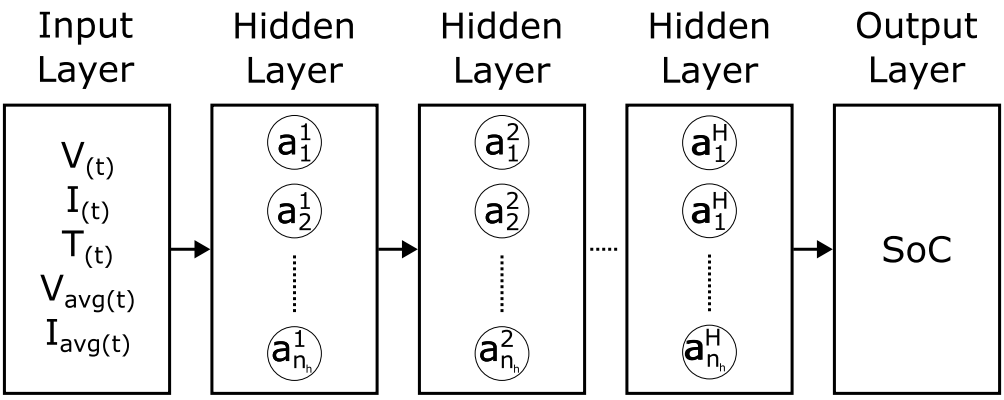


State of Charge Estimation of Lithium-Ion Batteries Using Deep Learning

Master's Thesis in Systems, Control and Mechatronics

SHAHROOZ FOROUHAN

DEPARTMENT OF ELECTRICAL ENGINEERING

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

Master's thesis in Systems, control and mechatronics
2022

State of Charge Estimation of Lithium-Ion Batteries Using Deep Learning

Shahrooz Forouhan

CHALMERS
UNIVERSITY OF TECHNOLOGY

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

State of Charge Estimation of Lithium-Ion Batteries Using Deep Learning
Shahrooz Forouhan

Supervisor: Suraj Danaraddi, Knightec
Supervisor: Yang Li, Chalmers University of Technology
Examiner: Changfu Zou, Chalmers University of Technology

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

Cover: Illustration of a deep MLP that calculates the state of charge.

Typeset in LaTeX
Printed by Chalmers Digital Printing
Gothenburg, Sweden 2022

State of Charge Estimation of Lithium-Ion Batteries Using Deep Learning
Shahrooz Forouhan
Department of Electrical Engineering
Chalmers University of Technology

Abstract

State of Charge (SoC) estimation of lithium-ion (Li-ion) batteries is an essential component for electric vehicles where SoC ensures that the batteries enjoy safe operations and longer service life. In this thesis, a couple of deep learning (DL) methods are investigated to estimate the SoC, aiming at achieving high accuracy with low computation. The DL methods that are used include Fully Connected Neural Networks (FCNN/MLP), Deep Feedforward Neural Network (FNN), Stochastic Delta Rule (DropConnect), and Residual Network (ResNet). The ResNet-based method with low complexity is specifically focused on due to its high performance compared to the other models under study. Public battery testing data under various ambient temperatures and dynamic charging/discharging profiles are used to mimic real-life scenarios. The dataset is then fed into the respective network where it is trained and validated. Next, it is put on a test set in order to observe how well it performs. The test showed that the ResNet models have the best performance compared to the other non-ResNet models. However, the non-ResNet models were faster than the ResNet model and managed to get decent performance even though it is not as good as the ResNet models. It is found that the ResNet with 5 residual blocks or skip connections outperforms other configurations in terms of performance and time. However, the ResNet model that had the best performance at the cost of time is ResNet $k = 10$.

Keywords: Battery management system, Deep feedforward neural network, Deep learning, Fully connected neural network, Machine learning, Neural network, Residual network, State of charge, Stochastic delta rule

Acknowledgements

I would like to thank my supervisors Suraj Danaraddi and Yang Li who both provided me with guidance in order to complete this thesis. They pushed me in the right direction and provided sources and data which was essential for this thesis. I would also like to thank Suraj's colleagues at Knightec, who provided me with physical and digital material that was needed to proceed with this thesis. Finally, I would like to thank my examiner Changfu Zou, who helped me with the logistics and information/material on how to plan and work on a master's thesis.

Shahrooz Forouhan, Gothenburg, July 2022

Table of Contents

List of Figures

List of Tables

List of Abbreviation

BMS	Battery Management System
CPU	Central Processing Unit
DL	Deep Learning
DropConnect	Stochastic Delta Rule
FCNN	Fully Connected Neural Network
FNN	Deep Feedforward Neural Network
GPU	Graphics Processing Unit
HWFET	Highway Fuel Economy Test Cycle
Li-ion	Lithium-ion
MAE	Mean Absolute Error
ML	Machine Learning
MSE	Mean Squared Error
MLP	Multilayered Perceptron
RAM	Random-Access Memory
ResNet	Residual Network
RMSE	Root-Mean-Square Error
SoC	State of Charge
SoH	State of Health
UDDS	Urban Dynamometer Driving Schedule

Nomenclature

α	Constant gradient
b	Bias
η	Learning rate
$f(x, \theta)$	Network/model
$f^*(x)$	Target function
g	Activation function
H	Number of hidden layers
I	Current
I_{avg}	Average current
$J(\theta)$	Loss function
k	Number of residual blocks
L	Horizon for calculating average values
m	Number of data entries
M	Number of nodes
n_h	Number of hidden units
n_x	Input size
n_y	Output size
N	Number of nodes
$Norm_X$	Normalized input data matrix
$Norm_Y$	Normalized output data matrix
P	Dropout
$SoC(t)$	Measured state of charge
$SoC^*(t)$	Predicted state of charge
θ	Model parameters
T	Temperature
t	Index of time

V	Voltage
V_{avg}	Average voltage
x	Input
X	Inputs data matrix
$\mathbb{X}$	The training set
y	Output
$\hat{y}$	Predicted value of y
Y	Output data matrix
w	Weight

1. Introduction

1.1 Background

Technological breakthroughs in lithium-ion (Li-ion) batteries have played a key role in improving the performance and extending the driving range of electric vehicles [1]. A Battery Management System (BMS) is an essential component for Li-ion battery-driven vehicles, where functionalities such as state estimation, fault diagnosis, cell equalization, and thermal management are integrated [2]. There is a growing interest in developing a prototype BMS as a knowledge-sharing platform where developers can increase their know-how on components and functionalities involved in an electric powertrain [3]. Amongst the various functionalities, battery State of Charge (SoC) estimation has received extensive research attention. SoC is essential for battery operation and knowing accurate SoC is important since it provides the information for ensuring the drivability and preventing damaging behaviors such as overcharge/overdischarge of the battery pack/cell [4].

However, there are some difficulties with conventional battery SoC estimation techniques which mostly are empirical or rely on mathematical models. Usually, there can be significant unmodeled nonlinear and highly complex processes in the dynamic behaviors of coupled electrical, electrochemical, and thermal systems of the battery packs [5]. This makes it hard to establish a simple while accurate model for SoC estimation under wide operating regimes. Also, the internal design for the batteries usually differs substantially in laboratory environments compared to real-world conditions, raising some concerns over the extrapolability of conventional models for SoC estimation [6], [7].

One active research area in BMS is improving the efficacy of SoC estimation for a Li-ion battery pack using machine learning (ML) algorithms. The benefit of using ML is that it can be very efficient and does not require specific domain knowledge in estimating the SoC, so the effects of unmodeled behaviors can be readily considered [8]. ML is excellent in capturing the nonlinear correlations that SoC bears, and the algorithm usually has faster convergence compared to other methods. ML methods can also examine SoC with good accuracy with few measurements and do not need any information about the batteries' chemistry, reactions, or other model parameters. These characteristics make ML appropriate for many real-life applications. Nevertheless, simply applying a neural network can lead to a maximum mean absolute error of 3.8% for SoC estimation [9], which does not compete with many state-of-the-art model-based techniques in terms of accuracy.

In order to improve the SoC estimation performance, deep learning (DL) algorithms

are proposed in the literature for developing data-driven methods [10]. However, developing a DL model is usually time-consuming and resource-demanding. In fact, an effective battery SoC estimation method should have low computational requirements for gaining high estimation accuracy [11].

In [12], it is discussed in general how ML can be used to estimate SoC for electric vehicles, and a deep neural network (DNN) is proposed for this purpose. The authors obtained the data from a laboratory where the temperature varies between −20 °C and 25 °C. The conclusion is that the DNN can estimate SoC well at different temperatures. For temperatures around 25 °C, SoC estimation has a mean absolute error of 1.10% and at around −20 °C degrees, the error is around 2.17%. The SoC estimation accuracy also increases when the authors add some random noise to their data and it also makes the DNN computationally more efficient.

Wang et al. [13] use a multilayer feedforward neural network (FNN) for SoC estimation and they argue that the algorithm can be used to interpret the responses of the battery. The authors identified that the mean absolute error is around 2%, and they argue that more tests on real-world conditions should be done, and they should be tested on "older" batteries in order to observe how it works. They also recommend SoC estimation should be combined with the consideration of the battery state of health (SoH).

In [14], the authors developed an ML method for SoC estimation based on a Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM). The model is trained with datasets on different temperatures. The authors show that the RNN has a mean absolute error of 0.573% and with changing temperatures the maximum error is around 1.606%. The training algorithm also saves time for researchers by decreasing the amount of testing for parameterization. In a patent application by Chemali and Prenidl [15], the authors describe more about the RNN and Convolutional Neural Network (CNN) for estimating the SoC. Under ambient temperatures of 10 °C and 25 °C, the estimation errors vary between 0.774% and 0.782%. Though, it is also acknowledged that the SoH of the batteries is important to study and further recommend that real-world scenarios (e.g., under different battery temperatures) should be investigated in order to achieve high estimation accuracy.

How et al. [16] use ML techniques based on the transformers to estimate SoC for Li-ion batteries. The self-supervised learning is adopted, and the obtained mean absolute error is around 0.44% at a constant ambient temperature and 0.7% for varying temperatures. The authors train the model for very few epochs with around 20% of the total data. It also shows that the developed model works for different Li-ion battery cells.

How et al. reveal in [17] that the neural network and ML methods are good for estimating SoC because batteries have highly nonlinear characteristics affected by factors such as aging, temperature, etc. However, the authors also note that conventional neural networks are constrained by the training process, and achieving high accuracy based on their hyperparameters can be very time-consuming.

In view of the above, ML/DL is found to be an interesting and effective method

for SoC estimation as it can improve the accuracy and robustness of the estimation without the need to delve into the complex internal mechanisms of the battery cells. Also, because technology has advanced through higher computation capabilities and big data availability, using ML/DL is expected to be promising and popularized in the future BMS of electric vehicles. Although a few ML/DL methods have been proposed for SoC estimation, after the literature review, the existing algorithms have the following problems:

1. Some of the algorithms need significant computational capability for SoC estimation, especially during the training process.

2. Many of the ML algorithms still have fairly high errors and they do not compare their results with other research that uses the same dataset. Because every battery cell varies, the results from one article cannot be properly compared to another article. Though, it should give a clue as to what the estimation should be.

1.2 Aim

The main aim of the present study is to investigate the possibility of using and comparing several state-of-the-art DL algorithms with different complexities for battery SoC estimation. These selected algorithms include Residual Network (ResNet), fully connected neural networks (FCNN), FNN, and Stochastic Delta Rule (DropConnect). The aim is to reveal what difficulties the designer might encounter when using each algorithm and what advantages/disadvantages the algorithms have for BMS applications.

The main questions that will be investigated in the following chapters are:

- [Q1] What advantages/disadvantages does each machine learning model have for estimating the SoC?

- [Q2] Which machine learning model is more suitable for estimating the SoC?

- [Q3] Would it be possible that a simple deep machine learning method can outperform a more complex/deeper machine learning method when estimating the SoC?

1.3 Scope

The scope of the study is to find the most suitable DL algorithms that can potentially have good SoC estimation performance from a given list of algorithms. The focus will be neither on investigating and comparing with other SoC estimation techniques such as model-based methods nor on other BMS functionalities such as the SoH estimation, fault diagnosis, and thermal management. In order to match with

real-world scenarios, models will be tested for data with various temperatures and charging/discharge rates.

1.4 Ethical and Sustainability Aspects

The Li-ion battery has had some controversies over the years. Mostly, the controversies come from countries where it gets mined and how it is wasted after it has been obsolete [18]. One issue is when it is over-mined, some hazardous waste can get into local rivers which can lead to contamination in their water resources. However, by optimizing the SoC, the Li-ion batteries will be able to be more effective, thus leading to less demand, consumption, and mining of the material.

The gathering of the data may also have an environmental impact. The data are either taken from a battery electric vehicle or a laboratory. Data taken from the battery electric vehicle for experimentation purposes may affect the battery in the car due to it getting worse over time, which will lead to faster battery replacements than if it was not used for those purposes. Batteries used in a laboratory also age and needs to be replaced. However, because the goal is to improve the battery and extent its life expectancy, the advantages outweigh the disadvantages.

1.5 Thesis Outline

The introductory chapter is followed by a theoretical chapter that provides a basis to help understand the thesis. It includes the theory behind how BMS and ML work both separately and together. Next, the methodology is presented including how the data are prepared for the training and how the DL models are designed. Next, the results are provided together with the discussion. Finally, the conclusion of the thesis and suggestions for future work are given.

2. Theory

In this chapter, the theory behind the methods and results is presented. The first part introduces the basic functionalities of a BMS. The second part briefly present how different ML/DL algorithms work in theory. Finally, theories for ML/DL-based SoC estimation are provided.

2.1 Battery Management System (BMS)

In general, the main functionalities of a BMS can include [19]:

- Protecting the safety of the operated device's operator and identifies unsafe conditions, and sends a signal if it happens.
- Preventing the battery cells from being damaged in failure situations.
- Extending the life expectancy of the battery.
- Keeping the battery in a position where it can maintain its design requirements and prevent deterioration.
- Sending information to the application controller on how to use the battery pack through, e.g., power limits.

Specifically, a BMS for vehicular applications where large and high voltage/current battery packs are used is designed to achieve the following tasks.

1. Having and checking high-voltage where it measures voltage, current, temperature; control contactor, pre-charge; ground-fault detection and thermal management.
2. BMS protects against overcharge/discharge/current, short circuit, and high temperatures.
3. BMS provides an interface that checks range estimation, data recording, reporting, and communications.
4. Has performance management where it checks the SoC, power-limit computation, and equalizes cells.

5. Looks at the diagnostics where it checks abuse detection, SoH, and state of life (SoL).

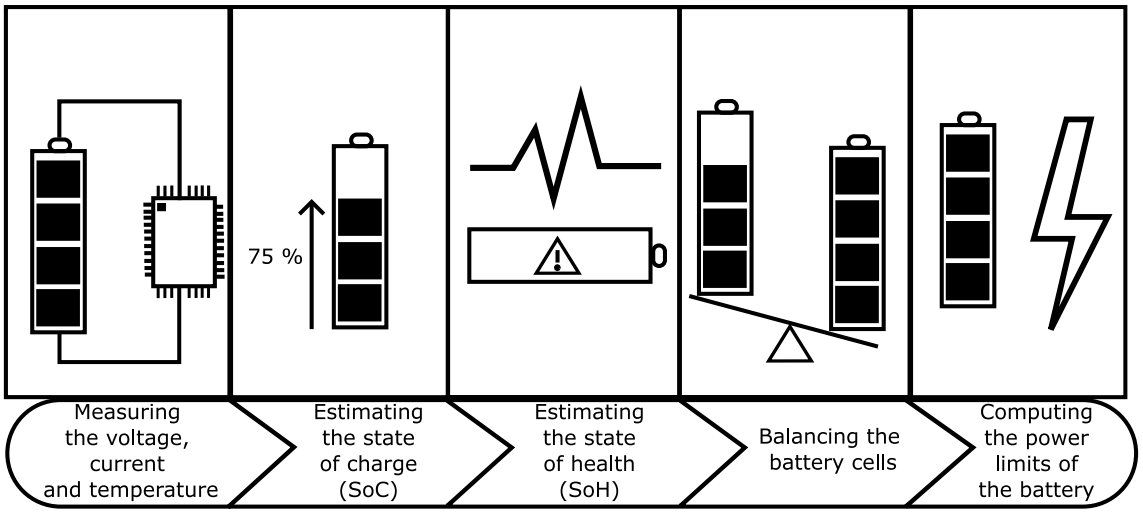


Figure 2.1: Main functionalities in a battery management system.

2.1.1 Measurements in a Battery Pack

The main variables that are measured by sensors in a battery pack are cell voltages, pack current, and battery surface temperature [19], and they are all important for estimating battery SoC. All the cell voltages are measured and used for estimating the SoC and SoH. The temperatures at battery surfaces are measured through an analog-to-digital (A2D) circuit. Finally, the current is used to ensure that the battery is safe and to observe and log abuse conditions.

2.1.2 SoC Estimation

Battery SoC changes through the passage of current. This can be from both charging and discharging the cell through the external circuitry, or due to internal self-discharge effects [19]. The SoC is important because it does balance, energy, and power computations. SoC can vary between 0% to 100%, where 0% means an empty battery and 100% is the battery is fully-charged. There currently exists no simple sensor technology for direct SoC measurement and it thus requires to be inferred from the aforementioned measurable variables. Accurate SoC estimation benefits the battery and usually, it can extend the battery lifetime. This is because if a battery experiences overcharge or overdischarge, it can cause permanent damage to the battery that can reduce its lifespan and efficiency.

Another benefit of accurate SoC estimation is that if an accurate SoC estimation is not available, a lot of consideration has to be given to a battery pack in order to avoid overcharge and overdischarge [19]. This can happen if a poor estimate is trusted. If it is known that the SoC estimation is good and what errors it usually has, then it can be used very determinedly and to its full potential/capacity.

In summary, a good SoC estimation needs to be consistent and trustworthy for all driving profiles in order to enhance the power system and its reliability [19]. Accurate and trustworthy SoC estimators allow the battery to be used to its fullest which means that there is no need to spend time over-engineering the battery pack. This leads to smaller and lighter batteries which also reduces costs. When the system is more reliable, it will lead to less damaged/weak batteries which would have otherwise gone to warranty and then be wasted.

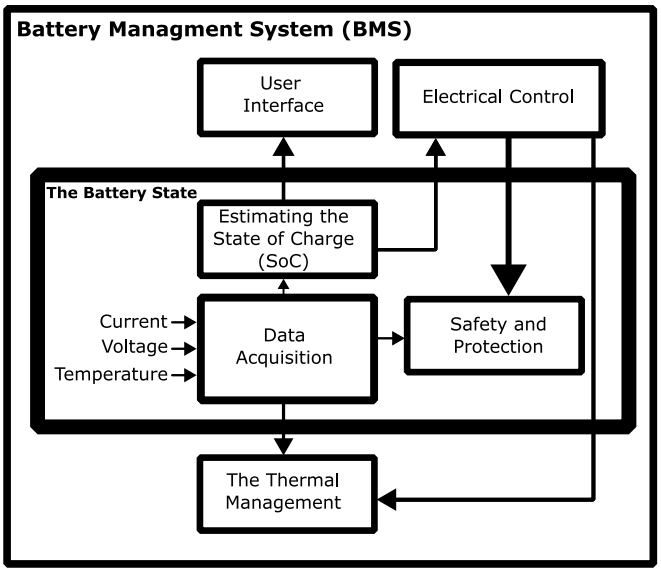


Figure 2.2: Schematic diagram of a typical battery management system.

2.2 Machine Learning

Goodfellow et al. [20] and Mehlig [21] discuss how neural networks work in general and how it is related to machine learning. A neural network is a mathematical model which uses data x (usually input) and y (usually output) to create a function $y = f(x, \theta)$, where θ represents the model parameters. The learning process is the identification of the parameters θ by using data of x as input and to approximate the target function $y = f^*(x)$. For example, a target function can predict the weather of tomorrow based on today's weather. Features, which are in every element in x, contain useful information buried in the data. For example, in a car, the features can be velocity, acceleration, and position.

For training a model, the input x_i and target $y = f^*(x_i)$ is used. Training is to minimize a loss function $J(\theta)$ based on the model's parameters and the training set $\mathbb{X} = (x_i, y_i)$. The loss function is used to calculate the model f and uses parameters θ that estimate f^* for our $\mathbb{X}$, which is the given set. The main loss function for regression is the mean-squared-error loss (MSE), given by

$$J(\theta) = \frac{1}{N}\sum_i (y_i - \hat{y}_i)^2 \tag{2.1}$$

where $\hat{y}_i$ is the predicted value of y (the target function) and N is the number of training examples. The loss function can be minimized using an optimizer based on a certain optimization algorithm such as the gradient descent method. For the optimizer, θ successively steps in the opposite direction from the gradient of the loss function. Here, the parameters are updated according to

$$\theta \leftarrow \theta - \eta(\nabla_\theta J(\theta)) \tag{2.2}$$

where $\nabla_\theta J(\theta)$ is the gradient of the loss function and η is the learning rate where $0 < \eta \ll 1$. When the parameters have been updated for all the training sets, one epoch has been finished and usually, the training needs many epochs before reaching a convergence.

The main goal of the training is to reach a set of parameters θ that does not overfit or underfit the dataset. Overfitting is the condition when the model has well-learned the training set, but it cannot get the same performance on other inputs, leading to a lack of extrapolability and generalization. Underfitting is the situation when the model has not understood/learned enough, and it can be improved by, for example, increasing the number of epochs.

A validation/test set is used to evaluate the trained model. The validation and test sets are very alike but have independent examples and data that are not used during the training. Tracking the training and validation loss while training a network, gives information on how well it is going. If the training and validation losses keep decreasing, the model is underfitted and has the potential to be improved further. On the other hand, if the training loss decreases while validation does not, the model is overfitted and needs further adjustment on the training parameters.

A gradient descent algorithm designed for big datasets is called batch gradient descent. For a big dataset, it takes extensive time to train the model and demand huge computational resource. Stochastic gradient descent (SGD) can be used to reduce the computational burden and it samples the training examples one at a time and performs the gradient descent for each sample. In the mini-batch stochastic gradient descent, samples for mini-batches are used instead of all data and the parameters are updated in each iteration.

The Adam optimizer [22] is similar to the SGD, while it uses an adaptive learning rate for all θ during the training. This can lead to improved results for some datasets. AdamW [23] is a modified Adam optimizer that penalizes θ when it becomes too large, and it prevents the problems such as overfitting to some degree and outperforms the Adam optimizer in general.

2.2.1 Machine Learning and Deep Learning

The original inspiration for a neural network comes from how the neurons work in the human brain [20]. The simplest neural network is the fully connected neural network (FCNN), also known as the multilayer perceptron (MLP). An FCNN has an input layer, one of several hidden layers, and an output layer. Here, a layer refers to a vector that has elements with affine transformations based on the elements in the previous layer, and it can have an activation function. An FCNN with a single hidden layer is shown in Figure 2.3, where every element in the hidden layer and the output layer is connected to all elements in the corresponding previous layer. The input layer $\mathbf{x} = [x_1, x_2, \cdots, x_{n_x}]^T \in \mathbb{R}^{n_x}$ accepts the input data. The input data are then transformed into the data in the hidden layer, denoted by a vector $\mathbf{a} = [a_1, a_2, \cdots, a_{n_h}]^T \in \mathbb{R}^{n_h}$. Finally, the data are transformed to the output $\mathbf{y} = [y_1, y_2, \cdots, y_{n_y}]^T \in \mathbb{R}^{n_y}$. The number of the input and output sizes n_x and n_y are dependent on the respective input and output as well as the target function, which the network used to estimate the hidden layer. Each element in the vector is called a node. In the transformation processes $x \to a \to y$, different weights, and biases are learnable parameters θ of the model.

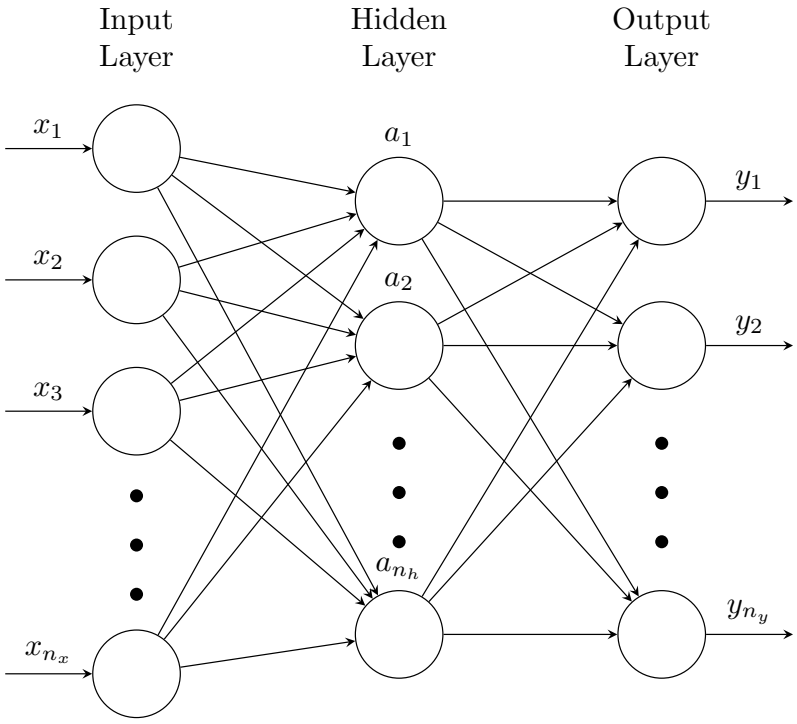


Figure 2.3: MLP with one hidden layer. The input has n_x nodes of x_k, the hidden layer has n_h nodes of a_j and the output has n_y nodes of y_i. The connections between the nodes are the weights $w^{[1]}_{j,k}$ and $w^{[2]}_{i,j}$. The weights calculate the affine transformations that are between the layers. Note that the bias and activation functions are not shown.

The element in the hidden layer a is calculated by

$$a_j = g^{[1]}(\sum_{k=1}^{n_x} w_{j,k}^{[1]} x_k + b_j^{[1]}) \quad \forall j \in \{1, 2, \cdots, n_h\} \tag{2.3}$$

where $w_{j,k}^{[1]}$, $b_j^{[1]}$, $g^{[1]}(\cdot)$ are the corresponding weight, bias, and activation function, respectively.

Similarly, the output is obtained by

$$y_i = g^{[2]}(\sum_{j=1}^{n_h} w_{i,j}^{[2]} a_j + b_i^{[2]}) \quad \forall i \in \{1, 2, \cdots, n_y\} \tag{2.4}$$

where $w_{i,j}^{[2]}$, $b_i^{[2]}$, $g^{[2]}(\cdot)$ are the corresponding weight, bias, and activation function, respectively. The parameters θ of this single-layer FCNN thus consists of all $w_{j,k}^{[1]}$, $b_j^{[1]}$, $w_{i,j}^{[2]}$ and $b_i^{[2]}$.

An MLP can have more than one hidden layer and hidden layers can have different numbers of nodes and activation functions. An MLP that has more than one hidden layer is called a deep MLP, and the respective method is called deep learning. In order to improve training speed and performance for the MLP, it is recommended to normalize the data so the data being processed have zero mean and unit variance [24].

2.2.1.1 Activation Functions and Skip Connection

The activation functions $g(\cdot)$ in the MLP are nonlinear functions and they can significantly affect the performance of the MLP [20]. One of the most popular activation functions that are widely used is the Rectified Linear Unit (ReLU) which is described by

$$g(z) = \max(0, z) \tag{2.5}$$

where z denotes the input of the activation function. The ReLU is illustrated in Figure 2.4.

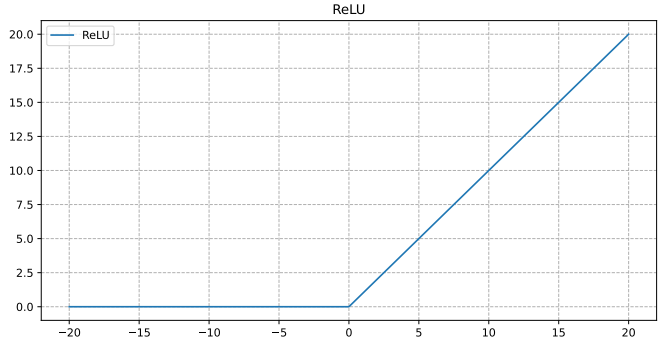


Figure 2.4: Illustration of ReLU.

Leaky ReLU is a variant of ReLU to improve the capability for handling negative inputs. As shown in Figure 2.5, the difference to the standard ReLU is that for the negative input, the Leaky ReLU has a small, nonzero, constant gradient α.

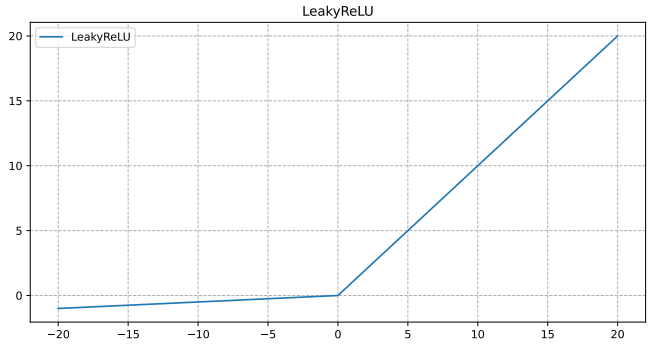


Figure 2.5: Illustration of Leaky ReLU.

The hyperbolic tangent activation function (tanh function) is also a popular activation function to use. It converts a real value in the input into an output between -1 to 1. Small or negative inputs give output closer to -1 while large inputs give output closer to 1. The tanh activation function is illustrated in Figure 2.6 and described by

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.6}$$

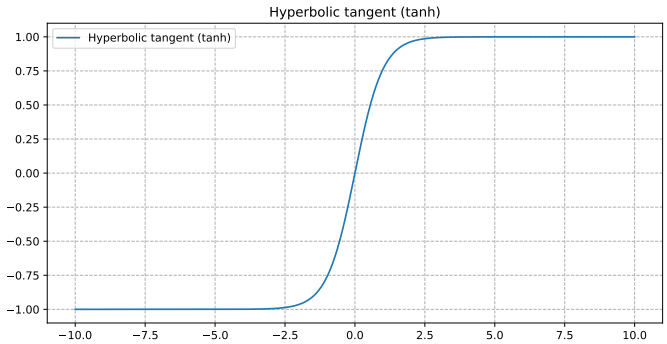


Figure 2.6: Illustration of tanh function.

Instead of using an exact activation function such as the ReLu, Leaky Relu, and tanh function as described above, a popular technique widely adopted is using residual connections or skip connections. As illustrated in Figure 2.7, in this method, the input of the neuron z is usually forwarded to the output and combined with the output $F(z)$ of the hidden layers, i.e.,

$$g(z) = F(z) + z \tag{2.7}$$

This method is computationally effective and usually gives higher precision than normal activation functions.

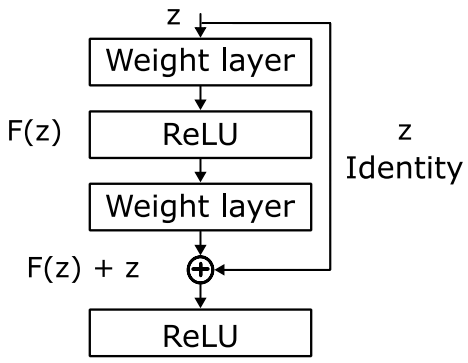


Figure 2.7: Illustration of skip connection.

When the output of a target has a continuous interval in a real number line, then a linear activation function is used as the output layer. This is used for models with linear regressive tasks and it is simply described by

$$g(z) = z \tag{2.8}$$

2.2.2 SoC Estimation With Machine Learning

For estimating battery SoC using machine learning, the main inputs can be the directly measured quantities such as battery voltage V, current I, and surface temperature T. However, in [12], it shows that using the averaged current and averaged voltage over a specific time horizon L as the inputs can effectively improve the SoC estimation performance. Having the average voltage and current is better than having antecedent values of current and voltage or only current, voltage, and temperature. This is because the number of weights for the first layer in the network increase in proportion to the number of extra inputs. If a network has ten neurons in the first hidden layer, then every extra input needs ten extra weights for an FCNN. Therefore, in order to obtain better computational efficiency, the averaged current I_{avg} and averaged voltage V_{avg} will be added as the inputs in the present investigation, as illustrated in Figure 2.8. At time instant indexed by t, they are defined by

$$I_{\text{avg}}(t) = \sum_{k=1}^{L} I(t-k) \tag{2.9}$$

$$V_{\text{avg}}(t) = \sum_{k=1}^{L} V(t-k) \tag{2.10}$$

Here, the numbers of nodes and layers, as well as the activation functions, can all affect the performance of the MLP, and they are to be determined during the design.

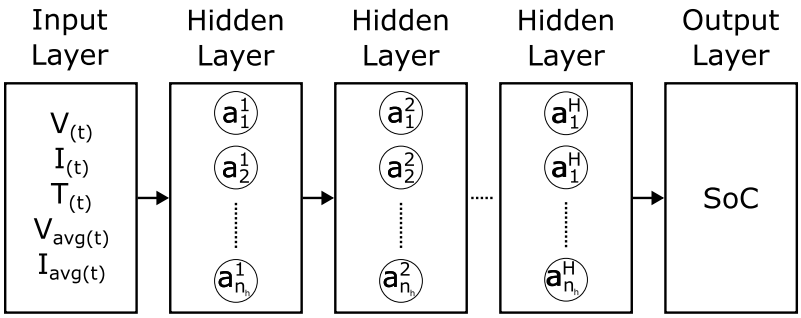


Figure 2.8: Deep MLP with multiple hidden layers for SoC estimation. The inputs are V, I, T, V_{avg}, and I_{avg}. It has H hidden layers, and the output is the SoC.

3. Methods

In this chapter, the main methods for the thesis are explained. The description of the dataset used in this work is given first, followed by an introduction of the available data and the data preprocessing procedure. Next, different ML and DL algorithms are presented. Finally, a framework for evaluating the performance of the ML/DL algorithm of SoC estimation is exhibited.

3.1 Data

A dataset for LG 18650HG2 lithium-ion batteries is used [25] and it is referred to as the LG data in the rest of the thesis. Compared to other public datasets of lithium-ion batteries, this dataset is very informative, and more details on how each experiment/testing was designed and carried out are available. The testing conditions are more comprehensive and can be better related to real-world scenarios, making it more suitable for the present study for this thesis.

3.1.1 Description of the LG Dataset

In the LG data, the following information is given. The tests were done at McMaster University in Canada by Phillip Kollmeyer. The battery is a new 3Ah LG HG2 cell with an 8 cu.ft. thermal chamber with a 75-A, 5-V Digatron firing circuits universal battery tester channel with voltage and current that has an accuracy of 0.1%.

The tests were done at different temperatures and the batteries were charged for every test with a 1C rate up to 4.2 V and 50 mA cut off. Eight different drive cycles are mixed and consist of UDDS, HWFET, LA92, and US06. The calculation for the drive cycle power profile is on an LG HG2 cell in a compact electric vehicle. The temperatures that were used for the data are 25 °C, 10 °C, 0 °C and −10 °C. The data are given in the format of .mat and .csv files. The data are identical for both formats and the .mat files were used in this work for data preprocessing in MATLAB. The following variables/columns are available in the dataset:

- `Time` (in seconds)
- `TimeStamp` (MM/DD/YYYY HH:MM:SS AM format)
- `Voltage` (cell terminal voltage, sense leads welded directly to battery terminal)
- `Current` (measured current in amperes)

- `Ah` (measured in amphere-hours, with Ah counter, typically reset after each charge, test, or drive cycle)
- `Wh` (measured watt-hours, with Wh counter, reset after each charge, test, or drive cycle)
- `Power` (measure power in watts)
- `Battery_Temp_degC` (battery case temperature, in the middle of battery, in degrees Celsius (°C) measured with an AD592 $+/-1$°C accuracy temperature sensor)

The columns that are of interest are `Voltage`, `Current`, `Battery_Temp_degC`, and `Ah` (for the SoC). The US06 data tested under temperatures of 25 °C, 10 °C, 0 °C, and −10 °C were preprocessed and used in this work. The goal is to use as little preprocessed data as possible to achieve good estimation accuracy in order to minimize the computational requirements needed. Evaluation will be done on how different ML algorithms can achieve this task with a reasonable but limited amount of data.

3.1.2 Data Preprocessing

Based on the recommendations from [25], the following steps are taken to preprocess the data in MATLAB:

1. Load datasets under drive cycles for temperature between −10 °C to 25 °C.
2. Using the `resample` function to resample the data to the resolution of 1 Hz.
3. Concatenate all the resampled data.
4. In the datasets, there are some missing data with blank spaces. Those missing data were addressed by looping over the datasets and removing the empty spaces with the function `cellfun`.
5. Calculate the SoC by dividing the Ah data by the cell nominal capacity of 3 Ah.
6. Calculate the averaged voltage and averaged current, the time horizon for averaging is selected to be $L = 500$ seconds and the obtained data are added to the dataset.
7. The preprocessed data of voltage, current, temperature, average current, and average voltage were stored in a dataset named X. The SoC data are stored in a dataset named Y, i.e.,

$$X = \{Current, Voltage, Temperature, Average\ Current, Average\ Voltage\}$$
$$Y = \{SoC\}$$

8. Normalize X and Y by

$$\text{Norm}_X = \frac{X - \text{mean}(X)}{\text{std}(X)}$$

$$\text{Norm}_Y = \frac{Y - \text{mean}(Y)}{\text{std}(Y)}$$

where the mean value and the standard deviation (std) of the data are calculated using MATLAB functions `mean` and `std`, respectively. With a slight abuse of notation, X and Y can either refer to the original or normalized data in the rest of the thesis.

9. X and Y are then combined in a variable Z for training, validation, and test. This is done by using the function `dividerand` in MATLAB which divides the data into these three sets using random indices. 80% of the data is divided into training, 10% for validation, and 10% for the test. When the dataset Z has been divided, the data for training, test, and validation are assigned to the respective X and Y.

3.2 Machine Learning

With preprocessed data available, the next goal is to choose simple while effective DL models for estimating the SoC. Indeed, there are a plethora of modern DL algorithms that are suitable for complex calculations with many inputs and outputs. However, for the present investigation, there are only five inputs and one output, and using an overly complex DL algorithm does not necessarily improve the performance but might introduce unnecessary computation. Hence, DL models such as ResNet need to be modified to match the simplicity of the low number of inputs and outputs of the dataset. Another motivation is that by simplifying complex DL models, the corresponding training time is expected to be reduced without sacrificing the SoC estimation accuracy.

3.2.1 Residual Network (ResNet)

Wang et al. [26] describe how a ResNet can be used on time series classification data for optimizing DL networks with many layers. In the ResNet, the skip connection is used in the activation functions, and a signal from early layers is added to later ones, as described in the Theory section.

Explaining it more in detail, as shown in Figure 2.7, the input signal of the first layer is added to the output of the third layer in a ResNet block. These blocks are called residual blocks. The skip connection makes the output different from the conventional activation functions where without a skip connection, the weights of the layers get multiplied by the weights and get a bias. The skip connections can avoid the problem of vanishing gradient that can occur in some neural networks. The skip connection also helps the model learn "identity functions" (see 2.7) which makes

higher layers perform as well as lower layers [26]. Usually, shallow networks can get a degraded performance after a while [26]. With the residual blocks, the network learns the "identity functions" and can get both good inputs and outputs. This also prevents the layers with many steps from degrading.

Therefore, it is expected that the ResNet should perform as well as and even better than a simple deep network [26]. There are, however, different depths that the ResNet can adopt. For the training, three different depths will be tested. As mentioned earlier, the model will be modified in order to simplify it which will lead to lower computational power and complexity. This is because the ResNet is usually used for high-dimensional data with many inputs and outputs whereas in this case, there are only five inputs and one output. Though, the main theory behind it is still there with the core being the system with skip connections. In the present work, the model was modified and shown in Figure 3.1 where the main residual block is enclosed with dotted lines.

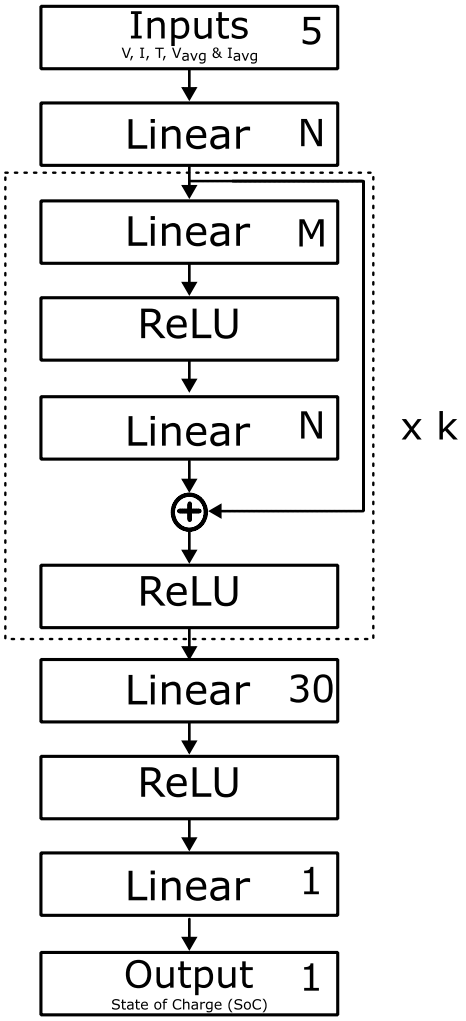


Figure 3.1: Structure of the ResNet based battery SoC estimation algorithm with k Skip Connections. N and M are the number of nodes.

In Figure 3.1, three different residual blocks ($k = 3$) are connected in series and the

number of nodes N and M in each block change. The initial N and M are obtained as the recommended values given in [26] and they were tuned to improve the training for the present investigation.

Table 3.1: Hyperparameters of the modified ResNet in Figure 3.1

Model	k	N	M
1	5	256	512
2		512	1024
3	10	256	512
4		512	1024
5	20	256	512
6		512	1024

3.2.2 Fully Connected Neural Network (FCNN/MLP)

The FCNN has some major advantages due to its structure agnostic [27]. In other words, there is no need to make any assumptions about the input regardless it is an image, video, time series, etc. There are a series of fully connected layers where every output is dependent on every input. The structure makes FCNN/MLP useful for many applications. However, there is a risk that they can perform worse than some networks designed for specific problems while FCNN has a general structure. For big datasets, training an FCNN can be quite slow due to the presence of many weights and biases. Clearly, for smaller datasets with few inputs and outputs, FCNNs can be an effective tool. An FCNN/MLP is created with a simple structure as shown in Figure 3.2.

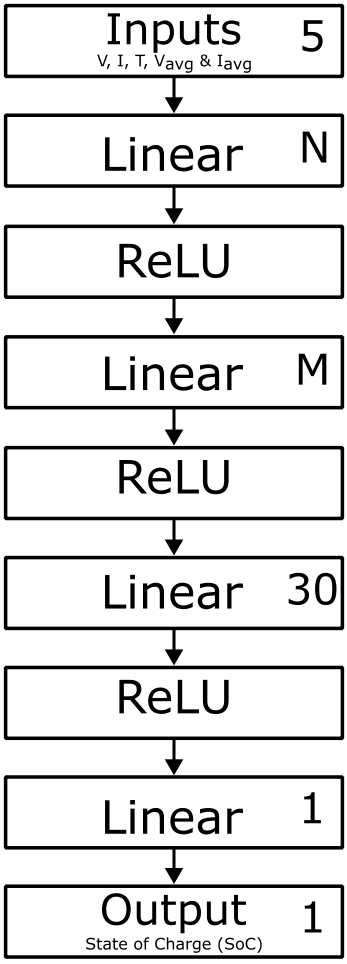


Figure 3.2: Structure of the FCNN-based battery SoC estimation algorithm. N and M are the number of nodes used for each activation function.

As can be observed, a different number of neurons is used and changed in order to identify the number of neurons that is the most suitable for the dataset. The neurons that were tested can be shown in Table 3.2. The numbers N and M of nodes recommended in [27] are modified to improve the training performance.

Table 3.2: Hyperparameters of the FCNN model Figure 3.2

Model	N	M
1	256	512
2	512	1024

3.2.3 Deep Feedforward Neural Network (FNN)

Vidal et al. show how the deep FNN is used for SoC estimation [28]. Compared to the other models, this model has been shown to work well in estimating the state of charge. The idea is that one of the machine learning models should use an already

established deep machine learning model for the state of charge and compare it with other ML models on the same dataset.

FNN is an MLP that has no recurrence. It only has a forward pass where data moves. It is used for optimization problems with many local minima. For the present dataset, the model structure in Figure 3.3 is adopted.

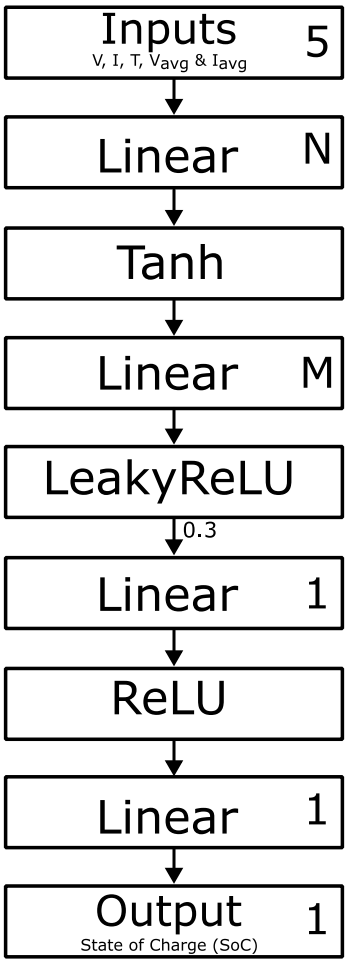


Figure 3.3: Structure of the FNN-based battery SoC estimation algorithm. N and M are the number of nodes used for each activation function. The constant gradient is 0.3 for the Leaky ReLU.

As can be observed, the FNN uses the tanh function and Leaky ReLU as the activation functions. The network model and the number of nodes for the FNN are based on the article [28] by Vidal et al. the number of nodes will be slightly modified compared to the original article to observe if the training will be improved.

Table 3.3: Hyperparameters of the FNN model in Figure 3.3

Model	N	M
1	120	60
2	1024	512

3.2.4 Stochastic Delta Rule (DropConnect)

Dropout is a simple and effective regularization technique that can be applied to deep learning models [29]. In this technique, neurons are randomly selected and ignored during the training phase. In other words, these neurons are "dropped out" and are not activated. They temporally are removed so there will not be any weight or updates on the neuron in the forward and backward passes.

During learning, the neuron weight becomes tuned for features that give specialization. Close neurons rely on this so-called specialization, and if it is taken too far, the model can become weak and specialized on the training data. This is called complex co-adaptations.

If a neuron is dropped from the network when it is training, there will be other neurons that have to represent the neurons that are dropped and make predictions for them instead. This leads to many independent internal representations that the network learns from. What happens is that the network will have fewer distributions from specific weights of some neurons. This leads to a network that can have generalization, which will make the training data have a lower chance to be overfitted.

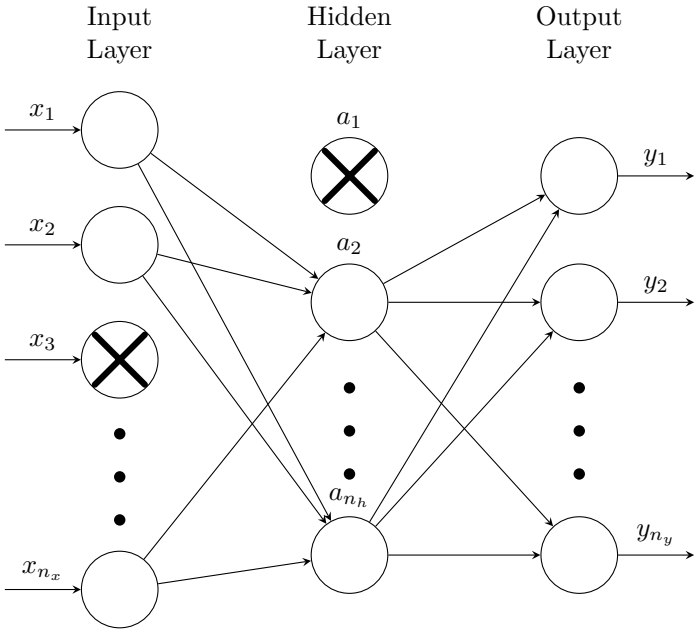


Figure 3.4: Schematic diagram of the dropout in an MLP.

For the model that is used, dropout is set to 20%, which means that for every five inputs, one will be excluded for each cycle. The learning rate (η) is recommended from the paper [29] to be set to quite high (around $0.1 - 0.01$). However, different learning rates will be tested in order to observe what works best for the specific model.

Using the information from the paper, the structure of the model applied here is shown in Figure 3.5.

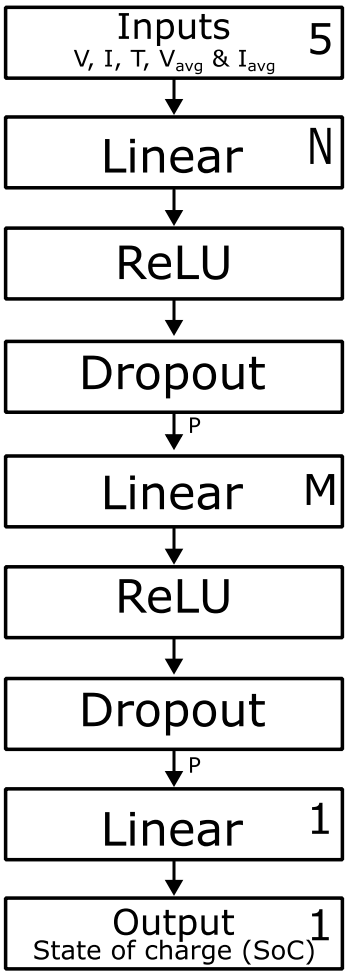


Figure 3.5: Structure of the DropConnect model for battery SoC estimation. N and M are the numbers of nodes used and the P is for setting the value of the dropout.

Table 3.4: Hyperparameters of the dropout model in Figure 3.5

Model	N	M	P
1	120	30	0.2
2	120	30	0.3
3	120	30	0.5
4	60	30	0.2
5	60	30	0.5

As can be observed, the recommended 20% dropout will be used but also 30% and 50% in order to observe how the training will change. The number of nodes is recommended to be set to quite small for the training.

3.2.5 Framework for Neural Network Training

For the training of the neural network, a lot of focus is on tuning hyperparameters and changing all the models slightly by changing the number of nodes to identify the best available model for every network. The main hyperparameter that is changed is the constant learning rate (η).

3.2.6 Experiment Settings

All the networks use the same optimizer, which is the AdamW but they have different constant learning rates. The learning rates are between 0.1 to 1×10^{-6}. The batch size was set to 300 and the number of epochs varies. To start, in order to identify how the models, perform and if they over-or underfit, the number of epochs is set to 300–600 for the training. When the best respective model has been identified, the number of epochs will increase in order to remove any underfitting of the data. The number of epochs is then set to 1200–2000.

3.2.7 Evaluation

The accuracy of the training and estimation is evaluated using MSE in order to identify the loss. However, the root-mean-square error (RMSE) and mean absolute error (MAE) are used to evaluate the model and see how well they perform. These calculations are used following previous research recommendations by [12]. The output of our y and $\hat{y}$ is the SoC for the measured and estimated value. The RMSE and MAE are defined by

$$RMSE = \sqrt{\frac{1}{m}\sum_{t=1}^{m}(SoC(t) - SoC^*(t))^2} \tag{3.1}$$

$$MAE = \frac{1}{m}\sum_{t=1}^{m}(|SoC(t) - SoC^*(t)|) \tag{3.2}$$

where $SoC(t)$ is the measured SoC and the $SoC^*(t)$ is the estimated SoC. The m is the number of data entries.

The SoC from the estimated and measured data from the best models will be shown from the training and the test in order to observe how well the models predict the state of charge using real numbers. Both the normalized and non-normalized RMSE and MAE will be presented. The whole process can be illustrated in Figure 3.6.

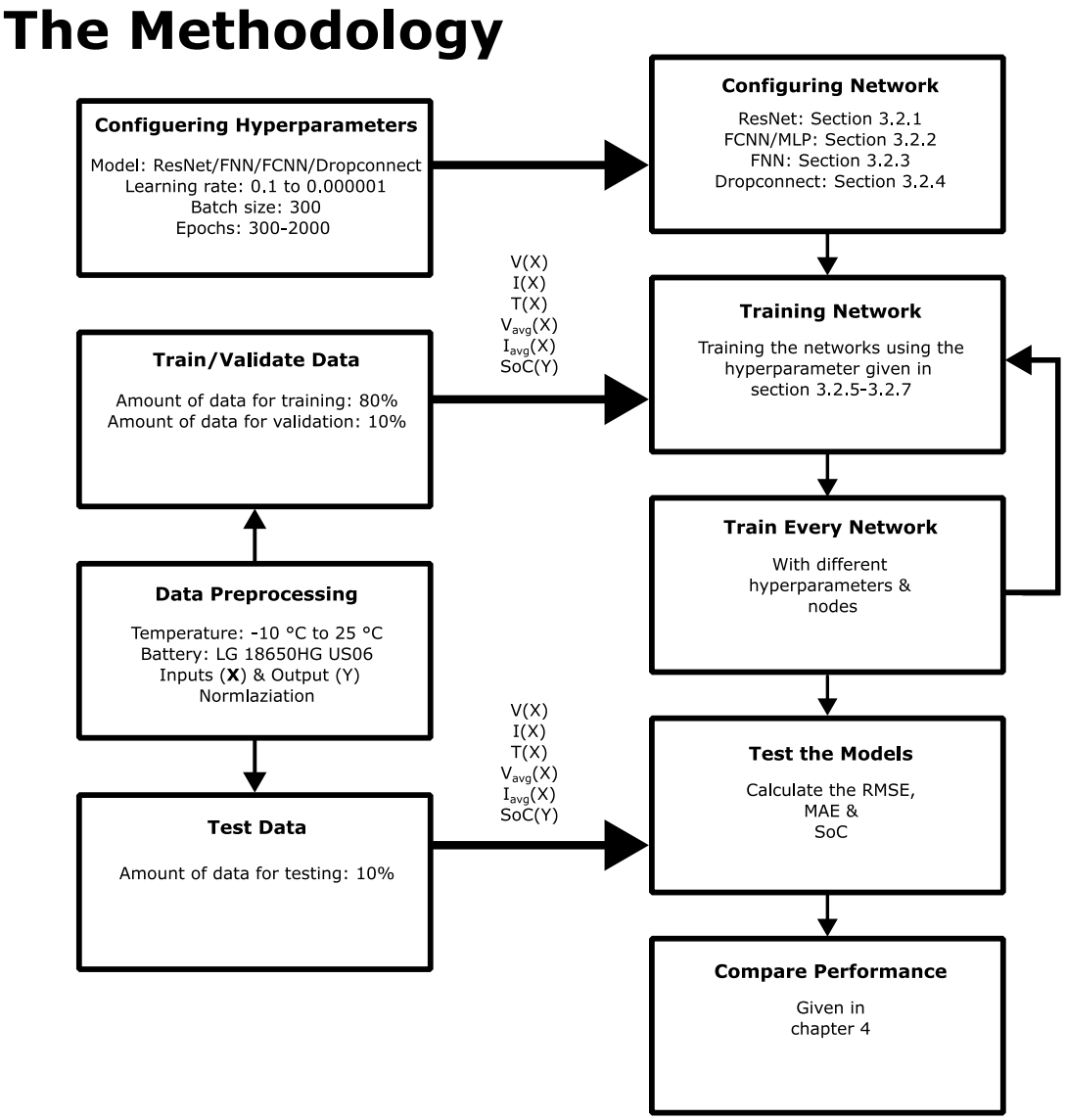


Figure 3.6: Overview of procedures of the proposed SoC estimation method on machine learning algorithms.

The data preprocessing was carried out using MATLAB 2019b and the ML was conducted in Python 3.8.12 with PyTorch [30] and TensorFlow [31] as the main libraries. All the simulations and computations were done on a computer with a GPU of Nvidia GeForce GTX 980 and a CPU of Intel Core i5-9600K with 16 GB of RAM. It should be mentioned that the GPU model used in this work is relatively old

and the performance is inferior compared to most high-performance GPUs available on the market. However, one of the objectives of this work is to see if an older GPU can effectively and efficiently run relatively simple DL models for battery SoC estimation.

4. Results & Discussion

In this chapter, the results using the algorithms described in Chapter 3 will be presented and discussed. First, the results for model training are presented. The best-trained models are selected by comparing the RMSE, MAE, and time for each model, and they are re-trained for longer time in order to reduce underfitting. Finally, six models are investigated based on the test data where the normalized/unnormalized MAE and RMSE of the algorithms will also be given. The estimated and measured SoCs will be compared finally.

4.1 Training Results

The configurations and performances of different models are shown in Tables 1 to 4. Although the ResNet has the most hyperparameters for model tuning, it is the network that has the highest potential to improve. The columns of the tables are described as follows. k is for the number of residual blocks in the ResNet. N and M are numbers of nodes in the model. P is the value of the dropout. The column "time" describes the number of hours and minutes it took for the training to finish. The RMSE and MAE show the highest and lowest values for the final ten epochs. All the results are obtained with batch size being 300.

Table 4.1: Configuration and performance of the ResNet (see Figure 3.1 for its structure).

k	N	M	η	Epochs	Time(h:m)	RMSE	MAE
5	256	512	1e-6	300	5:32	0.034-0.036	0.01-0.013
5	256	512	1e-5	1000	3:17	0.031-0.035	0.09-0.012
5	256	512	1e-4	1000	4:30	0.035-0.04	0.011-0.013
5	256	512	1e-3	300	0:46	0.052-0.056	0.02-0.025
5	512	1024	1e-6	300	3:43	0.043-0.044	0.02-0.022
5	512	1024	1e-4	300	1:23	0.032-0.042	0.01-0.015
5	512	1024	1e-3	300	0:54	0.048-0.056	0.02-0.025
10	256	512	1e-6	300	6:12	0.045-0.047	0.021-0.023
10	256	512	1e-4	300	4:46	0.036-0.041	0.011-0.017
10	256	512	1e-3	300	1:23	0.048-0.052	0.02-0.027
10	512	1024	1e-6	1000	7:53	0.034-0.037	0.011-0.013
10	512	1024	1e-5	1000	9:39	0.029-0.035	0.008-0.011
10	512	1024	1e-3	300	2:21	0.048-0.056	0.019-0.025
20	256	512	1e-6	300	9:43	0.045-0.048	0.018-0.022
20	256	512	1e-4	300	6:24	0.032-0.042	0.01-0.015
20	256	512	1e-3	300	2:04	0.048-0.059	0.017-0.026
20	512	1024	1e-6	300	11:21	0.042-0.045	0.015-0.017
20	512	1024	1e-4	1000	15:05	0.036-0.041	0.011-0.014

From Table 4.1, it can be observed that the ResNet with $k = 5$ requires the shorter time for training compared to other ResNets. The ResNet model seems to perform better with fewer nodes. Though, the RMSE and MAE have higher levels of fluctuations with more nodes.

As can be observed, the ResNet with $k = 10$ overall takes longer time for training than the ResNet with $k = 5$, but it is faster than the one with $k = 20$ and gets smaller MAE/RMSE with increased nodes.

The ResNet $k = 20$ takes longer time to train compared to the other ResNet models but it does achieve smaller MAE/RMSE. Furthermore, its MAE and RMSE have higher-levels of variance compared to the other ResNet models and it also takes longer time to run.

From Table 4.1, it can be seen that the ResNet models with $k = 5$ and $k = 10$ should be able to give good results for the test sets, but the ResNet with $k = 20$ should also be included in order to investigate if it can perform better in the test set than training.

Table 4.2: Configuration and performance of the FCNN (see Figure 3.2 for its structure).

N	M	η	Epochs	Time(h:m)	RMSE	MAE
256	512	1e-6	600	1:34	0.063-0.064	0.023-0.024
256	512	1e-5	600	1:51	0.054-0.056	0.021-0.023
256	512	1e-3	600	0:44	0.05-0.058	0.020-0.027
512	1024	1e-6	600	2:12	0.061-0.062	0.021-0.023
512	1024	1e-5	1200	2:56	0.042-0.045	0.018-0.021
512	1024	1e-3	600	1:17	0.049-0.053	0.020-0.022

From Table 4.2 for the FCNN, it can be observed that it is very computational efficient and gets relatively small MAE/RMSE. The selection of hyperparameters impact the RMSE/MAE of the FCNN more significantly than the ResNet models. A higher number of nodes leads to better results for the FCNN, although this is at the cost of increased training time.

Table 4.3: Configuration and performance of the FNN (see Figure 3.3 for its structure).

N	M	η	Epochs	Time(h:m)	RMSE	MAE
120	60	1e-4	1500	2:31	0.061-0.062	0.029-0.031
1024	512	1e-4	1200	1:34	0.058-0.059	0.026-0.031

As can be observed in Table 4.3, similarly to the FCNN, the FNN achieves very small MAE and RMSE in a relatively short time. The FNN performs better when the number of nodes is larger. The idea was to follow the model given in [28] and use that FNN as closely as possible without modifying it too much. This was done in order to observe and compare how well an already simple and proven DL model for SoC performs.

Table 4.4: Configuration and performance of the DropConnect neural network (see Figure 3.5 for its structure).

N	M	P	η	Epochs	Time(h:m)	RMSE	MAE
120	30	0.2	1e-3	1500	2:25	0.066-0.073	0.037-0.043
120	30	0.2	1e-2	1500	1:56	0.182-0.243	0.08-0.123
120	30	0.5	1e-3	1500	2:02	0.52-0.573	0.353-0.377
60	30	0.2	1e-3	1500	1:43	0.126-0.153	0.052-0.057
60	30	0.3	1e-3	1500	1:33	0.411-0.473	0.512-0.577

For the DropConnect model, it can be observed in Table 4.4 that when the dropout is larger the MAE/RMSE increases. Thus, the dropout should be kept at around 0.2.

Because the dropout drops a set number of nodes, it can also be observed that it completes its training faster than other models. The number of nodes in this model is lower than other models because the suggestion given in [29] recommended that the number of nodes should be relatively low (around 50-100).

For Tables 4.1 to 4.4, the figures of their training can be found in Appendix A.

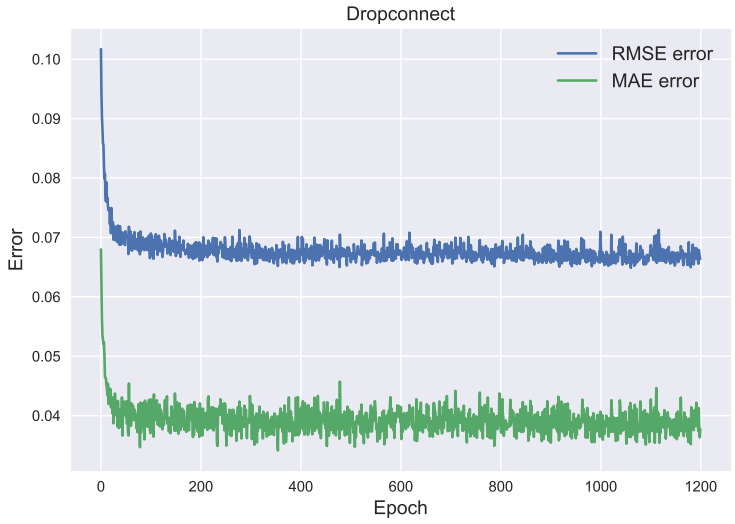
The next step is to select the best model from the respective network. The number of epochs will then be increased for those models that appear to be underfitted and then putting it on the test set. For the FCNN, the model that was chosen and could perform the best using Table 4.2 is the FCNN with $N = 512$ and $M = 1024$ with the learning rate $\eta = 1 \times 10^{-5}$. For the DropConnect model, from Table 4.4, the one that is chosen is the one with $N = 120$, $M = 30$ and $P = 0.2$ with the learning rate $\eta = 1 \times 10^{-4}$. From Table 4.3 for FNN, the model with the nodes $N = 512$ and $M = 1024$ with the learning rate $\eta = 1 \times 10^{-4}$ was chosen. Table 3.1 for the ResNet model $k = 1$, the model that is chosen is $N = 256$ and $M = 512$ with the learning rate $\eta = 1 \times 10^{-4}$. For the ResNet model $k = 2$ the model that is chosen is $N = 512$ and $M = 1024$ with the learning rate $\eta = 1 \times 10^{-5}$. For the final ResNet model $k = 3$, the model that is chosen is $N = 256$ and $M = 512$ with the learning rate $\eta = 1 \times 10^{-4}$.

As can be observed, for almost all the networks, the models with the highest nodes seem to be the best fitting for training. This suggests that more training could have been done on the DropConnect network by, for example, doubling the number of nodes used for the training in order to observe if it could have gotten a lower MAE/RMSE loss. A low learning rate is beneficial for all the models and when the learning rate was set to the highest for respective model, the RMSE and MAE loss was also high.

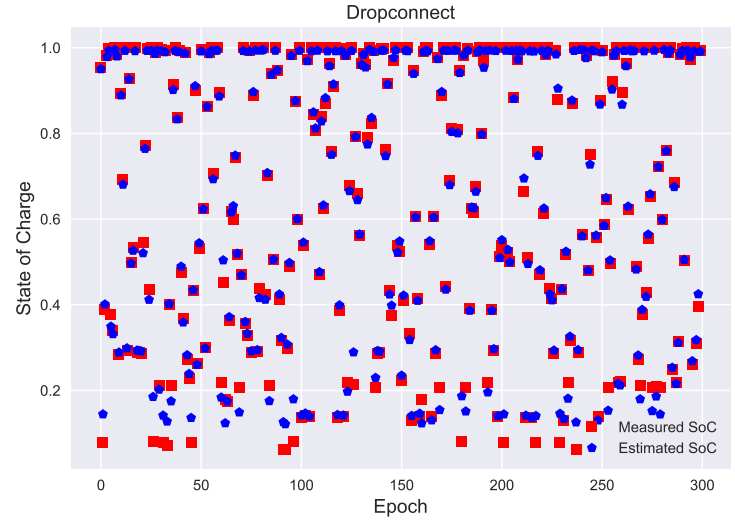
Choosing these models and then increasing the number of epochs to avoid underfitting, results in Table 4.5 are obtained:

Table 4.5: Variables/parameters for the final models.

Net.	k	N	M	P	η	Epochs	Time(h:m)	RMSE	MAE
FCNN	–	512	1024	–	1e-5	1200	2:56	0.043-0.045	0.017-0.021
Drop.	–	120	30	0.2	1e-4	1200	2:03	0.66-0.0.69	0.036-0.0.41
FNN	–	1024	512	–	1e-4	1200	2:17	0.057-0.058	0.027-0.030
ResN.	5	256	512	–	1e-5	1000	3:17	0.031-0.034	0.008-0.010
ResN.	10	512	1024	–	1e-5	1800	19:05	0.029-0.032	0.006-0.008
ResN.	20	256	512	–	1e-4	1000	8:46	0.034-0.045	0.011-0.015



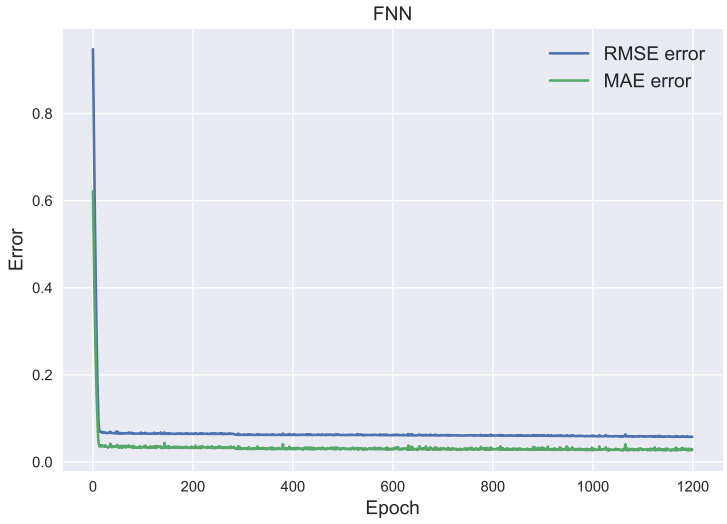
(a)



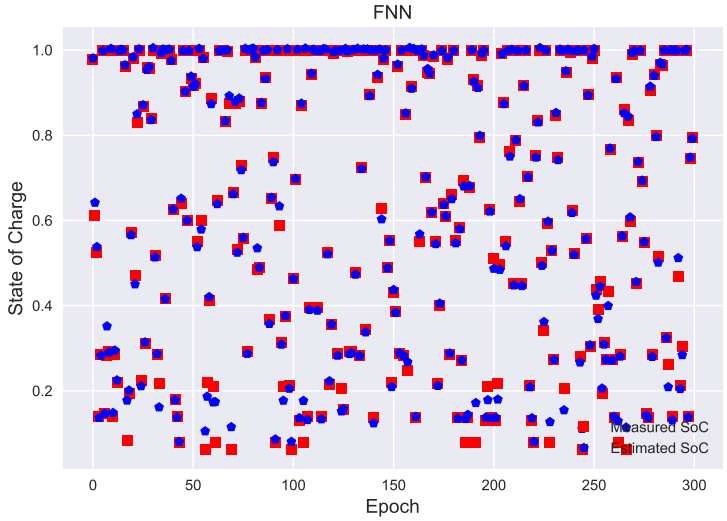
(b)

Figure 4.1: Performance of the DropConnect model for SoC estimation. (a) RMSE and MAE of the training of the DropConnect model. (b) The trained model for the DropConnect network (with random indices) for the measured (red) and estimated (blue) SoCs.

Figure 4.1a show that the DropConnect model reaches its lowest error around 200 epochs. In Figure 4.1b, it can be observed that the measured and estimated SoCs are not that close to each other, which corresponds well with the value of the error from Figure 4.1a.



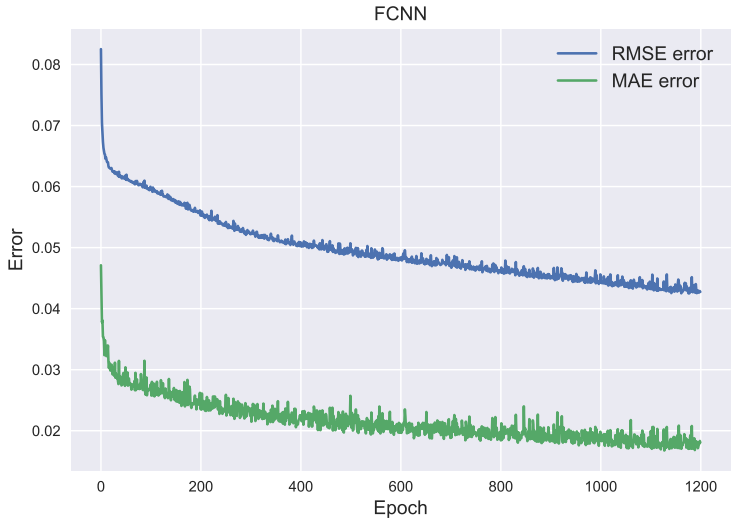
(a)



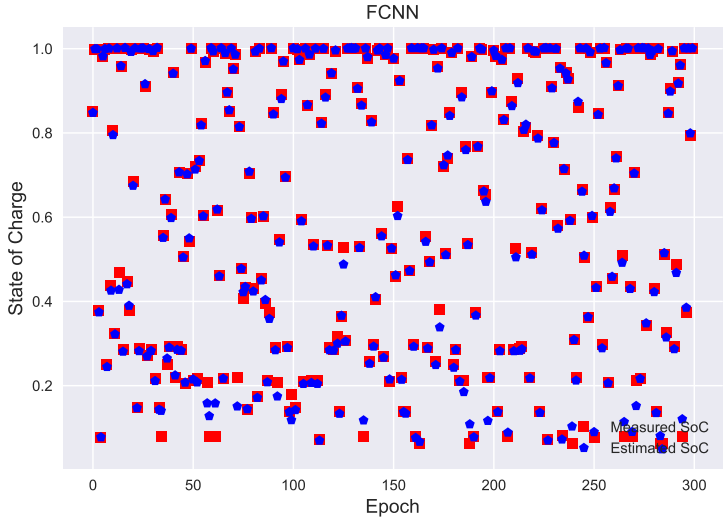
(b)

Figure 4.2: Performance of the FNN model for SoC estimation. (a) RMSE and MAE of the training of the FNN model. (b) The trained model for the FNN network (with random indices) for the measured (red) and estimated (blue) SoCs.

Figure 4.2a shows that the FNN model reaches its lowest error around 20 epochs which is much faster compared to Figure 4.1a. From Figure 4.1b, it can be observed that the measured and estimated SoCs are slightly better than Figure 4.1b. Figure 4.2b also corresponds well with the RMSE/MAE from Figure 4.2a.



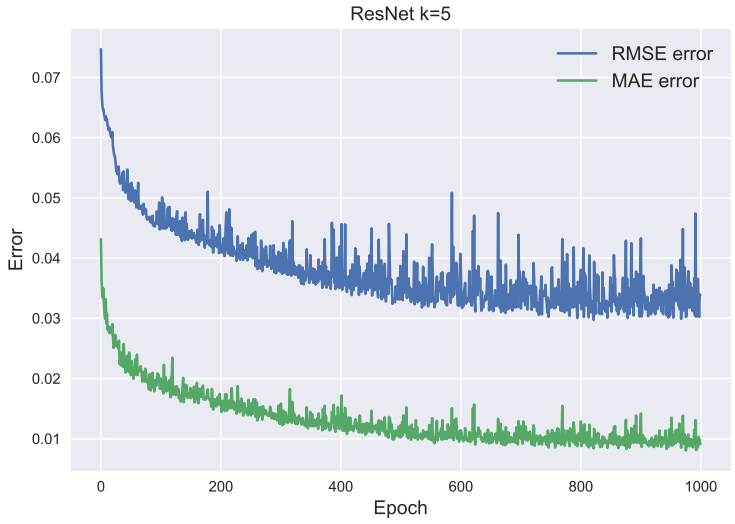
(a)



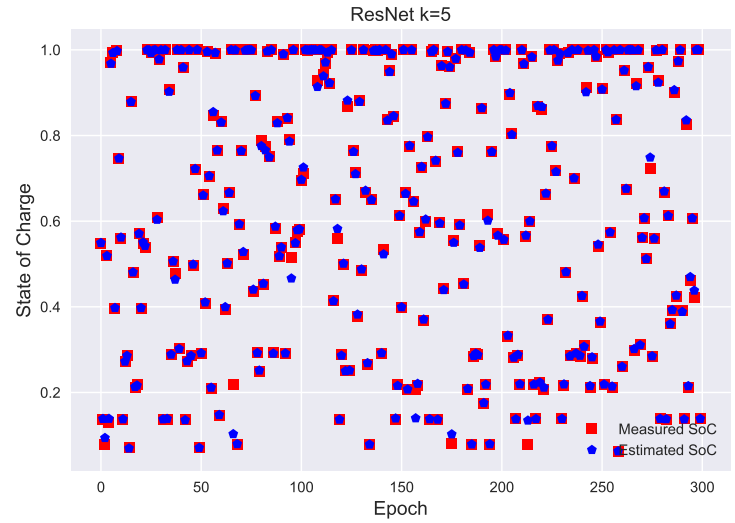
(b)

Figure 4.3: Performance of the FCNN model for SoC estimation. (a) RMSE and MAE of the training of the FCNN model. (b) The trained model for the FNN network (with random indices) for the measured (red) and estimated (blue) SoCs.

Figure 4.3a shows the results of the FCNN model. It can be observed that it looks underfitted compared to Figures 4.1a and 4.2a, which means it needed to be trained for more epochs to avoid underfitting. In 4.1b, it can be observed that the measured and estimated SoC are better than both Figures 4.1b and 4.2b. Figure 4.3b also corresponds well with the RMSE/MAE error from Figure 4.3a.



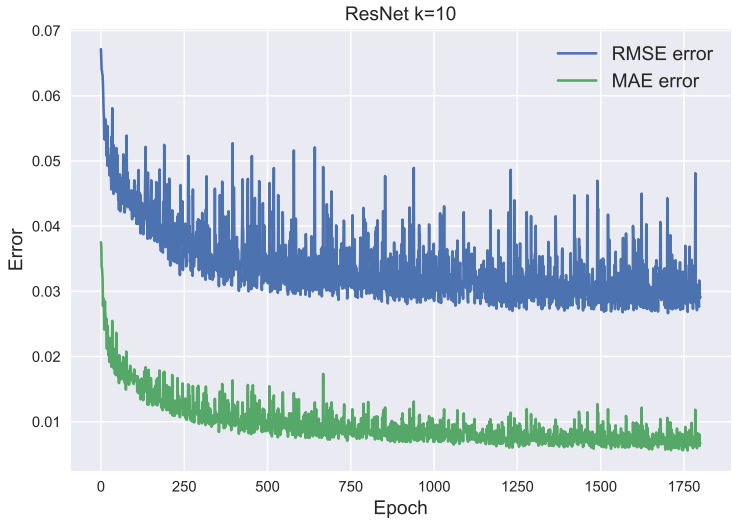
(a)



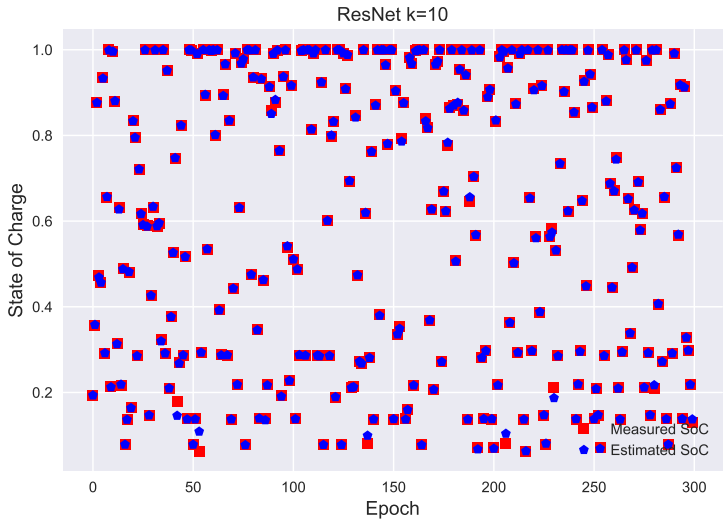
(b)

Figure 4.4: Performance of the ResNet $k = 5$ model for SoC estimation. (a) RMSE and MAE of the training of the ResNet $k = 5$ model. (b) The trained model for the ResNet $k = 5$ network (with random indices) for the measured (red) and estimated (blue) SoCs.

Figure 4.4a shows the results of the ResNet $k = 5$ model. It can be observed that like Figure 4.3a it could be underfitted and should therefore have been trained for more epochs to ensure that it was not underfitted. From Figure 4.4b, it can be observed that the measured and estimated SoC are better than Figures 4.1b, 4.2b and 4.3b. Figure 4.4b also corresponds well with the RMSE/MAE from Figure 4.4a where it performs better than Figures 4.1a, 4.2a and 4.3a.



(a)

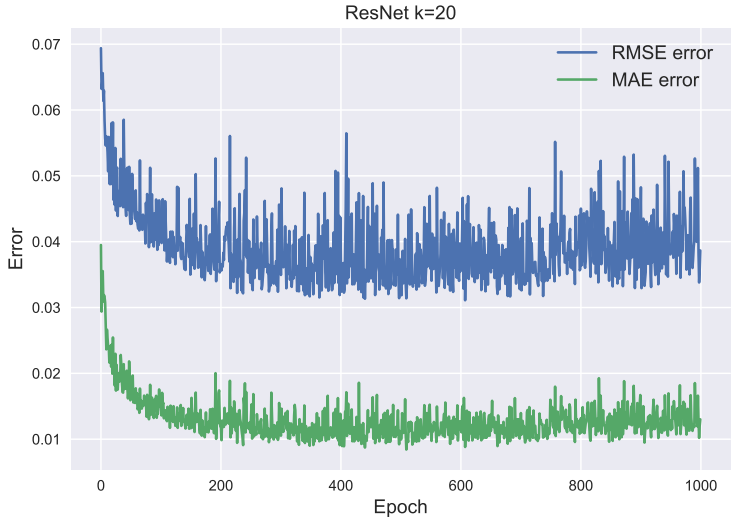


(b)

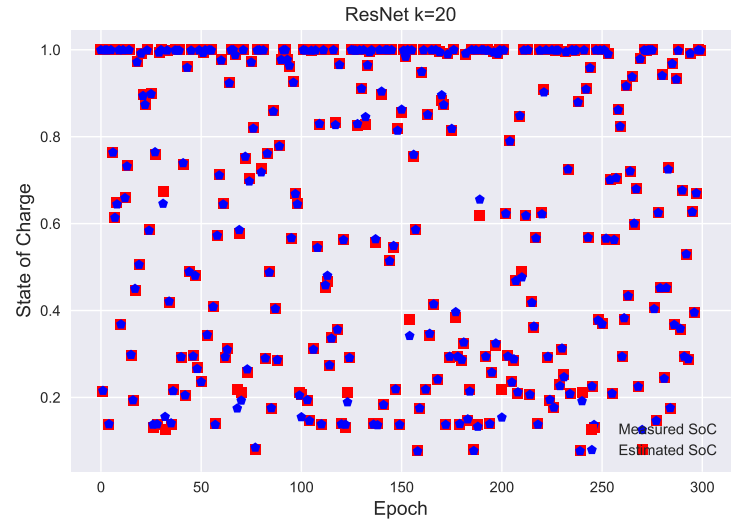
Figure 4.5: Performance of the ResNet $k = 5$ model for SoC estimation. (a) RMSE and MAE of the training of the ResNet $k = 10$ model. (b) The trained model for the ResNet($k = 10$ network (with random indices) for the measured (red) and estimated (blue) SoCs.

Figure 4.5a shows the results of the ResNet $k = 10$ model. It can be observed that like Figures 4.3a and 4.4a it could be slightly underfitted and should therefore been trained for more epochs. Though, Figure 4.5a has more fluctuations than Figures 4.1a, 4.2a, 4.3a and 4.4a due to the high learning rate and nodes. From Figure 4.5b, it can be observed that the measured and estimated SoC are better than Figures 4.1b, 4.2b and 4.3b but only slightly better than Figure 4.4a. Figure

4.5b also corresponds well with the RMSE/MAE from Figure 4.5a where it performs better than Figures 4.1a, 4.2a and 4.3a but is not that different from Figure 4.4a.



(a)



(b)

Figure 4.6: Performance of the ResNet $k = 5$ model for SoC estimation. (a) RMSE and MAE of the training of the ResNet $k = 5$ model. (b) The trained model for the ResNet $k = 5$ network (with random indices) for the measured (red) and estimated (blue) SoCs.

Finally, Figure 4.6a shows the results of the ResNet $k = 20$ model. It can be observed that unlike Figures 4.3a, 4.4a and 4.5a it overfits and should therefore have had a lower learning rate or/and nodes. Figure 4.6a has around the same level of fluctuation as Figure 4.5a and much more than Figures 4.1a, 4.2a, 4.3a and 4.4a due to the high

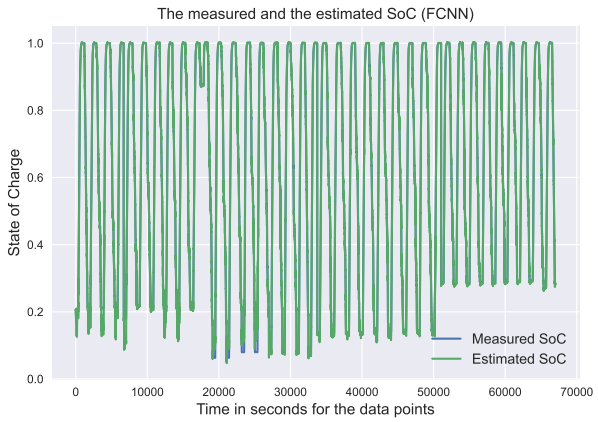
learning rate and nodes. From Figure 4.6b, it can be observed that the measured and estimated SoC are similar to Figures 4.4b and 4.5b but better than Figures 4.1b, 4.2b and 4.3b. Figure 4.6b corresponds well with the RMSE/MAE from Figure 4.6a but it performs worse than Figures 4.4a and 4.5a but better than Figures 4.1b, 4.2b and 4.3b.

4.2 Evaluating the Models

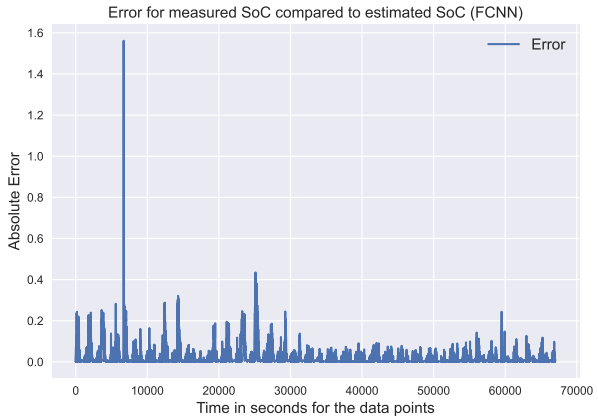
After the final training has been done on the six networks, the models are then put on the test set. The results in Table 4.6 summarize the performance of the respective model, where the columns MAE and RMSE refer to the un-normalized MAE and RMSE. Examples of the measured and estimated SoCs, as well as the SoC estimation error, are shown in Figures 4.7–4.12 for different models.

Table 4.6: Configuration and performance of the final models.

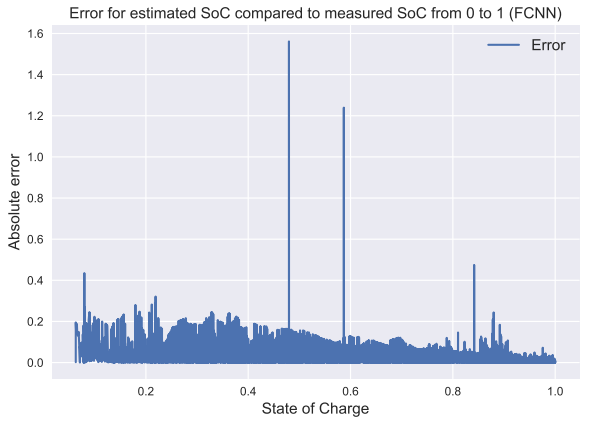
Network	k	MAE_{norm}	RMSE_{norm}	MAE	RMSE
FCNN	–	0.0179	0.0402	0.0059	0.0134
Dropcon.	–	0.0376	0.0648	0.0125	0.0215
FNN	–	0.0278	0.0556	0.0092	0.0185
ResNet	5	0.0088	0.0302	0.0029	0.0101
ResNet	10	0.0065	0.0252	0.0021	0.0084
ResNet	20	0.0128	0.0356	0.0042	0.0118



(a)

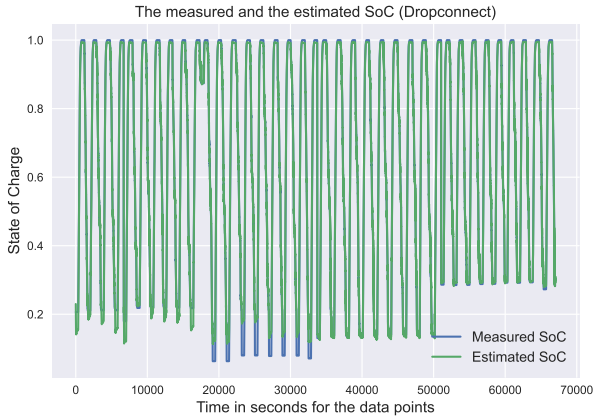


(b)

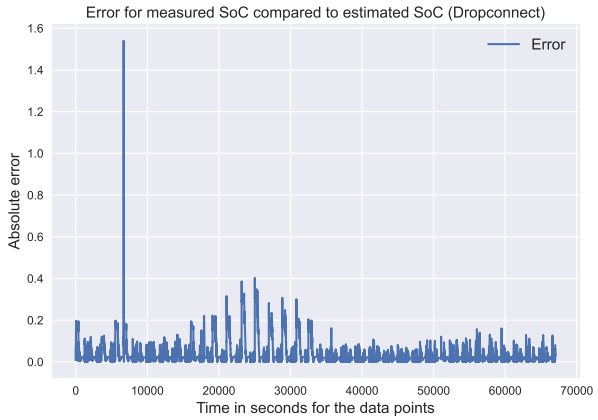


(c)

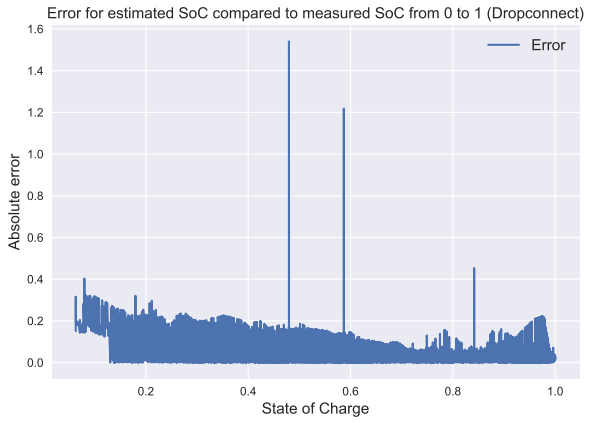
Figure 4.7: Performance of the FCNN model for SoC estimation. (a) Comparison of measured and estimated SoCs. (b) SoC estimation error. (c) Relationship between SoC estimation error and the SoC.



(a)

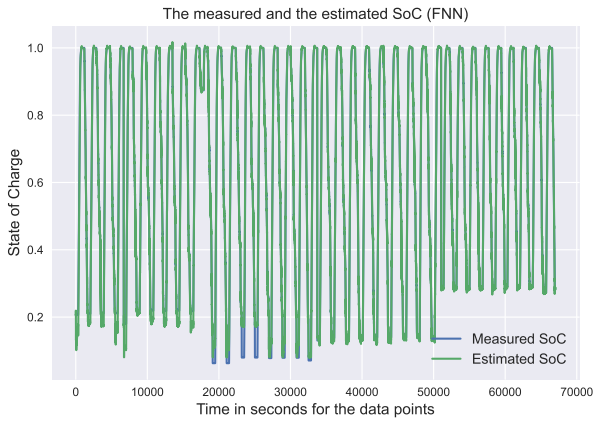


(b)

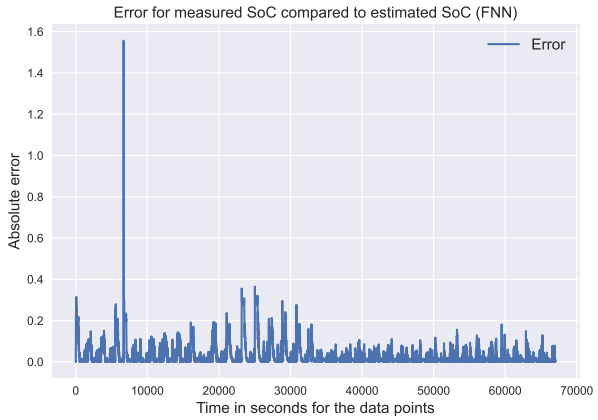


(c)

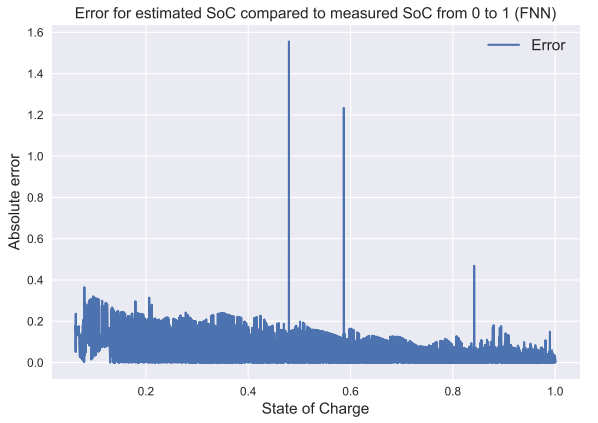
Figure 4.8: Performance of the DropConnect model for SoC estimation. (a) Comparison of measured and estimated SoCs. (b) SoC estimation error. (c) Relationship between SoC estimation error and the SoC.



(a)

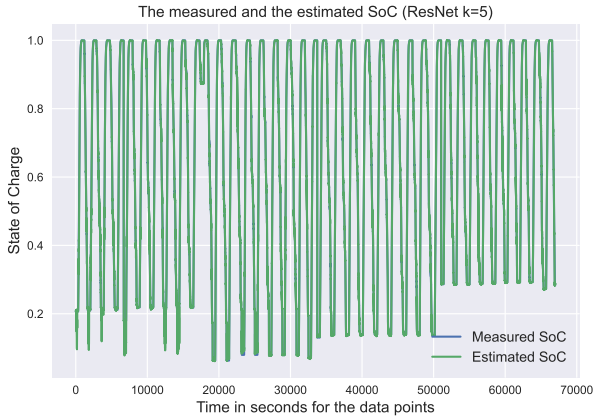


(b)

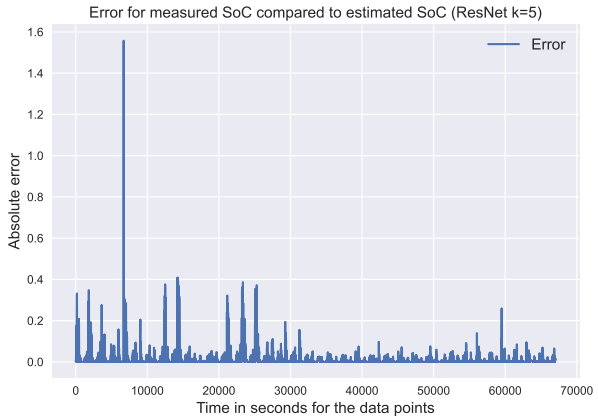


(c)

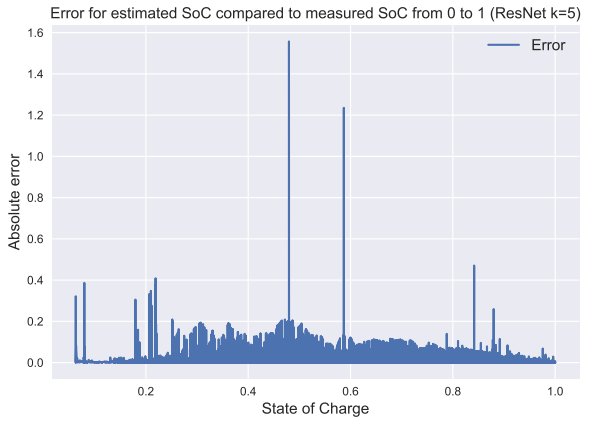
Figure 4.9: Performance of the FNN model for SoC estimation. (a) Comparison of measured and estimated SoCs. (b) SoC estimation error. (c) Relationship between SoC estimation error and the SoC.



(a)

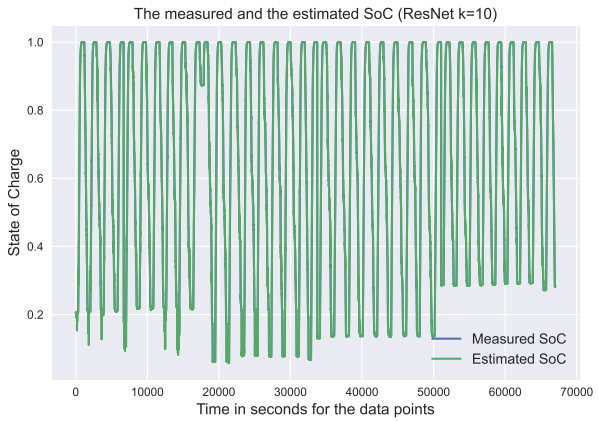


(b)

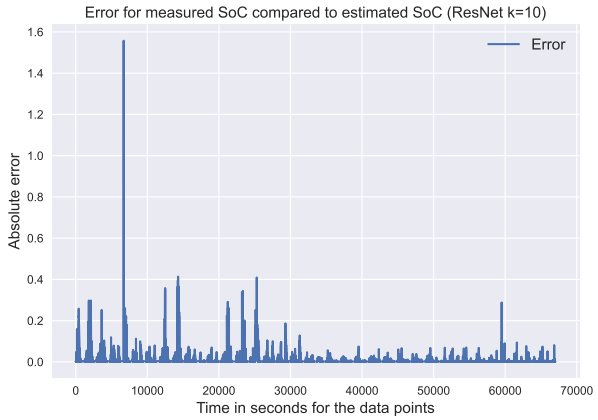


(c)

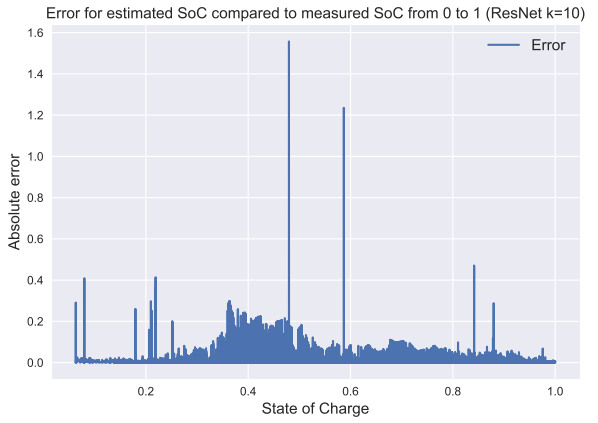
Figure 4.10: Performance of the ResNet $k = 5$ model for SoC estimation. (a) Comparison of measured and estimated SoCs. (b) SoC estimation error. (c) Relationship between SoC estimation error and the SoC.



(a)

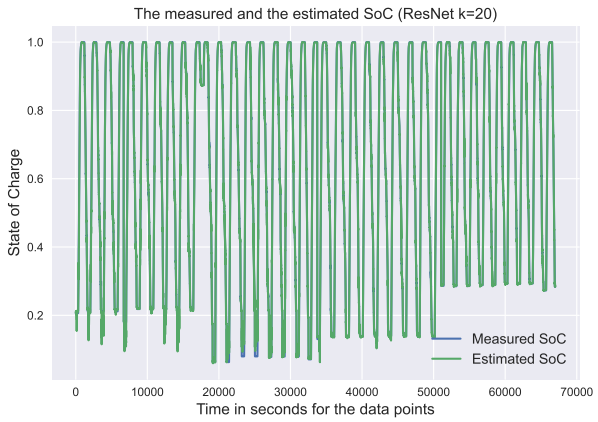


(b)

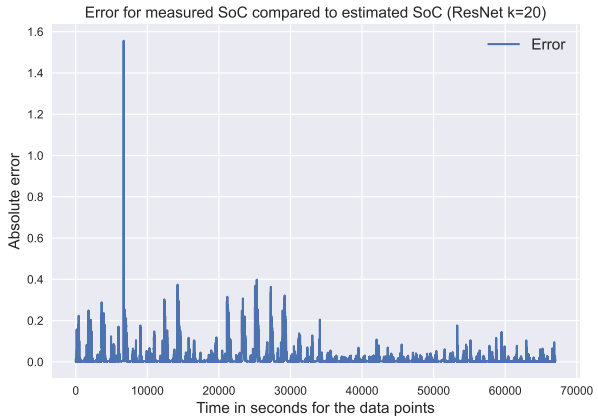


(c)

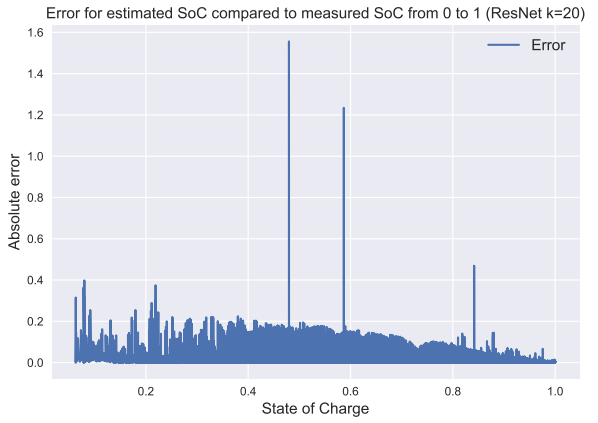
Figure 4.11: Performance of the ResNet $k = 5$ model for SoC estimation. (a) Comparison of measured and estimated SoCs. (b) SoC estimation error. (c) Relationship between SoC estimation error and the SoC.



(a)



(b)



(c)

Figure 4.12: Performance of the ResNet $k = 5$ model for SoC estimation. (a) Comparison of measured and estimated SoCs. (b) SoC estimation error. (c) Relationship between SoC estimation error and the SoC.

It can be observed from Table 4.6 that the ResNet models generally outperform the other models in terms of the calculated MAE/RMSE. However, as seen in Figures 4.7c, 4.8c, 4.9c, 4.10c, 4.11c and 4.12c, all the models have a high peak in the estimation error when the SoC is around 0.5. This is possibly due to a wrong or a sudden different measurement from the dataset but not the models themselves because the peak can be observed in all cases.

Within all non-ResNet models, the FCNN shown in Figure 4.7 is the best. The runtime for the FCNN is around 3 hours. This demonstrates that accurate SoC estimation can be trained without having complex DL algorithms that take a long time to run. From the estimated and measured SoC, the estimated accuracy struggles at a low SoC but is improved when the SoC is high, as can be observed in Figure 4.7.

It would be interesting to train the FCNN for more epochs because it is underfitted during the training and to observe if it could have reached the same performance as the ResNet models.

The DropConnect model seems to perform worst compared to all the other models in terms of the RMSE/MAE as shown in Table 4.6. Similar to the FCNN, the DropConnect model also struggle with estimating the SoC at low SoC levels and it is much more accurate at higher SoCs, as seen in Figure 4.8. Probably, the main reason is that when the number of nodes are low, removing nodes leads to worse training performance. This can be observed when the dropout was set above 0.2 (see Table 4.4), then the loss was much higher. However, the benefit of using the DropConnect is that training the network takes only 2 hours, which is the fastest among all the models (see Table 4.5).

For the DropConnect, it would be interesting to test how well it would perform if it is configured with more hidden layers and if better performance can be achieved with a dropout higher than 0.2.

In addition, the FNN performed worse compared to other models in terms of RMSE/MAE, as shown in Table 4.6. Nevertheless, the training time, like the DropConnect (see Table 4.5) was also around 2 hours which is faster than the other non-dropconnect models. Though, as shown in Figures 4.7c and 4.8c, the FNN has a larger error when the SoC is lower, see Figure 4.9c.

It would be interesting to change the configuration of the FNN model by, for example, switching from the tanh function to ReLU or changing the constant gradient (α) for the LeakyReLU to investigate if lower RMSE/MAE can be obtained. It would also be interesting to add more nodes and increase/decrease the learning rate in order to observe how much time it would take and if the RMSE/MAE becomes smaller.

The ResNet with $k = 5$ seems to be the best performer compared to all other models in terms of RMSE, MAE, and time (see Tables 4.5 and 4.6). The runtime for the training is around 3 hours and it gives better results than the FCNN, as can be seen

in Table 4.6 and Figure 4.9c. From Figure 4.10, compared to the other non-ResNet models (FNN/DropConnect/FCNN), the ResNet with $k = 5$ has a smaller absolute error at very low and very high SoC levels. Though, it has some large errors when the SoC is around 0.5 where some high spikes can be observed.

Similarly, Table 4.6, shows that the ResNet model with $k = 10$ performs better than other models in terms of the MAE and RMSE. However, the only downside of this model is that it takes around 19 hours for model training, as shown in Table 4.5. It shows that the RMSE/MAE can be reduced at the cost of time and numbers of residual blocks, although whether it is worth doing so depends on system requirements. From Figure 4.11, it can be seen that the ResNet model with $k = 10$ does well in estimating the SoC, but it also has the same issues as the case $k = 10$ with high spikes at different SoCs (see Figure 4.10).

It is intuitive to obtain better accuracy by further increasing the number of residual blocks, but surprisingly and unfortunately, from Table 4.6, the ResNet model with $k = 20$ performs worse than both the ResNets with $k = 5$ and $k = 10$. This suggests that adding more residual blocks does not always improve the model's accuracy. In addition, the ResNet with $k = 5$ managed to obtain smaller RMSE/MAE in around 3 hours than what ResNet $k = 20$ did which trained for 9 hours with the same number of epochs (1000). As seen in Figure 4.12, there also are spikes at the very low and very high SoCs. Nevertheless, even though it performs worse compared to the other ResNet models, the performance of the ResNet $k = 20$ is superior to the non-ResNet models, as can be observed in Table 4.6.

In short, the ResNet models are found to be superior in estimating the SoC in terms of accuracy, while the non-ResNet models investigated here are more computationally efficient. Comparing the obtained RMSE and MAE with those obtained in other research as mentioned in the introduction, the models obtained in the present work are all in the same order of magnitude in terms of accuracy and some models can achieve even smaller RMSE and MAE than those studies. As mentioned in the discussion, the best ResNet model in terms of both accuracy and training time is the one with $k = 5$ while the ResNet with $k = 10$ has the best performance.

5. Conclusion

The goal of this thesis was mainly to find answers to the research questions raised in the introduction section for DL-based battery SoC estimation. The answers to the questions are summarized as follows.

- [Q1] What advantages/disadvantages does each machine learning model have for estimating the SoC?

  Answer: The advantage of each machine learning model is that they all calculate the SoC well with the ResNet models being the best in terms of performance while the other models have an advantage in time. However, general disadvantages for each model are that it takes time to identify the correct hyperparameters and tune the models, so they become as efficient as possible. Though, when the parameters for the network have been identified then most of the work is done and only training is needed.

- [Q2] Which machine learning model is more suitable for estimating the SoC?

  Answer: From the results, all the ResNet models are quite superior compared to the other models. However, the only downside is that it mostly takes significantly more time to run the ResNet model compared to the non-ResNet models. The ResNet $k = 5$ has shown both in time and performance to be the most suitable in estimating the SoC compared to all the other models.

- [Q3] Would it be possible that a simple deep machine learning method can outperform a more complex/deeper machine learning method when estimating the SoC?

  Answer: The FCNN was one of the simple models that performed the best compared to the non-ResNet models and it is closest to the ResNet models. However, it could not quite reach the same performance as the ResNet models, but it was faster than them. Nevertheless, compared to research that uses complex DL models, the ResNet models obtained in the present work are all in the same order of magnitude in terms of accuracy and some models can achieve smaller RMSE and MAE than those studies.

5.1 Suggestions for Future Work

In the future, the following studies would be interesting to further research.

- Change up the models so they are more complex and also test the "original"

ResNet model with a powerful GPU in order to observe how it would perform in estimating the SoC. It would be interesting to add/remove the number of residual blocks to see if the results are similar or close to the pattern that the ResNet models in the thesis have.

- Use machine learning models based on transformers or other methods like genetic algorithms with a powerful GPU.

- Try all the trained models on a different dataset like for example the NASA or Panasonic dataset. If the performance would be almost or better than the LG dataset, then that would mean that it is highly likely that this would be an efficient tool to use in measuring the SoC in vehicles.

- Compare the estimated SoC with other non-machine learning methods like the Kalman filter on the same dataset in order to observe what advantages and disadvantages both have in terms of performance, time, and costs.

References

[1] I. B. Espedal, A. Jinasena, O. S. Burheim, and J. J. Lamb, "Current trends for state-of-charge (SoC) estimation in lithium-ion battery electric vehicles," *Energies*, vol. 14, no. 11, 2021. DOI: 10.3390/en14113284.

[2] R. Morello, R. Di Rienzo, R. Roncella, R. Saletti, and F. Baronti, "Hardware-in-the-loop platform for assessing battery state estimators in electric vehicles," *IEEE Access*, vol. 6, pp. 68 210–68 220, 2018.

[3] W. Wang, X. Wang, C. Xiang, C. Wei, and Y. Zhao, "Unscented Kalman filter-based battery SOC estimation and peak power prediction method for power distribution of hybrid electric vehicles," *IEEE Access*, vol. 6, pp. 35 957–35 965, 2018.

[4] K. W. E. Cheng, B. Divakar, H. Wu, K. Ding, and H. F. Ho, "Battery-management system (BMS) and SOC development for electrical vehicles," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 1, pp. 76–88, 2010.

[5] Y. Li, D. Karunathilake, D. M. Vilathgamuwa, *et al.*, "Model order reduction techniques for physics-based lithium-ion battery management: A survey," *IEEE Ind. Electron. Mag.*, 2021. DOI: 10.1109/MIE.2021.3100318.

[6] S. M. Rezvanizaniani, Z. Liu, Y. Chen, and J. Lee, "Review and recent advances in battery health monitoring and prognostics technologies for electric vehicle (EV) safety and mobility," *Journal of Power Sources*, vol. 256, pp. 110–124, 2014.

[7] Y. Li, B. Xiong, D. M. Vilathgamuwa, Z. Wei, C. Xie, and C. Zou, "Constrained ensemble Kalman filter for distributed electrochemical state estimation of lithium-ion batteries," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 1, pp. 240–250, 2021. DOI: 10.1109/TII.2020.2974907.

[8] F. Yang, X. Song, F. Xu, and K.-L. Tsui, "State-of-charge estimation of lithium-ion batteries via long short-term memory network," *IEEE Access*, vol. 7, pp. 53 792–53 799, 2019.

[9] S. Tong, J. H. Lacap, and J. W. Park, "Battery state of charge estimation using a load-classifying neural network," *Journal of Energy Storage*, vol. 7, pp. 236–243, 2016. DOI: https://doi.org/10.1016/j.est.2016.07.002.

[10] D. Liu, L. Li, Y. Song, L. Wu, and Y. Peng, "Hybrid state of charge estimation for lithium-ion battery under dynamic operating conditions," *International Journal of Electrical Power & Energy Systems*, vol. 110, pp. 48–61, 2019.

[11] B. Xiao, Y. Liu, and B. Xiao, "Accurate state-of-charge estimation approach for lithium-ion batteries by gated recurrent unit with ensemble optimizer," *IEEE Access*, vol. 7, pp. 54 192–54 202, 2019.

[12] E. Chemali, P. J. Kollmeyer, M. Preindl, and A. Emadi, "State-of-charge estimation of Li-ion batteries using deep neural networks: A machine learning approach," *Journal of Power Sources*, vol. 400, pp. 242–255, 2018. DOI: https://doi.org/10.1016/j.jpowsour.2018.06.104.

[13] W. Wang, N. Brady, C. Liao, *et al.*, "High-fidelity state-of-charge estimation of Li-ion batteries using machine learning," Aug. 2019.

[14] E. Chemali, P. J. Kollmeyer, M. Preindl, R. Ahmed, and A. Emadi, "Long short-term memory networks for accurate state-of-charge estimation of Li-ion batteries," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 8, pp. 6730–6739, 2018. DOI: 10.1109/TIE.2017.2787586.

[15] E. Chemali and M. Preindl, *Neural-network state-of-charge and state of health estimation*, US Patent App. 16/688,260, Mar. 2020.

[16] M. A. Hannan, D. N. T. How, M. S. H. Lipu, *et al.*, "Deep learning approach towards accurate state of charge estimation for lithium-ion batteries using self-supervised transformer model," *Sci Rep*, vol. 11, no. 1, p. 19 541, 2021. DOI: 10.1038/s41598-021-98915-8.

[17] D. N. T. How, M. A. Hannan, M. S. Hossain Lipu, and P. J. Ker, "State of charge estimation for lithium-ion batteries using model-based and data-driven methods: A review," *IEEE Access*, vol. 7, pp. 136 116–136 136, 2019. DOI: 10.1109/ACCESS.2019.2942213.

[18] H.-B. Choi, J.-S. Ryu, W.-J. Shin, and N. Vigier, "The impact of anthropogenic inputs on lithium content in river and tap water," *Nature communications*, vol. 10, no. 1, pp. 1–7, 2019.

[19] G. L. Plett, *Battery management systems, Volume II: Equivalent-circuit methods*. Artech House, 2015.

[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[21] B. Mehlig, "Machine learning with neural networks," *arXiv preprint arXiv:1901.05639*, 2019.

[22] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, Dec. 2014.

[23] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Bkg6RiCqY7.

[24] J. L. Ba, J. R. Kiros, and G. E. Hinton, *Layer normalization*, 2016. DOI: 10.48550/ARXIV.1607.06450.

[25] P. J. Kollmeyer, C. Vidal, M. Naguib, and M. Skells, "LG 18650HG2 Li-ion battery data and example deep neural network xEV SOC estimator script," 2020.

[26] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *2017 International joint conference on neural networks (IJCNN)*, IEEE, 2017, pp. 1578–1585.

[27] B. Ramsundar and R. B. Zadeh, *TensorFlow for deep learning: from linear regression to reinforcement learning*. " O'Reilly Media, Inc.", 2018.

[28] C. Vidal, P. Kollmeyer, M. Naguib, P. Malysz, O. Gross, and A. Emadi, "Robust xEV battery state-of-charge estimator design using a feedforward deep neural network," Apr. 2020. DOI: 10.4271/2020-01-1181.

[29] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[30] E. Stevens, L. Antiga, and T. Viehmann, *Deep learning with PyTorch*. Manning Publications, 2020.

[31] M. Abadi, A. Agarwal, P. Barham, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

A. Figures of the Models

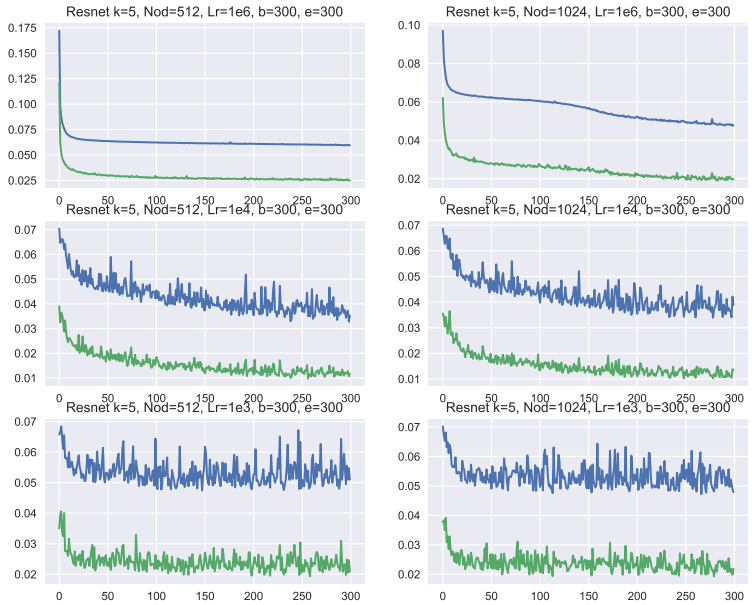


Figure A.1: RMSE and MAE for the training of ResNet models with $k = 5$.

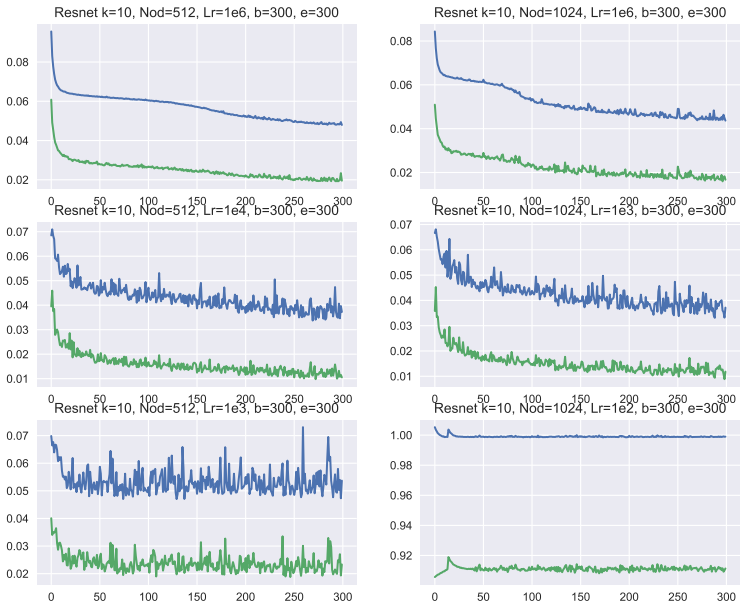


Figure A.2: RMSE and MAE for the training of ResNet models with $k = 10$.

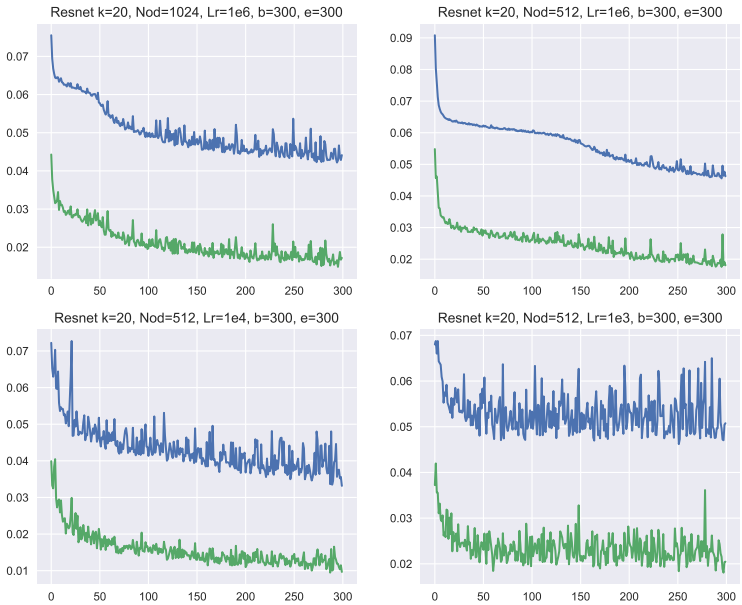


Figure A.3: RMSE and MAE for the training of ResNet models with $k = 20$..

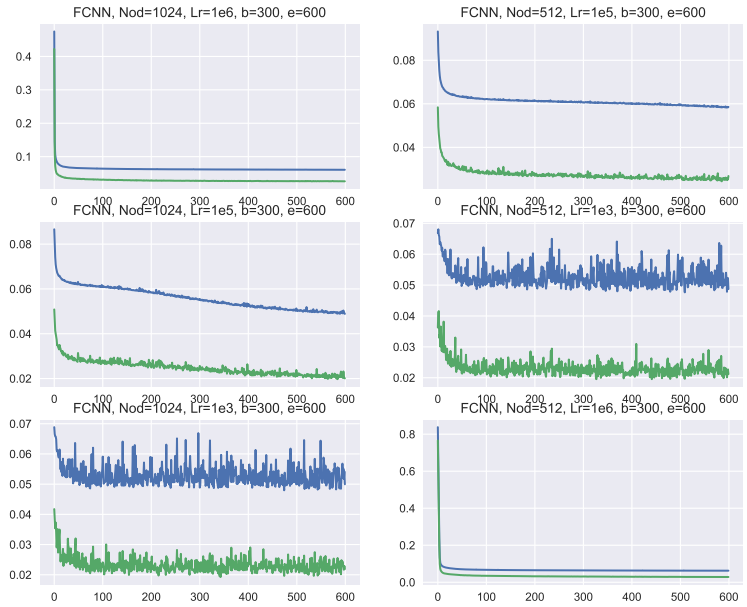


Figure A.4: RMSE and MAE for the training of FCNN

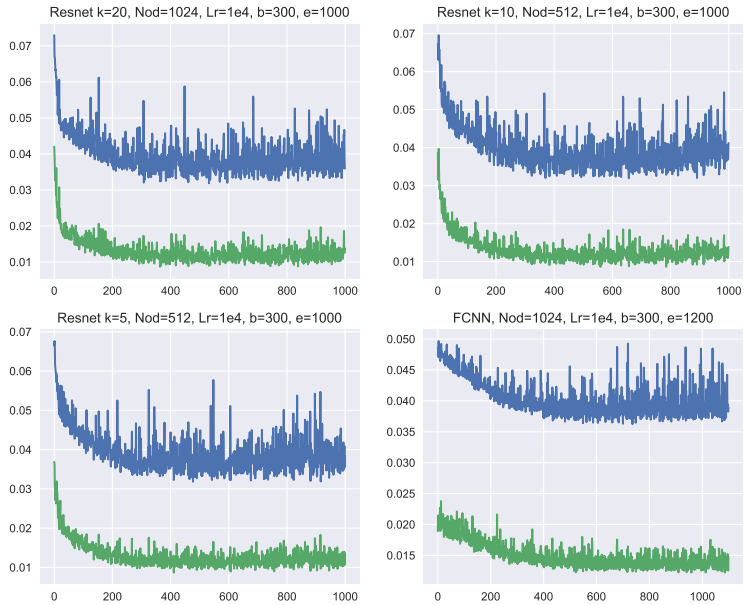


Figure A.5: RMSE and MAE for the training of ResNet and FCNN models.

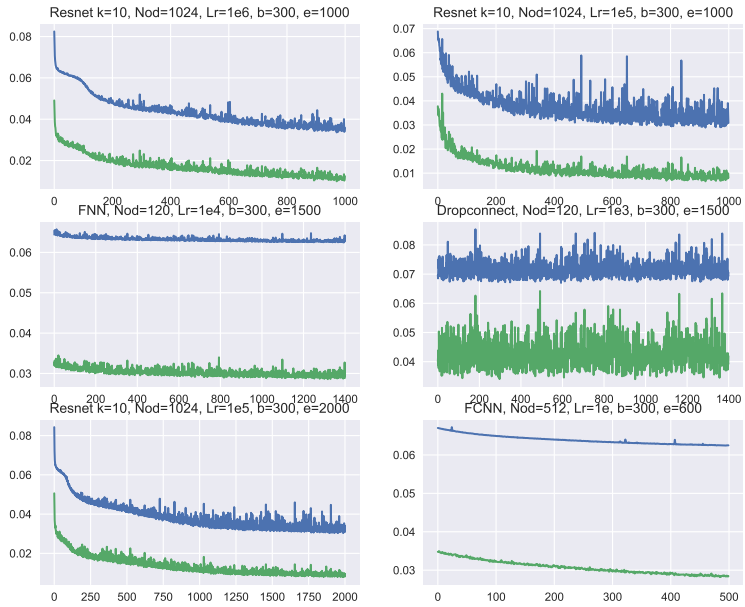


Figure A.6: RMSE and MAE for the training of different models.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY