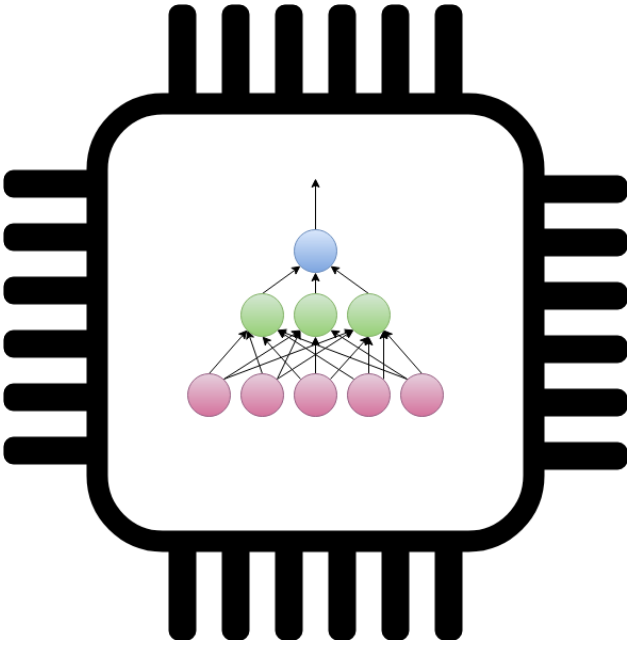
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Developing a prediction model for estimating hardware requirements based on standardized embedded system characteristics in the automotive domain

An application of Machine Learning on Hardware Estimation

Master's thesis in Computer science and engineering

Tobias Alldén

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Master's thesis 2019

Developing a prediction model for estimating hardware requirements based on standardized embedded system characteristics in the automotive domain

An application of Machine Learning on Hardware Estimation

Tobias Alldén

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

Developing a prediction model for estimating hardware requirements based on standardized embedded system characteristics in the automotive domain
An application of Machine Learning on Hardware Estimation
Tobias Alldén

Supervisors:
- Darko Durisic; Department of Computer Science and Engineering
- Miroslaw Staron; Department of Computer Science and Engineering
Examiner: Christian Berger, Department of Computer Science and Engineering.

Master's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2019

Abstract

Introduction: Modern day automotive vehicles often contain complex embedded systems that handle behavior of vehicles. These systems often comprise more than a hundred electronic control units (ECUs) that are responsible for different vehicle functionality, each requiring various degrees of computational power, and thus hardware requirements.

Objectives: This thesis presents the development and application of prediction models that are able to predict baseline hardware requirements for ECUs in terms of CPU, RAM and ROM metrics for vehicular embedded systems. These models are based on what common high-level features that ECUs have with the features largely being realized by standardized base software modules from the AUTOSAR Classic standard. The created models aim to introduce another way of validating hardware characteristics on ECUs or propose suitable hardware baselines for new ECU projects.

Methods: From the base software modules standardized by AUTOSAR, a grouping of said modules was presented to be able to characterize common ECU high-level features in Volvo Cars Corporation embedded systems. This characterization was then used as input to create regression neural networks that, given a set of features, are able to predict an estimation on what CPU, RAM and ROM size is required for a proposed ECU.

Results: The developed models are currently able to predict an estimation on resource utilization for 32-bit CPUs, RAM, and ROM at runtime for an ECU. This estimation is rounded up to a suitable baseline (total available measure on an ECU) based on identified Volvo Cars Corporation ECU baselines. The models are able to predict suitable hardware requirements for ECUs in 60-78% of cases.

Keywords: AUTOSAR, Hardware Prediction, Machine learning, Neural Networks, Base Software, Embedded

Acknowledgements

I would like to extend my most sincere thanks to *Miroslaw Staron* and *Darko Durisic*; for their continuous support and guidance in the pursuit of this thesis. Their insights and knowledge laid the foundation for my results and granted me the academic knowledge required to finalize this thesis.

I would also like to extend my sincerest thanks to my industrial supervisor *Daniel Svensson*, who helped me understand the wishes and context of Volvo Cars and guided me in my work.

Further, I would like to express my gratitude to my team at Volvo Cars; for giving me a warm welcome to the company and whose insight into the context and intricate details of the automotive industry as well as AUTOSAR helped me further my work and derive knowledge which would have otherwise been hard to obtain.

I would also like to thank *Oscar Evertsson* and *Rebecka Reitmaier* for reviewing and giving feedback to this report.

Finally, I would like to extend my deepest thanks to my family for their continuous support, without which none of this would have been possible.

Tobias Alldén, Gothenburg, June 2019

Contents

List of Figures xi

List of Tables xiii

List of Abbreviations xiv

1 Introduction 1

2 Theory 5
2.1 AUTOSAR . . . 5
2.1.1 AUTOSAR Reference Architecture . . . 5
2.2 Machine Learning and Neural Networks . . . 8
2.2.1 Neurons . . . 9
2.2.2 Types of Artificial Neural Networks . . . 10
2.2.3 Network Training . . . 11
2.2.4 Feed-Forward Neural Network . . . 12
2.2.5 Estimating the Efficiency of a Neural Network . . . 13

3 Related Works 15

4 Research Design 17
4.1 Research Questions . . . 17
4.2 Design Research Methodology . . . 18
4.3 Design Science Research Approach . . . 19
4.3.1 Awareness of the Problem . . . 21
4.3.2 Suggestions for Solution . . . 22
4.3.2.1 Data Collection . . . 22
4.3.3 Development . . . 23
4.3.4 Evaluation . . . 26
4.3.5 Conclusion . . . 26

5 Results 27
5.1 AUTOSAR BSW Logical Grouping . . . 27
5.2 Data Collection . . . 31
5.2.1 Collected data . . . 32
5.2.2 Training and Validation Data Split . . . 33
5.3 Hardware Validation and Selection at Volvo Cars Corporation . . . 36

5.4 Neural Network Construction . . . 37
5.4.1 Epochs Selection . . . 40
5.4.2 Model Accuracy . . . 41
5.5 Prediction Validation . . . 43
5.5.1 Identified Hardware Baselines . . . 43
5.5.2 CPU Validation . . . 44
5.5.3 RAM Validation . . . 44
5.5.4 ROM Validation . . . 44
5.6 Additional Model Candidates . . . 45
5.6.1 Multiple Regression Approach . . . 45

6 Discussion **47**
6.1 Data Analysis . . . 47
6.2 Resulting Models . . . 48
6.2.1 Comparison to Alternatives . . . 50
6.3 Usability of the Created Models for Volvo Cars Corporation . . . 50
6.4 Threats to Validity . . . 51
6.4.1 Threats to conclusion validity . . . 51
6.4.2 Threats to internal validity . . . 52
6.4.3 Threats to external validity . . . 52
6.4.4 Threats to construct validity . . . 53

7 Conclusion **55**
7.1 Results of the study . . . 55
7.2 Future Work . . . 56

A Appendix A - Interview Template for deducting Volvo Cars contextual knowledge and logical clustering **I**

B Appendix B - The Tested Configurations for the networks **IX**

C Appendix C - Informal interview for determining hardware validation and selection approaches at Volvo Cars Corporation **XVII**

List of Figures

2.1 The AUTOSAR ECU architecture with the three abstraction layers; the *Application Layer*, *Runtime environment* and *Base software* layer. Derived from (AUTOSAR consortium, 2017a) 6
2.2 AUTOSAR BSW Abstraction layers. Derived from: (*Layered Software Architecture* n.d.) 7
2.3 AUTOSAR functional clusters with a subset of the corresponding BSW modules. Derived from: (*Layered Software Architecture* n.d.) . . 8
2.4 An illustration of a biological neuron and an artificial neuron. Derived from (Raschka, 2015) 9
2.5 Three different kinds of neural networks. Derived from (Veen, 2017) . 11
2.6 Feed Forward Nerual Network with one hidden layer 12

4.1 Steps for conducting design science research 19
4.2 The general design science research-workflow followed in this thesis. The arrows indicate a transition from one activity to the next. The thick lines indicate a split into several activities that are performed in parallel or the joining of parallel activities. 21
4.3 Workflow when creating the neural networks for predicting the hardware measures 25

5.1 The resulting logical groupings of common BSW modules in ECUs . . 29
5.2 Visualizations of the occurrence of the different ECU characteristics . 32
5.3 RAM utilization boxplot for the training and validation set for our data, the measures are in Kilobytes 33
5.4 ROM utilization boxplot for the training and validation set for our data, the measures are in Kilobytes 34
5.5 CPU utilization boxplot for the CPU training and validation set for our data, the measures are in Mega-hertz 34
5.6 Scatter diagram for the training datasets on the CPU, RAM and ROM measures with the corresponding best-fit regression line. 35
5.7 Prediction accuracy on the validation dataset for the different tested configurations for the CPU network 38
5.8 Prediction accuracy on the validation dataset for the different tested configurations for the RAM network 39
5.9 Prediction accuracy on the validation dataset for the different tested configurations for the ROM network 39

5.10 The loss per number of epochs for the top CPU configuration in terms of validation accuracy (CPU1) 40
5.11 The loss per number of epochs for the top RAM configuration in terms of validation accuracy (RAM1) 40
5.12 The loss per number of epochs for the top ROM configuration in terms of validation accuracy (ROM1) 41
5.13 The predicted values and actual values of CPU resource utilization at runtime for the data in the validation dataset. The orange line signifies a line of best fit, i.e. correct predictions. The model visualized is the top configuration in terms of validation accuracy (CPU1) 41
5.14 The predicted values and actual values of RAM resource utilization at runtime for the data in the validation dataset. The orange line signifies a line of best fit, i.e. correct predictions. The model visualized is the top configuration in terms of validation accuracy (RAM1) . . . 42
5.15 The predicted values and actual values of ROM resource utilization at runtime for the data in the validation dataset. The orange line signifies a line of best fit, i.e. correct predictions. The model visualized is the top configuration in terms of validation accuracy (ROM1) . . . 42

A.1 Subset of AUTOSAR Modules mapped to the different AUTOSAR layers. Derived from (https://www.autosar.org) III
A.2 AUTOSAR Modules in their respective packages in the Vector Microsar implementation of the AUTOSAR Classic BSW stack. Derived from (https://www.vector.com/int/en/products/products-a-z/embedded-components/microsar/) III
A.3 Proposed grouping of AUTOSAR modules in logical clusters. VII

List of Tables

5.1 Count of the different characteristics the collected data 32
5.2 All the identified ECUs present in Volvo Cars Systems, on 32-bit single core processors 43
5.3 The non 32-bit ECUs identified 44
5.4 The top 4 candidates for the CPU neural network in terms of prediction accuracy for the validation data 44
5.5 The top 4 neural networks configurations for the RAM predictions in terms of prediction accuracy on the validation dataset. 44
5.6 The top 4 candidates for the ROM neural network in terms of prediction accuracy for the validation data 44
5.7 Multiple regression model predicted on the training data and computed accuracy for this model 45

B.1 Tested configurations for the CPU parameter IX
B.2 (Cont): Tested configurations for the CPU parameter X
B.3 Cont: Tested configurations for the CPU parameter XI
B.4 Tested configurations for the RAM parameter XII
B.5 (Cont): Tested configurations for the RAM parameter XIII
B.6 (Cont): Tested configurations for the RAM parameter XIV
B.7 Tested configurations for the ROM parameter XIV
B.8 (Cont): Tested configurations for the ROM parameter XV
B.9 (Cont): Tested configurations for the ROM parameter XVI

List of abbreviations

ANN	Artificial Neural Network
AUTOSAR	Automotive Open System Architecture
BSW	Base Software
CAN	Controller Area Network
CPU	Central Processing Unit
ECU	Electronic Control Unit
LIN	Local Interconnect Network
MAE	Mean Absolute Error
MOST	Media Oriented Systems Transport
MSE	Mean Squared Error
RAM	Random Access Memory
ROM	Read Only Memory (Flash Memory)
VCC	Volvo Cars Corporation
XCP	Universal Measurement and Calibration Protocol

1

Introduction

Software has become an integral part of modern day automotive vehicles and is often housed in the embedded systems on the Electronic Control Units(ECUs) that exists in vehicles. These types of embedded systems are rapidly growing in size and may now comprise over 100 ECUs per system, each with large amounts of compiled code performing the different functionality of the cars (Đurišić, 2017). With this increasing complexity, car manufacturers (OEMs) and subcontractors have begone collaborating to reach an industrial standard to standardize commodity functions and aspects surrounding the vehicle embedded systems. This standardization includes artifacts such as the development methodology for creating embedded systems and other process-related artifacts, while still promoting competitiveness in different areas such as customer offered functionality. The resulting standard, called AUTOSAR (AUTomotive Open System ARchitecture), is generally accepted as the de-facto standard for both the structure and behavior of the base software that handles the basic behavior of the ECUs, such as communication to other ECUs, as well as the development methodology for creating these embedded systems. (Durisic et al., 2016). The standard further promotes competitiveness by abstracting the top-level of the ECU to contain "software components" that is the functionality that is essentially marketed to the consumers and thus not standardized in AUTOSAR. Moreover, the methodology identifies different levels of developing parties, called tiers, for these systems, thus supporting the manufacturer-subcontractor relationship that is prevalent in modern day vehicle manufacturing.

The implementation of the base software and sourcing of the actual hardware for the different ECUs are generally done by subcontractors (identified as tier 2s by AUTOSAR). This hardware can both include peripherals on the ECUs such as different sensors and actuators on the unit, as well as the computational hardware, i.e. the central processing unit of the ECU. Given this, the decision on what a suitable amount of computational resources available on an ECU should be, such as CPU and memory specifications is not always performed by the OEM. As such, the OEM may not be involved in the selection of hardware requirements, nor in the validation of the hardware selection. Given this, the methodology creates a "chain of trust" in that the supplier delivers hardware that is apt and does not have an overhead that is too high in terms of computational resources, thus potentially yielding a large extra cost for the OEM. Finally, while the OEM validates the delivered system and its behavior, the validation is often in a "black-box" manner such that the validation ensures that the functionality of the system is as specified while leaving the hardware validation on the amount of available computational resources unchecked.

This thesis aims to address this potential threat by the introduction of models that can propose reasonable hardware requirements for an ECU, based on what AUTOSAR base software standardized functionality exists on an ECU. Reasonable hardware requirements is an amount of available resources that allow an ECU to perform its functionality even in the most load heavy of situations, while still having a low computational overhead thus allowing the OEM to save money on a per-ECU basis. As such, this thesis aims to introduce an addition to the current methodology of developing embedded systems by providing artifacts that allow for validating hardware resources of an ECU. Moreover, these artifacts may also aid in selecting a suitable hardware baseline for new ECU projects.

The models created during this thesis are based on what high-level functionality an ECU provides. The functionality is in itself mostly realized by the standardized software modules AUTOSAR describes in the BSW(Base Software) standardization. These modules are derived and split into logical clusters for different vehicle functionality that are common in vehicles following the AUTOSAR standardization, and then used to characterize ECUs by denoting which of these functionalities the ECUs have. The resulting characterizations are then used as input to the created models that are based on underlying machine learning models that are created from previous measures on CPU, RAM and ROM utilization for ECUs as well as the what formerly mentioned characteristic these ECUs fulfill. These models can then be used to propose an estimation on resource utilization at runtime for an ECU and thus give an indication on a suitable hardware baseline.

The findings of our study indicate that by regrouping the base software modules into logical clusters that are used to distinguish high-level functionality for ECUs we can effectively characterize ECUs and use these characteristics as a foundation for creating prediction models. These prediction models are built using neural networks and can be used to predict both the resource utilization at runtime for the CPU utilization in 32-bit single core CPU, RAM and ROM parameters, and in extension serve as the foundation to propose a hardware baseline, indicating what hardware requirements an ECU should have.

We have tested many configurations of these underlying neural network and are able to predict resource utilization at runtime for the 32-bit CPU, RAM and ROM parameters for ECUs. Moreover, these predictions can then be used to propose a suitable hardware baseline for these three parameters by rounding the number up to the nearest identified hardware baseline available in an ECU defined by what hardware baselines have been found in ECUs in use at Volvo Cars Corporation. The models created are able to predict such hardware baselines that in 60-70% of cases propose an ECU that is at par or better than what is actually used for a set of ECUs. These results thus indicate that the models created can be used to validate delivered ECUs from subcontractors to see whether the selected hardware baseline is feasible, or to propose a suitable hardware baseline for new ECU projects..

The thesis begins by introducing the theoretical foundation for understanding the concepts of the thesis and the techniques used for the created models. After this, the research methodology and research questions are addressed. This eventually leads to the results of our study where we present our findings. After this, we discuss the findings and address what validity threats exist. Finally, we conclude and generalize our findings, answer the research questions, and pave the way for future work based on the findings of this study.

2
Theory

This section aims to explain the theoretical foundation on which this thesis is built. As such, this section provides the necessary knowledge in order to comprehend the different tools and techniques used in this thesis.

2.1 AUTOSAR

The **AUT**omotive **O**pen **S**ystem **AR**chitecture(AUTOSAR) is the de-facto standard for building embedded software systems in modern-day personal vehicles. The standard is a collaboration comprising of over 150 partners in order to standardize both the architecture in the embedded systems of the vehicles, as well as the methodology surrounding these (Bo et al., 2010; Đurišic, 2015). This collaboration comes as an effect of the sheer size of these embedded systems in modern day vehicles, today often comprising of more than 100 ECUs per vehicle (Durisic et al., 2016). In order to standardize the architecture of these systems as well as the methodology for creating them, AUTOSAR proposes a reference architecture (see section 2.1.1) and a methodology for creating said systems. The AUTOSAR standard was initially introduced in 2003, with the first platform, now known as AUTOSAR Classic, launching in 2005 (AUTOSAR consortium, 2017b). Since then, the standard has been under continuous development and growth.

In order to facilitate new requirements from the next generation of smart vehicles, AUTOSAR also introduced a parallel standard to the Classic platform in 2017. This standard, called the Adaptive platform, aims to facilitate easy integration of new requirements such as *autonomous drive* and *connected vehicles* etc. with the existing embedded system functionality (Fürst and Bechter, 2016). Moreover, AUTOSAR defines a meta-model that facilitates how different people in the architectural chain should communicate in order to accommodate for different tooling and software used. This thesis will focus on features from the AUTOSAR Classic standard, as this is what is the most widely used at Volvo Cars Corporation.

2.1.1 AUTOSAR Reference Architecture

AUTOSAR Classic defines a reference architecture on how the ECU software in vehicles should be structured by splitting the software into 3 layers; the **Application Software**, **Runtime environment** and the **Basic software**-layers (Đurišić, 2017). These three layers each contain one abstraction level of the embedded soft-

ware. This structure facilitates the possibility of sourcing the different layers from subcontractors as each layer is defined by their ”black-box” behavior. Thus developers of a specific layer need not worry about the implementation of other layers but only fulfill the black-box behavior of their own layer and rely on the documented behavior of the other layers which the implemented layer will communicate with. For an illustration of this architecture, see figure 2.1.

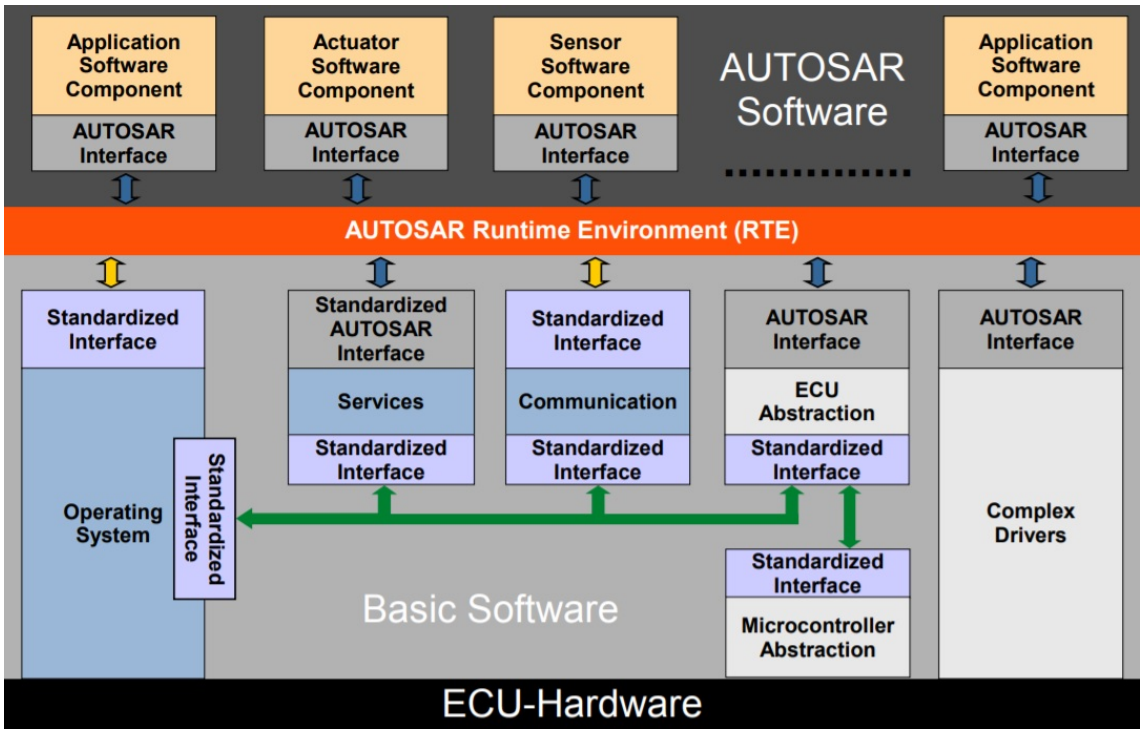


Figure 2.1: The AUTOSAR ECU architecture with the three abstraction layers; the *Application Layer*, *Runtime environment* and *Base software* layer. Derived from (AUTOSAR consortium, 2017a)

The layer with the highest abstraction level is the *Application layer*. This consists of a number of software components that each contains runnable software entities. These software components realize requested vehicle functionality and communicate with one another via the defined interfaces/ports (Đurišić, 2017; Bo et al., 2010). This layer can as such be seen as the one comprising the behavior that will be marketed to the consumers.

The second layer is called the *Runtime Environment layer* and handles the communication between the software components. This layer as such serves as a broker of information between the components as these may be located on different ECUs. By utilizing this runtime environment the location of the software components becomes irrelevant (Đurišić, 2017). The layer in itself is generated from the configuration and implementation of the base software layer at compile time.

Finally, the most ”basic” layer in terms of abstraction level is the *base software layer* (BSW) that consists of modules that will handle the basic ECU behavior and thus provides ECU-specific services to the software components (Đurišić, 2017). These modules are split into three different abstraction levels (figure 2.2), the *Services*, *ECU Abstraction*, *Complex Drivers* and *Microcontroller Abstraction*-layers, that each consist of modules that are related to that abstraction levels' corresponding ECU behaviour. For example, the *Microcontroller Abstraction Layer* consists of drivers and similar that are required to access some physical channels and behavior of the ECU.

As such, the modules standardized by AUTOSAR are split into functional abstraction levels and packages. In figure 2.3, a subset of the available BSW modules standardized by AUTOSAR are shown. In order to facilitate some ECU functionality, for example, communication via a CAN-interface (Controller Area Network), drivers from the *Microcontroller Abstraction Layer*, abstraction modules from the *ECU Abstraction Layer* and services from the *Services Layer* are required.

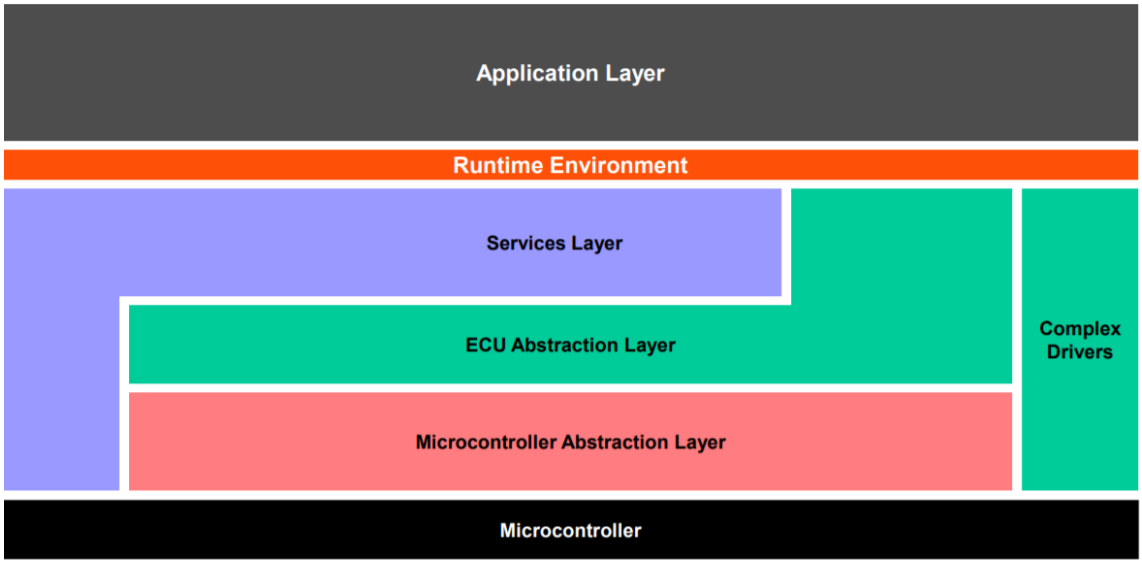


Figure 2.2: AUTOSAR BSW Abstraction layers. Derived from: (*Layered Software Architecture* n.d.)

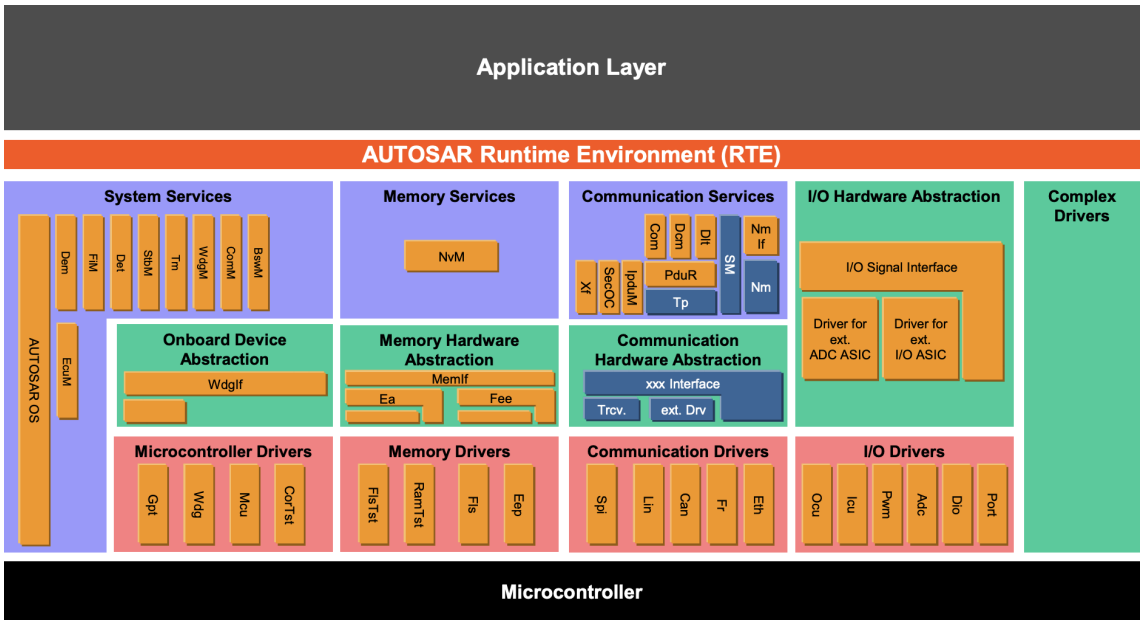


Figure 2.3: AUTOSAR functional clusters with a subset of the corresponding BSW modules. Derived from: (*Layered Software Architecture* n.d.)

The behavior of the different modules in these clusters is very well defined via extensive documentation provided by AUTOSAR. Moreover, these modules can be seen as realizations of different ECU characteristics that exist in ECUs. These characteristics include artifacts such as *Diagnostic Functionality*, *Communication on different mediums* such as Ethernet, CAN, LIN, FlexRay, etc. and various other characteristics that the base software instantiates via its different software modules. As such, ECUs can also be characterized based on what characteristics it has, i.e. what modules from the different functional clusters the ECU has in its base software layer.

2.2 Machine Learning and Neural Networks

With the increasing amount of information and data that various software system handles, approaches to compute and manage this data while still being able to identify important aspects and patterns from the data have made machine learning increasingly present in said systems. Machine learning, often based on dynamic statistical analysis on large amounts of data, comprises a large array of different approaches in predicting future behavior or patterns of a system based on previously collected data, one of which is Artificial Neural Networks.

Artificial Neural Networks(ANN) is a machine learning technique that mimics the intricate workings of a brain by introducing ”neurons” that works in a similar fashion to the neurons in a mammalian brain, i.e. takes inputs and computes some function on the input that yields an output. These neurons are then connected to one another in various ways to yield a network that given some input, computes an output for something we wish to predict (Dreyfus, 2005; Mehlig, 2019). The

network is then ”trained” in some fashion to become tailored to whatever problem domain we wish to make predictions in.

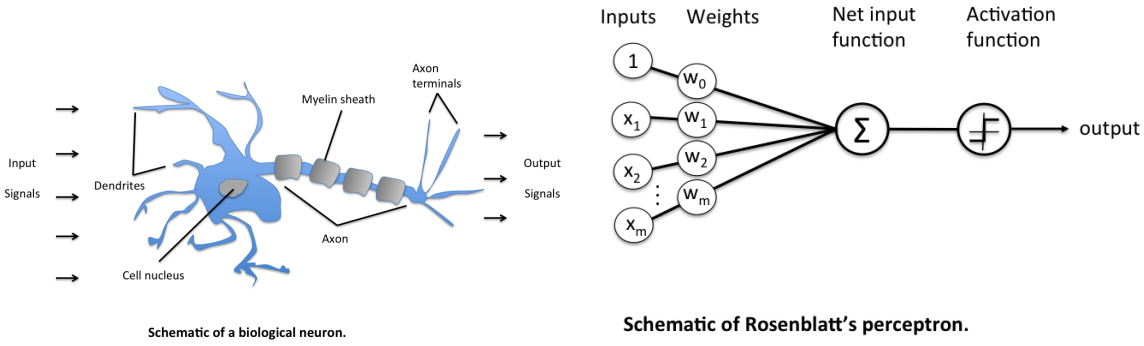
There are many usages for neural networks, often these are used when the relationship and patterns that exist in some data is so complex that they cannot easily be distinguished by humans or standard algorithms, but instead needs to be ”learned” utilizing an ANN. Two of the more common usages for ANN includes:

- **Regression:** We wish to be able to estimate an output given some input, for example estimating a function output for some input where the function itself is not known in advance. As such an ANN can be used for function approximation when the function itself is not known. Moreover, it can also be used for linear regression. (Radonja, 2012).
- **Classification:** We wish to classify data into some specific classes based on the characteristics of the data. One example of which is the possibility of clustering points together to identify cliques, or being able to identify and label data, for example identifying spam email (K. and Kumar, 2010).

As such, neural networks are a suitable approach when one either wishes to approximate some function, based on previous function inputs/outputs or when one wishes to be able to distinguish similar data characteristics and split the data into classes of some type. Moreover, as ANNs are widely used there exist an array of different types of network architectures that influence how the network behaves.

2.2.1 Neurons

The neurons are the fundamental building blocks of an artificial neural network. Similar to the functionality of a biological neuron, the artificial neuron takes several inputs and computes an output. For an illustration of the neuron and it’s artificial equivalence, refer to figure 2.4.



(a) Biological Neuron **(b)** Artificial neuron in ANN

Figure 2.4: An illustration of a biological neuron and an artificial neuron. Derived from (Raschka, 2015)

Thus, a neuron in a ANN is a nonlinear bounded function, that takes in a input vector $\mathbf{x} = [x_1, x_2, ..., x_n]$, and also a parameter vector $\mathbf{w} = [w_1, w_2, ..., w_n]$, and via a so called **activation function** yields a output by multiplying each input parameter x_i with the corresponding weight w_i, feeding the sum of all of these products into the activation function (Dreyfus, 2005). These parameters(weights) are then changed to fit the problem instance, such that the network produces what we want, the activity of changing these weights is called "training" and is explained more in section 2.2.3.

The **activation function** describes what kind of output the neuron should produce given its inputs and parameters. There are many different such functions, common functions include the *linear function*, *sigmoid function* and the *step function*. Selecting what activation function is needed is also dependent on the problem instance and what is the requested behavior for the network. Additionally, a neuron can have an additional term, called a **bias**, that slightly influence the behavior of the network. This can be used to accommodate factors that do not depend on the input such that the output of the neuron is as expected. The bias is often introduced as an extra input parameter to the problem.

Thus, the output y of a neuron can be explained by the following formula;

$$y = f(\sum_{i=1}^{n} w_i x_i + b)$$

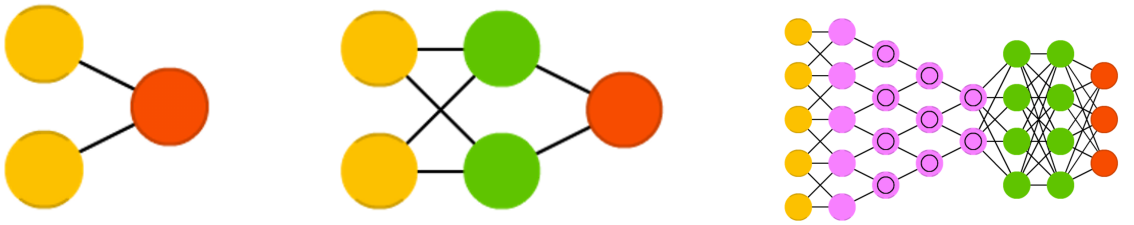
Where f is the activation function, $\mathbf{w} = (w_0,w_1,..,w_n)$ the parameter vector, $\mathbf{x}$ $(x_0,x_1,..,x_n)$ the input vector and $\mathbf{b}$ the bias.

2.2.2 Types of Artificial Neural Networks

There are many different types of neural networks that all share fundamental attributes such as interconnected neurons that are used to create some output from an input. How these connections between neurons are constructed and how layers and neuron structures are set is called a configuration or architecture of a network and is often what differs networks from one another. Below follows three different types of network architectures that exist (Čirkovs, 2017). These are further illustrated in figure 2.5.

- **Single Layer Perception:** The simplest kind of "network", this is essentially a stand-alone neuron that takes an input-vector and weight-vector, combines these and via an activation function computes a linear output. One example of this type of network is called *Rosenblatts' Perceptron* (Raschka, 2015). As these outputs a linear value, these can be used for linear regression.
- **Feed Forward Neural Network:** Also called *multilayer perceptron*. Contains several "layers" that are fully connected, and input 'flows' from one layer to the next, and does as such not contain backward-edges nor cycles. As this is what is intended to be used in this thesis, a more detailed description can be found in 2.2.4. Additionally, as these contain several neurons that form together with a complex function, these can be used to estimate more complex functions than the single layer perceptron.

- **Convolutional Neural Networks:** These kinds of networks are often used for image classification and processing in order to understand the characteristics of images. Often this type of networks contains many layers with different kinds of neurons that produce some classification. For example, classifying an image of a parrot and output ”parrot” (Čirkovs, 2017).



(a) Single Layer Perceptron

(b) Feed Forward Neural Network

(c) Convolutional Neural Network

Figure 2.5: Three different kinds of neural networks. Derived from (Veen, 2017)

As one can see, there are different kinds of networks that serves different purposes, as such, determining which one is suitable for a task often consists of recognizing the aspects of the task the intention of the network is.

2.2.3 Network Training

In order to make a neural network behave as expected, the network needs to be trained to ”tune” the weights of the neurons in the ANN. This is often done using observed data from whatever it is we want to make predictions on. There exist two types of training approaches, supervised and unsupervised training.

Supervised Training

In some cases, what function the neurons should compute analytically may not be known in advance. Instead, a set of data points that forms a sample of the function behavior may be available. In this case, one would like to approximate the function that yields the correct output given the input. This kind of training is called ”supervised training” as in this one provides the network with examples of input/output pairs that the network learns from. Supervised learning often comprises splitting the data into two sets, the *training set* and the *validation set*, where the training set is often bigger than the validation set. Using these two we train the network on our training set by using the network to compute an estimation of the output given the input. Once this estimation is computed we compare it against the correct output and slightly change the weights for all the neurons in the network to make sure that the network performs as intended. Once the training is completed, we validate the behavior on the validation set to ensure that the network also behaves correctly on data external to the training set. One example of supervised training for a feed-forward neural network is described in section 2.2.4.

Unsupervised Training

In unsupervised training, the network is not given any correct examples that it can use to fine-tune its inner working, but instead, it is provided with a large data-set that it uses to discover similarities, for example, clustering points. Unsupervised training for a neural network is thus about detecting familiarity.

Thus, depending on what data is available and what the intention of the network is, the choice of training becomes increasingly important.

2.2.4 Feed-Forward Neural Network

One of the most fundamental type of a neural networks is the so-called "Feed-Forward Neural Network" that consists of fully connected layers of neurons that send the output of one layer into layers that are further ahead in the network (see figure 2.6). This network type computes a nonlinear function on its inputs to yield a number of outputs (the number of output neurons). A feed-forward neural network is, in its' graphical representation, acyclic, i.e. there exist no cycles in the graph.

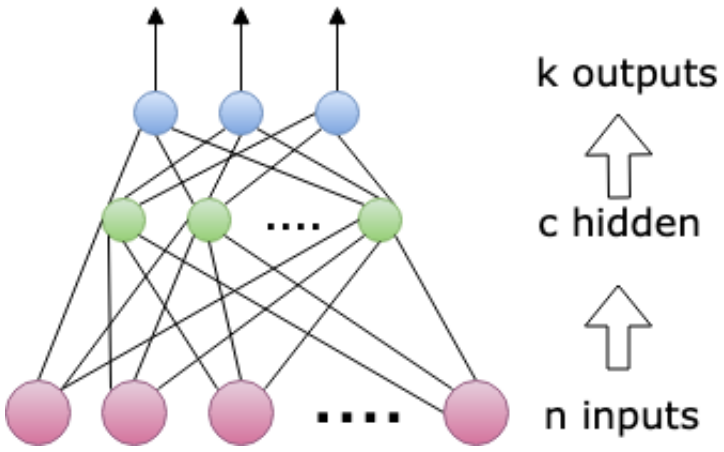


Figure 2.6: Feed Forward Nerual Network with one hidden layer

In the figure above, the neurons in the hidden layer computes **c** functions on the inputs where each output becomes a nonlinear output to the next layer of neurons. The final neurons in the last layer are called the *output nodes*.

The feedforward neural networks have been widely used in several fields and are very good for function approximation for complex functions. Other fields these have been used in includes *curve fitting*, *pattern classifications* and *system identification* (Tang, Tan, and Yi, 2007). In order to reach a satisfactory output from this network, the weights for every neuron needs to be approximated using some learning approach.

2.2.5 Estimating the Efficiency of a Neural Network

Once a neural network has been trained using some training approach and validated against some validation data, an analysis of the efficiency of the network will often be conducted. This estimation aims to answer whether the network can effectively be used to estimate or classify data in its problem domain. Often this analysis will revolve around what accuracy of correct predictions the network has once evaluated against the validation data. A common error function to evaluate the accuracy of a network is the mean squared error function, that works as the following:

$$e = \frac{1}{n} * \sum_{i}^{n} (\Theta_i - \hat{\Theta}_i)^2$$

Where Θ_i is the actual value for a sample i in the validation data, and $\hat{\Theta}_i$ is the predicted value. I.e. this value aims to sum up the squared distance between the predicted value and the actual value, and then compute the mean squared distance for the entire validation dataset. Another example of such an error function is the **MAE**(Mean Absolute Error) measure, which is the average absolute value of the distance between the actual value and the predicted value in the validation dataset.

What error measure is acceptable for a neural network is closely related to what the neural network is supposed to be used for. For some networks that are used to predict a rough estimate for some parameter, a larger error is generally acceptable, whereas a network for precise approximations, for example for some safety-related parameter, a smaller error measure is required. As such, what error is accepted is often decided based on the problem context, for example in an industrial project, the accepted error may be included in the project description.

3
Related Works

The undertaking of predicting hardware requirements for ECUs using machine learning approaches can be seen as supervised machine learning given that we wish to learn the combined factors on resource utilization for different groupings of running AUTOSAR BSW modules (characteristics). Increasing the number of characteristics does not necessarily cause a linear increase in used resources as these may have intricate relationships between them. The machine learning prediction models created during this thesis must as such "learn" the different relationships between these high-level characteristics and how they influence the resulting CPU, RAM and ROM utilization for ECUs. Additionally, since the models created during this thesis aims to introduce machine-learning tools to an already existing methodology, there may exist some difficulties of the adaptation of these new tools to the existing, non-machine learning approaches.

In order to determine what similar studies had been conducted in the area of hardware utilization prediction, hardware requirements selection in AUTOSAR compliant systems, hardware selection based on machine learning or machine learning adaptation in similar areas, a literature review was conducted by searching the *IEEE* and *Google Scholar* databases with the following phrases:

- AUTOSAR Hardware Selection
- ECU Hardware Estimation
- Hardware Estimation
- Machine Learning Hardware Estimation
- Machine Learning in Software Engineering
- Machine Learning AUTOSAR

Additionally, the keyword *"automotive"* was included in some of the keywords, and variations of these phrases were used to get wide coverage of studies in these fields. From this, it was determined that while machine learning for resource requirements prediction has seen little work in the automotive field, similar work has been conducted in predicting hardware utilization for large scale data-centers. Moreover, there exist several other studies of machine learning adaptation and usage in other fields of software engineering.

Machine learning has been used to predict hardware utilization for large scale data centers given that queries to said data-center grids can have various run-times and thus in order to balance the load on the servers, machine learning was introduced.

Ganapathi et al. (2009) showed that by accurately characterizing queries based on their features, accurate predictions of execution times and thus hardware requirements can be derived. This was achieved by constructing feature vectors for the queries and utilizing *kernel canonical correlation analysis* on the vectors to compute predictions on the runtime for the different queries. Doing this, accurate predictions were achieved, pointing to the fact that accurate characteristics are a valid foundation for machine learning prediction.

Moreover, Duan et al. (2009) showed that conducting machine learning prediction on activities in these high computational grids is possible by the introduction of a hybrid Bayesian-neural network that produced accurate predictions on execution time, and thus computational requirements based on the tasks and arguments for said activities. As such, this further points to the fact that accurate classification of activity aspects can be utilized to create accurate prediction models for computational resource requirements.

Additionally, machine learning has been used in several other areas of software engineering such as fault and defect prediction. In a study performed by R. Rana et al. (2014b), a framework aimed at making the adoption of machine learning approaches for organizations easier was proposed with fault prediction in mind. This study indicates that while adoption of machine learning has several advantages from a business standpoint such as increased prediction accuracy, the adaptation of machine learning tools for organizations is often not straightforward, a problem that the structured framework aimed to mitigate. Moreover, in a similar study by R. Rana et al. (2014a), it was shown that while companies perceive benefits of introducing machine learning as an addition to increase the accuracy defect predictions for products, the decision for adoption of the approaches often needs to include factors such as cost, generalizability and existing competence in machine learning to determine whether the adaptation is feasible. As such, while machine learning has several potential usages in the field of software engineering, the adaptation of these tools and approaches are often not straight-forward.

Given these studies, machine learning has advantages of uses in various areas of software engineering. Moreover, as machine learning has been used for hardware requirements prediction for cloud-computing centers, it also seems applicable to use as a prediction model for this thesis. More specifically, neural networks are intended to be used as these can create accurate regression models that can predict an outcome when the characteristic relationships are not known. Finally, while the machine learning models created in this thesis can mitigate validation and selection of hardware requirements for ECUs, the potential adoption of the models themselves can be difficult.

4
Research Design

This chapter describes the research questions this thesis wishes to answer as well as describing the methodological foundation on which this research method was conducted. Moreover, the specific variation of this methodology followed in this thesis is presented.

4.1 Research Questions

The following research questions and sub-questions are what this thesis aims to answer.

- **RQ1: How can hardware resource utilization be modeled given standardized software modules running on ECUs in the automotive domain?**
  - **RQ1.1**: Is standardized AUTOSAR BSW modules a suitable base for creating a prediction model to estimate hardware resource utilization of ECUs in the automotive domain?
  - **RQ1.2**: Can CPU, RAM and ROM utilization of an ECU be accurately estimated based on what AUTOSAR standardized characteristics an ECU has?
    - **RQ1.2.1** Are neural networks a more suitable model candidate than a simple multiple regression model?

The intended outcome of **RQ1** is to investigate whether the BSW modules standardized by AUTOSAR can be used as a foundation for hardware resource estimation. If a positive result is yielded here it would imply that resource utilization for ECUs can be derived based on what AUTOSAR-compliant base an ECU has.

- **RQ2: Is prediction models based on standardized AUTOSAR characteristics a suitable validation technique for hardware resources on an ECUs?**
  - **RQ2.1**: Can the prediction models propose an ECU that is at par or better in terms of hardware resource requirements than what have been selected?

RQ2 aims to investigate whether the models created answering **RQ1** can be successfully be used as a foundation to predict hardware requirements for an ECU, thus allowing for selection and validation of ECU hardware characteristics. As such, should a positive result be yielded here it would imply that suitable hardware for

ECUs can be derived based on what standard AUTOSAR functionality an ECU has, thus allowing for deriving ECU hardware requirements in early stages of developing ECUs or allowing to validate already delivered ECU on the feasibility of their hardware baselines.

4.2 Design Research Methodology

One of many research methodologies that exist is the **Design Research Methodology**. This methodology revolves around investigating some artifact in a context (Wieringa, 2014). In the methodology the following three parties are concerned.

- **Problem:** A problem is something that is wished to be improved upon in some environment. Examples of problems may be lacking operational or management processes, incomplete software or something else that is generally problematic or inefficient to a company.
- **Artifact:** An artifact is some entity that is created to aid in solving or improving upon the problem that may exist in an environment. Artifacts can be software or hardware, a new process or methods.
- **Environment/Context:** The context is in which the problem aimed to be improved by an artifact exists. This may be operational processes such as methods or areas such as software or hardware systems.

Design research is about introducing or investigating artifacts that are intended to solve a problem in some context. Further, as there often may exist a gap between what is developed and invented in academia to what is in practice in industry, design research becomes a suitable research methodology to introduce artifacts to solve problems by utilizing advancements in academia. This methodology has seen an increasing utilization in the later years, in fields ranging from education to software engineering and information systems (Dresch, Lacerda, and Valle, 2015). Branching from this methodology, the so-called **Design Science Research** operationalizes the concepts of design science and provides a workflow when the outcome of a study is intended to be an artifact solving some problem or improving upon an already existing solution. Moreover, the outcomes of design science research are contextual knowledge about the artifact/-s that have been created during the research (Johannesson, 2014). As we in this thesis want to introduce an artifact that aids in architectural design and validation of AUTOSAR-compliant systems, we utilize this research methodology.

4.3 Design Science Research Approach

As mentioned in the previous section, **Design Science Research** operationalizes the concepts of design science and provides a workflow when the outcome of a study is intended to be an artifact solving some problem or improving upon an already existing solution. As we in this thesis wish to improve upon an already existing methodology by introducing a tool that allows OEMs to validate sections of the hardware aspects of a delivered subsystem or propose a suitable hardware baseline for a new project, this becomes a suitable research approach. In our domain we have the following definitions:

1. **Problem:** Validating hardware requirements and overhead for ECUs in embedded system, or provide a suitable hardware baseline for a new ECU project.
2. **Artifact:** A tool that given AUTOSAR characteristics of the ECU can give a baseline recommendation for hardware characteristics.
3. **Context:** Validation of a delivered embedded system to an automotive OEM or proposing suitable hardware baselines for new ECU projects.

.

Following the proposed design workflow presented in (Dresch, Lacerda, and Valle, 2015), the steps necessary for conducting design science research and their respective outcomes are presented in the following figure.

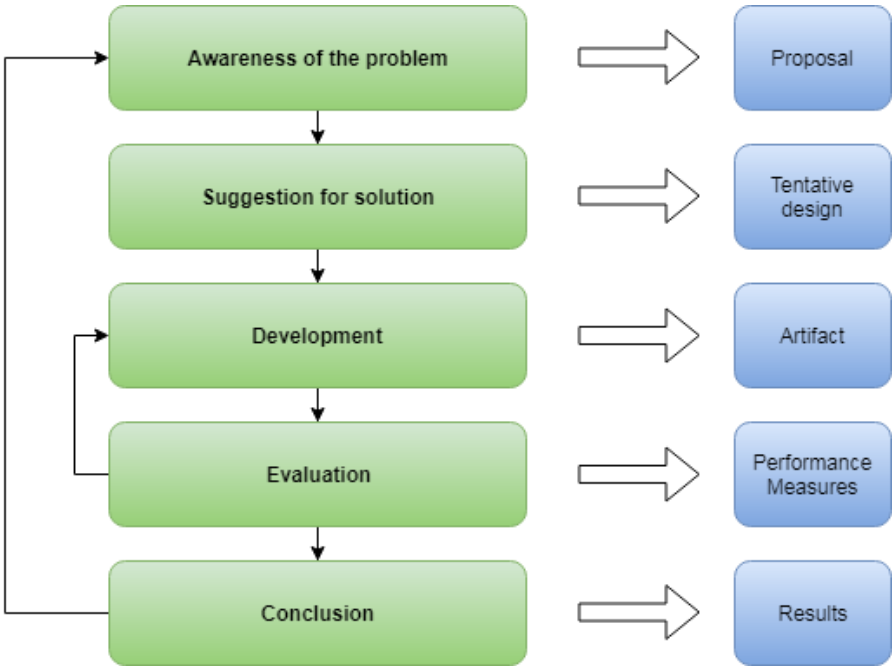


Figure 4.1: Steps for conducting design science research

Initially, design science research is about understanding the problem and gather knowledge about the domain such that the study can be conducted to yield a satisfactory artifact. Once this is done and the area is understood, the researcher moves on to an iterative approach for creating the artifact, evaluating it and makes adjustments until a satisfactory outcome is reached. As such, the different steps essentially

boils down to the following

1. **Awareness of the problem:** Understand the problem domain and the entities surrounding and influencing the problem domain, in order to yield a proposal for undertaking the research and what could be improved by introducing a new artifact. This could be reached by a case study in the area or other knowledge acquiring approaches such as a literature review.
2. **Suggestions for solution:** In this step a solution must be proposed to solve said problem, often by bridging academic, theoretical knowledge and the problem. This will yield a proposed design for an artifact. Examples of activities in this step include literature reviews on similar studies.
3. **Development:** Actually producing said artifact to solve the problem, using the design proposed in the previous step. Will yield the artifact.
4. **Evaluation:** Evaluating the artifact against some success criteria to see if the developed artifact solved or mitigated the problem. If not, the artifact could be developed further. This will yield performance measures on how usable the artifact is in regards to the problem. These performance measures could be yielded from a case study or experiments on the artifacts feasibility.
5. **Conclusion:** The final step is the concluding remarks on the research that yields the results. This can also yield more contextual problem knowledge that leads to a new research undertaking to create an alternative artifact should the created artifact prove to be inept. The results here can also be generalized to be used in a wider scope than what is presented in the study.

Applying this research methodology in our study can be boiled down to the same approach with the same steps. The resulting workflow can be seen in figure 4.2.

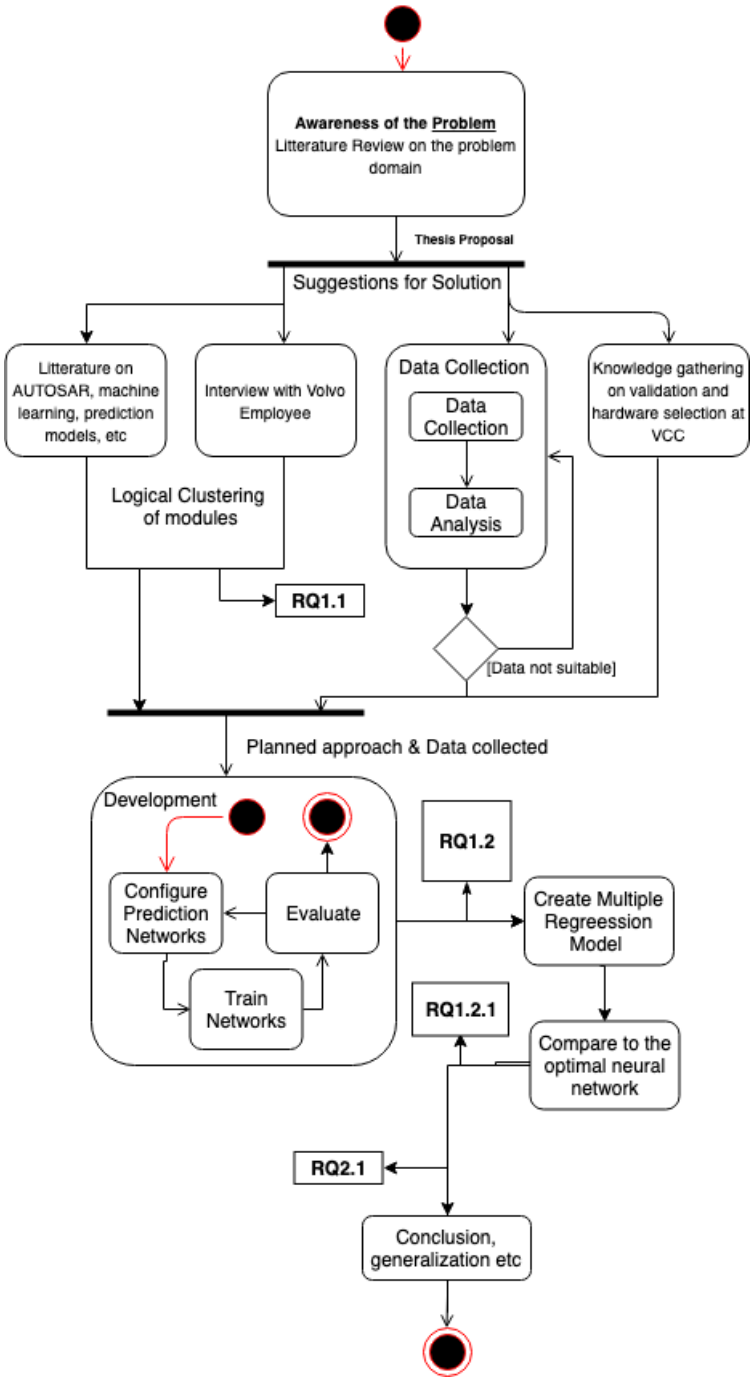


Figure 4.2: The general design science research-workflow followed in this thesis. The arrows indicate a transition from one activity to the next. The thick lines indicate a split into several activities that are performed in parallel or the joining of parallel activities.

4.3.1 Awareness of the Problem

Understanding the problem we have been faced with for this study was reached by industrial information and wishes from Volvo Cars Corporation. Moreover, in order to further understand the problem domain, literature studies on different aspects of

AUTOSAR, such as the methodology and the standardized modules as well as on machine learning was conducted. This yielded the first conceptual understanding and proposed an approach to solving the problem presented by the company.

4.3.2 Suggestions for Solution

In order to reach a more detailed understanding of the problem in its' context and to be able to quantify the necessary steps for reaching the intended outcome, an additional literature review was conducted on the areas related to the study such as AUTOSAR and machine learning. The resulting knowledge formed the initial idea for the models that were to be used in order to reach the intended outcome.

Moreover, in order to quantify how to best determine what software modules existed in a given ECU, such that ECUs could be characterized, as well as what AUTOSAR BSW modules are commonly found in Volvo Cars Corporation ECUs, an interview with a Volvo Cars employee was conducted. This interview allowed for the knowledge required to answer **RQ1.1**.

Furthermore, knowledge gathering on how ECUs and hardware characteristics are validated at Volvo Cars, as well as how suitable hardware baselines for new projects are chosen was conducted. This consisted of informal chats with employees at the company as well as a semi-structured interview with a system expert.

The outcomes of this step were as such the approach for creating the estimation-model as well as the theoretical foundation on which the models were built and information about the current methodology for selecting and validating hardware. In essence a formalization of the problem and potential solution.

4.3.2.1 Data Collection

Additionally, data on CPU and memory utilization from different ECUs at Volvo Cars was collected in this step. The data in itself was also analyzed to get the same foundation on the prediction model, i.e. that all ECUs had the same basis.
The data were collected from *Software Version Descriptions* (SWVD) that are created for any release of an ECU at VCC. These documents contain information on the hardware resources for the specific ECU, often comprising the following:

- ROM (Flash Memory) utilization and total ROM [kilobytes]
- RAM utilization and total RAM [kilobytes]
- EPROM utilization [kilobytes]
- CPU Load (% of total)

From these documents the requested data for different ECUs (CPU load, RAM and ROM utilization) was collected, data for various versions of said ECU was also collected as the resource utilization may change between releases (updated configurations and software implementations). Additionally, the total CPU clock frequency

of the different CPUs in the ECUs was collected by contacting the responsible teams for the ECUs and from this, the CPU frequency utilization at runtime was computed.

4.3.3 Development

Neural networks were selected as a viable candidate for the prediction model. This approach was chosen at it is not evident how the relationship between high-level features of ECUs and the resource utilization is defined. As such, the function that, given a set of high-level features for an ECU, can predict an estimation on CPU, RAM and ROM utilization is not known. Since we collect sample data of resource utilization for ECUs and can identify the high-level features of these ECUs we have examples from our problem domain and thus neural networks become suitable given that neural networks "learn" by example from problem domains by iteratively utilizing sample data to approximate the relations between the input and output.

Following the decision of utilizing neural networks, these prediction models were created utilizing the machine learning library Tensorflow™ (Martin Abadi et al., 2015b). This library was chosen as it is widely used in industry, is widely documented and contains a lot of prebuilt assets that assist in building the neural networks. Using this library reduced the time for building and configuring the networks.

The models consist of three separate neural networks, one for CPU utilization, one for RAM utilization and one for ROM utilization. These networks were trained on the data collected by splitting the data in a training and a validation set, training the models on the training set and validating on the validation set. The error function used to train the networks and tune the weights was chosen to be the *Mean Squared Error function (MSE)*, which is defined as:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(Y_i - \hat{Y}_i)^2$$

Where $\hat{Y}_i$ is the predicted value for a point i in the training set, and Y_i is the actual value. As such, the goal during this phase was to reach an MSE as low as possible on the training set, for different configurations of the network. This function was selected as it is commonly used in industry and yields large errors when there is a large difference between the predicted value and the actual value, thus by minimizing this error the networks predict measures that are close to the actual, and thus yields accurate predictions on resource utilization.

In order to find the most accurate configuration for the network, various parameters in the network were changed. However, as there exist a wide array of different network configurations and architectures, the approach of yielding the most efficient one followed a systematic experimentation approach where the expected outcome was a feed-forward neural network with a number of dense layers, as this type of network is generally applicable for regression. Moreover, in order to determine what characteristics should be tested, the number of different configurations were limited by selecting the following configurable parameters.

1. Number of hidden layers
2. Number of neurons per layer
3. Activation function per layer
4. Number of epochs (times the training data will be used to compute the neuron weights)

These parameters were selected to be the configurable ones as by changing any of these, we considerably influence the behavior of the network. Thus by testing different configurations for these parameters, we have covered a large array of different network behaviors. More than one of these parameters was not changed at a time. Furthermore, in order to limit the amount of different configurable parameters even further, the maximum number of tested hidden layers were limited to 5, as this would still yield a deep network able to identify hidden relationships between the features in its upper limit, while reducing the risk of overfitting the data should the dataset be small, which would potentially give an inaccurate validation result. Moreover, in relation to common aspects of software development, it was further decided to work in bases of 2 for the number of neurons. Finally, the activation functions selected was based on what was available in the Tensorflow library and further selected by creating a simple feed-forward neural network with a single hidden layer of 64 neurons and testing out every available activation function, selecting the ones that yielded smaller MSE measures. Moreover, the number of epochs (number of times the training data is used to train the network), was selected to be 50, as this would reduce the time for testing every configuration in comparison to using a larger number of epochs. Finally, the output neurons were selected to use the *relu* activation function, as this would always yield non-negative continuous outputs. As such, the following parameters to be tested were the following:

- **Number of hidden layers:** 1-5
- **Number of neurons in a layer:** [16, 32, 64, 128, 256]
- **Different Activation functions**
- **Output neuron using relu activation function**

The followed workflow for computing the best configurations for the 3 networks consisted of optimizing each hidden layer before adding on another one. By optimizing each layer before adding on another one can expect to send qualitative data to the next layer. More formally, each layer would be tested with each of the different candidates for the number of neurons, with each activation function. The most optimal configuration, determined by the evaluated prediction accuracy on the validation data described in section 4.3.4 would serve as the final configuration for that layer. Each layer would be configured in this way and would be done until the exhaustive search was completed and each layer had been optimized. Every configurations accuracy would be saved in order to be able to select the most accurate configuration. Once the most accurate configuration was found, the epochs would be selected by visualizing the training loss during the training of the optimal model and selected to a number where the loss would converge or before the loss would increase again. At this point, we can assume that by retraining the network again for the same

configuration and number of epochs, we would get a very similar network behavior, and thus provide a result that could be reached again.

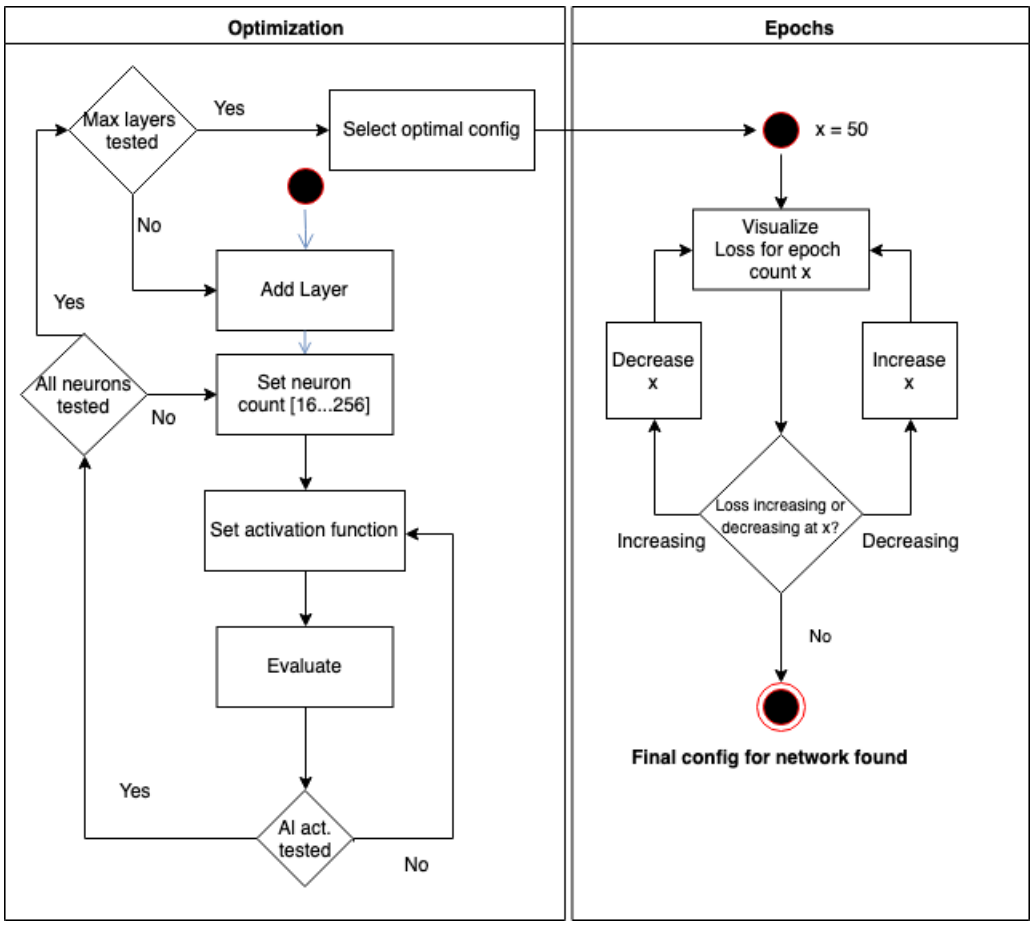


Figure 4.3: Workflow when creating the neural networks for predicting the hardware measures

Once the parameters had been selected in such a way that the highest accuracy had been found, it was concluded that the best possible result had been found. The resulting models answers **RQ1.2** and the accuracy of these answers **RQ2.1**

4.3.4 Evaluation

During the configuration testing of the different networks, these were evaluated against the validation data. More formally, this validation consisted of using the networks to estimate a measure for the respective parameter that was tested, for example estimating a measure of RAM utilization for the ECUs in the validation set. Once this estimation was completed it was rounded up to the nearest baseline available for the metric. This was done as hardware resources usually comes in predefined bases, for example, **2kb**, **4kb** and so forth. These baselines would be based on what baselines exists for hardware resources in ECUs at VCC. i.e. every estimation would be rounded up to the nearest metric available in VCC vehicles, up to the highest identified ECU.

Once this baseline was selected, it was compared against the actual baselines for the ECUs in the validation step. Should the predicted baseline for an ECU be the same as what was actually the baseline for the ECU, or the predicted baseline be lower than the actual while still being above the runtime utilization for the metric (CPU, RAM or ROM) thus giving some overhead, we would have a successful prediction. However, should the predicted baseline be above what the actual baseline was, or that the predicted baseline was lower than the runtime utilization for the respective metric, we would have an unsuccessful prediction. The accuracy of the different configurations was then defined as

$$\textbf{Accuracy} = \frac{Number\ of\ successful\ predictions}{Size\ of\ validation\ set} * 100\%$$

.
As such, the configuration that would have the highest such accuracy would be selected as the final network configuration.

Finally, in order to ensure that neural networks are a suitable approach for creating the models solving this specific problem, an alternative in the form of a simple multiple regression model would be created. This alternative would also utilize the identified characteristics as input and be created using the training data and validated against the validation data. The resulting accuracy of this model would then be compared against the neural network to see whether the neural network approach is apt. As such, the accuracy for the CPU, RAM and ROM-models for the optimal configurations of the neural networks would answer **RQ2.1**, and the comparison against the multiple regression approach would answer **RQ1.2.1**.

4.3.5 Conclusion

From the assessment of the tool, the results from the study were analyzed. This yielded a general assessment of the feasibility and usability of the artifact. Moreover, this concluded what impact this study has on the industrial field of embedded system design and whether or not the proposed artifact successfully solved the problem that was identified in the first step. Further, the study was generalized and extensions to the research field and new research undertakings was identified, such that the field could be further extended.

5
Results

This chapter presents the results yielded during this study. Moreover, the resulting models and their accuracy is presented.

5.1 AUTOSAR BSW Logical Grouping

The literature review on the AUTOSAR Classic base software modules allowed for the identification that the clustering AUTOSAR proposes for the grouping of BSW modules are not a suitable baseline for describing an ECU in a way that would allow for the creation of a prediction model. This fact became apparent given that AUTOSAR by default groups the BSW modules in functional clusters, such that, for example, all the communication drivers exist in one group. Consequently, there exist strong dependencies between these groups such that they cannot be isolated on their own. For example, in order to facilitate communication via a CAN-network, an ECU would require modules from several of the functional groupings.

Initially, the idea was to use the functional groups proposed by AUTOSAR as a feature vector to serve as input to the different prediction models, but as these groupings thus overlap for various high-level features (for example for different types of communication), the resulting feature vectors would be very similar for many ECU nodes. Given this, it was identified that regrouping the BSW-modules into logical clusters would allow for the separation of different high-level features and allow for easier identification of the different features for a given ECU. Moreover, data on what specific BSW-modules exists in a given ECU is currently not readily available at Volvo Cars Corporation. This lack of data is due to the fact that the base software is often delivered by subcontractors, that were reluctant to give away this information as this may infringe on their business interests. As there also exist over 100 stand-alone BSW modules standardized by AUTOSAR, these are not a realistic feature vector baseline either as each input vector would become very long. As such, restructuring the modules into logical clusters realizing high-level features became a more feasible approach as this would allow for easily distinguishing different features. As such, by a literature review on the different AUTOSAR BSW modules, a proposal for a logical grouping was created.

In order to validate the resulting understanding of the modules, and get the contextual knowledge required to assume that the proposed clustering was apt, an interview with a Volvo Cars engineer that works closely with the base systems of

ECUs and the AUTOSAR standard was conducted. Given that the engineer was considered to have such an advanced level of knowledge about the AUTOSAR standard and the context in which the standard is used at VCC, as well as the fact that the proposed clustering of modules was based on AUTOSAR documentation, the results of this single interview were deemed sufficient to ensure that the resulting clustering was realistic. Details of the interview can be found in Appendix A. The interview conducted with the Volvo Cars employee aimed to validate the proposed logical grouping of the base software modules and get contextual knowledge on what version of the AUTOSAR standard such that the potentially available BSW modules could be distinguished. Moreover, the interview aimed to yield knowledge of what standardized AUTOSAR functionality was used in Volvo Cars developed systems. During the interview, it was identified that most ECU teams at Volvo Cars use version **4.3.1** of the standard for the BSW in the ECUs. Following this, the engineer also shared the understanding that the functional clusters proposed by AUTOSAR are not a suitable baseline for a feature vector. Additionally, after the interview, the engineer was shown the resulting logical clustering and believed it was apt for explaining ECUs at VCC.
From the interview, six different types of ECUs (nodes) that exist in Volvo cars embedded systems were identified. These are:

- **CAN Nodes**: CAN-based communication, these have all of the default identified modules plus memory and diagnostics.
- **Flexray Nodes**: FlexRay-based communication, these have all of the default identified modules plus memory and diagnostics.
- **Ethernet Nodes**: Ethernet-based communication, these have all of the default identified modules plus memory and diagnostics.
- **LIN Nodes**: LIN based communication.
- **LIN Slaves**: Very simple LIN-nodes with limited computational power, such as actuators and sensors. only runs LIN communication.
- **MOST-Nodes**: Nodes that are connected to other nodes via a MOST-connection, an optical sound connection (mostly the infotainment nodes that handles sound).
- **Central Nodes**: These are high-computational central nodes that handle different clustered behaviors of other ECUs, i.e. other ECUs are connected to these types of nodes. These are often also nodes that fulfill the characteristics of the other types of nodes presented in this list.

A node can also be a mix of these types, for example, a node that fulfills the requirements for both CAN and Flexray.
The resulting logical grouping that was yielded from the interview and literature review can be seen in figure 5.1. The characteristics of these different groups reflect what is present in Volvo systems.

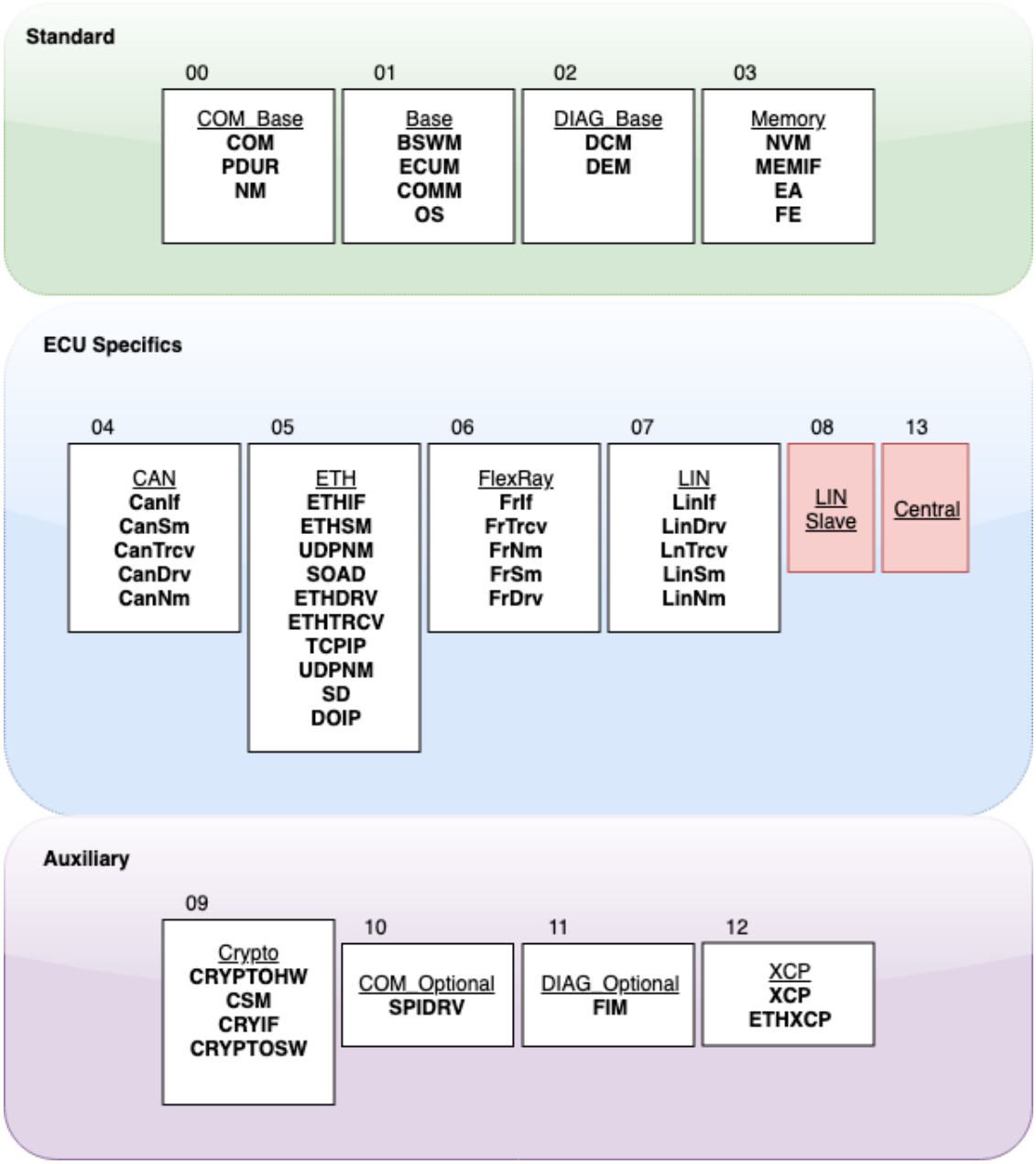


Figure 5.1: The resulting logical groupings of common BSW modules in ECUs

The logical groupings can be split into three distinct types:

1. **Standard Groupings:** These are groupings and characteristics that most of the nodes have, as they all require the base behavior of embedded systems, utilizes memory, communication and save diagnostic data.
2. **ECU Specific Groupings:** Generally these groups identify what type of communication and thus what type the node is, with the special LIN-slave characteristic to accommodate low computational units and the Central characteristic to distinguish potential high computational units that other nodes are connected to.
3. **Auxiliary Groupings:** These groups contain behavior that is often not standard in Volvo Cars ECUs but may nevertheless exist. The groupings here includes
   - **Crypto**: Possibility to encrypt sections of a system
   - **COM_Optional**: Possibility of reading and storing data in external memory
   - **DIAG_Optional**: Possibility of controlling running software components
   - **XCP**: Synchronous communication between nodes

These groupings all have an identifier that allows for creating a feature vector by denoting what groupings an ECU has. Finally, this logical grouping makes it easy to deduce what kind of characteristics a given ECU has.

5.2 Data Collection

The data used to create the models were collected as described in section 4.3.2.1 and was then characterized into the different logical clusters described in section 5.1. In order to characterize the ECUs, system typologies were consulted to get the communication details for the nodes and the SWVD-documents were consulted to get additional details. The information requested to grant an ECU the different features presented in figure 5.1 are described below.

0. **COM_Base:** Generally, being a standard CAN/FlexRay/Lin/Ethernet node, not being an infotainment node or a lin slave. I.e. have some sort of standard communication.
1. **Base:** Not being a LIN Slave, or infotainment node.
2. **DIAG_Base:** Not being a LIN Slave.
3. **MEMORY:** Not being a LIN Slave, or infotainment node.
4. **CAN:** Being connected to a CAN-bus (details of the system topology)
5. **ETH:** Being connected to an Ethernet-bus (details of the system topology)
6. **FlexRay:** Being connected to a FlexRay-bus (details of the system topology)
7. **LIN:** Being connected to a LIN-bus (details of the system topology)
8. **LIN Slave:** Being connected via a LIN-connection and having a very low level of computational resources (low RAM and ROM).
9. **Central:** Having several nodes connected to this node on the system topology, i.e. this node facilitates communication to other nodes.
10. **Crypto:** Specified in the SWVD that sections of the system are encrypted
11. **COM_Optional:** Specified in the SWVD that the SPI-driver is active or that the ECU uses SPI.
12. **DIAG_Optional:** Specified in the SWVD that the node uses FIM.
13. **XCP:** Specified in the SWVD that the node uses XCP.

These features were then used to create a binary feature vector where $x_i = 1$ means that the ECU is identified to have the i_{th} feature. For example the feature vector

$$x = \Big[1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0\Big]^T$$

is a specification of a standard CAN-node with XCP functionality. Moreover, during this phase, the ECUs that were deemed unfit for our prediction model was removed. These nodes was essentially only nodes that did not have any AUTOSAR modules (as determined in the interview described in section 5.1.

5.2.1 Collected data

The data used for the model contains several different ECU types, the count for each of the categories can be found in table 5.1, and a visualization of the module count in figure 5.2.

Characteristic	**Count**
COM_Base	101
Base	101
DIAG_Base	106
MEMORY	101
CAN	91
ETH	9
FlexRay	18
LIN	72
LIN Slave	15
CRYPTO	1
COM_Optional	25
DIAG_Optional	1
XCP	28
Central	5
Total datapoints	**121**

Table 5.1: Count of the different characteristics the collected data

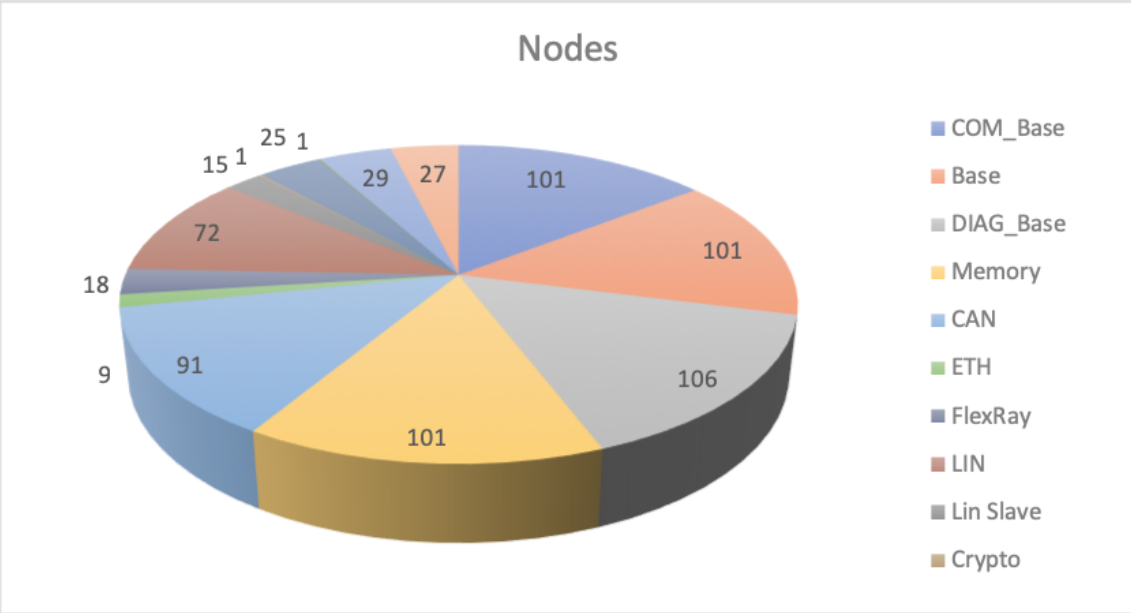
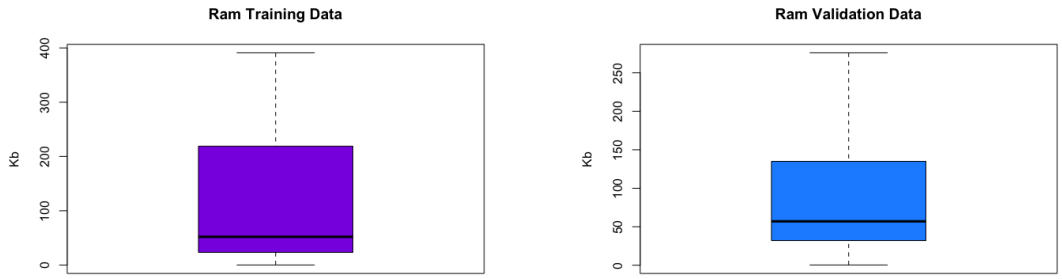


Figure 5.2: Visualizations of the occurrence of the different ECU characteristics

5.2.2 Training and Validation Data Split

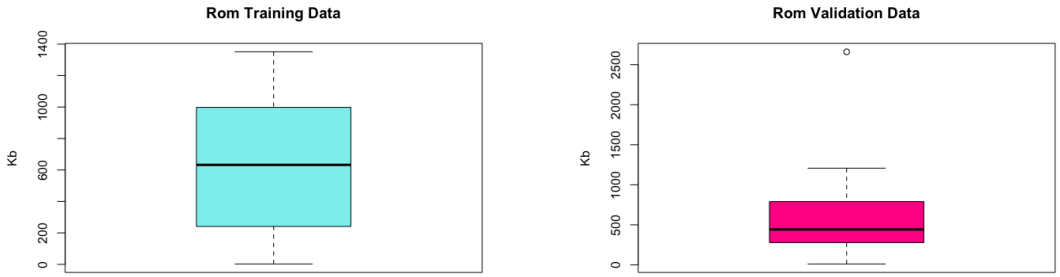
In order to be able to accurately validate the created models, the CPU and RAM/ROM datasets were split into two sets, training datasets that were used to create the prediction models for CPU, RAM and ROM estimation, and validation sets to determine whether the created models could successfully predict reasonable hardware requirements.
The validation sets consists of 14 measurements for the RAM and ROM predictions and 10 measurements for the CPU predictions, all from the same Volvo release (one of several projects for vehicles the company is undertaking). Moreover, all data points in the validation sets are from different ECUs within three major releases. As such, the training set consists of the remainder of the full dataset.

The amount of RAM and ROM used for the different ECUs in the training and validation dataset for the RAM and ROM parameters can be seen in figure 5.3 and figure 5.4. Moreover, the CPU measures for the training and validation dataset can be seen in figure 5.5.



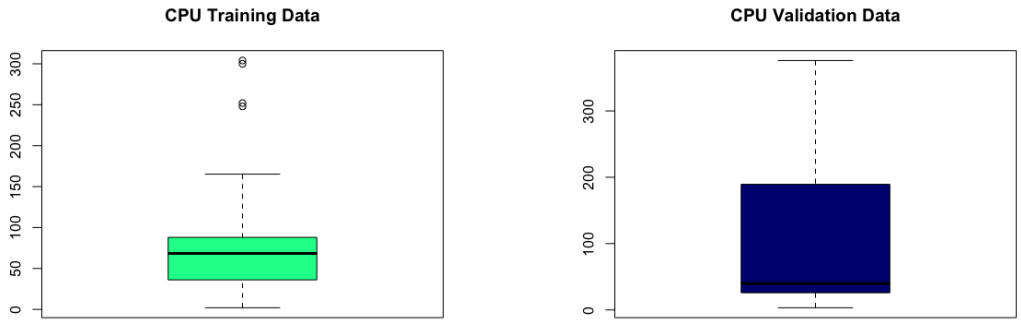
(a) Training Data **(b)** Validation data

Figure 5.3: RAM utilization boxplot for the training and validation set for our data, the measures are in Kilobytes



(a) Training Data **(b)** Validation Data

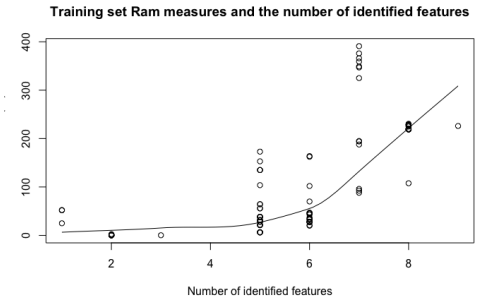
Figure 5.4: ROM utilization boxplot for the training and validation set for our data, the measures are in Kilobytes



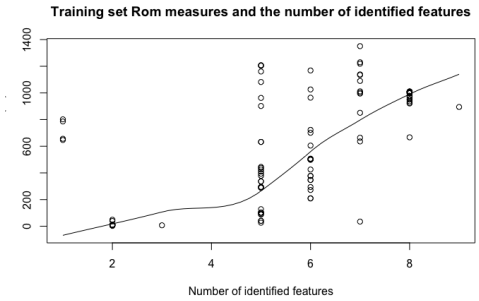
(a) Training Data **(b)** Validation Data

Figure 5.5: CPU utilization boxplot for the CPU training and validation set for our data, the measures are in Mega-hertz

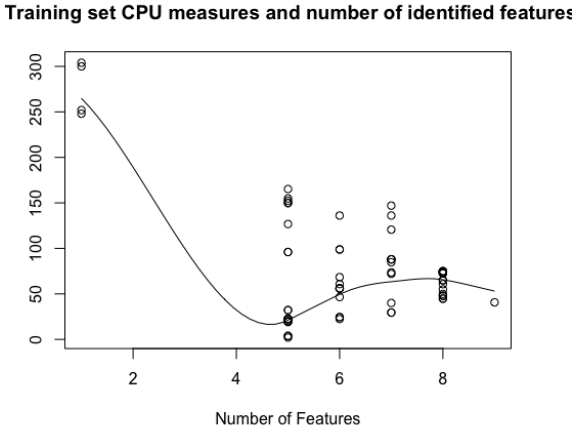
In order to firstly determine whether a standard linear regression model would be able to serve as foundation for a prediction model, we created a scatter-plot that visualizes the number of identified characteristics in correlation to the ROM, RAM and 32-bit CPU usage with a line of best fit, giving an indication whether a simple linear regression model could be used. From this we get the following figures:



(a) Training data RAM



(b) Training data ROM



(c) Training data CPU

Figure 5.6: Scatter diagram for the training datasets on the CPU, RAM and ROM measures with the corresponding best-fit regression line.

5.3 Hardware Validation and Selection at Volvo Cars Corporation

In order to be able to compare the prediction models created in this thesis to what is currently the methodology for validating and selecting hardware at VCC, contextual knowledge was required. In order to collect this knowledge, a semi-structured interview was held with a system specialist at Volvo Cars which had recently partaken in selecting a hardware baseline for a new ECU project. Moreover, this engineer has a lot of experience in building embedded systems at Volvo Cars Corporation. Additionally, once this interview was completed, another unstructured talk was conducted with another engineer which had also recently been involved in selecting hardware for a new ECU project. The formerly mentioned interview can be found in Appendix C.

Both of these knowledge gathering sessions indicated that in order to validate a given ECU, functional tests are run to validate the behavior of the ECU. Moreover, on selecting new hardware, the interviewee in the semi-structured interview explained that they based the selection of a suitable hardware baseline on a previous similar ECU, on which they added some extra overhead for the BSW layer and included a rough estimate on the estimated size and resource utilization for the top-level software components. Even so, these individuals failed to find suitable hardware with enough resources, causing them to have to rewrite the underlying algorithms for the software. As such, the selection of suitable hardware required tacit knowledge and a similar previous example from which an initial idea on a suitable baseline could be inferred.

From the informal chat with the other engineer, another approach was discussed in which they started with the ECU with the absolute highest amount of computational resources available to have a high overhead. As such, this approach required no former knowledge of similar ECU projects, but would instead cause a potential large extra initial cost per ECU should every project follow this model. Moreover, the engineer that partook in the informal chat also mentioned that subcontractors usually proposes a suitable ECU baseline which as such makes the hardware selection rely on subcontractor knowledge.

5.4 Neural Network Construction

The neural networks models were constructed using Tensorflow, and are feed-forward networks for regression. Three such networks have been created, one for each metric. These existing networks take an instance of the identified feature vector described in the previous section as input and output an estimation of the utilization for that metric.

In order to train these networks, the training datasets were split into two segments each, one for training the networks using supervised learning, and one segment for internally validating the networks during the training to set the weights of the neurons. The splits that have been used are 80% training size and 20% internal validation size for both the RAM/ROM dataset and the CPU dataset. During the development of the networks, different configurations have been tested in order to reach the lowest possible error, determined by the **mean squared error function**. To reduce the number of different configurations that were tested, three parameters were selected that changes for different configurations while any other parameter is fixed. The parameters that change for the configurations are; the *the number of hidden layers, activation functions* and the *number of neurons per layer*. In order to streamline the configuration search, a script was created that automatically computes the best network architecture by testing different configuration.

The activation functions that was chosen to be included in the possibilities were based on what activation functions are supported by the Tensorflow library. Secondly, every such activation function was tested in a simple feedforward network on the testing data to determine which ones were suitable candidates. After these had been tested in the simple network, the functions that gave a large MSE error after training in comparison to the others were discarded (several times larger than the others). The resulting activation functions that were chosen to be included in the tried configurations are listed below (Martin Abadi et al., 2015a).

- *Exponential Linear Unit*
- *Linear Activation Function*
- *Rectified linear unit*
- *Scaled Exponential Linear Units*
- *Sigmoid*
- *Softmax*
- *Softplus*
- *TanH*

The automation script for finding the best network configuration aimed to model the configuration search described in section 4.3.3 and as such aimed to optimize each layer in terms of prediction accuracy on the validation data before adding on another one. Each configuration consisted of trying out a combination of the selected neuron count and activation function, training and validating this network on the training and internal validation data. From this, the configuration was used on the validation

dataset to propose a suitable ECU baseline for the metric by using the trained configuration to estimate the resource utilization at runtime for the metric (CPU / RAM / ROM) and rounding up to the nearest identified ECU baseline. Every such configuration was tested 5 times in order to compute the average prediction accuracy (this is due to the fact that Tensorflow initializes networks with randomized weights and as such the accuracy may change slightly between training sessions). All of these configurations, their accuracy and their average error (MAE and MSE) were saved. The entire such set of tested configurations can be seen in Appendix B. The prediction accuracy for the different configurations can be seen in Figures 5.7 - 5.9.

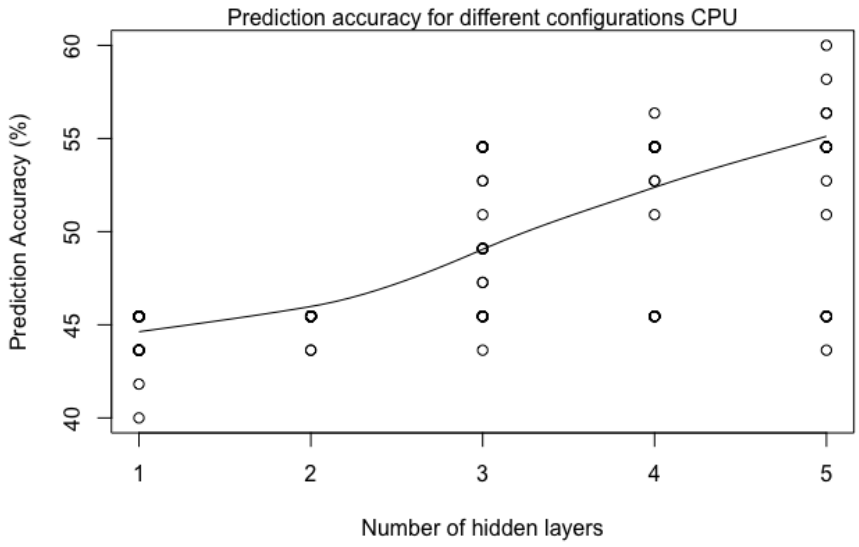


Figure 5.7: Prediction accuracy on the validation dataset for the different tested configurations for the CPU network

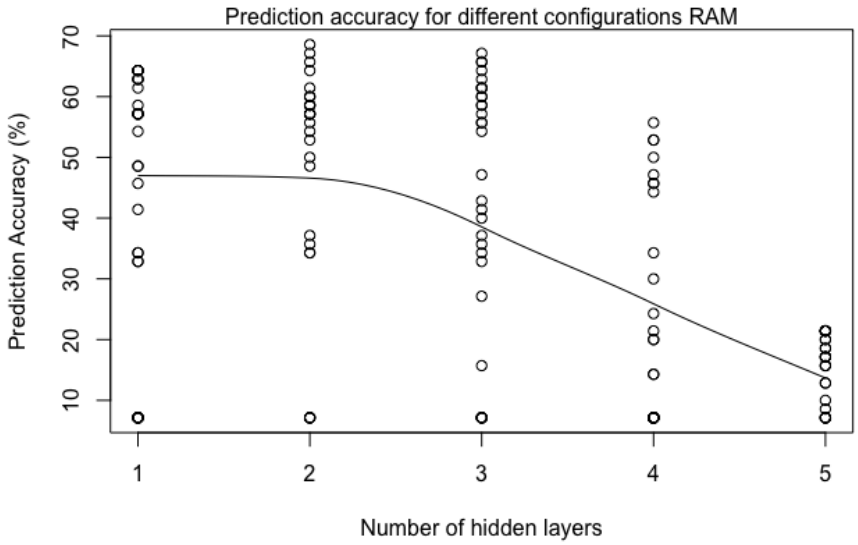


Figure 5.8: Prediction accuracy on the validation dataset for the different tested configurations for the RAM network

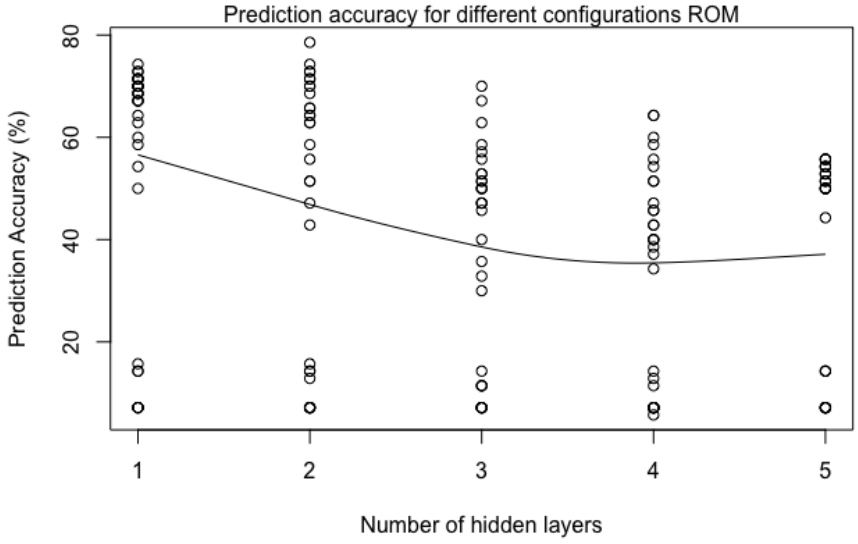


Figure 5.9: Prediction accuracy on the validation dataset for the different tested configurations for the ROM network

All of these top prediction was used in the validation step to finding the configuration with the highest percentage of successful prediction.

5.4.1 Epochs Selection

Selection of the number of epochs for the networks consisted of taking the top network for each parameter in terms of the prediction accuracy and visualizing the loss per number of epochs. Initially, the epoch count was set to 50 for each of the configurations. Should the losses decrease at this point the epochs was increased by 50 each time until the loss could be seen increasing or that the loss converged. The resulting loss visualizations for the top candidates in terms of validation accuracy (see section 5.5) are shown below.

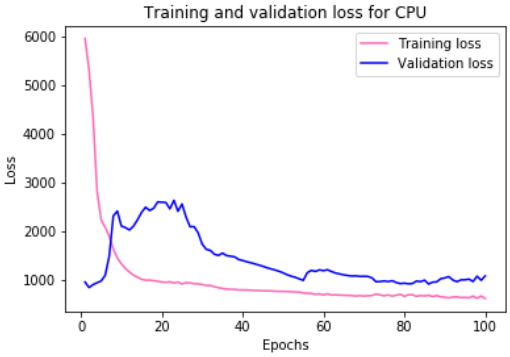


Figure 5.10: The loss per number of epochs for the top CPU configuration in terms of validation accuracy (CPU1)

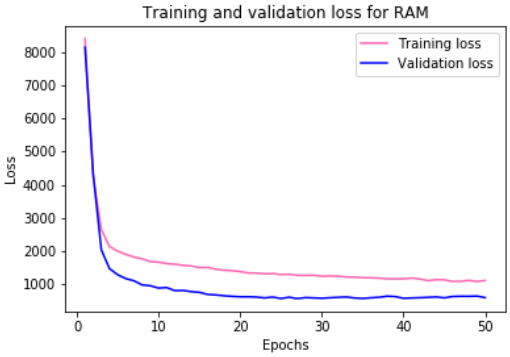


Figure 5.11: The loss per number of epochs for the top RAM configuration in terms of validation accuracy (RAM1)

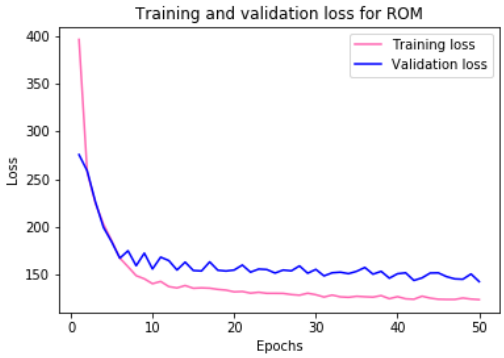


Figure 5.12: The loss per number of epochs for the top ROM configuration in terms of validation accuracy (ROM1)

5.4.2 Model Accuracy

Below, visualizations on the resource utilization predictions are shown. The shown networks belong to the top configurations in terms of validation accuracy (see section 5.5).

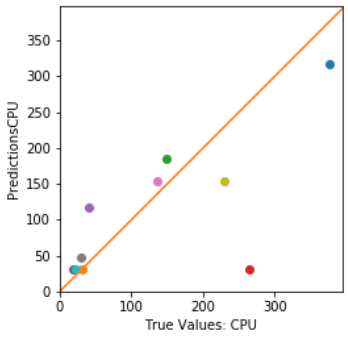


Figure 5.13: The predicted values and actual values of CPU resource utilization at runtime for the data in the validation dataset. The orange line signifies a line of best fit, i.e. correct predictions. The model visualized is the top configuration in terms of validation accuracy (CPU1)

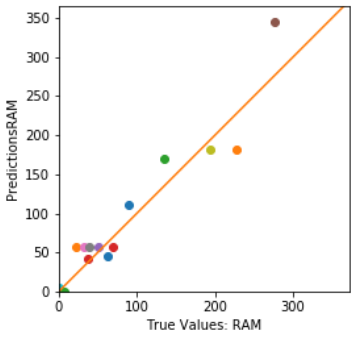


Figure 5.14: The predicted values and actual values of RAM resource utilization at runtime for the data in the validation dataset. The orange line signifies a line of best fit, i.e. correct predictions. The model visualized is the top configuration in terms of validation accuracy (RAM1)

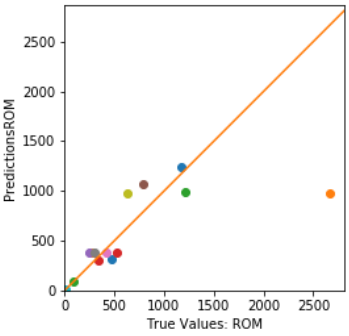


Figure 5.15: The predicted values and actual values of ROM resource utilization at runtime for the data in the validation dataset. The orange line signifies a line of best fit, i.e. correct predictions. The model visualized is the top configuration in terms of validation accuracy (ROM1)

5.5 Prediction Validation

As previously mentioned, every configuration tested were used to propose an estimation of the CPU, RAM and ROM utilization for the ECUs in the validation sets, after this was completed, it was rounded up to the nearest suitable baseline for the respective metric. These baselines are determined from what types of ECU metrics have been identified in Volvo Cars Corporation ECUs. The identified ECU baselines, as well as the accuracy and error of the top configuration candidates for the three metrics, are described in the following subsections.

5.5.1 Identified Hardware Baselines

Often total available hardware resources are defined in bases of 2, such that for example the RAM can have a size of 2kb, 4kb, 8kb, and so forth. However, this is not what has been identified in the SWVD for the ECUs used in VCC subsystems. By contacting the ECU teams and looking over what microcontrollers they have used in their projects we identified that some ECUs combines different storage chips such that the amount of total available RAM or ROM does not follow the standard convention of bases-of-2 measures. As such, we utilize the identified measures in VCC ECUs as suitable baseline hardware available for which the predicted measures will be rounded up. The identified ECU metrics follows in table 5.2

Id	CPU Frequency (MHz)	RAM Available (Kb)	ROM Available (Kb)
ECU1	32	8	127
ECU2	48	32	512
ECU3	48	32	512
ECU4	64	36	448
ECU5	64	40	576
ECU6	64	80	1024
ECU7	64	96	1500
ECU8	80	64	512
ECU9	80	64	768
ECU10	80	64	1024
ECU11	80	256	3000
ECU12	120	208	2112
ECU13	120	256	3072
ECU14	160	768	1792
ECU15	180	128	1024
ECU16	180	256	2048
ECU17	200	184	2048
ECU18	300	600	6872
ECU19	400	64	480
ECU20	451	208	2112
ECU21	1200	116	898

Table 5.2: All the identified ECUs present in Volvo Cars Systems, on 32-bit single core processors

Additionally, there also exist some non-32-bit ECUs that also have been used as suitable baselines for the RAM and ROM validation. These are shown below.

Id	RAM Available(kb)	ROM Available(kb)
ECU22	0.512	16
ECU23	1.024	16

Table 5.3: The non 32-bit ECUs identified

These will as such be used as our baseline for the validation of the models.

5.5.2 CPU Validation

Id	Num. hidden layers	Config	Accuracy	MSE	MAE
CPU1	5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)', '(linear, 256)', '(relu, 1)']	60%	5705.18	47.61
CPU2	5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)', '(softplus, 256)', '(relu, 1)']	50%	5869.58	48.14
CPU3	4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(linear, 256)', '(relu, 1)']	50%	6131.51	49.56
CPU4	5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)', '(linear, 32)', '(relu, 1)']	50%	6179.94	47.76

Table 5.4: The top 4 candidates for the CPU neural network in terms of prediction accuracy for the validation data

5.5.3 RAM Validation

Id	Num. hidden layers	Config	Accuracy	MSE	MAE
RAM1	2	['Input', '(relu, 16)', '(linear, 256)' ,'(relu, 1)']	68.57%	1010.50	22.44
RAM2	2	['Input', '(relu, 16)','(softplus, 256)','(relu, 1)']	67.14%	867.48	20.07
RAM3	3	['Input', '(relu, 16)','(relu, 16)','(relu, 32)', '(relu, 1)']	67.14%	704.19	18.95
RAM4	2	['Input', '(relu, 16)','(relu, 256)','(relu, 1)']	65.71%	831.29	20.18

Table 5.5: The top 4 neural networks configurations for the RAM predictions in terms of prediction accuracy on the validation dataset.

5.5.4 ROM Validation

Id	Num. hidden layers	Config	Accuracy	MSE	MAE
ROM1	2	['Input', '(linear, 16)', '(softplus, 32)', '(relu, 1)']	78.57%	302862.8	219.79
ROM2	2	['Input', '(linear, 16)', '(relu, 32)', '(relu, 1)']	74.29%	315975.3	222.08
ROM3	1	['Input', '(tanh, 256)', '(relu, 1)']	74.29%	251704.9	229.49
ROM4	1	['Input', '(relu, 256)', '(relu, 1)']	72.86%	266890.4	226.93

Table 5.6: The top 4 candidates for the ROM neural network in terms of prediction accuracy for the validation data

5.6 Additional Model Candidates

In addition to the neural classification network, one alternative has been created to study whether some additional prediction approach could be used. This approach consists of utilizing the statistics program R to compute a multiple regression model based on the training data to see what accuracy we can get from this on the validation data.

5.6.1 Multiple Regression Approach

The multiple regression alternative to the problem was created by utilization of the built in tools of R Studio to create a model based on the training data to predict CPU, RAM and ROM measures for the validation data. The results of predicting on the test data and rounding up to the nearest suitable baseline can be seen in the following table:

Id	Pred./Used CPU	Pred./Used RAM	Pred./Used ROM	Proposed/Actual CPU	Proposed/Actual RAM	Proposed/Actual ROM	Corr. CPU	Corr. RAM	Corr. ROM
ECUv1	276.00 / 376	45.25 / 63	722.75 / 467	300 / 400	64 / 64	768 / 480	No	Yes	No
ECUv2	46.25 / 31.36	31.40 / 22	354.19 / 278	48 / 64	32 / 36	448 / 448	Yes	Yes	Yes
ECUv3	131.88 / 149.8	123.03 / 135	1053.68 / 1208	160 / 200	128 / 184	1536 / 2048	Yes	No	Yes
ECUv4	46.25 / 149	31.40 / 68.6	354.19 / 522.4	48 / 1200	32 / 116	448 / 898	No	No	No
ECUv5	-	31.40 / 51	354.19 / 250	-	32 / 64	448 / 768	-	No	Yes
ECUv6	77.80 / 40	338.50 / 276	1024.38 / 790	80 / 160	600 / 768	1048 / 1792	Yes	Yes	Yes
ECUv7	-	31.40 / 32	354.19 / 418	-	32 / 768	448 / 768	-	Yes	Yes
ECUv8	46.25 / 18.4	31.40 / 38.4	354.19 / 294.6	48 / 80	32 / 64	448 / 1024	Yes	No	Yes
ECUv9	126.05 / 136.134	151.48 / 194	828.14 / 638	160 / 180	184 / 256	898 / 2048	Yes	No	Yes
ECUv10	-	0.55 / 0.301	14.46 / 11.256	-	1.024 / 0.512	16 / 16	-	No	Yes
ECUv11	85.64 / 29.44	149.93 / 89.5	1345.80 / 1175	120 / 64	184 / 96	1536 / 1536	No	No	Yes
ECUv12	126.05 / 228.9	151.48 / 227	828.14 / 2661	160 / 300	184 / 600	898 / 6912	No	No	No
ECUv13	16.95 / 22.4	10.39 / 6.4	251.2 / 93.6	32 / 32	32 / 8	256 / 127	Yes	No	No
ECUv14	-	37.66 / 37	333.99 / 347	-	40 / 67	448 / 512	-	Yes	Yes
Results							6/10 = 60%	5/15 = 36%	10/14 = 71%

Table 5.7: Multiple regression model predicted on the training data and computed accuracy for this model

From these results, it is shown that using the built-in multiple regression functionality of R-Studio allows for a 60% accuracy for the CPU predictions, a 36% accuracy for the RAM predictions and a 71% prediction accuracy for the ROM parameter.

6
Discussion

In this chapter, we present an analysis of our results presented in the previous chapter and discuss potential threats to the validity of the study.

6.1 Data Analysis

From the data we have collected, we can distinguish that the most common type of identified ECUs at Volvo Cars Corporation is the CAN-based ECU. Moreover, the standard characteristics are present in most of the CPU with the *DIAG_Base* characteristic being the most prevalent, which is also in line with that most of the ECUs in VCC vehicles should have diagnostic functionality. Moreover, the *Crypto*, *DIAG_Optional* and *COM_Optional* characteristics are very uncommon in our data. This rarity can also be an effect of that we failed to identify these metrics for some of the ECUs.

Unfortunately, only a smaller amount of data was able to be collected for building the models, this lack was caused by the fact that there currently exists few structured repositories of diagnostic data at the company. The absence of such a repository required us to manually having to deduct the data from PDF-documents which was a time-exhaustive process. Given this small dataset, there exists a chance of overfitting the data in our models which would mean a worse prediction accuracy for ECUs that is not similar to the ones on which the models are based.

Looking at the training and validation split presented in section 5.2.2 we can deduct that the training and validation datasets have a similar median and that the RAM and ROM measures have around the same inter-quartile range. We can also see that all validation sets have one outlier that has a higher measure than the rest of the data, this outlier will be an indication whether the model can handle data that is not similar to the one that has been used to build the models. Moreover, we can further see a smaller inter-quartile range in the training set than the validation set for the CPU measures, that may indicate that the validation dataset may be harder to predict accurately for the CPU parameter.

Additionally, when looking at the details for the collected data, some ECUs that have similarly identified features may have different resource utilization at runtime. This variation is most likely due to aspects of the running software components that exist in the top layer of the different ECUs. As these top-level software components

are not taken into account in our prediction models, the effect of these on the resulting resource utilization is not addressed. However, as these software components require ECU functionality realized by BSW modules in order to perform their behavior, the software component impact on the resulting resource utilization should also be correlated to the high-level features realized by the BSW modules. Given this, the resulting accuracy of the created models may as such be influenced by details of the top-level software components, and the effect of this may as such have had an impact on the points in the validation data that were unable to be correctly predicted using the created models. However, since we believe that the hardware utilization of the top-level software components and the identified features are correlated, we believe that by collecting a larger dataset and potentially identifying even more high-level features, one can successfully reduce the impact of the top-level software components on the accuracy of the models. Moreover, since the data that the created prediction models are built upon all include these high-level software components, an ”extra” amount of resources is included in the predictions to accommodate for these.

Finally, we can see that generally all of the metrics increases as an ECU has more features identified. Looking at the CPU measures however, we can see a high CPU measure for ECUs that only have one identified feature. These high-computational units are the MOST-nodes that handle the infotainment systems and as such only utilizes the diagnostic functionality of AUTOSAR while still requiring large amounts of CPU power to be able to perform its features. As such, these nodes may threat the prediction models potential accuracy for the CPU parameter. As these nodes thus utilize very little of the standardized functionality of AUTOSAR, one could argue that these are not representational of AUTOSAR-compliant ECUs and should be removed from the dataset. However, as these do in fact use some standardized AUTOSAR functionality and given the small size of the dataset, we have decided on leaving them in the dataset.

6.2 Resulting Models

From the results presented in section 5.4 and 5.5 we can see that for the CPU metric, the accuracy increases as we add more layers with the most accurate configurations having 4-5 layers. This rather deep network required to accurately model the CPU parameter indicates that the relationship between the identified high-level features and the metric is complicated. The necessity for the deep network may further indicate that a larger, more diverse dataset should be used to be able to accurately predict accurate measures using a more shallow network. As such, there may exist a big risk of overfitting to our specific data given the depth of the most optimal network.

Moving on to the RAM and ROM predictions we can instead see that as more layers are added to the model, the accuracy becomes much worse for the RAM network and seems to be a local minimum for the ROM prediction around 3-4 layers. As the accuracy becomes worse as the network becomes deeper, we can conclude that the

deeper networks are most likely overfitting to the training data and as such do not give a good prediction accuracy on the validation data.

Looking at the most optimal RAM and ROM configurations presented in sections 5.5.3 and 5.5.4, we can see that for the RAM parameter, 2-3 layers are the most optimal, and for the ROM configurations, 1-2 is the most optimal. These rather shallow networks lead us to the conclusion that the network might have a better prediction accuracy should our dataset become more diverse, i.e. that the network configuration does not suffer from the same potential overfitting-threats as the CPU counterpart.

Referring to the number of epochs for the three models presented in figures 5.10 - 5.12, we can see that the epochs converge at around 80 for the CPU, 25 for the RAM and 20 for the ROM parameters. As such these three measures are a good number of epochs for our most optimal models. Looking at the CPU loss, however, we can see that the loss varies a lot over the number of epochs and takes a long time to converge. Similarly, the ROM loss also varies a lot around what seems to be the converged value. From this we can conclude that the CPU parameter is more difficult for the network to accurately predict, thus requiring more times to learn from the data in order to make accurate predictions whereas the ROM and RAM parameters require less data usages to learn the correlation between the features and the measure. Additionally, the converged value is not as stable for the ROM parameter as for the RAM parameter.

Looking at the prediction accuracy presented in figures 5.13 - 5.15 for the three models we can see that the points are very close to the line of best fit for the RAM and ROM parameter, but not for the CPU parameter. Further, we can identify two clear outliers; one for the RAM configuration (brown point), and one for the ROM configuration (orange point). These two outliers correspond to two different ECUs where we predict the former to require more RAM than necessary and predict the latter to require much less than necessary. Both of these outliers have been identified to utilize 7 of the high-level features, and thus we believe that the small amount of data gathered is the reason that these points are mispredicted, as there are very few other similar points in our dataset. Moreover, the CPU predictions are not as closely spread around the line of best fit, thus leading us to the conclusion that the CPU utilization at runtime cannot be estimated with the same accuracy as the RAM and ROM parameter. The biggest outlier for the CPU predictions is the red point which corresponds to a high-computational ECU that does not fulfill the criteria to be branded as a *Central*-node. As such, this data point is an outlier in comparison to the other similar node given its high CPU utilization at runtime. From this misprediction, we draw the conclusion that there may exist a need to identify another feature that can explain these kinds of nodes further.

Even though there exist some outliers in our predictions for resource utilization at runtime, we manage to propose a suitable hardware baseline for the three parameters for most of our validation set. As such our results indicate that we can predict the

CPU, RAM and ROM utilization for ECUs by identifying their high-level feature largely realized by modules from the AUTOSAR base-software layer. Moreover, the accuracy of said predictions lies between 60-78% for all the three metrics. This leads us to the conclusion that neural networks are a suitable candidate for predicting a suitable baseline for ECUs based on what high-level characteristics said ECU has.

6.2.1 Comparison to Alternatives

As described in section 5.6, an alternative have been created to further give an indication to the feasibility of using neural networks as the base for building prediction models for estimating hardware requirements for ECUs. Our findings indicate that using a multiple regression approach is not as suitable as the neural network approach for predicting the RAM metric, where the model only has an accuracy of 35%. However, the multiple regression approach reaches an accuracy of 71% for the ROM parameter and 60% accuracy for the CPU parameter and as such gives similar or the same result as our neural network model. Although the multiple regression model is able to accurately predict baselines for the ROM and CPU parameters, we conclude that is is not apt as a foundation for a prediction model in predicting ECU resources in full, as the accuracy for the other parameter is lower in comparison to the neural network approach. As such, in our problem domain, with our current dataset, neural networks are a better approach for predicting accurate hardware baselines than a multiple regression model.

6.3 Usability of the Created Models for Volvo Cars Corporation

As described in section 5.3, the selection of new hardware is often based on previous tacit knowledge or subcontractor knowledge, as such a tool or methodology for validation and selection of hardware may be usable for Volvo Cars Corporation to get more control over hardware characteristics of ECUs. We as such believe that the models created in this thesis could be of large value to Volvo Cars Corporation allowing them to get an early indication of what hardware requirements are suitable for a new ECU that has a set of predefined high-level features. As the models created in this thesis are essentially based on previous similar examples of ECUs, the resulting models are similar to the selection of new hardware approach identified in the interview with the system expert. As such, the created models reduces the need for tacit knowledge, allowing people without advanced system design knowledge to get a proposed suitable baseline for an ECU. Moreover, utilizing the models proposed in this thesis would allow ECU teams to select an initial ECU hardware baseline, thus reducing the need to select the ECU with the highest overhead as identified in the informal chat with the engineer described in section 5.3. Additionally, by being able to accurately propose a suitable baseline for a new ECU project, there may be no need for an unnecessarily high initial baseline for new ECU projects to assure enough computational resources. By potentially reducing the amount of such initial overhead the company may save money on a per-ECU basis, and also reducing the

amount of unnecessary hardware which is good from an environmental standpoint on large scale manufacturing.

However, it is worth mentioning that while our models are 60-78% accurate, these are based on a small amount of data and may as such not be able to be generalized. While we believe these models can serve as an extension to the already existing methodology, allowing for validation and selection of hardware, we also believe that these models should be extended with a bigger, more diverse dataset in order to be able to solely focus on a machine-learning-based approach for hardware selection and validation.

6.4 Threats to Validity

The validity of a study expresses the level of integrity the study has, i.e. how rigid the results of the study are, thus removing any potential bias the researcher may have. The validity threats to this study were investigated and identified in the first and second step of the methodology followed in this study (see section 4.3). There are commonly four types of validity threats a researcher needs to address in a study: conclusion, internal, external and construct (Feldt and Magazinius, 2010).

6.4.1 Threats to conclusion validity

Threats to conclusion validity are often related to whether or not we can effectively state that our approach to the study and the outcome is related (Wohlin et al., 2012). In our case, these threats relate to whether there is a correlation between the AUTOSAR characteristics an ECU has and the processor and memory load at runtime on said ECU. As the identified characteristics describe high-level features of ECUs, the addition of a characteristic would as such often introduce new running artifacts, thus increasing the number of tasks and the resulting resource utilization.

Given that we are able to predict hardware baselines for the CPU, RAM and ROM parameters that are 60-78% accurate we can conclude that there is an apparent correlation between the features and the metrics. Additionally, identifying more similar features to the ones identified in this theses would potentially make the predictions even more accurate and thus minimize the chance for conclusion threats even more. However, since the running top-level software components of an ECU can vary a lot in size and performance, for example by sending and receiving different numbers of signals, an ECU having similar features could vary a lot in resource utilization. In order to combat this, the idea was to collect many different points for similar feature sets such that the details of the top-level software could be discarded, but as the number of collected points is rather low this may threaten the accuracy of the models. However, as we have a large variation in the resource utilization for our collected data and we are still able to reach an accuracy of 60-78%, the characteristics and the resulting hardware requirements seem to be correlated, and thus, the varying size and requirements of different running software components do not seem to damage the correlation between the features and the resource utilization,

and as such these threats do not impact our findings in such a way that it would nullify our results.

6.4.2 Threats to internal validity

The internal validity is about threats in the relationship between the approach and the outcome such that no internal workings of the approach could have influenced the outcome, i.e. there is a causal relationship between the input and output. In our study, this is about the inner workings of our neural networks such that they do not show a correlation that does not exist (i.e. there is any clear correlation between AUTOSAR characteristics and the resource utilization), for example, if the neural networks should produce a prediction that is seemingly random or very inaccurate in the evaluation. In that case, we could have an internal problem with the network configuration. In order to combat this, several configurations have been tested for each network. Moreover, as several of the tested configurations have a similar prediction accuracy to the most optimal one for each parameter, and that every such configuration has been tested 5 times in order to compute the average accuracy, we have effectively minimized the risk for the threat. As such, we can state that there exist very few threats to the internal validity of this study.

6.4.3 Threats to external validity

Threats to the external validity of a study concern the ability to generalize the study, i.e. can the findings in the study be generalized to cover more than the scope of the study or are the findings extremely specific. In our study these kinds of threats revolves around whether the correlation found in the resource utilization is specific to the implementations used for the modules in these logical clusters or the data used to train the network or if the network configuration should suffer from overfitting such that predictions that have different feature sets than what is identified in our data would yield an inaccurate prediction. In order to combat this, the idea was to collect a diverse dataset covering many different subsets of features for different ECUs. However, as unfortunately, we were able to only collect a rather small set of points these threats are present in our results. Moreover, as we have only been working with Volvo-specific systems we cannot make any claims about other systems from other OEMs.

Even though these threats persist, we do believe that since our findings seem successful and that since similar functionality can be found in any AUTOSAR-compliant system we can generalize our findings to other problem domains. Mainly our positivity to these threats relies on the fact that all AUTOSAR systems share the same foundation, regardless of implementer or specifics, given the highly standardized nature of AUTOSAR. Moreover, we do believe that the features identified can be extended to cover other systems or even be made more detailed to yield even more accurate predictions. As such, we believe that the foundation of the thesis is still rigid and that we can generalize our findings to similar systems, even though the threats persist.

6.4.4 Threats to construct validity

The final type of validity threat is threats to construct validity. These threats concerns whether or not the experiment setting accurately reflect what is intended to be studied. In our study, these threats concern whether the data that our tool is based on is reflective of the real life equivalence. As the measurements are collected from actual ECUs, and not simulations or hypothetical calculations of loads we have based our models or realistic data and have thus minimized the risk for these kinds of threats.

7
Conclusion

This chapter aims to draw conclusions from the findings presented in the previous chapter. Moreover, the future work that this thesis enables is discussed.

7.1 Results of the study

This study set out to develop and analyze prediction models for estimating hardware requirements for ECUs in embedded AUTOSAR-compliant systems. In the approach of this, different AUTOSAR features that are present in Volvo Cars Corporation developed cars were identified via a literature study and an interview. The results of these activities yielded a logical clustering of a subset of the AUTOSAR BSW modules realizing high-level features that are commonly found in Volvo Cars Corporation ECUs. Additionally, two other features were identified that account for different types of ECUs; LIN-slaves that are essentially sensors and actuators, and central ECUs that facilitates communication to several other ECUs. These features were then used to categorize data that had been collected on CPU, ROM and RAM utilization for different ECUs. After this was completed, three neural networks were created to predict CPU, RAM and ROM requirements based on the features of an ECU.

Based on the results from the previous chapter we can conclude that the identified features can effectively be used to create neural network models that are able to predict resource utilization at runtime for single core 32-bit CPUs, the RAM, and the ROM parameters. We have further concluded that these predictions are more accurate for the RAM and ROM predictions whereas the predictions for CPU utilization are more inaccurate. Moreover, we have identified that in order to predict accurately for these three parameters, the CPU parameter generally requires a deeper neural network in terms of hidden layers and number of epochs than the RAM and ROM counterpart.

We have identified that predicting suitable hardware requirements for ECUs are possible with an accuracy of 60-78%. This prediction consists of estimating the resource utilization at runtime for the three parameters and rounding up to the nearest baseline of total resources available for an ECU, based on what hardware baselines have been found in Volvo Cars Corporation ECUs. We have thus reached prediction models capable of predicting hardware requirements for ECUs and can address the research questions.

For *RQ1.1* we can conclude that AUTOSAR standardized BSW modules are a suitable base for predicting hardware resource utilization by restructuring these into logical clusters that identify high-level features of ECUs. These features can then be used to characterize ECUs and predict resource utilization at runtime for the CPU, RAM and ROM metrics and further be used to propose a suitable hardware baseline.

For *RQ1.2* we conclude that the resource utilization at runtime can be accurately modeled for an ECU given the identified characteristics of said ECU, based on the features identified in this thesis. We further conclude that the CPU predictions are not as accurate as of the RAM and ROM predictions.

For *RQ1.2.1* we conclude that neural networks can accurately model the resource utilization and serve as a foundation for prediction models capable of predicting suitable hardware requirements. Moreover, the neural networks are more accurate in predicting resource utilization than the multiple regression counterpart.

For *RQ2.1* we conclude that the models created are in the majority of cases able to predict a hardware baseline that is at par or better than what has actually been selected for the ECU. These models are in 60-78% of cases able to predict such a hardware baseline for the validation dataset selected for this study.

7.2 Future Work

While we have managed to get successful results on our thesis our results are based on a rather small dataset. As such we believe that this thesis may serve as a foundation for extending the work to both include more features that can characterize ECUs in more detail as well as extending the foundation that the created models are built on, namely the dataset. As our dataset only comprises some 121 points, it is in the lower regions of where neural networks can be applicable, and as such to be able to get better results that can cover a wider array of different ECUs, the dataset should be extended and the models retrained.

As this thesis only focuses on predictions on single core 32-bit CPUs, the dataset could further be extended with data for other ECU architectures such that a wider array of ECUs can be predicted. Currently, LIN-slaves cannot often be predicted in terms of CPU given that these often run 8bit CPUs. As such, in order to cover other types of CPUs, additional models need to be created and trained to cover more than one architecture.

Further, the created models could be extended with additional parameters to more accurately describe features of ECUs, for example by including the number of signals the ECU sends and receives as this influences what level of computational resources that are used to handle the signaling in between ECUs. Additionally, the sizes of the different signals, as well as the number and size of variables for the top-level soft-

ware components or additional details from the ECUs, could be included to make the models even more detailed. However, one big advantage of the created models is that they can be used early in the development of ECUs to propose a suitable hardware baseline before all the parts of the ECU have been created, for example, the software components. As such, the addition of detailed data from the software components to the model may render the possibility of using the models to propose initial hardware requirements impossible, but instead, introduce more detailed ways of validating finished ECUs. There as such exist some variations to the created models that can be used, for example, the addition of more high-level features to the characterization to be able to more accurately propose an initial hardware baseline, or by introducing aspects from the top-level software to use the models as a more detailed validation technique.

Finally, we believe that this thesis proves that machine-learning approaches can be a suitable addition to the already existing methodology to create and validate embedded systems in the automotive domain and can us such pave the way for additional research in the area, bringing a data-based future for the creation and validation of embedded vehicle systems.

Bibliography

AUTOSAR consortium (2017a). *AUTOSAR Layered Software Architecture v.4.3.1*. URL: https://www.autosar.org/.

— (2017b). *History*. URL: https://www.autosar.org/about/history/.

Bo, H. et al. (2010). "Basic Concepts on AUTOSAR Development". In: *2010 International Conference on Intelligent Computation Technology and Automation*. Vol. 1, pp. 871–873. DOI: 10.1109/ICICTA.2010.571.

Čirkovs, Andrejs (2017). *The mostly complete chart of Neural Networks, explained*. URL: https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464.

Dresch, Aline, Daniel Pacheco Lacerda, and Antunes José Antônio Valle (2015). *Design science research: a method for science and technology advancement*. Springer.

Dreyfus, G. (2005). *Neural Networks: An Overview*. URL: https://doi.org/10.1007/3-540-28847-3_1.

Duan, R. et al. (2009). "A Hybrid Intelligent Method for Performance Modeling and Prediction of Workflow Activities in Grids". In: *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 339–347. DOI: 10.1109/CCGRID.2009.58.

Đurišic, Darko (2015). *Measuring the evolution of automotive software models and meta-models to support faster adoption of new architectural features*. Division of Software Engineering, Department of Computer Science & Engineering, University of Gothenburg.

Đurišić, Darko (2017). *Measuring the evolution of meta-models, models and design requirements to facilitate architectural updates in large software systems*. Division of Software Engineering, Department of Computer Science & Engineering, University of Gothenburg.

Durisic, D. et al. (2016). "Addressing the need for strict meta-modeling in practice - a case study of AUTOSAR". In: *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pp. 317–322.

Feldt, Robert and Ana Magazinius (2010). "Validity Threats in Empirical Software Engineering Research - An Initial Survey." In: pp. 374–379.

Fürst, S. and M. Bechter (2016). "AUTOSAR for Connected and Autonomous Vehicles: The AUTOSAR Adaptive Platform". In: *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*, pp. 215–217. DOI: 10.1109/DSN-W.2016.24.

Ganapathi, A. et al. (2009). "Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning". In: *2009 IEEE 25th International Conference on Data Engineering*, pp. 592–603. DOI: 10.1109/ICDE.2009.130.

Johannesson, Paul (2014). *Introduction to design science*. Springer.

K., M. and R. Kumar (2010). "Spam Mail Classification Using Combined Approach of Bayesian and Neural Network". In: *2010 International Conference on Computational Intelligence and Communication Networks*, pp. 145–149. DOI: 10.1109/CICN.2010.39.

Layered Software Architecture. URL: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf.

Martin Abadi et al. (2015a). *Module: tf.keras.activations*. Software available from tensorflow.org. URL: https://www.tensorflow.org/api_docs/python/tf/keras/activations.

— (2015b). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

Mehlig, B. (2019). *Artificial Neural Networks*. URL: https://arxiv.org/abs/1901.05639.

Radonja, P. (2012). "Comparison of generalized profile function models based on linear regression and neural networks". In: *11th Symposium on Neural Network Applications in Electrical Engineering*, pp. 41–46. DOI: 10.1109/NEUREL.2012.6419959.

Rana, Rakesh et al. (2014a). "The Adoption of Machine Learning Techniques for Software Defect Prediction: An Initial Industrial Validation". In: *Knowledge-Based Software Engineering*. Ed. by Alla Kravets et al. Cham: Springer International Publishing, pp. 270–285. ISBN: 978-3-319-11854-3.

Rana, R. et al. (2014b). "A framework for adoption of machine learning in industry for software defect prediction". In: *2014 9th International Conference on Software Engineering and Applications (ICSOFT-EA)*, pp. 383–392.

Raschka, Sebastian (2015). *Single-Layer Neural Networks and Gradient Descent*. URL: https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html#frank-rosenblatts-perceptron.

Tang, Huajin, Kay Chen. Tan, and Zhang Yi (2007). *Neural Networks: Computational Models and Applications*. Springer-Verlag Berlin Heidelberg.

Veen, Fjodor van (2017). *Neural network zoo prequel: cells and layers*. URL: `https://www.asimovinstitute.org/author/fjodorvanveen/`.

Wieringa, Roel J. (2014). *Design science methodology for information systems and software engineering*. Springer.

Wohlin, Claes et al. (2012). *Experimentation in software engineering*. Springer, pp. 102–104.

A

Appendix A - Interview Template for deducting Volvo Cars contextual knowledge and logical clustering

Interview template for deducting the soundness of a proposed logical grouping of the AUTOSAR BSW modules and get contextual knowledge of module usage at Volvo Cars Corporation

Interview about deducting AUTOSAR modules on ECUs

Length: 45-60 minutes

Interviewee:

AUTOSAR Employed / **Volvo Employee** / Both

Introduction

This semi-structured interview is about gathering data on the different BSW modules that exist in the AUTOSAR classic standard, and how the dependencies between these are structured. Further, the specifics of which that are prevalent in Volvo Cars Corporation will be discussed.

The results of this interview will be used to construct a logical grouping of the BSW modules in order to be able to better characterize ECUs from their features. This will, in turn, be used to generate a prediction model that can predict ECU hardware requirements based on said characteristics. Participation in this interview will be completely anonymous.

Verbal Consent

Would you like to participate in this interview? And are I allowed to record your answers in order to derive results that I might have missed to document?

- **Verbal Consent WAS obtained from the study participant**
- Verbal consent was NOT obtained from the study participant

Background Information

What are your background in relation to AUTOSAR?
The interviewee is an employee at Volvo Cars Corporation that specifically works with AUTOSAR and is a stakeholder in the development in the standard

AUTOSAR usage at Volvo Cars Corporation.

These questions aim to get the contextual information required on AUTOSAR usage at Volvo Cars.

What versions of the AUTOSAR Classic standard is mostly used at Volvo Cars Corporation?

The baseline AUTOSAR version at Volvo Cars Corporation is 4.0.3. The idea is to update this to 4.3 for all ECU teams. However, most of the ECU teams still run 4.0.3, where most teams that run 4.3 are ethernet based nodes that require time-synchronization.

Moreover, some teams may run a version of AUTOSAR that is between 4.0.3 and 4.3, but no one uses a version lower than 4.0.3. Thus this version becomes a good foundation to base the logical clusters on.

(Referring to Figure A.1 & Figure A.2) Are there some modules/ functional groups that are standardized in AUTOSAR that are not used at Volvo Cars Corporation?

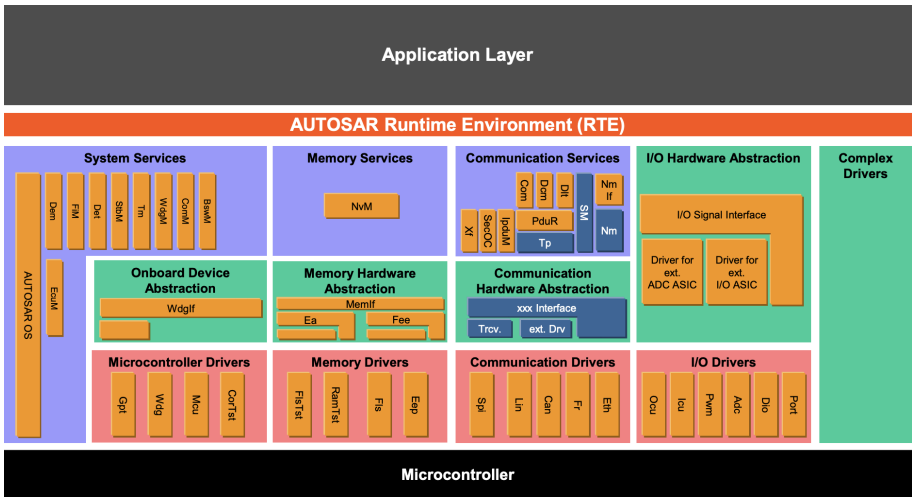


Figure A.1: Subset of AUTOSAR Modules mapped to the different AUTOSAR layers. Derived from (https://www.autosar.org)

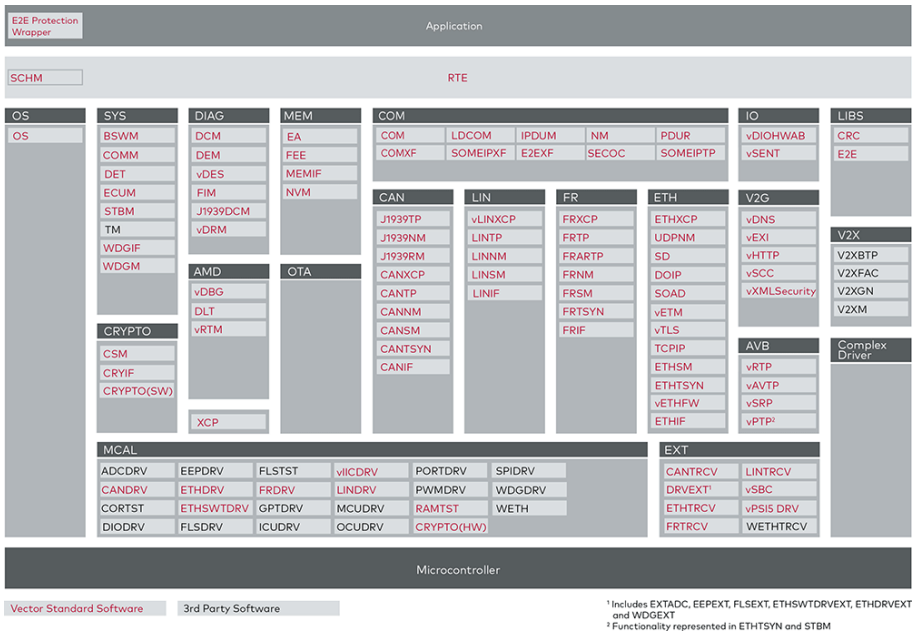


Figure A.2: AUTOSAR Modules in their respective packages in the Vector Microsar implementation of the AUTOSAR Classic BSW stack. Derived from (https://www.vector.com/int/en/products/products-a-z/embedded-components/microsar/)

At Volvo Cars Corporation, the following modules/ functional groupings are not used:

- **J1393:** Variation of CAN communication, more for trucks and thus not used at VCC
- **V2X:** Used for communication between vehicles and external entities, not used Currently at VCC, will however potentially be used in future cars
- **V2G:** Used for battery charging and communication with internet to be able to measure the cost for charing, generally not used at VCC, except perhaps in battery management nodes.
- **CANTT:** Not used at VCC

Using AUTOSAR BSW modules as a predictive base

This part of the interview aims to get knowledge about how one can group the BSW modules into more logical clusters.

Would you say that the way AUTOSAR structures the different BSW modules in functional groups. Or the way, for example, vector structures the modules in "packages" are a suitable base for explaining characteristics of an ECU. I.e. can the different packages be isolated on their own in different ECUs. For example, are the modules in the Communication Services (image A.1) or COM (image A.2) enough to introduce communication on an ECU?

Usually, an ECU behavior requires modules from all the different abstraction levels in order to function. For example, communication requires service modules from the service layer and drivers from the ECU abstraction level. As such, the functional grouping of AUTOSAR is not a valid foundation for characterizing ECUs, as, for example, all nodes that use communication will use very similar functional groupings.

From the AUTOSAR documentation, I have created the following logical grouping. Would you consider this a more suitable base for deducting what AUTOSAR modules are present on an ECU, and does it cover the required modules for the specific behaviors? (Figure 5.1)

Generally he believes that this is a very good way of conducting the deductive characterizing of ECUs, however, some parts of my proposed diagram should be changed. These follow:

Large COM:

1. **DLT** is for development testing, should not be in production vehicles unless the developers have forgotten to remove it/turn it off. Remove from model.
2. **LDCOM** is not used for all large communication ECUS (CAN, FlexRay, Ethernet), very low frequency. Remove from model.
3. **SPIDRV** is only used if the ECU communicates with some external entity, like a external flash-memory. Should be made optional.
4. **SECOC** Remove.
5. **IPDUM** Not Common, Remove.

J1939: Remove

TTCAN: Remove

ETH *All the modules that exist here in the proposed base are used in Ethernet ECUs. But as VCC rarely uses Ethernet nodes, all of the Ethernet-related modules can be included in this package, as there will be very little variations in these nodes.*

CRYPTO: *Optional behavior, all CRYPTO-modules can be included here.*

IO *Remove*

Lin *There exist specific lin nodes that only functions as a sensor or actuator and contain very little, if any AUTOSAR features, thus an additional characteristic should be introduced for these. These are generally identified by a very low amount of RAM and ROM.*

Diagnostics *All AUTOSAR nodes at VCC uses DEM and DCM. Move these into their own package in Mandatory.*

1. **DI** is not used in production, only during development. Remove.
2. **FIM** Move to an optional package.

MEMORY *All AUTOSAR nodes run NVM as they collect diagnostics, and as such run FEE or EA, most should run FEE. But as these usually have about the same performance, they will not be a valid foundation for characteristics. Thus all the memory modules can be grouped together and moved to the mandatory layer.*

XCP *Move to optional, add ETHXCP*

The other modules from Figure A.3 not covered in the proposed base, what should be added?

1. The **AVB** cluster is an Ethernet protocol to send time-synchronized audio. Potentially Infotainment uses these. Use as a separate module in the ECU Specific layer.
2. The **AMD** cluster contains some diagnostics and vector-specific modules, these are not a viable foundation as these are potentially not used at VCC.

What kinds of nodes exist in Volvo produced Vehicles?

There exist 6 base types of modules:

1. **CAN:** CAN-based communication. Will have all Standard modules
2. **Flexray:** FlexRay-based communication. Will have all Standard modules.
3. **Ethernet:** Ethernet-based communication. Will have all Standard modules.
4. **Lin:** Lin based Communication. Will have all Standard modules.
5. **Lin-Slaves:** Simple nodes that are essentially actuators or sensors. Will not have the standard modules
6. **MOST-Nodes:** Nodes that are connected to other nodes via a MOST-connection, an optical sound connection (mostly the infotainment nodes that handles sound). Will only have the DIAG_BASE standard module

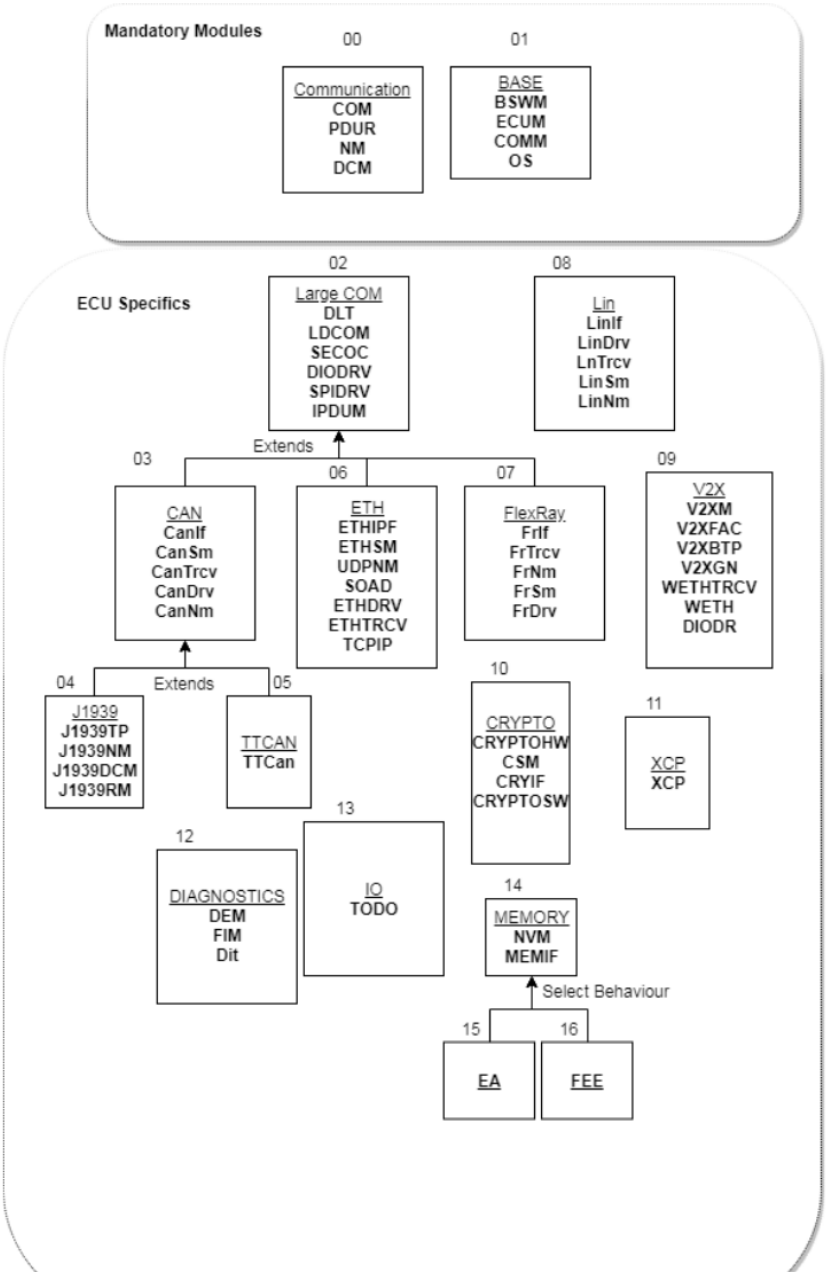


Figure A.3: Proposed grouping of AUTOSAR modules in logical clusters.

B

Appendix B - The Tested Configurations for the networks

These are all the configurations tested for the three types of networks, sorted on prediction accuracy.

Tested configurations for the CPU parameter

Layer Count	Layer Structure	Accuracy (%)	MAE	MSE
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](linear, 256)(relu, 1)	60	47.6153350830078	5705.1775390625
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](softplus, 256)(relu, 1)	58.1818181818182	48.1369506835938	5869.5783203125
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](linear, 256)(relu, 1)	56.3636363636364	49.5633094787598	6131.509765625
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](linear, 32)(relu, 1)	56.3636363636364	47.7620445251465	6179.938671875
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](linear, 64)(relu, 1)	56.3636363636364	47.2441703796387	6271.75693359375
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](elu, 256)(relu, 1)	56.3636363636364	47.4862548828125	6050.47529296875
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](elu, 16)(relu, 1)	54.5454545454545	46.9350631713867	6769.525
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](linear, 16)(relu, 1)	54.5454545454545	48.4045806884766	6683.91572265625
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](relu, 16)(relu, 1)	54.5454545454545	51.313240814209	6975.87998046875
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](selu, 16)(relu, 1)	54.5454545454545	52.4974914550781	7189.496484375
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](softplus, 16)(relu, 1)	54.5454545454545	50.5003921508789	6957.28935546875
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](relu, 32)(relu, 1)	54.5454545454545	51.5259315490723	7108.773828125
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](elu, 64)(relu, 1)	54.5454545454545	50.0399032592773	6782.6306640625
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](sigmoid, 64)(relu, 1)	54.5454545454545	53.214762878418	7440.62080078125
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](linear, 256)(relu, 1)	54.5454545454545	51.4243766784668	6657.42646484375
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](elu, 16)(relu, 1)	54.5454545454545	50.2120666503906	6964.9947265625
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](linear, 16)(relu, 1)	54.5454545454545	49.9534339904785	6984.11591796875
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](selu, 16)(relu, 1)	54.5454545454545	50.0738265991211	6993.7662109375
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](softplus, 16)(relu, 1)	54.5454545454545	50.1611770629883	6927.0142578125
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](elu, 32)(relu, 1)	54.5454545454545	50.1177040100098	6920.79052734375
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](linear, 32)(relu, 1)	54.5454545454545	49.9422424316406	6868.005078125
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](relu, 32)(relu, 1)	54.5454545454545	49.9924346923828	6885.99091796875
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](selu, 32)(relu, 1)	54.5454545454545	50.5000617980957	7031.49794921875
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](elu, 64)(relu, 1)	54.5454545454545	48.858780670166	6714.57890625
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](linear, 64)(relu, 1)	54.5454545454545	48.3829162597656	6548.660546875
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](relu, 64)(relu, 1)	54.5454545454545	48.9275619506836	6731.95732421875
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](selu, 64)(relu, 1)	54.5454545454545	49.257300567627	6762.66611328125
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](softplus, 64)(relu, 1)	54.5454545454545	48.703466796875	6639.76494140625
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](tanh, 64)(relu, 1)	54.5454545454545	51.7739814758301	7483.78603515625
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](elu, 256)(relu, 1)	54.5454545454545	49.3411804199219	6287.9337890625
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](relu, 256)(relu, 1)	54.5454545454545	49.2036880493164	6262.8666015625
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](selu, 256)(relu, 1)	54.5454545454545	49.0780853271484	6307.17333984375
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](softplus, 256)(relu, 1)	54.5454545454545	49.5082496643066	6263.17412109375
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](tanh, 256)(relu, 1)	54.5454545454545	50.4960006713867	7047.12890625
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](elu, 16)(relu, 1)	54.5454545454545	50.3416542053223	6769.57451171875
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](relu, 16)(relu, 1)	54.5454545454545	49.7905799865723	6785.66396484375
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](selu, 16)(relu, 1)	54.5454545454545	49.4958229064941	6628.70751953125
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](softplus, 16)(relu, 1)	54.5454545454545	47.6070571899414	6283.55087890625
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](elu, 32)(relu, 1)	54.5454545454545	47.3381675720215	6322.24609375
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](relu, 32)(relu, 1)	54.5454545454545	47.8931335449219	6533.80224609375
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](selu, 32)(relu, 1)	54.5454545454545	47.7562805175781	6356.64501953125
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](softplus, 32)(relu, 1)	54.5454545454545	46.927808380127	6233.92880859375
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](relu, 64)(relu, 1)	54.5454545454545	47.3103126525879	6256.5427734375
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](selu, 64)(relu, 1)	54.5454545454545	47.4560188293457	6302.0916015625
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](sigmoid, 64)(relu, 1)	54.5454545454545	51.5648445129395	7343.0323242187 5
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](softplus, 64)(relu, 1)	54.5454545454545	47.3471664428711	6453.06923828125
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](tanh, 64)(relu, 1)	54.5454545454545	51.0843338012695	7152.676171875

Table B.1: Tested configurations for the CPU parameter

Layer Count	Layer Structure	Accuracy (%)	MAE	MSE
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](tanh, 256)(relu, 1)	54.5454545454545	47.7026870727539	6566.30771484375
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](linear, 32)(relu, 1)	52.7272727272727	51.773078918457	7142.446484375
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](relu, 256)(relu, 1)	52.7272727272727	50.4784614562988	6775.41533203125
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](tanh, 16)(relu, 1)	52.7272727272727	66.1424652099609	11045.7387695312
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](tanh, 32)(relu, 1)	52.7272727272727	57.1553520202637	8639.17646484375
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](tanh, 32)(relu, 1)	52.7272727272727	55.5852996826172	8418.38544921875
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](tanh, 16)(relu, 1)	50.9090909090909	67.6750648498535	11033.1737304688
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](sigmoid, 16)(relu, 1)	50.9090909090909	66.1284362792969	11181.86015625
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](sigmoid, 16)(relu, 1)	50.9090909090909	67.6417739868164	11682.5119140625
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](selu, 32)(relu, 1)	49.0909090909091	52.8402694702148	7228.46259765625
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](relu, 64)(relu, 1)	49.0909090909091	51.8483383178711	7233.0896484375
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](selu, 64)(relu, 1)	49.0909090909091	51.0544708251953	7068.6228515625
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](softplus, 64)(relu, 1)	49.0909090909091	51.6324310302734	7122.09658203125
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](elu, 256)(relu, 1)	49.0909090909091	52.0633460998535	7081.89541015625
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](selu, 256)(relu, 1)	49.0909090909091	53.3679962158203	7203.96943359375
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](elu, 32)(relu, 1)	47.2727272727273	53.2965675354004	7271.12509765625
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](linear, 64)(relu, 1)	47.2727272727273	51.5671768188477	7055.180078125
1	['Input'](elu, 16)(relu, 1)	45.4545454545455	68.692350769043	8989.88173828125
1	['Input'](selu, 16)(relu, 1)	45.4545454545455	118.091697692871	28066.986328125
1	['Input'](softmax, 16)(relu, 1)	45.4545454545455	99.6991012573242	22249.821875
1	['Input'](softplus, 16)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
1	['Input'](tanh, 16)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
1	['Input'](elu, 32)(relu, 1)	45.4545454545455	63.6468437194824	8253.68291015625
1	['Input'](linear, 32)(relu, 1)	45.4545454545455	65.0038864135742	7615.81376953125
1	['Input'](softmax, 32)(relu, 1)	45.4545454545455	99.7254776000977	22261.7296875
1	['Input'](tanh, 32)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
1	['Input'](elu, 64)(relu, 1)	45.4545454545455	61.8823463439941	8097.8013671875
1	['Input'](linear, 64)(relu, 1)	45.4545454545455	65.656005859375	7834.2998046875
1	['Input'](selu, 64)(relu, 1)	45.4545454545455	64.3919235229492	7963.3029296875
1	['Input'](softmax, 64)(relu, 1)	45.4545454545455	99.7062545776367	22253.206640625
1	['Input'](softplus, 64)(relu, 1)	45.4545454545455	60.94365234375	8168.1689453125
1	['Input'](tanh, 64)(relu, 1)	45.4545454545455	65.0041412353516	10018.1705078125
1	['Input'](elu, 256)(relu, 1)	45.4545454545455	59.8535720825195	8201.828515625
1	['Input'](linear, 256)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
1	['Input'](relu, 256)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
1	['Input'](selu, 256)(relu, 1)	45.4545454545455	60.8849967956543	8387.1849609375
1	['Input'](sigmoid, 256)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
1	['Input'](softmax, 256)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
1	['Input'](softplus, 256)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
1	['Input'](tanh, 256)(relu, 1)	45.4545454545455	64.1261779785156	9606.1455078125
2	['Input', '(sigmoid, 16)'](elu, 16)(relu, 1)	45.4545454545455	61.4627830505371	9379.0060546875
2	['Input', '(sigmoid, 16)'](linear, 16)(relu, 1)	45.4545454545455	61.1204368591309	9359.8359375
2	['Input', '(sigmoid, 16)'](relu, 16)(relu, 1)	45.4545454545455	60.7793907165527	9289.3689453125
2	['Input', '(sigmoid, 16)'](selu, 16)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
2	['Input', '(sigmoid, 16)'](sigmoid, 16)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
2	['Input', '(sigmoid, 16)'](softmax, 16)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
2	['Input', '(sigmoid, 16)'](softplus, 16)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
2	['Input', '(sigmoid, 16)'](elu, 32)(relu, 1)	45.4545454545455	61.2983039855957	9509.7755859375
2	['Input', '(sigmoid, 16)'](linear, 32)(relu, 1)	45.4545454545455	60.9376319885254	9371.6208984375
2	['Input', '(sigmoid, 16)'](relu, 32)(relu, 1)	45.4545454545455	61.3888786315918	9510.8990234375
2	['Input', '(sigmoid, 16)'](selu, 32)(relu, 1)	45.4545454545455	62.8201583862305	9941.0736328125
2	['Input', '(sigmoid, 16)'](sigmoid, 32)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
2	['Input', '(sigmoid, 16)'](softmax, 32)(relu, 1)	45.4545454545455	99.8348022460937	22309.64921875
2	['Input', '(sigmoid, 16)'](softplus, 32)(relu, 1)	45.4545454545455	61.6677505493164	9665.3638671875
2	['Input', '(sigmoid, 16)'](elu, 64)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
2	['Input', '(sigmoid, 16)'](linear, 64)(relu, 1)	45.4545454545455	60.9977897644043	9544.5123046875
2	['Input', '(sigmoid, 16)'](relu, 64)(relu, 1)	45.4545454545455	61.0348808288574	9539.6939453125
2	['Input', '(sigmoid, 16)'](selu, 64)(relu, 1)	45.4545454545455	61.785962677002	9737.6466796875
2	['Input', '(sigmoid, 16)'](sigmoid, 64)(relu, 1)	45.4545454545455	69.819221496582	12592.291015625
2	['Input', '(sigmoid, 16)'](softmax, 64)(relu, 1)	45.4545454545455	99.8644546508789	22322.786328125
2	['Input', '(sigmoid, 16)'](softplus, 64)(relu, 1)	45.4545454545455	55.3957077026367	7865.4025390625

Table B.2: (Cont): Tested configurations for the CPU parameter

Layer Count	Layer Structure	Accuracy (%)	MAE	MSE
2	['Input', '(sigmoid, 16)'](tanh, 64)(relu, 1)	45.4545454545455	66.9102020263672	11442.2990234375
2	['Input', '(sigmoid, 16)'](elu, 256)(relu, 1)	45.4545454545455	55.6921051025391	7877.55966796875
2	['Input', '(sigmoid, 16)'](linear, 256)(relu, 1)	45.4545454545455	54.6916702270508	7426.15556640625
2	['Input', '(sigmoid, 16)'](relu, 256)(relu, 1)	45.4545454545455	54.2372032165527	7546.55361328125
2	['Input', '(sigmoid, 16)'](selu, 256)(relu, 1)	45.4545454545455	57.2469772338867	8307.541796875
2	['Input', '(sigmoid, 16)'](sigmoid, 256)(relu, 1)	45.4545454545455	57.8768356323242	8334.67734375
2	['Input', '(sigmoid, 16)'](softmax, 256)(relu, 1)	45.4545454545455	99.7995697021484	22294.055078125
2	['Input', '(sigmoid, 16)'](softplus, 256)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
2	['Input', '(sigmoid, 16)'](tanh, 256)(relu, 1)	45.4545454545455	60.0825012207031	9218.45732421875
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](sigmoid, 16)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](softmax, 16)(relu, 1)	45.4545454545455	99.7554580688477	22274.5578125
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](softmax, 32)(relu, 1)	45.4545454545455	99.7929595947266	22291.13515625
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](softplus, 32)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](tanh, 32)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](softmax, 64)(relu, 1)	45.4545454545455	99.7921661376953	22290.78203125
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](tanh, 64)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](sigmoid, 256)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](softmax, 256)(relu, 1)	45.4545454545455	99.835221862793	22309.8359375
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](softplus, 256)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](tanh, 256)(relu, 1)	45.4545454545455	56.0274269104004	7622.29111328125
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](relu, 16)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](softmax, 16)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](sigmoid, 32)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](softmax, 32)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](softplus, 32)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](sigmoid, 64)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](softmax, 64)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](sigmoid, 256)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
4	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)'](softmax, 256)(relu, 1)	45.4545454545455	99.3675567626953	22320.770703125
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](linear, 16)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](softmax, 16)(relu, 1)	45.4545454545455	99.7727630615234	22282.20078125
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](sigmoid, 32)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](softmax, 32)(relu, 1)	45.4545454545455	99.0850509643555	22285.376953125
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](elu, 64)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](softmax, 64)(relu, 1)	45.4545454545455	99.8541763305664	22318.228125
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](relu, 256)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](selu, 256)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](sigmoid, 256)(relu, 1)	45.4545454545455	118.155815124512	28067.361328125
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](softmax, 256)(relu, 1)	45.4545454545455	99.9032257080078	22339.9828125
1	['Input'](linear, 16)(relu, 1)	43.6363636363636	67.6072174072266	8324.77578125
1	['Input'](relu, 32)(relu, 1)	43.6363636363636	65.5690643310547	8511.271875
1	['Input'](selu, 32)(relu, 1)	43.6363636363636	66.1048217773437	8163.50068359375
1	['Input'](sigmoid, 32)(relu, 1)	43.6363636363636	73.5528533935547	11209.2404296875
1	['Input'](softplus, 32)(relu, 1)	43.6363636363636	61.6367630004883	8134.6615234375
1	['Input'](relu, 64)(relu, 1)	43.6363636363636	60.095849609375	7903.766015625
1	['Input'](sigmoid, 64)(relu, 1)	43.6363636363636	70.7892196655273	10867.1677734375
2	['Input', '(sigmoid, 16)'](tanh, 16)(relu, 1)	43.6363636363636	74.0744537353516	13111.2841796875
2	['Input', '(sigmoid, 16)'](tanh, 32)(relu, 1)	43.6363636363636	71.3382934570313	13064.139453125
3	['Input', '(sigmoid, 16)', '(tanh, 16)'](sigmoid, 32)(relu, 1)	43.6363636363636	60.3389350891113	8990.78125
5	['Input', '(sigmoid, 16)', '(tanh, 16)', '(sigmoid, 32)', '(relu, 16)'](tanh, 16)(relu, 1)	43.6363636363636	68.0092895507812	10840.4038085938
1	['Input'](relu, 16)(relu, 1)	41.8181818181818	68.9990859985352	9012.05302734375
1	['Input'](sigmoid, 16)(relu, 1)	40	78.7878005981445	12488.4560546875

Table B.3: Cont: Tested configurations for the CPU parameter

Tested configurations for the RAM parameter

Layer Count	Layer Structure	Accuracy (%)	MAE	MSE
2	['Input', '(relu, 16)'](linear, 256)(relu, 1)	68.5714285714286	22.4371849060059	1010.50150146484
2	['Input', '(relu, 16)'](softplus, 256)(relu, 1)	67.1428571428571	20.0688812255859	867.478796386719
3	['Input', '(relu, 16)', '(relu, 16)'](relu, 32)(relu, 1)	67.1428571428571	18.9525409698486	704.193786621094
2	['Input', '(relu, 16)'](relu, 256)(relu, 1)	65.7142857142857	20.1788879394531	831.291772460938
3	['Input', '(relu, 16)', '(relu, 16)'](relu, 64)(relu, 1)	65.7142857142857	23.1338119506836	1443.92736816406
3	['Input', '(relu, 16)', '(relu, 16)'](linear, 256)(relu, 1)	65.7142857142857	18.8083831787109	885.855688476562
1	['Input'](relu, 32)(relu, 1)	64.2857142857143	26.6040645599365	1530.29951171875
1	['Input'](selu, 32)(relu, 1)	64.2857142857143	27.4051498413086	1727.62426757812
1	['Input'](linear, 64)(relu, 1)	64.2857142857143	27.3481338500977	1683.1966796875
1	['Input'](relu, 64)(relu, 1)	64.2857142857143	26.487618637085	1417.24685058594
1	['Input'](softplus, 256)(relu, 1)	64.2857142857143	25.8792678833008	1268.4873046875
2	['Input', '(relu, 16)'](relu, 64)(relu, 1)	64.2857142857143	21.6676734924316	898.00615234375
3	['Input', '(relu, 16)', '(relu, 16)'](relu, 16)(relu, 1)	64.2857142857143	20.3586711883545	778.516613769531
1	['Input'](selu, 64)(relu, 1)	62.8571428571429	27.8353439331055	1618.63413085938
1	['Input'](elu, 16)(relu, 1)	62.8571428571429	27.7223510742188	1730.930859375
1	['Input'](elu, 64)(relu, 1)	62.8571428571429	27.3215950012207	1561.07924804687
3	['Input', '(relu, 16)', '(relu, 16)'](softplus, 16)(relu, 1)	62.8571428571429	20.902974319458	825.46572265625
2	['Input', '(relu, 16)'](softplus, 64)(relu, 1)	61.4285714285714	20.286088180542	777.506384277344
1	['Input'](tanh, 256)(relu, 1)	61.4285714285714	21.2893444061279	945.316577148438
3	['Input', '(relu, 16)', '(relu, 16)'](selu, 32)(relu, 1)	61.4285714285714	18.678303527832	725.8224609375
3	['Input', '(relu, 16)', '(relu, 16)'](elu, 64)(relu, 1)	61.4285714285714	20.2400131225586	1227.9341796875
3	['Input', '(relu, 16)', '(relu, 16)'](softplus, 64)(relu, 1)	61.4285714285714	21.4167694091797	1531.32241210937
2	['Input', '(relu, 16)'](linear, 32)(relu, 1)	60	22.5148155212402	926.273474121094
2	['Input', '(relu, 16)'](selu, 256)(relu, 1)	60	20.5771636962891	798.08291015625
2	['Input', '(relu, 16)'](sigmoid, 256)(relu, 1)	60	19.3456161499023	951.468859863281
3	['Input', '(relu, 16)', '(relu, 16)'](elu, 32)(relu, 1)	60	19.4380199432373	879.955261230469
3	['Input', '(relu, 16)', '(relu, 16)'](linear, 64)(relu, 1)	60	24.824955368042	1568.75463867188
2	['Input', '(relu, 16)'](softplus, 16)(relu, 1)	58.5714285714286	21.5783130645752	881.956848144531
2	['Input', '(relu, 16)'](selu, 32)(relu, 1)	58.5714285714286	21.6203590393066	897.072399902344
3	['Input', '(relu, 16)', '(relu, 16)'](linear, 16)(relu, 1)	58.5714285714286	22.2053607940674	837.504284667969
3	['Input', '(relu, 16)', '(relu, 16)'](tanh, 256)(relu, 1)	58.5714285714286	24.6351409912109	1270.30791015625
1	['Input'](relu, 256)(relu, 1)	58.5714285714286	25.3152851104736	1156.33582763672
2	['Input', '(relu, 16)'](softplus, 32)(relu, 1)	58.5714285714286	20.6050605773926	843.734765625
2	['Input', '(relu, 16)'](elu, 256)(relu, 1)	58.5714285714286	21.849772644043	969.058801269531
2	['Input', '(relu, 16)'](tanh, 256)(relu, 1)	57.1428571428571	18.8986194610596	773.790002441406
3	['Input', '(relu, 16)', '(relu, 16)'](softplus, 256)(relu, 1)	57.1428571428571	22.3238636016846	1484.20043945312
1	['Input'](linear, 16)(relu, 1)	57.1428571428571	27.022257232666	1664.46726074219
1	['Input'](selu, 16)(relu, 1)	57.1428571428571	27.4562992095947	1714.6224609375
1	['Input'](elu, 256)(relu, 1)	57.1428571428571	26.1353446960449	1350.75734863281
1	['Input'](linear, 256)(relu, 1)	57.1428571428571	27.9000370025635	1596.53933105469
1	['Input'](selu, 256)(relu, 1)	57.1428571428571	26.1895881652832	1294.92717285156
2	['Input', '(relu, 16)'](elu, 16)(relu, 1)	57.1428571428571	23.6859233856201	1002.41156005859
2	['Input', '(relu, 16)'](linear, 16)(relu, 1)	57.1428571428571	22.952513885498	962.734631347656
2	['Input', '(relu, 16)'](elu, 32)(relu, 1)	57.1428571428571	21.888060760498	897.553076171875
2	['Input', '(relu, 16)'](linear, 64)(relu, 1)	57.1428571428571	21.5563823699951	836.737060546875
2	['Input', '(relu, 16)'](tanh, 64)(relu, 1)	55.7142857142857	12.7254266738892	363.028314208984
3	['Input', '(relu, 16)', '(relu, 16)'](relu, 256)(relu, 1)	55.7142857142857	22.1954128265381	1095.26579589844
3	['Input', '(relu, 16)', '(relu, 16)'](selu, 256)(relu, 1)	55.7142857142857	23.6795162200928	1585.57282714844
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](selu, 16)(relu, 1)	55.7142857142857	41.0246658325195	3332.612890625
1	['Input'](tanh, 32)(relu, 1)	54.2857142857143	18.1349153518677	742.602697753906
2	['Input', '(relu, 16)'](selu, 16)(relu, 1)	54.2857142857143	27.498454284668	1292.32583007812
3	['Input', '(relu, 16)', '(relu, 16)'](sigmoid, 256)(relu, 1)	54.2857142857143	19.2536277770996	900.440026855469
2	['Input', '(relu, 16)'](tanh, 16)(relu, 1)	52.8571428571429	16.270408821106	582.041546630859
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](relu, 32)(relu, 1)	52.8571428571429	25.3293952941895	1245.94764404297
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](selu, 32)(relu, 1)	52.8571428571429	26.8932655334473	1434.13366699219
2	['Input', '(relu, 16)'](sigmoid, 64)(relu, 1)	50	17.1350021362305	689.900836181641
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](relu, 16)(relu, 1)	50	44.3661483764648	3447.74116210938
1	['Input'](tanh, 16)(relu, 1)	48.5714285714286	16.4190021514893	622.259893798828
1	['Input'](softplus, 32)(relu, 1)	48.5714285714286	26.4976051330566	1562.33000488281
2	['Input', '(relu, 16)'](sigmoid, 32)(relu, 1)	48.5714285714286	14.0964958190918	460.852703857422
3	['Input', '(relu, 16)', '(relu, 16)'](tanh, 64)(relu, 1)	47.1428571428571	28.8277202606201	1850.67819824219
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](elu, 32)(relu, 1)	47.1428571428571	39.910404586792	3149.11687011719
1	['Input'](sigmoid, 256)(relu, 1)	45.7142857142857	24.0732566833496	1280.48531494141

Table B.4: Tested configurations for the RAM parameter

Layer Count	Layer Structure	Accuracy (%)	MAE	MSE
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](elu, 16)(relu, 1)	45.7142857142857	53.5837738037109	6429.20029296875
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](linear, 16)(relu, 1)	45.7142857142857	49.9926872253418	3803.16240234375
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](linear, 32)(relu, 1)	44.2857142857143	24.6469581604004	1139.41812744141
3	['Input', '(relu, 16)', '(relu, 16)'](tanh, 16)(relu, 1)	42.8571428571429	21.0366840362549	846.323266601563
1	['Input'](sigmoid, 32)(relu, 1)	41.4285714285714	23.9367340087891	1279.10812988281
3	['Input', '(relu, 16)', '(relu, 16)'](sigmoid, 16)(relu, 1)	41.4285714285714	20.1763469696045	880.313433837891
3	['Input', '(relu, 16)', '(relu, 16)'](sigmoid, 64)(relu, 1)	40	23.058081817627	1721.51470947266
2	['Input', '(relu, 16)'](softmax, 256)(relu, 1)	37.1428571428571	41.0675048828125	4323.875390625
3	['Input', '(relu, 16)', '(relu, 16)'](sigmoid, 32)(relu, 1)	37.1428571428571	16.6300270080566	517.465258789063
2	['Input', '(relu, 16)'](softmax, 32)(relu, 1)	35.7142857142857	39.8673400878906	4154.95717773438
3	['Input', '(relu, 16)', '(relu, 16)'](softmax, 16)(relu, 1)	35.7142857142857	39.2523948669434	4147.36657714844
1	['Input'](softmax, 32)(relu, 1)	34.2857142857143	39.4227863311768	4167.7181640625
2	['Input', '(relu, 16)'](softmax, 64)(relu, 1)	34.2857142857143	42.2330394744873	4388.02092285156
3	['Input', '(relu, 16)', '(relu, 16)'](tanh, 32)(relu, 1)	34.2857142857143	28.9200885772705	1579.15971679688
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](sigmoid, 32)(relu, 1)	34.2857142857143	37.306615447998	2711.53388061523
1	['Input'](softmax, 256)(relu, 1)	34.2857142857143	38.6612678527832	4123.48552246094
2	['Input', '(relu, 16)'](softmax, 16)(relu, 1)	34.2857142857143	40.6428840637207	4327.66945800781
1	['Input'](softmax, 16)(relu, 1)	32.8571428571429	38.8617206573486	4092.88757324219
1	['Input'](softmax, 64)(relu, 1)	32.8571428571429	38.7530979156494	4126.82353515625
3	['Input', '(relu, 16)', '(relu, 16)'](softmax, 64)(relu, 1)	32.8571428571429	40.2661521911621	4243.10627441406
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](softplus, 16)(relu, 1)	30	49.5115055084229	4515.64162597656
3	['Input', '(relu, 16)', '(relu, 16)'](softmax, 256)(relu, 1)	27.1428571428571	45.3680030822754	4867.02453613281
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](softmax, 32)(relu, 1)	24.2857142857143	59.4351722717285	7654.32744140625
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](softplus, 32)(relu, 1)	21.4285714285714	62.9648452758789	6371.03427734375
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](elu, 32)(relu, 1)	21.4285714285714	62.1296684265137	6413.2013671875
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](selu, 32)(relu, 1)	21.4285714285714	61.7426940917969	6425.133984375
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](elu, 64)(relu, 1)	21.4285714285714	63.3681259155273	6371.81796875
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](linear, 64)(relu, 1)	21.4285714285714	62.0812698364258	6399.54306640625
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](relu, 64)(relu, 1)	21.4285714285714	62.3026557922363	6396.41767578125
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](selu, 64)(relu, 1)	21.4285714285714	61.9159820556641	6410.9837890625
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](elu, 256)(relu, 1)	21.4285714285714	62.3297271728516	6387.1904296875
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](linear, 256)(relu, 1)	21.4285714285714	62.3240798950195	6387.433203125
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](selu, 256)(relu, 1)	21.4285714285714	62.2524353027344	6390.0150390625
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](selu, 64)(relu, 1)	20	64.3753875732422	6342.18056640625
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](softmax, 256)(relu, 1)	20	60.7413429260254	7151.1982421875
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](softplus, 32)(relu, 1)	20	62.2182159423828	6402.73173828125
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](relu, 256)(relu, 1)	20	63.9850540161133	6360.0083984375
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](sigmoid, 256)(relu, 1)	18.5714285714286	61.4228927612305	6446.5111328125
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](tanh, 256)(relu, 1)	18.5714285714286	61.3794303894043	6485.68642578125
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](softmax, 16)(relu, 1)	17.1428571428571	62.1854415893555	6921.671875
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](softmax, 32)(relu, 1)	17.1428571428571	61.5490798950195	6940.65439453125
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](softmax, 64)(relu, 1)	17.1428571428571	62.1368087768555	6924.9255859375
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](softmax, 256)(relu, 1)	17.1428571428571	62.0962554931641	6928.64013671875
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](selu, 16)(relu, 1)	17.1428571428571	61.0574165344238	6483.11015625
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](relu, 32)(relu, 1)	15.7142857142857	61.0851753234863	6479.133203125
3	['Input', '(relu, 16)', '(relu, 16)'](softmax, 32)(relu, 1)	15.7142857142857	67.0958969116211	8039.14189453125
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](tanh, 32)(relu, 1)	15.7142857142857	60.3236419677734	6653.98876953125
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](tanh, 64)(relu, 1)	15.7142857142857	59.9715553283691	6634.58466796875
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](softmax, 16)(relu, 1)	14.2857142857143	64.7871482849121	6941.62783203125
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](softmax, 64)(relu, 1)	14.2857142857143	60.7869117736816	7135.0166015625
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](relu, 16)(relu, 1)	12.8571428571429	60.4784553527832	6549.4212890625
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](softplus, 16)(relu, 1)	12.8571428571429	60.9588584899902	6493.40048828125
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](sigmoid, 64)(relu, 1)	10	60.5727638244629	6545.8826171875
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](sigmoid, 16)(relu, 1)	8.57142857142857	60.7275970458984	6585.6029296875
1	['Input'](relu, 16)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
1	['Input'](sigmoid, 16)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
1	['Input'](softplus, 16)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
1	['Input'](elu, 32)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
1	['Input'](linear, 32)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
1	['Input'](sigmoid, 64)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
1	['Input'](softplus, 64)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
1	['Input'](tanh, 64)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
2	['Input', '(relu, 16)'](relu, 16)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
2	['Input', '(relu, 16)'](sigmoid, 16)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
2	['Input', '(relu, 16)'](relu, 32)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
2	['Input', '(relu, 16)'](tanh, 32)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
2	['Input', '(relu, 16)'](elu, 64)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
2	['Input', '(relu, 16)'](selu, 64)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
3	['Input', '(relu, 16)', '(relu, 16)'](elu, 16)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
3	['Input', '(relu, 16)', '(relu, 16)'](selu, 16)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
3	['Input', '(relu, 16)', '(relu, 16)'](linear, 32)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
3	['Input', '(relu, 16)', '(relu, 16)'](softplus, 32)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
3	['Input', '(relu, 16)', '(relu, 16)'](selu, 64)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
3	['Input', '(relu, 16)', '(relu, 16)'](elu, 256)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](sigmoid, 16)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](tanh, 16)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375

Table B.5: (Cont): Tested configurations for the RAM parameter

Layer Count	Layer Structure	Accuracy (%)	MAE	MSE
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](tanh, 32)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](elu, 64)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](linear, 64)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](relu, 64)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](sigmoid, 64)(relu, 1)	7.14285714285714	58.8310134887695	6803.03896484375
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](softplus, 64)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](tanh, 64)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](elu, 256)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](linear, 256)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](relu, 256)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](selu, 256)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](sigmoid, 256)(relu, 1)	7.14285714285714	59.5741554260254	6675.1984375
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](softplus, 256)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
4	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)'](tanh, 256)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](elu, 16)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](linear, 16)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](tanh, 16)(relu, 1)	7.14285714285714	72.428137207031 3	6828.09765625
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](linear, 32)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](sigmoid, 32)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](softplus, 64)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375
5	['Input', '(relu, 16)', '(relu, 16)', '(elu, 16)', '(sigmoid, 16)'](softplus, 256)(relu, 1)	7.14285714285714	88.5857925415039	14803.2021484375

Table B.6: (Cont): Tested configurations for the RAM parameter

Tested configurations for the ROM parameter

Layer Count	Layer Structure	Accuracy (%)	MAE	MSE
2	['Input', '(linear, 16)'](softplus, 32)(relu, 1)	78.5714285714286	219.797821044922	302862.8125
2	['Input', '(linear, 16)'](relu, 32)(relu, 1)	74.2857142857143	222.082034301758	315975.35
1	['Input'](tanh, 256)(relu, 1)	74.2857142857143	229.487869262695	251704.890625
1	['Input'](relu, 256)(relu, 1)	72.8571428571429	226.926083374023	266890.375
2	['Input', '(linear, 16)'](softplus, 16)(relu, 1)	72.8571428571429	237.820852661133	297688.46875
2	['Input', '(linear, 16)'](selu, 32)(relu, 1)	72.8571428571429	228.49582824707	307614
1	['Input'](sigmoid, 256)(relu, 1)	72.8571428571428	226.403927612305	237745.728125
1	['Input'](elu, 32)(relu, 1)	71.4285714285714	223.776629638672	247434.821875
1	['Input'](selu, 64)(relu, 1)	71.4285714285714	220.689810180664	253162.953125
1	['Input'](elu, 256)(relu, 1)	71.4285714285714	221.545419311523	264569.459375
1	['Input'](selu, 256)(relu, 1)	71.4285714285714	223.654052734375	266029.896875
2	['Input', '(linear, 16)'](relu, 16)(relu, 1)	71.4285714285714	240.313565063477	288903.0875
2	['Input', '(linear, 16)'](tanh, 64)(relu, 1)	71.4285714285714	238.403494262695	244638.834375
1	['Input'](selu, 16)(relu, 1)	70	225.549249267578	240920.94375
1	['Input'](selu, 32)(relu, 1)	70	225.328265380859	250060.49375
1	['Input'](elu, 64)(relu, 1)	70	221.142437744141	253199.025
1	['Input'](linear, 256)(relu, 1)	70	239.903677368164	280770.246875
2	['Input', '(linear, 16)'](elu, 32)(relu, 1)	70	233.393334960937	314667.275
2	['Input', '(linear, 16)'](linear, 32)(relu, 1)	70	241.064077758789	294798.54375
3	['Input', '(linear, 16)', '(linear, 16)'](sigmoid, 256)(relu, 1)	70	232.342431640625	235497.028125
1	['Input'](relu, 64)(relu, 1)	70	233.383538818359	255291.4
1	['Input'](elu, 16)(relu, 1)	68.5714285714286	228.196057128906	238700.45625
1	['Input'](relu, 32)(relu, 1)	68.5714285714286	233.576138305664	249632.65625
2	['Input', '(linear, 16)'](elu, 16)(relu, 1)	68.5714285714286	238.667105102539	278596.340625
1	['Input'](tanh, 64)(relu, 1)	68.5714285714286	237.49235534668	251478.871875
1	['Input'](linear, 64)(relu, 1)	67.1428571428572	238.347174072266	266229.36875
1	['Input'](softplus, 64)(relu, 1)	67.1428571428572	240.236361694336	267608.9375
1	['Input'](linear, 32)(relu, 1)	67.1428571428571	238.646716308594	260418.3125
3	['Input', '(linear, 16)', '(linear, 16)'](tanh, 256)(relu, 1)	67.1428571428571	242.727200317383	233701.725
2	['Input', '(linear, 16)'](linear, 64)(relu, 1)	65.7142857142857	241.694564819336	296895.55
2	['Input', '(linear, 16)'](relu, 64)(relu, 1)	65.7142857142857	233.303628540039	329855.7625
1	['Input'](sigmoid, 64)(relu, 1)	64.2857142857143	244.57458190918	263343.125
2	['Input', '(linear, 16)'](sigmoid, 64)(relu, 1)	64.2857142857143	246.784432983398	233185.946875
2	['Input', '(linear, 16)'](linear, 256)(relu, 1)	64.2857142857143	242.468096923828	298276.1625
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](sigmoid, 256)(relu, 1)	64.2857142857143	256.616464233398	279308.6125
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](tanh, 256)(relu, 1)	64.2857142857143	258.178601074219	268349.58125
1	['Input'](relu, 16)(relu, 1)	62.8571428571429	243.242810058594	251330.890625
2	['Input', '(linear, 16)'](tanh, 32)(relu, 1)	62.8571428571429	251.698071289063	274543.009375
2	['Input', '(linear, 16)'](selu, 64)(relu, 1)	62.8571428571429	226.213070678711	333376.49375
3	['Input', '(linear, 16)', '(linear, 16)'](linear, 64)(relu, 1)	62.8571428571429	241.361389160156	316599.6875
1	['Input'](tanh, 16)(relu, 1)	60	275.176885986328	292675.290625

Table B.7: Tested configurations for the ROM parameter

Layer Count	Layer Structure	Accuracy (%)	MAE	MSE
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](selu, 256)(relu, 1)	60	250.258349609375	351324.775
3	['Input', '(linear, 16)', '(linear, 16)'](elu, 16)(relu, 1)	58.5714285714286	260.562301635742	382198.94375
1	['Input'](tanh, 32)(relu, 1)	58.5714285714286	253.146691894531	258778.225
2	['Input', '(linear, 16)'](sigmoid, 32)(relu, 1)	58.5714285714286	275.910934448242	304580.33125
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](linear, 16)(relu, 1)	58.5714285714286	233.917324829102	327906.4125
3	['Input', '(linear, 16)', '(linear, 16)'](softplus, 16)(relu, 1)	57.1428571428571	254.3509765625	390000.78125
2	['Input', '(linear, 16)'](selu, 256)(relu, 1)	55.7142857142857	304.467059326172	488499.8125
3	['Input', '(linear, 16)', '(linear, 16)'](selu, 16)(relu, 1)	55.7142857142857	274.066751098633	431755.625
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](selu, 64)(relu, 1)	55.7142857142857	306.329827880859	436294.33125
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](elu, 256)(relu, 1)	55.7142857142857	360.745147705078	352769.14375
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](relu, 256)(relu, 1)	55.7142857142857	359.524237060547	352707.2875
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](selu, 256)(relu, 1)	55.7142857142857	359.763989257813	352467.1125
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](softplus, 256)(relu, 1)	55.7142857142857	355.894421386719	352364.8
1	['Input'](sigmoid, 32)(relu, 1)	54.2857142857143	269.092385864258	269067.8625
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](selu, 16)(relu, 1)	54.2857142857143	239.949487304687	356606.75625
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](elu, 64)(relu, 1)	54.2857142857143	371.764910888672	357288.175
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](linear, 64)(relu, 1)	54.2857142857143	365.841912841797	355212.9625
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](relu, 64)(relu, 1)	54.2857142857143	368.711932373047	356849.6875
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](softplus, 64)(relu, 1)	54.2857142857143	365.683343505859	355578.48125
3	['Input', '(linear, 16)', '(linear, 16)'](selu, 32)(relu, 1)	52.8571428571429	312.224694824219	495589.975
3	['Input', '(linear, 16)', '(linear, 16)'](selu, 64)(relu, 1)	52.8571428571429	267.184927368164	388142.06875
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](linear, 32)(relu, 1)	52.8571428571429	372.316204833984	358452.88125
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](relu, 32)(relu, 1)	52.8571428571429	375.033160400391	360172.4875
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](selu, 32)(relu, 1)	52.8571428571429	377.758837890625	360218.50625
2	['Input', '(linear, 16)'](elu, 256)(relu, 1)	51.4285714285714	319.127124023437	495509.14375
2	['Input', '(linear, 16)'](relu, 256)(relu, 1)	51.4285714285714	291.897509765625	445041.86875
3	['Input', '(linear, 16)', '(linear, 16)'](softplus, 32)(relu, 1)	51.4285714285714	308.991619873047	506355.0625
3	['Input', '(linear, 16)', '(linear, 16)'](elu, 64)(relu, 1)	51.4285714285714	325.489642333984	527801.8375
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](selu, 32)(relu, 1)	51.4285714285714	258.998355102539	385606.35
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](elu, 64)(relu, 1)	51.4285714285714	271.648779296875	388908.9625
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](linear, 16)(relu, 1)	51.4285714285714	380.194799804687	362714.18125
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](softplus, 16)(relu, 1)	51.4285714285714	381.691967773438	363897.81875
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](elu, 32)(relu, 1)	51.4285714285714	382.657000732422	362733.525
1	['Input'](sigmoid, 16)(relu, 1)	50	294.154162597656	311334.703125
3	['Input', '(linear, 16)', '(linear, 16)'](relu, 32)(relu, 1)	50	282.51123046875	446813.00625
3	['Input', '(linear, 16)', '(linear, 16)'](selu, 256)(relu, 1)	50	314.125830078125	490648.4
3	['Input', '(linear, 16)', '(linear, 16)'](softplus, 256)(relu, 1)	50	290.909826660156	417108.475
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](elu, 16)(relu, 1)	50	387.77607421875	366419.29375
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](relu, 16)(relu, 1)	50	387.247528076172	367087.28125
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](selu, 16)(relu, 1)	50	386.029986572266	365203.5625
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](tanh, 16)(relu, 1)	50	382.705975341797	385402
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](tanh, 32)(relu, 1)	50	388.6544921875	367989.9375
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](tanh, 64)(relu, 1)	50	391.521160888672	365550.34375
3	['Input', '(linear, 16)', '(linear, 16)'](relu, 16)(relu, 1)	47.1428571428571	262.689910888672	388468.15625
3	['Input', '(linear, 16)', '(linear, 16)'](tanh, 32)(relu, 1)	47.1428571428571	298.640447998047	354533.01875
3	['Input', '(linear, 16)', '(linear, 16)'](softplus, 64)(relu, 1)	47.1428571428571	295.434149169922	442235.1875
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](softplus, 32)(relu, 1)	47.1428571428571	283.257598876953	433039.35625
2	['Input', '(linear, 16)'](softplus, 64)(relu, 1)	47.1428571428571	262.217391967773	394551.8875
3	['Input', '(linear, 16)', '(linear, 16)'](relu, 64)(relu, 1)	45.7142857142857	316.928546142578	484318.46875
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](softplus, 16)(relu, 1)	45.7142857142857	287.275689697266	430579.31875
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](relu, 16)(relu, 1)	45.7142857142857	288.392517089844	415820.8375
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](tanh, 256)(relu, 1)	44.2857142857143	387.516271972656	362758.68125
2	['Input', '(linear, 16)'](tanh, 16)(relu, 1)	42.8571428571429	411.027996826172	411213.66875
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](tanh, 32)(relu, 1)	42.8571428571429	379.061370849609	383889.46875
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](tanh, 64)(relu, 1)	42.8571428571429	350.488641357422	329268.4375
3	['Input', '(linear, 16)', '(linear, 16)'](sigmoid, 64)(relu, 1)	40	417.819091796875	387329.0125
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](elu, 16)(relu, 1)	40	294.83459777832	395521.45625
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](tanh, 16)(relu, 1)	40	406.0205078125	448188.83125
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](sigmoid, 64)(relu, 1)	40	371.855670166016	339942.26875
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](sigmoid, 32)(relu, 1)	38.5714285714286	390.287664794922	403898.16875
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](elu, 32)(relu, 1)	37.1428571428571	272.708880615234	404535.4125
3	['Input', '(linear, 16)', '(linear, 16)'](tanh, 64)(relu, 1)	35.7142857142857	422.120715332031	395671.7125
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](sigmoid, 16)(relu, 1)	34.2857142857143	408.860729980469	432516.275
3	['Input', '(linear, 16)', '(linear, 16)'](tanh, 16)(relu, 1)	32.8571428571429	432.432641601563	459618.35625
3	['Input', '(linear, 16)', '(linear, 16)'](sigmoid, 32)(relu, 1)	30	426.602661132812	459016.175
1	['Input'](softmax, 64)(relu, 1)	15.7142857142857	505.627166748047	664668.9875
2	['Input', '(linear, 16)'](softmax, 32)(relu, 1)	15.7142857142857	506.079437255859	664913.6875
1	['Input'](softmax, 16)(relu, 1)	14.2857142857143	506.583630371094	664643.0375
1	['Input'](softmax, 256)(relu, 1)	14.2857142857143	506.813836669922	664874.8125
2	['Input', '(linear, 16)'](softmax, 64)(relu, 1)	14.2857142857143	507.099188232422	665000.225
2	['Input', '(linear, 16)'](softmax, 256)(relu, 1)	14.2857142857143	507.070825195313	664918.4125

Table B.8: (Cont): Tested configurations for the ROM parameter

Layer Count	Layer Structure	Accuracy (%)	MAE	MSE
3	['Input', '(linear, 16)', '(linear, 16)'](softmax, 256)(relu, 1)	14.2857142857143	507.345153808594	665038.6
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](softmax, 256)(relu, 1)	14.2857142857143	508.2921875	666209.5125
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](softmax, 16)(relu, 1)	14.2857142857143	507.486401367188	665603.7875
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](softmax, 64)(relu, 1)	14.2857142857143	506.978594970703	664890.1375
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](softmax, 256)(relu, 1)	14.2857142857143	507.053765869141	664957.6875
2	['Input', '(linear, 16)'](softmax, 16)(relu, 1)	12.8571428571429	507.708868408203	665280.9125
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](softmax, 16)(relu, 1)	12.8571428571429	530.741485595703	692837.2625
3	['Input', '(linear, 16)', '(linear, 16)'](softmax, 16)(relu, 1)	11.4285714285714	522.609130859375	672262.8375
3	['Input', '(linear, 16)', '(linear, 16)'](softmax, 32)(relu, 1)	11.4285714285714	513.796038818359	667581.225
3	['Input', '(linear, 16)', '(linear, 16)'](softmax, 64)(relu, 1)	11.4285714285714	522.551416015625	672183.6125
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](softmax, 64)(relu, 1)	11.4285714285714	511.811383056641	666277.65
1	['Input'](linear, 16)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
1	['Input'](softplus, 16)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
1	['Input'](softmax, 32)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
1	['Input'](softplus, 32)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
1	['Input'](softplus, 256)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
2	['Input', '(linear, 16)'](linear, 16)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
2	['Input', '(linear, 16)'](selu, 16)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
2	['Input', '(linear, 16)'](sigmoid, 16)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
2	['Input', '(linear, 16)'](elu, 64)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
2	['Input', '(linear, 16)'](sigmoid, 256)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
2	['Input', '(linear, 16)'](softplus, 256)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
2	['Input', '(linear, 16)'](tanh, 256)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
3	['Input', '(linear, 16)', '(linear, 16)'](linear, 16)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
3	['Input', '(linear, 16)', '(linear, 16)'](sigmoid, 16)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
3	['Input', '(linear, 16)', '(linear, 16)'](elu, 32)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
3	['Input', '(linear, 16)', '(linear, 16)'](linear, 32)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
3	['Input', '(linear, 16)', '(linear, 16)'](elu, 256)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
3	['Input', '(linear, 16)', '(linear, 16)'](linear, 256)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
3	['Input', '(linear, 16)', '(linear, 16)'](relu, 256)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](linear, 32)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](relu, 32)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](linear, 64)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](relu, 64)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](softplus, 64)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](elu, 256)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](linear, 256)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](relu, 256)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](softplus, 256)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](sigmoid, 16)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](sigmoid, 32)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](softmax, 32)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](softplus, 32)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](selu, 64)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](sigmoid, 64)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](linear, 256)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
5	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)', '(softmax, 32)'](sigmoid, 256)(relu, 1)	7.14285714285714	653.846801757812	855252.8125
4	['Input', '(linear, 16)', '(linear, 16)', '(linear, 16)'](softmax, 32)(relu, 1)	5.71428571428571	535.852270507812	698949.7375

Table B.9: (Cont): Tested configurations for the ROM parameter

C

Appendix C - Informal interview for determining hardware validation and selection approaches at Volvo Cars Corporation

Interview about deducting strategies for deciding hardware

Length: 45-60 minutes

Introduction

This semi-structured interview is about gathering data on how a suitable hardware baseline (microcontroller) is chosen for an ECU. As well as what validation techniques exist for determining the efficiency of an ECU.

The results of this interview will be used as a foundation to validate a tool created that given a set of high-level features, can propose a suitable hardware baseline. This validation will consist of comparing the current approach for hardware approximation to the tool. All information gathered here will be completely anonymous.

Verbal Consent

Would you like to participate in this interview? And are I allowed to record your answers in order to derive results that I might have missed to document?

1. **Verbal Consent WAS obtained from the study participant**
2. Verbal consent was NOT obtained from the study participant

Background Information

What is your relation to the ECUs created at Volvo, and the hardware selection? *Have worked many years with Volvo systems, leads the technical development at the respective ECU team.*

Hardware Validation at Volvo

These questions aim to get an understanding of how hardware delivered from subcontractors are validated.

How are the ECUs delivered from subcontractors validated? I.e. how do you determine whether these ECUs behave as they should?

This team develops some of the software in-house, and the BSW is developed by subcontractors. By sharing a testing suite with the subcontractor, the team can determine whether the BSW functions as it should. As the main responsibility for the hardware lies on this team for their ECU, they ensure that the software takes up fewer resources than what is present on the ECU.

Deciding on hardware for a new ECU

These questions aim to get an understanding of how a suitable baseline is selected for a new ECU project.

How do you decide on a suitable microcontroller for a new ECU? I.e. how do you know what hardware baselines are necessary?

When the interviewee's team determined an ECU; they looked back on previous similar ECUs as a baseline for determining a suitable hardware baseline, from this they did an estimation on the applications that should run on the ECU.
The application needed to have 2-4 different variants of the same behavior to cover all the use cases of the ECU. As such, a large amount of ROM and RAM was required. From their estimation on their own software component size, they realized that there was no sufficient ECU that could handle these requirements and as such, they decided on rewriting the algorithms for the SWC.

From these activities, they got an estimation on the ECU HW requirements, which they used to choose a suitable microcontroller. From this, they also talked to the hardware subcontractors that delivers the peripherals of the ECU (i.e. the functional parts of the ECU that is not the processor, for example, all the ports) to come to an

agreement if the chosen micro-controller was apt.

As such, choosing an ECU is largely dependent on knowledge and data from previously delivered ECUs that are similar to what is developed. Should this knowledge be unavailable, finding suitable hardware becomes difficult.

Do you think that a tool that can propose a suitable hardware baseline could assist in this decision?

Yes. Especially if one could further extend the tool and get even more precise measures.