a 30x30 matrix containing normalized values ranging from 0 - 1 representing the relation between black and white, where 1 is white and 0 is black. The output $y$ was a two-index vector,
\[ y = [a_1, a_2] \]

where \( a_1 \) and \( a_2 \) are the two outcomes to be predicted: car horn or no car horn. During primary development and testing of the neural network, the division of number of car horns in relation to non car horns were 50/50. Without proper tuning of the hyperparameters, the network would tend to choose the type of sample which was in majority. Optimization of performance and reduction of size happened with the help of tuning hyperparameters, as referenced in 2.

### 3.5 STM32 Architecture

The neural network is trained on an external computer and then converted from Python code to C-optimized code to use with the microcontroller. With the network in place on the STM32F746NG MCU, the processes surrounding the network including processing audio input and reading the predicted output from the microcontroller needed to be implemented. Since the neural network was implemented as image recognition, the conversion of input audio from the microphones to spectrogram was performed.

![Diagram](image)

**Figure 26:** *Process overview of input audio recorded on the MEMS microphones, transformed via fast Fourier transform and structured within a spectrogram to be analyzed by the neural network.*

Each of the steps in this process was done by finding and implementing
resources such as STM32-based software or resources made for the ARM M7-Cortex processor. Below describes the usage of each of the resources in order to get the proof-of-concept working.

3.5.1 Board Specific Package (BSP)

Controlling the peripherals of the STM32f746g-discovery board was done by implementing board specific packages. These packages contained component-specific drivers and code for simplifying the use of the discovery board. The BSP was used primarily for audio input and control of the LCD display when developing the preprocessing framework.

Audio input was recorded using the two micro-electromechanical system (MEMS) microphones. The MEMS microphones capture sound using pulse density modulation (PDM). The PDM signal was then converted in the audio codec component wm8994 into pulse code modulated (PCM) signal which was used as the input for preprocessing and finally in the network. The conversion between PDM and PCM is done by decimation, or downsampling of the signal. This is done by only reading samples as set by the decimation factor. The decimation factor was set to 64 which implies reading one of 64 samples, reducing the sampling frequency by a factor of 64. In contrast with analog microphones implemented with an analog-to-digital converter, where a measurement is taken at for each sample rate, the MEMS microphones operate at a higher frequency range. In order to provide a sampling rate of 44.1 kHz as used in this project, the sampling frequency of the MEMS microphones, $f_{\text{MEMS}}$, is to be set at a sampling frequency

$$f_{\text{MEMS}} = f_s \times \text{decimation factor}$$

This transfer process is done through the serial audio interface (SAI), which connects the MEMS microphone with the audio codec via I2C. The PDM signal, converted into PCM is then transferred via direct memory access (DMA) to the input buffer to be used in the spectrogram. The DMA transfer is handled by a processor separate from the main CPU, in order to offload memory transfers from CPU and optimize performance.

3.5.2 STM32_AI_AudioPreprocessing_Library

The preprocessing the audio input into a spectrogram was done by implementing STM32_AI_AudioPreprocessing_Library developed by STM to be used in development on the STM32 platform. The key factor in choosing this specific library is its base in the Keil CMSIS library. The setup of the
spectrogram and Mel spectrogram structs in the C-code was similar to the librosa library, used in creating the spectrograms in the data set used to train the network.

3.5.3 CMSIS

The CMSIS library is highly useful for matrix and vector multiplication along with dot product calculations which are prevalent in neural networks\[10\]. Quoting L Lai, N Suda, V Chandra in the paper "CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs": "... CMSIS-NN, efficient kernels developed to maximize the performance and minimize the memory footprint of neural network (NN) applications on Arm Cortex-M processors ... "\[10\].

The vector based operations was also well suited for handling the preprocessing of the audio signals in the form of normalization.

3.5.4 STM32Cube & X-Cube AI

X-Cube-AI extends the STM32Cube software with a neural network library generator. The software can take networks trained with frameworks such as Keras in order import and convert the network from Python code to C-optimized code \[23\]. The conversion of the network to C-optimized code results in an application which can be used to run the inference. The X-Cube AI extension of the STM32Cube development environment has been used in order to quickly implement and iterate networks on the STM32F746g-disco.

3.6 Preprocessing audio input

Preprocessing the audio input was done by implementing the Audio processing library as described in section 3.5.2 in order to create the spectrogram. Vector-based computations in order to optimize the performance was done via the CMSIS library. The spectrogram figures used in training the neural network has the shape of 30x30 values ranging from 0 - 1. On the STM32 platform, the main objective in a successful implementation is preprocessing the data similarly to the training data. The preprocessing of the audio input was done by transforming it into spectrograms with 30 frequency bins. By continually transforming input audio into spectrogram columns and storing in a spectrogram buffer of size 30x30 (900 values), a full spectrogram figure was created consisting of 30x30 values, ranging from 0 - 1 arranged in a similar way as the training data. This 900-values-long buffer is then input into the neural network inference function.

37
Figure 27: Spectrogram created on the STM32F746g-discovery board. The spectrogram shows the moment of a Yaris car horn, which can be seen as the dark line crossing the width of the spectrogram.

3.7 Optimizing performance

3.7.1 Runtime

The total runtime of both the preprocessing and the neural network was done by counting the amount of completed runs the program could within a set timeframe and calculating the average time for one complete process. The process from audio input to neural network output seen in fig. 28.

Figure 28: Runtime factors of the entire process from taking input audio to performing inference and making a prediction.

The major factors affecting the total runtime were the preprocessing steps and the neural network. This entire process was also affected by the
buffer size of the input audio. The buffer size sets the size of the input vector. During implementation, two buffer sizes were predominantly evaluated: 512 and 1024 samples, where a sample was a floating point value stored in the input vector. The vector was then transformed into the spectrogram and preprocessed before being input into the neural network inference process. Minimizing the runtime consists of evaluating the size of the neural network. This affects the complexity of the network and also the memory footprint. The preprocessing was mainly computationally heavy when transforming the input data from time domain to frequency domain with Fourier transform. The process was done with fast short time Fourier transform (STFT), implemented with the CMSIS library. Along with Fourier transforms, vector calculations were done with CMSIS as well, for example in the case of normalizing the spectrogram.

3.7.2 Accuracy

Evaluating the precision of the networks was primarily done while training the network. There was a slight gap in the correlation between the training data and the input data on the microcontroller. This could be attributed to a different quality of input audio between the microcontroller and the hardware used to record the training data.

4 Experiment

An experiment was conducted in order to assess the performance and accuracy of different iterations of neural networks implemented on the MCU. The process was done in iterations, changing the shape and type of the network in order to assess the performance of many different types of networks. The required materials in both hardware and software is as follows:

- Computer with STM32CubeIDE (version 1.0.2) developing platform, plus the X-Cube AI embedded software package (version 4.0.0) added.
- STM32F746g-discovery evaluation board with mini USB to USB cable.
- Project files flashed onto the STM32F746g-discovery board with the STM32CubeIDE software ¹.
- Soundfiles for testing the network ².

¹Link to project github resource: https://tinyurl.com/aliss-project-files
²Link to sound files: https://tinyurl.com/aliss-audio-files
The setup of the experiment was done by placing the microcontroller at an approximate 25cm distance from the computer, standing on its four legs. The volume of the computer was set to halfway of maximum volume. The omnidirectional MEMS microphones on the STM32F746g-discovery board minimized the directional dependence of the microphones [22]. The experiment was done by playing a 150 seconds long clip on the computer containing recorded noise from a traffic intersection in Uppsala using a mobile device. Dispersed around the audio file are samples of the Yaris car horn, in the case of the single warning signal, or multiple warning signals, depending on which functionality is to be assessed.

![Figure 29: Experiment setup for evaluating the system accuracy and performance.](image)

The STM32F746g-discovery board was set into debugging mode and halted with a breakpoint before starting its main program. The debugging session was unhalted at the same time as the audio file was started. This activated its main program, which was the loop containing audio input, preprocessing and finally the neural network inference. The prediction from the neural network is, in the case of single warning signal type, either 0 for no Yaris car horn or 1 for a Yaris car horn. For multiple warning signals, the output is either 0 for no warning signal, 1 for Yaris, 2 for bicycle bell and 3 for the other car horn. The prediction of each run was saved in a vector, which was continually filled as the program runs in loops. At the point in which the audiofile was ended, the program on the STM32 MCU was again halted. Data could be gathered from a vector with the saved prediction
outputs. The analysis of the output vector was done in order to find an approximate average runtime of one loop of the main program, from taking the length of the vector and dividing the total elapsed time (five minutes).

Further evaluation of the output vector was done in order to assess the accuracy of the predictions. With the average time of each runtime, each prediction could be compared with the actual labels. In order to compare the data with the true values, the vector containing values gathered from the experiment extended by using interpolation from its actual size to the size of the true labels, which was 1500. This data was then taken and assessed using the confusion matrix in order to calculate precision, recall and finally the F1-score of the predictions.

5 Results

Certain plots most relevant for the result and discussion of the thesis are presented below. The remaining plots of results are for the sake of brevity placed in the appendix.

5.1 Classification of a single warning signal

In the case of classification of a single warning signal IE the Yaris car horn, the DNN model trained with 100ms-long samples with 30 mel-bins and buffer size 512 achieved the best F1-score, as shown in fig. 30. The runtime of the best performing networks were similar at 120ms.
Figure 30: F1-score and performance of the system when classifying the Yaris car horn, with buffer size 512.

Figure 31: Result from the DNN model trained with 50ms-long samples, 30 mel-bins and buffer size 512. The predictions from the network in relation to the "true" values.

However, assessing the experiment results of the system detailed in fig. 31 provides some more insight into the accuracy of the system. The DNN model captures instances within the longer car horns, but is more prone to falsely classifying a car horn where there were no car horn. In contrast, the results of the experiment with the DNN model, trained with 50ms samples and buffer size 512 can be seen in fig 32. The model can be seen as less prone to falsely classifying the car horn, but is not classifying longer car horns in
the same way.

Figure 32: Result from experiment with the DNN model, trained with 50ms, buffersize 512.

5.2 Classification of multiple warning signals

5.2.1 Multiple neural networks

One of the methods of classifying multiple warning signals is by training individual neural networks in order to recognize a specific signal. Using STM32CubeMX with X-cube AI, there is a capability to implement several neural networks, limited by the memory capacity of the MCU. These individual networks were evaluated using the same experiment as evaluating the classification of a Yaris car horn. In fig. 33, the F1-scores of the models classifying the bicycle bell are shown. The DNN model, trained with 50ms-long samples, 30 mel-bins achieved the highest F1-score and with a 512 buffer size.
Figure 33: *F1-score and performance of the system when classifying a bike bell, with buffer size 512.*

However, evaluating the experiment results show the DNN model as falsely predicting several bike bells. This can be seen in fig. 34.

Figure 34: *Experiment result from the DNN model, displaying predicted results vs. true results.*

Similarly, the highest F1-score when classifying the car horn was the DNN model, trained with 100ms-long samples, with 30 mel-bins and 512 buffer size as is shown in fig. 35. Those results, however, are indicative of the poor accuracy of the predictions.
Again, evaluating the experiment results shows a large inconsistency in the predictions.

5.2.2 Single neural network classifying multiple warning signals

The same networks trained with a data set containing multiple warning signals achieved low F1-scores, as seen in fig. 37. The highest F1-score in relation to runtime was the DS-CNN model, trained with 50ms-long samples, buffer size 512 and 30 mel-bins. Similar F1-scores and runtime were achieved by the DNN model, trained with 100ms-long samples, buffer size 1024 and 30 mel-bins.
Assessing the two individual experiment results in fig. 39, it is shown that, although the DNN accurately captures ambient noises, it does not predict any other warning signal other than the Yaris. The DNN is able to classify more types of warning signals, but does result in false classifications.
6 Discussion

6.1 Relevance of experiment results

The results of the implementation shows that the DNN model is the highest performing in the majority of cases. This stems from the evaluation of the F1-score in relation with the runtime. By also assessing the results of the experiment, one observes the behaviour of the model. The experiment consisted of saving the predicted outputs of the system into a vector. The length of the vector, or the number of predictions made during the 150 seconds experiment could then be used to evaluate the runtime of one loop in each model.

One factor in evaluating the experiment run is that the vector with the predictions is not the same length as the vector containing the "true" values to be compared with. The way to address this issue was to interpolate the vector with experiment data onto a vector with length 1500. The interpolation itself adds a layer of uncertainty to the data, since it is not the truly
measured data but an approximation of it, extended onto a longer vector. However, comparing the extended vector with the original one there is a large coherence between the two, as can be seen in fig. 40.

They are not perfect, and thus the results should come over some scrutiny. Within the thesis boundaries of the project itself however, the F1-scores of the models relative each other still hold some value when comparing the performance. The outcome of interpolating the gathered data onto a longer vector therefore eliminates some of the thesis’ relevance on the global machine-learning arena. Still, the conclusions drawn from the data can still be used to provide a conclusive answer to the question of this thesis: is it possible to implement a neural network on a STM32 microcontroller in order to recognize the warning signal of a Yaris Car horn and can it be done within reasonable boundaries of accuracy and performance?
6.2 Experiment setup

One of the pitfalls of the experiment setup stems from the division of number of warning signals versus non-warning signal. In the experiment audio file, there were 112 samples of the Yaris car horn, in relation to the 1388 samples of ambient noise. With most models under performing and classifying all samples as '0' or as non-horn, the resulting F1-score was still high for a very bad performance. Producing a test audio file with labeling with a 50/50 ratio between car horn and ambient noises would result in better results.

6.3 Incoherence between training data and actual data

The reason for the multitude of different data sets used in the training of the networks stems from the performance of the STM32 in spectrogram creation. The STM32 implemented a mel-scaled spectrogram with 30-bins, with a 30x30 matrix with values from 0-1 similarly to the input data in the data sets used to train. The MEMS microphones on the STM32F746g-discovery board did not, however, record the exactly same quality of audio input as the microphone used to create the data set. During implementation of the spectrogram creation on the STM32 MCU, the BSP allowed to listen to the audio input. Along with the signal being high-pitched, when listening to the surrounding ambient noise, there was a constant background buzzing sound at a high frequency. When playing a sound, the buzzing disappeared.

The spectrogram creation on the STM32 MCU was done by first filling the 30x30 buffer with spectrogram inputs and then every averaging the value of every 6 pixels row-wise. This instead created a 30x5 spectrogram. The reason behind this is to create a consistent appearance between the spectrogram created with the stm32 and the training data, which in the case of 50ms samples had the same appearance of a 30x5 spectrogram, as can be seen in fig. 23. However, as one can see comparing fig. 27 (spectrogram created on the stm32) to 23 (training data spectrogram), there are some similarities and differences. This non-coherence brings a level of error to the result.

This incoherence also brings a level of abstraction between the results of training the networks and the actual performance of the networks. The validation accuracy has been consistently high, around 99% in most cases. This level of accuracy is certainly not translated to the actual end product. In order to create a product with relevant performance, this gap needs to be addressed.
6.4 Performance of the system

The fastest networks, with the best F1-score had a runtime of 120ms. This is a bit above the limitations of the envisioned system, where a 100ms runtime was the goal. The buffer size did not have a large impact on the accuracy of the system, but did affect the runtime.

The overall runtime was longer when implementing the networks in order to recognize multiple warning signals. The alternate solution in which multiple networks are implemented on the STM32 MCU in order to individually specify the warning signals was not tested in its entirety. The runtime of the individual networks with the best performance was around 120ms each. If three networks are to be implemented on the MCU, the total runtime is therefore approximately 360ms. This is definitely beyond the limitations of the project.

6.5 Classification of multiple warning signals and scaling the system

The models trained to classify multiple warning signals performed with a very low accuracy. This could stem from several factors. One is the variety of data used in the data sets for the bicycle bell and the car horn. In the case of the Yaris car horn, where the results were better, there were 8 different recorded audio samples used in the data set, containing a variety of the Yaris car horn passing the recording microphone in different speeds and the length of the car honking. The other warning signals were constructed of 2 samples (other car horn) and 1 sample (bicycle bell), which were dispersed throughout the audio file. This could be a factor.

The scaling opportunities of this solution is therefore reliant on two factors raised in this discussion: the incoherence between training data and measured data and the performance of the system. By reducing the inconsistencies, the optimization of the network would become simpler, allowing more streamlined networks.

6.6 Choice of neural network

In general the DNN model performed the best, producing relatively accurate results with the shortest runtime. With only 2842 parameters, the network is quite small. However, the implementation of the network on the STM32 MCU required a RAM memory of 144kB, which is almost half of the 320kB of RAM available on the STM32F746NG MCU. The small number of parameters contributed to a large RAM memory requirement, and expansion of
the network to the same number of parameters used within the other models was not possible due to the amount of RAM memory needed. Scaling the DNN model would therefore be difficult in order to extend into classifying multiple signals. The next best-performing network was the CRNN. While the memory requirement on the flash memory is large, 125.68kB, the flash memory has more memory capacity. The runtime was similar to that of the DNN. If scaling the model, the convolutional recurrent neural network would be a preferrable choice. For specified and quick classification of a key warning signal, a deep neural network would be best suited.

7 Conclusion

A number of neural networks have been trained with spectrogram samples and implemented on the STM32F746NG MCU. The goal was to evaluate the STM32 MCU’s ability to classify both a Yaris car horn and also multiple warning signals. The system is capable of recognizing the Yaris signal with some accuracy within a runtime of 120ms. Challenges in optimizing the network performance stem from incoherence between the training data and the measured data on the STM32 MCU.
References


A Visualization of models

Figure 41: Topology of the DNN model
Figure 42: Topology of the CNN model
Figure 43: Topology of the CRNN model
Figure 44: Topology of the DS-CNN model
B Results from experiments: F1-score versus runtime

![Graph 1](image1.png)

![Graph 2](image2.png)
F1-score in relation to runtime of system classifying multiple warning signals warning sign(s) with buffer size 1024

F1-score in relation to runtime of system classifying honk warning sign(s) with buffer size 1024

F1-score in relation to runtime of system classifying bike warning sign(s) with buffer size 1024

60
C  Experiment results - True vs. Predicted values
(Buffer size: 512)

C.1 Yaris car horn
C.2 Multiple warning signals
C.3 Bicycle bell warning signal
True values vs. predictions. Network type: CNN, trained with 100ms samples, mel bins = 30, input audio buffer size = 512

True values vs. predictions. Network type: DSCNN, trained with 100ms samples, mel bins = 30, input audio buffer size = 512

True values vs. predictions. Network type: DNN, trained with 100ms samples, mel bins = 30, input audio buffer size = 512

True values vs. predictions. Network type: CRNN, trained with 50ms samples, mel bins = 30, input audio buffer size = 512
C.4 Car horn warning signal
D Experiment results - True vs. Predicted values
(Buffer size: 1024)

D.1 Yaris car horn
True values vs. predictions. Network type: CNN, trained with 100ms samples, mel bins = 128, input audio buffersize = 1024

True values vs. predictions. Network type: CNN, trained with 100ms samples, mel bins = 30, input audio buffersize = 1024

True values vs. predictions. Network type: CNN, trained with 100ms samples, mel bins = 128, input audio buffersize = 1024

75
D.2 Multiple warning signals
D.3 Bicycle bell warning signal
D.4 Car horn warning signal

- True values vs. predictions. Network type: CNN, trained with 100ms samples, mel bins = 30, input audio buffer size = 1024

- True values vs. predictions. Network type: CNN, trained with 100ms samples, mel bins = 30, input audio buffer size = 1024

- True values vs. predictions. Network type: DNN, trained with 100ms samples, mel bins = 30, input audio buffer size = 1024