A Convolutional Neural Network to Predict the State-of-Lithiation of Lithium-ion Batteries with the Nickel-Manganese-Cobalt-Oxide Chemistry

by

Sam Ly

A thesis

presented to the University of Waterloo

in fulfillment of the

thesis requirement for the degree of

Master of Applied Science

in

Chemical Engineering

Waterloo, Ontario, Canada, 2023

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Sam Ly was the sole author for chapters 1 and 2 which were written under the supervision of Dr. Jeff Gostick and were not written for publication.

This thesis consists in part of one manuscript written to be submitted for publication. Chapters 3-5 was taken from the manuscript, tentatively named "Machine Learning Surrogate Modeling of State-of-Lithiation for Design and Optimization of Lithium-Ion Battery Microstructures". The contribution of each author is as follows:

Sam Ly – Writing, editing, and interpretation of results as presented in chapters 3-5. Created the electrochemical model and designed and coded the pipeline for creating many electrode geometries using the RSA algorithm by interfacing COMSOL LiveLink™ for MATLAB®. Designed and wrote the code for processing data from FEM simulations to an array-based format in section 3.2. Ideation of usage and implementation of "watershed masks" as per section 3.2.3. Technical implementation of content in section 3.3 and training of ML models.

Dr. Jeff Gostick – Supervised the project, writing and editing of materials in parts chapter 3. Came up with the idea of using RSA to generate different electrode geometries in section 3.1. Gave additional feedback to Sam Ly which led to Sam Ly implementing the use of "watershed masks" as depicted in Figure 3.4 and the creation of a pixel-perfect match – between FEM and ML SoL maps – algorithm of SoL map reconstruction as seen in Figure 3.6.

Dr. Hamed Fathiannasab – Helped supervised the creation of the electrochemical model. Dr. Hamed Fathiannasab helped Sam Ly derive equation (2.28) and gave advice in validating the electrochemical model. Editing of manuscript and interpretation of results. Dr. Fathiannasab identified that particles should be masked between prediction and ground-truth so only values corresponding to SoL should be compared. This advice led to Sam Ly implementing the "watershed masks".

Dr. Mohammad Amin Sadeghi – Writing, editing, and interpretation of results from chapters 3-5. Dr. Sadeghi created Figure 3.1, Figure 3.2, and Figure 3.3. Dr. Sadeghi came up with the idea that samples should be shuffled per electrode instead of per sample, which lead to Sam Ly creating Figure 4.1. Dr. Sadeghi created Figure 4.2 and he and Sam Ly collaborated to create Figure 4.4.

Dr. Niloo Misaghian – Came up with the idea of SoL reconstruction from the ML predictions to compare against FEM SoL maps, as depicted by Figure 3.6.

Abstract

A machine learning (ML) model was developed to study the discharge behavior of a $Li_xNi_{0.33}Mn_{0.33}Co_{0.33}O_2$ half-cell with particle-scale resolution. The ML model could predict the state-of-lithiation of the particles as a function of time and C-rate. Although direct numerical simulation has been well established in this area as the prevalent method of modeling batteries, computational expense increases going from 1D-homogenized model to particle-resolved models. The model was trained on a total of sixty different electrodes with various lengths for a total of 4 different C-rates: 0.25, 1, 2, and 3C. The ML model uses convolutional layers, resulting in an image-to-image regression network. To evaluate model performance, the root mean squared error was compared between the state of lithiation (SoL) predicted by the ML model and ground truth results from pore-scale direct numerical simulation (DNS) on unseen electrode configurations. It was shown that the ML model can predict the SoL within 3.76% accuracy in terms of relative error, but almost an order of magnitude faster than the DNS approach.

Acknowledgements

Firstly, I would like to thank Dr. Sarah Meunier for the numerical computation course I took during my undergraduate studies and being my mentor for my Capstone project. I began to consider graduate studies during your course. I love the effort you put into preparing your courses and I respect how much you care for your students. In a similar vein, I would like to acknowledge Professor Hans de Sterck for the two amazing numerical methods courses. It still blows my mind how hyperbolic partial differential equations could have shocks and rarefactions. Shocks are not spoken much if not at all in our department, and this knowledge really made it "click" for me when reading literature on battery materials with propagating shock behavior. I believe the knowledge from AMATH 741 could really open up more areas of research within chemical engineering and I think it would be interesting to see which new avenues of research we could tackle if more students from my department take your course. I will treasure my experiences in both numerical methods and numerical analysis courses.

I am thankful to Kyu Min Lee, Hao Yu, Nik Burton, and Mike McKague for being awesome group mates. Hao, I'll always treasure that one time we played soccer during Thanksgiving. I only used the cleats I bought once, but it was fun, nonetheless. I also want to congratulate you again on publishing – you were the first of us to do so. Kyu, you are probably one of the funniest guys I have ever met, and extremely hard working. I still have not visited Red Lobster, but perhaps soon. Anyways, I know you will succeed in your career. I would like to thank you, Nik, for listening to my concerns nearing the end of my studies. I can tell that you will be very successful as you continue into your PhD studies and when you eventually go into industry. You have a very critical mind, and it is always nice picking your brain and talking to you about research and other stuff. Thank you for taking me to Grandview church and introducing me to wonderful people there, Mike. I think the network extraction project is very cool, and it is amazing how much coding expertise you have. As I've told you before, I could see you becoming a professor and I think you would be great one at that if you go down that route. Last but not least, I would like to thank Dr. Amin Sadeghi for reading and editing my paper. I learned a lot more about machine learning from you as well as technical presentation.

I would like to acknowledge the University of Waterloo for the Engineering Excellence in master's Fellowship and the Natural Sciences and Engineering Research Council of Canada (NSERC) for the CGS-M scholarship. It would have been difficult to finance my studies otherwise, for which I am grateful for.

I would like to acknowledge CMC Microsystems in providing computational software, specifically COMSOL Multiphysics, in order to complete my studies.

Last but not least, thank you for being there for me Gillian. I love you and your family.

Dedication

To my parents, without their sacrifices my continued education would not have been possible.

Table of Contents

List of Figures

List of Tables

Nomenclature

j_o	Exchange current density of lithium intercalation reaction, molar form ($\frac{mol}{m^2 \cdot s}$).
j	Butler-Volmer reaction rate, molar form ($\frac{mol}{m^2 \cdot s}$).
k_a	Reaction rate constant of the anodic direction of the lithium intercalation reaction ($\frac{m}{s}$).
k_c	Reaction rate constant of the cathodic direction of the lithium intercalation reaction ($\frac{m}{s}$).
α_a	Anodic Tafel coefficient for the lithium intercalation reaction.
α_c	Cathodic Tafel coefficient for the lithium intercalation reaction.
C_s	Intercalated lithium-ion concentration in NMC111 host structure ($\frac{mol}{m^3}$).
$C_{s,0}$	Initial intercalated lithium-ion concentration in NMC111 host ($\frac{mol}{m^3}$).
$C_{s,surf}$	Surface intercalated lithium-ion concentration in NMC111 host ($\frac{mol}{m^3}$).
$C_{s,max}$	Maximum intercalated lithium-ion concentration in NMC111 host ($\frac{mol}{m^3}$).
SoL	State-of-lithiation, $SoL = \frac{C_s}{C_{s,max}}$.
C_e	Lithium-ion concentration in electrolyte ($\frac{mol}{m^3}$).
$C_{e,ref}$	Reference concentration for exchange current density rate equation ($\frac{mol}{m^3}$).
η	Activation overpotential of lithium intercalation reaction (V).
ϕ_s	Electric potential of the solid particle or binder phases (V).
ϕ_e	Electric potential of the electrolyte (V).
U	Equilibrium potential of lithium intercalation in NMC111 host structure (V).
F	Faraday's constant, $96485 \frac{C}{mol}$.
R	Ideal gas constant, $8.3145 \frac{J}{mol \cdot K}$.
T	Cell operation temperature (K).
i_e	Electrolyte current density ($\frac{A}{m^2}$).

i_{app}	Applied current density specified in multiples of C-rate ($\frac{A}{m^2}$).
κ	Ionic conductivity ($\frac{S}{m}$). This is a function of the concentration of ionic species.
σ	Electrical conductivity of a solid phase ($\frac{S}{m}$).
$\frac{\partial ln f_{\pm}}{\partial ln C_e}$	Change of mean molar activity with respect to change in concentration of the ionic species in the electrolyte.
t_o^+	Lithium-ion transference number.
D_e	Lithium-ion diffusivity in electrolyte ($\frac{m^2}{s}$)
D_s	Lithium-ion diffusivity in NMC host structure ($\frac{m^2}{s}$).
$D_e^{pos,eff}$	Effective diffusivity for electrolyte diffusion in the positive electrode region, found in the P2D model where $D_e^{pos,eff} = \frac{\varepsilon_{pos,e}}{\tau_{pos,e}} D_e$.
$D_e^{sep,eff}$	Effective diffusivity for electrolyte diffusion in the porous separator, found in both P2D and resolved models where $D_e^{sep,eff} = \frac{\varepsilon_{sep}}{\tau_{sep}} D_e$.
a	Specific surface area of the active material, found in the P2D model ($\frac{m^2}{m^3}$).
$\tau_{pos,e}$	Tortuosity factor of the positive electrode region.
$\varepsilon_{pos,e}$	Void fraction of the electrolyte phase in the positive electrode region.
ε_{sep}	Void fraction of the electrolyte phase in the separator.
Γ_ϕ	Shorthand for Ohm's law, used to define boundary conditions in the FEM solver ($\frac{A}{m^2}$).
Γ_C	Shorthand for Fick's law, used to define boundary conditions in the FEM solver ($\frac{mol}{m^2 \cdot s}$).
P_i	NMC particle i in the positive electrode.
L	Through-plane thickness or length of the positive electrode region (m).
h_{cell}	In-plane thickness or width of the cell (m).
Q_{1c}	1C capacity of cell ($\frac{Ah}{m^2}$).

List of Abbreviations

Adam	A gradient-based neural network optimizer which uses adaptive estimates of lower-order moments (Kingma & Ba, 2015)
BEV	Battery Electric Vehicle
CNN	Convolutional Neural Network
DFN	Doyle-Fuller-Newman model, also known as the P2D model
DL	Deep Learning
DNS	Direct Numerical Simulation
DoD	Depth-of-Discharge
EDT	Euclidean Distance Transform
FEM	Finite Element Method
FNN	Feed-Forward Neural Network
GHG	Greenhouse Gas
GPU	Graphics Processing Unit
ICE	Internal Combustion Engine
ML	Machine Learning
MSE	Mean Squared Error
NMC	Nickel Manganese Cobalt oxide
NN	Neural Network
ReLU	Rectified Linear Unit
RMSE	Root Mean Squared Error
RSA	Random Sequential Addition algorithm as described by Torquato (Torquato, 2002)
SGD	Stochastic Gradient Descent
SoL	State-of-Lithiation
SoL map	State-of-Lithiation map

Chapter 1
Introduction

Climate change has become a rising concern today. It has been typically attributed to the anthropogenic emission of greenhouse gases (GHG), notably carbon dioxide (IPCC, 2014). GHGs originate from various sources including agriculture, oil and gas production and use, transport, poorly insulated buildings, etc. The transportation sector is a significant emitter as it is mostly comprised of vehicles using petroleum-based fuels, which converts hydrocarbons into carbon dioxide. In 2020, the transport sector in Canada accounted for 24% of the national GHG emissions alone (Environment and Climate Change Canada, 2022). While large industrial-scale applications – like commercial aircrafts or cargo ships – are difficult to decarbonize, there is potential to decarbonize passenger vehicles using mass adaptation of all-battery electric vehicles (BEVs). BEVs do not produce any tailpipe emissions, meaning BEVs do not contribute to the local air pollution nor emit GHGs during operation. By transitioning to EVs, Canadians can significantly reduce their carbon footprint, and help mitigate the impacts of climate change. More than 80% of the electricity mix comes from hydroelectricity in many Canadian provinces (Regulator, 2021). In a coal-dominated energy grid, BEVs are about as carbon efficient as internal combustion engine (ICE) vehicles, but are far more efficient on cleaner grids so can leverage the increasing use of green electricity generation (IRENA, 2017).

1.1 Lithium-ion Battery Operation

In this thesis, a Convolutional Neural Network (CNN) was developed to predict the discharge performance of lithium-ion battery half-cell quicker than conventional direct numerical simulations (DNS). First, background on lithium-ion battery operation is given before the thesis work is addressed as it is the predominant battery technology in BEVs (Salgado et al., 2021). Lithium-ion batteries consists of a porous negative and positive electrode amongst other components. Active materials are held together in the electrode regions by a conductive binder (Xu et al., 2019). Graphite is commonly used as the active material in negative electrodes due to its high energy capacity and low cost and is coupled with a lithium alloy as the positive electrode. Chemistries such as iron phosphate, nickel-cobalt-aluminum oxide, or nickel-manganese-cobalt (NMC) oxide are typically used in BEVs (Salgado et al., 2021). Lithium-ion battery operation is visualized in Figure 1.1. During discharge lithium ions are de-intercalated from the graphite particles in the negative electrode and diffuse and migrate through the separator and into the positive electrode particles. Simultaneously, an electric current pass from the

negative electrode to the positive electrode through external circuitry, which drives a load. The reverse – or charge – process occurs in the reverse direction with lithium ions originating from the positive electrode active materials. In contrast to other technologies, such as the lead-acid battery, the active species would simply convert on the surface of the electrode (Sulzer, Chapman, Please, Howey, & Monroe, 2019). This means the active material is limited to the initial amount in the electrolyte whereas lithium-ion batteries have higher energy capacity as it features intercalation so there is active species in both electrolyte and in the materials.

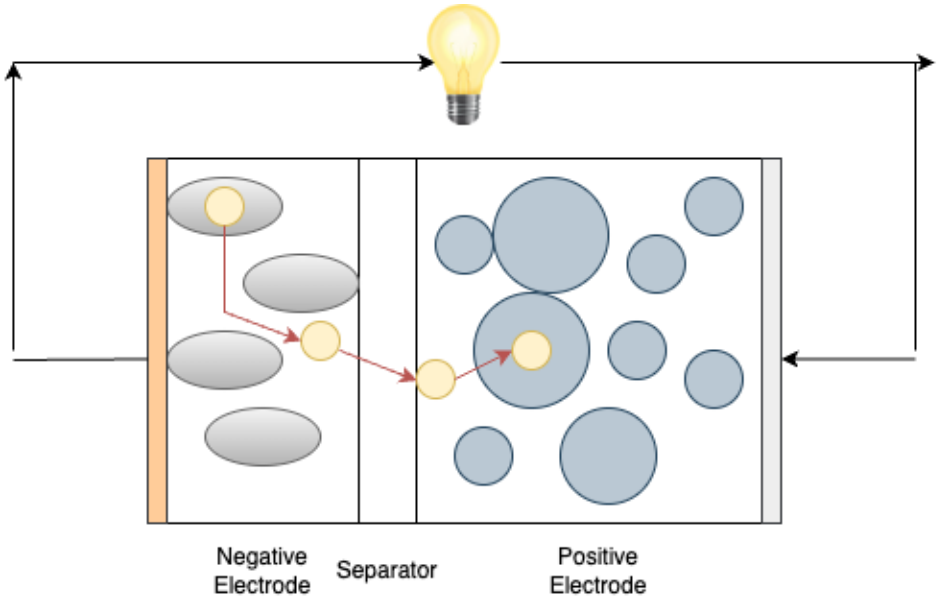

Figure 1.1: Schematic depicting the discharge operation of a lithium-ion battery. Intercalated lithium ions (yellow) diffuse out of graphite particles (grey) in the negative electrode, through the electrolyte, and intercalated into the NMC particles (grey blue) of the positive electrode. Simultaneously, electrons pass through the negative electrode through the conductive matrix, the copper foil, powering an application, through an aluminum foil, and pairs with the intercalated lithium ion in the NMC particle.

1.2 Current Issues

Even though EVs are commercially available the high cost, limited range, and limited tolerance to fast-charging limits wider adoption (Cano et al., 2018). These issues stem from the usage of lithium-

ion batteries. Other related issues are thermal runaway reactions (Ai, Kraft, Sturm, Jossen, & Wu, 2020; Tomaszewska et al., 2019) and mechanical degradation through cracking and – eventually – the disintegration of active materials (Fathiannasab, Zhu, & Chen, 2021; Xu et al., 2019). Though these issues are not the focus of this thesis. This thesis tackles the issue of range by modelling cell discharge behavior. The capacity of a battery is related to the amount of active material in the cell. Increasing the thickness of the cell would increase the amount of active material. However, this naïve approach limits the supposed capacity increase to low discharge rates (Doyle, Newman, Gozdz, Schmutz, & Tarascon, 1996). Lu *et. al.* showed than thin NMC half-cells could be discharged at higher rates for longer compared to thicker cells (Lu et al., 2021). This phenomenon is due to concentration polarization. Even though thick cells are designed for high-capacity usage, a thin cell with less active material can last longer in some scenarios. This results in a severe limitation in cell design – to balance between range and efficiency, the current trend in BEV manufacturing is to either use more thin cells or "overdesign" energy cells (Gallagher et al., 2016). Though this limitation may be alleviated by considering and, specifically, improving the architecture of the cell. Recent works achieved better capacity utilization by choosing a suitable particle-size-distribution (Lu et al., 2020, 2021), specific particle placement (Lu et al., 2020), and electrode perforation (Chen et al., 2020). To evaluate performance, one could either experimentally discharge the cell or model the cell performance. Therefore, cell architectural could be designed towards minimizing losses through concentration and evaluated using electrochemical simulations.

1.3 Current and Proposed Modeling Approaches

A physics-based modeling approach could be used to estimate the cell discharge behavior for lithium-ion batteries. The governing equations are described by a system of partial differential algebraic equations. There are two models which use a similar set of equations, as will be discussed shortly. Transport is described by concentrated solution theory in the electrolyte (Newman, John; Bennion, Douglas; Tobias, 1965) and intercalation in NMC can be modeled with Fick's second law. One form of battery modeling considers a macroscopic description of the electrode, where volume-averaged quantities are considered instead of the exact geometrical details. This is referred to as porous electrode theory (Newman, John; Tiedemann, 1975). Usage of both concentrated solution and porous electrode theories is called Newman's framework, and for lithium-ion batteries specifically the Doyle-Fuller-

Newman (DFN) model (Doyle, Fuller, & Newman, 1993; Doyle et al., 1996; Doyle & Newman, 1995). The other form of model considers the exact internal geometry of active materials, otherwise known as the microstructure. Typically only the active materials in the modeled electrode domains are resolved (Chen et al., 2020; Lu et al., 2021; Xu et al., 2019). This model can be referred to as microstructure-resolved modeling and the technique is called DNS. This model is more physical than the DFN model as it does not assume a uniform particle size. Particles have various sizes and take on various shapes within an electrode. Further, neither the average particle size nor the number-averaged particle size could capture phenomena like slow voltage relaxation in a lithium-ion battery with the DFN model (Kirk, Please, & Jon Chapman, 2021). More importantly, the tortuous pathways are described by the geometry itself whereas a tortuosity correction needs to be applied in the DFN model. The techniques described in (Chen et al., 2020; Lu et al., 2021; Xu et al., 2019) are related to reducing the tortuosity of the electrode microstructure, so accurate modeling of tortuosity is important.

Although the microstructure-resolved model is more physical than the DFN model, it is limited in terms of the time it takes to simulate battery behavior (Hutzenlaub et al., 2014) and by extension is limited to only modeling a few active materials within the electrode (Lu et al., 2020, 2021; Xu et al., 2019). Therefore, this work develops another method to screen electrode microstructure quicker than conventional methods. Cells in BEVs tend to be designed to be thin but tightly packed (Gallagher et al., 2016). This setting would benefit from microstructural design as tight packings tend to result in higher tortuosity. Additionally, in the interest of developing cells with better range, it is essential that cells can perform efficiently at various discharge rates. The conventional method in solving the microstructure-resolved model is to use multi-physics software, such as COMSOL Multiphysics due to its commercial availability. Though simulations are time consuming as it requires meshing at the onset, and matrices to be assembled and solved at every timestep. In this regard, this research proposes to use Deep Learning (DL) to tackle the research objective. The success of DL has revolutionized the field of artificial intelligence and opened up new opportunities for innovation and discovery (Lecun, Bengio, & Hinton, 2015). LeCun showed CNNs was able to identify digits in zip code recognition using only raw images instead of complex feature engineering of previous generations of machine learning (ML) models (LeCun et al., 1989). In 2012, Krizhevsky trained a deep CNN – which was previously thought computationally infeasible – using an efficient graphic process unit (GPU) implementation of the convolution operation (Krizhevsky, Alex; Sutskever, Ilya; Hinton, 2012). The trained AlexNet network

had state-of-the-art performance on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2012), beating other techniques at the time by a significant margin.

In terms of optimizing battery microstructures, Takagishi *et. al.* identified the need to accelerate the microstructure design process. They generated microstructures to predict the electrolyte, kinetic, and diffusional resistance using a feed-forward network (FNN). However, their approach considered a simplified electrochemical model. Moreover, it explored manufacturing parameters instead of modeling cell performance. This work aims to use CNNs to estimate cell behavior at various discharge rates to understand the performance of a given microstructure across a wide spectrum. This can be used to rapidly screen microstructural design by looking and the resulting discharge behavior. The model was intended as an accurate surrogate using simulated electrochemical data from the microstructure-resolved model on a NMC half-cell – with lithium foil as the negative electrode and NMC porous electrode – with randomly sized and located circles to represent NMC particles. As a first assumption, only 2D microstructures were considered and it was trained on 4 different C-rates. To understand discharge behavior, the concentration of lithium inside particles and average concentration along the through-plane (the direction of transport) was visualized. As an external outcome, mechanical degradation of particles is a contributing factor in reducing the lifespan of a cell. Fathiannasab *et. al.* showed stresses are highly correlated to the concentration gradient instead the active particles (Fathiannasab et al., 2021). Therefore, quick, and accurate prediction of lithium concentration inside active materials would be a breakthrough in designing high-performance lithium-ion batteries both in terms of range and durability.

The rest of the thesis is organized in the following order: the theory of battery operation at both continuum and microscale along with how CNNs work is presented; the methods of obtaining the DNS solution and ML dataset; the results are discussed; and ends with the conclusions and recommendations.

Chapter 2
Background and Theory

The following sections provides the background for the inner workings of lithium-ion battery operation, and equations are presented for both the continuum and micro-scale approaches for modeling batteries. A half-cell was considered in this work, so the descriptions and equations reflect the use of a lithium foil as the negative and NMC particles for the positive electrode. Both approaches share similar equations, so the sets of equations are contrasted to highlight the differences between the models. The micro-scale equations – as described in section 2.2.3 – were solved using COMSOL Multiphysics. COMSOL Multiphysics was used in this work as it is commercially available Finite Element Analysis (FEA) software. The purpose of COMSOL was solely to generate training data for the CNN, meaning another commercial software could have been utilized. It should be emphasized that no new Finite Element Method (FEM) or numerical methods was developed in this work. Though for completeness, COMSOL implements the Galerkin Finite Element Method (FEM) which uses the same basis and test functions (COMSOL Multiphysics, 2015). The theory of how CNNs work is then presented, and the different components involved in the training and operation of a Neural Network (NN) are discussed. This chapter concludes with the discussion of some image processing algorithms which were used in the pre-processing steps to identify individual particles and create particle masks to create data for this NN framework.

2.1 Lithium-ion Operation at the Microscale

This work only considers the electrochemical operation of a battery cell, which translates to new cell with no existing degradation and at isothermal operation. Thus mechanical, thermal, and other effects are out of the scope of this thesis. Figure 1.1 illustrates the dynamics of lithium-ion battery operation at the microscale, though in the present work only a half-cell is considered to the porous graphite negative electrode is replaced by lithium foil. When the cell is connected to a load, lithium ions are released into the liquid electrolyte and move to the positive electrode region due to diffusion due to concentration gradients and hindered due to migration due to the electrolyte potential. When lithium ions reach a positive electrode particle, they intercalate into the particles, which is controlled by electrode kinetics (Figure 2.1). This process creates an ionic current equal in magnitude to the electronic current, measurable with an ammeter. The ions then further shuttle within the crystalline structure of

the active NMC particles, which is typically described by diffusion, though this behavior may be governed by different dynamics for phase-change materials such as lithium iron phosphate using the Cahn-Hilliard equation (Bazant, 2013). Active materials with multiple phases with respect to the intercalated lithium-ion concentration are said to show solid solution behavior (Bazant, 2013; F. Wang & Tang, 2020; Zeng, 2015). Lithium iron phosphate is such a material, as exhibited by a strong voltage plateau during discharge, meaning a lithium-rich and lithium-poor are present in the cell (Zeng, 2015). On the particle level, this translates into a lithium-rich and lithium-poor region separated by a discontinuity – in mathematics, this is known as a shock. As lithium iron phosphate particles are charged or discharged, the shock would propagate forward. This model shows excellent agreement in comparison between the simulated voltage curve and experimental results for lithium iron phosphate half-cells. In particular, using the Cahn-Hilliard equation Zeng showed the simulated voltage curve displayed the classic flat plateau region at low discharge, but also show how the voltage curve would conform to a "typical" discharge curve at high discharge rates (Zeng, 2015). This is in contrast to other studies which attempt to use Fick's second law with a high and low diffusivity for the lithium-rich and poor phases (Kashkooli et al., 2016). In comparison, the concentration profile from Fick's second law is smooth and continuous and does not have jump discontinuities.

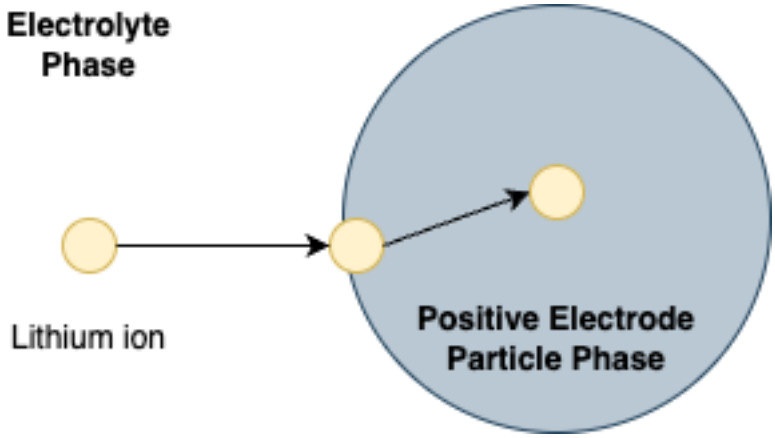

Figure 2.1: Intercalation of lithium ion from electrolyte phase into an electrode particle during discharge.

The chemical reactions for the dissolution of lithium ions from the foil and intercalation into NMC are as follows:

$$Li^{+}_{(l)} + e^{-} \rightleftharpoons Li_{(s)}$$
$$E^{0}(V)\ (vs.\ Li/Li^{+}) = 0 \tag{2.1}$$

$$xLi^{+}_{(l)} + xe^{-} + Ni_{0.33}Mn_{0.33}Co_{0.33}O_{2_{(s)}} \rightleftharpoons Li_{(1-x)}Ni_{0.33}Mn_{0.33}Co_{0.33}O_{2_{(s)}} \tag{2.2}$$

The thermodynamic potentials are typically reported relative to lithium for convenience in the lithium-ion battery literature as opposed to the Standard Hydrogen Potential which is used most everywhere else. Regarding equation (2.2), x is the SoL which represents the fraction of free sites available for lithium to intercalate into within the active material. The nickel, manganese, and cobalt have subscripts of 0.33 to indicate the molar ratio of each chemical in the metal oxide, an equal amount is used so NMC111 is indicated in equation (2.2). Different alloys have different ranges of x to be intercalated. For instance, Li_xCoO_2 can only use around half of its theoretical capacity of $274\ mAh\ g^{-1}$ due to structural instability issues past $4.35\ V$ (Li et al., 2021); this translates to cycling between roughly 0.43 to $1\ SoL$. NMC111 is reported to intercalate between $0 - 0.975$ (Zheng et al., 2013). This property is one of the reasons for lithium-ion batteries being a high energy density storage device; active species are stored within the active materials as opposed to species being plated on top of the electrodes, in lead-acid batteries for instance. Since intercalation materials can host varying amount of lithium within the material, the equilibrium potential is a function of x compared to equation (2.1). Technically, the potential is defined as the thermodynamically reversible potential, though in practice, this data is obtained under very low discharge (or C) rates; 0.02C as reported in (Kashkooli et al., 2016). The equilibrium potential plot for NMC111 is shown in Figure 2.2.

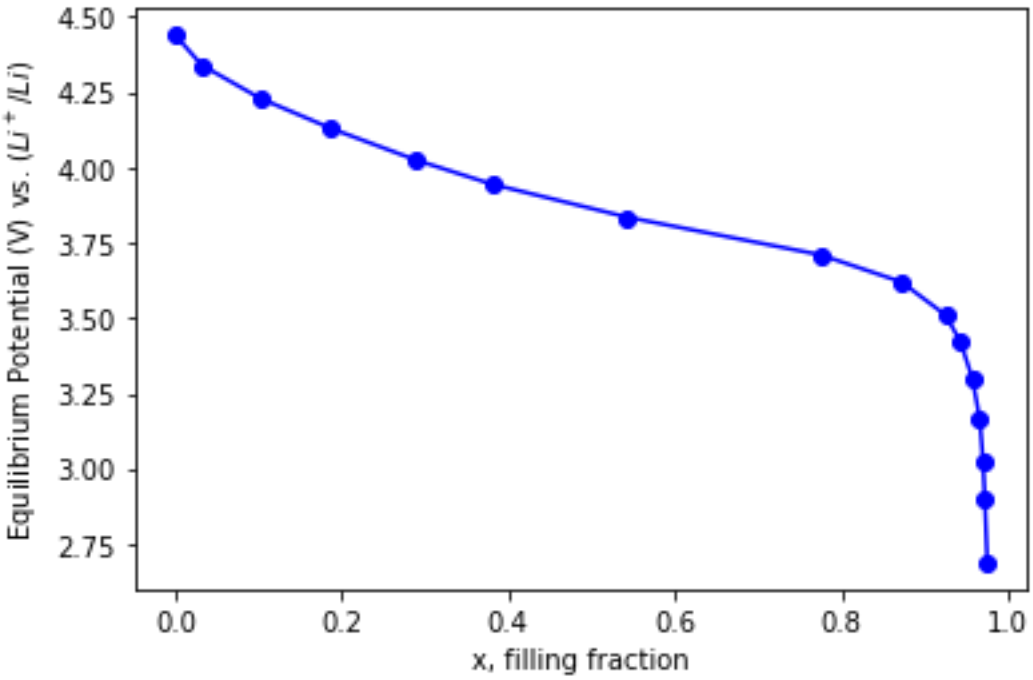

Figure 2.2: Equilibrium potential of NMC111 as a function of $\boldsymbol{x}$, data from (Zheng et al., 2013).

2.2 Governing Equations: From Continuum to Microscale

2.2.1 Models Contrasted

By far the predominant physics-based model for the lithium-ion battery is Doyle-Fuller-Newman (DFN) model (Kirk et al., 2021). The DFN model is reviewed in section 2.2.2. The DFN model uses concentrated solution theory (Newman, John; Bennion, Douglas; Tobias, 1965) and porous electrode theory (Newman, John; Tiedemann, 1975). Concentrated solution theory considers the concentrated effects of transport in comparison to a dilute-solution theory such as the Nernst-Planck equation. The interested reader could refer to Bizeray *et. al.* (Bizeray, Howey, & Monroe, 2016) for an in-depth comparison between concentrated and dilute theories or Newman and Thomas-Alyea (Newman, John; Thomas-Alyea, 2004) for a reference text. Porous electrode theory refers to use of volume-averaged quantities instead of usage of the exact internal geometry of the electrode (Newman, John; Tiedemann, 1975). In effect, the solid electrode phase is superimposed on the electrolyte phase in the region marked "positive electrode region" in Figure 2.3. For intercalation electrodes, another domain is used for the

particle phase as marked by “NMC particles” in the same figure. The second model does not have a formal name but is referred to as the microstructure-resolved or micro-scale equations in this work. This model is reviewed in section 2.2.3. As shown in Figure 2.4, the micro-scale model does not utilize porous electrode theory as the active material geometry is resolved within the electrode region. However, like the DFN model, transport in the electrolyte phase is modeled with concentrated solution theory.

Aside from the difference in the geometries between the two models, one major difference between the DFN and microstructure-resolved models could be contrasted between equation (2.8) and equations (2.20) and (2.26), respectively. These are the concentrated solution theory equations and represent transport in the electrolyte. Notably, the influx or efflux of lithium ions due to lithium intercalation is handled differently between the two models. The DFN models this as a homogeneous reaction as observed in the third term in the right-hand side of equation (2.8) whereas the microstructure-resolved model accounts for transport within the electrolyte phase (equation (2.20)) and lithium intercalation separately (equation (2.26)). The DFN model is said to account for the intercalation reaction as a homogeneous reaction whereas the latter as a heterogeneous reaction. In other words, the reactions are modeled as a bulk reaction in the DFN while the microstructure-resolved model models the reaction on the surface of the particles. The different approach to handling the reaction is why the DFN model is sometimes referred to as homogeneous modeling and similarly heterogeneous modeling for the resolved model. Another difference is that resolved modeling is flexible in terms of the size, location, and specific placement of particles, which affects battery performance (Chen et al., 2020; Lu et al., 2020, 2021). This is in contrast with the homogeneously sized spherical particle with symmetry assumption in the DFN model.

2.2.2 The DFN Pseudo-2D Model for Lithium-ion Batteries

The equations in this section describes the DFN model (Doyle et al., 1993; Doyle & Newman, 1995; Doyle et al., 1996) and is depicted in Figure 2.3. This model is often referred to as pseudo-2D since the transport of ions in the electrolyte is considered as a 1D model, while intercalation into the NMC particles is modeled as a second transport process occurring at each location in the 1D model. This is drawn as perpendicular paths to the 1D transport at Γ_p using equation (2.16). Referring again to Figure 2.3**,** **t**he 1D domain represents the cross-section where lithium ions travel from left to right, from the

negative electrode (Li foil) – with equation (2.9) in Ω_{sep} – to the cathode current collector – with equation (2.8) in Ω_{elec} – during cell discharge. The dependent variables in this domain are C_e, ϕ_e, and ϕ_s, representing the concentration of lithium in the electrolyte, electrolyte potential, and solid phase potential, respectively. ϕ_s is only defined in the subdomain marked as the "positive electrode region" (i.e., cathode, Ω_{elec}) which assume there is no voltage loss in the anode. There is a lithium ion sink in the positive electrode region as lithium is intercalated into the 2D rectangular domain representing the NMC111 particles. The lithium concentration 1D domain is exchanged with the NMC domain, which acts as source of Li ions for the reaction. The dependent variable in the NMC domain, Ω_p, is C_s, the concentration of lithium ions intercalated in the solid NMC host structure, modeled by equation (2.14). The particles were assumed to be spherical with uniform radius, which is estimated from microscopy. Moreover, symmetry is assumed in this model, so $r = 0$ is the center of a particle whereas $r = R$ is the outer radius, where intercalation occurs. The rest of this section will discuss the equations of the DFN model.

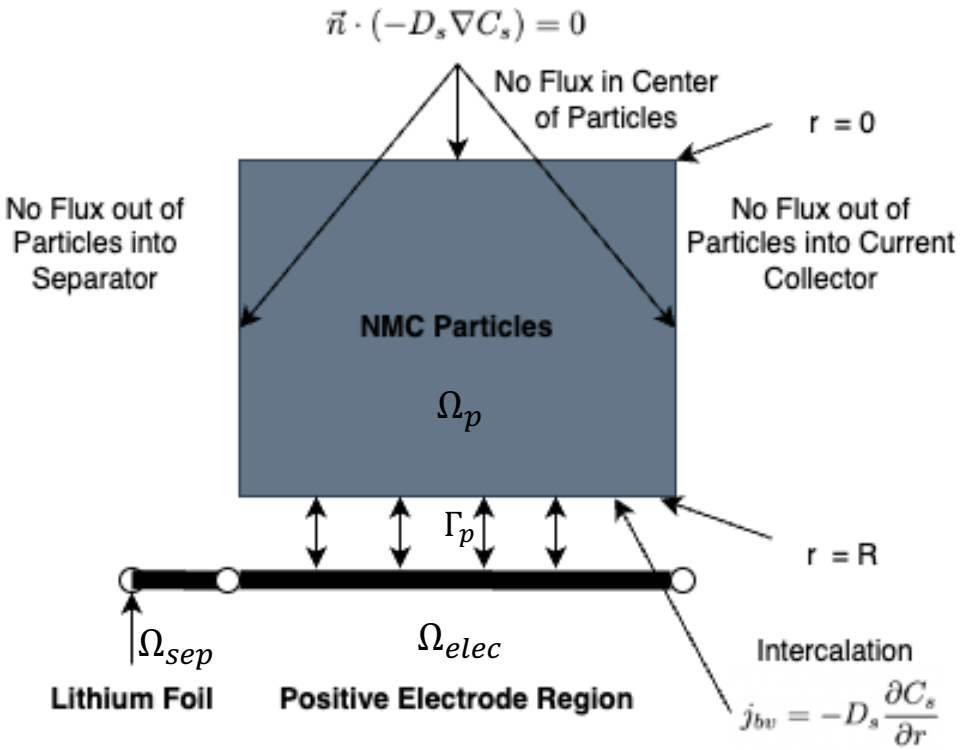

Figure 2.3: Computational domain of the DFN Pseudo-2D model representing a half-cell with a lithium foil and positive electrode composed of NMC111 particles. The rectangular domain, $\boldsymbol{\Omega_p}$,

represents the NMC particles across the length of the positive electrode, where lithium intercalates through the outer radius, $\boldsymbol{\Gamma_p}$. Electrolyte transport occurs in $\boldsymbol{\Omega_{sep}}$ and $\boldsymbol{\Omega_{elec}}$.

Electrode Kinetics

The rate of intercalation of Li into the NMC active particles is described by the Butler-Volmer equation:

$$i = i_0 \left(\exp\left[\frac{\alpha_a F \eta}{RT}\right] - \exp\left[\frac{-\alpha_c F \eta}{RT}\right] \right) \; [=] \frac{A}{m^2} \tag{2.3}$$

The value of i_0, the exchange current density, is defined as:

$$i_0 = F(k_c)^{\alpha_a}(k_a)^{\alpha_c}\left(C_{s,max} - C_{s,surf}\right)^{\alpha_a}\left(C_{s,surf}\right)^{\alpha_c}\left(\frac{C_e}{C_{e,ref}}\right)^{\alpha_a} \tag{2.4}$$

The Butler-Volmer equation is the simplest type of dependence on current density based on the surface overpotential of an electrode and the composition adjacent to the electrode surface (Newman, John; Thomas-Alyea, 2004). It assumes a first-order dependence on concentration and is applicable for a single chemical reaction, though it could be extended to multi-step reactions. This equation is used to describe the surface kinetic reaction on the NMC particles. A positive value means the reaction proceeds in the anodic direction and negative for cathodic direction. A value of 0 would correspond to no net intercalation. Equation (2.2) is the kinetic equation of the system and is written in the cathodic direction. The coefficients α_a and α_c are called the Tafel coefficients, where $\alpha_a + \alpha_c = 1$ and α_a, $\alpha_c \geq 0$. These quantities describe the symmetry factor which describes the fraction of applied potential which promotes the anodic (α_a) and cathodic reaction (α_c), respectively (Newman, John; Thomas-Alyea, 2004). In this system the anodic reactants are $C_{s,surf}$ and the cathodic reactants are $C_{s,max} - C_{s,surf}$ and C_e (Doyle et al., 1993).The exchange current density could be viewed as a reference kinetic parameter which describes the "background" current density in absence of an applied voltage. It is a function of the concentration of lithium ions as well as kinetic parameters, which derived as $i_0 = nFk_a^{\alpha_c}k_c^{\alpha_a}C_{R,}^{\alpha_c}C_O^{\alpha_a}$ as equation (8.23) (Newman, John; Thomas-Alyea, 2004). $C_{s,surf}$ and $C_{s,max}$ represent the intercalated lithium ions on the surface of NMC and maximum concentration of lithium

ions in the active material host structure, and $C_{e,ref}$ is the reference concentration of lithium in the electrolyte phase taken to be $1\frac{mol}{m^3}$ (Xu et al., 2019). Note that $C_{s,max}$ is a material constant and that as $C_{s,surf}$ approaches $C_{s,max}$, i_0 approaches 0 and the reaction stops. The kinetic parameters are k_a and k_c are the reaction rate constant of the anodic and cathodic directions of the lithium intercalation reaction, and α_a and α_c have been reported to be equal in the anodic and cathodic direction (Smith & Wang, 2006), $\alpha_a = \alpha_c = 0.5$.

$$\eta = \phi_s - \phi_e - U\left(\frac{C_{s,surf}}{C_{s,max}}\right)[=]\,V \tag{2.5}$$

Then, the activation overpotential, η, depends on the solid- and electrolyte-phase potentials, as well as the electrode potential of the material, U.

Lithium-ion Transport in the Electrolyte Phase

Lithium-ion transport in the electrolyte phase is modeled using the Onsager-Stefan-Maxwell equations as typically used in the DFN model (Bizeray et al., 2016). In the void space of the electrode the charge flux is given by:

$$i_e = -\kappa\nabla\phi_e + \frac{2RT}{F}\kappa\left(1 + \frac{\partial ln f_{\pm}}{\partial ln C}\right)(1 - t_0^+)\nabla ln C_e\,[=]\frac{A}{m^2} \tag{2.6}$$

where κ is the ionic conductivity varying with lithium concentration, $f_{\pm}$ is the mean molar activity coefficient of the electrolyte, and t_0^+ is the lithium-ion transference number. The ionic current originated from the concentrated solution theory work of Newman *et. al.* (Newman, John; Bennion, Douglas; Tobias, 1965) and was applied to lithium-ion battery modeling by Doyle *et. al.* (Doyle et al., 1993). Due to the absence of reactions in the separator region, the ionic current is conserved in the separator region:

$$\nabla \cdot i_e = 0 \tag{2.7}$$

$$\varepsilon_{pos,e}\frac{\partial C_e}{\partial t} = \nabla \cdot \left(D_e^{pos,eff}\nabla C_e\right) - \frac{\varepsilon_{pos,e}}{\tau_{pos,e}}\frac{t_0^+}{F}i_e + aj(1 - t_0^+)\,[=]\frac{mol}{m^2 \cdot s} \tag{2.8}$$

where $\varepsilon_{pos,e}$, $\tau_{pos,e}$, and a are the volume fraction of the electrolyte phase in the positive electrode region, the associated diffusional tortuosity, and the specific surface area of the active particles,

respectively. $D_e^{pos,eff} = \frac{\varepsilon_{pos,e}}{\tau_{pos,e}} D_e$, the effective lithium-ion diffusivity in the electrolyte. Bruggeman's relation is commonly used, where $\tau = \varepsilon^{-0.5}$, thus $D_e^{pos,eff} = \varepsilon_{pos,eff}^{1.5} D_e$. Bruggeman's relation was originally derived for spherical particles or cylindrical cross-sections (Tjaden, Cooper, Brett, Kramer, & Shearing, 2016) and is used in models due to the simplicity of estimating the tortuosity only using porosity in absence of experimental values (Ai et al., 2020; Kirk et al., 2021; Marquis, Sulzer, Timms, Please, & Chapman, 2019). Though even for spherical particles like NMC, this relation is a close approximation but is not exact and the accuracy degrades severely for differently shaped particles like graphite with anisotropic values (Ebner, Chung, García, & Wood, 2014). The governing equation in the separator is similar though without the reaction term:

$$\varepsilon_{sep} \frac{\partial C_e}{\partial t} = \nabla \cdot \left(D_e^{sep,eff} \nabla C_e \right) - \varepsilon_{sep}^{1.5} \frac{t_0^+}{F} i_e \tag{2.9}$$

The initial conditions are as follows:

$$\phi_e = 0\ V \tag{2.10}$$
$$C_e(t = 0) = 1000 \frac{mol}{m^3}$$

At the lithium foil, the boundary conditions are the applied current density, usually defined in terms of C-rate, and the potential at the foil is taken as the ground. The definition of C-rate is the current density which discharges the cell at inverse of C hours. So 1C would discharge the cell in 1 hour and 2C would discharge the cell in half an hour, etc.:

$$-\vec{n} \cdot \Gamma_{C_e}(x = 0) = \frac{i_{app}}{F} \tag{2.11}$$
$$-\vec{n} \cdot i_e(x = 0) = i_{app}$$
$$\phi_e(x = 0) = 0\ V$$

where Γ_{C_e} is a shorthand for the diffusional and migration fluxes. There is no flux of either concentration or ionic current at the current collector:

$$\vec{n} \cdot \left(-D_e^{eff} \nabla C_e \right)(x = L) = 0 \tag{2.12}$$

$$\vec{n} \cdot i_e(x = L) = 0$$

Lithium-ion Intercalation

The current density is governed by Ohm's law where σ_s is the conductivity of the lumped solid phase:

$$i_s = -\sigma_s \nabla \phi_s \tag{2.13}$$

Lithium ions diffuse due to Fick's law where D_s is the diffusivity of lithium ions inside the active material:

$$\frac{\partial C_s}{\partial t} = D_s \frac{\partial^2 C_s}{\partial r^2} \tag{2.14}$$

The initial conditions are:

$$C_s(t = 0) = C_{s,0} \tag{2.15}$$

$$\phi_s(t = 0) = U\left(\frac{C_{s,0}}{C_{s,max}}\right)$$

Lithium ions enter the active material as follows:

$$j_{BV} = -D_s \frac{\partial C_s}{\partial r}(r = R) \tag{2.16}$$

There is no flux of lithium ions at the "walls" (Figure 2.3):

$$-\vec{n} \cdot D_s \nabla C_s\big(x = x_{sep}\big) = 0 \tag{2.17}$$

$$-\vec{n} \cdot D_s \nabla C_s(r = 0) = 0$$

$$-\vec{n} \cdot D_s \nabla C_s(x = L) = 0$$

Lastly, the electronic current to the solid matrix is described as follows:

$$\frac{\partial \phi_s}{\partial x}\left(x = x_{sep}\right) = 0 \tag{2.18}$$

$$i_{app} - i_e = -\sigma_s \frac{\partial \phi_s}{\partial x}(x = L)$$

2.2.3 Microstructure-Resolved Modeling of Lithium-ion Batteries

The geometry of the microstructure-resolved model is illustrated in Figure 2.4. Instead of a separate domain as in the DFN model, the NMC particles are embedded within the positive electrode domain. In this thesis the porous separator and binders were not resolved, thus effective values were used to correct for their reduction in fluxes. The white space in the figure represents the electrolyte phase. Because only 2D domains were considered here it was not possible to have percolation of both the solid and void phase simultaneously. This means if the conductive binder was resolved, it would not be possible for both ionic transport and electronic transport of the current due to path of conductive binder and porous electrolyte void phase being blocked. In 3D, there would be pathways for both ionic and electrical transport to go from one point in the domain to the other – otherwise known as percolation. Instead, in this case a fictitious binder phase is superimposed onto the electrolyte phase in the positive electrode region to provide electrical conductivity for the electrical current. This was accomplished by assigning a non-zero electrical conductivity to the liquid electrolyte phase, which is a tactic used by other microstructural models (Chen et al., 2020; Lu et al., 2021; Xu et al., 2019). The dependent variables modeled are identified: ϕ_e, C_e, and ϕ_s are modeled in the combined electrolyte phase in the separator and positive electrode region; and C_s and ϕ_s are modeled in the NMC particles. Governing equations and initial and boundary conditions common to both the DFN and microstructure-resolved models are referred to whereas specific equations to the latter model are introduced here.

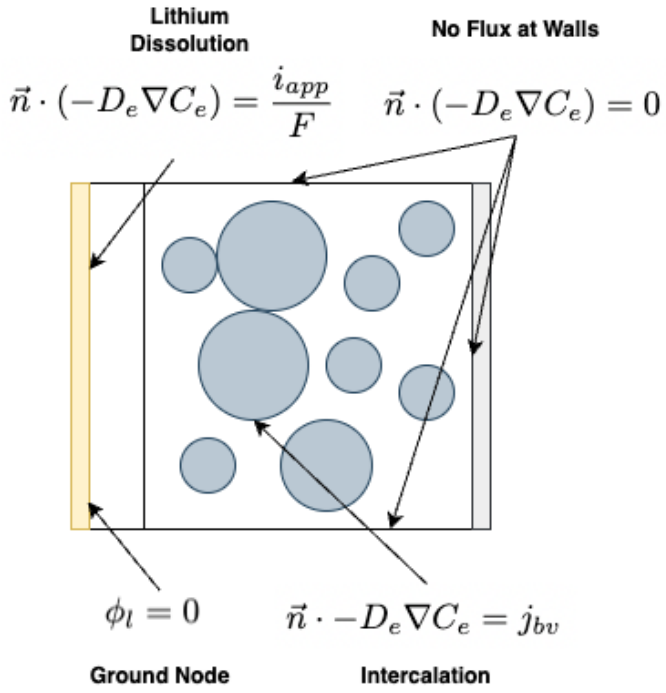

Figure 2.4: Computational domain of the microstructure-resolved model representing a half-cell with a lithium foil and positive electrode composed of NMC111 particles. The regions are, from left to right, lithium foil, separator, and positive electrode region.

Lithium-ion Transport in the Electrolyte Phase

The kinetic equations from (2.3) – (2.5) and the ionic current equation (2.6) are the same between both models. In the microstructure-resolved model there are no sources or sinks for ionic current within the bulk electrolyte phase:

$$\nabla \cdot i_e = 0 \tag{2.19}$$

As the active particles were resolved, compared to equation (2.8), effective correlations are not required to correct for the reduction in flux due to the porous nature of the electrode:

$$\frac{\partial C_e}{\partial t} = \nabla \cdot (D_e \nabla C_e) - \frac{t_0^+}{F} i_e \tag{2.20}$$

The porous separator was not resolved, so equation (2.9) is common between both models. The initial and boundary conditions (2.10) and (2.11) and (2.12), respectively, are still relevant. However, due to the resolved particles, the intercalation reaction results in a source or sink of lithium ions and ionic current. The ionic current is:

$$-\vec{n} \cdot \Gamma_{\phi_l} = j_{bv} F, \forall i \in P_i \tag{2.21}$$

The flux of lithium ions across the electrolyte/NMC particle interface is:

$$-\vec{n} \cdot \Gamma_{C_e} = j_{bv}, \forall i \in P_i \tag{2.22}$$

Lithium-ion Intercalation

Symmetry is not assumed in this model so a more general form of Fick's second law is used:

$$\frac{\partial C_s}{\partial t} = \nabla \cdot (D_s \nabla C_s) \tag{2.23}$$

The electronic current, equation (2.13), is used to describe the electronic current. Though as there are two phases for the conductive binder and NMC particles, compared to one lumped phase in the DFN model, this equation was used for each. Conservation of charge in the solid phases applies separately to each phase:

$$\nabla \cdot \left(-\sigma_{NMC,Binder} \nabla \phi_{s,NMC,Binder}\right) = 0 \tag{2.24}$$

The initial conditions are the same as equation (2.15). In addition, the initial potential of the conductive binder is assumed to be at the same potential as the NMC particles:

$$\phi_{s,\{NMC,Binder\}}(x, t = 0) = U_{NMC}\left(\frac{C_{s,0}}{C_{s,max}}\right) \tag{2.25}$$

The source of lithium ions from the electrolyte phase into the NMC particles is as represented with a Neumann boundary condition and the second boundary indicates there is no flux of lithium out of the domain:

$$-\vec{n} \cdot \Gamma_{C_s} = -j_{bv}, \forall i \in P_i \tag{2.26}$$

$$-\vec{n} \cdot \Gamma_{C_s}(x = x_{current\ collector}, t) = 0$$

where P_i are the NMC particles in the positive electrode region. There is electronic current at the electrolyte/NMC particle interface and the boundary condition at the current collector is the applied current density:

$$-\vec{n} \cdot \Gamma_{\phi_s} = -j_{bv}F, \forall i \in P_i \tag{2.27}$$

$$-\vec{n} \cdot \Gamma_{\phi_s}(x = x_{current\ collector}, t) = -i_{app}$$

The 1C rate for the electrode could be estimated as follows:

$$Q_{1C} = \frac{FC_{s,max}}{3600h_{cell}} \sum_{i=1}^{N_{particles}} \pi R_i^2\ [=] \frac{Ah}{m^2} \tag{2.28}$$

where h_{cell} is the height of the cell in meters.

2.3 Deep Learning

In this thesis a Convolutional Neural Network (CNN) was used to predicate state of lithiation as a function of both discharge rate and discharge time. This model was trained on the output of the heterogeneous model described above. A foreword should be given that DL is largely a practice, though there are some works which provide theoretical results in providing a basis in terms of why some models or techniques work well (Chollet, 2021e), it is important to mention the practical side of DL in the best interest of the reader. In this section, the fundamental details of CNNs and deep learning in general are described. First, the CNNs are compared with a simple fully connected Feed-Forward Neural Network (FNN) to contrast the differences between CNN and FNN. Though the work herein does not leverage fully connected FNNs, so this section will be brief. There will be a brief section on autoencoders as the autoencoder topology was used in this work. Then, the remainder of this section will dive into the mechanics of training NNs.

2.3.1 Fully Connected Feed-Forward Neural Networks

The FNN could be viewed as a building block from which other NN models are based on. The term "feed-forward" refers to data from the input only propagating in the forward direction, from the inputs to the output(s). A schematic of a fully connected FNN is shown in Figure 2.5. A sample calculation is shown in the figure with arbitrarily chosen weights and biases. Starting from the input vector, NN prediction proceeds with matrix vector multiplication and addition, followed by a non-linear activation in each layer until the output unit. More details of the mathematical formulae describing the FNN are provided below. The notation for an element in the weight matrix – in this figure – is w_{ij} where i is the "neuron" unit (top-down) in the next layer and j is neuron unit in the current layer. The task of this particular network is to learn an approximation, $\hat{y}$, to some function y from the independent variables x_1 and x_2, thus this is an example of a single-output model though there is no restriction on the number of output units in general. NNs could be configured with an arbitrary number of hidden layers with an arbitrary number of hidden units in each respective layer; in the depicted figure, there are 2 hidden layers with 3 hidden units each. The fully connected moniker arises as each neuron is connected to every other neuron in the next layer. For a single hidden layer FNNs are known to be universal approximators given a sufficiently large number of hidden neurons (Hornik, Stinchcombe, & White, 1989). However, this is not a particularly useful statement in practice. The theorem states an arbitrary function could be represented by a shallow NN but it does not state precisely how many hidden units are required to accomplish this. Additionally, shallow networks with more units tend to perform poorly in terms of generalization, whereas deeper models can reduce the number of units required and generalization error (Goodfellow, I; Bengio, Y.; Courville, 2016a). Montufar *et. al.* showed deep networks comprised of rectifier nonlinearities could represent functions with a number of regions that is exponential to the depth of the network (Montúfar, Pascanu, Cho, & Bengio, 2014).

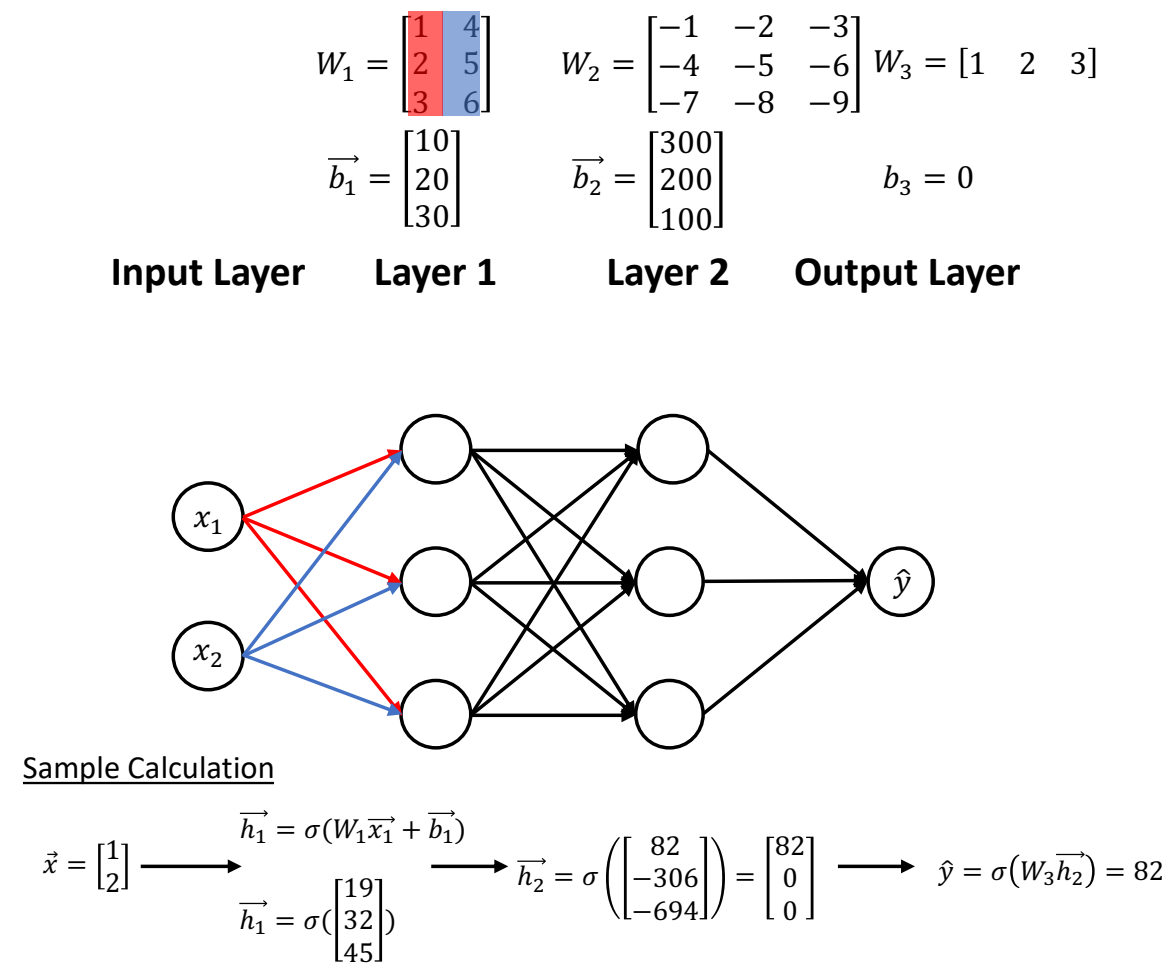

Figure 2.5: A schematic of a fully connected FNN with two hidden layers. There are two input units, and each hidden layer has three units. Given an input $\vec{\boldsymbol{x}} = [\boldsymbol{x_1} \quad \boldsymbol{x_2}]^{\boldsymbol{T}}$, the NN predicts an output $\hat{\boldsymbol{y}}$ from optimizing some loss metric. Arbitrarily chosen weights ($\boldsymbol{W_1}$, $\boldsymbol{W_2}$, and $\boldsymbol{W_3}$) and biases ($\overrightarrow{\boldsymbol{b_1}}$, $\overrightarrow{\boldsymbol{b_2}}$, and $\boldsymbol{b_3}$) are shown at the top of the figure. The weights between the input layer and first hidden layer are colorized to show the relation between the illustration and the corresponding weight matrix. A sample calculation is shown at the bottom, depicting the prediction process starting from the input vector $\vec{\boldsymbol{x}}$. The formula for the output of a fully connected layer is described for the first layer and omitted for the successive layers due to similarity. The non-linear activation function used is the Rectified Linear Unit.

Mathematically, a general fully connected FNN is represented by:

$$\tilde{u} = f^{(N)}(f^{(N-1)}(\dots f^{(1)}(\vec{x}))) \tag{2.29}$$

The mapping of a layer n, $f^{(n)}\left(x^{(n-1)}\right)$, acts on the output of the previous layer $x^{(n-1)}$. Specifically, this operation is represented using an affine transform followed by element-wise activation using a non-linearity:

$$f^{(n)}\left(\vec{x}^{(n-1)}\right) = \sigma(W^{(n)}\vec{x}^{(n-1)} + \vec{b}^{(n)}) \tag{2.30}$$

where $W^{(n)}$ and $\vec{b}^{(n)}$, and σ are the weight matrix and bias vector of layer n, and σ is the non-linear activation function for the layer. Activation functions are critical in providing nonlinearities to the model as a network without activation functions will be composed of linear operations and multiple applied linear operations will only result in a linear result. Nonlinearities are essential in understanding more complex representations (Montúfar et al., 2014). The weights and biases are updated during the optimization process, colloquially known as "training", while the activation functions are a design choice. In principle, activation functions could be different layer-by-layer but is typically set constant throughout the network. Common activation functions are sigmoid, hyperbolic tangent, and the Rectified Linear Unit (ReLU). ReLU is a common choice as it does not suffer from the "vanishing gradient" problem for "deep" NNs, which may be problematic for the sigmoid and hyperbolic activation functions as the corresponding derivatives can tend to 0 in deeper networks. Though the sigmoid and hyperbolic tangent are useful when the range of the output is expected to be bounded by some value – i.e., $range(sigmoid) \in [0, 1]$ and $range(tanh) \in [-1, 1]$. The ReLU function is as follows:

$$\sigma(x) = max(0, x) \tag{2.31}$$

2.3.2 Convolutional Neural Networks

CNNs have been successful in vision-based tasks due to the properties of translational invariance and ability to learn hierarchies of features (Chollet, 2021d; Goodfellow, I; Bengio, Y.; Courville, 2016c; Lecun et al., 2015). Using images as an example, translational invariance refers to the property of the NN being able to learn a pattern in one portion of the image and recognize that pattern in a different section. CNNs can accomplish this property whereas fully connected FNNs, for instance, can learn to recognize a cat in a bottom-left corner of an image but will struggle to recognize the presence of a cat in the top-right corner of another image (Chollet, 2021d). Due to the convolution and because of

translational invariance, CNNs are able to learn local patterns whereas FNNs are limited to global patterns. The ability to learn hierarchies of features is another key property, which relates the depth of the CNN as will be discussed shortly. Returning to the cat example, CNNs learn to recognize edges on a cat and layers deeper in the network learn larger sets of features, such as the eyes, nose, or ears. CNNs accomplish this by performing convolutions using kernels learned during training. The convolution operation and "learned" kernel are discussed in more detail below. These kernels have a few parameters which amount to CNNs being memory efficient compared to the fully connected FNN architecture as the parameters of the learned filters are "shared" across an image whereas FNNs learn values to weight matrices which is applied to the entire input space, because of this these layers are sometimes called dense layers (Goodfellow, I; Bengio, Y.; Courville, 2016c).

The building block of a CNN is the convolutional layer which performs the convolution operation. Mathematically this is represented as (Goodfellow, I; Bengio, Y.; Courville, 2016c):

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{2.32}$$

where $*$, I, K, and S denote the convolution operation, input, kernel (or equivalently, filter), and feature map respectively. The equation above is written for a two-dimensional image, though this operation can be generalized to N dimensions. Convolution operation applies the kernel, K, across the entire input, I. A natural question would be how a kernel should be sized. The size of kernels used in image based CNNs, as a common practice, are sized 3×3 or 5×5 (Chollet, 2021d) and the depth dimension is specified by the number of channels in the tensor. More concretely, it turns out a good practice is to start off with small filters (3×3) (Karpathy, Andrej; Li, 2022) as multiple stacked convolutional layers leads to a receptive field of a larger filter, but with more complex representations and fewer parameters to learn. For instance, three stacked 3×3 convolutional layers has the equivalent receptive field of a 7×7 kernel (Chollet, 2021a; Karpathy, Andrej; Li, 2022; O'Shea, Keiron; Nash, 2015; Simonyan & Zisserman, 2015), and this tactic achieved better results than a single 7×7 filter on the 1000-class ILSRVC-2014 classification problem (Simonyan & Zisserman, 2015). However, that is not to say stacking layers is not the only approach. Szegedy *et. al.* proposed the Inception layer which uses differently sized convolutions from 1×1, 3×3, and 5×5 using the same input, I, to learn features from multiple scales (Szegedy et al., 2015). Both networks by (Simonyan & Zisserman, 2015) and (Szegedy et al., 2015) had state-of-the-art results on ILSRVC-2014 though these works used different approaches. Previously, computer vision experts designed kernels by hand, which resulted in kernels

like the Sobel operator (Pratt, 2001). In contrast, kernels in CNNs function as the "neurons" described in the fully connected FNN in Figure 2.5, thus kernels are "learned" during NN training. The Sobel operator for detecting vertical changes is represented by:

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.33}$$

Zero padding or "same" padding better captures information from the border, but used more as a convenience factor to so the feature map would have the same dimensions as the input (Chollet, 2021a; Karpathy, Andrej; Li, 2022; Simonyan & Zisserman, 2015; Szegedy, Ioffe, Vanhoucke, & Alemi, 2017). This is important in more advanced NN architectures to combine information from different sources such as the Inception V4 or Inception ResNet architectures (Szegedy et al., 2017). The other form of padding is called the "valid" padding, which does not pad the input. In terms of computation, although it is known that the convolution could be efficiently computed using the Fast Fourier Transform (Cooley, James W.; Lewis, Peter A. W.; Welch, 1967) which would require padding for accurate computation due to periodicity, this is not the default implementation of the convolution in DL frameworks. In fact, implementation exploits the fact that a convolution is a linear operation and thus could be represented by a large weight matrix (Karpathy, Andrej; Li, 2022; TensorFlow Developers, 2022) – this does not require padding for accurate computation. The convolution operation is demonstrated on a 4×4 array using the Sobel operator as follows:

$$I * G_y = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} * \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \longrightarrow S = \begin{bmatrix} -32 & -32 \\ -32 & -32 \end{bmatrix}$$

Figure 2.6: Illustration of the convolution operation on input, $\boldsymbol{I}$, using the Sobel operator, $\boldsymbol{G_y}$, with "valid" padding to obtain the feature map, $\boldsymbol{S}$.

And, on an image, the feature map after applying the Sobel operator could be visualized as:

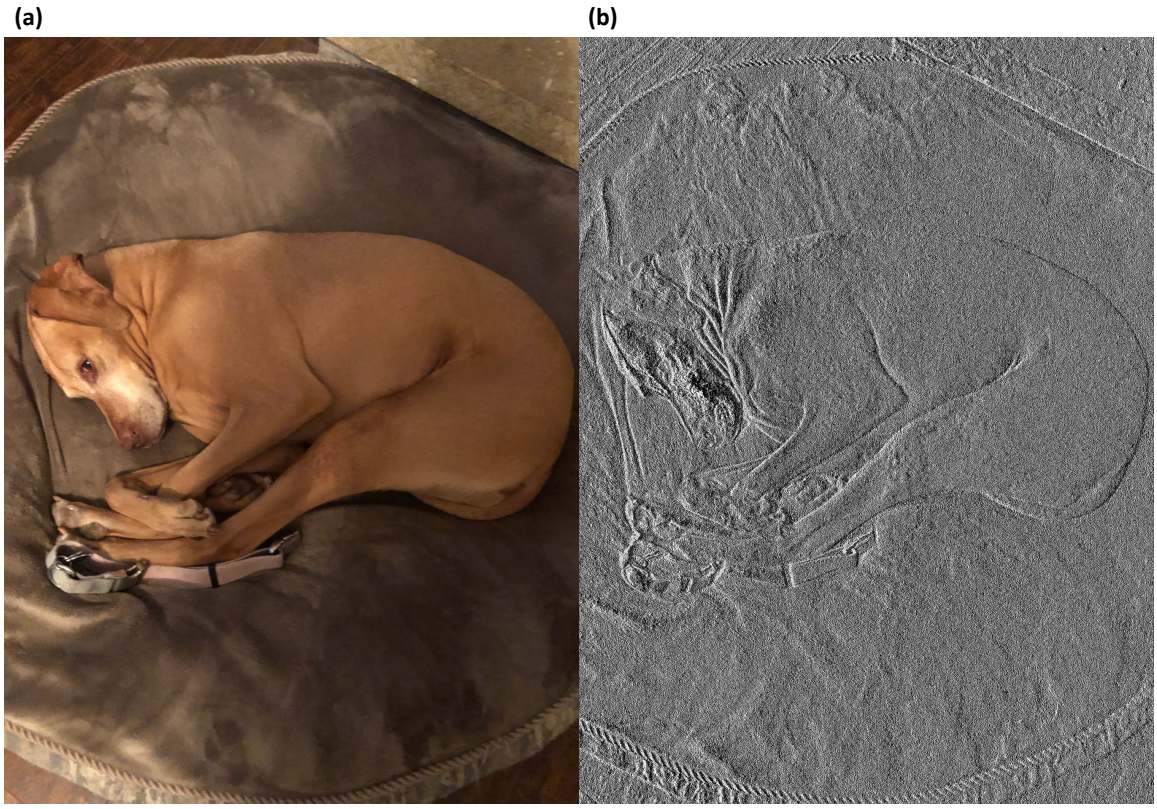

Figure 2.7: Image of a dog (a) and the feature map of applying the Sobel operator, $\boldsymbol{G_y}$, (b).

A user-specified number of filters is specified for a convolutional layer in a CNN, where the elements in the kernels are learned during the training process. The number of feature maps from the output of a convolutional layer corresponds to the number of filters specified. Then, if the CNN is comprised of multiple convolutional layers, the size of the succeeding kernels has additional dimensional corresponding to the number of feature maps created in the previous layer. A non-linear activation is applied similarly to equation (2.30) in a CNN, though a convolution operation is used and the bias in CNNs are scalar instead of vector. Although the choice of activation function is largely architecture dependent, the ReLU activation has been shown to be largely successful in CNNs. Krizhevsky trained CNNs on the CIFAR-10 dataset, a relatively small dataset in comparison to the ILSVRC-2012 dataset, and observed 6x faster training using the ReLU function in comparison to the hyperbolic tangent (Krizhevsky, Alex; Sutskever, Ilya; Hinton, 2012). Although a concern with the ReLU function are "dead neurons" as it either predicts 0 or a non-zero value (equation (2.31)), the ReLU function had

more competitive results for image classification compared to the hyperbolic tangent function on various datasets (Glorot, Xavier; Bordes, Antoine; Bengio, 2011). However, it has been a decade since the paper by Krizhevsky and there have been many more activation functions proposed. Since then, the computational power has also greatly improved. At the time of writing, the RTX 4080 is commercially available whereas Krizhevsky was limited to the GTX 580 (Krizhevsky, Alex; Sutskever, Ilya; Hinton, 2012). As such, one may not be as constrained to try different activation functions due to recent advances in compute power.

Another important consideration is down sampling, which reduces the width and height dimensions of an image. Down sampling reduces the amount of parameters that subsequent layers process (Chollet, 2021d) and the pooling operation helps the CNN model become less sensitive to local artifacts which helps to better achieve translational invariance (Goodfellow, I; Bengio, Y.; Courville, 2016c). There are different ways to down sample an image – in tasks where location matters more such as image segmentation, the strided convolution works better than max-pooling (Chollet, 2021a). Although settings are configurable, the 2×2 pooling operation with a stride of 2 is common place (Karpathy, Andrej; Li, 2022; Simonyan & Zisserman, 2015) as larger pooling operations are very destructive in terms of information discarded. Down sampling is illustrated in Figure 2.8. In general, max-pooling is performed on the resulting feature map after a convolution. In this case, max-pooling is used with a stride of 2, meaning that the maximum value of each 2×2 region is selected, and a subset is created.

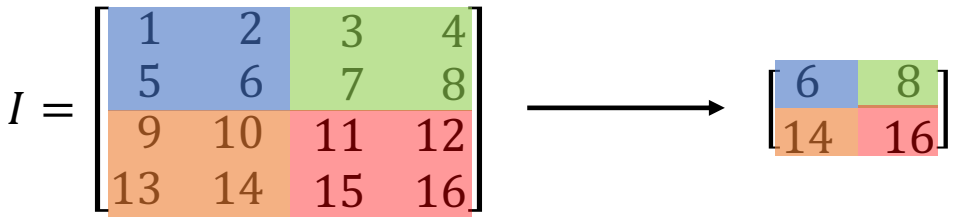

Figure 2.8: Max-pooling using a $\mathbf{2 \times 2}$ window on input, $\boldsymbol{I}$, with a stride of 2.

2.3.3 The Autoencoder Topology

Autoencoders are a type of neural network that are designed to learn efficient representations of data in an unsupervised manner. Autoencoders are a specific form of the FNN which could be trained using the same framework as will be described in section 2.3.3. The "unsupervised manner" refers to the network using the inputs as the target value, i.e., the outputs are the same as the inputs (Goodfellow, I;

Bengio, Y.; Courville, 2016b). It consists of an encoder and a decoder network that work in tandem to compress the input data into a lower-dimensional space, called the latent dimension, and then reconstruct it back to its original form as shown in Figure 2.9. The encoder network takes in the input data and maps it to a compressed representation, also known as a latent space, while the decoder network takes the compressed representation and generates the reconstructed output. After training, the decoder is discarded – the value of this methodology is the ability to learn compressed representations of the input. Autoencoders have various applications such as image denoising, data compression, anomaly detection, and dimensionality reduction (Goodfellow, I; Bengio, Y.; Courville, 2016b).

As will be discussed in the methods section, the CNN trained in this work was clearly not trained in an unsupervised manner, i.e., it had different outputs than inputs. Therefore, the trained CNN is not claimed to be an autoencoder, rather it uses the autoencoder topology. However, the usage of this topology was motivated by the "feature extraction" functionality due to down sampling from input and up sampling to predict another image (Agostini, 2020; Ronneberger, Fischer, & Brox, 2015).

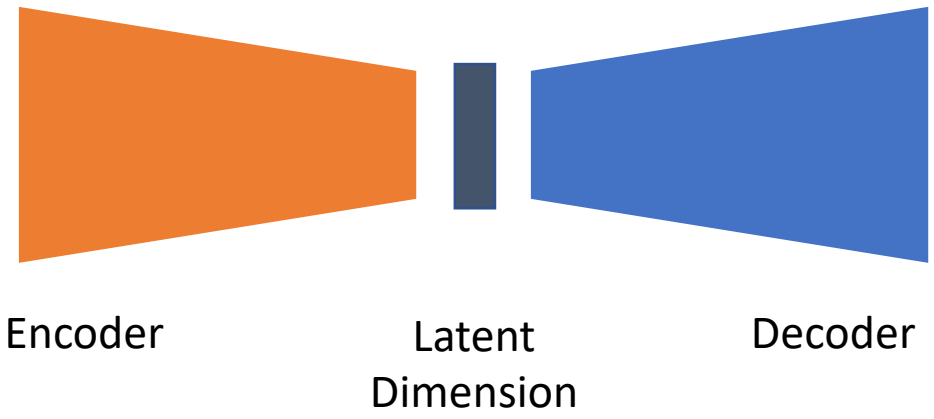

Figure 2.9: The autoencoder topology. The encoder compresses an input to a latent dimension representation. The decoder upscales the latent vector to an output with the same dimensions as the input.

2.3.4 Optimization of Neural Networks

The weights and biases in a NN are updated during the training process, usually using Stochastic Gradient Descent (SGD). Gradients of the weights and biases are computed using the chain rule of calculus and then updated using a formula like:

$$\theta_t = \theta_{t-1} - \alpha_t \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta f_i(\theta_{t-1}) \tag{2.34}$$

where t, i, n, θ, α, and f is the current epoch, the i^{th} training sample, the number of samples in a training dataset, the updated parameter which is either the weights or biases, the step size, and the objective function being optimized. The stochastic or the random part refers to each batch of data is drawn at random (Chollet, 2021b) (chapter, 2.4.3). The term $\nabla_\theta f$ refers to the gradient of the cost function in which the NN is evaluated on with respect to the learnable parameters (Goodfellow, I; Bengio, Y.; Courville, 2016e), $w_{ij}^{(l)}$ and $b_i^{(l)}$, where (l) is the l[th] hidden layer. In a feedforward network the gradient is computed starting from the output of the model. Using the chain rule of calculus, the gradient with respect to the weights and biases of each layer is computed until the input layer is reached. This algorithm is known as the Backpropagation algorithm (Rumelhart & Hinton, G. E.; Williams, 1986). It should be clarified that backpropagation is merely a method to compute the gradients with respect to some ancestors in the network – the interested reader could refer to chapter 6.5 of (Goodfellow, I; Bengio, Y.; Courville, 2016d) for more information on backpropagation, software implementation details, and specifically algorithm 6.4 for an algorithm for backpropagation for a deep NN. The weights and biases are then updated according to equation (2.34). In ML literature, the step size is also called the "learning rate"; this is a key parameter which dictates the degree of how much the NN parameters are updated based on a signal. The objective is also called the "loss function" and is typically the squared difference between the ground-truth data the NN is trying to predict compared against the current predicted value:

$$f = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.35}$$

where y and $\hat{y}$ are the true and predicted values. Equation (2.34) was written for updating parameters on the entire dataset, though in practice small batch sizes are used to balance between computational limitations and pure SGD which uses a batch size of 1. Also, in practice different optimizers work better for different types of data and model architectures, which further modifies the second term in

equation (2.34). In this work, the *Adam* optimizer (Kingma & Ba, 2015) was used which implements an averaged and squared gradient term in its update formula.

The vanishing gradient and exploding gradient problems are issues one may occur in applying SGD to train DNNs. The vanishing gradient problem is where the gradients tend towards 0, which halts the NN from further training. On a similar note, the exploding gradient can occur when a component of gradient (with respect to the weights or biases) or many components are infinity. Mathematically the max norm is $||\vec{x}||_\infty = \max(|x_i|)$ and across the entire vector, the 2-norm is $\left||\vec{x}|\right|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$. Though within the deep learning literature, the term "exploding gradient" is used more frequently. These issues typically occur in recurrent neural networks (Pascanu, Mikolov, & Bengio, 2013) or deep FNNs (Glorot & Bengio, 2010). Glorot and Bengio also attributed the difficulty in training deep NNs to saturating activation functions, e.g., the sigmoid function, and suggested the usage of non-saturating nonlinearities instead (Glorot & Bengio, 2010). As discussed in section 2.3.1, more complex representations could be learned with deeper NNs. Thus, the exploding gradient problem may be problematic. Though these issues may occur, one should not avoid training deep NNs given the success of deep architectures (Simonyan & Zisserman, 2015; Szegedy et al., 2015). Techniques such as batch normalization, where the weights of a layer are rescaled, have been shown to help with training by smoothing out the loss landscape (Santurkar, Tsipras, Ilyas, & Madry, 2018). Batch normalization could simply be added after a hidden layer. Another technique involves using residual connections, which for the first time, allowed training of very deep NNs (He, Zhang, Ren, & Sun, 2016). There are other methods in the literature proposed to tackle the vanishing and exploding gradients issue, though this area is in active research.

2.4 Image Processing

2.4.1 The Distance Transform

The distance transform is an operator applied to binary images. Binary labels of 0 and 1, assigned to each pixel, are termed the "background" and "foreground" of the image, respectively. Applying a distance transform on an image replaces the foreground with values computed from some distance

metric (Rosenfeld & Pfaltz, 1968). Common metrics are city block, chessboard, Euclidean, etc. The distance metric for the Euclidean Distance Transform (EDT) is:

$$d = \sqrt{(i-h)^2 + (j-k)^2} \tag{2.36}$$

where (i, j) and (h, k) are the location of the foreground pixel and the location of the closet background pixel to the foreground, in a 2D image, respectively. Usage of the EDT is shown in Figure 2.10.

2.4.2 The Watershed Algorithm

The Watershed transformation acts on a distance transform to obtain "watershed lines" by interpreting the distance values of elevations and findings 'basins' analogous to physical catchment basins, also known as watersheds. Watershed algorithms are typically used for separating and identifying different objects in an image, otherwise known as object segmentation. The implementation used in this work segments different objects by assigning labels to pixels into separate basins. In the marker-based watershed algorithm, markers are used to guide the "flooding" process in the algorithm. Markers could be determined manually, automatically using a criterion such as the local minima of the gradient of an image or using the local maxima of a distance function to the background (Van Der Walt et al., 2014). The EDT is used in this work. The following figure illustrates the marker-based watershed algorithm.

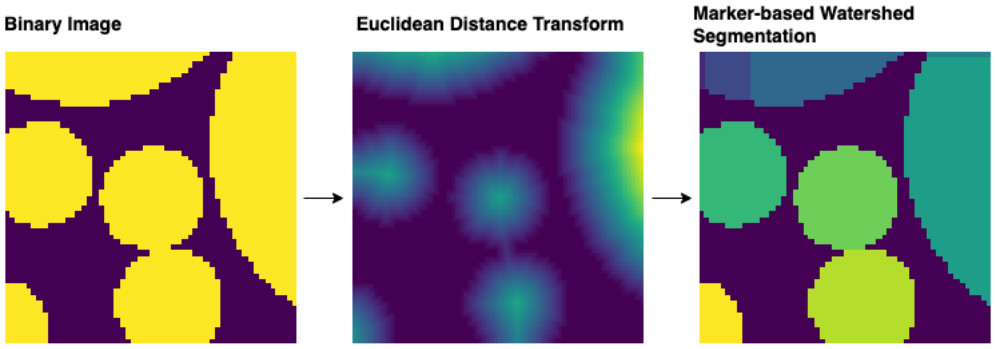

Figure 2.10: Workflow for the marker-based watershed segmentation algorithm. The local maximum is determined from the EDT of a binarized image on the foreground (yellow – left), and objects are segmented based on local maximum markers.

In this work, circular geometry was considered for the active material geometry with clearance, so most particles are not touching. In general, a predefined, idealized shape may not be expected from tomographic data of active materials, and furthermore, many particles may be touching or connected. Therefore, the watershed algorithm discussed herein plays an important part in segmenting particles for identification and further scientific analysis.

2.4.3 Upscaling and Downsampling

In this thesis, upscaling and downsampling of images was performed to zoom images to a uniform size as inputs to the CNN. The incorporation of non-uniform particle sizes necessitated this procedure. Afterwards, the predictions were zoomed to the original size to compare to ground-truth data. Zooming an image is performed by zooming an array using nearest neighbor interpolation for order 0 and spline interpolation from orders 1 to 5. Nearest neighbor interpolation is used in this work as it is well suited in preserving the original image after zooming to one size and back to the original, as shown in the following figure.

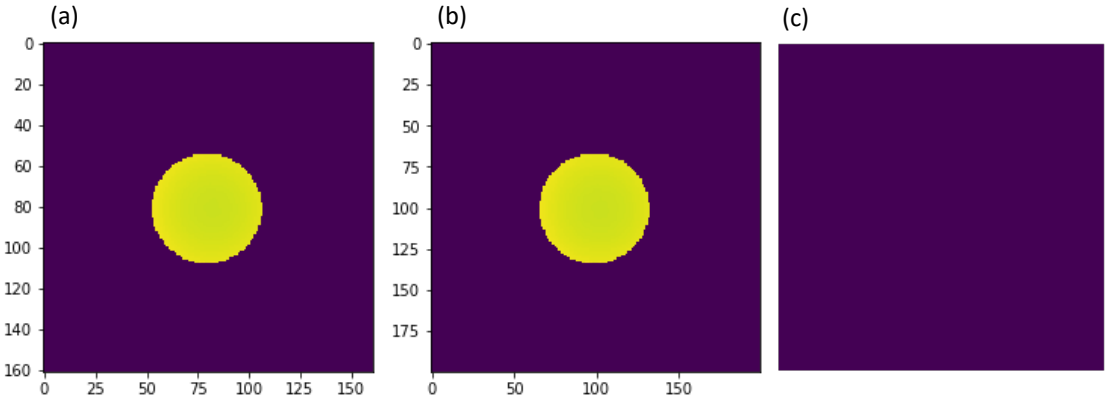

Figure 2.11: Original image sized $\mathbf{161 \times 161}$ (a), zoomed image using nearest neighbor interpolation to $\mathbf{200 \times 200}$ (b), and (c) is the difference between the "unzoomed" image of (b) and the original image (c).

In many cases, zooming an image from size X to Y and back from Y to X results in the same image as illustrated in Figure 2.11(c). However, this may not be true in the general case. Another way to compare for equality is as follows:

$$abs(a - b) \leq (a_{tol} + r_{tol} * abs(b)) \tag{2.37}$$

where a and b is the numeric value associated to a pixel belonging to the original image and the comparator, and a_{tol} and r_{tol} are the absolute and relative tolerance, respectively. This equation is evaluated elementwise, and the compared image is only the same as the original image if the equation is true for all pixels a and b.

To demonstrate the effect of higher-order zoom, the image in Figure 2.11(b) is upscaled using first order interpolation instead of nearest neighbor, as per Figure 2.12(b). Using higher-order zoom introduces artifacts around the border. It is desirable to match the original image as close as possible when returning a zoomed image to its original size. Artifacts are introduced on the first usage of zooming using a high order, resizing the zoomed image to its original size would only introduce more artifacts. Therefore, nearest neighbor interpolation is most suitable for this application.

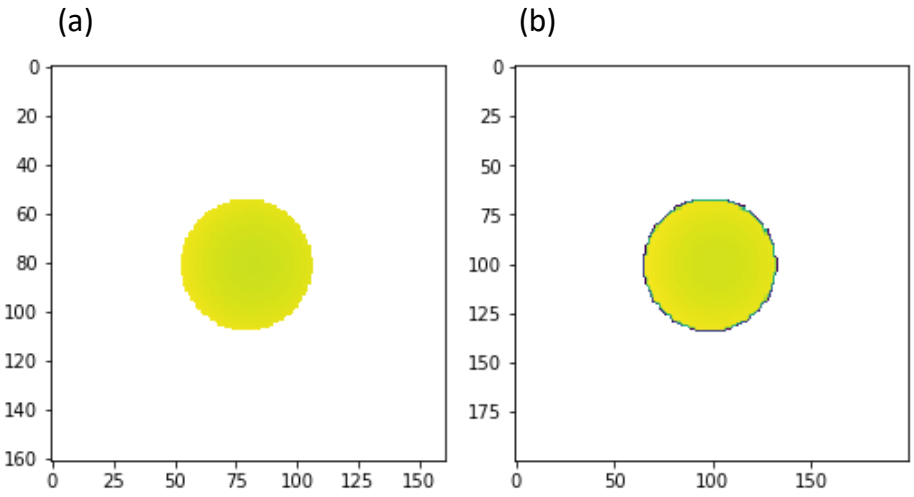

Figure 2.12: Original image sized 161×161 (a), zoomed image using first order interpolation to $\mathbf{200 \times 200}$ (b).

2.4.4 Multi-Dimensional Interpolation

The FEM solves partial differential equations on a domain by solving appropriate interpolant functions on smaller segments of the original domain called "finite elements" (Seshu, 2004). In this work, arrays representing images were used as inputs to the CNN. FEM solutions could be processed in such a format using an interpolation strategy – it should be noted that interpolation in this section pertains to interpolation of FEM solution data and not the spline interpolation in section 2.4.3. Figure 2.13 depicts the mesh of an NMC particle found in the positive electrode.

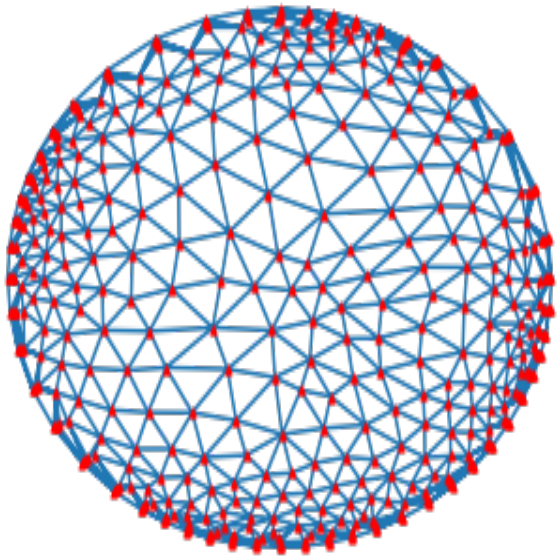

Figure 2.13: A mesh of a domain representing an NMC particle. The original meshing process was performed using COMSOL Multiphysics v5.6 when solving for the FEM solution. Qhull (Barber, Dobkin, & Huhdanpaa, 1996) was used as an intermediate step to recreate the mesh from the vertices (red markers) before interpolating for the empty space inside the convex hull using linear barycentric interpolation.

Solutions for the variable of interest are stored in the mesh vertices in the FEM as illustrated in the red markers as illustrated in Figure 2.13. In a FEM framework, the values within the elements are evaluated from the solved interpolant functions. Though those values are considered missing in an array-based format. Accordingly, the mesh could be recreated using a triangulation algorithm (Barber et al., 1996) and missing values in an element could be interpolated using the vertices immediately enclosing it using linear barycentric interpolation. In other words, values were interpolated within each triangle.

Chapter 3
Machine Learning Dataset Generation, Model Training, and Evaluation

3.1 Electrochemical Model

Prior to the electrochemical simulations being conducted, positive electrode geometries were generated differing by cell lengths and porosity. Each electrode in the dataset consists of half-cells where the NMC particles are represented by polydisperse circles with varying radii randomly generated by a Random Sequential Addition (RSA) algorithm as described by Torquato (Torquato, 2002) (using a custom MATLAB implementation, see Figure 3.1). Circles were used as a first approximation since NMC particles have been reported as relatively spherical within literature (Ebner et al., 2014; Khan, Agnaou, Sadeghi, Elkamel, & Gostick, 2021; Lu et al., 2020). Although particles are spherical instead of circular – i.e., 3D instead of 2D – this work trained CNNs to predict concentration in 2D images to showcase the concept of using a CNN to predict battery operation as a proof-of-concept before any work was attempted at the 3D level. Also, while simulations could have been conducted with 3D geometries and training been conducted on 2D slices of the simulation data, this may have been problematic in terms of the definition of the C-rate. For instance, suppose a 3D electrode geometry was filled with spherical particles of R. Then the correct C-rate corresponds to this particular radius. Suppose that for a given slice all the selected particles are cut such that the resulting image would correspond to a 2D slice with circular particles with radius R_i. Since this radius is smaller than R, the 1C capacity would be lower than the original case. This would result in a mismatch in discharge rates. Using a 2D geometry directly would always model the radius of R, albeit the 3D representation are cylinders instead of spheres. Moving on, 2-dimensional analysis of other chemistries may require different geometries to be considered and a full 3-dimensional implementation could use voxel images obtained by x-ray tomography, but this is outside the scope of this preliminary study. The porosity and cell lengths were motivated by Table I in (Gallagher et al., 2016). The actual porosity values used in this work is mentioned later. The separator lengths used in simulation in this work is taken to be 20 μm (Xu et al., 2019).

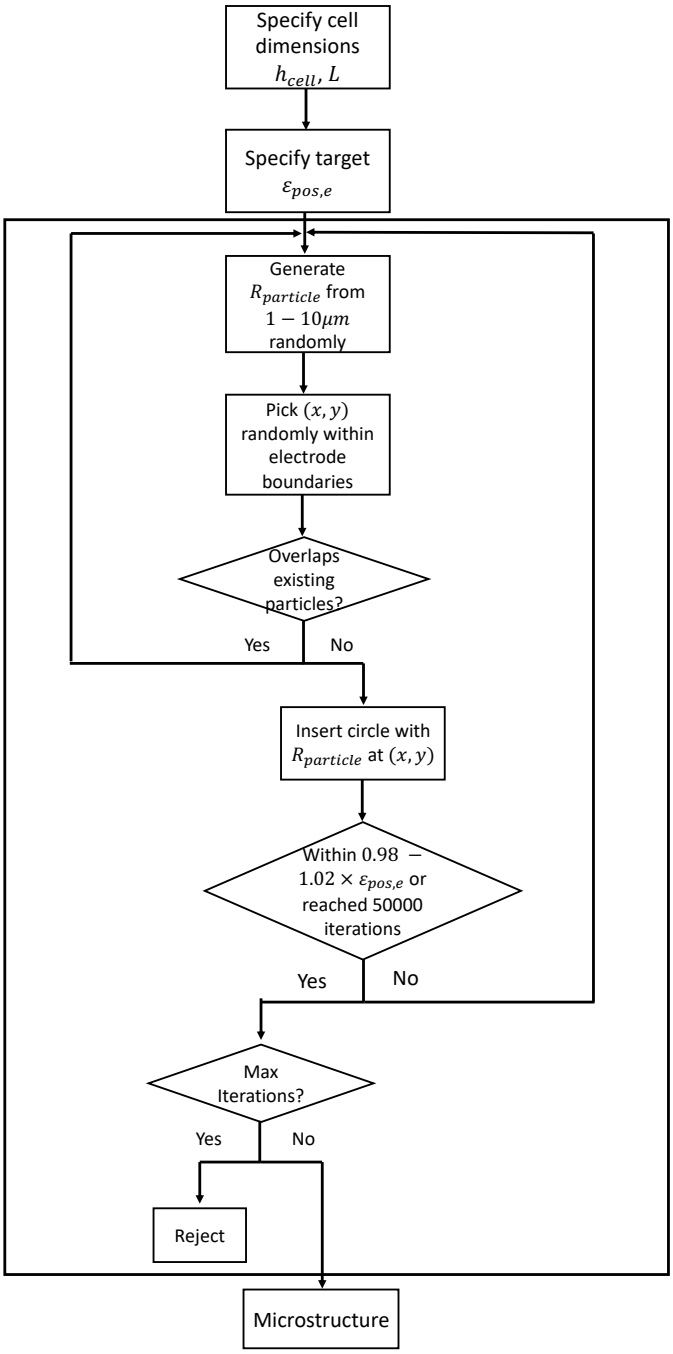

Figure 3.1: 2D positive electrode geometry with circular particles or "microstructure" generated using the RSA algorithm. Cells differed by the specified $\boldsymbol{L}$ and $\boldsymbol{\varepsilon_{pos,e}}$, and the arrangement and sizes of the particles due to random selection. The particle locations and radii, cell length, actual porosity, and diffusional tortuosity were exported for further processing.

The simulations were governed by the equations found in section 2.2.3. The physical parameters are tabulated in Table 3.1. After the geometry was created using the RSA algorithm, simulation parameters, variables, material properties, and governing equations were set programmatically using the COMSOL LiveLink™ for MATLAB®. The applied current density was specified in terms of C-rates, initially $0.25\ C$ is used. Then, the simulation is solved as a time dependent study using segregated steps with Anderson Acceleration on until either theoretical discharge time is reached or the lower cut-off voltage of 3.1 V. Segregated steps refer to the process of solving each individual physics separately instead of solving a multi-physics simulation as a fully coupled problem in each time step. Anderson acceleration is useful for solving linear or almost linear problems using the segregated solver (COMSOL Multiphysics, 2015). The microstructure-resolved equations could be solved either as fully coupled or using a segregated solver in 2D, though solving these equations in 3D with the segregated method can speed up computations. The concentration values in the NMC particles are exported for further processing. This is repeated corresponding to 0.5, 1, 2, and 3 C. A new microstructure is generated, and this process is repeated for the C-rates until all electrochemical simulations for all microstructures (L and $\varepsilon_{pos,e}$) was completed.

Table 3.1: Physical parameters for electrochemical simulations used in this work.

Parameter (unit)	**Value**	**Parameter (unit)**	**Value**
σ_{NMC} $(\frac{S}{m})$	100 (Zheng et al., 2013)	$C_{s,0}$ $(\frac{mol}{m^3})$	980
σ_{binder} $(\frac{S}{m})$	1	$C_{e,ref}$ $(\frac{mol}{m^3})$	1 (Xu et al., 2019)
κ $(\frac{S}{m})$	Function of C_e, (Doyle et al., 1996)	$C_{e,0}$ $(\frac{mol}{m^3})$	1000 (Xu et al., 2019)
t_o^+ (1)	0.363 (Valoen & Reimers, 2005)	$\varepsilon_{sep,l}$ (1)	0.4
$\frac{\partial ln f_{\pm}}{\partial ln C_e}$ (1)	0 (Doyle et al., 1996)	U (V)	Function of C_s, (Zheng et al., 2013)
k_c $(\frac{m}{s})$	2×10^{-11} (Smith & Wang, 2006)	D_e $(\frac{m^2}{s})$	7.5×10^{-10}

k_a $(\frac{m}{s})$	2×10^{-11} (Smith & Wang, 2006)	D_s $(\frac{m^2}{s})$	5×10^{-13} (Zheng et al., 2013)
α_c (1)	0.5 (Smith & Wang, 2006)	SOC_{max} (1)	0.975 (Zheng et al., 2013)
α_a (1)	0.5 (Smith & Wang, 2006)	SOC_{min} (1)	0 (Zheng et al., 2013)
$C_{s,max}$ $(\frac{mol}{m^3})$	48900		

Figure 3.2 was generated according to the procedure outlined here and illustrates several results that are of engineering interest: (a) the concentration of lithium ions in the electrolyte phase and (b) the lithium-ion concentration within the NMC particles under 3 C-rate discharge. The magnitude of the lithium-ion concentration in the electrolyte and the resulting gradient is an indication of concentration polarization in the cell, and the concentration in the particles constitute the SoL. Although several variables were solved for by the DNS solver, only the solid concentration C_s as a function of time was tracked for training the NN model. This corresponds to figures similar to Figure 3.2 (bottom), which for convenience we refer to as "SoL maps" (state-of-lithiation maps). The CNN was tasked to predict the fractional SoL map in each particle given the C-rate and time step as well as additional inputs. The concentration and time were exported from the DNS solver for the purpose of creating a dataset amenable to training a CNN, and this will be discussed in the following section.

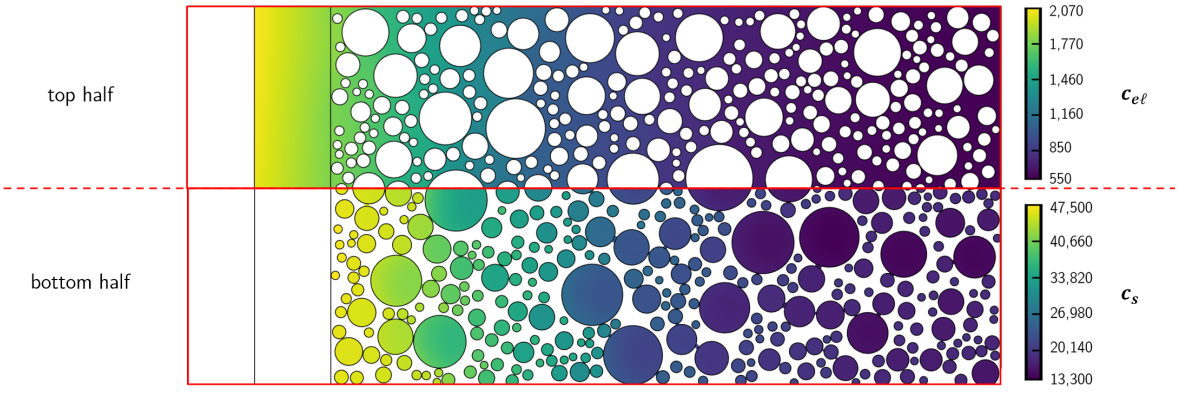

Figure 3.2: Results from FEM simulation for a microstructure under 3 C-rate discharge at $t = 600\ s$. C_e and C_s are the lithium-ion concentration in the electrolyte and NMC phase, respectively.

3.2 Dataset Generation

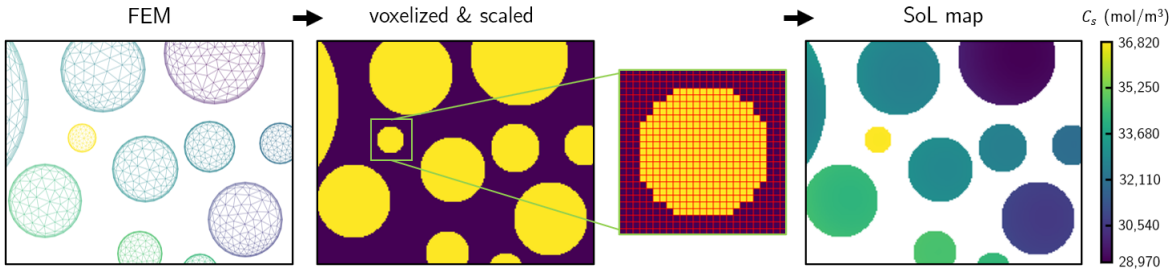

Figure 3.3: Data processing steps from the DNS solution (left) to a binarized image of the electrode microstructure (middle) to the target SoL map (right). The NN model only considers the SoL of the center particle for training/prediction, and the SoL of the neighboring particles is only shown here for illustration purposes.

Recall that the DNS solver stores SoL values at the FEM mesh nodes, so it cannot be directly fed into the CNN workflow. Therefore, the DNS results need to be processed into a pictorial format, i.e., as a 2D array of SoL values. Additionally, instead of having the ML model to directly output the SoL map of an entire microstructure, the model predicts the SoL of isolated particles to facilitate the learning process, as illustrated in Figure 3.3. In this figure, the first column represents the SoL distribution within a cropped region obtained from the FEM solution, and the last column represents the processed SoL to be used for training the NN model. For this purpose, the vertices of the NMC mesh from the FEM domain is projected onto a 2D array of 0s and 1s, representing the electrolyte and NMC particles, and then the FEM solution is interpolated over the mesh to fill in the "missing" pixels within the NMC particles. It should be noted that the NN model developed in this work was designed to predict the SoL of the center particle in each cropped out region. In Figure 3.3 (right), however, the SoL values for the neighboring particles are also shown, but only for illustration purposes. The following sections elaborate on the data processing steps.

3.2.1 Electrodes Used in Training

Starting from the DNS solution, the concentration of lithium in each NMC particle was saved from the simulations and normalized by the maximum intercalated lithium concentration, $C_{s,max}$, to obtain the fractional SoL. The ML dataset comprised of data from 60 random microstructure realizations in form

of circle packings with lengths of 48, 77, 101, 129, 154, and 176 μm, a constant width of 100 μm, porosities between 35% to 60%, and particle radii between 1 and 10 μm. The average statistics of the electrodes generated in this work is compared with values from a real microstructure in Table 3.2.

Table 3.2*:* The length, porosity, tortuosity, and mean radii of the 60 randomly generated microstructures. The length and porosity of a real NMC111 microstructure used in a coin cell was tabulated (Xu et al., 2019). The real microstructure tortuosity reported is evaluated from (Ebner et al., 2014) corresponding to a porosity of 0.5.

	60 Electrode Dataset			**Real Microstructure**
	Minimum	Average	Maximum	
Length (μm)	48	119	176	40
Porosity	0.391	0.491	0.609	0.5
Tortuosity	1.47	1.71	1.91	1.45 – 1.55
Mean Radius (μm)	1.67	2.29	3.33	N/A

FEM simulations of galvanostatic discharge were performed on these microstructures at 5 C-rates of 0.25, 0.5, 1, 2, and 3C. SoL values were evenly sampled 12 times during discharge based on the theoretical time to fully discharge a cell at the specified C-rate. For demonstration, the specifications of 5 out of 60 microstructures are tabulated in Table 3.3.

Table 3.3. Specification of five microstructures seen during training.

	Microstructures				
	I	**II**	**III**	**IV**	**V**
Length (μm)	154	176	48	176	129
Number of Particles	247	531	118	287	317
Porosity	0.55	0.43	0.49	0.53	0.58
Tortuosity	1.64	1.80	1.79	1.63	1.50
Mean Radius (μm)	2.53	1.97	2.12	2.50	2.91

3.2.2 Generating dataset: Voxelization of Microstructural images and SoL maps

The developed NN model takes in a binary image that represents the electrode microstructure with 0s and 1s representing the electrolyte and the NMC particles, respectively, and outputs an image of the same size with each representing the corresponding SoL. For each FEM simulation, the center coordinates, and the radii of the NMC particles are known. Also, simulations were conducted on an unstructured triangular mesh, so the interpolate module from SciPy was used to interpolate these values on an orthogonal grid to create 2D images of SoL. Further, the input images could be created from a threshold operation as values representing SoL belong to the NMC particle phase. The coordinates were scaled to obtain a resolution of $0.2\ \mu m$ per pixel.

3.2.3 Generating dataset: Isolation of Particles

In addition to the binary image, the EDT of the input image, as shown in Figure 3.4 (left), was fed to the NN model since it has been shown to facilitate the training (Santos et al., 2020, 2021). In particular, the motivation was the concentration profile would be higher at the outer radius and lower in the center, therefore the EDT of the image would look similar to the concentration profile within the particle. It is not strictly necessary to have the EDT of the image as the input to the network (though this is not a particularly expensive operation), one could opt to feed in the raw input image though it facilitated the training albeit not by a significant margin. As mentioned in previous sections, the NN model was designed to predict SoL for individual particles. Note, however, that the layout of the neighboring particles affects mass transport and consequently the SoL of the particle of interest. Therefore, the neighboring particles were also included in the input image to make NN predictions more robust. To standardize how many particles to include, starting from the center of an arbitrary particle of interest with a radius of R_i, the width and height of the input image was defined by $3R_i$. Since the particle sizes are distributed between $1\ \mu m$ and $10\ \mu m$, the extracted images will have different sizes. To standardize the inputs, the resulting images were scaled to be 150 by 150 pixels using the nearest neighbor interpolation. Note that the zoom factor for each image was recorded and fed to the NN model as metadata, which will be discussed in the following sections. Finally, since each input image consists of many particles, only one of which is of interest, using watershed segmentation a mask was generated for the particle of interest, as shown in Figure 3.4 (middle), which is later used by the NN model to only return the SoL for this particle, as shown in Figure 3.4 (right).

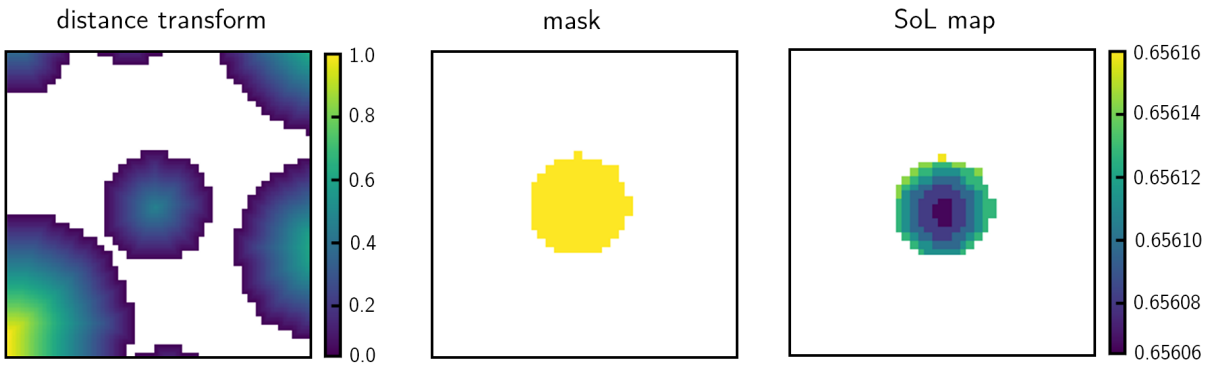

Figure 3.4. Each input image (left) is a cut-out of a region of the microstructure centered around a particle followed by the Euclidian distance transform. The figure in the middle is the mask of the particle of interest and the one on the right is the SoL map for the respective particle.

3.2.4 Metadata

In addition to the image-based inputs to the CNN, a feature vector was fed into the network to predict the SoL. This vector consists of 4 critical parameters necessary for the network to predict the dynamic nature of cell operation and the other 5 may be considered as metadata to facilitate the learning process. The parameters constituting the feature vector are tabulated in Table 3.4. The critical parameters were time, C-rate, zoom factor, and distance from separator. Many of the same images may be fed to the network which correspond to a particle of interest, but battery discharge is transient, thus without the time and C-rate parameters the output would be multimodal. Regarding distance from separator, during discharge the particles near the separator will reach a higher SoL sooner than the particles near the current collector so the distance from the separate was included in the metadata (Lu et al., 2021; Xu et al., 2019). Additionally, smaller particles typically experience uniform lithiation whereas larger particles would have a larger concentration gradient so the particle radius was provided (Lu et al., 2021; Xu et al., 2019). The term “metadata” is defined as information that would help the model train but are not critical. Porosity is an example of such; the model could theoretically infer the porosity from the image fed into the network but having this value precomputed would facilitate the model learn. It is a common practice to scale each variable to a standard range (i.e., $0 \rightarrow 1$) to help ML models better learn the data (Obaid, Hadeel S.; Dheyab, Saad Ahmed; Sabry, 9AD). These features are all trivial to obtain to do not require any computational resources that would undermine the speed-up obtained by the model.

Table 3.4: Features (metadata) used in the ML model to aid with predicting SoL. These values were simply scaled by their respective maximum value so their values, between $[\mathbf{0}, \mathbf{1}]$, are fed to the model with a similar magnitude.

Features	Normalization Value	Description
x	1	x-coordinate of center of particle (1)
y	100	y-coordinate of center of particle (μm)
R	10	radius of NMC particle (μm)
L	176	length of electrode (μm)
zoom factor	9	discussed in text
C-rate	3	discussed in text
Time	14400	time since discharge (s)
distance from Separator	1	discussed in text
porosity	1	discussed in text

3.3 Machine Learning Model

3.3.1 Machine Learning Architecture

The ML workflow in this work was written in Python and using the TensorFlow library (TensorFlow Developers, 2022). An autoencoder architecture was used to build the NN model. This architecture is composed of two parts, the encoder and the decoder as shown in Figure 3.5. The ReLU activation function was used for majority of the model. The encoder takes in a 2D image as input and compresses it by consecutively applying convolutional filters followed by max pooling with a stride of 2, which eventually results in a 1D array of compressed information. The decoder part, which is essentially the encoder but in reverse order, takes in this 1D array and decompresses it to eventually reconstruct a 2D image of the same size as the input. Upscaling is handled by nearest neighbour interpolation (Odena, Dumoulin, & Olah, 2017). An additional convolutional layer with the sigmoid activation function was added so the output would be bounded between 0 and 1. Finally, the feature vector was introduced to

the NN model by adding them to the compressed 1D array in the bottleneck section, i.e., the junction between the encoder and the decoder.

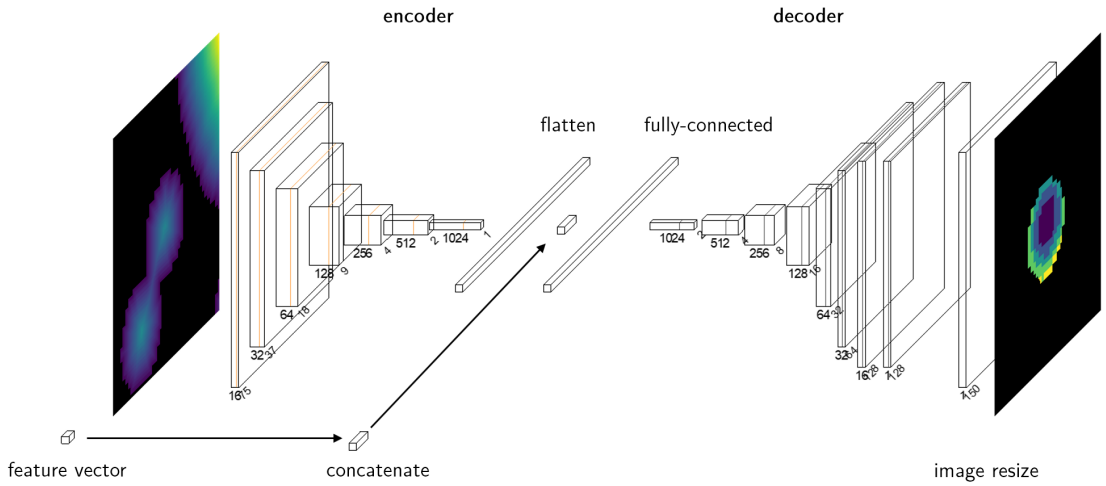

Figure 3.5: The image-to-image regression NN architecture used in this study. The images on the left and right are the input and target images, respectively. The blocks in the encoder and decoder sections represent Conv2D with max pooling and Conv2D with resizing layers each activated with ReLU.

3.3.2 Model Training

A common practice within ML is to split the overall dataset into training, validation, and test datasets. Optimization of the model is performed on training and validation is used to tune the hyperparameters of the model. In this work, this splitting was done on a randomly shuffled dataset, however, it would be prudent to perform shuffling according to a per electrode basis. Training was performed on a shuffled dataset on a per sample basis, though test data in Figure 4.1 is conducted on a separate dataset for an unbiased estimate of the performance of the model on unseen data – more details are provided in later sections. The data loading function are provided in Appendix A.2. The model was trained by minimizing the loss function as per:

$$\mathcal{L} = \frac{1}{N_s N_x N_y} \sum_{k=1}^{N_s} \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} \left(\hat{Y}_{kij} - Y_{kij}\right)^2 \tag{3.1}$$

where N_s is the number of samples, and N_x and N_y are the number of pixels along x and y axes, respectively. In simple terms, the loss function, $\mathcal{L}$, is defined as the mean squared error (MSE) between the predicted SoL and the ground truth SoL values from FEM simulations. The code for evaluating the loss function is provided in Appendix A.3. The root mean squared error (RMSE) is defined as:

$$\text{RMSE} = \sqrt{\mathcal{L}} \tag{3.2}$$

This value could also be reported in terms of a percentage in this case; SoL is fraction – from $0 - 1$ – so multiplication by 100 % reads as "x % SoL RMSE". To clarify, this interpretation of RMSE is possible as SoL can be interpreted as a percentage, so RMSE and root mean percentage squared error are separate entities.

3.3.3 Processing for ML-Predicted SoL Map and Averaged SoL Profile

Recall that the model predicts the SoL for isolated particles of interest, similar to Figure 3.4. These individual images could be reconstructed into a SoL map for comparison against the voxelized DNS equivalent. A flow chart depicting the algorithm for reconstructing a SoL map with predicted images from the NN model is shown in Figure 3.6 and the code is provided in Appendix A.4.

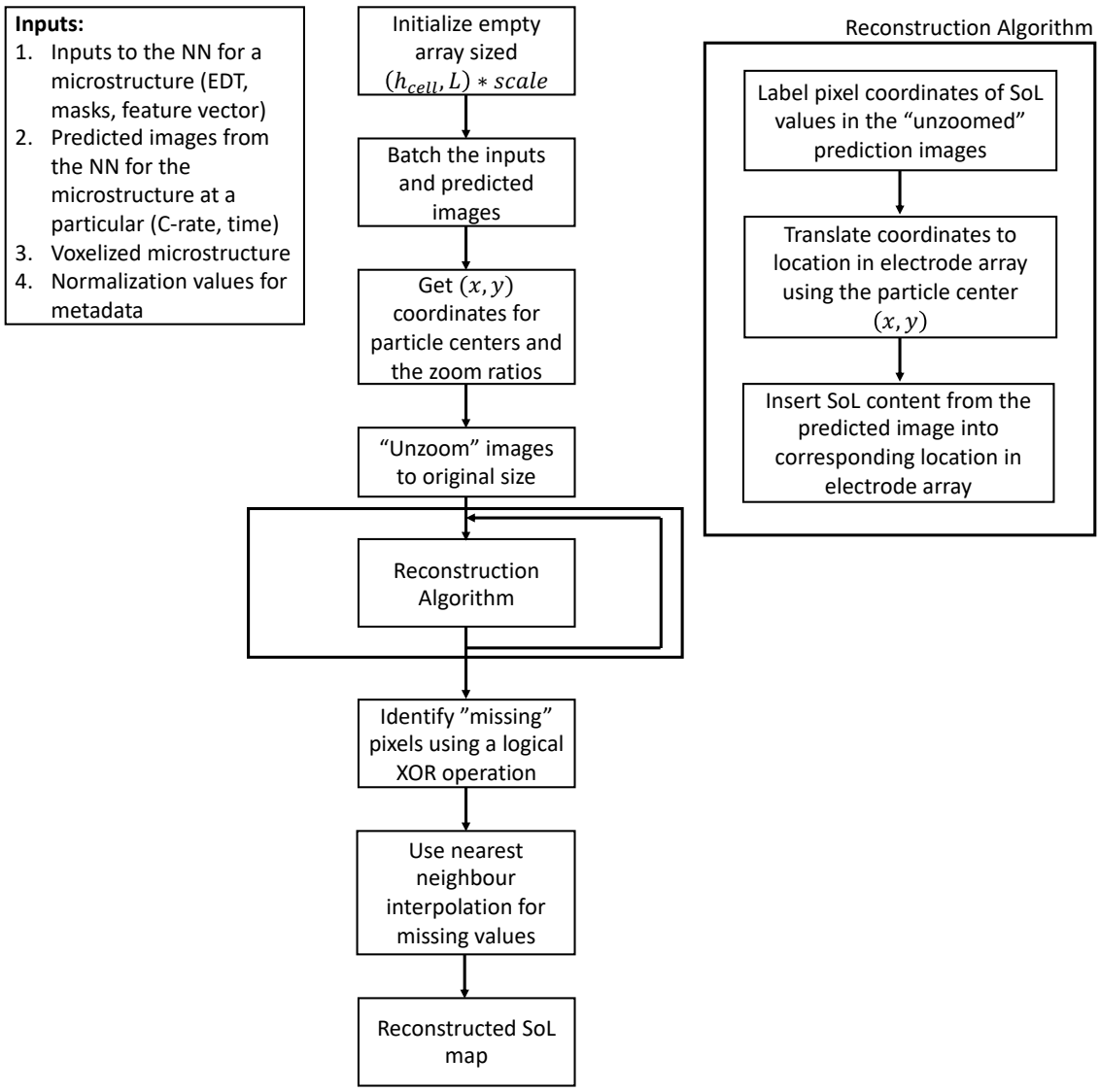

Figure 3.6: Algorithm for reconstructing a SoL map with the predicted images from the NN model. The inputs to the model are numbered on the top-left. Within the algorithm, the "reconstruction" step is performed for all of the particles in a microstructure; this step is shown on the top-right.

One quantitative metric to evaluate the performance of the model is to compare the reconstructed against the voxelized DNS SoL maps. The RMSE was used in this work. It should be noted that the pixels corresponding to the void phase were excluded and only pixels corresponding to the NMC phase were evaluated. In other words, the formula for this RMSE is similar to equation (3.2) but

normalization is the number of pixels corresponding to the NMC phase. Another visualization used in this work is plots of the average SoL along the length of the electrode. The average SoL at a certain point x was averaged across the y or in-plane direction of the cell. Also, the minimum, maximum, or standard deviation of SoL values could be estimated. The data for these plots were obtained for both reconstructed and voxelized DNS SoL maps.

Chapter 4
Results and Discussion

4.1 Generation of Reference Data

FEM simulations were performed on a 2021 MacBook Pro with the Apple M1 Pro chip. Generation of 60 electrodes with 5 C-rates until 100% Depth-of-Discharge (DoD) took ~ 26 hours The proposed ML workflow was benchmarked against FEM and tabulated for a half-cell configuration as per Table 4.1. This benchmark suggest that the proposed workflow is on average an order of magnitude faster than the DNS solution. Based on Table 4.1, the ML runtime for the C-rate of 0.25 is almost twice that of other C-rates. This extra overhead is because the first run involves two additional steps for computing the distance transform and the watershed transform of the input image, which are then cached to speed up predictions at other C-rates. The results indicate that the FEM runtime decreases with increasing C-rate while the ML model has a relatively constant runtime around 65 seconds. Increasing the C-rate decreases the discharge time so the DNS solver would have to perform less time-stepping whereas the ML framework typically outputs 12 SoL maps. It should be noted that the metadata incorporated in this work is relatively simple (see Table 3.4), with the most time-consuming metric computed being the zoom factor. If a three-dimensional surrogate would be constructed with this ML approach, convolutions would run in linear time. In comparison, solving the assembled matrices in the DNS run in either quadratic or cubic time depending on if an iterative or direct algorithm is used, respectively. Therefore, a significant speed-up may be possible in implementing a three-dimensional model similar to the model presented here.

Table 4.1: Runtime comparison between the ground-truth FEM and proposed ML-based framework. Benchmark was conducted on an electrode with a length of 176 $\boldsymbol{\mu m}$, porosity of 0.435, tortuosity of 1.86, and a total of 553 particles.

C-Rate	FEM runtime (s)	ML runtime (s)	Speedup
0.25	954	104.24	9.2
0.5	838	66.54	12.6
1	730	65.50	11.1
2	539	64.69	8.3
3	427	65.30	6.5

4.2 Model Training

A total of 60 electrodes were generated, on which battery discharge was simulated using FEM at the following C-rates: 0.25C, 0.5C, 1C, 2C and 3C. The resulting dataset had 1,009,065 samples which were generated by taking snapshots of the solution for a particular electrode and C-rate at different time steps. 70% of the data were used for training, 15% for validation, and the remaining 15% for testing. The inputs and targets were represented as pixel images, and the values were scaled between $[0, 1]$. The model was trained by minimizing the loss function as per equation (3.1). The NN was trained using a stochastic gradient descent approach. The Adam optimizer (Kingma & Ba, 2015) was used with a batch size of 256 (to saturate the GPU memory for optimal performance) with an initial learning rate of $lr = 10^{-6}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 10^{-7}$. The network was trained for 200 epochs where the learning rate was scheduled to halve every 50 epochs with an exponential decay between each epoch (Y. Wang, Xiao, & Cao, 2022). The training and validation loss were computed across the respective datasets after each epoch.

In terms of optimized hyper-parameters, two candidate autoencoder models were compared, the first with 16, 32, 64, 128, 256, and 512 filters and the second with an additional layer with 1024 filters, depicted in Figure 3.5. The training and validation loss of the big and smaller models were 2.52×10^{-6} and 3.89×10^{-5} compared to 8.63×10^{-6} and 2.95×10^{-5}. Thus, the smaller model was chosen based on its lower validation loss. The NN was trained using TensorFlow v2.10 on a server computer

with a 2.10 GHz Intel Xeon CPU and two Nvidia Quadro RTX 8000 48GB GPUs. Each epoch on average took 428 seconds for a total training time of 1,428 minutes.

4.2.1 Loss Curves

Figure 4.1 shows the loss curves for training and test datasets. It should be noted that this curve is not from the model training but was constructed *post hoc*. The training data in the figure was obtained in a similar fashion to the training data used during model training, *viz.* a 70% random shuffling on a per sample basis. The test data shown represents data generated from two electrodes separately from the original 60 electrode with new random particle arrangements. Based on this figure, the training loss is decreasing indefinitely while the test loss is at its minimum at around epoch 25, and then starts to increase. This indicates that the model starts to overfit (i.e., memorize the features in the training dataset) at epoch 25 and therefore, the model state at this stage was stored for later evaluations and is referred to as the "best model".

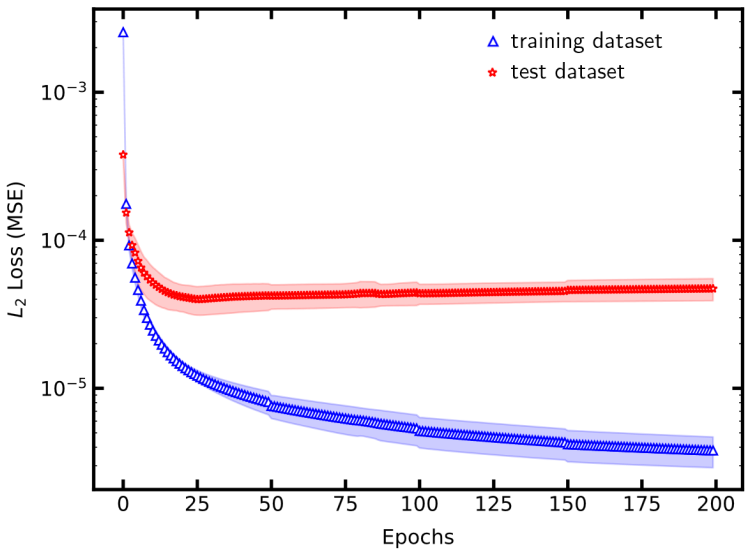

Figure 4.1: Training data from the randomly shuffled dataset on a per image basis, 70% of the overall dataset was used (blue triangle markers). The red star markers represent data from two electrodes generated separately and is unseen data. The shaded regions are one standard deviation of loss values obtained from three independent trainings. The two-electrode dataset loss suggests that optimal model performance occurs at around epoch 25.

4.3 Model Evaluation

To evaluate the best model in physical terms rather than just reporting the L_2 loss, the average relative error of SoL was computed for both the training and test datasets. For the training set, the average relative error was evaluated to be $\sim 0.3\%$, which is not surprising since the NN model has already "seen" them during training. For the test set, which have not been seen by the NN during training, the average relative error was evaluated to be less than 1%. This performance indicates that the NN model has been able to generalize well beyond the training set. For qualitative demonstration, Figure 4.2 shows the SoL values for isolated particles predicted by the NN model against ground truth for randomly selected particles at different time points and C-rates, accompanied by the respective relative error computed at individual pixels. Based on our analyses and as confirmed by this figure, not only the average relative error for the entire test dataset is below 1%, but that for individual particles and even individual pixels within each particle is still maintained below 1%.

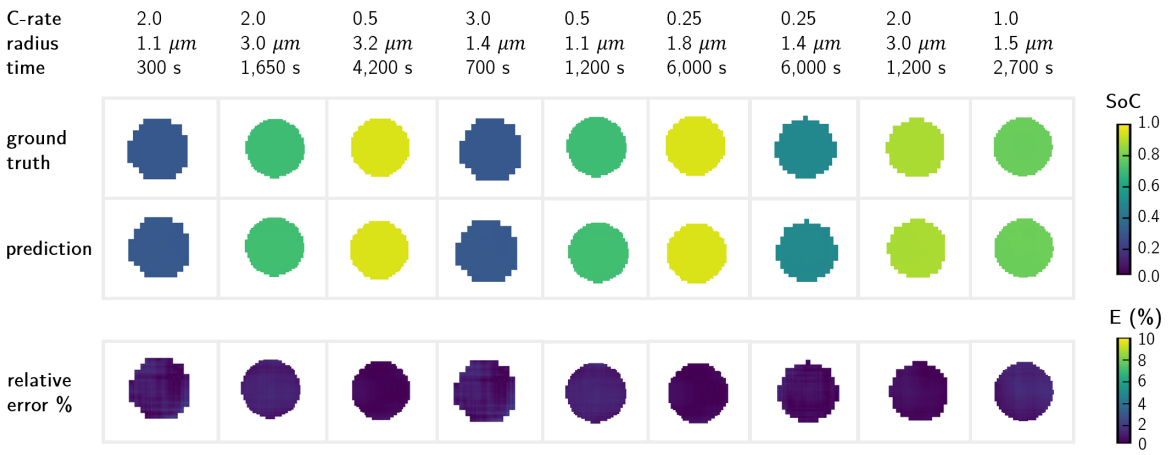

Figure 4.2: Comparison of ML predictions for SoL against ground truth for randomly selected particles from the test dataset.

Since concentration gradients within the particles have been shown to correlate to stress (Fathiannasab et al., 2021) and therefore degradation, it is of interest to see if the ML model is able to predict such gradients. Although the results presented in Figure 4.2 show an acceptable agreement between ML predictions and ground truth for SoL, it is difficult to judge model accuracy for SoL gradients,

especially that such gradients are very subtle within each particle and therefore, the color bar needs to be rescaled to minimum and maximum SoL of individual particles.

4.4 Comparison on Whole Electrode

Thus far, an ML model has been trained to predict the SoL for isolated particles in a given electrode microstructure using the procedure described in section 3.2 and with representative results shown in Figure 4.2. However, the current form is not of practical use, so the predicted images must be recombined to create a continuous SoL map for the entire electrode. Figure 4.3 shows the SoL map for the entire electrode at different C-rates at 50% depth of discharge as predicted by the ML model and compared against the FEM solution for a microstructure randomly selected from the test dataset.

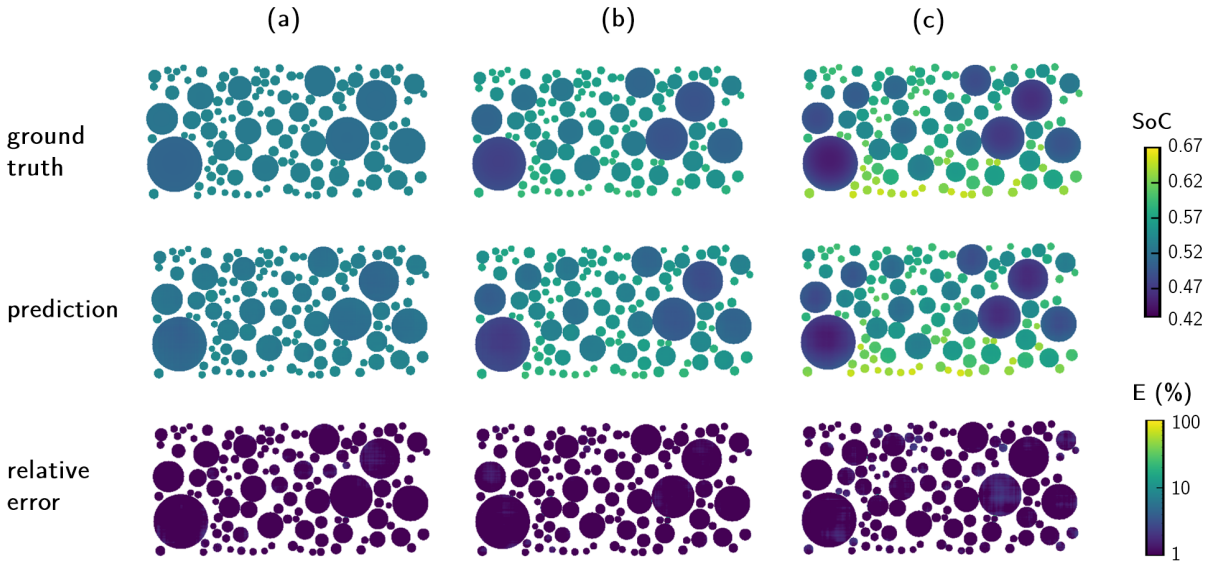

Figure 4.3: SoL (colorbar) at 50% discharge capacity based on C-rates, current flows from bottom to up. The generated solmaps are for microstructure 3, as tabulated in Table 3.3. Subplots (a) are for 0.25C at 7200s, (b) for 1C at 1800s, and (c) for 3C at 600s, the left and right panels are solmaps generated from the Machine Learning model and from FEM, respectively. Note the logarithmic scaling of the relative error color map.

The ML predictions in this figure have been reconstructed, i.e., the ML model was used to predict the SoL map for individual particles and in the end, they were combined to generate the SoL map shown in this figure. Based on this figure, the average relative error is consistently below 5%, which indicates that the ML model has been able to generalize beyond the training dataset with acceptable accuracy. Although the average relative error for all three C-rates is below 5%, the results show higher error as C-rate increases. To explain this behavior, note that NN models tend to perform better on continuous data (Grinsztajn, Oyallon, & Varoquaux, 2022). Consequently, relatively sharp gradients are more difficult to capture for NN models. In the context of this work, discharging a battery at high C-rates leads to a Li concentration distribution with relatively sharp gradients near the membrane, which possibly explains the worse performance of the ML model at high C-rates.

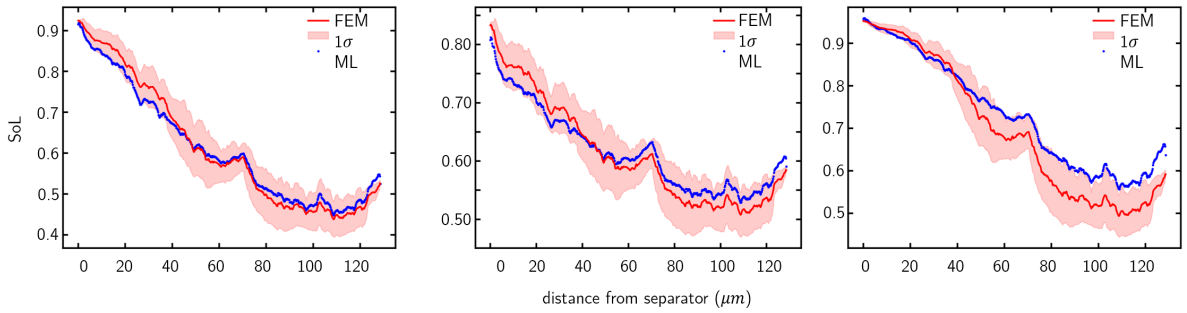

Figure 4.4: Plane-averaged SoL distribution for an unseen microstructure at 3C (left), 1.5C (middle), and 3.5C (right) discharge rates at 50% depth of discharge ($\boldsymbol{t = 900s}$) as a function of distance from the membrane. The solid line is the FEM solution, and the blue dots are ML predictions. The shaded region shows one standard deviation of SoL values for the FEM solution.

Turning attention now to the spatial distribution of SoL on the electrode scale, Figure 4.4 shows the plane-averaged SoL as a function of distance from the membrane at 3C (left), 1.5C (middle), and 3.5C (right) discharge rates and at the theoretical 50% depth of discharge, i.e., $t = 900\ s$, for an unseen microstructures. The shaded red region represents one standard deviation of SoL values for the FEM solution at the x-axis along the through-plane direction while the solid line and markers represent the average SoL at that point for FEM and ML model, respectively. Note that 3C discharge rate was seen during training (though the microstructure was not) unlike 1.5C and 3.5C, which were chosen to

evaluate the generalizability of the ML model in the interpolation and extrapolation regimes, respectively. Based on this figure, the maximum relative error for 3C, 1.5C, and 3.5C discharge rates were calculated to be 5%, 4.5%, and 15%, respectively, which suggests good agreement in the interpolation regime, and acceptable agreement in the extrapolation regime. This sharp increase in relative error in the extrapolation regime is not surprising since NN models typically perform best when operating in the interpolation regime (Chollet, 2021c). The SoL values were spatially pooled in Figure 4.5 to visualize the error of the unseen electrode during discharge at various C-rates. To help the NN model to extrapolate, one could use physics informed neural networks (PINNs), which at its core involves adding a physics-based term to the loss function (Raissi, Perdikaris, & Karniadakis, 2019), but that is beyond the scope of this work.

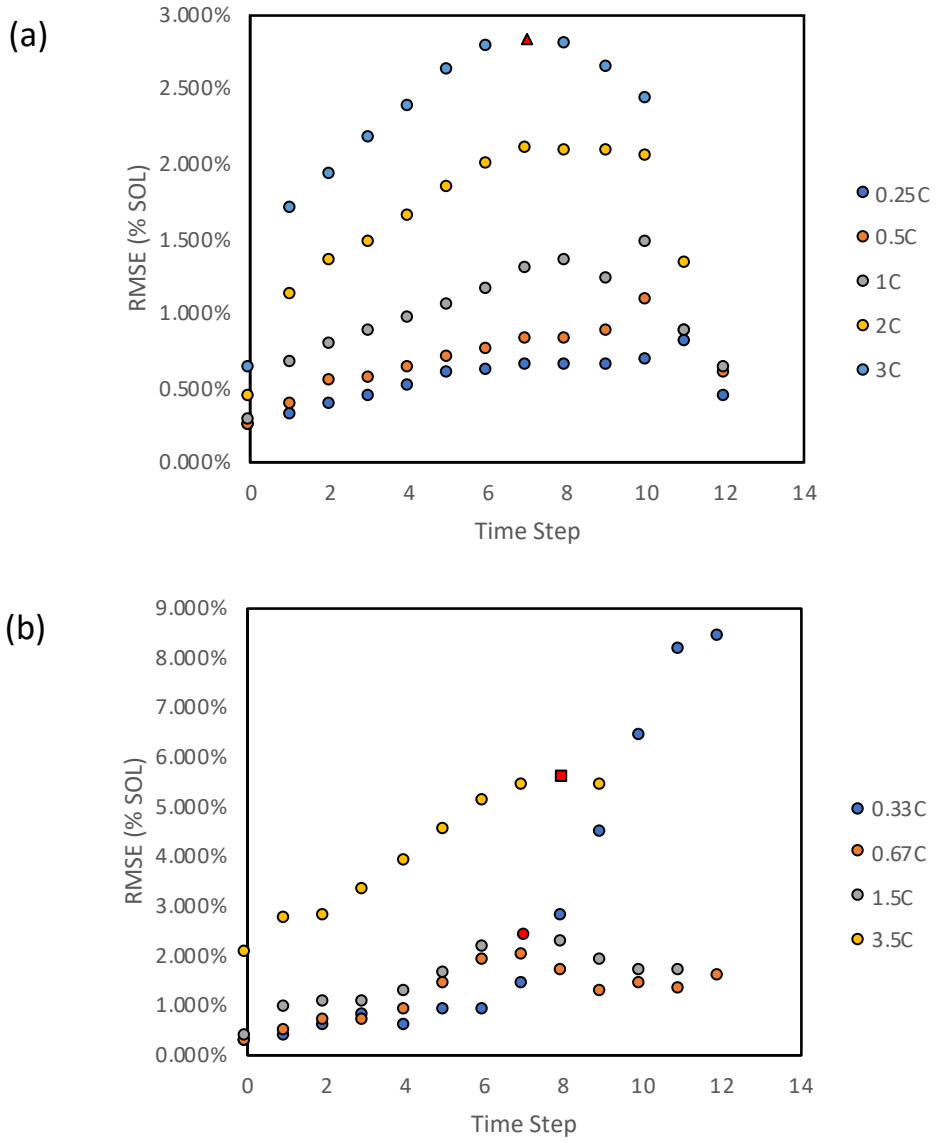

Figure 4.5: RMSE on the time scale between voxelized FEM and CNN-predicted solmaps from beginning of discharge to fully discharged. The time steps were normalized against the theoretical time of discharge, so a complete discharge would end at a time step of 12. Values were computed on C-rates seen (a) and not seen (b) during training, corresponding to the "unseen" electrode. The SoL profile across the length of the electrodes corresponding to red markers: triangle in (a), and square and circle (b); were plotted in Figure 4.4 to understand the performance of the CNN on the highest observed error values on a "seen", interpolated, and extrapolated C-rates, respectively.

Chapter 5
Conclusions and Recommendations

A machine learning framework was developed to explore the possibility of predicting the state-of-lithiation within the NMC particles of a Li-ion battery electrode. The model was intended as an accurate surrogate with the aim of rapidly predicting the SoL given a new microstructure to avoid the use of computationally expensive direct numerical simulations. To simplify this task in this proof-of-concept study, the particles were assumed to be circular, with radii between 1 and 10 μm, and with a porosity range of 35-60%. For establishing ground truth values required for training the ML model, FEM simulations of battery discharge were conducted at various C-rates and depth of discharge. As for the ML model, an auto encoder convolutional neural network was used and trained on the data generated from FEM simulations. The inputs to the ML model were the binary image representing the microstructure, as well as its Euclidean distance transform, which was used to facilitate the training. To further aid the training, additional metadata was also extracted and fed to the ML model. The ML model was designed to predict the SoL for isolated particles rather than for the entire electrode microstructure. For this reason, to predict the SoL for the entire electrode, the model needs to be run multiple times and the results pieced back together to reconstruct the entire electrode. Due to this approach, we evaluated the model performance at two levels: isolated particles, and the entire electrode. For the former, the model was shown to be able to predict the SoL of individual particles with reasonable accuracy, although it struggled to accurately predict the concentration gradient in larger particles at high C-rates. For the latter, however, the model was able to accurately capture the SoL distribution across the entire electrode with reasonable accuracy. We also evaluated the performance of the ML model in predicting the SoL for C-rates unseen during the training. We tested different C-rates both in the interpolation and extrapolation regimes. In the former, the model performed very similar to those C-rates seen during training with a maximum relative error of 5% (for a particular unseen microstructure), while in the latter, the model accuracy significantly dropped although it still maintained within an acceptable range with a maximum relative error of 15% (for the same microstructure). Conventional methods of predicting state-of-lithiation within electrode particles solve partial differential algebraic equations using numerical methods such as the finite element method. The main idea of this framework was to reduce the computation time to predict the state-of-lithiation for battery discharge in heterogeneous models. Our benchmarks indicate that computation time is at least an order of magnitude quicker in

comparison to finite element simulations, and we expect even higher speedups in the 3D case. This work – to the best of our knowledge – is the first to leverage neural networks to circumvent the requirement of solving PDEs in understanding battery discharge behavior. This work is a proof-of-concept which indicates that machine learning models could be a viable alternative to study battery behavior, although we considered a simplified circular geometry for NMC particles as a first approximation.

There are a few recommendations to expand upon the work presented here:

- This work could be extended to study battery discharge behavior on real tomography images. The usage of the watershed masks presented here is flexible and could be applied to segment individual particles in a real microstructure.
- The samples should be shuffled in terms of a per-electrode basis. Randomly shuffling the dataset in a per-sample basis can incorporate data from other microstructures unintentionally. It would be beneficial to have a larger validation dataset composed of randomly shuffled electrode data to do a more rigorous search of hyperparameters.
- Relating to the previous point, selection of optimal hyperparameters could be achieved through an ablation study. This means identifying a set of candidate hyperparameters and conduct a parametric study on how a hyperparameter affects the loss as well as identifying the values of the best parameters.
- The Laplacian convolutional filter could be used to construct an additional loss term in the optimization process to incorporate a physics-informed loss into the neural network. It would be interesting to see if the concentration gradients are improved in the prediction.

References

Agostini, L. (2020). Exploration and prediction of fluid dynamical systems using auto-encoder technology. *Physics of Fluids*, *32*(6). https://doi.org/10.1063/5.0012906

Ai, W., Kraft, L., Sturm, J., Jossen, A., & Wu, B. (2020). Electrochemical Thermal-Mechanical Modelling of Stress Inhomogeneity in Lithium-Ion Pouch Cells. *Journal of The Electrochemical Society*, *167*(1), 013512. https://doi.org/10.1149/2.0122001jes

Barber, C. B., Dobkin, D. P., & Huhdanpaa, H. (1996). The Quickhull Algorithm for Convex Hulls. *ACM Transactions on Mathematical Software*, *22*(4), 469–483. https://doi.org/10.1145/235815.235821

Bazant, M. Z. (2013). Theory of chemical kinetics and charge transfer based on nonequilibrium thermodynamics. *Accounts of Chemical Research*, *46*(5), 1144–1160. https://doi.org/10.1021/ar300145c

Bizeray, A. M., Howey, D. A., & Monroe, C. W. (2016). Resolving a Discrepancy in Diffusion Potentials, with a Case Study for Li-Ion Batteries. *Journal of The Electrochemical Society*, *163*(8), E223–E229. https://doi.org/10.1149/2.0451608jes

Cano, Z. P., Banham, D., Ye, S., Hintennach, A., Lu, J., Fowler, M., & Chen, Z. (2018). Batteries and fuel cells for emerging electric vehicle markets. *Nature Energy*, *3*(4), 279–289. https://doi.org/10.1038/s41560-018-0108-1

Chen, K. H., Namkoong, M. J., Goel, V., Yang, C., Kazemiabnavi, S., Mortuza, S. M., … Dasgupta, N. P. (2020). Efficient fast-charging of lithium-ion batteries enabled by laser-patterned three-dimensional graphite anode architectures. *Journal of Power Sources*, *471*(June), 228475. https://doi.org/10.1016/j.jpowsour.2020.228475

Chollet, F. (2021a). 8.1.1. The convolution operation. *Deep Learning with Python*, 204–209.

Chollet, F. (2021b). *Deep Learning with Python*. *Deep Learning with Python*. https://doi.org/10.1007/978-1-4842-5364-9

Chollet, F. (2021c). Fundamentals of Machine Learning. In *Deep Learning with Python* (2nd ed., pp. 146–177). Shelter Island: Manning.

Chollet, F. (2021d). Introduction to deep learning for computer vision. In *Deep Learning with Python* (2nd ed., pp. 201–237). Shelter Island: Manning.

Chollet, F. (2021e). Introduction to Keras and TensorFlow. In *Deep Learning with Python* (2nd ed., pp. 87–88). Shelter Island: Manning.

COMSOL Multiphysics. (2015). *The COMSOL Multiphysics Reference Manual*. Retrieved from https://doc.comsol.com/5.5/doc/com.comsol.help.comsol/COMSOL_ReferenceManual.pdf

Cooley, James W.; Lewis, Peter A. W.; Welch, P. D. (1967). Application of the Fast Fourier Transform to Computation of Fourier Integrals, Fourier Series, and Convolution Integrals. *IEEE Transactions on Audio and Electroacoustics*, *15*(2), 79–84. https://doi.org/10.1109/TAU.1967.1161904

Doyle, M., Fuller, T. F., & Newman, J. (1993). Modeling of Galvanostatic Charge and Discharge of the Lithium/Polymer/Insertion Cell. *Journal of The Electrochemical Society*, *140*(6), 1526–1533. https://doi.org/10.1149/1.2221597

Doyle, M., & Newman, J. (1995). The use of mathematical modeling in the design of lithium/polymer battery systems. *Electrochimica Acta*, *40*(13–14), 2191–2196. https://doi.org/10.1016/0013-4686(95)00162-8

Doyle, M., Newman, J., Gozdz, A. S., Schmutz, C. N., & Tarascon, J. (1996). Comparison of Modeling Predictions with Experimental Data from Plastic Lithium Ion Cells. *Journal of The Electrochemical Society*, *143*(6), 1890–1903. https://doi.org/10.1149/1.1836921

Ebner, M., Chung, D. W., García, R. E., & Wood, V. (2014). Tortuosity anisotropy in lithium-ion battery electrodes. *Advanced Energy Materials*, *4*(5), 1–6. https://doi.org/10.1002/aenm.201301278

Environment and Climate Change Canada. (2022). *Canadian Environmental Sustainability Indicators: Greenhouse gas emissions*. Gatineau. https://doi.org/10.1016/B0-12-348530-4/00094-1

Fathiannasab, H., Zhu, L., & Chen, Z. (2021). Chemo-mechanical modeling of stress evolution in all-solid-state lithium-ion batteries using synchrotron transmission X-ray microscopy tomography. *Journal of Power Sources*, *483*(September 2020), 229028. https://doi.org/10.1016/j.jpowsour.2020.229028

Gallagher, K. G., Trask, S. E., Bauer, C., Woehrle, T., Lux, S. F., Tschech, M., … Jansen, A. N. (2016). Optimizing Areal Capacities through Understanding the Limitations of Lithium-Ion Electrodes. *Journal of The Electrochemical Society*, *163*(2), A138–A149. https://doi.org/10.1149/2.0321602jes

Glorot, Xavier; Bordes, Antoine; Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, PMLR*, *15*, 315–323. https://doi.org/10.1002/ecs2.1832

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research*, *9*, 249–256.

Goodfellow, I; Bengio, Y.; Courville, A. (2016a). Architecture Design. In *Deep Learning* (pp. 197–203). Boston, Massachusetts: The MIT Press.

Goodfellow, I; Bengio, Y.; Courville, A. (2016b). Autoencoders. In *Deep Learning* (pp. 502–525). Boston, Massachusetts: The MIT Press.

Goodfellow, I; Bengio, Y.; Courville, A. (2016c). Convolutional Networks. In *Deep Learning* (pp. 330–372). Boston, Massachusetts: The MIT Press.

Goodfellow, I; Bengio, Y.; Courville, A. (2016d). Deep Feedforward Networks. In *Deep Learning* (pp. 204–221). Boston, Massachusetts: The MIT Press.

Goodfellow, I; Bengio, Y.; Courville, A. (2016e). Numerical Computation. In *Deep Learning* (p. 85). Boston, Massachusetts: The MIT Press.

Grinsztajn, L., Oyallon, E., & Varoquaux, G. (2022). Why do tree-based models still outperform deep learning on tabular data?, (NeurIPS). Retrieved from http://arxiv.org/abs/2207.08815

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Decem*, 770–778. https://doi.org/10.1109/CVPR.2016.90

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366. https://doi.org/10.1016/0893-6080(89)90020-8

Hutzenlaub, T., Thiele, S., Paust, N., Spotnitz, R., Zengerle, R., & Walchshofer, C. (2014). Three-dimensional electrochemical Li-ion battery modelling featuring a focused ion-beam/scanning

electron microscopy based three-phase reconstruction of a LiCoO2 cathode. *AABC Asia 2014 - Advanced Automotive Battery Confernce, AABTAM Symposium - Advanced Automotive Battery Technology, Application and Market, AABTAM 2014*, *21*, 131–139. https://doi.org/10.1016/j.electacta.2013.10.103

IPCC. (2014). *Climate Change 2014 Synthesis Report Summary Chapter for Policymakers*. *Ipcc*. Retrieved from https://archive.ipcc.ch/pdf/assessment-report/ar5/syr/AR5_SYR_FINAL_SPM.pdf

IRENA. (2017). *Electric Vehicles: Technology Brief*. Abu Dhabi. Retrieved from https://www.irena.org/publications/2017/Feb/Electric-vehicles-Technology-brief

Karpathy, Andrej; Li, F.-F. (2022). CS231n Convolutional Neural Networks for Visual Recognition. Retrieved March 26, 2023, from https://cs231n.github.io/convolutional-networks/

Kashkooli, A. G., Farhad, S., Lee, D. U., Feng, K., Litster, S., Babu, S. K., … Chen, Z. (2016). Multiscale modeling of lithium-ion battery electrodes based on nano-scale X-ray computed tomography. *Journal of Power Sources*, *307*, 496–509. https://doi.org/10.1016/j.jpowsour.2015.12.134

Khan, Z. A., Agnaou, M., Sadeghi, M. A., Elkamel, A., & Gostick, J. T. (2021). Pore Network Modelling of Galvanostatic Discharge Behaviour of Lithium-Ion Battery Cathodes. *Journal of The Electrochemical Society*, *168*(7), 070534. https://doi.org/10.1149/1945-7111/ac120c

Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 1–15.

Kirk, T. L., Please, C. P., & Jon Chapman, S. (2021). Physical Modelling of the Slow Voltage Relaxation Phenomenon in Lithium-Ion Batteries. *Journal of The Electrochemical Society*, *168*(6), 060554. https://doi.org/10.1149/1945-7111/ac0bf7

Krizhevsky, Alex; Sutskever, Ilya; Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, *1*, 1097–1105. https://doi.org/10.1201/9781420010749

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, *1*(4), 541–551. https://doi.org/10.1162/neco.1989.1.4.541

Li, J., Lin, C., Weng, M., Qiu, Y., Chen, P., Yang, K., … Pan, F. (2021). Structural origin of the high-voltage instability of lithium cobalt oxide. *Nature Nanotechnology*, *16*(5), 599–605. https://doi.org/10.1038/s41565-021-00855-x

Lu, X., Bertei, A., Finegan, D. P., Tan, C., Daemi, S. R., Weaving, J. S., … Shearing, P. R. (2020). 3D microstructure design of lithium-ion battery electrodes assisted by X-ray nano-computed tomography and modelling. *Nature Communications*, *11*(1), 1–13. https://doi.org/10.1038/s41467-020-15811-x

Lu, X., Zhang, X., Tan, C., Heenan, T. M. M., Lagnoni, M., O'Regan, K., … Shearing, P. R. (2021). Multi-length scale microstructural design of lithium-ion battery electrodes for improved discharge rate performance. *Energy and Environmental Science*, *14*(11), 5929–5946. https://doi.org/10.1039/d1ee01388b

Marquis, S. G., Sulzer, V., Timms, R., Please, C. P., & Chapman, S. J. (2019). An Asymptotic Derivation of a Single Particle Model with Electrolyte. *Journal of The Electrochemical Society*, *166*(15), A3693–A3706. https://doi.org/10.1149/2.0341915jes

Montúfar, G., Pascanu, R., Cho, K., & Bengio, Y. (2014). On the number of linear regions of deep neural networks. *Advances in Neural Information Processing Systems*, *4*(January), 2924–2932.

Newman, John; Bennion, Douglas; Tobias, C. W. (1965). Mass Transfer in Concentrated Binary Electrolytes. *Berichte Der Bunsengesellschaft*, *69*(7), 561–656. https://doi.org/https://doi.org/10.1002/bbpc.19650690712

Newman, John; Thomas-Alyea, K. E. (2004). *Electrochemical Systems*. *Nucl. Phys.* (3rd ed., Vol. 13). Hoboken: John Wiley & Sons, Inc.

Newman, John; Tiedemann, W. (1975). Porous-Electrode Theory with Battery Applications. *American Institute of Chemical Engineers Journal*, *21*(1), 25–41. https://doi.org/https://doi.org/10.1002/aic.690210103

O'Shea, Keiron; Nash, R. (2015). An Introduction to Convolutional Neural Networks. https://doi.org/https://doi.org/10.48550/arXiv.1511.08458

Obaid, Hadeel S.; Dheyab, Saad Ahmed; Sabry, S. S. (9AD). The Impact of Data Pre-Processing Techniques and Dimensionality Reduction on the Accuracy of Machine Learning. *2019 9th Annual Information Technology, Electromechanical Engineering and Microelectronics Conference (IEMECON)*, 279–283. https://doi.org/10.1109/IEMECONX.2019.8877011

Odena, A., Dumoulin, V., & Olah, C. (2017). Deconvolution and Checkerboard Artifacts. *Distill*, *1*(10), 1–8. https://doi.org/10.23915/distill.00003

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *30th International Conference on Machine Learning, ICML 2013*, (PART 3), 2347–2355.

Pratt, W. K. (2001). Edge Detection. In *Digitial Image Processing: PIKS Scientific Inside* (3rd ed., pp. 443–508). New York: John Wiley & Sons, Inc.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, *378*, 686–707. https://doi.org/10.1016/j.jcp.2018.10.045

Regulator, C. E. (2021). Provincial and Territorial Energy Profiles – Saskatchewan. Retrieved from https://www.cer-rec.gc.ca/en/data-analysis/energy-markets/provincial-territorial-energy-profiles/provincial-territorial-energy-profiles-quebec.html

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Segmentation. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9351*, 234–241. https://doi.org/https://doi.org/10.1007/978-3-319-24574-4_28

Rosenfeld, A., & Pfaltz, J. L. (1968). Distance functions on digital pictures. *Pattern Recognition*, *1*(1), 33–61. https://doi.org/10.1016/0031-3203(68)90013-7

Rumelhart, D. E. ., & Hinton, G. E.; Williams, R. J. (1986). Learning Representations by Back-Propagating Errors. *Nature*, *323*, 533–536. https://doi.org/https://doi.org/10.1038/323533a0

Salgado, R. M., Danzi, F., Oliveira, J. E., El-Azab, A., Camanho, P. P., & Braga, M. H. (2021). The latest trends in electric vehicles batteries. *Molecules*. https://doi.org/10.3390/molecules26113188

Santos, J. E., Xu, D., Jo, H., Landry, C. J., Prodanović, M., & Pyrcz, M. J. (2020). PoreFlow-Net: A 3D convolutional neural network to predict fluid flow through porous media. *Advances in Water Resources*, *138*(November 2019). https://doi.org/10.1016/j.advwatres.2020.103539

Santos, J. E., Yin, Y., Jo, H., Pan, W., Kang, Q., Viswanathan, H. S., … Lubbers, N. (2021). *Computationally Efficient Multiscale Neural Networks Applied to Fluid Flow in Complex 3D Porous Media*. *Transport in Porous Media* (Vol. 140). Springer Netherlands. https://doi.org/10.1007/s11242-021-01617-y

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization? *Advances in Neural Information Processing Systems*, *2018-December*(NeurIPS), 2483–2493.

Seshu, P. (2004). 1. Introduction. In *Textbook of Finite Element Analysis* (1st ed., pp. 1–4). New Delhi: Prentice Hall India. Retrieved from http://www.worldcat.org/title/textbook-of-finite-element-analysis/oclc/166413523&referer=brief_results

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 1–14.

Smith, K., & Wang, C. Y. (2006). Solid-state diffusion limitations on pulse operation of a lithium ion cell for hybrid electric vehicles. *Journal of Power Sources*, *161*(1), 628–639. https://doi.org/10.1016/j.jpowsour.2006.03.050

Sulzer, V., Chapman, S. J., Please, C. P., Howey, D. A., & Monroe, C. W. (2019). Faster Lead-Acid Battery Simulations from Porous-Electrode Theory: Part II. Asymptotic Analysis. *Journal of The Electrochemical Society*, *166*(12), A2372–A2382. https://doi.org/10.1149/2.0441908jes

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017). Inception-v4, inception-ResNet and the impact of residual connections on learning. *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, 4278–4284. https://doi.org/10.1609/aaai.v31i1.11231

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., … Rabinovich, A. (2015). Going deeper with convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *07-12-June-2015*, 1–9. https://doi.org/10.1109/CVPR.2015.7298594

TensorFlow Developers. (2022). TensorFlow (v2.8.2). *Zenodo*. https://doi.org/10.5281/ZENODO.6574269

Tjaden, B., Cooper, S. J., Brett, D. J., Kramer, D., & Shearing, P. R. (2016). On the origin and application of the Bruggeman correlation for analysing transport phenomena in electrochemical systems. *Current Opinion in Chemical Engineering*, *12*, 44–51. https://doi.org/10.1016/j.coche.2016.02.006

Tomaszewska, A., Chu, Z., Feng, X., O'Kane, S., Liu, X., Chen, J., … Wu, B. (2019). Lithium-ion battery fast charging: A review. *ETransportation*, *1*, 100011. https://doi.org/10.1016/j.etran.2019.100011

Torquato, S. (2002). 3. Statistical Mechanics of Many-Particle Systems. In S. Antman, S.S.; Marsden, J.E.; Sirovich, L.; Wiggins (Ed.), *Random Heterogeneous Materials* (pp. 78–114). New York: Springer Science.

Valøen, L. O., & Reimers, J. N. (2005). Transport Properties of LiPF6-Based Li-Ion Battery Electrolytes. *Journal of The Electrochemical Society*, *152*(5), A882. https://doi.org/10.1149/1.1872737

Van Der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., … Yu, T. (2014). Scikit-image: Image processing in python. *PeerJ*, *2014*(1), 1–18. https://doi.org/10.7717/peerj.453

Wang, F., & Tang, M. (2020). Thermodynamic Origin of Reaction Non-Uniformity in Battery Porous Electrodes and Its Mitigation. *Journal of The Electrochemical Society*, *167*(12), 120543. https://doi.org/10.1149/1945-7111/abb383

Wang, Y., Xiao, Z., & Cao, G. (2022). A convolutional neural network method based on Adam optimizer with power-exponential learning rate for bearing fault diagnosis. *Journal of Vibroengineering*, *24*(4), 666–678. https://doi.org/10.21595/jve.2022.22271

Xu, R., Yang, Y., Yin, F., Liu, P., Cloetens, P., Liu, Y., … Zhao, K. (2019). Heterogeneous damage in Li-ion batteries: Experimental analysis and theoretical modeling. *Journal of the Mechanics and Physics of Solids*, *129*(2019), 160–183. https://doi.org/10.1016/j.jmps.2019.05.003

Zeng, Y. (2015). *Mathematical Modeling of Lithium-ion Intercalation Particles and Their Dynamics*. Masschusetts Institute of Technology. Retrieved from

https://dspace.mit.edu/handle/1721.1/99062

Zheng, W., Shui, M., Shu, J., Gao, S., Xu, D., Chen, L., … Ren, Y. (2013). GITT studies on oxide cathode LiNi1/3Co1/3Mn 1/3O2 synthesized by citric acid assisted high-energy ball milling. *Bulletin of Materials Science*, *36*(3), 495–498. https://doi.org/10.1007/s12034-013-0480-1

Appendix A
Python Code

The samples of the Python code used in this work could be found here. Typing definitions used within the framework are defined herein. The Extract-Transform-Load functionality is used to load data from persistent memory into random-access memory, and the code for the custom loss function used during training is defined. The last code block reconstructs a SoL map from the output images as described in section 3.3.3. The table below specifies the version numbers of the most important software packages used in this work.

Table A.1 Key Python software packages used in this work and the specific versions.

Software	Version
NumPy	1.22.4
Numba	0.56
Pandas	1.4.3
PoreSpy	2.1
SciKit-Image	0.19.3
SciPy	1.9.0
TensorFlow	2.10

1. Typing Definitions

```
01: from typing import Tuple, TypedDict, List, Callable
02: import numpy as np
03: import tensorflow as tf
04:
05: meshgrid = Tuple[np.ndarray, np.ndarray]
06:
07:
08: ETL_key_fn = Callable[[int], tf.types.experimental.TensorLike]
09: ETL_key_and_tensor_fn = Callable[
10:     [int, tf.types.experimental.TensorLike], tf.types.experimental.TensorLike
11: ]
```

```
12: ETL_fn = ETL_key_and_tensor_fn
13:
14:
15: class Circle_Info(TypedDict):
16:     x: str
17:     y: str
18:     R: str
19:
20:
21: class Microstructure_Data(TypedDict):
22:     id: int
23:     length: int
24:     porosity: str
25:     tortuosity: str
26:     circles: List[Circle_Info]
27:
28:
29: class Metadata(TypedDict):
30:     micro: str
31:     x: float
32:     y: str
33:     R: str
34:     L: int
35:     zoom_factor: float
36:     c_rate: str
37:     time: str
38:     dist_from_sep: float
39:     porosity: float
40:
41:
42: class Metadata_Normalizations(TypedDict):
43:     L: int
44:     h_cell: int
45:     R_max: float
46:     zoom_norm: float
47:     c_rate_norm: float
48:     time_norm: int
49:
50:
51: META_INDICES = {
```

```
52:     "x": 0,
53:     "y": 1,
54:     "zoom_factor": 4,
55: }
56:
```

2. Extract-Transform-Load Functionality

```
001: from typing import List, Dict, Tuple, Callable, Union
002:
003: import tensorflow as tf
004: import numpy as np
005:
006: from scipy.ndimage import zoom
007: from scipy import ndimage as ndi
008: from skimage.segmentation import watershed
009: import porespy as ps
010:
011: from utils import typings
012:
013:
014: class ETL_Functions():
015:
016:     @staticmethod
017:     def load_npy_arr_from_dir(
018:         pic_num: int,
019:         data_dir: str,
020:         img_dir: str,
021:         img_size: int,
022:         tf_img_size: int,
023:         order: int = 0,
024:     ) -> tf.types.experimental.TensorLike:
025:         path = tf.strings.join(
026:             [data_dir, "/", img_dir]
027:         )
028:         fname = tf.strings.join(
029:             [path, "/", tf.strings.as_string(pic_num), ".npy"])
030:
031:         img = tf.numpy_function(
032:             np.load, [fname], tf.uint16,
033:         )
```

```
034:        img = tf.cast(img, dtype=tf.float32)
035:
036:        zoom_factor = tf_img_size / img_size
037:        img = tf.py_function(
038:            lambda a: zoom(a, (zoom_factor, zoom_factor, 1), order=order),
039:            [img],
040:            tf.float32,
041:        )
042:        return img
043:
044:    @ staticmethod
045:    def gather_img_from_np_arr(
046:        idx_num: int,
047:        arr_imgs: np.ndarray,
048:        img_size: int,
049:        tf_img_size: int,
050:        order: int = 0,
051:    ) -> tf.types.experimental.TensorLike:
052:        img = tf.gather(arr_imgs, idx_num)
053:        img = tf.convert_to_tensor(img)
054:
055:        zoom_factor = tf_img_size / img_size
056:        img = tf.py_function(
057:            lambda a: zoom(a, (zoom_factor, zoom_factor, 1), order=order),
058:            [img],
059:            tf.float32,
060:        )
061:        return img
062:
063:    @ staticmethod
064:    def dist_transform_input(
065:        idx_num: int,
066:        inp_img: tf.types.experimental.TensorLike,
067:    ):
068:        # Solid and paddings
069:        thres = tf.math.less(inp_img, 1.0)
070:        distance = tf.py_function(
071:            ndi.distance_transform_edt, [thres], tf.float32
072:        )
073:
```

```
074:        # Normalize Distance Transform
075:        dist_max = tf.math.reduce_max(distance)
076:
077:        distance = tf.cast(distance, tf.float32)
078:        dist_max = tf.cast(dist_max, tf.float32)
079:
080:        distance = tf.math.scalar_mul(1 / dist_max, distance)
081:
082:        return distance
083:
084:    @ staticmethod
085:    def get_mask(
086:        idx_num: int,
087:        inp_img: tf.types.experimental.TensorLike,
088:        tf_img_size: int,
089:        width_wrt_radius: float = 3,
090:        encoding: float = 1.0,
091:    ):
092:        center_loc = tf_img_size // 2
093:
094:        # Threshold input image to boolean where `True` is the solid phase. The
095:        # solid phase is taken to be less than the pore encoding as greater
096:        # than the padding encoding of `0.0`.
097:        cond1 = tf.math.less(inp_img, encoding)
098:        cond2 = tf.math.greater(inp_img, 0.0)
099:
100:       thres = tf.math.logical_and(cond1, cond2)
101:       thres = tf.cast(thres, tf.bool)
102:
103:       # Use the EDT on the solid phase with `True` values
104:       distance = tf.py_function(
105:           ndi.distance_transform_edt, [thres], tf.float32)
106:
107:       # Return array where `True` entries represent the peaks
108:       peaks = tf.numpy_function(
109:           ps.filters.find_peaks, [distance], tf.bool,
110:       )
111:
112:       # Label the `peaks` in an array with unique integer numbers for the
113:       # peaks
```

```
114:     markers, _ = tf.py_function(
115:         ndi.label, [peaks], [tf.int32, tf.int64],
116:     )
117:
118:     # Take the negative of distance since watershed fills "basins"
119:     distance = tf.math.scalar_mul(-1., distance)
120:     # Watershed segmentation
121:     labels = tf.numpy_function(
122:         watershed, [distance, markers], tf.int32,
123:     )
124:
125:     # Find label of the center particle and then do a boolean
126:     # for the center image
127:     center_lab = labels[center_loc, center_loc]
128:
129:     # Get indices in the array which correspond to labels for the center
130:     # particle
131:     label_indices = tf.where(
132:         tf.math.equal(labels, center_lab)
133:     )
134:
135:     # Update a boolean image where `True` correspond to the watershed
136:     # region for the particle of interest
137:     shape = tf.shape(label_indices)
138:     update = tf.ones(shape[0], tf.bool)
139:
140:     mask_im = tf.zeros_like(thres, tf.bool)
141:     mask_im = tf.tensor_scatter_nd_update(
142:         mask_im, label_indices, update,
143:     )
144:
145:     perturb = tf.cast(
146:         tf.math.ceil(tf_img_size / width_wrt_radius / 8), tf.int32
147:     )
148:
149:     def update_mask_perturb(y, x, ret_im):
150:         # `y` and `x` are +1 or -1
151:         label = tf.where(
152:             tf.math.equal(labels, labels[
153:                 center_loc + y * perturb,
```

```
154:                 center_loc + x * perturb,
155:             ])
156:         )
157:         update = tf.ones(tf.shape(label)[0], tf.bool)
158:         ret_im = tf.tensor_scatter_nd_update(
159:             ret_im, label, update,
160:         )
161:         return ret_im
162:
163:     mask_im = update_mask_perturb(-1, -1, mask_im)
164:     mask_im = update_mask_perturb(-1, 1, mask_im)
165:     mask_im = update_mask_perturb(1, -1, mask_im)
166:     mask_im = update_mask_perturb(1, 1, mask_im)
167:
168:     return tf.math.logical_and(mask_im, thres)
169:
170:   @ staticmethod
171:   def format_metadata(
172:     idx_num: int,
173:     metadata_lookup_table: tf.lookup.StaticHashTable,
174:   ) -> tf.types.experimental.TensorLike:
175:     key = tf.strings.as_string(idx_num)
176:     as_str = metadata_lookup_table[key]
177:     str_nums = tf.strings.split(as_str, sep="-")
178:     ret = tf.strings.to_number(str_nums, out_type=tf.dtypes.float32)
179:     return ret
180:
181:   @ staticmethod
182:   def fake_output(
183:     idx_num: int,
184:     tf_img_size: int,
185:   ) -> tf.types.experimental.TensorLike:
186:     _ = idx_num
187:     blank_im = np.zeros(
188:         (tf_img_size, tf_img_size, 1)
189:     )
190:     blank_im = tf.convert_to_tensor(blank_im)
191:     return blank_im
192:
193:   @ staticmethod
```

```
194:    def configure_for_performance(
195:        ds: tf.data.Dataset,
196:        batch_size: int,
197:        AUTOTUNE,
198:    ):
199:        #
https://www.tensorflow.org/tutorials/load_data/images#using_tfdata_for_finer_control
200:        ds = ds.cache()
201:        ds = ds.batch(batch_size)
202:        ds = ds.prefetch(buffer_size=AUTOTUNE)
203:        return ds
204:
205:
206: class ETL_2D():
207:
208:    def __init__(
209:        self,
210:        metadata: Dict[str, typings.Metadata],
211:        metadata_norm: typings.Metadata_Normalizations,
212:        criteria_arr: List[int],
213:        batch_size: int,
214:        tf_img_size: int,
215:        process_input_fns: List[typings.ETL_fn],
216:        process_target_fns: List[typings.ETL_fn],
217:    ):
218:        self.AUTOTUNE = tf.data.AUTOTUNE
219:        self.metadata_norm = metadata_norm
220:        self.batch_size = batch_size
221:        self.tf_img_size = tf_img_size
222:
223:        self.metadata_lookup = self.get_static_hash_table(
224:            metadata,
225:            process_value_fn=lambda num: self._format_metadata(
226:                num,
227:                metadata,
228:            ),
229:            default_value="",
230:        )
231:
232:        self.starting_ds = tf.data.Dataset.from_tensor_slices(criteria_arr)
```

```
233:
234:        self.input_fns = process_input_fns
235:        self.output_fns = process_target_fns
236:
237:    def get_ml_dataset(self) -> tf.data.Dataset:
238:
239:        inp_ds = self.starting_ds.map(
240:            self._process_input_path, num_parallel_calls=self.AUTOTUNE)
241:        out_ds = self.starting_ds.map(
242:            self._process_output_path, num_parallel_calls=self.AUTOTUNE)
243:
244:        inp_ds = ETL_Functions.configure_for_performance(
245:            inp_ds, self.batch_size, self.AUTOTUNE)
246:        out_ds = ETL_Functions.configure_for_performance(
247:            out_ds, self.batch_size, self.AUTOTUNE)
248:
249:        ret_ds = tf.data.Dataset.zip((inp_ds, out_ds))
250:        return ret_ds
251:
252:    def amend_metadata_to_loader(
253:        self,
254:        input_ds: tf.data.Dataset,
255:        mask_ds: tf.data.Dataset,
256:        target_ds: tf.data.Dataset,
257:    ):
258:
259:        meta_ds = self.starting_ds.map(
260:            lambda arr_idx: ETL_Functions.format_metadata(
261:                arr_idx, self.metadata_lookup),
262:            num_parallel_calls=self.AUTOTUNE,
263:        )
264:        meta_ds = ETL_Functions.configure_for_performance(
265:            meta_ds, self.batch_size, self.AUTOTUNE)
266:
267:        ret_ds = tf.data.Dataset.zip(
268:            ((input_ds, mask_ds, meta_ds), target_ds)
269:        )
270:        return ret_ds
271:
272:    def _process_input_path(
```

```
273:        self,
274:        arr_idx: int,
275:        input_im: tf.types.experimental.TensorLike = None,
276:    ) -> Tuple[
277:            tf.types.experimental.TensorLike,
278:            tf.types.experimental.TensorLike,
279:            tf.types.experimental.TensorLike,
280:    ]:
281:        for fn in self.input_fns:
282:            input_im = fn(arr_idx, input_im)
283:
284:        # Use the "raw input" to create a mask for the center particle of
285:        # interest. Order matters (for `input_im`)
286:        mask_im = ETL_Functions.get_mask(
287:            arr_idx, input_im, self.tf_img_size,
288:        )
289:
290:        # Now create a normalized distance transform on the particle phase to
291:        # use as the input image to the Neural Network.
292:        input_im = ETL_Functions.dist_transform_input(arr_idx, input_im)
293:
294:        metadata = ETL_Functions.format_metadata(
295:            arr_idx,
296:            self.metadata_lookup,
297:        )
298:
299:        return input_im, mask_im, metadata
300:
301:    def _process_output_path(
302:        self,
303:        arr_idx: int,
304:        target_im: tf.types.experimental.TensorLike = None,
305:    ) -> tf.types.experimental.TensorLike:
306:        for fn in self.output_fns:
307:            target_im = fn(arr_idx, target_im)
308:
309:        return target_im
310:
311:    def get_static_hash_table(
312:        self,
```

```
313:        hash,
314:        process_value_fn: Callable[[int], str],
315:        default_value: str = "",
316:    ) -> tf.lookup.StaticHashTable:
317:        r''' The default value depends on what the data type of the key is.
318:        '''
319:        keys: List[Union[str, None]] = [None for _ in range(0, len(hash))]
320:        vals: List[Union[str, None]] = [None for _ in range(0, len(hash))]
321:
322:        for i, key in enumerate(hash.keys()):
323:            val = process_value_fn(key)
324:
325:            keys[i] = str(key)
326:            vals[i] = val
327:
328:        keys = tf.constant(keys)
329:        vals = tf.constant(vals)
330:
331:        ret_table = tf.lookup.StaticHashTable(
332:            tf.lookup.KeyValueTensorInitializer(keys, vals),
333:            default_value=default_value,
334:        )
335:
336:        return ret_table
337:
338:    def _format_metadata(
339:        self,
340:        pic_num: int,
341:        metadata: Dict[str, typings.Metadata],
342:    ) -> str:
343:
344:        hash = metadata[str(pic_num)]
345:
346:        x = float(hash["x"])
347:        y = float(hash["y"]) / self.metadata_norm["h_cell"]
348:        R = float(hash["R"]) / self.metadata_norm["R_max"]
349:        L = float(hash["L"]) / self.metadata_norm["L"]
350:        zoom = float(hash["zoom_factor"]) / self.metadata_norm["zoom_norm"]
351:        c_rate = float(hash["c_rate"]) / self.metadata_norm["c_rate_norm"]
352:        time = float(hash["time"]) / self.metadata_norm["time_norm"]
```

```
353:        porosity = float(hash["porosity"])
354:        dist_from_sep = float(hash["dist_from_sep"])
355:
356:        as_float = [x, y, R, L, zoom, c_rate, time, porosity, dist_from_sep]
357:
358:        s = "-"
359:        ret = s.join(str(num) for num in as_float)
360:        return ret
361:
362:
############################################################################
######
363: # DECORATORS
############################################################################
364:
############################################################################
######
365:
366:
367: def leave_first(fn, *args):
368:     def wrapped(
369:         first_arg: int,
370:     ) -> typings.ETL_key_fn:
371:         return fn(first_arg, *args)
372:     return wrapped
373:
374:
375: def ignore_key(
376:     fn: typings.ETL_key_fn,
377: ) -> typings.ETL_key_and_tensor_fn:
378:     def wrapped(arr_idx, tensor):
379:         _ = arr_idx
380:         return fn(tensor)
381:     return wrapped
382:
383:
384: def ignore_tensor(
385:     fn: typings.ETL_key_fn
386: ) -> typings.ETL_key_and_tensor_fn:
387:     def wrapped(arr_idx, tensor):
```

```
388:         _ = tensor
389:         return fn(arr_idx)
390:     return wrapped
391:
```

3. Loss Function

```
01: import tensorflow as tf
02:
03:
04: class Mask_MSE(tf.keras.losses.Loss):
05:
06:     def __init__(self, name="mask_mask"):
07:         super().__init__(name=name)
08:
09:     def call(self, y_true, y_pred):
10:         # Evaluate pixels on the intersection where `y_pred` and `y_true` have
11:         # valid input
12:
13:         # Just making sure we're not ignoring negatives, since if the
14:         # predictions are negative within the valid region then we're
15:         # artificially masking out these regions.
16:         mask_pred_greater = tf.math.greater(y_pred, 0.0)
17:         mask_pred_lesser = tf.math.less(y_pred, 0.0)
18:
19:         mask_pred = tf.math.logical_or(mask_pred_greater, mask_pred_lesser)
20:         mask_true = tf.math.greater(y_true, 0.0)
21:
22:         # Intersection between both masks
23:         mask = tf.math.logical_and(mask_pred, mask_true)
24:         mask = tf.cast(mask, tf.float32)
25:
26:         y_pred = tf.math.multiply(y_pred, mask)
27:         y_true = tf.math.multiply(y_true, mask)
28:
29:         return tf.math.reduce_mean(tf.square(y_pred - y_true))
30:
```

4. SoL Map Reconstruction

```
001: from typing import Tuple, Dict
```

```
002:
003: import numpy as np
004: import tensorflow as tf
005:
006: import scipy.ndimage as ndi
007: from scipy.interpolate import griddata
008:
009:
010: from utils import typings
011:
012:
013: ############################################################
014: # PREDICT "SOLMAP" FROM MACHINE LEARNING PREDICTIONS #
015: ############################################################
016:
017:
018: def electrode_sol_map_from_predictions(
019:     input_dataset: tf.data.Dataset,
020:     predicted_imgs: np.ndarray,
021:     micro_mask: np.ndarray,
022:     L_electrode: int,
023:     norm_metadata: Tuple[int, int, int, float, int, int],
024:     batch_size: int,
025:     scale: int = 10,
026: ) -> Tuple[np.ndarray, Dict[str, float]]:
027:     r'''`electrode_sol_map_from_predictions` takes an electrode (in the form of
028:     an tf.data.Dataset) its predicted SOL output at a certain C-rate and time
029:     step to "patch" all the particles back into whole electrode, thus returning
030:     a "State-of-Lithiation"-map output.
031:
032:     This output could be compared to ground-truth values from COMSOL to
033:     investigate the performance of the Machine Learning model, or as a stand-
034:     alone tool to get SOL maps quicker than the Direct Numerical Solution
035:     solution.
036:
037:     Note: "input_dataset" and "predicted_imgs" should be in the same order,
038:         i.e., they should not be shuffled. Otherwise, the SOL will not be in
039:         the correct order.
040:
041:     Inputs:
```

```
042:    - input_dataset: tf.data.Dataset
043:    - predicted_imgs: np.array
044:    - micro_mask: the microstructure with NMC phase a `True`
045:    - L_electrode: electrode length (um)
046:    - norm_metadata: values used to normalize the metadata during model
047:        training
048:    - batch_size: the `batch_size` of `input_dataset`
049:    - scale: int; scales the resolution of the outputted image.
050:    '''
051:
052:    tf.experimental.numpy.experimental_enable_numpy_behavior()
053:
054:    _, h_cell, _, zoom_norm, _, _ = norm_metadata
055:
056:    solmap = np.zeros(
057:        (h_cell * scale, L_electrode * scale, 1),
058:        dtype=np.float32,
059:    )
060:
061:    predicted_imgs_ds = tf.data.Dataset.from_tensor_slices(predicted_imgs)
062:    predicted_imgs_ds = predicted_imgs_ds.batch(batch_size)
063:
064:    # For each input (image, mask, metadata) in the dataset, extract the color
065:    # from the Machine Learning output and place it in the coordinate in the
066:    # electrode.
067:    for data_batch, pred_batch in zip(input_dataset, predicted_imgs_ds):
068:
069:        in_batch, _ = data_batch
070:        _, _, meta_batch = in_batch
071:
072:        x_centers = _ML_Pred_to_Solmap.get_meta_elem("x", meta_batch) * \
073:            tf.cast(L_electrode, tf.float32)
074:        y_centers = _ML_Pred_to_Solmap.get_meta_elem("y", meta_batch) * \
075:            tf.cast(h_cell, tf.float32)
076:        zoom_factors = _ML_Pred_to_Solmap.get_meta_elem(
077:            "zoom_factor", meta_batch) * tf.cast(zoom_norm, tf.float32)
078:
079:        # Zoom images back to their original sizes
080:        unzoomed_imgs = tf.map_fn(
081:            _ML_Pred_to_Solmap.zoom_tensor_ret_img,
```

```
082:         (pred_batch, zoom_factors),
083:         fn_output_signature=tf.RaggedTensorSpec(
084:           shape=None,
085:           ragged_rank=1,
086:           dtype=tf.float32,
087:         )
088:       )
089:
090:       # Get the center pixel of a prediction image
091:       prediction_centers = tf.map_fn(
092:         lambda im: tf.math.scalar_mul(
093:           tf.cast(1/2, tf.float32),
094:           tf.cast(tf.shape(im.to_tensor())[0], tf.float32),
095:         ),
096:         unzoomed_imgs,
097:         fn_output_signature=tf.TensorSpec(
098:           shape=(),
099:           dtype=tf.float32,
100:         )
101:       )
102:
103:       # Get a tensor of masks for the SoL values (RaggedTensor)
104:       sol_value_mask = tf.math.greater(unzoomed_imgs, 0.0)
105:
106:       for batch_idx in range(pred_batch.shape[0]):
107:         x = x_centers[batch_idx]
108:         y = y_centers[batch_idx]
109:         center = prediction_centers[batch_idx]
110:
111:         zoomed_pred_img = unzoomed_imgs[batch_idx].to_tensor()
112:
113:         # Meshgrids where a non-zero pixel is the x or y coordinate of a
114:         # pixel which has a predicted SoL value.
115:         sol_X, sol_Y = _ML_Pred_to_Solmap.sol_pixel_meshgrid(
116:           sol_value_mask[batch_idx])
117:
118:         # Translate the meshgrids so we are:
119:         #   - subtract predicted image center (circle centered at (0, 0))
120:         #   - add the scaled (x, y) center particle coordinates
121:         elec_X, elec_Y = _ML_Pred_to_Solmap.sol_to_electrode_meshgrid(
```

```
            (sol_X, sol_Y, sol_value_mask[batch_idx], x, y, center),
            scale,
        )

        elec_X = elec_X.numpy()
        elec_Y = elec_Y.numpy()

        # Take SoL values from the "unzoomed" prediction images and place
        # it in the corresponding meshgrid location in the SoLmap
        elec_Y[elec_Y >= np.shape(solmap)[0]] = np.shape(solmap)[0] - 1
        elec_X[elec_X >= np.shape(solmap)[1]] = np.shape(solmap)[1] - 1

        solmap[
            elec_Y,
            elec_X,
            :,
        ] = zoomed_pred_img.numpy()[sol_Y, sol_X, :]

solmap_mask = solmap > 0.0

# Find pixels that are not common to both the `microstructure` and `solmap`
mismatched_pixels = np.logical_xor(micro_mask, solmap_mask)
# Filter extra particle - only pixels in found in `micro_mask` are valid
mismatched_pixels = micro_mask * mismatched_pixels

# Perform nearest neighbour interpolation for the `missing_indices` using
# data from the `existing_sol_indices`
missing_indices = np.where(mismatched_pixels)
existing_sol_indices = np.where(solmap_mask)

interpolated_sol_values = griddata(
    existing_sol_indices,
    solmap[existing_sol_indices],
    missing_indices,
    method="nearest",
)

solmap[missing_indices] = interpolated_sol_values

# Another round of masking pixels (excess)
```

```
    excess_pixels = np.logical_xor(solmap > 0.0, micro_mask)
    solmap = ~excess_pixels * solmap

    # Return some statistics to show how many pixels were missing
    stats = {
        "num_pix_missing": np.sum(mismatched_pixels),
        "per_pix_missing": np.sum(mismatched_pixels) / np.prod(
            solmap_mask.shape) * 100,
    }

    return solmap, stats


class _ML_Pred_to_Solmap():

    @ staticmethod
    def get_meta_elem(
        elem: str,
        meta_tensor: tf.types.experimental.TensorLike
    ):
        return tf.gather(
            meta_tensor,
            typings.META_INDICES[elem],
            axis=1,
        )

    @ staticmethod
    def zoom_tensor_ret_img(inputs):
        img, zoom_factor = inputs

        out_img_size = 1 / zoom_factor

        unzoomed_img = tf.py_function(
            lambda arr: ndi.zoom(
                arr, (out_img_size, out_img_size, 1), order=0
            ),
            [img], tf.float32,
        )

        unzoomed_img = tf.RaggedTensor.from_tensor(unzoomed_img)
```

```
202:     return unzoomed_img
203:
204:   @ staticmethod
205:   def sol_pixel_meshgrid(sol_value_mask):
206:     # Get the pixel location of where SoL values are in the direct Machine
207:     # Learning predictions
208:
209:     zoomed_im = sol_value_mask.to_tensor()
210:     zoomed_img_size = tf.shape(zoomed_im)[0]
211:
212:     # Linearly spaced array
213:     img_linspace = tf.range(0, zoomed_img_size)
214:
215:     # Meshgrid with coordinates corresponding to `zoomed_im` size
216:     X, Y = tf.meshgrid(img_linspace, img_linspace)
217:
218:     def fn(T): return tf.math.multiply(
219:       T, tf.cast(zoomed_im[:, :, 0], tf.int32))
220:
221:     X = fn(X)
222:     Y = fn(Y)
223:
224:     return X, Y
225:
226:   @ staticmethod
227:   def sol_to_electrode_meshgrid(tup, scale):
228:     # There may be some controversy in choosing which pixel would be the
229:     # center either using `ceil`, `floor`, or `round` approach.
230:     #
231:     # For odd-sized images this is easy, it's just 1/2 the image rounded
232:     # down, but for evenly-sized images this might be controversial:
233:     #   5-pixels:
234:     #      5 / 2 = 2.5 ==> 2 ([] [] [x] [] [])
235:     #   4-pixels:
236:     #      4 / 2 = 2   ==> 2 ([] [x] [] [])
237:
238:     X, Y, sol_value_mask, x, y, center = tup
239:
240:     sol_value_mask = sol_value_mask.to_tensor()
241:
```

```
242:        def fn(T, coord):
243:            return tf.cast(sol_value_mask, tf.float32) * scale * coord + \
244:                (tf.cast(tf.expand_dims(T, axis=-1), tf.float32) - center)
245:
246:        # Translate SoL meshgrid to its location in the electrode
247:        elec_X = tf.cast(tf.math.round(fn(X, x)), tf.int64)
248:        elec_Y = tf.cast(tf.math.round(fn(Y, y)), tf.int64)
249:
250:        # Reshape since meshgrid should be (dim, dim) and not (dim, dim, 1)
251:        elec_X = elec_X[:, :, 0]
252:        elec_Y = elec_Y[:, :, 0]
253:
254:        return elec_X, elec_Y
255:
```