universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Data-Driven State-of-Health Estimation and Prognostics for Lithium-Ion Batteries Using Convolutional Neural Networks

verfasst von | submitted by

Tobias Heinzle

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2025

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 821
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Mathematik
Betreut von | Supervisor:	Ass.-Prof. Yurii Malitskyi PhD

Abstract

In the world of the 21st century, the lithium-ion battery has become a staple component of almost every wireless electronic device. It is also the technological backbone of the electric mobility transformation and a key technology for the shift away from fossil energy. Driven by its ubiquitous use, methods for accurate estimation of the current maximum capacity and reliable predictions of future capacity have received considerable research attention in recent years. However, it still remains challenging to produce robust estimates of the current maximum capacity, or to predict future degradation behavior. Many methods require very restrictive conditions and high-fidelity measurements. In this thesis, the usage of deep learning techniques for state of health estimation and prediction of the capacity curve knee point is evaluated. By leveraging unsupervised feature extraction, ensemble learning, and prediction post-processing, a flexible method for state of health estimation, based on short CC-CV charge segments, is developed. In addition, the same techniques are applied to the task of knee point prediction. A novel labeling strategy for the knee point is presented to capture the uncertainties that lie in different localization methods. Based on the model outputs, a probability distribution for the knee point is constructed, and an effective measure of prediction confidence is derived from the variance. All the experiments in the thesis were conducted using open-source, freely available datasets of battery degradation experiments.

Zusammenfassung

In der Welt des 21. Jahrhunderts ist der Lithium-Ionen-Akku nicht mehr wegzudenken. Er ist die wichtigsten Komponenten beinahe aller kabellosen elektronischen Geräte und darüber hinaus auch das Rückgrat der Elektromobilität, sowie eine Schlüsseltechnologie der Wende zu klimaneutralen Energiequellen. Aufgrund dieser allgegenwärtigen Nutzung wurden in den vergangenen Jahren zahlreiche Methoden zur Schätzung und Prognose der momentanen maximalen Batteriekapazität entwickelt. Jedoch ist es derzeit immer noch schwierig, robuste Schätzungen der Kapazität oder des zukünftigen Verschleißes während des Regelbetriebs vorzunehmen. Viele Methoden können nur in sehr spezifischen Situationen angewandt werden oder benötigen überaus genaue Messungen. In dieser Arbeit wird die Verwendung von Deep-Learning-Modellen zur Schätzung der derzeitigen maximalen Kapazität und zur Vorhersage des Kapazitätsknickpunktes evaluiert. Ein flexibles maschinelles Lernverfahren, welches ausgehend von kurzen Ladesegmenten den Gesundheitszustand einer Akkuzelle schätzen kann, wird präsentiert, außerdem wird eine neuartige Strategie zur Bestimmung und Vorhersage des Knickpunktes der Kapazitätskurve ausgearbeitet. Um die Unsicherheiten der zugrundeliegenden Daten berücksichtigen zu können, wird ein Model entwickelt, welches eine Wahrscheinlichkeitsverteilung für den Knickpunkt prognostiziert. Aus der Varianz dieser Verteilung lassen sich effektive Indikatoren für die Sicherheit der Prognosen ableiten. Alle Experimente in dieser Arbeit wurden mit frei verfügbaren Open-Source-Daten aus systematischen Batterieverschleißtests durchgeführt.

Acknowledgments

First and foremost, I want to thank Prof. Yurii Malitskyi for his strong guidance, for the unwavering support and for always making time to answer my abundant questions. Furthermore, I also want to thank Dr. Darko Stern for the constructive feedback and the many engaging talks, that lead to valuable insights and understanding. Lastly, I thank my family and my girlfriend for the moral support during my entire studies and the writing of this thesis.

In addition, I gratefully acknowledge the financial support of the Austrian Federal Ministry of Climate Action, Environment, Energy, Mobility, Innovation and Technology, and the Austrian Federal Ministry of Digital and Economic Affairs implemented by Austria Wirtschaftsservice (aws) and the Austrian Research Promotion Agency (FFG) in the frame of the Important Project of Common European Interest (IPCEI) on Microelectronics and Communication Technologies (ME/CT).

Contents

1. Introduction

The Lithium-ion battery is used by the majority of the worldwide population on a daily basis. Just mobile phones alone create large demand, with the global number of smartphone users estimated at about 4.88 billion [1]. Another growing field is the adoption of electric vehicles, with almost 14 million new cars registered in 2023, a 35% increase compared to to 2022 [2]. As a key technology powering the storage of renewable energy and the shift away from fossil fuels, Lithium batteries form a cornerstone of current efforts toward climate neutrality and green energy. In the face of rapid growth in battery usage worldwide, efficient management and monitoring during operation has been pushed into the limelight. Two important subtasks in the field are the accurate estimation of the current state of health, as well as reliable forecasting of future performance and end of life.

In this thesis, the estimation of the current maximum capacity and the prediction of the capacity curve knee point are investigated. This is done by leveraging open-source experimental data, collected in testbed configurations. Several methods are compared for estimating the state of health, based on partial charge segments, and a novel strategy of labeling and predicting knee points is presented.

The following Section 1 contains a brief overview of battery terminology and the general problem setting. In Section 2, the current state of research and open source datasets are presented and Section 3 explains the background of the methods used in this thesis. Next, in Section 4 the specific datasets and pre-processing steps are detailed and Section 5 describes the concrete experiment setups. In Section 6 the results of the experiments are discussed and Section 7 conclusively wraps up the findings.

1.1. Battery structure and function

Lithium-Ion batteries are manufactured in many different shapes and sizes. The basic building block of a battery is called **cell** and it consists of three fundamental parts [3]:

- **Electrolyte**: A liquid solution that enables the transfer of Lithium-Ions between the two solid electrodes.
- **Anode**: The negative electrode, made from metal or alloy. During the discharge process, the anode releases electrons to the external circuit. This frees up Lithium-Ions that were previously intercalated in its structure.
- **Cathode**: The positive electrode, usually made in a layered structure of metallic oxide, sulfides and carbon material. The cathode takes up positively charged ions from the electrolyte during the discharge process.

A schematic of a cylindrical cell is sketched in Figure 1. In Figure 2 a simplified diagram is shown that illustrates the underlying principles. The anode and cathode are separated by a porous isolation layer and the whole cell is filled with an electrolyte solution. During discharge, as electrons continuously flow out to the external circuit, more and more positively charged Lithium ions are released to the electrolyte. They then migrate over to the cathode and intercalate into its structure. During charging, this process is reversed. The electrolyte acts as an isolator, thus no electrons can flow between the anode and cathode directly. As the battery is charged, a potential difference is built up between the negatively charged, electron rich anode and the positively charged cathode.

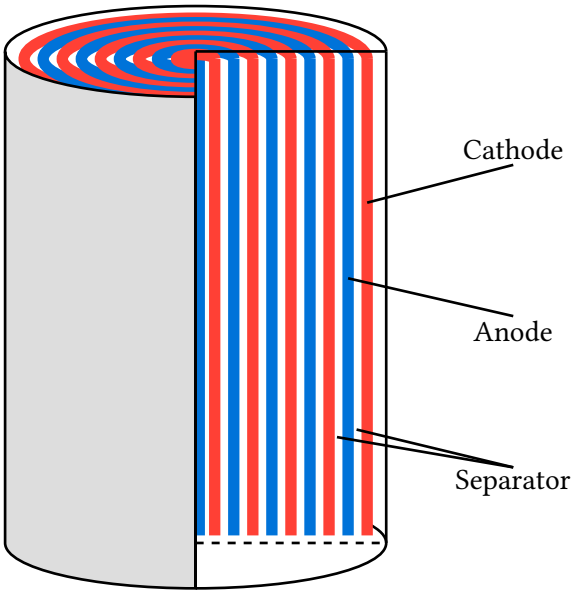


Figure 1: Schematic of the internal construction of a cylindrical battey cell. It consists of multiple anodes, cathodes and separation layers stacked on top of each other and it is filled with a liquid electrolyte.

Smaller batteries such as the ones used in smartphones and other handheld electronic devices usually consist of a single cell only. Larger batteries like those used for electric vehicles are made of many cells grouped together in battery packs. The data used throughout the experiments was collected from cycling single commercial cylindrical cells of type 18650 [4]. These cells have a relatively small form factor, with an 18 mm diameter and a height of 65 mm, hence the name. They are widely used in consumer electronics, for example in devices such as power tools, laptop battery packs or cameras. As members of large battery packs they have also been used by Tesla their electric vehicles [5]. For the remainder of the thesis however, only single cells will be considered and the words battery and cell will be used interchangeably.

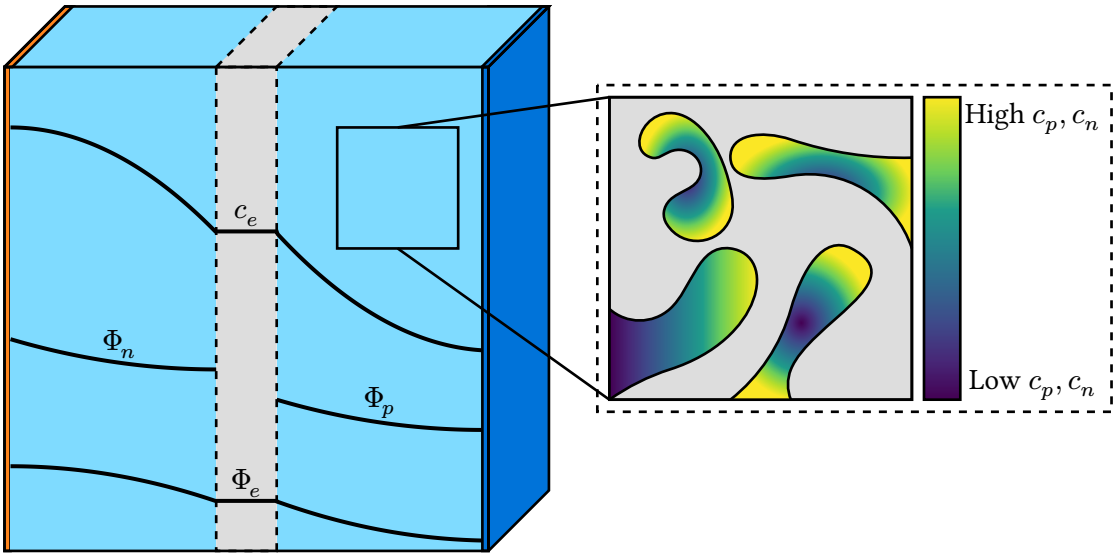


Figure 2: Simplified geometry of a cell cross section as used in e.g. the Doyle-Fuller-Newman model [6]. The left and right sections of the cells are the two electrodes (light blue), and in between there is a separator (gray). These solid parts of the cell are made of porous solid materials. All empty space inside the pores is filled by a liquid electrolyte. In the smaller zoomed in plot, a sketch of this porous microstructure is shown. Gray area represents the empty space filled by electrolyte and the colored areas correspond to the solid structure of the electrode. In the diagram, also the electrochemical variables inside the battery are shown: Ion concentration in the electrolyte c_e, electrolyte potential Φ_e, electrode potentials Φ_n and Φ_p and concentration of intercalated lithium in the negative/positive electrode c_n and c_p (yellow/blue).

1.2. State of health estimation

There are two main ways to define the **state of health (SoH)** of a battery [7], either based on capacity

$$\mathrm{SoH}_{\mathrm{capa}}(t) = \frac{C(t)}{C_0} \tag{1}$$

or internal resistance

$$\mathrm{SoH}_{\mathrm{res}}(t) = \frac{R_0}{R(t)}. \tag{2}$$

The constants C_0, R_0 denote the initial capacity and resistance and $C(t), R(t)$ the capacity and resistance at time t. As the battery degrades, capacity reduces and resistance increases. Throughout the remainder of the thesis, the SoH will be defined in terms of the capacity as in (1). When the current maximum capacity reaches a certain threshold, e.g. about 80% of the nominal capacity, the battery is considered to be at its **end of life (EoL)**. The amount of time left from until the EoL is called the **remaining useful life (RuL)**. The precise EoL threshold usually depends on the application [8].

Determining the current maximum capacity is not easy during normal operation, as it requires the full charge and subsequent discharge of the battery at a very low current. Rates of about 0.2 times nominal capacity and lower are required to accurately measure the current cell capacity by means of integration. This approach is called coulomb counting [7], and it is based on the following physical principle:

$$Q = \int_{t_0}^{t_1} \eta(s) I(s) ds. \tag{3}$$

The above equation (3) states that the charge Q gained in the time interval $[t_0, t_1]$ is the integral of the applied current $I(s)$ and the charging efficiency $\eta(s)$. The sensitivity of this method to precise measurements and identical conditions during integration, unfortunately, severely limit the applicability of such schemes [9]. However, for many scenarios, frequent and accurate estimations of the SoH are needed to optimize system performance and longevity. In the experiments conducted for this thesis, the SoH was estimated based on voltage and current curves of charge cycles performed at varying rates and starting from only partially discharged cells. A typical charge to full capacity of a Lithium-Ion battery is shown in Figure 3 and consists of two sequential stages:

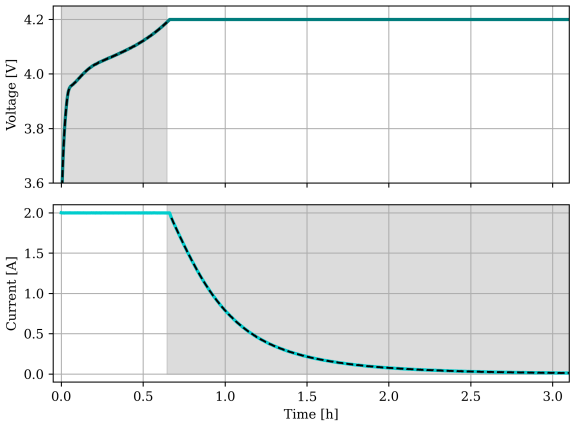


Figure 3: Example of a charge from empty to full capacity at rate of 2 times the nominal capacity. The top shows the voltage profile and on the bottom the applied current profile is plotted.

- **Constant Current (CC) stage:** In this stage, the charging current is kept constant, and the voltage of the battery steadily increases over time. This part usually carries a lot of information about the battery state of health, which can be extracted by analyzing the voltage curve V with respect to the throughput Q. If the charge is performed at a very low rate, it is possible to measure a characteristic voltage curve called the **Open Circuit Voltage (OCV)**. The method called Incremental Capacity Analysis [10], looks at the $\frac{\mathrm{d}Q}{\mathrm{d}V}$ curve of the OCV to identify different modes of degradation via the features of this curve. Even though the low charge rates needed to reliably observe the OCV are

almost never given in real operation conditions, the voltage curve during the CC still contains a lot of valuable information.

- **Constant Voltage (CV) stage:** As the maximum voltage of 4.2 V is reached, it is then held constant at this level and the current is allowed to vary. The charging goes on until the current reaches 0, this indicates that the battery is fully charged. The CV segment duration changes quite substantially as the battery degrades, becoming much longer relative to the CC stage. While the CV section has received less attention in battery literature, it is also a valuable input for machine learning models.

The above sequence is called a CC-CV charge. Later during Section 4.2.1 and Section 5.1.1 it will be explained how measurements of these charge segments can be used to estimate current capacity of a battery, without requiring restrictively low charge and discharge currents.

1.3. Knee point prediction

A typical service life of a battery can be split into two parts, an initial period of relatively slow decline of capacity, and a later period of rapid degradation. The point of transition from one rate of decay to the other is called the **knee point** of the capacity curve. Figure 4 shows an example of a knee point at cycle 750. In practice, this transition occurs over the course of several cycles, so defining the precise location of the knee point for a given capacity curve can be tricky even for humans [11]. For example, although the knee point is mentioned repeatedly in [12], a clear definition is never stated. A possible approach to mitigate this uncertainty will be discussed in Section 4.2.3.

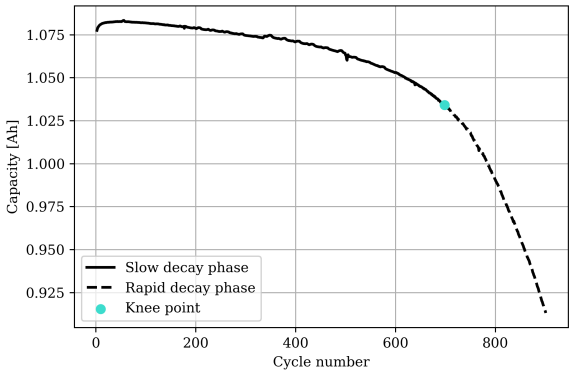


Figure 4: Capacity curve with a very pronounced two phase decay pattern. It shows a rapid decline after the knee point, which was determined via the Bacon-Watts method, see Section 4.2.3.

The goal of the experiments in this thesis is to predict the remaining cycles until the knee point occurs, to estimate the uncertainty associated with the model output. Predictions of this type can be useful in many applications, in particular when assessing the potential second life usage of a battery or when planning maintenance or replacement schedules. While $\frac{\mathrm{d}Q}{\mathrm{d}V}$ analysis and classical methods can be used to estimate the knee cycle [13], many approaches run into the same issues as mentioned above, as they depend on high fidelity measurements of the OCV curve. Usually long testing scenarios are required to asses the knee location relative to the current cycle. In an effort to circumvent these restrictions, the second part of the thesis will explore the estimation of knee point location based on regular cycling data, while also considering the uncertainty in the knee point labeling process.

2. Related Work

Over the course of the last three decades, significant progress has been made in the domain of battery modeling and simulation. The introduction of the pseudo 2D model for batteries by Doyle, Fuller and Newman [6] in 1993, achieving a balance between detailed resolution of microscopic effects and computational efficiency, marked the advent of powerful simulation software for battery cells [14]. Even though advances in numerical and theoretical aspects have opened up many new possibilities in the study of lithium-ion batteries, state of the art modeling software such as PyBamm [15] still has limitations when it comes to modeling degradation of batteries and simulating long term usage effects [16]. Numerical simulation additionally requires accurate parametrization of the underlying equations, a task that by itself is also be quite difficult. Because of this, the very accurate physical models are not often used in SoH estimation or RuL prediction. In fact, many online SoH estimation techniques employ a much simpler approximation for the battery, the equivalent circuit model [14]. The most widely used approaches in SoH and **state of charge (SoC)** estimation of batteries are Kalman Filter methods [17], [18], [19] , used for online parametrization of a simple equivalent circuit model. From the parametrized model the SoH can then be inferred, for example by comparing the estimated internal resistance as in (1). In [20], it is stated that for online SoC estimation, about 53% of studies from 2009-2018 in the IEEE database use Kalman Filter methods.

While SoC and SoH estimation are widely studied, the prediction of battery knee points directly from operational data has received considerably less attention. Some studies have tried to shed light on the underlying physical principles that lead to the knee point [21]. The phenomena is studied under several names, including “nonlinear aging”, “two-phase degradation” or “capacity drop-off”. Even though it has been linked to different degradation reactions such as solid-electrolyte interphase (SEI) growth or lithium plating [16], it is challenging to accurately simulate the complex interactions of these different degradation mechanisms. Even if there are reliable models available, forecasts rely on accurate parametrization of the model, which limits the applicability [21]. Because of these roadblocks on the modeling side, data driven methods based on deep neural networks are becoming more popular.

2.1. Deep learning based SoH estimation

In recent years, the usage of neural networks for SoH estimation has been explored by many different researchers. The authors in [22] leverage information in the form of charge protocols measured from batteries during operation. They make use of convolutional feature extraction models and train simultaneously in a supervised manner on a dataset labeled with SoH measurements and in an unsupervised fashion on a reconstruction task where the SoH target is unknown. A different strategy is employed in [23], where the authors use mathematical features extracted from charge segments as input to a two stage training procedure. First two neural networks are trained, an estimation model and a dynamics model. The dynamics model is used to model the underlying equations governing the decay of battery SoH, while the prediction model learns to output the current SoH. During the second stage, the weights of the dynamics model stay frozen and only the prediction model is fine tuned on new data.

Both of the methods above try to achieve accurate SoH estimates on out of distribution data, using two different approaches. This thesis aims to combine aspects of both of the above methods and evaluate different feature extraction strategies on short charge segments.

2.2. Data driven knee point prediction

For the prediction of capacity drop off, machine learning algorithms using deep neural networks have also shown promising results. In [24], several full cycles of battery operational data are fed as an input to a model consisting of convolutional input layers and a fully connected head. Another angle is tackled in [25], where the authors use LSTM networks and features extracted via incremental capacity analysis to predict the cycle location of the knee point, as well as the capacity at which it will occur. Both of the above results use different labeling strategies for the knee point, which in practice do not produce the same results. While the Bacon-Watts model [26] is used throughout several of the studies, it is not clear if it is the optimal method. In [27], the authors account for uncertainty in model predictions by applying a method called Monte Carlo Dropout [28], to estimate both the knee point location and the standard deviation of the prediction. However, they also use just the Bacon-Watts method for labeling and do not take into account the uncertainty that stems from different labeling approaches. Techniques for modeling of uncertainty in the label data as well as the model predictions are known from the computer vision domain [29]. In particular, this thesis leverages results on labeling via gaussian densities, as employed in the analysis of medical images [30] to account for both the modeling and labeling variance. Besides the Bacon-Watts model, two additional mathematically motivated knee point localization strategies are used to derive mean and variance of the knee point locations.

2.3. Open source datasets

In addition to the machine learning methods, also a sizeable amount of open source battery aging datasets has been published, covering many different cell chemistries and enabling data driven investigation of battery degradation mechanisms [31]. However most of the open source data includes only laboratory tests and relatively few datasets from applications such as electric vehicle drive data are made public [32]. All of the data used in this thesis is openly available online, an in-depth description is given in Section 4.1.

2.4. Contributions

In this thesis, deep learning techniques for estimating the SoH directly from a CC-CV charge segment are evaluated, based on feature engineering and unsupervised feature extraction. The performance of these methods is compared, and using an ensemble of models as well as prediction post-processing, the quality and robustness of the model is further increased.

For knee point prediction, a novel labeling approach is explored, and the feasibility of including label uncertainty in the training process is evaluated. The three different labeling approaches for the curve knee lead to an empirical distribution of the knee point. These distribution are then used as labels and an ensemble learning approach is pursued to predict knee point density functions. The variance of the predicted density serves as an indicator of model confidence.

3. Methods

In contrast to the traditional model based procedures for SoH estimation using Coulomb counting and equivalent circuit models [7], the methods considered in the following sections are based on machine learning algorithms that operate on raw data. Instead of devising an algorithm to estimate SoH or the knee point based on physical models, a parameterized function is fitted to the data. The function parameters are determined using optimization algorithms, this process is called "learning".

In this thesis, all problems considered involve regression. Different methods are applied to several problems in battery state of health estimation and health prognostics. First, the general framework of regression is presented and then the specific methods are discussed.

3.1. Regression

Given a set of inputs $x_i, ..., x_N \in \mathbb{R}^n$ and corresponding outputs $y_i, ..., y_N \in \mathbb{R}^m$, the objective of regression is to fit a parametrized function $f_\theta(x) : \mathbb{R}^n \to \mathbb{R}^m$ to the data, such that the errors

$$\|f_\theta(x_i) - y_i\| \tag{4}$$

in a given norm $\| \ . \ \|$ are as small as possible. Examples for regression tasks are the prediction of future stock prices from data of the past, or estimating a persons age based on facial images. In practice, a performance measure $\mathcal{L}$ called **loss function** is used to compute the error over the entire dataset. Consider the prediction matrix $\hat{Y} = (f_\theta(x_1), ..., f_\theta(x_N))^T$ and the target matrix $Y = (y_1, ..., y_N)^T$. Then,

$$\mathcal{L}\left(\hat{Y}, Y\right) : \mathbb{R}^{N\times m} \times \mathbb{R}^{N\times m} \to \mathbb{R}^+, \tag{5}$$

and the regression problem boils down to finding the optimal parameter values θ^*, given by

$$\theta^* = \arg\min_\theta \mathcal{L}\left(\hat{Y}, Y\right). \tag{6}$$

A crucial part of training a regression model is the choice of loss function $\mathcal{L}$. Two popular options are:

- **Mean Squared Error (MSE):**

$$\mathcal{L}\left(\hat{Y}, Y\right) := \frac{1}{N}\sum_i \|f_\theta(x_i) - y_i\|_2^2. \tag{7}$$

- **Mean Absolute Error (MAE):**

$$\mathcal{L}\left(\hat{Y}, Y\right) := \frac{1}{N}\sum_i \|f_\theta(x_i) - y_i\|_1. \tag{8}$$

How to find the optimal parameters θ in practice depends on the computational structure of f_θ. The models used in this thesis are all deep neural networks, trained with first order optimization algorithms.

3.2. Evaluation of a regression model

To asses the quality of model outputs can be a quite challenging task. At first glance, looking at the loss function seems to be a suitable choice of metric. The loss can be low, but the learned model might still fail to generalize or exhibit poor performance in a different aspect. It is hence not advisable to only focus on one value alone, the true model quality can usually not be captured by a single score. In the list below, it is assumed that the targets y_i are scalar real values. Some widely used metrics for regression models are the following [33]:

- **Mean Absolute Error (MAE):** Besides being used as a loss function, the MAE is also a suitable metric for model performance in general. It is the most straightforward error measure for a regression problem, consisting of a simple average of all absolute errors over a set of samples.

$$\text{MAE} \quad = \quad \frac{1}{N} \sum_{i=1}^{N} |\ f_\theta(x_i) - y_i| \tag{9}$$

- **Root Mean Squared Error (RMSE):** This measure emphasizes more the error on large outliers. Since the squaring operation amplifies large error contributions before summing up, it is more sensitive to single examples with large deviations from the target.

$$\text{RMSE} \quad = \quad \sqrt{\frac{1}{N} \sum_{i=1}^{N} \| f_\theta(x_i) - y_i \|_2^2} \tag{10}$$

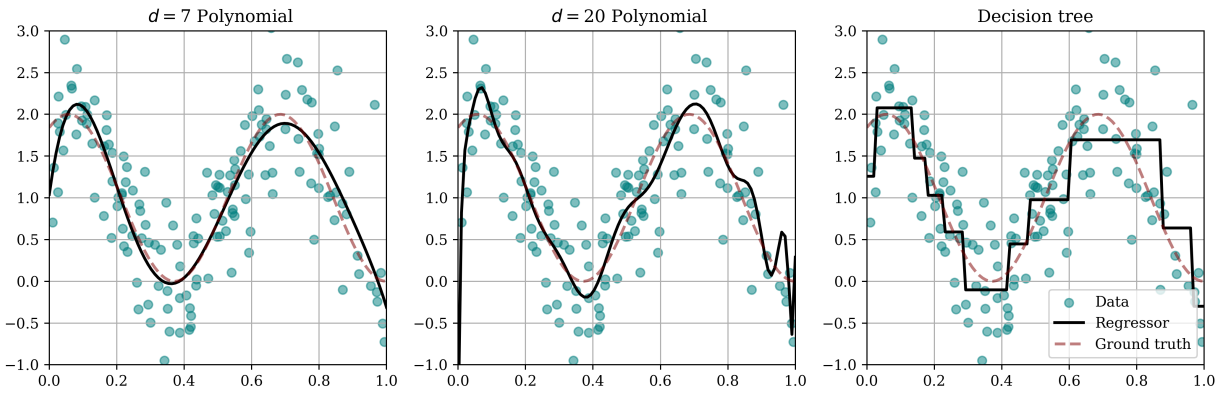
- **Determination Coefficient R^2:** In contrast to the previous two metrics, the R^2 score aims to measure a goodness-of-fit that is bounded from above by 1 and unbounded on the negative scale. If a regressor perfectly predicts every y_i the score will be 1, and if it predicts the mean all y_i for every input x_i then the score will be exactly zero. The scores are however not bounded from below, so a regressor can achieve an arbitrarily low negative score.

$$R^2 \quad = \quad 1 - \frac{\sum_{i=1}^{N} \left(f_\theta(x_i) - y_i \right)^2}{\sum_{i=1}^{N} \left(\overline{y} - y_i \right)^2} \quad \text{where} \quad \overline{y} = \frac{1}{N} \sum_{i=1}^{N} y_i \tag{11}$$

- **Mean Absolute Percentage Error (MAPE):** The ratio of the absolute error to the absolute value of the target. Measures the average relative error of the predictions.

$$\text{MAPE} \quad = \quad \frac{1}{N} \sum_{k=1}^{N} \frac{|f_\theta(x_i) - y_i|}{|y_i|} \tag{12}$$

Since every choice of metric emphasizes different strengths or weaknesses of the predictions, it is important to consider a set of several meaningful performance measures. In Figure 5, some metrics are compared on a simple regression problem to illustrate this point.



MSE	0.26	0.24	**0.23**
$P_{<\frac{1}{3}}$	0.48	**0.52**	0.5
MAPE	**1.07**	1.16	1.18

Figure 5: Example of different metric choices. The data in this example is generated by $g(x) = 1 + \sin(10x + 1)$. A set of N points x_i are sampled uniformly in $[0, 1]$, then the labels are computed via $y_i = g(x_i) + \varepsilon, \varepsilon \sim \mathcal{N}(0, 0.25)$. While the plots are ordered from left to right in descending order with respect to MSE, in the MAPE metric the order is reversed. When considering the number of points with $|f_\theta(x_i) - y_i| < \frac{1}{3}$, denoted as $P_{<\frac{1}{3}}$, the $d = 20$ polynomial scores the highest.

It is also important to evaluate a machine learning model not only on its training data, but rather on some representative set of previously unseen inputs. The choice of this holdout data can skew results into a particular direction and compromise the evaluation of model quality and generalization capabilities if chosen incorrectly. In the case of battery data, it would for example be insufficient to mix measurements of different cells together into a big dataset and then extract the holdout segment as a random sample. This would inflate metrics on the holdout and fail to effectively measure the generalization capability of the model. In an actual engineering applications, the measurements are collected per cell and new data will always come from a cell that the model has never seen before during training. It is thus important to split the data into holdout and training segments on a cell level, to ensure that the testing mirrors real operating conditions as closely as possible.

3.3. Neural Networks

When it comes to parametrized regression models, **Neural Networks (NN)** offer unparalleled flexibility. The most basic form of a neural network is the **fully connected (FC)** architecture. It consists of several layers, each layer containing of a fixed number of nodes, also called **neurons**. The nodes of subsequent layers are all connected via edges, each neuron of the previous layer is connected to each neuron of the next layer by a single edge, as depicted in Figure 6. Every neuron has a bias parameter $b_i \in \mathbb{R}$, and each edge has a weight parameter $w_i \in \mathbb{R}$. All parameters of an entire layer can be represented as a weight matrix $W \in \mathbb{R}^{m \times n}$ and a bias vector $b \in \mathbb{R}^n$, where m is called the input dimension and n is the output dimension.

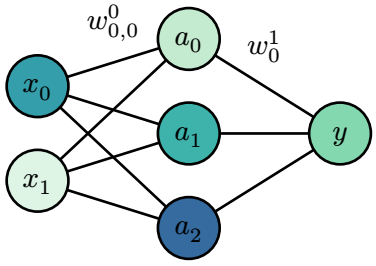


Figure 6: Neural network with input dimension 2, a single hidden layer of width 3 and output dimension 1. The entries of the weight matrices W_i are given by the scalars $w^i_{j,k}$. Each node of the network also has an additional scalar bias parameter b^i_k. The vector b_i denotes the bias vector of the i-th layer. To evaluate the neural network for an input vector $x = (x_0, x_1)^T$, first compute the vector of activations of the hidden layer as $a = \sigma(W^0 \cdot x + b_0)$. The final output of the network is an affine linear function of the activations $y = W_1 a + b_1$.

The only additional ingredient is a nonlinear activation function $\sigma : \mathbb{R} \to \mathbb{R}$. Then, a neural network with d layers is defined via

$$\begin{aligned} a_0 &= x \\ a_i &= \sigma(W_i a_{i-1} + b_i) \text{ for } i = 1, ..., d-1 \\ y &= W_d a_{d-1}, \end{aligned} \tag{13}$$

where x is the input to the network and y is the output. W_i and b_i denote the weight matrix and bias vector of the i-th layer. The activation function σ is applied element wise after each layer and turns these otherwise affine linear maps into a class of functions with rather powerful approximation properties [34]. Their ability to approximate any function arbitrarily well, given enough parameters, makes them a popular choice for machine learning applications. If a neural network is composed of more than one hidden layers, it is called a Deep Neural Network (DNN). Throughout this thesis several deep neural networks will be used as regressors.

3.3.1. Convolutional neural networks

Convolutional neural networks (CNN) are a subclass of neural networks that excel on spatially related data, such as 2D image or measurement data sampled at regular time intervals [35]. The basic building block is called convolution layer, which performs discrete mathematical convolution on the input with some kernel K_θ. The output of a 1D convolution layer applied to an input vector $x \in \mathbb{R}^n$, with kernel $K_\theta \in \mathbb{R}^l$ and stride $h \in \mathbb{N}$ is given by

$$y_i = \begin{pmatrix} x_{ih} \\ x_{ih+1} \\ \vdots \\ x_{ih+l-1} \end{pmatrix}^T \cdot K_\theta \quad \text{with} \quad K_\theta = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_l \end{pmatrix}. \tag{14}$$

The operation is visualized in Figure 7. Usually, an activation function σ is applied to the output of the convolution layer to introduce a nonlinearity. The kernel K_θ is learned during training. In practice, using convolution layers can significantly reduce the amount of parameters used in the model, compared to fully connected layers.

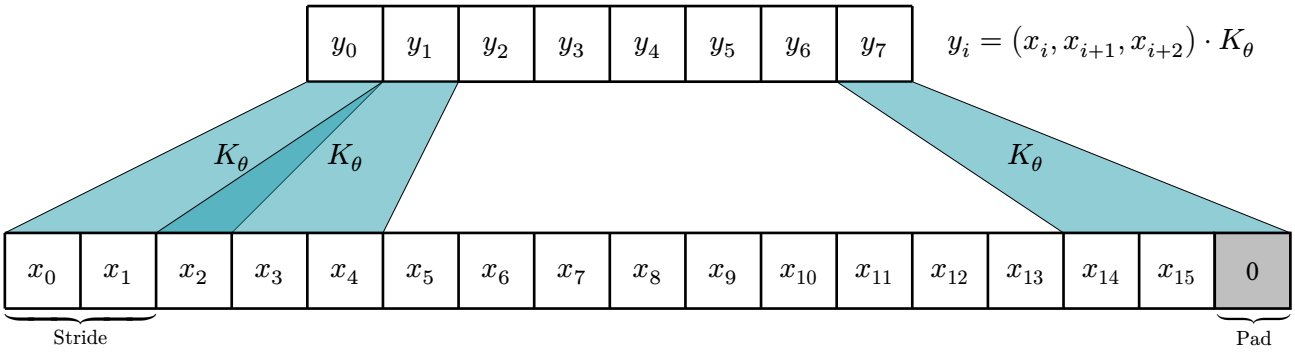


Figure 7: Illustration of 1D convolution with a kernel of size 3 and stride 2. Because the kernel $K_\theta \in \mathbb{R}^3$ gets moved two indices in each step, the output dimension is half of the input dimension. In the last step, the kernel reaches over the border of the input vector, thus it needs to be padded at the end.

3.4. Training of neural networks

Neural networks are usually trained by minimizing the loss function using iterative first order optimization algorithms [35]. The simplest algorithm of this type is called **Gradient Descent (GD)**, and for the MSE loss function (7), the update for the weight vector $\theta_k \in \mathbb{R}^m$ at step k is given by

$$\begin{aligned} \theta_{k+1} &= \theta_k - \alpha \nabla_\theta \mathcal{L}(\theta_k) \\ &= \theta_k - \frac{\alpha}{N} \sum_i \nabla_\theta \, \| f_{\theta_k}(x_i) - y_i \|_2^2 \\ &= \theta_k - \frac{\alpha}{N} \sum_i 2 \boldsymbol{J}_{\theta_k}(x_i)^T \big(f_{\theta_k}(x_i) - y_i \big). \end{aligned} \tag{15}$$

In the above equation, $\alpha \in \mathbb{R}^+$ is called the step size or learning rate, and $\boldsymbol{J}_{\theta_k} \in \mathbb{R}^{m \times l}$ denotes the Jacobian of f_θ with respect to θ. Modern machine learning software packages such as PyTorch [36] and JAX [37] offer powerful automatic differentiation capabilities, that handle the computation of the Jacobians programmatically without resorting to numerical approximations. In the case where $m = 1$, the matrix $\boldsymbol{J}_{\theta_k}$ just becomes the gradient vector $\nabla_\theta f_{\theta_k}$.

Immediately a problem arises. The evaluation of the gradient $\nabla_\theta \mathcal{L}(\theta_k)$ involves the calculation of N Jacobians of f_{θ_k}, one for each training example (x_i, y_i). Because of this computational difficulty, instead of taking the gradient with respect to the entire training data set, only a smaller, randomly sampled batch is used in each step. Since the gradient at step k is now a random variable G_k with mean $\nabla_\theta f_{\theta_k}$, the resulting algorithm is called **Stochastic Gradient Descent (SGD)**.

3.4.1. Advanced training algorithms and regularization

In general, the proper choice of the step size α is not clear a priori, since the objective function $\mathcal{L}$ is usually not convex. Also, the size of the batches can affect the convergence. All of these auxiliary parameters are called hyperparameters, and their values can significantly impact optimization performance. To mitigate the necessity for parameter tuning, several improvements of the standard SGD method have become popular in deep learning. One of the most widely used modifications of SGD is called Adam [38], and its update is more complicated than that of SGD. It has yet more hyperparameters, however it makes up for this increased complexity with an empirically faster rate. For the experiments in this thesis, a variant of Adam called AdamW [39], in particular the optax [40] implementation, is used. AdamW uses weight decay, a technique to improve generalization capabilities of the trained neural network. Weight decay tries to keep the norm of the weights θ small during training, effectively driving them towards 0. This in turn increases the robustness of the model and counteracts overfitting [35]. For SGD, the weight decay update is defined as

$$\theta_{k+1} = (1 - \lambda)\theta_k - \alpha \nabla_\theta \mathcal{L}(\theta) \tag{16}$$

which is equivalent to L_2 regularization, and amounts to optimizing the loss

$$\mathcal{L}_{\text{reg}}(\theta) = \mathcal{L}(\theta) + \frac{\lambda}{2\alpha}\|\theta\|_2^2 \tag{17}$$

for some regularization parameter $\lambda \in (0, 1)$. For Adam however, L_2 regularization does not work as well and instead the weight decay is decoupled from the original loss calculation, leading to the AdamW algorithm.

4. Data

4.1. Sources

For running the experiments, data from three different sources was gathered, the NASA randomized battery usage dataset [41], the XJTU battery degradation dataset [42] and the dataset by Severson et. al. for cycle life prediction [43]. All of the sources used batteries with an 18650 form factor, but with different chemistries. In Table 1, dataset size and battery chemistry are listed. In total the data of 207 cells is considered, Figure 9 shows sample profiles for each of the three datasets. The measurements were collected from batteries cycled in a testbed configuration. In all cases, voltage, current and temperature measurements at regular intervals are available. The data on a cell level is structured as a series of experiment cycles, where a sequence of different load profiles is applied repeatedly. For two of the three datasets, periodically a full reference charge and subsequent reference discharge of the cell is performed. The result of a full discharge can be used to measure the capacity of the cell.

Source	Chemistry	Nominal Capacity	Nominal Voltage	Dataset Size
XJTU	Nickel Manganese Cobalt	2000 mAh	3.6 V	55
Severson	Lithium Iron Phosphate / Graphite	1100 mAh	3.3 V	124
NASA	Lithium Cobalt Oxide / Lithiated Carbon	2000 mAh	3.6 V	28

Table 1: Data overview. For the NASA dataset the chemistry, nominal capacity and nominal voltage are not explicitly stated in the original article [44] or on the website, but the values here are inferred from statements in their paper (chemistry) or from the data itself (nominal capacity/voltage).

4.1.1. NASA randomized battery usage dataset

This dataset includes data of 28 cells cycled with random discharge profiles until a certain voltage threshold is reached, then charged again using a CC-CV protocol. This cycling is preformed until the measured capacity reaches about 60% of the nominal capacity, with capacity measurements performed at fixed intervals via a reference charge and discharge cycle. The batteries are divided into 7 distinct batches of 4 cells each.

- **Batch 1:** All charge and discharge currents are uniformly sampled between −2.25 C and 2.25 C.
- **Batch 2:** Batteries are charged to 4.2 V and subsequently discharged using currents randomly sampled between 0.25 C and 2 C.
- **Batch 3:** Same as batch 2, but the batteries are not charged until 4.2 V, but rather for a random time interval ranging 30 minutes to 4 hours.
- **Batch 4:** Same as batch 2, but discharge currents are biased towards the higher values.
- **Batch 5:** Batteries are charged to 4.2 V; Random discharge currents between 0.25 C and 2.5 C.
- **Batch 6:** Same as batch 5, but with a higher ambient temperature.
- **Batch 7:** Same as batch 5, but the currents are biased towards the lower end of the scale.

The discharge cycles always last until a voltage of 3.2 V is reached.

4.1.2. XJTU battery degradation dataset

The dataset consisting of 55 cells is divided into 6 distinct batches:

- **Batch 1 & 2:** For the first two batches, with 8 and 15 batteries respectively, the cells are repeatedly fully discharged and charged with a fixed load.
- **Batch 3 & 4:** The third and fourth batch contain 8 cells each and are cycled using variable load strategy. For each discharge, a different current is applied. The currents are cycled through a set of

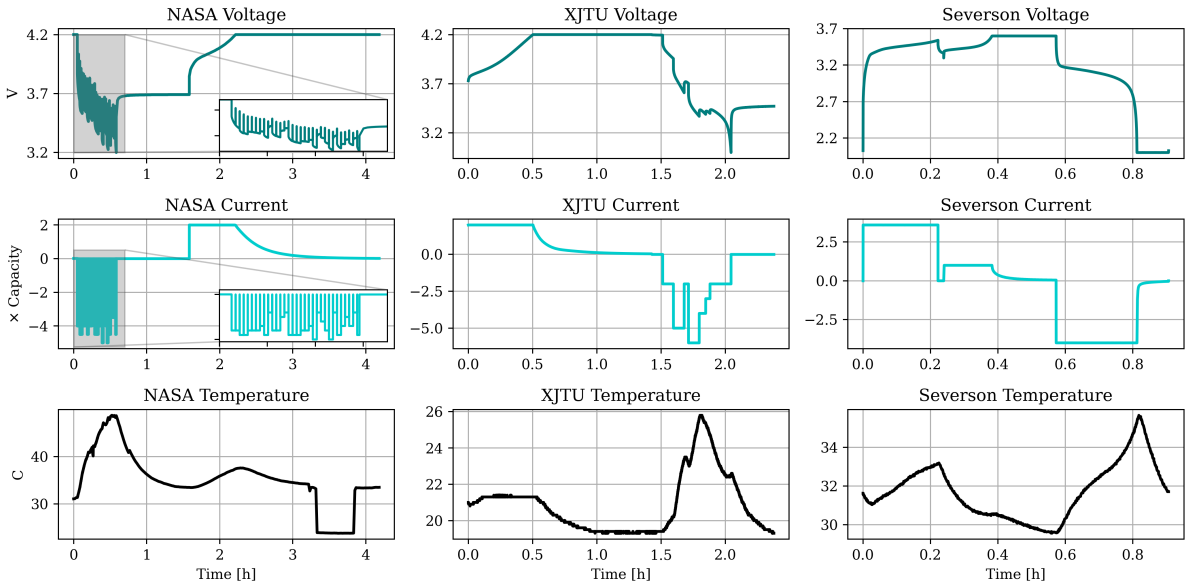


Figure 9: Example profiles for the different datasets. Each column shows a single cycle of data from one cell in the dataset. In the NASA plots, the detailed structure of the random discharges is shown in the magnified section.

four predefined levels. For batch 3 the cell is always cycled until it is empty, batch 4 does not fully discharge the cells.

- **Batch 5:** Again a batch of 8, discharged using random currents similar to the NASA dataset. For this batch, the cells are also not fully discharged.
- **Batch 6:** The final batch simulates the load profile for a battery used in a geosynchronous earth orbit satellite. For this profile, the cell is not guaranteed to be fully charged after each cycle.

4.1.3. Severson et. al. cycle life prediction dataset

The dataset is composed of three batches of commercial LFP/graphite cells manufactured by A123 Systems, model APR18650M1A124. In total, 124 cells were cycled with different fast charging profiles. In particular, the charging consists of two stages with high C rates until 80% state of charge, then the rate is dialed down and the batteries are charged until full capacity with a CC-CV schedule at 1C. This segment is followed by a complete constant current discharge. The cycling is stopped once a cell's true discharge capacity reaches about 80% nominal capacity. Capacity is inferred from the full discharge after each experiment. Out of the 124 total cells, 5 cells are cycled with only a single fast charging stage until 80% SoC.

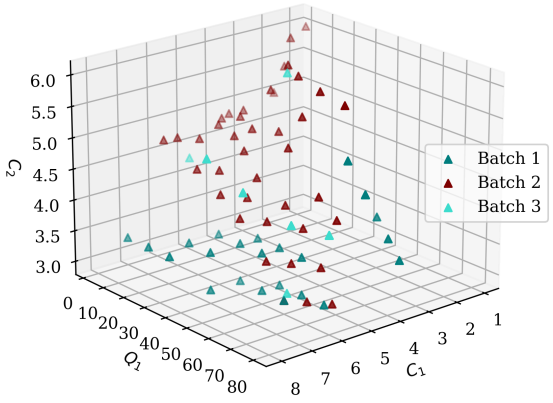


Figure 8: Parameter space of the Severson dataset, with axes C_1 and C_2 for the two charge rates and Q_1 for the SoH threshold when the rates are switched. The three different batches are highlighted.

The remaining 119 cells are cycled in a two step way, with two predetermined rates that get switched at a fixed SoC threshold. This amounts to a three dimensional parameter space for the experiments, initial rate C_1, switch threshold Q_1 and later rate C_2. In Figure 8, the parameter space of the experiments is plotted.

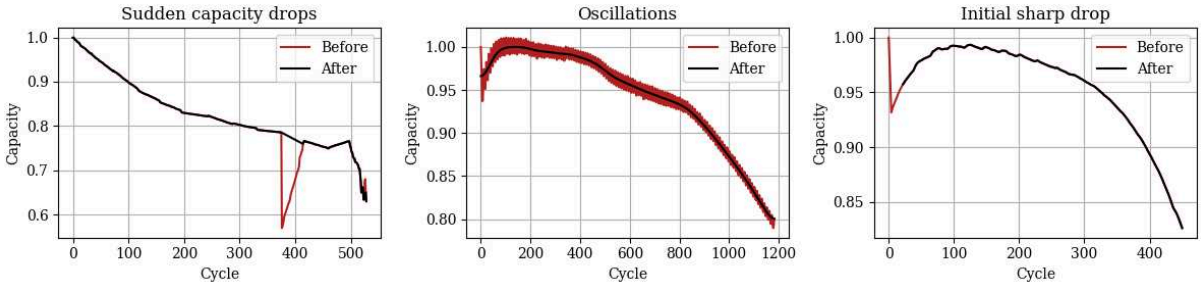


Figure 10: Examples of irregularities in the NASA and XJTU data. The curve on the left is from NASA, to middle and right plots are XJTU cells. Two of the cells show an initial increase of capacity, a clear signs of anode overhang.

4.2. Pre-Processing

The pipeline to extract data for the two different experimental setups was different. In the following two sections, the data selection, labeling and cleaning process is explained for both the SoH estimation and knee point prognostics experiments.

4.2.1. Data preparation for SoH estimation

For this scenario, the NASA and XJTU data was merged and cleaned to produce a combined dataset. Since the Severson data has a considerably different cycling protocol and also different nominal capacity and voltage, it was not used in the SoH experiments. Preprocessing for the SoH estimation was done in the following steps:

- **Label interpolation:** To determine the true SoH during the physical testing, a full reference discharge with low currents was performed in regular intervals. This provides a sparse set of labels, which is then propagated to all other cycles by means of linear interpolation. in addition, all capacities are normalized with respect to their initial capacity measurements.
- **Charge segment extraction:** For a cycle segment to qualify as a suitable model input, it must contain a charge to full capacity. If this is the case, a further filter is applied, such that only the segments which contain the full required voltage and current ranges are admitted.
- **Removal of outliers:** It is possible that subsequent reference charges can have large differences, notably in the case of the first initial capacity measurement and its successor, and larger sawtooth-like irregularities in some of the experimental data, as shown in Figure 10. The affected measurements are removed from the dataset.
- **Smoothing of oscillations:** Some of the cells exhibit regular oscillations between subsequent reference charges. The data of these cells is smoothed via Gaussian filtering.

After these steps, 23840 suitable input data points, corresponding to 71 different cells remain. It should be noted that many cells exhibit an initial slight increase in measured capacity, this is however not a measurement irregularity. Instead, it is a physical phenomena caused by a battery design concept called **anode overhang** [45], where the anode is intentionally constructed slightly larger than the cathode. Due to delayed activation of the superfluous active material, the battery can thus gain some capacity over the first few cycles. In Figure 10 two such capacity curves are shown.

4.2.2. Data preparation for Knee Point prediction

In the experiments for the knee point prediction problem, only the Severson dataset was used. As before, the preprocessing can be broken down into several steps:

- **Label generation:** As for the NASA and XJTU datasets, the labels for cell capacity were inferred from the full discharge sections in the data. Since each cycle in the dataset contains such a full discharge, no interpolation is necessary. However, in the case of knee point prediction the identifi-

cation of the knee point requires some extra care. Three different methods for calculating the knee point were used to generate a set of possible label candidates.

- **Filtration based on labeling agreement:** Based on the three generated label candidates for each curve, only curves where the points show sufficient agreement are kept.
- **Outlier removal:** Cycles that exhibit artifacts such as sudden, sharp drops in measured capacity are removed. If the entire curve features excessively noisy capacity measurements, it is completely discarded.

Because the capacities in this dataset do not exhibit the oscillations present in the NASA or XJTU data, no smoothing is performed on the capacity measurements.

4.2.3. Knee point labeling strategies

To label the knee points of the charge curves, three different strategies were used. In the literature, many different methods of estimating this point exist, for example by analyzing the magnitude of the gradient [25], or by means of fitting a special function to the data and defining the knee point as the point of maximum curvature of this function [24]. Each of the methods arrives at a slightly different result.

One might ask why it is not just possible to simply take the maximum of the curvature as the knee point. Curvature of a univariate function $f : \mathbb{R} \to \mathbb{R}$ is defined as

$$\kappa := \frac{|f''|}{(1 + f'^2)^{\frac{3}{2}}}. \tag{18}$$

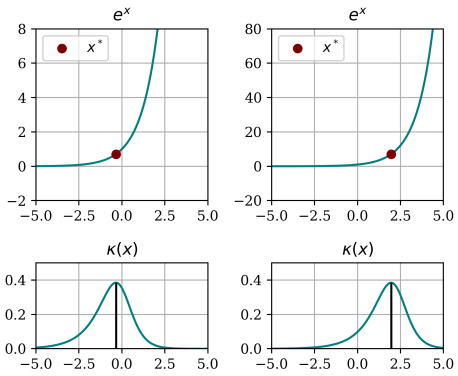


Figure 11: Effect of scaling on the exponential e^x. The right plot has its y axis scaled to 10 times the height. This corresponds to plotting $\frac{1}{10}e^x$ with an unscaled y axis, shifting the perceived knee point of the curve to the right. In the lower plot, the curvature function $\kappa(x)$ and its maximum is shown for e^x and $\frac{1}{10}e^x$.

Consider now the function $f(x) = ae^{bx}$ with $a > 0$ and $b \in \mathbb{R}$. Then the curvature function of f is given by

$$\kappa(x) = \frac{|ab^2e^{bx}|}{(1 + a^2b^2e^{2bx})^{\frac{3}{2}}}. \tag{19}$$

The value of x that maximizes the curvature $\kappa(x)$ is

$$x^* = \frac{\log\left(\frac{1}{ab}\right) - \frac{\log(2)}{2}}{b}. \tag{20}$$

It is clear from (18) and (20) that the point of maximum curvature is in general not scaling invariant. If in the above example the a parameter is decreased, the point x^* moves to the right. An example of this invariance is shown in Figure 11. The precise location of the knee point of the capacity curve is thus dependent on certain scaling choices.

For the above reason, three different strategies were used to determine potential candidates for knee points. The data used were the measured capacities y_i and their cycle indices x_i, for $i = 0, \ldots, n$. Note that in general, because of data cleaning $x_i \neq i$.

***L_2* optimal three point linear interpolation:** Consider the the function that connects the points $(x_0, y_0), (x_k, y_k)$ and (x_n, y_n) by straight lines:

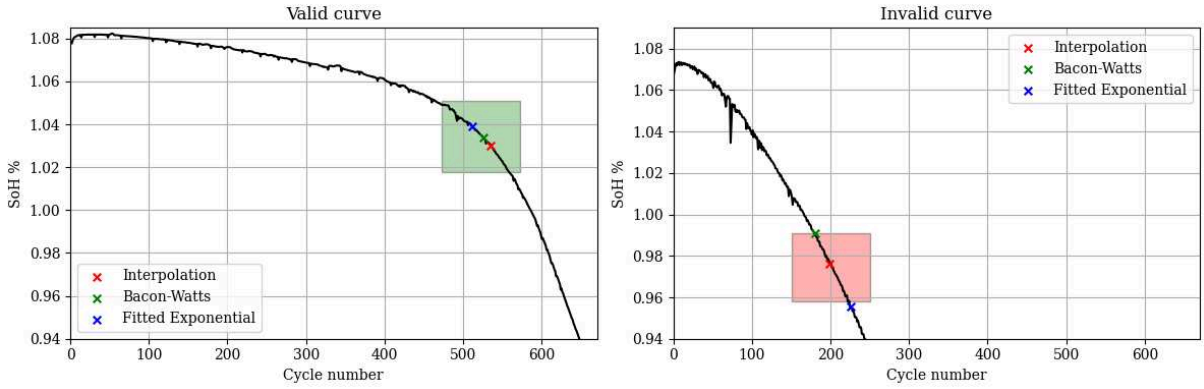


Figure 12: Two curves with corresponding knee point labels. The left curve is valid for use in training, since all three knee point labels fit inside the green box of size $100 \times \frac{1}{30}$. On the right side, the box cannot be shifted such that the green and the blue point both fit inside, thus it is labeled invalid.

$$f_k(x) := \begin{cases} y_0 + \frac{y_k - y_0}{x_k - x_0}(x - x_0) \text{ if } x \leq x_k \\ y_k + \frac{y_n - y_k}{x_n - x_k}(x - x_k) \text{ if } x > x_k \end{cases}. \tag{21}$$

It is straightforward to find k^*, such that

$$k^* = \arg\min_k \sum_i (y_i - f_k(x_i))^2, \tag{22}$$

and $x^*_{\text{interp}} := x_{k^*}$ is a natural candidate for a knee point. In practice, the first 100 values of the y_i and x_i sequences were truncated, to avoid distortions due to the frequently observed initial rise in capacity stemming from anode overhang.

Curvature of a fitted exponential with consistent scaling: Despite the above problems with curvature, it can still be used to localize knee points, provided a consistent approach to scaling is followed. As a further complication, estimating the curvature based on finite difference approximations of the capacity values is subject to numerical inaccuracy. To mitigate the effects of measurement noise, instead of directly considering the curvature of smoothed versions of the curve, a negative exponential was fitted to the data. For this function, the point of maximum curvature is easily determined. In particular, the parameters $\alpha := (\alpha_0, \alpha_1, \alpha_2)$ of

$$f_\alpha(x) := \alpha_0 - \alpha_1 e^{\alpha_2 x} \tag{23}$$

were determined such that $f_\alpha(x)$ best approximates the capacity curve in the L_2 norm. Note that for the fitting of the exponential, in order to ensure $x_0 = 0$, the x_i values are always reindexed if necessary. After fitting the curve, it is rescaled in a consistent way by choosing $c_{\min} = 0.88$ and $c_{\max} = 1.1$ as fixed thresholds for the capacity in Ah. Almost all curves in the dataset are cycled until they reach 0.88 Ah of capacity, corresponding to exactly 80% SoH. The upper threshold 1.1 corresponds to the nominal capacity of the battery. For all cells in the dataset, $f_\alpha(x) \in [c_{\min}, c_{\max}]$ always holds for all relevant x. After rescaling the output of the fitted function, the input is rescaled as well with respect to the number of cycles performed, x_n, resulting in the transformation

$$\hat{f}_\alpha(z) = \frac{1}{c_{\max} - c_{\min}}(f_\alpha(x_n \cdot z) - c_{\min}). \tag{24}$$

The new function $\hat{f}_\alpha : [0,1] \to [0,1]$ represents the decay in battery capacity from full health until failure (assumed to be at 80% SoH), normalized w.r.t. experiment duration. This particular choice of scaling ensures that the point of maximum curvature is localized consistently for all experiments. The knee point cycle x^*_{exp} is then defined as the point of maximum curvature of $\hat{f}_\alpha$ multiplied by x_n.

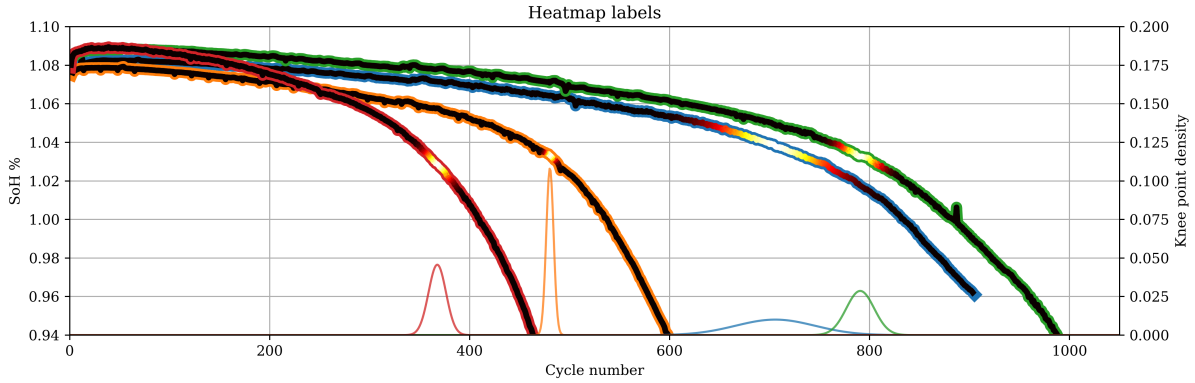


Figure 13: Different Gaussian heatmaps for the knee points. The black lines with coloured outlines are the capacity curves, and the colored Gaussians are the respective densities for the knee cycle. Some of the distributions are rather concentrated, when the three labeling methods show high agreement. In the case where the individual labels are spaced further apart the distribution is much flatter.

Bacon-Watts [26] transition point: An alternative approach to localize the knee point is to define it as the point of transition between two linear regimes. It involves finding an approximation of the data in the the form

$$f_\alpha(x) = \alpha_0 + \left(\alpha_1 + \alpha_2 \tanh\left(\frac{x-\alpha_3}{\gamma}\right)\right)(x-\alpha_3). \tag{25}$$

Intuitively, this fits two straight lines to the data and models the change from the initial to the final slope by scaling α_2 with the term $\tanh\left(\frac{x-\alpha_3}{\gamma}\right)$. The parameter γ is kept fixed, and can be used to control the smoothness of this change in slope. If γ is chosen very small the tanh acts as a sign function and the result are two connected straight lines, with a sharp knee at α_3. If γ is large, the transition becomes smoother and takes the shape of an arc. After fitting the curve, the knee point is chosen to be $x^*_{BW} := \alpha_3$. This way of determining the knee point is called the Bacon-Watts method [24].

After identifying three different candidates for knee points, only curves where all knee points (x^*, y^*) fit inside a rectangle that is 100 cycles wide and 0.033 tall on the capacity scale. An example of this filtering process is shown in Figure 12.

4.2.4. Accounting for labeling uncertainty

Because of the above mentioned diversity in identifying the knee point, it is of interest to account for this uncertainty in the labels during the training process. Instead of labeling the location directly, a Gaussian distribution can be used for localizing the knee point. By computing the mean μ and standard deviation σ of the three points, a Gaussian heatmap can be derived. It is assumed that the knee point location is normally distributed, $x^* \sim \mathcal{N}(\mu, \sigma^2)$ with mean μ and variance σ^2 estimated from the set of label candidates. In Figure 13, the method is illustrated. This technique is a well known technique in the domain of computer vision with applications in e.g. medical image processing [30] or image segmentation [29].

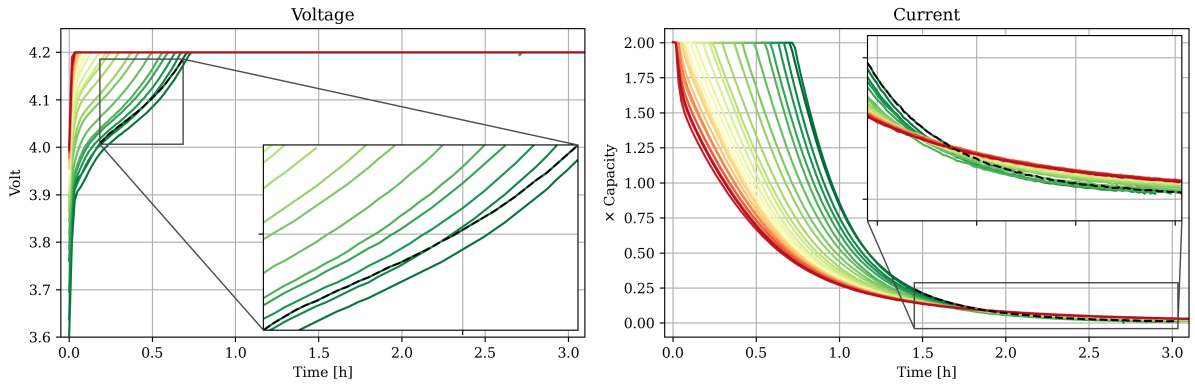


Figure 14: Training data used as input for SoH estimation. All curves in this plot are taken from a single cell out of the NASA dataset. The left plot shows voltage profiles of a charge to full capacity, colored based on the state of health of the battery. A green color indicates that the battery is healthy and still has a high capacity, red lines correspond to cycles closer to the end of life, where the capacity has already significantly degraded. On the right side, the evolution of the current profiles is illustrated. As the cells capacity degrades, the CC segment shortens significantly, and the CV segment increases in length.

5. Experiments

The main topic of this thesis is evaluation of an array of machine learning techniques, applied to battery health estimation and capacity drop off prognostics. In the following section, the problem set up and model choices are explained for two separate problems, namely current state of health estimation and prediction of capacity knee points. The exact model architectures used are detailed in the Appendix.

5.1. Estimation of battery SoH

An accurate estimation of the state of health (SoH) of a battery is of critical importance in many battery use cases. The first section of experiments investigates the feasibility of SoH estimation based on short sections of regulare charges to full capacity. Several different approaches are tested and compared against each other.

5.1.1. Experiment setup

During regular operation, a battery is usually only partially discharged before recharging starts again. Thus, requiring the full data of a complete discharge and subsequent charge to full capacity at a low current would severely limit the opportunities for SoH estimation. Instead of performing lengthy capacity tests every hundred cycles or so, it is much more convenient to use regular operational profiles as a basis for SoH estimation. For this reason, only partial segments of charge data collected during continuous operation and with a variety of currents, are used as model inputs. This enables estimation of SoH much more frequently, as long as a charge to full capacity was performed. The exact data segments used are:

- **CC voltage from 4.0 to 4.192 V:** This consists of only the later half of the CC segment, in addition to the voltage values, also the times of the measurements are used.
- **CV current from 0.2 to 0.01 A:** Also for the CV stage, only the final half of the segment is taken, again paired with time of the measurements.

A sample of the dataset with the input data highlighted is shown in Figure 14. As the SoH decays, the CC stage becomes shorter, while the CV stage lasts longer. It can also be seen that the current curve drops off faster as the battery reaches more advanced stages of degradation.

All of the machine learning algorithms that follow take this data, in some processed form, as their input and then aim to predict the SoH as a value ranging from 0 to 1, representing no remaining capacity and full nominal capacity respectively.

5.1.2. Model architectures

The following experimental setups are evaluated:

- **Autoencoder features:** First an autoencoder consisting of several convolutional layers is trained to reduce the input curves into a small intermediate representation. Based on these extracted embeddings, a feedforward neural network is fitted.
- **Pretrained convolution layers:** The training is split into two phases, pretraining and finetuning. First, a regressor is trained consisting of several convolution layers and a feedforward head. After pretraining, the convolution layers are frozen and a fresh MLP is fitted to the intermediate representations from the convolution layers.
- **Hand-picked features:** For this approach, an array of 16 pre defined mathematical features is used as the model input.

For all of the above architectures, both a single model and an ensemble were trained and evaluated.

5.1.2.1. Autoencoder feature extraction

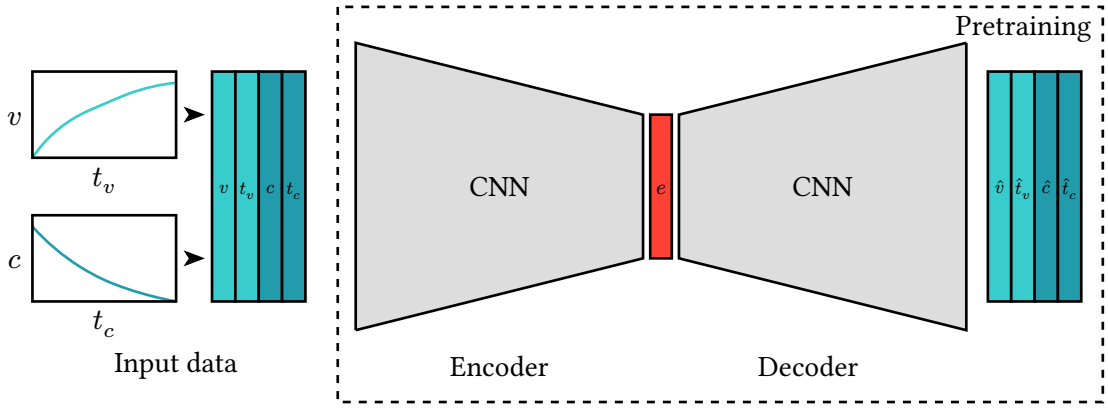


Figure 15: Architecture used in the auto-encoder setting. Instead of training directly on a regression task, the pretraining aims to reconstruct the original input matrix from a reduced representation e. The size of e is considerably smaller than the input matrix. The encoder network is a convolution network and the decoder is essentially a mirrored version of the encoder. Based on the learned representation, an MLP regression head is fitted during fine tuning and the weights of the encoder remain fixed.

The motivation for the use of an autoencoder is the desire to have robust and expressive features derived from the raw data, but without the need for manual feature engineering. Consider inputs x_i belonging to some vector space V with $\dim(V) = n$. The goal is to extract low dimensional embedding vectors $e_i \in \mathbb{R}^r$ out of the raw data x_i, with $r \ll n$. One way to extract such embeddings is by means of two chained neural networks, an **encoder network** $E : V \to \mathbb{R}^r$ that maps an input signal $x \in V$ to a low dimensional representation $e \in \mathbb{R}^r$, and a **decoder network** $D : \mathbb{R}^r \to V$ that reconstructs the original input x from the embedding e. The networks are then trained simultaneously to act as the identity on V. During training, the networks are composed and the loss is given as the MSE of the of the reconstruction:

$$\mathcal{L}(x) = \frac{1}{N} \sum_{i=1}^{N} \|D(E(x_i)) - x_i\|^2. \tag{26}$$

After training, the encoder is used to generate embeddings for all datapoints, which are then fed as inputs into a shallow MLP regressor. For comparison, four different variants of this architectural pattern were considered:

- **Convolutional with pooling layers:** A sequence of 1D convolution kernels applied along the long axis of the inputs. In between these layers, a ReLU activation followed by max pooling is used.
- **Convolutional without pooling:** Instead of applying pooling layers, more convolution layers are sequentially applied.
- **Convolutional with single channel input layers:** Instead of considering kernels that operate on all 4 input channels as the first layer, a preliminary input layer is introduced that applies 1D convolution along each channel separately.
- **Fully connected:** Only as a comparison baseline, a fully connected neural network with tanh activation.

The ReLU activation function is simply defined as $\max(0, x)$. In the experiments, $V = \mathbb{R}^{4\times 64}$ or $V = \mathbb{R}^{4\times 32}$ and the inputs x_i are matrices containing the voltage vector v, current vector c and their corresponding time vectors t_v and t_c as the rows. Since the data is sampled at constant time intervals, the number of measurements that fall into the range of admissible values can vary for different cycles. For this reason,the vectors v, c, t_v and t_c are obtained by evaluating the linear interpolant of the measurements at 64 or 32 evenly spaced points.

5.1.2.2. Pretrained convolution layers

Similar to the two stage training in the autoencoder experiments, this technique also leverages the feature extraction capabilities of convolutional neural networks. Instead of learning an intermediate representation from a reconstruction task, the encoder and head are trained at the same time, directly on the regression objective. After the initial pretraining is done, the encoder layers are frozen, and a freshly initialized regression head is fine tuned on the embeddings. The active weights during training are shown in Figure 16. The architecture of the encoder and head are kept identical to the autoencoder experiment.

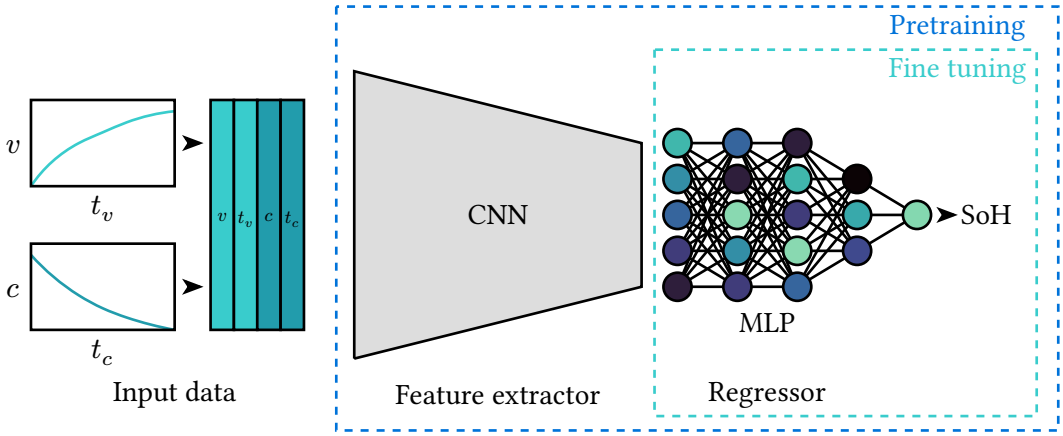


Figure 16: Architecture used in two-phase training regime. Training is performed on the regression objective directly both during pretraining and fine tuning.

5.1.2.3. Custom features

In this approach, 16 mathematical features derived from the voltage and current curves were considered, similar to [23]. These features are not domain specific, like for example peaks of the dQ/dV curve, but rather simple mathematical and statistical properties of the two time series. A set of 8 features was computed for both segments, totaling in 16 input values for the model. For the CC segment, the features are based on the voltage curve, for CV section on the current data. Some of the features consider both the quantity observed as well as the time of each observation, namely the **segment duration**, **numerical integral** and **curve slope**. The rest of the features are computed purely based on the vector values disregarding the time. These are **mean**, **standard deviation**, **kurtosis**, **skewness** and **entropy**. The standard deviation, kurtosis, and skewness are all statistical properties computed from the moments of a distribution [46]. For a set of values $x_i \in \mathbb{R}$, $i = 1, \ldots, N$, the r-th moment is defined as

$$m_r = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^r, \tag{27}$$

where μ is the mean of the x_i. In Table 2, a description of all the features is listed. After computing these features and normalizing, a simple MLP regressor is trained on the extracted features to estimate the current SoH. Besides the change in input dimension, the architecture is identical to the head used in the earlier experiments, see Figure 17.

Feature	Formula
Mean μ	$\frac{1}{N}\sum_{i=1}^{N} x_i$
Standard deviation σ	$\sqrt{\frac{\sum_{i=1}^{N}(x_i-\mu)^2}{N}} = \sqrt{m_2}$
Skewness	$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{x_i-\mu}{\sigma}\right)^3 = \frac{m_3}{m_2^{\frac{3}{2}}}$
Kurtosis	$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{x_i-\mu}{\sigma}\right)^4 = \frac{m_4}{m_2^2}$
Entropy	$-\sum_{i=1}^{N} x_i \log(x_i)$
Average slope	$\frac{1}{N}\sum_{i=1}^{N}\frac{x_{i+1}-x_{i-1}}{2\Delta t}$
Numerical integral (Simpson rule)	$\frac{\Delta t}{3}\sum_{i=1}^{\frac{N}{2}}(x_{2i-1}+4x_{2i}+x_{2i+1})$
Segment duration	$t_N - t_1$

Table 2: Custom features for SoH estimation. The symbol Δt denotes the sampling interval of the measurements. All of these features are computed for both sequences, the CC voltages and the CV currents.

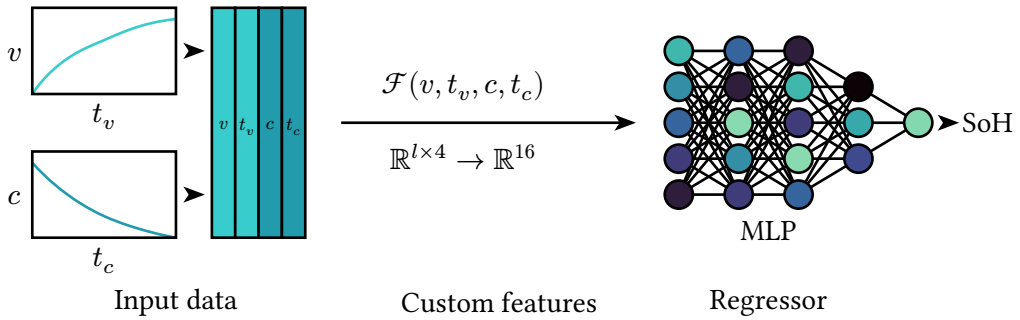


Figure 17: Architecture used for custom features. The mapping $\mathcal{F}$ transforms the measurement vectors v, t_v, c, t_c with length l into a single column vector. Note that different datapoints can have varying numbers of rows l, but this does not affect the output shape.

5.1.3. Ensemble learning

To increase the robustness of machine learning systems one can train many networks in parallel and then aggregate their predictions. This practice is called ensemble learning and a strategy to further boost model performance and generalization capabilities. For all of the above strategies, ensembles of multiple identical neural networks with different initialization were trained to compare against the

single models. There are several strategies to compute a final prediction from the individual ensemble outputs. One could simply pick the model that scored the lowest validation loss at the end of training, but disregarding all other predictions is quite wasteful. Another more sensible way is to average the predictions, either with equal weights or weighted by their validation performance. Assume an ensemble of N models, and let $\hat{y} \in \mathbb{R}^N$ be their predictions of some target value $y \in \mathbb{R}$. Then the predictions of the ensemble can be modeled as samples of a random variable $Y \sim \mathcal{N}(\mu, \sigma^2)$, with mean μ and unbiased sample variance [47] σ^2 computed as

$$\mu = \frac{1}{N} \sum_{i=1}^{N} \hat{y}_i \quad \text{and} \quad \sigma^2 = \sum_{i=1}^{N} \frac{(y_i - \mu)^2}{N-1}. \tag{28}$$

Using this interpretation, the standard deviation σ provides a measure of confidence in the mean prediction μ.

5.1.4. Prediction postprocessing

In order to further improve the prediction quality after training, an output postprocessing step can be applied. By averaging the past k predictions, oscillations or outliers at inference time can be partially dealt with. In the experiments, several different weighting schemes for tha past predictions are evaluated and an optimal window size k is determined for each scheme.

5.2. Knee point prognostics

Besides knowing of the current state of health of a battery, the estimation of the future evolution of this state of health is also of interest. Part of this thesis investigates the prediction of the knee point of a battery capacity curve. Two different scenarios are considered. For the first setting, the model is asked to directly predict the remaining cycles r until the Bacon-Watts knee point is reached. In the second part of the prognostics experiments, the labeling uncertainty of the knee point is encoded into the regression target and the model has to predict both μ_r and σ_r of the Gaussian density centered at the knee point location. Using ensemble learning, a Gaussian mixture distribution is constructed from the model outputs in order to provide more informative predictions.

5.2.1. Experiment setup

In contrast to the SoH estimation experiments, where the models were tasked to estimate based on very limited segments of charge data, for knee point prediction the entire profile of a cycle is used as model input. As an additional data dimension, the temperature curve is added as well. Besides the consequence that the custom features from Table 2 are no longer useful, there is still the issue that the segments have different lengths. Similar as in Section 5.1.2.1, a convolutional autoencoder is trained on the task of reconstructing interpolated voltage, charge, temperature and time vectors. In particular, inputs are of size 4×512, corresponding to a resolution of 512 evaluations of the interpolant at equidistant points. After the unsupervised autoencoder training, the obtained sequence embeddings x_i are used as input to a fully connected MLP regression head. During the training of this second model, the weights of the encoder are kept frozen. As already discussed in Section 4.2.3, two different types of regression targets are evaluated. In the first case, where the number of remaining cycles $r^* \in \mathbb{N}$ is used as the label, the models are asked to predict only a single value. The loss for these experiments is standard MSE loss, so given a neural network model $f_\theta(x)$, the loss is computed from a batch of embeddings x_i and targets r_i^*, $i = 1, ..., N_{\text{batch}}$ via

$$\mathcal{L}_{\text{point}}(\theta) = \frac{1}{N_{\text{batch}}} \sum_{i=1}^{N_{\text{batch}}} (f_\theta(x_i) - r_i^*)^2. \tag{29}$$

In the case of gaussian labels μ^* and σ^*, the model is again asked to predict two values, the distance μ^* of the current cycle to the knee point, and also the corresponding standard deviation σ^* of the three labels. Again, the MSE loss function is used

$$\mathcal{L}_{\text{heatmap}}(\theta) = \frac{1}{N_{\text{batch}}} \sum_{i=1}^{N_{\text{batch}}} \| f_\theta(x_i) - \begin{pmatrix} \mu_i^* \\ \sigma_i^* \end{pmatrix} \|^2. \tag{30}$$

When considering an ensemble of n models, an additional modeling choice presents itself. Since the model produces n different means μ_k and standard deviations σ_k, the outputs need to be combined to arrive at a single output distribution. Three different ways are considered for modeling the final distribution [48]:

- **Normal distribution with averaged parameters:**

$$W \sim \mathcal{N}(\overline{\mu}, \overline{\sigma}^2) \quad \text{with} \quad \overline{\mu} = \frac{1}{n} \sum_k \mu_k \quad \text{and} \quad \overline{\sigma} = \frac{1}{n} \sum_k \sigma_k. \tag{31}$$

- **Mean of n Gaussian random variables:**

$$Y = \frac{1}{n} \sum_k X_k \quad \text{with} \quad X_k \sim \mathcal{N}(\mu_k, \sigma_k^2). \tag{32}$$

This means that $Y \sim \mathcal{N}(\overline{\mu}, \hat{\sigma}^2)$ with $\overline{\mu} = \frac{1}{n} \sum_k \mu_k$ and $\hat{\sigma}^2 = \frac{1}{n^2} \sum_k \sigma_k^2$.

- **Mixture distribution of n Gaussians:**

$$Z = X_k \quad \text{with} \quad k \sim \mathcal{U}(\{1, ..., n\}) \quad \text{and} \quad X_k \sim \mathcal{N}(\mu_k, \sigma_k^2). \tag{33}$$

In general, the density of Z for a Gaussian mixture model is a convex combination of the individual densities of the X_k. In the particular case of equal weights, this amounts to $f_Z(x) = \frac{1}{n} \sum_k f_{X_k}(x)$.

All three of the above options lead to a different output distributions for the ensemble, as illustrated in Figure 18. The resulting density function can then be interpreted to infer the location of the knee point, as well as the models confidence in this location. While the mean prediction $\overline{\mu} = \sum_k \mu_k$ is the same for the three methods, they have different variance. These differences will be further discussed in Section 6.2.4. Note that if all ensemble members output the same μ and σ, then $X_k = X_1$ for all k and consequently W and Z are equivalent:

$$f_Z(x) = \frac{1}{n} \sum_k f_{X_k}(x) = f_{X_1}(x) = f_W(x) \tag{34}$$

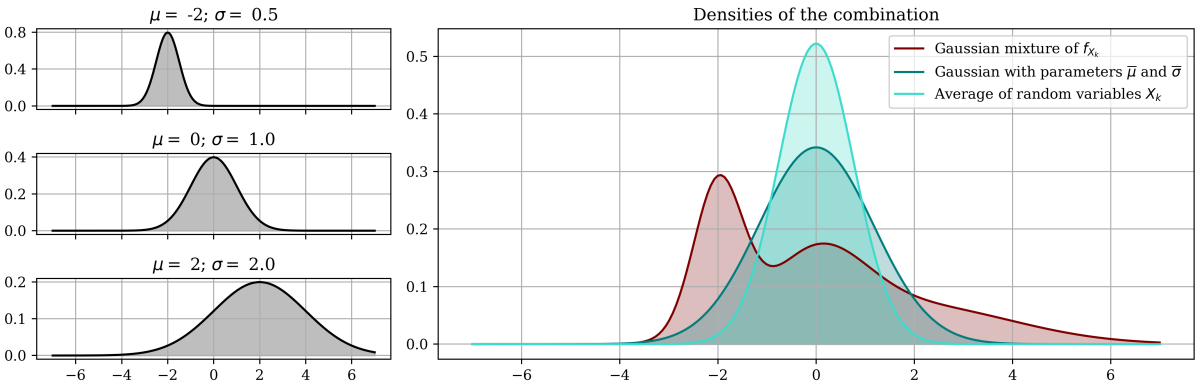


Figure 18: Comparison of the resulting densities for the three different combination options. The left plots show the individual densities, while the right side compares the resulting distributions for each method.

6. Results and Discussion

6.1. SoH estimator comparison

A large part of the SoH estimator performance stems from the quality of the input features for the regression head. Because of this, some preliminary experiments are conducted to evaluate feature extraction methods before moving on to the regression objective. The following section will first present the results of a series of experiment concerning unsupervised signal reconstruction. Based on the results of this task, a suitable encoder architecture is chosen. In the subsequent section, the performance of the most promising approaches is evaluated for SoH estimation, and compared against a model trained on the mathematical features listed in Table 2. The final parts of the SoH results discussion will showcase methods for improving estimation performance by means of ensemble learning and further post-processing after training.

6.1.1. Autoencoder performance

Before training the regression heads, an encoder architecture for the automatic feature extraction needs to be determined. For this purpose, a variety of autoencoder architectures were trained on the combined dataset of XJTU and NASA. Training was performed using 4-fold cross validation splitting, this amounts to partitioning the data into 4 sets and training on three of them while using the fourth as holdout data. For each split, three separate runs were preformed with different random initialization. The result of this comparison in terms or MAE and RMSE are given in Table 3 and Figure 20, averaged over all runs and splits. All subsequent experiments use the best performing encoder architectures in this preliminary evaluation. The chosen architectures were evaluated on the downstream task of estimating SoH, either based directly on the embeddings obtained from unsupervised training, or by fitting new encoders of the same design, using the two step pretraining/finetuning method.

The best performing approach is the convolutional autoencoder with ReLU activation, max pooling layers and a single channel convolution layer in the beginning (CNN ReLU SCC). The inclusion of this single channel as the first layer of the encoder is motivated by qualitatively observing the reconstruction performance of the different architectures, see Figure 19. While the convolutional architecture with ReLU and max pooling (CNN ReLU Pooling) achieves high scores, its reconstructions have a distinct staircase pattern. Adding one-dimensional convolution layers that operate only on a single

Model architecture	RMSE holdout	RMSE train	MAE holdout	MAE train
CNN ReLU SCC 32	$\mathbf{6.093} \pm 3.439$	$\mathbf{4.385} \pm 1.385$	$\mathbf{2.227} \pm 0.842$	$\mathbf{2.105} \pm 0.824$
CNN ReLU Deep 32	$6.611 \pm \mathbf{1.696}$	5.320 ± 0.772	2.950 ± 0.716	2.770 ± 0.686
CNN ReLU SCC 10	6.818 ± 2.882	5.346 ± 1.760	2.593 ± 0.689	2.482 ± 0.648
CNN ReLU Pooling 32	6.965 ± 2.190	$5.978 \pm \mathbf{0.298}$	$3.085 \pm \mathbf{0.278}$	$3.004 \pm \mathbf{0.231}$
CNN ReLU Deep 10	7.469 ± 2.817	5.763 ± 0.883	3.214 ± 0.659	3.020 ± 0.661
CNN ReLU Pooling 10	8.153 ± 2.418	6.965 ± 1.182	3.728 ± 0.967	3.595 ± 0.967
CNN ReLU SCC 16	8.406 ± 4.991	6.420 ± 3.483	3.223 ± 2.560	3.063 ± 2.481
CNN ReLU Pooling 16	8.719 ± 5.070	6.986 ± 2.158	3.606 ± 1.174	3.512 ± 1.100
CNN ReLU Deep 16	8.892 ± 3.922	7.540 ± 3.486	3.729 ± 2.016	3.533 ± 1.944
FC tanh; $4 \times 32 \to 8$	30.116 ± 30.692	18.600 ± 8.843	8.439 ± 2.048	8.137 ± 2.001
FC tanh; $4 \times 64 \to 8$	46.444 ± 36.680	27.359 ± 18.085	7.668 ± 2.337	7.207 ± 2.417
FC tanh; $4 \times 64 \to 10$	52.264 ± 38.483	26.606 ± 13.937	7.574 ± 1.715	7.031 ± 1.795

Table 3: Comparison of the different autoencoder architectures for CC-CV embedding. Arranged in descending order based on the mean RMSE on the holdout. The number written after the model name is the embedding dimension, all metrics are on the scale 10^{-3}. For each architecture, the metrics are stated as the mean of all 12 training runs, with the corresponding standard deviation.

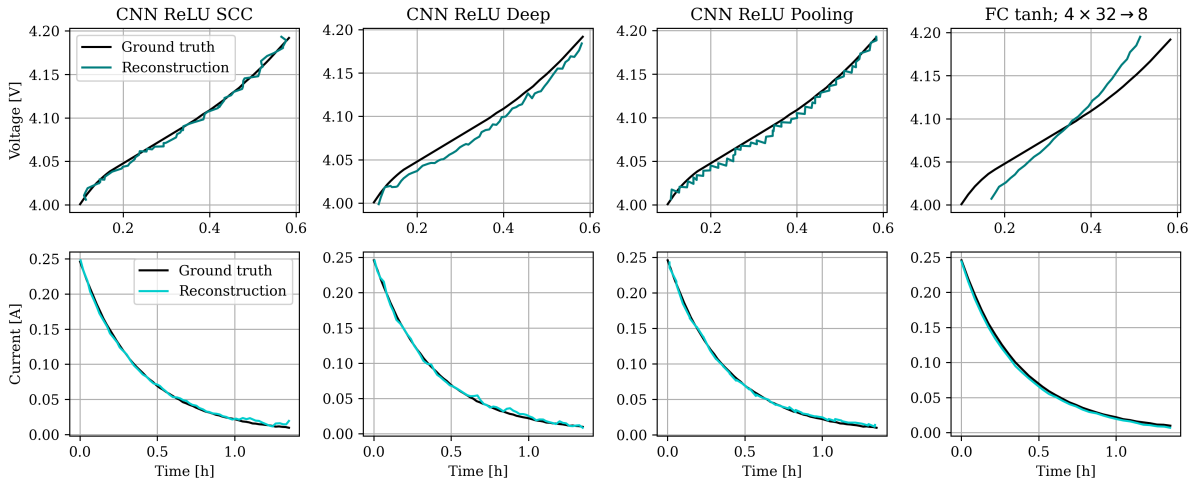


Figure 19: Examples of autoencoder reconstruction performance. The samples shown are taken from the best models of all splits, evaluated on holdout data. It is notable that the fully connected model has smoother predictions, however it often fails in reconstructing the voltage curve.

channel of the data, enables the network to overcome this sawtooth pattern. To make an informed choice of embedding size, several dimensions were experimented with and ultimately the size 10 embeddings as well as size 32 were chosen. The performance of different embedding sizes is plotted in Figure 20 a), and 10 appears to be a local minimum among the smallest, while 32 is a larger size that still offers a significant reduction of the input dimension.

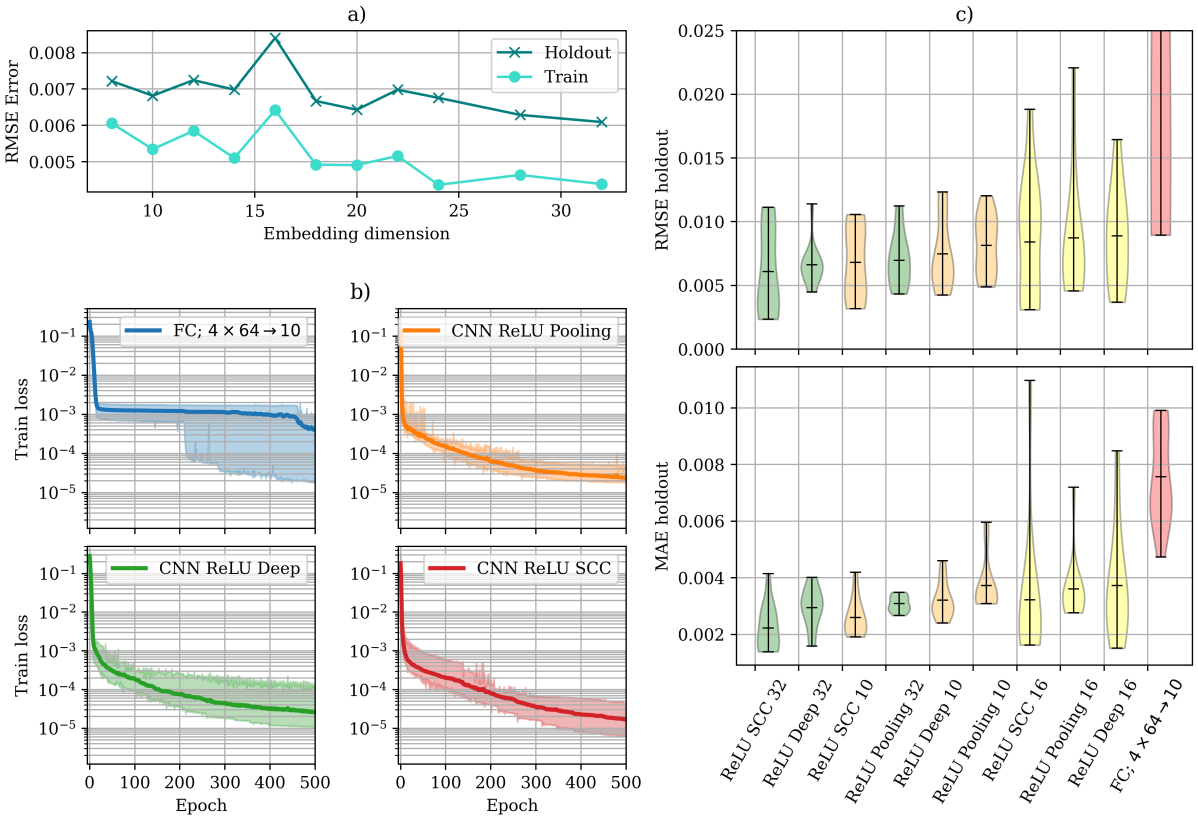


Figure 20: a) Performance on holdout data for the CNN ReLU SCC architecture for various embedding sizes. The graph shows a trend of increased reconstruction accuracy as the embedding dimension increases. **b)** Training loss curves of the three different autoencoder architectures with embedding size 10. The mean of all training runs is plotted as a solid line, while all losses were inside the shaded area. **c)** Error distributions over all training runs for CNN based approaches for selected embedding sizes. Color coded according to Table 3.

Fully connected architectures were not further pursued, they exhibit much worse error distribution than all other models. A surprising observation is that even though the reconstructions of the fully connected (FC) architecture often look visually the smoothest, if performs considerably worse in terms of error metrics. For the FC model, non-tanh activations perform even worse and are thus omitted from the discussion. Ultimately the CNN ReLU SCC architecture outperformed all other methods in an otherwise equal setting.

It is interesting to see how the learned features from the autoencoder relate to the engineered features. On the folwoing page, in Table 4, this is shown for the CNN ReLU SCC autoencoder with a size 10 embedding. While there are some obvious correlations with duration of the segments and the mean of the current sequence, none of the embedding dimensions show a particular correlation with the SoH. The situation is remarkably different for the embeddings obtained via pretraining, as shown in Table 5. Almost all of the features are to different degrees correlated with the capacity of the cell. The pretraining on the regression objective evidently produces embeddings that are more connected to the SoH, when compared with the autoencoder embeddings based on signal reconstruction.

	1	2	3	4	5	6	7	8	9	10
Capacity	0.437	0.486	−0.187	−0.214	−0.543	0.050	0.090	−0.281	−0.573	−0.029
Duration_c	0.262	−0.979	0.472	0.953	0.968	0.778	0.709	0.978	0.632	−0.781
Entropy_c	0.383	0.415	−0.560	−0.141	−0.385	0.039	0.138	−0.204	−0.642	−0.138
Integral_c	−0.187	−0.931	0.643	0.699	0.947	0.413	0.316	0.766	0.876	−0.412
Kurtosis_c	−0.464	−0.757	0.698	0.428	0.801	0.102	−0.003	0.508	0.909	−0.155
Skewness_c	−0.617	−0.589	0.774	0.209	0.634	−0.114	−0.224	0.293	0.888	0.052
Slope_c	−0.478	−0.690	0.763	0.345	0.703	0.050	−0.063	0.427	0.893	−0.033
μ_c	0.417	0.673	−0.867	−0.368	−0.691	−0.087	0.029	−0.435	−0.859	0.139
σ_c	−0.335	−0.196	0.707	−0.001	0.194	−0.134	−0.214	0.029	0.446	0.087
Duration_v	0.814	−0.648	0.112	0.928	0.589	0.999	0.992	0.886	0.045	−0.977
Entropy_v	0.557	0.589	−0.610	−0.219	−0.599	0.074	0.178	−0.301	−0.822	−0.115
Integral_v	0.557	−0.829	0.436	0.948	0.762	0.916	0.854	0.941	0.397	−0.858
Kurtosis_v	−0.392	−0.436	0.174	0.184	0.474	−0.047	−0.080	0.243	0.507	0.026
Skewness_v	0.338	0.300	0.169	−0.107	−0.366	0.096	0.077	−0.151	−0.275	−0.033
Slope_v	0.276	0.523	−0.743	−0.290	−0.502	−0.116	−0.010	−0.351	−0.712	0.066
μ_v	−0.438	−0.436	0.018	0.167	0.494	−0.091	−0.100	0.226	0.461	0.043
σ_v	0.490	0.569	−0.376	−0.239	−0.596	0.044	0.107	−0.312	−0.691	−0.042

Table 4: Correlation of the size 10 embeddings learned by the autoencoder CNN ReLU SCC with the custom features, colored by magnitude. It can be seen that the learned features do capture certain aspects of the custom features, for example Nr. 6 almost perfectly tracks the duration of the CC segment, while Nr. 2 is strongly negatively correlated with the duration of the CV part. Not all cases are so clear, for example Nr. 3 exhibits a considerable negative correlation with the mean of CV currents, but is otherwise harder to interpret. A general concentration of focus on the duration of the CC segment is apparent. Notably, none of the features by themselves are strongly correlated to capacity.

	1	2	3	4	5	6	7	8	9	10
Capacity	−0.831	−0.848	0.889	−0.715	0.875	−0.811	−0.892	0.593	−0.869	−0.478
Duration_c	0.615	0.354	−0.420	−0.125	−0.622	0.771	0.359	−0.713	0.318	−0.471
Entropy_c	−0.537	−0.490	0.397	−0.305	0.407	−0.243	−0.153	−0.126	−0.420	0.030
Integral_c	0.849	0.684	−0.700	0.281	−0.810	0.822	0.576	−0.549	0.643	−0.148
Kurtosis_c	0.752	0.670	−0.696	0.405	−0.702	0.649	0.573	−0.368	0.663	0.097
Skewness_c	0.645	0.629	−0.643	0.468	−0.581	0.473	0.501	−0.169	0.627	0.239
Slope_c	0.824	0.773	−0.774	0.529	−0.777	0.660	0.606	−0.325	0.745	0.204
μ_c	−0.556	−0.484	0.515	−0.257	0.498	−0.443	−0.364	0.206	−0.471	−0.045
σ_c	0.072	0.092	−0.106	0.068	−0.020	−0.047	−0.004	0.162	0.084	0.087
Duration_v	0.062	−0.235	0.144	−0.638	−0.125	0.384	−0.128	−0.609	−0.263	−0.769
Entropy_v	−0.780	−0.749	0.696	−0.537	0.678	−0.538	−0.506	0.131	−0.705	−0.158
Integral_v	0.323	0.028	−0.081	−0.445	−0.328	0.493	−0.011	−0.539	−0.033	−0.702
Kurtosis_v	0.611	0.599	−0.570	0.452	−0.563	0.535	0.547	−0.251	0.582	0.137
Skewness_v	−0.551	−0.576	0.570	−0.511	0.547	−0.575	−0.649	0.363	−0.596	−0.244
Slope_v	−0.517	−0.436	0.396	−0.227	0.433	−0.310	−0.168	0.024	−0.394	0.052
μ_v	0.703	0.713	−0.701	0.602	−0.677	0.666	0.713	−0.376	0.726	0.267
σ_v	−0.763	−0.736	0.704	−0.548	0.692	−0.618	−0.595	0.256	−0.716	−0.171

Table 5: Correlation of embeddings learned via supervised pretraining with the custom features, colored by magnitude. The encoder architecture is the same as in Table 4, but used in a different training scenario. Compared to the autoencoder embeddings, almost all dimensions correlate quite strongly with capacity.

6.1.2. Regression performance

In the comparison of SoH regressor, there are five main metrics of interest, RMSE, MAE, MAPE, R^2 score and the percentage of absolute errors inside a 5% margin. To ensure a fair comparison, all methods were trained on the same 4-fold splits of the data with matching train and holdout data for each split. In total, 32 models were trained based on various encoder architectures. Throughout this section, only the results of the best architectures are listed and compared against models trained on the features from Table 2. The three categories consist of models trained on the **unsupervised, autoencoder extracted features**, models obtained during **pretraining** and finally the **finetuned** models, based on the embeddings extracted by the pretrained encoders. The custom mathematical features outperformed the learned features by a significant margin in the single model setting. The detailed results are stated in Table 6. As suggested in Section 6.1.1, the CNN ReLU SCC architectures come closest to the custom features. Consistent with the observations from Table 5, the feature extractors obtained via pretraining seem to offer more informative embeddings for the specific task of SoH regression. Finetuning on a pretrained encoder with embedding size 32 shows the best results, while the autoencoder embeddings surprisingly are most effective with the lower dimension of 16. The autoencoders are in general worse when compared to finetuned models across all architectures, as apparent in Figure 21. However, only one of the models obtained during pretraining was able to beat the best autoencoder approach.

Model architecture	**R^2 holdout**	**R^2 train**	**RMSE holdout**	**RMSE train**
Custom Features 16	**0.821** $\pm$ 0.040	**0.940** $\pm$ 0.007	**3.142** $\pm$ 0.290	**1.838** $\pm$ 0.102
Finetune SCC 32	0.811 $\pm$ 0.047	0.928 $\pm$ **0.006**	3.218 $\pm$ 0.345	2.020 $\pm$ **0.084**
Pretraining SCC 32	0.788 $\pm$ **0.024**	0.897 $\pm$ 0.021	3.440 $\pm$ **0.168**	2.405 $\pm$ 0.243
Autoencoder SCC 16	0.781 $\pm$ 0.034	0.882 $\pm$ 0.023	3.491 $\pm$ 0.224	2.566 $\pm$ 0.242
Model architecture	**MAE holdout**	**MAE train**	**MAPE holdout**	**MAPE train**
Custom Features 16	**2.108** $\pm$ 0.176	**1.300** $\pm$ 0.115	**2.324** $\pm$ 0.176	**1.441** $\pm$ 0.118
Finetune SCC 32	2.149 $\pm$ 0.197	1.406 $\pm$ **0.101**	2.378 $\pm$ 0.185	1.565 $\pm$ **0.102**
Pretraining SCC 32	2.331 $\pm$ 0.147	1.702 $\pm$ 0.189	2.578 $\pm$ 0.139	1.890 $\pm$ 0.200
Autoencoder SCC 16	2.368 $\pm$ **0.140**	1.793 $\pm$ 0.183	2.620 $\pm$ **0.136**	1.994 $\pm$ 0.203
Model architecture	**Error < 5% holdout**	**Error < 5% train**	**Error < 5% total**	
Custom Features 16	**0.877** $\pm$ 0.035	**0.964** $\pm$ 0.008	**0.943** $\pm$ 0.011	
Finetune SCC 32	0.869 $\pm$ **0.026**	0.954 $\pm$ **0.004**	0.933 $\pm$ **0.008**	
Pretraining SCC 32	0.851 $\pm$ 0.028	0.921 $\pm$ 0.022	0.904 $\pm$ 0.017	
Autoencoder SCC 16	0.847 $\pm$ 0.034	0.914 $\pm$ 0.015	0.897 $\pm$ 0.017	

Table 6: SoH estimation performance metrics for the single regression models. RMSE, MAE and MAPE values are on the scale of 10^{-2}.

It is also interesting to asses the performance of the model trained on only one of the datasets, NASA or XJTU. Table 7 describes the results of training the top two architectures on a single dataset only, while Table 8 shows how the models trained on the merged dataset perform on the subsets NASA and XJTU. For the custom features, both methods perform virtually identical in either setting, with slightly improved R^2 and RMSE scores for the merged data model. The MAE, MAPE and the ratio of errors inside the 5% margin all marginally decrease as more data is added, but differences are small overall. Across all metrics, the variance is lower for the model trained on more data.

For the extracted features, results differ a bit more between the different data configurations. On the XJTU cells, the merged data model also scores higher R^2 and RMSE and the other metrics experience a small decrease. Additionally, the model trained on the full dataset has again reduced variance in all metrics. On the NASA data, being the smaller dataset, the predictions are less accurate with more

training data, when compared with the specialized model. However, since the metric values are so close, the results are deemed inconclusive and will not be discussed further.

Using only NASA data			
Model architecture	R^2		**RMSE**
Custom Features 16	0.760 ± 0.228		3.721 ± 1.723
Finetune SCC 32	0.742 ± 0.156		4.042 ± 1.197
Model architecture	**MAE**	**MAPE**	**Error < 5%**
Custom Features 16	2.600 ± 1.076	3.054 ± 1.163	0.819 ± 0.106
Finetune SCC 32	2.854 ± 0.802	3.360 ± 0.856	0.799 ± 0.081

Using only XJTU data			
Model architecture	**R2 holdout**		**RMSE holdout**
Custom Features 16	0.664 ± 0.154		2.736 ± 0.801
Finetune SCC 32	0.595 ± 0.194		3.024 ± 0.850
Model architecture	**MAE**	**MAPE**	**Error < 5%**
Custom Features 16	1.857 ± 0.483	1.972 ± 0.506	0.917 ± 0.051
Finetune SCC 32	1.836 ± 0.523	1.938 ± 0.527	0.899 ± 0.073

Table 7: Holdout metrics for models trained only on either NASA or XJTU data. RMSE, MAE, and MAPE on the scale of 10^{-2}.

Trained on merged data, evaluated on NASA			
Model architecture	R^2		**RMSE**
Custom Features 16	0.772 ± 0.176		3.711 ± 1.366
Finetune SCC 32	0.718 ± 0.177		4.182 ± 1.322
Model architecture	**MAE**	**MAPE**	**Error < 5%**
Custom Features 16	2.625 ± 0.884	3.086 ± 0.958	0.815 ± 0.093
Finetune SCC 32	3.036 ± 0.936	3.560 ± 1.020	0.768 ± 0.097

Trained on merged data, evaluated on XJTU			
Model architecture	R^2		**RMSE**
Custom Features 16	0.676 ± 0.131		2.721 ± 0.634
Finetune SCC 32	0.622 ± 0.149		2.944 ± 0.674
Model architecture	**MAE**	**MAPE**	**Error < 5%**
Custom Features 16	1.906 ± 0.416	2.015 ± 0.430	0.903 ± 0.070
Finetune SCC 32	1.908 ± 0.429	2.015 ± 0.432	0.889 ± 0.072

Table 8: Holdout metrics for models trained on the entire dataset, evaluated on either NASA or XJTU. RMSE, MAE, and MAPE on the scale of 10^{-2}

6.1.3. Ensemble performance

By means of ensemble learning, the holdout performance can be improved further in every metric. For each of the feature extraction methods, a group of 50 models was trained simultaneously with different initialization. In all cases, the ensemble members are fully connected MLPs with two hidden layers of width 16. The mean predictions of the ensemble consistently improved evaluation scores, and additionally provided a measure of prediction uncertainty by estimating the standard deviation of the ensemble outputs. The metrics for the ensembles are compared in Table 9, computed for the respective mean predictions. As a basis for the ensemble finetune, the pretrained encoders from the single model experiments were used, and an ensemble of MLP heads was trained. When evaluating strictly the ensembles, the top finetuned architecture based on the ReLU SCC embeddings of size 32 is essentially able to match the performance of the custom features. The differences in the metrics are well inside the standard deviation range of the respective values. In Figure 21, the performance of the autoencoders is visualized.

Model architecture	**R^2 holdout**	**R^2 train**	**RMSE holdout**	**RMSE train**
50x Custom Features 16	$\mathbf{0.841 \pm 0.035}$	$\mathbf{0.943} \pm 0.006$	$\mathbf{2.961} \pm 0.281$	$\mathbf{1.802} \pm 0.086$
50x Finetune SCC 32	0.834 ± 0.049	$0.936 \pm \mathbf{0.003}$	3.006 ± 0.392	$1.896 \pm \mathbf{0.054}$
50x Autoencoder SCC 16	0.811 ± 0.038	0.892 ± 0.012	$3.227 \pm \mathbf{0.271}$	2.461 ± 0.139
Model architecture	**MAE holdout**	**MAE train**	**MAPE holdout**	**MAPE train**
50x Custom Features 16	$\mathbf{1.960} \pm 0.190$	$\mathbf{1.246} \pm 0.088$	$\mathbf{2.158} \pm 0.190$	$\mathbf{1.381} \pm 0.088$
50x Finetune SCC 32	1.994 ± 0.218	$1.284 \pm \mathbf{0.061}$	2.206 ± 0.204	$1.430 \pm \mathbf{0.056}$
50x Autoencoder SCC 16	$2.192 \pm \mathbf{0.142}$	1.704 ± 0.103	$2.424 \pm \mathbf{0.123}$	1.896 ± 0.108
Model architecture	**Error < 5% holdout**	**Error < 5% train**	**Error < 5% total**	
50x Custom Features 16	$\mathbf{0.881} \pm 0.031$	$\mathbf{0.962} \pm 0.005$	$\mathbf{0.942} \pm 0.007$	
50x Finetune SCC 32	$0.880 \pm \mathbf{0.029}$	$0.958 \pm \mathbf{0.002}$	$0.939 \pm \mathbf{0.006}$	
50x Autoencoder SCC 16	0.857 ± 0.039	0.917 ± 0.011	0.902 ± 0.013	

Table 9: SoH estimation performance metrics for the ensembles of 50 models. RMSE, MAE and MAPE values are on the scale of 10^{-2}.

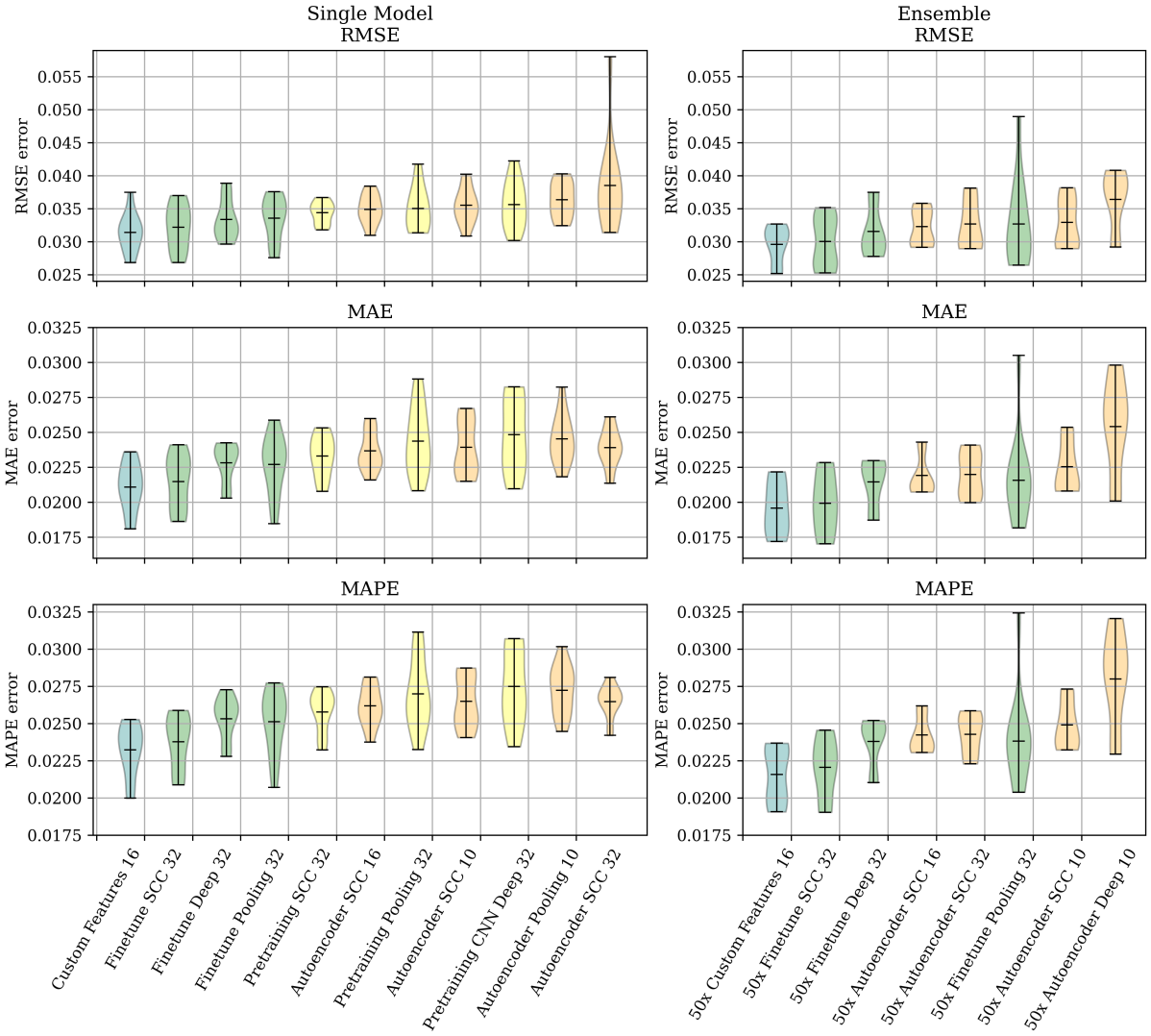


Figure 21: SoH estimation performance of the ensemble and single regressors visualized. The distributions are estimated from the 12 separate training runs performed for each architecture.

6.1.4. Prediction postprocessing

For the two best ensembles, based on custom features and the size 32 ReLU SCC pre-trained embeddings, prediction postprocessing was evaluated. By leveraging not just the current, but the k latest predictions, it is possible to increase the accuracy of the estimates, reducing oscillations and outliers. In particular, given the past k model outputs y_i, with y_0 the oldest and y_{k-1} the latest, and weights $w_i \in [0, 1]$ with $i = 0, 1, ..., k-1$ such that $\sum_i w_i = 1$, the final prediction y_{post} is computed as

$$y_{\text{post}} = \sum_{i=0}^{k-1} w_i y_i \tag{35}$$

Five different weighting strategies were considered:

- **Constant equal weights:**

$$\hat{w}_i = 1 \tag{36}$$

- **Logarithmic weights:**

$$\hat{w}_i = \ln\left(e + (20 - e)\frac{i}{k-1}\right) \tag{37}$$

- **Linear weights:**

$$\hat{w}_i = i + 1 \tag{38}$$

- **Gaussian weights:**

$$\hat{w}_i = \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-\left(2 - \frac{2i}{k-1}\right)^2}{2}\right) \tag{39}$$

- **Exponential weights:**

$$\hat{w}_i = \exp\left(-2 + \frac{2i}{k-1}\right) \tag{40}$$

For the final coefficients, the above values $\hat{w}_i$ are normalized

$$w_i = \frac{\hat{w}_i}{\sum_i \hat{w}_i}. \tag{41}$$

All of the above weighting schemes lead to slightly different resulting curves. Since at least k predictions are needed to compute the weighted average, for the first $k-1$ cycles just the raw estimates are used. In Figure 22, the visual differences and smoothing effects of weights (36) to (40) are compared.

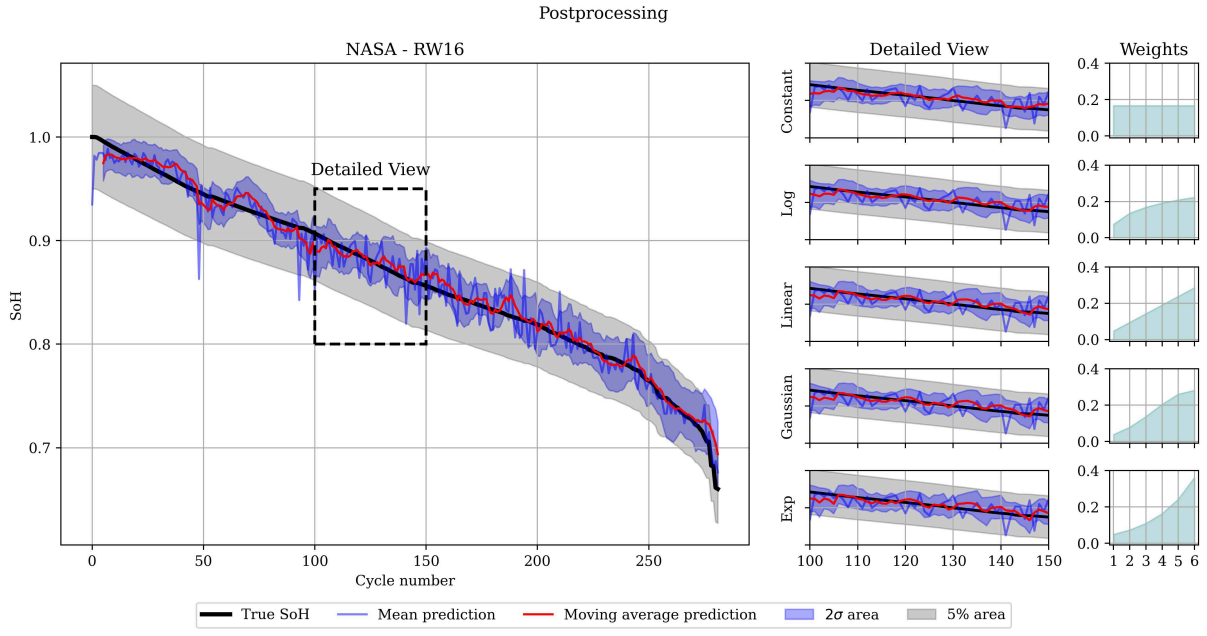


Figure 22: Smoothing effect of different weighting strategies using the past 6 estimates of the custom feature based ensemble, evaluated on a holdout curve.

In Figure 23, the performance of the different strategies is compared on the holdout splits, showing the clear advantage of prediction averaging over just using the raw model outputs. The equally weighted average of the past 6 predictions appears to be the best strategy. With this smoothing of the model

outputs, close to 90% of all predictions fall into the 5% range of the true SoH, see Table 10. Both the custom and learned features benefit from the post-processing to a similar degree.

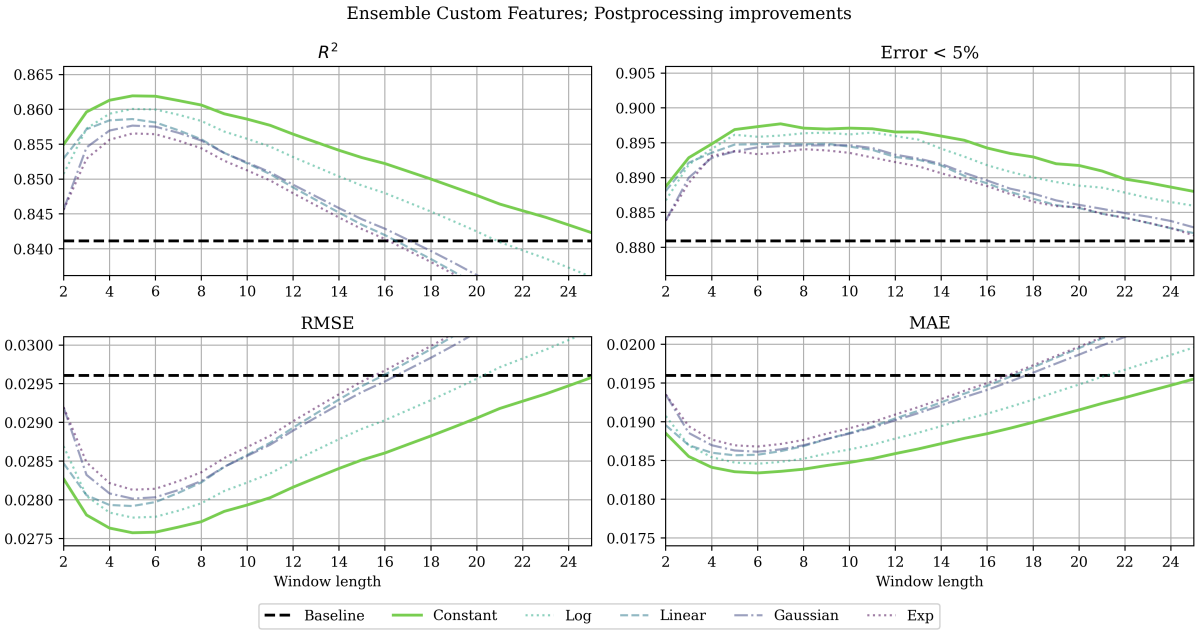


Figure 23: Improvements of SoH predictions, depending on the window length of the running average. Different curves represent the various weighting schemes and all methods were evaluated on the holdout data. Based on the extreme points of the graphs, the final window size is chosen as 6, striking a balance over all metrics. Lower is better for the bottom two plots and higher is better for the upper two.

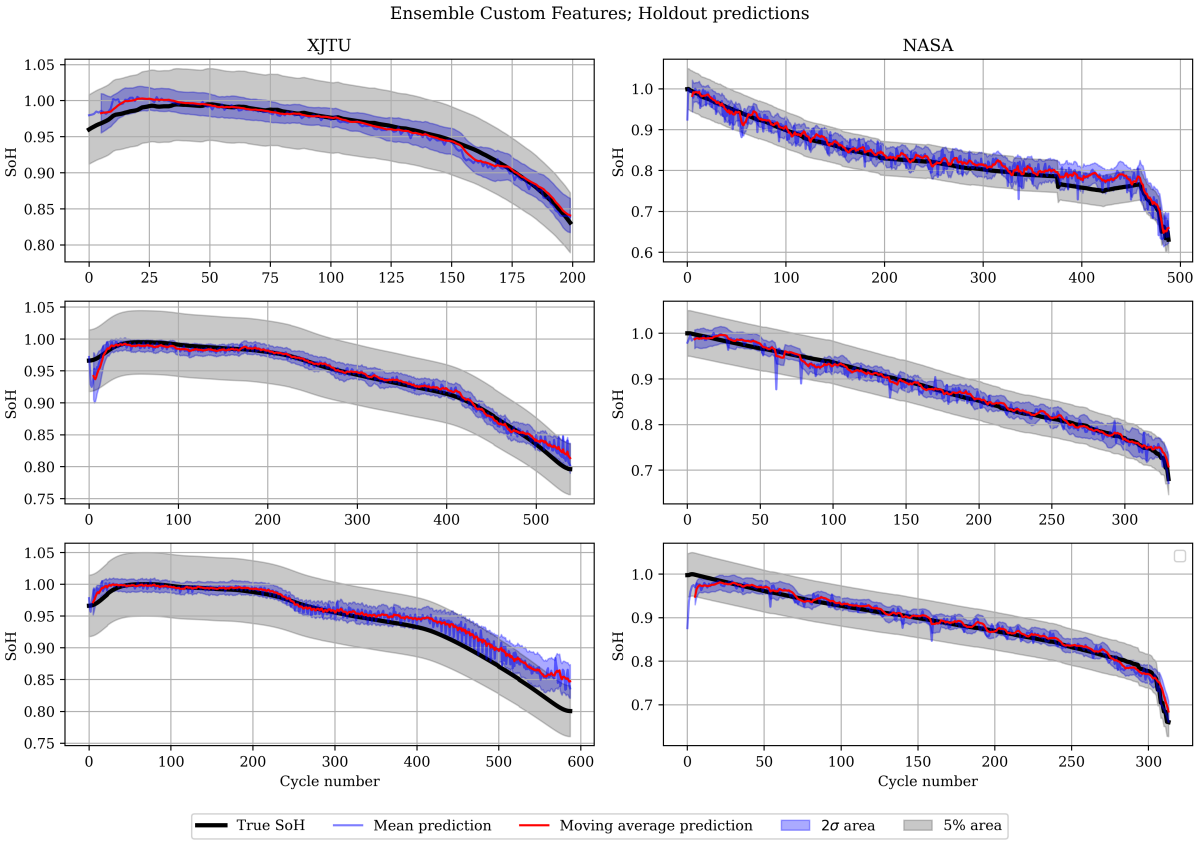


Figure 24: SoH predictions of the feature based ensemble for a sample of representative curves out of the holdout data.

Model architecture	R^2 holdout	RMSE holdout	MAE Holdout	Error < 5% holdout
50x Custom Features 16	0.861 ± 0.030	2.758 ± 0.267	1.833 ± 0.213	0.897 ± 0.020
50x Finetune SCC 32	0.849 ± 0.034	2.877 ± 0.287	1.918 ± 0.197	0.889 ± 0.028
Improvements over raw predictions				
50x Custom Features 16	0.020 ± 0.004	-0.202 ± 0.030	-0.125 ± 0.041	0.016 ± 0.014
50x Finetune SCC 32	0.015 ± 0.015	-0.129 ± 0.121	-0.075 ± 0.087	0.009 ± 0.016

Table 10: SoH prediction performance metrics and improvements for the top two architectures, using an average of the past 6 model outputs. Compared against the baseline of the raw, unprocessed estimates. All figures for holdout data. RMSE and MAE values are on the scale of 10^{-2}.

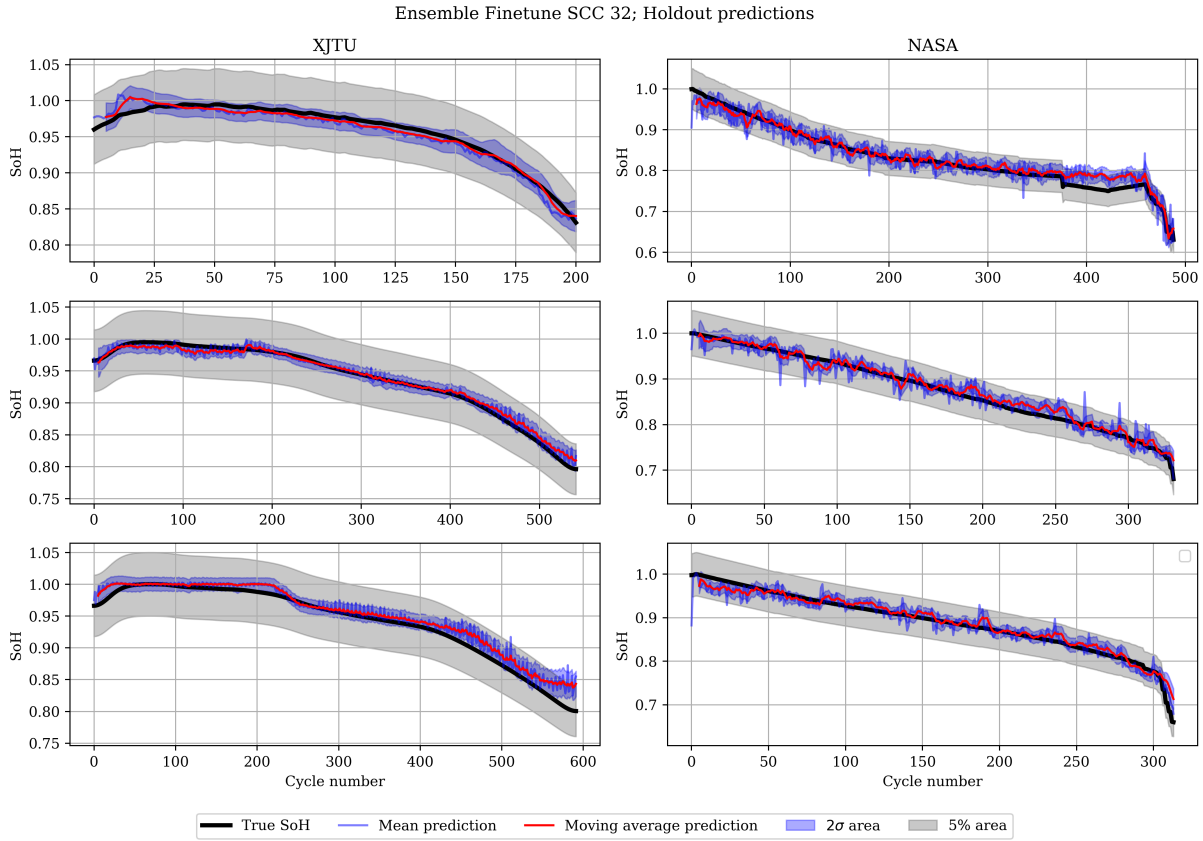


Figure 25: SoH predictions of the finetuned SCC 32 model, for the same sample of representative curves shown in Figure 24.

The above results demonstrate the effectiveness of neural network for estimating battery SoH based on partial charging segments. While the base models perform best with the custom engineered features, through ensemble learning and post-processing, the gap between the learned and custom engineered features can be closed almost completely. The good performance of the estimators warrants the future investigation of possible model extensions and improvements, such as the usage of multiple charge cycles as model input, by means of concatenation or through the usage of recurrent neural networks such as LSTMs. Another possible area of investigation is the applicability of the model on out of distribution data from other sources, not just the holdout. These topics were not investigated here and are left for future work.

6.2. Knee point prediction

Similar as for SoH regression, the prediction of the battery knee cycle also demands an efficient feature extraction technique. In contrast to the previous task however, because the segments are not constrained to very narrow specifications, it is not so straightforward to design robust and universally applicable features for the entire signal of a cycle. For this reason, all knee point estimation methods are based on embeddings obtained via unsupervised signal reconstruction training. As before, several autoencoder strategies were compared against each other. Based on the findings during these experiments, regression models were trained for both the point prediction labels as well as the knee point heatmaps. As with SoH estimation, ensemble learning and prediction postprocessing were used in order to improve prediction quality and reliability. Due to the twofold nature of uncertainty in the ensemble outputs, both in the label data as well as the model predictions, a method of assessing prediction trustworthy-ness is presented.

6.2.1. Autoencoder performance

The encoder models used for knee point prediction are markedly different from the models in the SoH section. The first aspect is the training, which for the knee point prediction is only conducted in an unsupervised manner. Secondly, curve knee predictions are based on a more complicated input matrix. The entire data segment of a cycle is used, and an additional temperature channel is provided as input. Due to the longer and more intricate signal, the number of sampling points is also higher at 512 samples. This larger input dimension of 4×512 requires a higher parameter count in the encoder. While the signal reconstruction quality generally correlates with embedding size, the smaller 4×32 embeddings perform slightly better than 20×8 in terms of reconstruction metrics, see Figure 26.

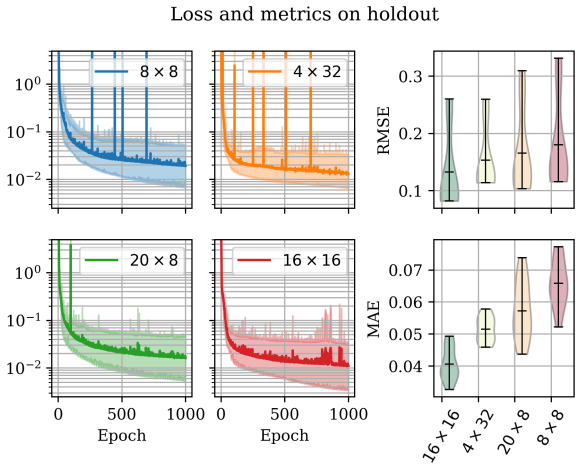


Figure 26: Left: Validation loss curves with lower and upper bounds for the 12 training runs, per architecture. **Right:** Visualization of the error bounds achieved by the different encoder architectures.

One interesting aspect is the slightly reduced variance of the smaller embedding size of 4×32, see Table 11. The biggest advantage of the larger 16×16 embeddings lies in the MAE holdout performance, where the next best mean score is 20 percent higher. The worst embedding size 8×8 is not further pursued, as it performs significantly worse in the all error metrics. While the results of this section establish some expectations for the models built upon these embeddings, Section 6.2.2 on the next page will highlight how the reconstruction results do not strictly carry over to the main objective of estimating capacity curve knee point locations.

Model architecture	RMSE holdout	RMSE train	MAE holdout	MAE train
16×16	$\mathbf{1.324} \pm 0.714$	$\mathbf{0.900} \pm 0.073$	$\mathbf{0.406} \pm 0.053$	$\mathbf{0.396} \pm 0.045$
4×32	$1.535 \pm \mathbf{0.564}$	$1.208 \pm \mathbf{0.072}$	$0.515 \pm \mathbf{0.040}$	$0.507 \pm \mathbf{0.034}$
20×8	1.657 ± 0.735	1.206 ± 0.100	0.573 ± 0.094	0.542 ± 0.067
8×8	1.804 ± 0.816	1.309 ± 0.082	0.658 ± 0.076	0.623 ± 0.060

Table 11: Autoencoder performance comparison of the four different embedding dimensions on the scale 10^{-1}.

6.2.2. Knee point prediction performance

Surprisingly, good reconstruction performance of the autoencoders does not necessarily translate directly to the prediction accuracy of the regression models. In contrast to the SoH estimation, the model with the best holdout R^2 does not have the top scores in all other categories. While it is clear that ensembles overall provide improved performance, the best embedding size is different for single regressors and ensembles. Looking at the 5% margin, it becomes clear that the knee point prediction is substantially less reliable than the SoH estimation. None of the models even surpass 20% on the holdout split. It should be noted that the training of the models was somewhat biased towards larger distances to the knee point, since many more datapoints during training were sampled far away from the knee. In Table 12, a comprehensive summary of the metrics is given. Generally, the results lie quite closely together, and while the improvement due to ensemble learning is evident, the differences between the embedding sizes are rather small.

Model architecture	R^2 holdout	R^2 train	RMSE holdout	RMSE train
Ensemble 20 × 8	**0.776** ± 0.073	0.996 ± **0.000**	**79.300** ± 11.926	10.544 ± **0.623**
Ensemble 16 × 16	0.766 ± 0.061	**0.997** ± 0.001	81.380 ± 10.356	**9.413** ± 1.210
8 × 8	0.764 ± 0.082	0.972 ± 0.013	81.444 ± 15.45	28.044 ± 6.14
Ensemble 4 × 32	0.747 ± 0.044	0.996 ± 0.001	84.702 ± **6.534**	11.221 ± 1.516
Ensemble 8 × 8	0.745 ± **0.032**	0.992 ± 0.002	85.528 ± 8.459	14.679 ± 1.364
16 × 16	0.717 ± 0.120	0.984 ± 0.004	88.567 ± 18.486	21.180 ± 2.280
20 × 8	0.693 ± 0.110	0.984 ± 0.003	91.891 ± 15.646	21.391 ± 2.082
4 × 32	0.620 ± 0.223	0.896 ± 0.283	100.906 ± 25.578	37.417 ± 42.739

Model architecture	MAE holdout	MAE train
Ensemble 20 × 8	**53.246** ± 9.399	6.471 ± **0.363**
Ensemble 16 × 16	55.195 ± 6.703	**5.466** ± 0.572
8 × 8	54.401 ± 10.886	19.962 ± 4.847
Ensemble 4 × 32	55.611 ± **4.973**	6.553 ± 0.748
Ensemble 8 × 8	56.648 ± 6.805	9.220 ± 0.800
16 × 16	60.869 ± 12.047	14.624 ± 1.348
20 × 8	62.594 ± 11.761	15.282 ± 1.854
4 × 32	70.383 ± 23.692	28.321 ± 36.081

Model architecture	Error < 5% holdout	Error < 5% train	Error < 5% total
Ensemble 20 × 8	0.176 ± 0.055	0.761 ± **0.009**	0.615 ± **0.013**
Ensemble 16 × 16	0.167 ± 0.043	**0.803** ± 0.021	**0.645** ± 0.019
8 × 8	**0.197** ± 0.042	0.413 ± 0.081	0.359 ± 0.063
Ensemble 4 × 32	0.185 ± 0.058	0.764 ± 0.025	0.620 ± 0.029
Ensemble 8 × 8	0.179 ± 0.041	0.670 ± 0.029	0.547 ± 0.024
16 × 16	0.162 ± **0.031**	0.512 ± 0.031	0.425 ± 0.029
20 × 8	0.169 ± 0.046	0.495 ± 0.049	0.414 ± 0.036
4 × 32	0.148 ± 0.052	0.404 ± 0.128	0.340 ± 0.103

Table 12: Results of the knee point prediction experiments, evaluated for all holdout cycles up until the knee point.

Switching to the heatmap targets provides a similar picture, see Table 13. Compared to the point predictions, the ensembles decisively take the top spots. In particular R^2 and RSME scores are slightly improved, but the 5% error percentage is still consistently low as before. Overall, the single point and heatmap predictions achieve very similar results when considering only the location of the knee point. Comparing only the point predictions however does not take into account that the model outputs in

this case provide additional information. Analysis of the output probability density leads to additional insights, this aspect will be discussed later in Section 6.2.4.

Model architecture	R^2 holdout	R^2 train	RMSE holdout	RMSE train
Ensemble 20 × 8	**0.787** ± 0.083	0.968 ± **0.003**	**76.466** ± 13.102	30.295 ± **1.101**
Ensemble 8 × 8	0.781 ± **0.067**	0.965 ± 0.005	78.359 ± **11.348**	31.628 ± 1.675
Ensemble 16 × 16	0.763 ± 0.106	0.967 ± 0.004	80.778 ± 17.769	31.053 ± 1.394
Ensemble 4 × 32	0.754 ± 0.102	0.964 ± 0.005	82.501 ± 16.437	32.262 ± 1.577
8 × 8	0.696 ± 0.128	0.976 ± 0.009	87.267 ± 18.161	24.929 ± 4.429
20 × 8	0.673 ± 0.142	0.981 ± 0.005	89.519 ± 18.217	22.021 ± 3.139
16 × 16	0.667 ± 0.171	**0.983** ± 0.007	90.606 ± 21.965	**21.031** ± 3.863
4 × 32	0.600 ± 0.165	0.968 ± 0.024	100.067 ± 19.500	27.752 ± 9.206

Model architecture	MAE holdout	MAE train
Ensemble 20 × 8	**52.646** ± 12.573	26.068 ± **0.902**
Ensemble 8 × 8	54.470 ± **8.497**	26.695 ± 1.365
Ensemble 16 × 16	56.306 ± 12.516	26.386 ± 1.193
Ensemble 4 × 32	56.452 ± 12.167	26.854 ± 0.911
8 × 8	59.444 ± 12.551	17.822 ± 3.263
20 × 8	62.559 ± 13.042	15.960 ± 2.857
16 × 16	61.698 ± 13.297	**14.764** ± 3.171
4 × 32	68.744 ± 11.551	19.966 ± 7.111

Model architecture	Error < 5% holdout	Error < 5% train	Error < 5% total
Ensemble 20 × 8	**0.189** ± 0.093	0.155 ± 0.040	0.164 ± 0.018
Ensemble 8 × 8	0.161 ± 0.057	0.176 ± 0.034	0.173 ± 0.022
Ensemble 16 × 16	0.159 ± 0.070	0.160 ± 0.037	0.160 ± 0.029
Ensemble 4 × 32	0.170 ± 0.0720	0.175 ± **0.022**	0.174 ± **0.018**
8 × 8	0.174 ± 0.032	0.432 ± 0.066	0.368 ± 0.049
20 × 8	0.169 ± 0.038	0.457 ± 0.074	0.385 ± 0.059
16 × 16	0.156 ± **0.029**	**0.495** ± 0.080	**0.410** ± 0.064
4 × 32	0.146 ± 0.032	0.405 ± 0.103	0.341 ± 0.080

Table 13: Metrics for the models trained with Gaussian bell curves as targets.

6.2.3. Postprocessing of knee point predictions

The same strategy for postprocessing as for SoH estimation also works for the knee point prediction. Through averaging of past predictions, the metrics can all be raised and finally the percentage of predictions inside the 5% margin crosses the 20% threshold. Since the model predicts the distance to the knee point, the average needs to account for this by subtracting the cycles that have passed since the prediction:

$$y_{\text{post}} = \sum_{i=0}^{k-1} w_i (y_i - k + (i+1)) \tag{42}$$

While the mean value of the metrics initially improves with larger window sizes, there is an optimal length after which performance decreases again. In general, as one approaches the knee point, predictions tend to get more accurate, so a larger window sizes means that the older, worse predictions affect the postprocessed result for a longer period. A larger window also means that there are more cycles in the beginning, where the strategy cannot be used at all. In Figure 27, the effect of the window size on the metrics is visualized for different weighting schemes. In an effort to strike a balance between

all metrics, a window size of 8 was chosen. The improvements due to post-processing are stated in Table 14, in particular the MAE and RMSE get lowered by 5% and 7% respectively.

Model architecture	R^2 holdout	RMSE holdout	MAE Holdout	Error < 5% holdout
Ensemble 20×8	0.791 ± 0.053	72.627 ± 7.969	48.649 ± 7.202	0.212 ± 0.049
Improvements over raw predictions				
Ensemble 20×8	0.003 ± 0.047	-4.107 ± 5.322	-3.798 ± 8.110	0.026 ± 0.071

Table 14: Performance metrics and improvements for the postprocessed knee point predictions with Gaussian labels, using an average of the past 8 model outputs. Compared against the baseline of the raw, unprocessed estimates. All figures for holdout data.

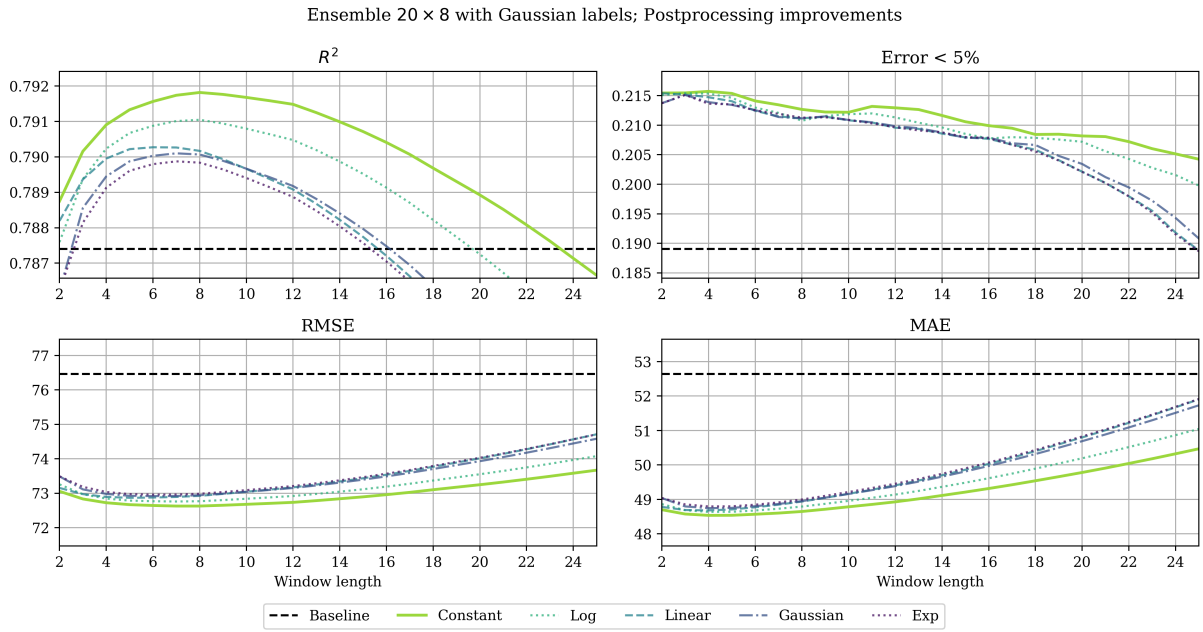


Figure 27: Improvements in metrics for knee point predictions due to post-processing strategies outlined in Section 6.1.4. The constant kernel performs best overall, for the bottom two plots lower is better, on the top higher is better.

6.2.4. Analysis of output densities

As outlined in Section 5.2.1, how to derive the final probability distribution of the knee point from the ensemble predictions is a modeling choice. While the mean predictions are always given by the mean $\overline{\mu} = \frac{1}{N}\sum_k \mu_k$ for all three approaches described in Section 5.2.1, they result in vastly different variances. To be specific, only the Gaussian mixture model provides useful information about the confidence of the ensemble in the final prediction. The variance of a mixture model Z with components X_k, as defined in (33), is given by

$$\begin{aligned} \mathrm{Var}(Z) &= \mathbb{E}[Z^2] - \mathbb{E}[Z]^2 \\ &= \sum_k p_k \mathbb{E}[X_k^2] - \left(\sum_k p_k \mathbb{E}[X_k]\right)^2 \\ &= \sum_k p_k \left(\mathrm{Var}(X_k) + \mathbb{E}[X_k]^2\right) - \left(\sum_k p_k \mathbb{E}[X_k]\right)^2 . \end{aligned} \tag{43}$$

Since for an equally weighted combination of the ensemble outputs, it holds that $p_k = \frac{1}{n}$ for all k and all X_k are Gaussians with mean μ_k and variance σ_k^2, the variance reduces to

$$\mathrm{Var}(Z) = \frac{1}{n}\sum_k (\sigma_k^2 + \mu_k^2) - \left(\frac{1}{n}\sum_k \mu_k\right)^2. \tag{44}$$

Note that the individual predictons μ_k appear in the variance, so if the μ_k are spaced far appart, this increases the variance. The mixture variance serves as an indicator for the prediction confidence, as motivated Figure 28. Ensemble performance on two holdout curves is compared, with the σ areas sketched out. On the input data where the model performs poorly, it also displays large variance, as each ensemble member predicts quite differently. On the other hand, when the model achieves good prediction accuracy, it displays very little variance in the final output distribution.

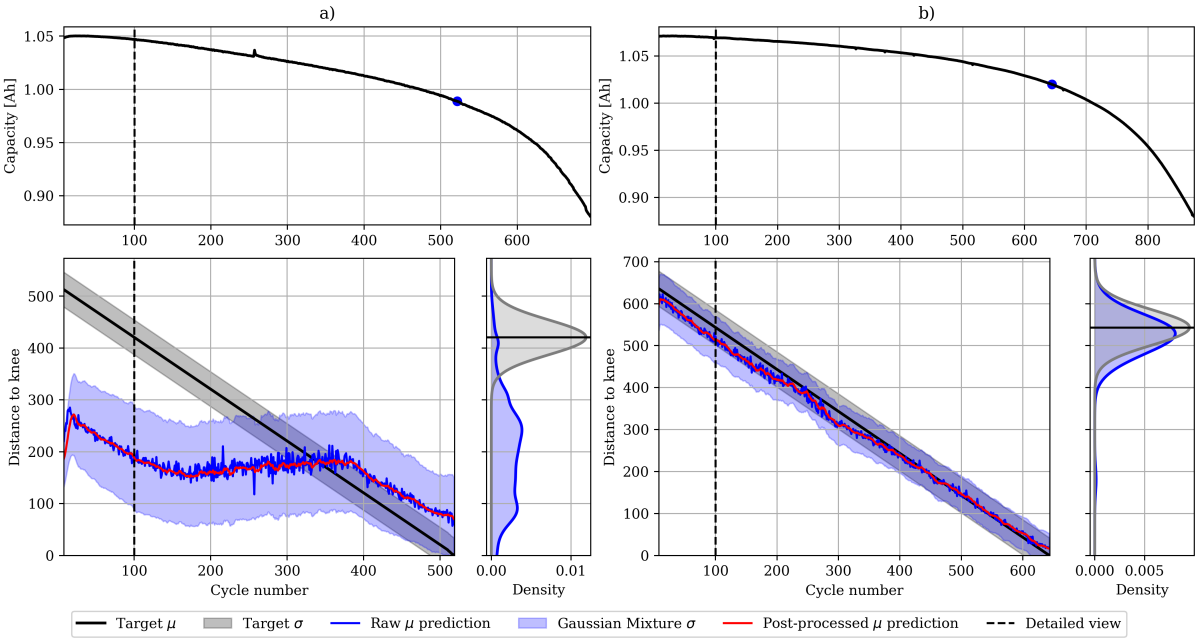


Figure 28: Comparison of ensemble predictions using the Gaussian mixture modeling approach for on holdout data. On the top, the capacity curve is shown, the bottom compares the model predictions with the labels at each cycle. The smaller vertical plots visualize the target distribution and the predicted mixture distribution along the dotted line. Plot **a)** shows a curve where the model performs poorly, severely underestimating the distance to the knee initially. However, the bad performance also manifests itself in the properties of the density function. It is rather flat with multiple peaks, and it suggests very low agreement between the different ensemble members. In contrast **b)** shows a curve where the model performs very well, and there the final density almost perfectly resembles the label, with only a single peak. The shape of the peak indicates very similar predictions throughout the entire ensemble.

For the other two combination methods, the information carried by the density is much less useful. Modeling the final prediction as an average of random variables $Y = \frac{1}{n}\sum_k X_k$ as in (32) leads to a variance of

$$\mathrm{Var}(Y) = \mathrm{Var}\left(\frac{X_1}{n} + \ldots + \frac{X_n}{n}\right) = \sum_k \frac{\mathrm{Var}(X_k)}{n^2} = \sum_k \frac{\sigma_k^2}{n^2}. \tag{45}$$

The final method (31) constructs a random variable W as a Gaussian with parameters $\overline{\mu}$ and $\overline{\sigma}$, by averaging the outputs of the model. The variance is then of course

$$\mathrm{Var}(W) = \overline{\sigma}^2 = \left(\frac{1}{n}\sum_k \sigma_k\right)^2. \tag{46}$$

Neither of the above expressions (45) or (46) includes the individual means μ_k, thus the spread of model predictions does not affect the variance in any way. Because the Gaussian mixture is the only

combination strategy that is able to capture this information, it is the best choice. In Figure 29, this effect is demonstrated.

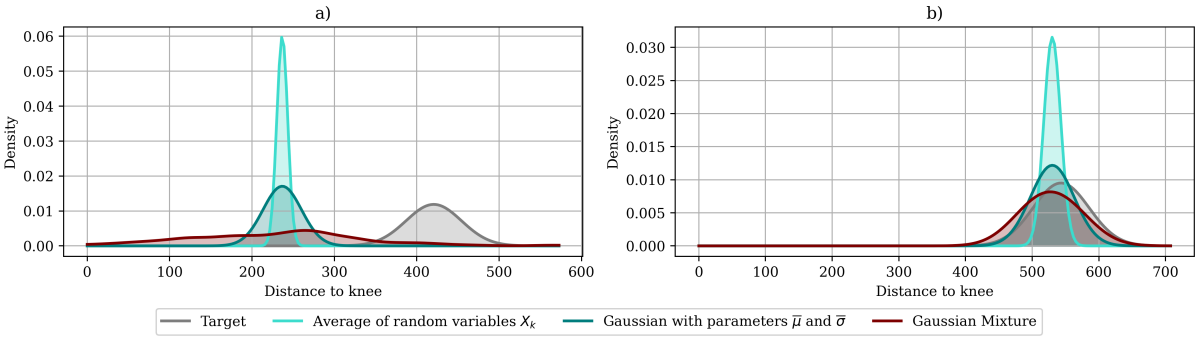


Figure 29: The densities obtained by the three different ways of combining the individual ensemble outputs. Plots **a)** and **b)** correspond to the same curves displayed in Figure 28, evaluated at cycle number 100. While the Gaussian mixture shows higher variance for the sample where the model is wrong, the other two methods actually have lower variance in that case. Hence, only the mixture model provides useful additional information through its density.

Further analysis of the model predictions offers a more comprehensive picture. When comparing the accuracy of the mean predictions with the σ values over the entire holdout data, visualized in Figure 30, it becomes clear that the above intuitive observation is also supported by the data. As a threshold, a value of $\sigma \approx 70$ can be deduced, and if only the predictions inside this area are deemed trustworthy, it is possible to elevate model performance even further. Rejecting all predictions with $\sigma \geq 70$, amounts to excluding roughly 27.6% of all data points. However the remaining 72.4% are vastly more accurate. The improvements achieved over the baseline are stated in Table 15. RMSE and MAE errors are greatly reduced by 42.5% and 47.3% respectively, on the restricted set. Furthermore, R^2 and the amount of errors inside the 5% margin are also increased. Thus, the results strongly support the usefulness of Gaussian mixture variance as a measure of prediction reliability. In Figure 31, it can be seen that qualitatively, the variance based selection does indeed reject only the initial, inaccurate estimates, but as predictions get closer to the true value, they also have small enough σ to be accepted. Note that σ is computed from the smoothed predictions, and thus the rejected sections are usually longer, connected segments at the beginning of the curve, when predictions are not as good. Additionally, curves where the model performs poorly overall can reliably be identified, as demonstrated in Figure 31.

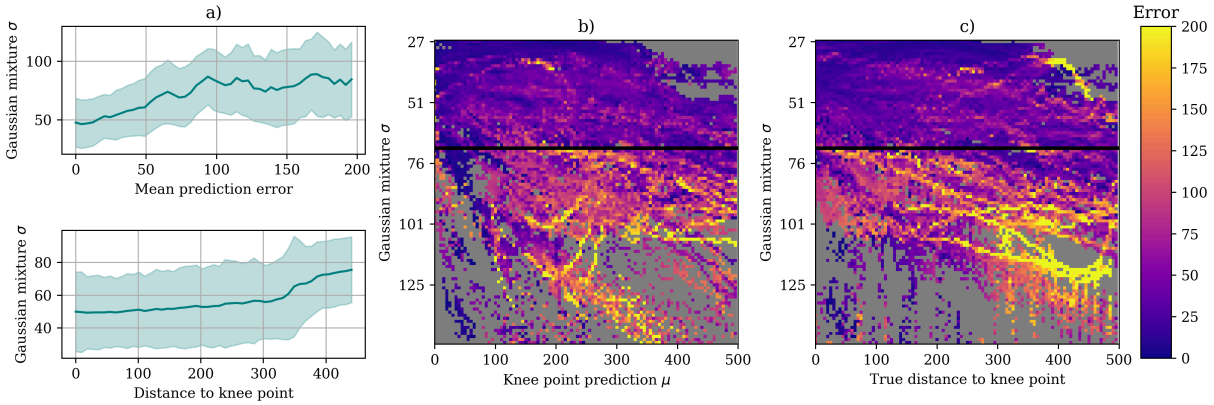


Figure 30: Dependence of the prediction error on the distance to the knee point and the σ of the Gaussian mixture. The two plots in column **a)** show the standard deviation of the Gaussian mixture, plotted against the mean prediction errors and true distance to the knee. While the correlation of low prediction error with a smaller σ value is evident, the true distance to the knee point has only little effect on σ. In **b)**, it can be seen that high error (yellow) correspond in general to large values of both σ and μ. Lower values of $\sigma < 70$ appear to be a strong indicator of low error (blue). The horizontal black line corresponds to $\sigma = 70$. Plot **c)** shows the dependence of the prediction accuracy on both σ and the true label. Again, the area with $\sigma < 70$ contains most of the good model outputs and model accuracy deteriorates as the predicted variance becomes large. The gray values in the plots are points where no data is available.

Model architecture	R^2 holdout	RMSE holdout	MAE Holdout	Error < 5% holdout
Ensemble 20×8	0.839 ± 0.061	53.611 ± 9.894	35.747 ± 7.423	0.222 ± 0.052
Improvements over raw predictions				
Ensemble 20×8	0.051 ± 0.048	-22.854 ± 8.022	-16.899 ± 7.988	0.032 ± 0.104

Table 15: Performance metrics and improvements for the post-processed heatmap predictions. Only the datapoints where $\sigma < 70$ are considered, and the predictions are averaged over the past 8 model outputs. Compared against the baseline of the raw, unprocessed estimates. All figures for holdout data.

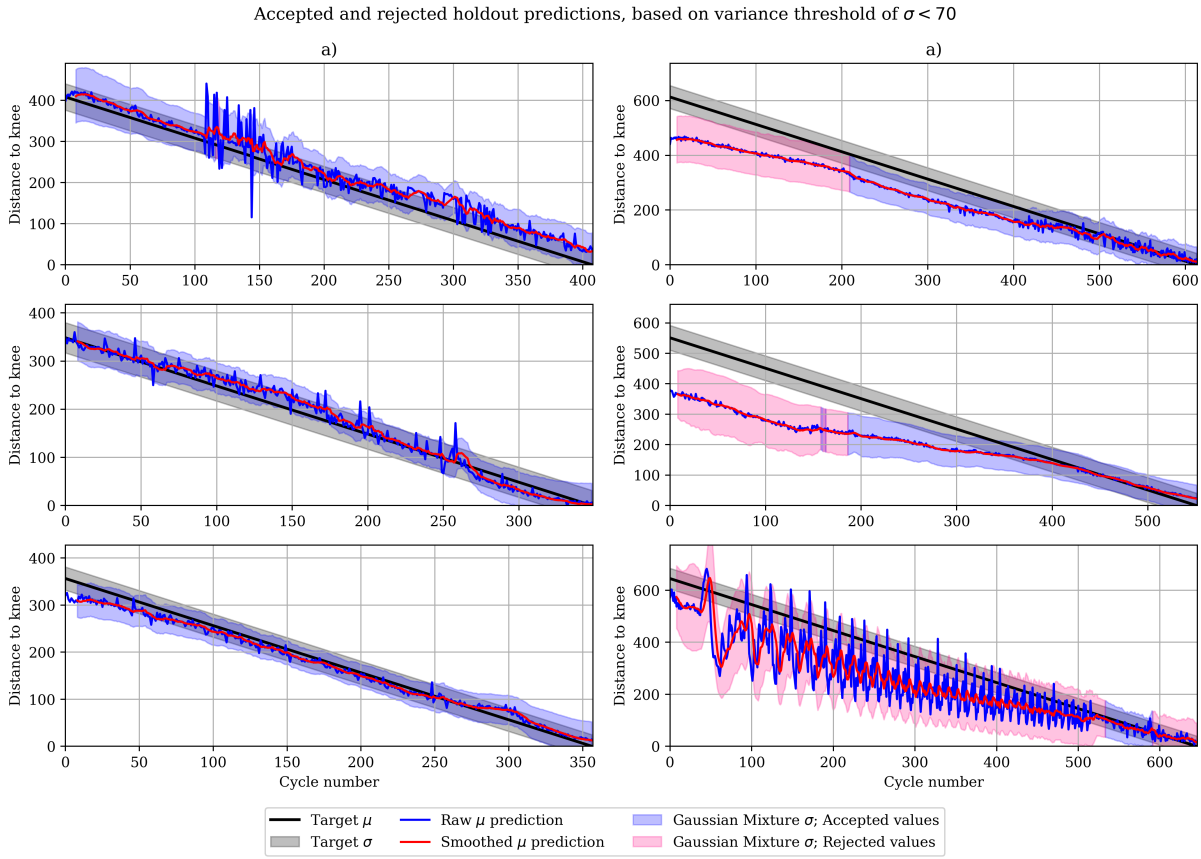


Figure 31: Visualization of the accepted and rejected model outputs on the holdout data, based on the variance threshold $\sigma < 70$. In column **a)**, the means are very accurate and all of them are accepted, with one single exception for the top curve, during a segment where the predictions oscillate very strongly. Column **b)** shows curves where the threshold rules out a significant amount of the predictions. For the upper two cases, approximately 200 of the initial predictions are discarded, whereas the later, more accurate estimates are kept. In the bottom plot, one of the cells where the model struggles the most is shown. Through the variance threshold, the highly oscillating predictions are indentified and pruned.

Overall, the localization and prediction of the knee point using Gaussian densities provides a significant benefit when in terms of interpretability and reliability of predictions, and no compromise with respect to mean prediction accuracy has to be made when compared against the single point labels. The above results exemplify the feasibility of using a purely data-driven model, based on CNN feature extraction and ensemble learning for the task of battery knee point estimation. In particular the novel approach of dealing with the labeling uncertainty has lead to promising results. Possible improvements of the method include the usage of multiple cycles as input, with techniques such as LSTM or other residual neural network architectures. Another interesting extension of the work would be the exploration of transfer learning and the evaluation of the model performance on out of distribution samples. In particular, the generalization from testbed experimental profiles to real world usage data is not explored in this thesis, but of crucial importance to engineering applications.

7. Conclusion

Throughout this thesis, the feasibility of deep learning methods for SoH estimation and knee point prediction was evaluated. The underlying data was gathered entirely from different open source, freely available datasets. After rigorous cleaning and pre-processing, a collection 207 total cells comprised of three different battery chemistries was constructed.

For the SoH estimation, 16 statistical and mathematical features, as well as convolutional neural network encoders were used for feature extraction, with voltage and current curves extracted from partial CC-CV charges as input. Preliminary experiments identified the pretraining and subsequent finetuning of a CNN encoder with an MLP regression head as the most capable architecture besides the custom feature based approach. Using ensemble learning of 50 networks trained in parallel with different initialization, as well as post processing by averaging past predictions resulted in a very capable regressor. With an R^2 score of 0.861 for the custom features and 0.859 for the learned features, a very good fit was achieved. This conclusion is further supported by the percentage of predictions inside the 5% margin of the true SoH, which amount to 89.7% and 88.9% for custom and learned features respectively. These results suggest that learned embeddings can rival feature engineering methods when used in a suitable training regime. However, while high prediction accuracy is achieved when data is readily available, it remains to be examined further what performance can be expected when generalizing from limited data. The inclusion of physical equations into the model architecture or training also could be a promising extension.

In the knee point prognostics task, first the localization of the knee point was studied. In order to unify different localization strategies, and as a method to include labeling disagreement into the training data, the knee points were labeled using a Gaussian density function with μ the average of all label candidates and σ^2 their variance. Then, a purely unsupervised feature extraction technique was employed to transform voltage, current and temperature curves of a full cycle into embeddings. This was done by pretraining an autoencoder on reconstructing the original signals, the resulting embeddings were subsequently used as input for an ensemble of 100 shallow neural networks. The ensemble was trained to predict μ and σ of the knee point location. From the ensemble outputs, a Gaussian mixture density can be constructed by averaging the individual densities, and its variance provides valuable information for categorizing prediction trustworthy-ness.

Using a post-processing scheme of averaging the past 8 predictions, the final model achieved an R^2 score of 0.791, with a mean average prediction error (MAE) of about 48 cycles. Compared to the roughly 400 to 600 cycles from the start of the experiment until the knee point, this accuracy is quite good. When restricting the model to only those predictions where variance of the Gaussian mixture density is reasonably small, in particular $\sigma < 70$, the accuracy can be substantially improved to an MAE of only 35 cycles, with a corresponding R^2 score of 0.839.

It can be concluded that the inclusion of labeling uncertainty into the training, paired with ensemble learning, offers a convenient way to arrive at more robust and interpretable knee point predictions. The resulting output densities contain rich information, both about the agreement of the individual ensemble members, as well as uncertainty associated with the labels. These techniques significantly enhance the usefulness and utility of the final model outputs. Models that receive profiles of multiple cycles as input remain to be explored, as well as the adaptation of the method to data from real world applications, instead of testbed cycling protocols.

In summary, the outcomes of this thesis conclusively support the usage of deep learning methods in battery prognostics and analytics, while highlighting the usefulness of considering the underlying uncertainties. Without relying on expert knowledge of battery chemistry and physics, it is possible to achieve accurate prognostics in a purely data driven fashion using regular operational data.

[1] S. Gill, “How many people own smartphones in the world? (2024-2029).” [Online]. Available: https://prioridata.com/data/smartphone-stats/

[2] “IEA Global EV Outlook 2024,” *IEA Reports*, 2024, [Online]. Available: https://www.iea.org/reports/global-ev-outlook-2024

[3] S. A. Arote, “Fundamentals and perspectives of lithium-ion batteries,” *Lithium-ion and Lithium–Sulfur Batteries*. in 2053-2563. IOP Publishing, p. 1, 2022. doi: 10.1088/978-0-7503-4881-2ch1.

[4] B. A. Johnson and R. E. White, “Characterization of commercially available lithium-ion batteries,” *Journal of Power Sources*, vol. 70, no. 1, pp. 48–54, 1998, doi: https://doi.org/10.1016/S0378-7753(97)02659-1.

[5] M. Kane, “What batteries are Tesla using in its electric cars?.” [Online]. Available: https://insideevs.com/news/587455/batteries-tesla-using-electric-cars/

[6] M. Doyle, T. F. Fuller, and J. Newman, “Modeling of Galvanostatic Charge and Discharge of the Lithium/Polymer/Insertion Cell,” *Journal of The Electrochemical Society*, vol. 140, no. 6, p. 1526, Jun. 1993, doi: 10.1149/1.2221597.

[7] A. Stefanopoulou and Y. Kim, “10 - System-level management of rechargeable lithium-ion batteries,” *Rechargeable Lithium Batteries*. in Woodhead Publishing Series in Energy. Woodhead Publishing, pp. 281–302, 2015. doi: https://doi.org/10.1016/B978-1-78242-090-3.00010-9.

[8] M. Arrinda *et al.*, “Application Dependent End-of-Life Threshold Definition Methodology for Batteries in Electric Vehicles,” *Batteries*, vol. 7, no. 1, 2021, doi: 10.3390/batteries7010012.

[9] K. Movassagh, A. Raihan, B. Balasingam, and K. Pattipati, “A Critical Look at Coulomb Counting Approach for State of Charge Estimation in Batteries,” *Energies*, vol. 14, no. 14, 2021, doi: 10.3390/en14144074.

[10] I. Bloom *et al.*, “Differential voltage analyses of high-power, lithium-ion cells: 1. Technique and application,” *Journal of Power Sources*, vol. 139, no. 1, pp. 295–303, 2005, doi: https://doi.org/10.1016/j.jpowsour.2004.07.021.

[11] W. Diao, S. Saxena, B. Han, and M. Pecht, “Algorithm to Determine the Knee Point on Capacity Fade Curves of Lithium-Ion Cells,” *Energies*, vol. 12, no. 15, 2019, doi: 10.3390/en12152910.

[12] “IEEE Recommended Practice for Sizing Lead-Acid Batteries for Stationary Applications,” *IEEE Std 485-2020 (Revision of IEEE Std 485-2010)*, vol. 0, no. , pp. 1–69, 2020, doi: 10.1109/IEEESTD.2020.9103320.

[13] W. Gao *et al.*, “Comprehensive study of the aging knee and second-life potential of the Nissan Leaf e+ batteries,” *Journal of Power Sources*, vol. 613, p. 234884, 2024, doi: https://doi.org/10.1016/j.jpowsour.2024.234884.

[14] F. B. Planella *et al.*, “A continuum of physics-based lithium-ion battery models reviewed,” *Progress in Energy*, vol. 4, no. 4, p. 42003, Jul. 2022, doi: 10.1088/2516-1083/ac7d31.

[15] V. Sulzer, S. G. Marquis, R. Timms, M. Robinson, and S. J. Chapman, “Python Battery Mathematical Modelling (PyBaMM),” *Journal of Open Research Software*, Jun. 2021, doi: 10.5334/jors.309.

[16] S. E. J. O'Kane *et al.*, “Lithium-ion battery degradation: how to model it,” *Phys. Chem. Chem. Phys.*, vol. 24, no. 13, pp. 7909–7922, 2022, doi: 10.1039/D2CP00417H.

[17] J. Kim and B. H. Cho, “State-of-Charge Estimation and State-of-Health Prediction of a Li-Ion Degraded Battery Based on an EKF Combined With a Per-Unit System,” *IEEE Transactions on Vehicular Technology*, vol. 60, no. 9, pp. 4249–4260, 2011, doi: 10.1109/TVT.2011.2168987.

[18] S. Liu, X. Dong, X. Yu, X. Ren, J. Zhang, and R. Zhu, "A method for state of charge and state of health estimation of lithium-ion battery based on adaptive unscented Kalman filter," *Energy Reports*, vol. 8, pp. 426–436, 2022, doi: https://doi.org/10.1016/j.egyr.2022.09.093.

[19] M. Hossain, M. Haque, and M. Arif, "Kalman filtering techniques for the online model parameters and state of charge estimation of the Li-ion batteries: A comparative analysis," *Journal of Energy Storage*, vol. 51, p. 104174, 2022, doi: https://doi.org/10.1016/j.est.2022.104174.

[20] P. Shrivastava, T. K. Soon, M. Y. I. B. Idris, and S. Mekhilef, "Overview of model-based online state-of-charge estimation using Kalman filter family for lithium-ion batteries," *Renewable and Sustainable Energy Reviews*, vol. 113, p. 109233, 2019, doi: https://doi.org/10.1016/j.rser.2019.06.040.

[21] P. M. Attia *et al.*, "Review—"Knees" in Lithium-Ion Battery Aging Trajectories," *Journal of The Electrochemical Society*, vol. 169, no. 6, p. 60517, Jun. 2022, doi: 10.1149/1945-7111/ac6d13.

[22] J. Lu, R. Xiong, J. Tian, C. Wang, and F. Sun, "Deep learning to estimate lithium-ion battery state of health without additional degradation experiments," *Nature Communications*, vol. 14, no. 1, p. 2760, May 2023, doi: 10.1038/s41467-023-38458-w.

[23] F. Wang, Z. Zhai, Z. Zhao, Y. Di, and X. Chen, "Physics-informed neural network for lithium-ion battery degradation stable modeling and prognosis," *Nature Communications*, vol. 15, no. 1, p. 4332, May 2024, doi: 10.1038/s41467-024-48779-z.

[24] S. Sohn, H.-E. Byun, and J. H. Lee, "CNN-based Online Diagnosis of Knee-point in Li-ion Battery Capacity Fade Curve," *IFAC-PapersOnLine*, vol. 55, no. 7, pp. 181–185, 2022, doi: https://doi.org/10.1016/j.ifacol.2022.07.441.

[25] T. Wang *et al.*, "Capacity degradation analysis and knee point prediction for lithium-ion batteries," *Green Energy and Intelligent Transportation*, vol. 3, no. 5, p. 100171, 2024, doi: https://doi.org/10.1016/j.geits.2024.100171.

[26] D. W. Bacon and D. G. Watts, "Estimating the Transition between Two Intersecting Straight Lines," *Biometrika*, vol. 58, no. 3, pp. 525–534, 1971, Accessed: Jan. 10, 2025. [Online]. Available: http://www.jstor.org/stable/2334387

[27] Y. Ke, Y. Jiang, R. Zhu, W. Peng, and X. Tan, "Early Prediction of Knee Point and Knee Capacity for Fast-Charging Lithium-Ion Battery With Uncertainty Quantification and Calibration," *IEEE Transactions on Transportation Electrification*, vol. 10, no. 2, pp. 2873–2885, 2024, doi: 10.1109/TTE.2023.3304670.

[28] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning," in *Proceedings of The 33rd International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, Eds., in Proceedings of Machine Learning Research, vol. 48. New York, New York, USA: PMLR, 2016, pp. 1050–1059. [Online]. Available: https://proceedings.mlr.press/v48/gal16.html

[29] A. Kendall and Y. Gal, "What uncertainties do we need in Bayesian deep learning for computer vision?," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, in NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 5580–5590.

[30] C. Payer, M. Urschler, H. Bischof, and D. Stern, "Uncertainty Estimation in Landmark Localization Based on Gaussian Heatmaps," in *Uncertainty for Safe Utilization of Machine Learning in Medical Imaging, and Graphs in Biomedical Image Analysis*, in Lecture Notes in Computer Science. Springer, Oct. 2020, pp. 42–51. doi: 10.1007/978-3-030-60365-6_5.

[31] Q. Mayemba, R. Mingant, A. Li, G. Ducret, and P. Venet, "Aging datasets of commercial lithium-ion batteries: A review," *Journal of Energy Storage*, vol. 83, p. 110560, 2024, doi: https://doi.org/10.1016/j.est.2024.110560.

[32] H. He *et al.*, "EVBattery: A Large-Scale Electric Vehicle Dataset for Battery Health and Capacity Estimation," 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:257687419

[33] D. Chicco, M. J. Warrens, and G. Jurman, "The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation," *PeerJ Comput. Sci.*, vol. 7, no. e623, p. e623, Jul. 2021.

[34] P. Kidger and T. Lyons, "Universal Approximation with Deep Narrow Networks." [Online]. Available: https://arxiv.org/abs/1905.08539

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[36] A. Paszke *et al.*, "PyTorch: an imperative style, high-performance deep learning library," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA: Curran Associates Inc., 2019.

[37] J. Bradbury *et al.*, "JAX: composable transformations of Python+NumPy programs." [Online]. Available: http://github.com/jax-ml/jax

[38] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *International Conference on Learning Representations*, p. , 2014.

[39] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Bkg6RiCqY7

[40] DeepMind *et al.*, "The DeepMind JAX Ecosystem." [Online]. Available: http://github.com/google-deepmind

[41] B. Bole, C. Kulkarni, and M. Daigle, "Randomized Battery Usage Data Set." [Online]. Available: https://www.nasa.gov/intelligent-systems-division/discovery-and-systems-health/pcoe/pcoe-data-set-repository/

[42] F. Wang, Z. Zhai, B. Liu, S. Zheng, Z. Zhao, and X. Chen, "Open access dataset, code library and benchmarking deep learning approaches for state-of-health estimation of lithium-ion batteries," *Journal of Energy Storage*, vol. 77, p. 109884, 2024, doi: https://doi.org/10.1016/j.est.2023.109884.

[43] K. A. Severson *et al.*, "Data-driven prediction of battery cycle life before capacity degradation," *Nature Energy*, vol. 4, no. 5, pp. 383–391, May 2019, doi: 10.1038/s41560-019-0356-8.

[44] B. Bole, C. S. Kulkarni, and M. Daigle, "Adaptation of an electrochemistry-based Li-ion battery model to account for deterioration observed under randomized use," *Proc. Annu. Conf. Progn. Health Manag. Soc.*, vol. 6, no. 1, Sep. 2014.

[45] G. Angenendt, "Blog – about the anode overhang effect and how to manage it." [Online]. Available: https://www.accure.net/battery-knowledge/battery-anode-overhang-effect

[46] R. Sharma and R. Bhandari, "Skewness, kurtosis and Newton's inequality," *Rocky Mountain Journal of Mathematics*, vol. 45, no. 5, pp. 1639–1643, 2015, doi: 10.1216/RMJ-2015-45-5-1639.

[47] H.-O. Georgii, *Stochastik*. Berlin, New York: De Gruyter, 2009. doi: doi:10.1515/9783110215274.

[48] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2016.

Appendix

7.1. SoH estimation architectures

CNN ReLU Pooling						
	Layer type	**$In_{channels}$**	**$Kernel_{size}$**	**$Out_{channels}$**	**Stride**	**σ**
Encoder	Conv 1D	4	3	32	1	ReLU
	Max Pooling	32	2		2	
	Conv 1D	32	3	32	2	ReLU
	Max Pooling	32	3		3	
	Conv 1D	32	3	10	1	
	Max Pooling	10	3		1	
Decoder	Upsample 1D	10	3			
	ConvTranspose 1D	10	3	32	1	
	Upsample 1D	32	3			ReLU
	ConvTranspose 1D	32	3	32	2	
	Upsample 1D	32	2			ReLU
	ConvTranspose 1D	32	3	4	1	

CNN ReLU Single Channel Convolution						
	Layer type	**$In_{channels}$**	**$Kernel_{size}$**	**$Out_{channels}$**	**Stride**	**σ**
Encoder	Conv 1D SC	4	5	4	1	
	Conv 1D	4	3	32	1	ReLU
	Max Pooling	32	2		2	
	Conv 1D	32	3	32	2	ReLU
	Max Pooling	32	3		3	
	Conv 1D	32	3	10	1	
	Max Pooling	10	3		1	
Decoder	Upsample 1D	10	3			
	ConvTranspose 1D	10	3	32	1	
	Upsample 1D	32	3			ReLU
	ConvTranspose 1D	32	3	32	2	
	Upsample 1D	32	2			ReLU
	ConvTranspose 1D	32	3	4	1	
	ConvTranspose 1D SC	4	5	4	1	

CNN ReLU Deep						
	Layer type	**$\text{In}_{\text{channels}}$**	**$\text{Kernel}_{\text{size}}$**	**$\text{Out}_{\text{channels}}$**	**Stride**	**σ**
Encoder	Conv 1D	4	3	32	2	ReLU
	Conv 1D	32	3	24	2	ReLU
	Conv 1D	24	3	16	2	ReLU
	Conv 1D	16	3	16	2	ReLU
	Conv 1D	16	3	16	2	ReLU
	Conv 1D	16	3	10	2	
Decoder	ConvTranspose 1D	10	3	16	2	ReLU
	ConvTranspose 1D	16	3	16	2	ReLU
	ConvTranspose 1D	16	3	16	2	ReLU
	ConvTranspose 1D	16	3	24	2	ReLU
	ConvTranspose 1D	24	3	32	2	ReLU
	ConvTranspose 1D	32	3	4	2	

Fully Connected; $4 \times 64 \to 10$				
	Layer type	**In_{dim}**	**Out_{dim}**	**σ**
Encoder	Reshape	4×64	256	
	Linear	256	128	tanh
	Linear	128	64	tanh
	Linear	64	32	tanh
	Linear	32	16	tanh
	Linear	16	10	
Decoder	Linear	10	16	tanh
	Linear	16	32	tanh
	Linear	32	64	tanh
	Linear	64	128	tanh
	Linear	128	256	
	Reshape	256	4×64	

Regression Head			
Layer type	**In_{dim}**	**Out_{dim}**	**σ**
Linear	n_{features}	32	ReLU
Linear	32	16	ReLU
Linear	16	8	ReLU
Linear	8	1	

Ensemble Regression Head			
Layer type	**In_{dim}**	**Out_{dim}**	**σ**
Linear	n_{features}	16	ReLU
Linear	16	16	ReLU
Linear	16	1	

7.2. Knee point prediction architectures

CNN 8 × 8						
	Layer type	**In$_{channels}$**	**Kernel$_{size}$**	**Out$_{channels}$**	**Stride**	**σ**
Encoder	Conv 1D	4	3	32	2	ReLU
	Conv 1D	32	3	32	2	ReLU
	Conv 1D	32	3	16	2	ReLU
	Conv 1D	16	3	16	2	ReLU
	Conv 1D	16	3	16	2	ReLU
	Conv 1D	16	3	8	2	
Decoder	ConvTranspose 1D	8	3	16	2	ReLU
	ConvTranspose 1D	16	3	16	2	ReLU
	ConvTranspose 1D	16	3	16	2	ReLU
	ConvTranspose 1D	16	3	32	2	ReLU
	ConvTranspose 1D	32	3	32	2	ReLU
	ConvTranspose 1D	32	3	4	2	

CNN 20 × 8						
	Layer type	**In$_{channels}$**	**Kernel$_{size}$**	**Out$_{channels}$**	**Stride**	**σ**
Encoder	Conv 1D	4	3	32	2	ReLU
	Conv 1D	32	3	32	2	ReLU
	Conv 1D	32	3	16	2	ReLU
	Conv 1D	16	3	16	2	ReLU
	Conv 1D	16	3	16	2	ReLU
	Conv 1D	16	3	20	2	
Decoder	ConvTranspose 1D	20	3	16	2	ReLU
	ConvTranspose 1D	16	3	16	2	ReLU
	ConvTranspose 1D	16	3	16	2	ReLU
	ConvTranspose 1D	16	3	32	2	ReLU
	ConvTranspose 1D	32	3	32	2	ReLU
	ConvTranspose 1D	32	3	4	2	

CNN 16 × 16						
	Layer type	**In$_{\text{channels}}$**	**Kernel$_{\text{size}}$**	**Out$_{\text{channels}}$**	**Stride**	**σ**
Encoder	Conv 1D	4	3	32	2	ReLU
	Conv 1D	32	3	32	2	ReLU
	Conv 1D	32	3	16	2	ReLU
	Conv 1D	16	3	16	2	ReLU
	Conv 1D	16	3	16	2	
Decoder	ConvTranspose 1D	16	3	16	2	ReLU
	ConvTranspose 1D	16	3	16	2	ReLU
	ConvTranspose 1D	16	3	32	2	ReLU
	ConvTranspose 1D	32	3	32	2	ReLU
	ConvTranspose 1D	32	3	4	2	

CNN 4 × 32						
	Layer type	**In$_{\text{channels}}$**	**Kernel$_{\text{size}}$**	**Out$_{\text{channels}}$**	**Stride**	**σ**
Encoder	Conv 1D	4	3	32	2	ReLU
	Conv 1D	32	3	16	2	ReLU
	Conv 1D	16	3	8	2	ReLU
	Conv 1D	8	3	4	2	
Decoder	ConvTranspose 1D	4	3	8	2	ReLU
	ConvTranspose 1D	8	3	16	2	ReLU
	ConvTranspose 1D	16	3	32	2	ReLU
	ConvTranspose 1D	32	3	4	2	

Regression Head			
Layer type	**In$_{\text{dim}}$**	**Out$_{\text{dim}}$**	**σ**
Linear	n_{features}	32	ReLU
Linear	32	16	ReLU
Linear	16	8	ReLU
Linear	8	1	

Ensemble Regression Head			
Layer type	**In$_{\text{dim}}$**	**Out$_{\text{dim}}$**	**σ**
Linear	n_{features}	16	ReLU
Linear	16	16	ReLU
Linear	16	1	

Regression Head Heatmap (used for both, ensemble and single models)			
Layer type	**In$_{\text{dim}}$**	**Out$_{\text{dim}}$**	**σ**
Linear	n_{features}	32	ReLU
Linear	32	24	ReLU
Linear	24	16	ReLU
Linear	16	8	ReLU
Linear	8	2	