Machine Learning for Spectroscopic Data Analysis: Challenges of Limited Labelled Data

by

Matthew Dirks

B. Computer Information Systems, Okanagan College, 2011
M. Sc. Computer Science, University of British Columbia, 2014

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

The Faculty of Graduate and Postdoctoral Studies
(Computer Science)

The University of British Columbia
(Vancouver)

December 2023

© Matthew Dirks, 2023
The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

**Machine Learning for Spectroscopic Data Analysis: Challenges of Limited Labelled Data**

submitted by **Matthew Dirks** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy** in **Computer Science**.

**Examining Committee:**

David Poole, Professor, Department of Computer Science, University of British Columbia  
*Supervisor*

Jim Little, Professor emeritus, Department of Computer Science, University of British Columbia  
*Supervisory Committee Member*

Lee Groat, Professor, Earth, Department of Ocean and Atmospheric Sciences, University of British Columbia  
*Supervisory Committee Member*

Alan Wagner, Associate Professor, Department of Computer Science, University of British Columbia  
*University Examiner*

Bern Klein, Professor, Department of Mining Engineering, University of British Columbia  
*University Examiner*

Kristian Hovde Liland, Professor, Department of Mechanical Engineering and Technology Management, Norwegian University of Life Sciences  
*External Examiner*

**Additional Supervisory Committee Members:**

Siamak Ravanbakhsh, Assistant Professor, School of Computer Science, McGill University  
*Supervisory Committee Member*
Abstract

Extracting meaningful information from spectra, such as sample composition, proves to be challenging. Building prediction models with supervised learning requires labelled data which is often limited. To overcome the challenge of limited data, this thesis explores various strategies spanning the gamut from models reliant on domain knowledge to those primarily data-driven.

Leveraging domain knowledge, an approximate, fully-differentiable X-ray fluorescence (XRF) simulator is developed and used in two models. In the first model, the simulator is fit to an observed spectrum. The resulting parameter values are mapped to element concentrations by regression modelling. In the second model, the simulator is embedded in an auto-encoder (AE) neural network. The AE learns the inverse function of the simulator while also adapting to the data when instrument or environment parameters are unavailable. An experiment comparing the AE to standard regression models found improved predictions for 11 elements. Another AE model is developed that uses more general domain knowledge about spectra, which applies to any type of spectrum containing peak-shaped structures. With this model, a statistically significant decrease in prediction error compared to state-of-the-art models was found for predicting tin concentration (with $p < 0.00001$) in the results of 10×10-fold cross-validation, and it was tied for best on 11 out of 32 elements. A benefit of both AE models is that they can utilize unlabelled data in semi-supervised learning to lower the requirements for ground truth data.

Neural networks require extensive hyperparameter optimization (HPO) which depends on validation data to estimate performance accurately. HPO works poorly when the validation set score is noisy; noisy validation scores are typical of small datasets. Ensembling is used to lower the variance, resulting in a neural network configuration that performs as well as an expertly-chosen configuration.

A final prediction model combines information from multiple spectrometers, which is particularly challenging for small datasets. Several sensor fusion methods are compared, including a parallel-input convolutional neural network (CNN). Results of 10-fold cross-validation found that high-level PLS-based methods were best, though neural network models were competitive.
Lay Summary

A spectrum is a measure of electromagnetic radiation, such as light, given off by a sample. When interpreted correctly, spectral data can inform us about the sample, for example, the quality of a mango, the content of a rock, the cancer in a skin sample, and many other practical use cases. Predicting this information from a new spectrum is difficult. Predictive models range from physics-based, to being just learned from data. Learning from data typically requires training on a large dataset consisting of examples of spectra and ground truth; however, the number of examples is often limited (due to expense or other factors). A range of several AI models, leveraging different amounts of expert knowledge and data, is explored. These models are compared to standard models.
Preface

The research presented in this thesis comprises three published research projects led by the author and two unpublished projects. The author designed the experiments, implemented the solutions, programmed and executed the experiments, and analyzed the results. All projects were conducted under the supervision of David Poole.

- Chapter 2: Unpublished work, except for Section 2.3 which was part of the publication [69] in Chapter 3.

- Chapter 3: This work was published [69] in the journal of X-ray Spectrometry, titled “Auto-Encoder Neural Network Incorporating X-ray Fluorescence Fundamental Parameters with Machine Learning.” It was led by the author with assistance in conceptualization, reviewing, and editing by co-author David Poole. An earlier version [67] appeared in the workshop on Perception as Generative Reasoning at the Neural Information Processing Systems conference in 2019.

- Chapter 4: Unpublished work.

- Chapter 5: This work was published [68] in the Chemometrics and Intelligent Laboratory Systems journal, titled “Automatic Neural Network Hyperparameter Optimization for Extrapolation: Lessons Learned from Visible and Near-Infrared Spectroscopy of Mango Fruit.” It was led by the author with assistance in conceptualization, reviewing, and editing by co-author David Poole.

- Chapter 6: This work was published [70] in the Chemometrics and Intelligent Laboratory Systems journal, titled “Spectral Sensor Fusion for Prediction of Li and Zr in Rocks: Neural Network and PLS Methods.” It was led by the author; co-author David Turner arranged sample and data collection, provided feedback, and wrote portions pertaining to geological interpretation; and David Poole assisted in conceptualization, reviewing, and editing.
Additionally, anonymous reviewers provided feedback on all the published works (two reviewers each) during the peer-review process. A significant portion of research and data collection showcased in Chapters 2, 3, 4, and 6 was performed at the facilities of MineSense Technologies Ltd. through a collaboration between academia and industry. The collaboration was supported by Mitacs, a Canadian non-profit organization fostering collaborations to help drive innovation, economic growth, and knowledge exchange between academia and industry in Canada.
Table of Contents

Abstract ............................................................... iii
Lay Summary ......................................................... iv
Preface ................................................................. v
Table of Contents .................................................... vii
List of Tables ........................................................ xii
List of Figures ........................................................ xiii
List of Abbreviations ................................................ xiv

1 Introduction ......................................................... 1
  1.1 Spectroscopy Methods ........................................ 4
     1.1.1 X-ray fluorescence (XRF) .............................. 4
     1.1.2 Visible to short-wave infrared (Vis-NIR-SWIR) .... 5
     1.1.3 Laser-induced breakdown spectroscopy (LIBS) ...... 6
  1.2 Datasets ......................................................... 6
     1.2.1 Rock Dataset ........................................... 7
     1.2.2 Mango Dataset ........................................ 9
  1.3 Models .......................................................... 10
     1.3.1 Model-based Fitting ................................... 10
     1.3.2 Learning Models ....................................... 11
     1.3.3 Predict-the-Average .................................. 15
  1.4 Model Evaluation and Model Selection ....................... 15

2 Model-Based Fitting in XRF ..................................... 17
  2.1 Introduction .................................................. 17
2.2 Domain Knowledge in XRF ........................................... 18
2.3 XRF Simulator ......................................................... 20
2.4 Proposed Method (FitSim) ............................................. 21
2.5 Conclusion ............................................................... 24

3 Auto-Encoder using Simulator .......................................... 25
  3.1 Introduction ............................................................ 25
  3.2 Related Work: Integrating Domain Knowledge in Machine Learning 27
  3.3 Dataset ................................................................. 27
  3.4 Models ................................................................. 28
     3.4.1 OLS ..................................................................... 28
     3.4.2 LASSO .................................................................. 28
     3.4.3 FCNN .................................................................. 28
     3.4.4 CNN .................................................................... 29
     3.4.5 Analysis-by-Synthesis ............................................... 31
  3.5 Results ................................................................. 33
  3.6 Discussion ................................................................. 34
  3.7 Conclusion ................................................................. 36

4 Auto-Encoder using Spectra Decoder ................................... 38
  4.1 Introduction ............................................................... 38
  4.2 Background of Semi-Supervised Learning and Auto-Encoders ... 39
  4.3 Models ................................................................. 40
     4.3.1 LASSO ................................................................. 40
     4.3.2 PLS .................................................................... 40
     4.3.3 FitSim ................................................................. 40
     4.3.4 CNN-1 ............................................................... 40
     4.3.5 CNN-2 ............................................................... 41
     4.3.6 SpectraAE ........................................................... 41
  4.4 Dataset ................................................................. 44
  4.5 Experiment Setup ......................................................... 45
     4.5.1 Neural Network Training ........................................... 45
     4.5.2 Data Partitioning ..................................................... 46
     4.5.3 Hyperparameter Optimization (HPO) ......................... 46
     4.5.4 Evaluation Phase and Statistical Testing ...................... 47
  4.6 Results and Discussion .................................................. 51
     4.6.1 Model Comparison ................................................... 51
     4.6.2 Comparison to AutoSim ........................................... 55
     4.6.3 Spectrum Reconstruction .......................................... 55
     4.6.4 Other Types of Spectroscopy ..................................... 57
## 4.7 Conclusion

60

## 5 Hyperparameter Optimization

5.1 Introduction

5.2 Materials and Methods

5.2.1 Dataset

5.2.2 Baseline Neural Network

5.2.3 Repeatability

5.2.4 Hyperparameter Search Space

5.2.5 Automatic Hyperparameter Optimization

5.2.6 Choice of Validation Set

5.2.7 Neural Network Ensemble Averaging

5.2.8 Experiment

5.3 Results and Discussion

5.3.1 CNNB Variance

5.3.2 Singles-HPO Evaluations Without Ensembling

5.3.3 CNNB Evaluation With Ensembling

5.3.4 Singles-HPO Evaluations With Ensembling

5.3.5 Ensembles-HPO Evaluations

5.3.6 Generalizability of Validation Set

5.3.7 Baseline Compared to HPO Configurations

5.3.8 Lower Bound

5.4 Conclusion

82

## 6 Sensor Fusion

6.1 Introduction

6.2 Materials and Methods

6.2.1 Data and Experiment Setup

6.2.2 PLS-based Models

6.2.3 Neural Network Models

6.2.4 Compute Hardware and Software

6.3 Results and Discussion

6.3.1 Neural Network HPO

6.3.2 Interpreting Behaviour of High-level Sensor Fusion

6.3.3 Modelling Considerations and Limitations

6.4 Conclusion

104

## 7 Conclusion

7.1 Conclusion

7.2 Future Work
List of Tables

Table 3.1 Elements where each model achieved the best MSE . . . . . . . 34
Table 4.1 Paired $t$-test results for Li and Zr. . . . . . . . . . . . . . . . 52
Table 5.1 Hyperparameters’ search space. . . . . . . . . . . . . . . . . . . 65
Table 5.2 Partitioning choices and number of data points in each set. . . . 67
Table 5.3 Average compute times. . . . . . . . . . . . . . . . . . . . . . . 71
Table 5.4 The best hyperparameter values found in each HPO experiment. 77
Table 6.1 Frozen hyperparameters for NN2 as a result of NN1 HPO. . . . 97
Table 6.2 Resulting hyperparameter configurations after HPO. . . . . . . 97
Table 6.3 NNLS model weights. . . . . . . . . . . . . . . . . . . . . . . 101
Table A.1 Results of cross-validation. . . . . . . . . . . . . . . . . . . . . 148
Table B.1 Scores for all 48 elements. . . . . . . . . . . . . . . . . . . . . 151
Table B.2 Paired $t$-test results part 1/8. . . . . . . . . . . . . . . . . . . . 152
Table B.3 Paired $t$-test results part 2/8. . . . . . . . . . . . . . . . . . . . 153
Table B.4 Paired $t$-test results part 3/8. . . . . . . . . . . . . . . . . . . . 154
Table B.5 Paired $t$-test results part 4/8. . . . . . . . . . . . . . . . . . . . 155
Table B.6 Paired $t$-test results part 5/8. . . . . . . . . . . . . . . . . . . . 156
Table B.7 Paired $t$-test results part 6/8. . . . . . . . . . . . . . . . . . . . 157
Table B.8 Paired $t$-test results part 7/8. . . . . . . . . . . . . . . . . . . . 158
Table B.9 Paired $t$-test results part 8/8. . . . . . . . . . . . . . . . . . . . 159
Table B.10 List of hyperparameters and candidate values. . . . . . . . . 161
List of Figures

Figure 1.1 Example drill core source for samples. ............................. 7
Figure 1.2 Three example spectra. ............................................. 8

Figure 2.1 Architecture of the XRF simulator. ............................... 22

Figure 3.1 Architecture of AutoSim. ............................................. 30
Figure 3.2 Lithium results. ...................................................... 34
Figure 3.3 Aluminum results. .................................................. 34
Figure 3.4 Example spectra and AutoSim reconstructions. ............... 35

Figure 4.1 Neural network architecture of CNN-1 .......................... 41
Figure 4.2 Neural network architecture of CNN-2. ......................... 42
Figure 4.3 SpectraAE’s decoder architecture. .............................. 43
Figure 4.4 XRF spectrum example. .......................................... 44
Figure 4.5 Dataset partitioning. ................................................. 47
Figure 4.6 Performance evaluation procedure. .............................. 48
Figure 4.7 Example of matched pairs data. ................................ 49
Figure 4.8 Test-set results of Li and Zr. ..................................... 53
Figure 4.9 RMSE boxplots. ..................................................... 54
Figure 4.10 Comparison of AutoSim to SpectraAE. ....................... 56
Figure 4.11 Example reconstructions from SpectraAE in red and observed XRF spectra in blue. The x-axis is the channel index. .... 58

Figure 5.1 Neural network architecture and hyperparameter search space. 63
Figure 5.2 Experiment overview in pseudo-code. ......................... 66
Figure 5.3 Calibration, validation, and test sets. ........................... 68
Figure 5.4 RMSE boxplot. ..................................................... 72
Figure 5.5 CNN_B test-set RMSE as a function of ensemble size. .... 74
Figure 5.6 Test-set RMSE versus validation-set RMSE. .................. 79
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Sensor fusion levels.</td>
<td>83</td>
</tr>
<tr>
<td>6.2</td>
<td>Data splitting procedure.</td>
<td>85</td>
</tr>
<tr>
<td>6.3</td>
<td>Pseudo-code overview of experiments.</td>
<td>86</td>
</tr>
<tr>
<td>6.4</td>
<td>DUPLEX algorithm.</td>
<td>87</td>
</tr>
<tr>
<td>6.5</td>
<td>NN1 single-sensor model.</td>
<td>91</td>
</tr>
<tr>
<td>6.6</td>
<td>NN2 architecture.</td>
<td>93</td>
</tr>
<tr>
<td>6.7</td>
<td>NN3 architecture.</td>
<td>94</td>
</tr>
<tr>
<td>6.8</td>
<td>Test-set results of Li and Zr prediction tasks.</td>
<td>99</td>
</tr>
<tr>
<td>7.1</td>
<td>Gamut of models.</td>
<td>104</td>
</tr>
<tr>
<td>7.2</td>
<td>Multi-sensor auto-encoder reconstructions.</td>
<td>109</td>
</tr>
<tr>
<td>A.1</td>
<td>Results of remaining elements.</td>
<td>149</td>
</tr>
<tr>
<td>C.1</td>
<td>Validation RMSE as a function of ensemble size.</td>
<td>163</td>
</tr>
</tbody>
</table>
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>AE</td>
<td>auto-encoder</td>
</tr>
<tr>
<td>Ag</td>
<td>silver</td>
</tr>
<tr>
<td>AI</td>
<td>artificial intelligence</td>
</tr>
<tr>
<td>Be</td>
<td>beryllium</td>
</tr>
<tr>
<td>CNN</td>
<td>convolutional neural network</td>
</tr>
<tr>
<td>Cu</td>
<td>copper</td>
</tr>
<tr>
<td>CV</td>
<td>cross-validation</td>
</tr>
<tr>
<td>DM</td>
<td>dry matter (is a measure of carbohydrates which is an indicator of quality in mango fruit)</td>
</tr>
<tr>
<td>EDXRF</td>
<td>energy-dispersive X-ray fluorescence</td>
</tr>
<tr>
<td>ELU</td>
<td>exponential linear unit</td>
</tr>
<tr>
<td>FC</td>
<td>fully-connected layer (of a neural network, also known as a dense layer)</td>
</tr>
<tr>
<td>Fe</td>
<td>iron</td>
</tr>
<tr>
<td>FWHM</td>
<td>full width at half maximum (describes the width of a peak in a spectrum)</td>
</tr>
<tr>
<td>Ga</td>
<td>gallium</td>
</tr>
<tr>
<td>HPO</td>
<td>hyperparameter optimization</td>
</tr>
<tr>
<td>HS</td>
<td>hyperspectral</td>
</tr>
<tr>
<td>ICP-MS</td>
<td>inductively coupled plasma mass spectrometry</td>
</tr>
</tbody>
</table>
keV  kiloelectron volts
kV  kilovolts
LASSO  least absolute shrinkage and selection operator
Li  lithium
LIBS  laser-induced breakdown spectroscopy
LM  Levenberg–Marquardt algorithm
LR  learning rate
MAE  mean absolute error
MSE  mean squared error
NIR  near-infrared range of the light spectrum
nm  nanometer
NN  neural network
OLS  ordinary least squares
PCA  principal component analysis
PLS  partial least squares
ppm  parts per million
PtA  predict-the-average
ReLU  rectified linear unit
RMSE  root mean squared error
SNV  standard normal variate
SVM  support vector machine
Vis-NIR-SWIR  visible to short-wave infrared range of the light spectrum
Vis-NIR  visible to near-infrared range of the light spectrum
SWIR  short-wave infrared range of the light spectrum
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDXRF</td>
<td>wavelength-dispersive X-ray fluorescence</td>
</tr>
<tr>
<td>XRD</td>
<td>X-ray diffraction</td>
</tr>
<tr>
<td>XRF</td>
<td>X-ray fluorescence</td>
</tr>
<tr>
<td>Zr</td>
<td>zirconium</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Spectroscopy deals with electromagnetic radiation that is absorbed, reflected, emitted, or scattered by a material. Spectroscopy is useful in diagnosing medical conditions such as COVID-19 [253], nutritional deficiencies [204], and cancer [77, 91, 157]. Spectroscopy is also used to find signs that Mars may have had water [182], analyze bones for forensic investigations [41, 146], assess levels of SARS-CoV-2 (COVID-19) in wastewater [262], test food and water quality [10, 82, 87, 146, 164], and many other uses [3, 41, 144, 146, 146, 180].

A spectrum is a distribution of electromagnetic radiation as a function of channel. A channel in a spectrum represents energy, wavelength, or frequency, depending on the type of spectrometry. Spectroscopic data analysis, sometimes called spectral analysis, aims to interpret and extract meaningful information from spectra.

Spectral analysis varies in difficulty depending on the spectroscopic technique. Visible to short-wave infrared (Vis-NIR-SWIR), for example, is difficult to deconvolve because the underlying physical process that produces the spectrum is not well understood:

“In order to characterize absorption features, it is necessary to understand the physics that control the absorption process. However, absorptions in geologic surfaces containing intimately mixed mineralogies are complex and only partially understood.” [223]

X-ray fluorescence (XRF), on the other hand, is very well understood [21, 95, 101, 169], but the difficulty lies in identifying peaks and ascribing the area under those peaks to one chemical element or another. While each element produces characteristic peaks at known energies [65, 95, 131], the exact identification of elements can be challenging [80, 101] due to potential ambiguities caused by overlapping peaks,
measurement noise, and missing information for elements that XRF cannot detect (such as light elements). Many models for XRF require specification of instrument and environment properties. This thesis considers XRF applications where instrument and environment properties are unavailable or they vary. For example, in sensors out in the field, such as on a mining shovel or conveyor belt [16], rocks are constantly moving (leading to varying angles of incidence and distances). There may also be other factors not accounted for (such as dust or rain) which are difficult to model.

The difficulty of spectral analysis, and, more generally, the problem of extracting chemically relevant information from data, led to the interdisciplinary field of chemometrics\textsuperscript{1} [248] at the intersection of statistics and analytical chemistry. Recently, machine learning has also been involved in chemometrics. Machine learning has found its way into many fields as a tool for converting raw data into meaningful predictions, and spectral analysis is well-suited to benefit.

The most prevalent form of machine learning [140], supervised learning, requires labelled data. Supervised learning models are trained to map input data to output labels [93]. Labels are the ground truth, which serves as a trusted reference point by which the model is trained and tested. For example, a model may be trained to infer chemical concentrations from spectra. The goal is for the model’s predictions to be as close as possible to the ground truth when presented with new samples. A dataset containing data (such as spectra) and ground truth labels is referred to as labelled data.

Labelled data is often limited, because, while obtaining spectra is typically cheap, obtaining the corresponding ground truth is often expensive or time-consuming. Limited labelled data is a common occurrence in practical applications and is likely to continue being scarce. One approach to alleviate the demand for labelled data is by incorporating domain knowledge in a prediction model. Domain knowledge—provided by domain experts—refers to the expertise of a specific subject area, such as the characteristics, nuances, and intricacies of the domain or industry.

Research in machine learning has gone back and forth between research that focuses primarily on incorporating domain knowledge and research that focuses on domain-independent learning, utilizing as little human knowledge as possible by learning from data [229, 254]. Enormous amounts of data are required in many ma-

\textsuperscript{1}The term chemometrics was introduced circa 1970 [248].
chine learning applications [98, 221]. Small data applications are, nevertheless, common (e.g., [151, 177, 246, 251, 255, 257, 258, 263]). Various strategies to overcome the challenge of limited data are explored in this thesis, which span the gamut of models from those relying primarily on domain knowledge to those relying primarily on learning from data, including models that blend both together. Blending data with domain knowledge is an active area [27] of research (e.g., theory-guided data science [124] and physics-informed machine learning [158]). In a NIR News article as of 2022, incorporating domain knowledge in machine learning for near-infrared (NIR) spectroscopy was deemed an open area of research [186].

First, a fully-differentiable XRF simulator is developed leveraging domain knowledge, where “fully-differentiable” indicates that every operation in the simulator is capable of being differentiated. Differentiation is a requirement for training machine learning models using the backpropagation algorithm. The simulator is used in two models. In the first model (Chapter 2), simulator parameters are tuned to match a simulated spectrum to an observed spectrum. The resulting parameter values are then calibrated, through partial least squares (PLS) regression, to estimate element concentrations. This approach requires little ground truth data but is difficult to configure. In the second model (Chapter 3), the simulator is embedded in an auto-encoder (AE) neural network architecture that learns the inverse function of the simulator. A third model (Chapter 4) is developed that does not rely on specific domain knowledge about the underlying physics that generated the spectrum. This model is an AE which is applicable to any type of spectrum that contains peak-shaped structure. A benefit of both AE models [136] is that they can utilize unlabelled data in semi-supervised learning to lower the requirements for ground truth data.

Machine learning models often have parameter settings that need to be configured before the learning process begins. These parameters, which are not learned during training the model, are called hyperparameters. Neural networks require extensive hyperparameter optimization (HPO) which depends on the validation set providing an accurate estimate. HPO fails to locate the best configuration when the validation set score is noisy; noisy validation scores are typical of small datasets. A framework for HPO is developed (in Chapter 5) using ensembling to lower the variance.

A final prediction model (Chapter 6) is developed to combine information from multiple spectrometers, which is particularly challenging for small datasets. Sev-

---

2 Roughly speaking, it is recommended to have hundreds of thousands or more. According to Jeff Dean (2017), Google Senior Fellow at Google Brain, “if you only have 10 examples of something, it’s going to be hard to make deep learning work. If you have 100,000 things you care about, records or whatever, that’s the kind of scale where you should really start thinking about these kinds of techniques.” (https://goo.gl/AWfsrK)
eral sensor fusion methods are compared, including a parallel-input convolutional neural network (CNN) that utilizes the ensembling HPO framework.

## 1.1 Spectroscopy Methods

Spectra from three types of spectroscopy are utilized in this thesis: XRF (Chapters 2, 3, 4, and 6), Vis-NIR-SWIR (Chapters 5 and 6), and laser-induced breakdown spectroscopy (LIBS) (Chapter 6).

### 1.1.1 X-ray fluorescence (XRF)

There are two types of XRF: energy-dispersive X-ray fluorescence (EDXRF) and wavelength-dispersive X-ray fluorescence (WDXRF). In this thesis, all references to XRF refer to the energy-dispersive type of XRF. XRF is a non-destructive analytical technique used to determine the elemental composition of a sample [21, 95, 103, 169].

An XRF spectrometer emits X-rays. When X-rays irradiate a sample, atoms absorb the energy and eject electrons, creating vacancies. These vacancies cause electrons from outer shells to transfer to inner shells and, consequently, characteristic X-rays (which are photons) are emitted with an energy equal to the difference between the two binding energies of the corresponding shells. The spectrometer tracks the number of photons and their energies and builds a histogram of energy versus photon counts (see Figure 1.2).

Performing spectral analysis of XRF spectra presents challenges due to several factors, including:

- An XRF sensor emits X-rays of a particular energy, depending on the sensor, which can only excite a subset of elements in the periodic table. The specific elements that can be detected depend on the energy levels involved in the transition of electrons from outer shells to inner shells in an atom. For example, in iron (Fe), 6.403 keV is required for the K-L$_3$($K\alpha$) transition which means that source X-rays at 6.403 keV or higher are capable of causing this transition. The higher the element is on the periodic table (by atomic number), the more energy is required.

- The greater the distance from sensor to sample, the more that elements lower on the periodic table fluoresce too weakly to detect.

- Characteristic peaks often overlap, and their intensities vary depending on limited detector resolution [21], distance, composition, measurement noise [80], and other factors.
• Sample composition non-linearly affects the element intensities in an XRF spectrum [102].

• Matrix effects, inelastic and elastic scattering, thermal effects, Bremsstrahlung radiation, and attenuation through air are some of the other effects that must be considered when analyzing XRF spectra [8, 169, 211].

In addition to mining and geology, as used in this thesis, XRF analysis has applications in planetary exploration (e.g., of Mars [182]), health care [63], archaeology [18, 40], cultural heritage [200], geochemistry [144], and more [184, 239].

1.1.2 Visible to short-wave infrared (Vis-NIR-SWIR)

In reflectance spectroscopy [100, 128, 223], a light source with a broad range of wavelengths illuminates a sample and the amount of reflected light is measured. When electromagnetic energy (light) meets the surface of a sample, a combination of specular reflection, transmission, absorption, diffraction, and multiple scattering from adjacent particles occurs [125]. The reflected light is then analyzed to determine how much of each wavelength is absorbed or reflected by the sample. The resulting spectrum exhibits absorption features specific to the sample’s composition and crystal structure [100], which provides mineralogical information (in the case of rocks). The underlying physics governing the absorption process is not well understood in geological samples due to complex mixtures of minerals [223].

In terms of spectral analysis, the spectrum is composed of a continuum with absorptions (upside-down peaks) superimposed. Continuum refers to the underlying, smooth, and featureless background signal in a spectrum. This background signal arises from various sources, such as the environment and the instrument, and other factors that contribute to the overall intensity of the measured reflectance.

The continuum is commonly removed from a spectrum prior to analysis (e.g., [54, 56]). Since the underlying process is not well understood [223], the continuum may contain information about the composition of the sample. The implication of this is that continuum should not be removed in pre-processing, or, if it is removed, both artifacts (i.e., the spectrum of absorptions and the spectrum of only continuum) should be included equally in the modelling process. In this thesis, the continuum is not removed from the spectra.

Reflectance spectroscopy is useful for assessing mineralogy in rocks. Reflectance spectroscopy can distinguish allotropes (different structural forms) based on their distinct optical properties and spectral fingerprints. For example, it can distinguish diamonds from the graphite in a pencil, both of which are made of pure carbon; the peaks in an XRF spectrum, on the other hand, would only detect carbon, not its form.
Reflectance spectroscopy is frequently used in remote sensing of Earth [34, 125] and has been used to remotely study planets [100], asteroids [222], and the moon [179]. It has also been used to analyze nutrients in drinking water, wastewater, and river water [87]. Food quality is another important application, where reflectance spectroscopy is used to assess quality attributes of, for example, blueberries [82], pears [164], and mangoes [10]. Other applications include assessment of crop conditions, inspection of forestry operations, and wildfire detection, among others [3].

1.1.3 Laser-induced breakdown spectroscopy (LIBS)

LIBS is a technique that uses a high-energy laser pulse to create a plasma on the sample surface. The plasma emits light that can be analyzed by a spectrometer. Due to laser ablation, a small portion of the material is destroyed (through evaporation), but LIBS is considered non-destructive in geology because the spot size is around 0.2 mm in diameter, which is negligible compared to the mass of a whole rock.

LIBS has some unique benefits, such as the capability to detect not just details concerning the concentrations of elements, but also some isotopic ratios and some allotropes. For this reason, it is useful both for determining elemental concentrations like XRF and mineral concentrations (like Vis-NIR-SWIR spectroscopy). LIBS is also able to detect elements lower on the periodic table than XRF [209]. Issues with LIBS include ambiguities due to matrix effects, and sensitivity to moisture which may require samples to be dried before analysis [209].

Like other spectroscopy methods, LIBS is widely applicable to many industries [59, 146, 180]. LIBS has been used on Mars, on rovers [58, 81], to identify the composition of rock samples. It can be used to sort scrap metal, including computer and phone parts [180]. Biomedical applications include the detection of nutritional deficiencies [204] (in specimens of skin tissue, fingernails, and teeth) and early detection of cancer [77, 91, 157]. In cultural heritage and archaeology, LIBS has been used to analyze pottery, copper-age weapons, bronze statues, marble, bones and teeth of Homo sapiens, Roman coins, and more [41]. Applied to paintings, LIBS can reveal overpainting and retouching beneath layers of paint [41]. LIBS may be used in real-time coal analysis for improvement in utilization efficiency and pollution reduction [209]. It can also be used to stop terrorism through the detection of explosives and other dangerous chemicals [59].

1.2 Datasets

Two datasets are used in this thesis: a rock dataset provided by MineSense Technologies Ltd. and a mangoes dataset by Anderson et al [10, 11].
dataset is used in Chapter 5. The rock dataset is used in the other chapters in this thesis.

1.2.1 Rock Dataset

The rock dataset consists of 491 rock core samples, each about 8 cm in length, obtained from a lithium mineral exploration project. These samples are representative of unprocessed rocks encountered in mining operations. Figure 1.1 shows a few of these samples. Three types of spectroscopy were used to analyze each sample: X-ray fluorescence (XRF), visible to short-wave infrared (Vis-NIR-SWIR), and laser-induced breakdown spectroscopy (LIBS). Each spectrometer was used to scan each rock sample in four orientations, and the resulting spectra were averaged together. Examples are shown in Figure 1.2.

After obtaining spectral measurements, 177 samples were destructively analyzed using geochemical assays to determine ground truth. A third-party assay lab was paid to perform geochemical analysis. Resource limitations constrained the number of samples that could be studied. Not only is collecting sensor data and obtaining ground truth time-consuming and expensive, but the assay also destroys the rock in the process, eliminating any possibility of further inquiry or studies.

![Figure 1.1: Example drill core source for samples.](image)
Figure 1.2: One example spectrum from each spectrometer: X-ray fluorescence (XRF), visible to short-wave infrared (Vis-NIR-SWIR), and laser-induced breakdown spectroscopy (LIBS).

X-ray fluorescence (XRF)

The XRF instrument utilized is an EDXRF spectrometer with a 50 kV X-ray tube with a silver (Ag) target and a detector that has a beryllium (Be) window. It generates a spectrum with 1024 channels from 2 keV to 33 keV (kiloelectron volts). The spot size on the sample is around 2 cm in diameter. The unit is automatically moved to be approximately 10 cm away from the rock sample using a laser distance sensor.
**Visible to short-wave infrared (Vis-NIR-SWIR)**

A portable Vis-NIR-SWIR spectrometer was used to collect high-resolution spectral reflectance data in the range of 350 to 2500 nm with 1 nm intervals (2151 spectral channels). The instrument is an ASD TerraSpec Halo model from Malvern Panalytical that carried out auto-calibration. The technique is a contact method and the analysis spot size is approximately 2 to 3 cm in diameter.

**Laser-induced breakdown spectroscopy (LIBS)**

The LIBS instrument is a FiberLIBS model from Secopta (Teltow, Germany) equipped with a 1064 nm Q-switched Nd:YAG laser that has a pulse duration of 1.5 ns, an energy output of 3 mJ, a maximum pulse rate of 100 Hz, and a spot size of around 0.2 mm. The spectrometer is a Czerny–Turner type with a CCD detector providing 2048 channels from 229.21 to 499.58 nm with a full width at half maximum (FWHM) of 0.20 to 0.5 nm. For each of the four rock sample orientations, 100 laser pulses are produced, resulting in a total of 400 spectra that are averaged to produce one spectrum per rock sample.

**Geochemical Analysis**

Ground truth element concentrations—which are used as the labels in prediction modelling—are obtained by destructive geochemical assay providing gold-standard composition estimates\(^3\). ALS, a commercial assay laboratory, carried out the analysis using their proprietary ME-MS61 and ME-XRF10 procedures \(^7\). The analysis uses 4-acid digestion followed by inductively coupled plasma mass spectrometry (ICP-MS). For zirconium (Zr), 38 samples were above 500 ppm (which is the limit for ME-MS61); these overlimit samples were subsequently analyzed by ALS's ME-XRF10 procedure. Concentration values that are below the limit of detection are replaced with 0. The samples were also accompanied by qualitative descriptions of the rock type, provided by a geologist during the drilling campaign.

### 1.2.2 Mango Dataset

The dataset of mangoes \(9–11, 160\) consists of 11,362 Vis-NIR-SWIR spectra from 4675 mangoes. Mangoes spanned four seasons, encompassing ten plant varieties and two growing regions. Data from the fourth season were set aside as an independent test set. Short-wave infrared (SWIR) spectroscopy has demonstrated

\(^3\)The elements tested for include Ag, Al, As, Ba, Be, Bi, Ca, Cd, Ce, Co, Cr, Cs, Cu, Fe, Ga, Ge, Hf, In, K, La, Li, Mg, Mn, Mo, Na, Nb, Ni, P, Pb, Rb, Re, S, Sb, Sc, Se, Sn, Sr, Ta, Te, Th, Ti, Tl, U, V, W, Y, Zn, and Zr.
effectiveness in the non-invasive assessment of dry matter (DM) content (% fresh weight) in mango fruit.

Non-invasive fruit assessment collected Vis-NIR-SWIR spectra using a F750 Produce Quality Meter (Felix Instruments, Camas, USA). The spectral bands range 300 – 1100 nm with approximately 3.3 nm pixel resolution and 10 nm optical resolution. Four scans were collected per sample, and the resulting spectra were averaged together.

Spectra collection and destructive reference analysis (to obtain DM %) were conducted within a single day. DM serves as the ground truth label in prediction modelling. DM % is an index of total carbohydrates (including starch and sugars) which is used as an indicator of quality and harvest maturity. In mangoes, higher dry matter percentages are generally associated with better taste and sweetness.

Section 5.2.1—where this dataset is used—provides further details.

1.3 Models

Models used for spectroscopic data analysis typically fall into two broad categories: physics-based and data-driven models. Data-driven models utilize observed data without explicitly modelling the underlying mechanisms, often leveraging machine learning and statistical techniques, to uncover complex patterns within spectral data. Physics-based models in spectroscopy utilize fundamental principles that govern the interaction of light with matter, employing well-established physical laws and mathematical equations. Models from both categories are used in this thesis, for which a brief background is provided.

1.3.1 Model-based Fitting

In spectroscopic data analysis, what is called model-based fitting is a physics-based approach where parameters of a forward model, such as element amounts, are tuned such that the difference between the observed spectrum and the spectrum simulated by the model is minimized. The model-fitting optimization problem is typically framed as a non-linear least-squares problem [32] and the standard algorithm to solve it is the Levenberg–Marquardt (LM) algorithm [154].

The pioneering work by Rietveld [202] describes the Rietveld refinement method for deconvolving X-ray diffraction (XRD) spectra using model-based fitting. In Rietveld refinement, peaks are modelled as Gaussian functions whose parameters (halfwidth, position, etc.), along with other model parameters, are optimized to fit the XRD spectrum. TOPAS is a recent computer program that includes Rietveld refinement [57].

A model-based fitting approach was used in XRF spectroscopy in the AXIL
program [80, 238, 241] and later in PyMCA [216] and GIMPy [46]. Other applications for model-based fitting include XRD [5, 53, 156, 205], NIR [47, 123, 179, 223], and a fused XRD and XRF model [39].

1.3.2 Learning Models

Multivariate calibration relates multi-wavelength or multi-energy spectral responses to concentrations [120]. Several methods for multivariate calibration have been developed, such as partial least squares (PLS) [247]. Recently, methods from artificial intelligence (AI), machine learning, and statistical learning have also proven useful [224, 251]. In particular, neural networks have risen in popularity as a method of multivariate calibration. Multiple spectroscopy techniques have successfully utilized convolutional neural networks (CNNs) for classification [2, 52, 83, 97, 143, 148] and regression tasks [60, 151, 258, 263]. For XRF, fully-connected neural networks whose inputs are concentrations or peak intensities of the desired elements have been investigated (e.g., [108, 145]). However, these don’t operate on full-spectrum XRF directly. While convolutional neural networks (CNNs) have been used in XRF imaging applications (e.g., [61, 226]), to our knowledge, the application of recent advancements in neural networks to XRF spectral analysis for modelling sample composition has not been investigated in detail at the time of writing.

Most models have hyperparameters, which are settings that are not learned during training the model. For example, in principal component analysis (PCA) and PLS, the number of components must be chosen. In neural networks, there are typically many hyperparameters. Neural networks are configured by choosing an architecture (such as number of layers) and hyperparameter values (such as learning rate). In this thesis, architecture settings are considered in the same manner as hyperparameters, and any specific architecture and assignment of all the hyperparameters is referred to as a hyperparameter configuration.

LASSO

Least absolute shrinkage and selection operator (LASSO) is a least squares linear regression model that optimizes squared error plus a penalty term, which is the sum of the absolute value of the model’s coefficients ($L_1$-norm on the model parameters) [231] times a constant. The penalty terms make the model more likely to set coefficients to zero, which induces sparsity. The objective function in LASSO is:
argmin_{\beta} \sum_{i=1}^{n} \left( y_i - \sum_{j=1}^{d} X_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^{d} |\beta_j| 

\text{(1.1)}

where \( n \) is the number of examples, \( d \) is the number of features, \( \beta \) is a weight vector (model coefficients) of length \( d \), \( y_i \) is the ground truth of the \( i^{th} \) example, \( X \) is an \( n \times d \) matrix of input data, and \( \lambda \) is a hyperparameter that determines the amount of \( L_1 \) regularization (the strength of the penalty term).

This model has been used in spectroscopy [42, 181]. Since spectra often contain a large number of features (which are channels), feature selection is a common task in spectroscopic analysis. LASSO is used to combine feature selection with regression in one model, effectively reducing the number of channels (e.g., wavelengths) participating in the model. In this thesis, LASSO models utilize the software framework provided by Scikit-Learn [189].

**PLS**

Partial Least Squares (PLS) [92, 243, 247, 249] is a statistical method employed for modelling relationships between sets of variables, especially in situations where there is a large number of predictors (input variables) and possibly a smaller number of observations. PLS is designed to handle multicollinearity, which is when many input variables are correlated to each other. This makes PLS a natural choice for spectroscopy applications (e.g., it has been applied to LIBS [42, 72] and vibrational spectroscopy [1]).

The basic idea behind PLS is to extract latent variables, also known as components, from both the predictor variables, \( X \), and the response variables, \( Y \), by finding linear combinations of the original variables. PLS decomposes both the \( X \) and \( Y \) matrices into a series of orthogonal components. These components are identified iteratively, by sequentially choosing a component that maximizes the covariance between \( X \) and \( Y \), and then removing the effects of the component to identify subsequent components. The orthogonality ensures that the components are uncorrelated with each other. By emphasizing the covariance between \( X \) and \( Y \), PLS effectively separates signal from noise.

The method provides interpretable results through scores and loadings. Scores represent the projection of the original data onto the latent variables and can be interpreted as new coordinates in the subspace. Loadings are the weights (or coefficients) in a linear combination that effectively project the original data into the subspace defined by the latent variables.

PLS is related to ordinary least squares (OLS) and principal component regression (PCR) [176, 220]. Both PLS and PCR are designed to handle many correlated
input features. In PCR, one performs a principal component analysis (PCA) on the input data (design matrix), chooses $k$, the number of components to use, then performs regression as usual in the new $k$-dimensional space. In PLS, however, the idea is to decompose both the design matrix ($X$) and response matrix ($Y$) such that both $X$ and $Y$ participate in forming the projection. This means that PLS seeks to maximize variance in the projected space, as in PCA, but additionally biases the responses toward directions yielding accurate predictions [233].

In this thesis, PLS models use the PLS-1 implementation found in Scikit-Learn [189].

Artificial Neural Networks

Neural networks are universal function approximators [111, 171, 264], capable of learning and representing complex relationships between inputs and outputs, making them versatile tools across various domains. A neural network [93, 140] consists of a series of processing layers. Each layer takes the output of the previous layer as input, applies a function to it (the function depends on the type of layer), and outputs the result to the next layer(s). A network with many layers is considered a “deep neural network” [171]. Each layer may have a set of parameters, such as weights and biases, that must be learned from data through training. The training process of neural networks involves adjusting the network’s parameters to minimize a loss function (such as the difference between predicted and actual outputs). This is achieved through the use of backpropagation, an optimization algorithm that calculates gradients of the loss function with respect to the network parameters.

A layer includes a non-linear function (called an activation function [37]) before passing the output to the next layer. Common activation functions include tanh, sigmoid [159], ReLU [173], and ELU [55]. The activation function may be omitted at the output layer, depending on the prediction task.

In a spectroscopic analysis application, the input to the neural network may be a spectrum. The value from each channel of the spectrum is passed to the first layer. Two types of layers are used in this thesis: fully-connected layers and convolutional layers.

Fully-Connected Layers

A fully-connected (FC) layer, also known as a dense layer, is computed as a linear combination of the layer’s inputs. It is defined by one hyperparameter, the number of units. It is called “fully-connected” because every unit is connected to every input unit in a linear function; the output of each unit in the fully-connected layer
is computed by taking a weighted sum of its inputs (and 1, a bias term). Fully-connected layers are useful for spectra because they learn dependencies between features. For example, a fully-connected layer may learn the relationship between concentrations of elements within a mineral.

**Convolutional Layers**

Convolution operations were used in the analysis of XRF spectra before their use was popularized in neural networks [80]. The convolution of a spectrum with a filter was used to identify peaks, where the filter’s weights were hand-picked to match to the shape of the expected peak or the peak’s first or second derivative. The filter may be a doublet peak if those are expected. A difficulty was that the filter’s size had to match exactly with the size of the peaks in the spectrum [80].

A convolutional layer in a neural network [93] uses a filter where the weights in the filter are learned from data. CNNs were first developed for images, where each training example is a 2D matrix [138, 139]. Here, we define CNNs for 1D data. One-dimensional (1D) convolutional layers are commonly used for analyzing sequential data, including spectra [2, 6, 52, 60, 75, 83, 148, 151, 160, 208, 237, 251, 263]. The input is a vector. In a 1D convolutional layer, a filter (a vector of weights) is applied to the input in a sliding window fashion. For each position of the window, the weights of the filter are multiplied with the input elements underneath the window and summed (mathematically, a dot product is used), then the window is moved one or more units (called the “stride”) over and the process is repeated until the end of the input vector. Filter weights are learned through backpropagation during neural network training. Hyperparameters of convolutional layers include:

- Filter size: The number of weights in the filter.
- Number of filters: The number of unique filters to pass over the input.
- Stride: The step size the filter takes when sliding over the input. A larger stride reduces the dimensions of the output but may miss out on important details; in this thesis, the smallest stride possible (which is 1) is used.
- Padding: Whether or not to pad the input with values (usually zeroes) before and after the input vector. In this thesis, zero-padding is used which appends zeroes before and after the input such that the output is the same size as the input.
- Dilation: Spacing between values in the filter. For example, a dilation rate of 1 means no gaps (which is standard convolution, and is the setting used in
this thesis), and a dilation of 2 means that every second value in the input is seen by the filter.

A convolutional neural network (CNN) is any neural network that contains at least one convolutional layer. CNNs for spectra typically follow a similar pattern: one or more convolutional layers followed by one or more fully-connected layers, with an activation function after each layer. Convolution layers incorporate some general domain knowledge about spectral data. Specifically, a convolution layer captures the notion that features are likely to be locally correlated. Local correlation is exhibited by any spectrum with peak-shaped structures. This is true for XRF spectra, for instance, because neighbouring channels’ intensities are correlated within the background continuum, and within peaks which are caused by limited detector resolution [21].

1.3.3 Predict-the-Average
A simple straw-man prediction model is to use the average value of the target, which we call the predict-the-average (PtA) model. For example, on rocks, one may “predict” every new rock’s lithium (Li) concentration to be that of the average Li concentration. For a given element, if a model’s error is on par with PtA, then it is not extracting any useful signal from the sensor data.

1.4 Model Evaluation and Model Selection
Since many models are compared throughout this thesis, some background on model evaluation and model selection [104, 245] is provided. The goal of model evaluation is to assess how well a trained model will perform on new samples. A challenge in evaluation is overfitting [93], when a model begins to tailor its predictions to the specificities of the training dataset (like spurious correlations) that don’t generalize well to new samples. Overfitting occurs when a model essentially “memorizes” the training data rather than learning its underlying structure. Evaluating an overfitting model on the same data used for training produces overly optimistic performance claims. A better evaluation procedure is the holdout method [245], in which the dataset is divided into two sets: a training set and an independent test set. The score on the test set is a measure of the model’s generalization ability, or, how well it will perform on new samples.

Model selection, when several models are compared and the best one selected, is itself prone to overfitting. Overfitting in model selection occurs when a model trained on the training set and evaluated on the test set gets lucky—the model achieved a good test-set score purely by chance. The risk of overfitting is higher
when the test set is small [93, 245], which is common when labelled data is limited (as considered in this thesis). Model selection is conducted for two purposes:

1. Comparing model families: When different types of models are compared for the purpose of evaluating which type is best, such as a comparison between PLS and a particular neural network.

2. Hyperparameter optimization (HPO): When different instances of model family—models with different hyperparameter configurations—are compared for the purpose of picking the best hyperparameter values.

In any model selection scenario, the holdout method may be repeated with different randomly chosen subsets of the dataset. Since the probability of multiple trained models achieving good test-set scores purely by chance is small, the average test-set score will be a better indicator of the model’s generalization ability. Two widely adopted subsampling strategies are used in this thesis, both of which are forms of cross-validation (CV) [50, 219, 245]:

1. **k-fold cross-validation** [84] partitions the data into $k$ subsets. In each iteration of CV, one subset (the “validation set”) is used to estimate the model’s performance while the remaining $k - 1$ subsets (the “training set”) are used for training. By repeating this process $k$ times and calculating the performance across all folds, we obtain an estimate of how well the model will generalize to new data. This estimate tends to be slightly conservative [50] relative to a model trained on the entire dataset because it uses less data.

2. **Repeated k-fold cross-validation** [245] is $k$-fold cross-validation where the whole procedure is repeated a number of times with different random shuffles of the data. Ten repeats and ten folds, which we refer to as $10 \times 10$-fold CV, is a standard choice [43, 44, 240, 245] and is used in Chapter 4.
Chapter 2

Model-Based Fitting in XRF

From the gamut of models spanning reliance on domain knowledge to reliance on learning from data, this chapter focuses on leveraging expert domain knowledge. We interviewed domain experts in geology and physics from MineSense Technologies Ltd. and the University of British Columbia and asked for domain knowledge of the most important aspects of an XRF spectrum, most of which described how a spectrum is created. Since most of the domain knowledge described a forward model, a simulator of XRF spectra was designed. The XRF simulator (Section 2.3) was published as part of the publication [69] described in Chapter 3.

2.1 Introduction

Model-based fitting (described in 1.3.1) for XRF typically requires specification of both instrument and environment properties. For example, PyMCA [216] requires details of the spectral distribution of the X-ray tube’s beam, which, if not already known, involves “measuring a series of well-characterized samples covering most of the tube spectral emission range” [216]. We consider XRF applications where instrument and environment properties are unavailable or they vary from one XRF spectrometer to another. For instance, in field sensors like those on mining equipment [16], rocks are in constant motion, resulting in changing angles of incidence and distances. Factors like dust and weather further complicate accurate modelling.

In this thesis, an XRF simulator is designed from the domain knowledge provided which approximates the effects of the instrument and the environment. The simulator uses transition energies and probabilities of all elements and parameterized distributions to approximate other fundamental and instrument parameters. Simulator parameters are optimized using gradient descent; the simulator is programmed to support automatic differentiation, enabling the utilization of ADAM
Gradient descent algorithms explore a high-dimensional space given by the parameters which may contain many points with a near-zero gradient in some dimensions. Near-zero gradients are an issue for many first-order gradient-based optimizers [64], but is less [218] of an issue for ADAM [126] and other methods based on RMS-Prop (root mean squared propagation). We found multiple rounds of gradient descent fitting using ADAM performed best: we configured the first round to fit the strongest peaks in this dataset, which were iron (Fe) and copper (Cu) according to the experts. Then, additional rounds fit more elements or other simulator parameters. The resulting parameter values are then calibrated, through PLS regression, to estimate element concentrations for each rock sample. The overall method we call FitSim, which consists of two steps: (1) fitting XRF simulator parameters to data, and (2) using the parameter values to predict element concentrations.

Domain knowledge provided by experts is outlined first (Section 2.2) followed by implementation details of the XRF simulator (Section 2.3). Section 2.4 describes FitSim, a method for quantifying element concentrations by fitting the XRF simulator parameters to observed spectra. Experimental results are presented later, in Chapter 4, where FitSim is compared to other models.

### 2.2 Domain Knowledge in XRF

From interviews with domain experts, geology and physics experts provided details about XRF behaviour and the interaction of X-rays with rocks that affect the spectra, summarized here:

- $(K_1)$ Each element in the periodic table produces a set of “peaks” centered at known energies.$^1$

- $(K_2)$ The element peaks occur with known ratios, given by the probability of electrons fluorescing for each transition.

- $(K_3)$ Peaks are Lorentzian shaped, or a combination of Lorentzian and Gaussian.

- $(K_4)$ The spectrum from each element’s transition peaks are summed in the final spectrum.

- $(K_5)$ If two elements are present in a sample, their spectra are also added together.

$^1$These peak locations occur at specific transition energies. $K-L_3$ is the strongest one, but there can be up to 45 others. Some energies will be out of range of our sensor, and some will be too small to see.
Matrix effects may cause the amount of one element to increase or decrease the photon count of others. A matrix is the set of elements surrounding or hosting the element(s) of interest. When an element is irradiated by X-rays and fluoresces, the emitted energy may, for example, be re-absorbed by another element in the matrix (if that element is lower in the periodic table) which itself fluoresces and is subsequently detected as a response for that element in the observed spectrum.

20–21 keV contains Compton peaks (which are not related to elemental composition).

Raleigh peaks may occur at 21–24 keV due to elements in the construction of the sensor itself (e.g. rhodium, tungsten, molybdenum, or silver).

Peaks between 2–5 keV are due to thermal effects.

Attenuation will cause lower keV bands to lose photon counts (due to absorption into the air between sensor and target).

Detection efficiency will cause higher keV bands to lose photon counts.

Bragg scattering may occur: this is coherent scattering that occurs within the sample when photons interact with the crystalline structure of a sample.

Elements with higher energies of fluorescence will be more easily identified in smaller concentrations because penetration depth increases for higher keV.

Continuum is the broad and continuous distribution of X-rays that is present in addition to the characteristic X-ray lines emitted by the elements of interest. The continuum arises from various sources including "bremsstrahlung radiation" (or, backscatter in the spectrum) and outer electron shell transitions in the sample [78].

More specifics on the physics and established mathematical models may be found in the literature [78, 102, 131].

2Fun fact: the sky is blue because of Rayleigh scattering. Rayleigh occurs more in lower wavelengths, so blue light is scattered more than other colours (violet scatters too, but there's less of it).

3For more information, see http://www.xrf.guru/styled-12/styled-13/index.html
2.3 XRF Simulator

The domain knowledge provided describes an approximate forward model for producing a rock sample’s XRF spectrum which is used to design an XRF simulator. The paths of individual photons are not simulated, but rather the expected histogram of photon counts for each energy (bin) using known X-ray transition energies and associated probabilities. X-ray transition energies were downloaded from NIST’s X-Ray Transition Database\(^4\) and the probabilities of each transition are from the Evaluated Atomic Data Library\(^5\). All K, L1, L2, and L3 transition types are included in the model. We refer to the set of transitions for element \(e\) as \(T_e\) and the energy and probability for transition \(t \in T_e\) as \(t_i\) and \(t_p\) respectively.

Firstly, a spectrum is modelled as a function\(^6\) from energy, \(i\), to photon count (or intensity):

\[
\text{spectrum: } i \mapsto \text{photon count} \quad (2.1)
\]

The peak caused by a transition is modelled as a Lorentzian function\(^7\), \(L\), which outputs a spectrum:

\[
L(t_i, t_p, \Gamma) \mapsto \text{spectrum} \quad (2.2)
\]

\[
L(t_i, t_p, \Gamma)(i) = \frac{t_p \cdot (\frac{\Gamma}{2})^2}{(i - t_i)^2 + (\frac{\Gamma}{2})^2} \quad (2.3)
\]

where \(t_i\) is the location of the center of the Lorentzian peak (given by the energy of transition \(t\)), \(t_p\) is the height of the peak (given by the probability of transition \(t\)), and \(\Gamma\) is the width of the peak.

Peaks from all transitions are combined by summing up the spectra, channel-wise, produced by all the transitions of all the elements:

\[
\theta = (\theta_{\text{Cu}}, \theta_{\text{Fe}}, \text{etc}) \quad (2.4)
\]

\[
g: \theta \mapsto \text{spectrum} \quad (2.5)
\]

\[
g(\theta)(i) = \sum_{e \in E} \left( \sum_{t \in T_e} L(t_i, t_p, \Gamma)(i) \right) \quad (2.6)
\]

---


\(^5\)The 1997 release of the Evaluated Atomic Data Library (EADL97) is available from Nuclear Data Services of the International Atomic Energy Agency at https://www-nds.iaea.org/epdl97/libsa ll.htm.

\(^6\)The notation “name: \(a, b, c \mapsto x\)” denotes a function named “name” with 3 arguments \((a, b, c)\) that returns \(x\).

\(^7\)A Lorentzian has a similar shape to a Gaussian but is more narrow around the peak with longer tails. Another profile may also fit well, such a Gaussian or Voigt function.
where $\theta_e$ is the amount of scaling needed to scale $L$ into photon counts.

We assume the instrument and environment properties at deployment time are unknown ahead of time. Instead, they are approximated by models for the background continuum, attenuation, and efficiency. First, attenuation and efficiency curves are jointly represented by the product of two sigmoid curves, approximated by

$$S: a_1, c_1, a_2, c_2 \mapsto \text{spectrum} \quad (2.7)$$

$$S(a_1, c_1, a_2, c_2)(i) = \frac{1}{1 + e^{a_2(c_2-i)} + e^{a_1(c_1-i)}} \quad (2.8)$$

$S$ is used to calibrate for the specific instrument and environment and is parameterized by four parameters ($a_1$, $c_1$, $a_2$, $c_2$) which are shared by all spectra produced by the same experimental setup.

The last piece is the background continuum which is complex to characterize from theory and is typically approximated [80]. To tackle this, the Bézier curve, common in computer graphics, is adopted for its simplicity with few parameters. The background continuum is modelled as a Bézier curve, $b$, which is a function of two global parameters, $p_1$ and $p_2$, and $\alpha$ that depends on the rock sample:

$$b: p_1, p_2, \alpha \mapsto \text{spectrum} \quad (2.9)$$

$$b(p_1, p_2, \alpha)(i) = \alpha(3p_1i(1-i)^2 + 3p_2i^2(1-i)) \quad (2.10)$$

Finally, the complete spectrum for a sample is produced by generating $g$, scaling it by $S$, and adding the background continuum, $b$:

$$f: \theta, \alpha, \Gamma, a_1, c_1, a_2, c_2, p_1, p_2 \mapsto \text{spectrum} \quad (2.11)$$

$$f(\cdots)(i) = g(\theta)(i) \times S(a_1, c_2, a_2, c_2)(i) + b(p_1, p_2, \alpha)(i) \quad (2.12)$$

This function, $f$, constitutes the XRF simulator and is depicted graphically in Figure 2.1. Note that $t_i$ and $t_p$ are known from fundamental parameters, $\theta$ and $\alpha$ are specific to a rock sample, and $\Gamma$, $a_1$, $c_1$, $a_2$, $c_2$, $p_1$, and $p_2$ are global parameters. Rock-specific and global parameters must be tuned to data.

### 2.4 Proposed Method (FitSim)

Fitting model parameters is a common approach (see Section 1.3.1) to deconvolving a spectrum. The simulator was designed to be differentiable using the TensorFlow Python package allowing an optimizer, such as gradient descent, to update the parameters to minimize a loss function consisting of the sum of two terms:
Figure 2.1: Architecture of the XRF simulator; an approximate forward model of EDXRF spectra based on domain knowledge.

1. The error between the simulated spectrum and the observed spectrum:

\[ v_1 = [y_1, y_2, \ldots, y_n] \]  
\[ v_2 = [f(\beta)(i) | i \in I] \]  
\[ \mathcal{L}(\beta) = MSE(v_1, v_2) \]

where \( v_1 \) is the observed spectrum, \( n \) is the number of channels in the spectrum (1024 in the case of XRF), \( v_2 \) is the simulated spectrum, \( I \) is the set of energies, \( \beta \) is the set of simulator parameters (\( \theta, \alpha, \Gamma, a_1, c_1, a_2, c_2, p_1, \) and \( p_2 \)), and \( MSE(v_1, v_2) \) is \( \frac{1}{n} \sum_{i=1}^{n} (v_{1i} - v_{2i})^2 \).

2. A penalty term that penalizes negative element amounts which helps the optimization to avoid unrealistic values:

\[ \text{penalty} = \gamma \sum_{e \in E} \min(\theta_e, 0)^2 \]

where \( \theta_e \) is the value of \( \theta \) corresponding to element \( e \), and \( \gamma \) adjusts the strength of the soft penalty relative to the fit (we used 0.01 because this worked best with ADAM).

This model defines a high-dimensional search space. The prevalence of points with a near-zero gradient in some dimensions (such as local minima and saddle points) that are not a global minimum can slow down [64] first-order gradient-based optimization methods. Adaptive optimizers, such as ADAM [126], can often find a good solution more quickly than other optimizers [218]. We found that using ADAM and multiple rounds of gradient descent fitting worked well on the
rock dataset. The gradient descent rounds are performed in sequence, with initial parameter values of each round set to the final values of the previous round. In each round described below, the specified parameters are optimized while all others are held fixed at the value obtained in the prior round.

1. Fe and Cu amounts ($\theta_{Fe}, \theta_{Cu}$) are optimized first because the samples are high in these elements and these elements produce strong peaks in the XRF spectrum.

2. Continuum approximation parameters ($p_1, p_2, \alpha$) are optimized, with Fe and Cu amounts held fixed. For this round, mean absolute error (MAE) is used as the loss function instead of mean squared error (MSE) because a better fit is produced. It fits better because tall peaks in the spectrum act like outliers which have too much “pull” when MSE is used [170, 195] whereas MAE fits the majority of channels overall better.

3. 24 more elements (26 in total) are optimized.

4. Attenuation and efficiency approximation parameters ($a_1, c_1, a_2, c_2$) are optimized. A reasonable initial value is important; we used $a_1 = 1$ and $a_2 = -1$, which describes the steepness and direction of the sigmoid curves (1 is a wide, gradual incline, whereas 10 is a very sharp transition; positive means the line rises from left to right, negative means the line decreases), and $c_1 = 0$ and $c_2 = 24$, which describes the location of the sigmoids in keV (a good guess is the far left and far right of the spectrum).

5. The element amounts are optimized further.

6. FWHM ($\Gamma$) is optimized.

7. The element amounts are optimized further, but with a lower step size, in order to fine-tune the fit more precisely in this final round.

A manual process is used to configure the rounds. The sequence of rounds can be used to incorporate domain knowledge, by starting with a simpler problem then gradually increasing problem complexity. Gradually changing the objective function towards the desired one has been shown [96] to help the optimization process discover more effective local minima. For instance, users can prioritize likely elements, based on domain knowledge of the region and its geology, and fit them first. Subsequent rounds fit additional elements as desired, such that elements with probable peaks are fitted early and ending with smaller, less significant elements. Intermediate rounds may also be included to fit global simulator parameters, which
can be difficult to fit because they affect the entire spectrum. Once configured, the sequence of gradient descent rounds is run on all the rock samples automatically; this step does not require any labelled data. The resulting parameter values are then calibrated (using labelled data), through PLS regression, to predict the concentration of any element. This prediction model is compared to other models in Chapter 4.

2.5 Conclusion

Existing forward models strive to be as accurate as possible to the underlying physics, but this isn’t always practical when instrument and environment details are unavailable. An XRF simulator designed from domain knowledge approximates attenuation and efficiency curves such that instrument and environment properties are not required.

FitSim tunes the XRF simulator parameters, such as element amounts, until the simulated spectrum is as close as possible to the observed spectrum over a number of sequential user-configured optimization rounds. The final fitted parameter values are used to predict element concentrations in Chapter 4. It should be acknowledged that when dealing with a new dataset or a new domain, the sequence of gradient descent rounds should be adapted.
Chapter 3

Auto-Encoder using Simulator

The space of prediction models has been described [168, 235] as spanning from knowledge-intensive to knowledge-free and that “staying at one end or the other of the spectrum of possible learning systems simplifies the learning problem by allowing strong assumptions to be made about the nature of what needs to be learned. However, the middle ground is appealing; it offers the possibility that synergistic combinations of theory and data will result in powerful learning systems” [235]. The approach in this chapter sits in this “middle ground.” The work in this chapter was published [69] in the journal of X-ray Spectrometry and an earlier version [67] appeared in the workshop on Perception as Generative Reasoning at the Neural Information Processing Systems conference in 2019.

3.1 Introduction

In X-ray spectrometry, the knowledge-intensive end includes the “fundamental parameters method” [65, 131]. Fundamental parameters methods, including model-based fitting methods such as FitSim (Chapter 2), use physical principles and established parameters of X-rays and their interactions with a target sample, the environment, and the spectrometer. Fundamental parameters methods are often used to quantify elemental composition because the underlying physical principles are well-understood in XRF [65, 131].

We consider XRF applications where fundamental parameter methods are impractical, typically because instrument or environment parameters are unavailable. For instance, in sensors mounted on mining equipment\(^1\), moving rocks introduce angle and distance variations and environmental variations complicate modelling.

---

\(^1\)See example mining applications at https://www.youtube.com/watch?v=zHCzqIP_OuA, https://www.youtube.com/watch?v=IKyVvRk7sig, and https://www.youtube.com/watch?v=eH84AaxI HvQ.
Neural networks have been used as an alternative [118, 122, 145] to the fundamental parameters method. However, neural networks require a lot of labelled data but obtaining spectra with corresponding elemental composition is often expensive. For example, in mining, a geochemical assay requires grinding and crushing large rock samples which is an expensive process.

Aside from training with more data, the performance of machine learning models can also be improved by incorporating domain knowledge; this is an attractive way to improve generalizability [96, 124]. Utilization of fundamental parameters within a neural network has only been achieved indirectly. For instance, early work [150] calculates the theoretical relative intensities of each element using fundamental parameters (and instrument parameters). A small fully-connected neural network (with 1 hidden layer of size 10 and 1 output) maps these intensities to concentrations. This network does not benefit from machine learning advancements made in the last 20 years. Recent work [118] trains a convolutional neural network on simulated spectra generated using fundamental parameters before fine-tuning the model on real spectra. Our approach differs from both of these in that we incorporate a forward model, that simulates a spectrum given the elemental composition, directly in a neural network.

The proposed neural network learns the inverse of the fundamental parameters method alongside simulator parameters used to approximate the effects of the instrument and environment. Fundamental parameters, specifically transition energies and probabilities, are built into the simulator. The specific neural network architecture we use is an auto-encoder [4, 109] where the encoder transforms spectra to a lower-dimensional encoding representing properties of the rocks and the decoder translates properties of rocks back into spectra through a trainable simulator.

The model is tested on a rock dataset from a lithium mineral exploration project to demonstrate its potential. We found that the auto-encoder model outperforms the baselines and other neural networks on 11 of the 18 elements for which some methods are better than predicting the average. The 11 elements include some low-Z\(^2\) elements (Li, Mg, Al, and K) and high-Z elements (Sn and Pb). This achievement is noteworthy because low-Z and high-Z elements are outside the suitable range for common XRF spectrometers to directly measure. This success is likely attributed to the ability of neural networks to learn correlations and non-linear relationships combined with domain knowledge in the forward model.

\(^2\)Z is the atomic number of an element. The atomic number determines an element’s position in the periodic table. Low-Z and high-Z elements refer to light and heavy elements respectively.
3.2 Related Work: Integrating Domain Knowledge in Machine Learning

There are various ways to incorporate domain knowledge in neural networks. Although not exhaustive, several illustrative examples are given:

- Gülçehre et al. [26] introduce domain knowledge in the form of intermediate ground truth. They train a neural network model for the intermediate supervised task before transitioning to the primary target task.

- D’Andrea et al [62] incorporate domain knowledge of LIBS using a physics-based model to process a spectrum, whose output is used as input to a neural network.

- Another approach is to break down a given task into simpler sub-tasks, followed by manual coding of selected sub-tasks using domain knowledge [174].

- When domain knowledge pertains to connections between variables, a sparsely-connected neural network can be used where connections correspond to a priori known relationships between variables [215].

- For domain knowledge about the importance of features, Jenul et al use a Bayesian approach to feature selection combining data-driven features with expert judgments [116].

In this chapter, a mathematical model simulating XRF spectra is embedded within a neural network architecture (see Section 3.4.5).

3.3 Dataset

Labelled data from the rock dataset (Section 1.2.1) is used in this study, consisting of 177 drill core rock samples with XRF spectra and corresponding geochemical assays of 48 elements.

3.4 Models

Prediction models, such as linear regression or neural networks, require data for training. In this study, the training data set consists of a spectrum per rock labelled with corresponding composition provided by geochemical analysis. Labelled data is often limited because of the excessive cost and time, and destructive nature,
of geochemical analysis. Limited labelled data is challenging because highly-parameterized models, such as neural networks, are prone to overfitting. We test a range of models with different numbers of parameters and different regularization schemes, explained below.

All models are evaluated by 10-fold cross-validation [84] (Section 1.4). The MSE across the 10 folds is an estimate of generalization ability with standard error (SE) [93, 104] of this prediction estimated\(^3\) as:

\[
\frac{1}{\sqrt{k}} SD\{CV_1, \ldots, CV_k\}
\]

where \(k\) is 10, \(SD\) is the standard deviation, and \(CV_i\) is MSE of the \(i^{th}\) fold.

### 3.4.1 OLS

Ordinary least squares (OLS) is a method for estimating the parameters of a linear regression model. In this case, a *simple linear regression* [133] model is used which has one input and one output variable. We train one linear regression model per element. In each model, the input is the photon count at the K-L\(_3\) (K\(\alpha\)) line for the element and the output is the element’s concentration. These models have the fewest parameters of the models tested in this chapter.

### 3.4.2 LASSO

We train a LASSO (Section 1.3.2) model for each element. The inputs are the full spectrum.

### 3.4.3 FCNN

Three neural networks are considered, of which the first is a single-layer fully-connected neural network (FCNN). It uses one fully-connected layer (Section 1.3.2). The model’s input, a spectrum, is directly connected to 48 outputs representing element concentrations. A ReLU [159] activation function is used on the output layer to clip negative values to zero. As per standard practice, early-stopping, dropout, and \(L_1\) regularization are used to reduce over-fitting [119].

For all the neural networks tested in this chapter, grid search found the following hyperparameters: learning rate is 0.001, early-stopping patience is 1000 epochs, \(L_1\) regularization factor is 0.001, and dropout starts at 50% probability and gradually backs off until dropout is disabled at epoch 10000. Training is run until early-stopping criteria is met. FCNN trained for 16000 epochs on average. All the

---

\(^3\)The standard deviation used to estimate SE is commonly used but tends to underestimate the true standard deviation [93] which cannot be unbiasedly estimated [24]. Other methods have been proposed [19, 94]. Later in this thesis, we switch to showing boxplots [133] to visualize the distribution because this gives more information to the reader.
neural networks were programmed in Python version 3.6 using TensorFlow version 1.13.

3.4.4 CNN

The second neural network is a convolutional neural network (CNN, see Section 1.3.2). This type of model is common in other spectroscopy domains [2, 6, 52, 60, 75, 83, 148, 151, 160, 251, 263] but not in XRF. The CNN’s architecture can be seen in Figure 3.1 where it is equivalent to the Encoder module followed by the Output module. The first layer is a convolutional layer with one 20-unit filter. Then, there are two fully-connected layers each with 15 units. All the layers use sigmoid [159] activation functions, including on the output, which allows the network to learn non-linearities. The fully-connected layers learn the dependencies between features that may be present in the data. Early-stopping, dropout, and $L_1$ regularization are used to reduce over-fitting. This model trained for 12000 epochs on average.
Figure 3.1: Architecture of AutoSim. The CNN architecture from Section 3.4.4 is equivalent to the Encoder module followed by the Output module. The Encoder module is a function that takes in a spectrum and outputs the intermediate representation, or encoding. The XRF Simulator module takes in the intermediate representation ($\theta$ and $\alpha$) and outputs a simulated spectrum. The Output module takes in the intermediate representation and outputs the concentration (%) of each element.
3.4.5 Analysis-by-Synthesis

The final model is an implementation of analysis-by-synthesis [99], which we call AutoSim, using an auto-encoder neural network. Auto-encoders [4, 109] are commonly used to build low-dimensional informative representations [27, 137]. Analysis-by-synthesis has been used in computer vision, for instance, where a graphics engine generates images and the neural network learns the inverse process which translates images into properties of physical objects [232, 250]. AutoSim consists of 3 modules (shown in Figure 3.1): Encoder, XRF Simulator, and Output, described below.

**XRF Simulator Module**

The XRF Simulator module, shown in Figure 3.1, is a function that takes in the intermediate representation (\( \theta \) and \( \alpha \)) and outputs a simulated spectrum. In a standard auto-encoder, the output of the encoder is the input to a decoder neural network. In AutoSim, the decoder of a typical auto-encoder is replaced by the XRF simulator developed in Section 2.3. Using the TensorFlow software library, the XRF Simulator is programmed to be fully-differentiable; differentiation is a requirement of the backpropagation algorithm used to train neural networks. Unlike FitSim (Chapter 2), global parameters are learned during neural network training and rock-specific parameters (\( \theta \) and \( \alpha \) parameters, from the XRF simulator in Section 2.3) are tied directly to the output of the Encoder module.

**Encoder**

The Encoder module (shown in Figure 3.1) is a function that takes in a spectrum and outputs the intermediate representation, or encoding. The intermediate representation consists of the parameters, \( \theta \) and \( \alpha \), needed by the simulator (Section 2.3). The Encoder architecture details are the same as the CNN model (Section 3.4.4).

**Output Module**

The Output module, shown in Figure 3.1, is a function that takes in the intermediate representation (\( \theta \) and \( \alpha \), defined in Section 2.3) and outputs the concentration (%) of each element. Note that \( \theta \) is unitless (because it is a scaling factor used to obtain photon counts). A benefit of keeping \( \theta \) and element concentrations separate is that we can output predictions for elements that aren’t represented in the XRF spectrum. For example, lithium (Li) is not directly measurable in XRF, so the amount of lithium in the intermediate representation would rightly be zero, but the
The model can still learn to predict the lithium concentration given correlations to other elements in the sample.

The **Output** module consists of two fully-connected layers with $L_1$ regularization, sigmoid activation function, and dropout. The first fully-connected layer connects to 100 hidden units and the second layer connects to the output. All 48 elements are predicted simultaneously, in the one model. AutoSim trained for 55000 epochs on average.

**Objective Function**

The objective function that is minimized is a mix of a reconstruction error—how well the simulator fits the input spectrum—and a prediction error—how accurate the prediction is.

As in an auto-encoder, a reconstruction error, $J_r$, is minimized causing the **Encoder** and **XRF Simulator** modules to learn to reconstruct the training spectra. $J_r(x_s, x'_s)$ is the mean squared error (MSE) between observed, $x_s$, and reconstructed, $x'_s$, spectra:

$$J_r(x_s, x'_s) = \frac{\sum_{i=1}^{m} (x_{si} - x'_{si})^2}{m}$$  \hspace{1cm} (3.1)

where $x_{si}$ and $x'_{si}$ are the $i^{th}$ channel of the observed and reconstructed spectra of sample $s$ respectively, and $m = 1024$ is the number of channels in the spectrum (indexed by $i$). $x'_s$ is the reconstructed spectra produced by the **XRF Simulator** function, $f$.

Prediction error, $J_p$, penalizes the difference between predicted concentrations and ground truth (given by geochemical analysis):

$$J_p(y_s, y'_s) = \frac{\sum_{e \in E} (y_{se} - y'_{se})^2}{|E|}$$  \hspace{1cm} (3.2)

where $y_{se}$ and $y'_{se}$ are the actual and predicted (respectively) concentrations of element $e$ and sample $s$, $E$ is the set of all elements in use, and $|E|$, the number of elements, is 48. $y'_s$ is produced by the **Output** module. Finally, a weighted sum of the reconstruction error and prediction error yields the loss function, $J(w)$, used to train the model:

$$J(w) = \sum_s J_p(y_s, y'_s) + \beta J_r(x_s, x'_s)$$  \hspace{1cm} (3.3)

where $s$ is a sample, $w$ is the set of all tunable weights (which are the neural network weights from the **Encoder** and **Output** modules and the global parameters from the **XRF Simulator**) and $\beta$ is the weight of the reconstruction loss (which is a
hyperparameter). Optimizing $J_r$ and $J_p$ together has been shown to be better than optimizing prediction error, $J_p$, alone and better than pre-training with the reconstruction error followed by optimizing the prediction error [136].

### 3.5 Results

The OLS, LASSO, FCNN, CNN, and AutoSim models are evaluated and compared. The CNN is included as an ablation study to test what happens when the decoder is removed from AutoSim. 48 elements were included in geochemical analysis, 30 of these were removed from the results because none of the models were able to do better than predicting the mean. Some elements have very low abundance and little variation in the samples studied, making the results inconclusive for these elements. Of the remaining 18, AutoSim achieved the best MSE on 11 elements, summarized in Table 3.1. A full list of expected prediction errors and standard errors for all 18 elements is given in Appendix A.1. OLS (Section 3.4.1) is not applicable to low-Z elements (low-Z elements are the light elements with a low atomic number towards the beginning of the periodic table), so no results are reported for these. Low-Z elements are particularly challenging for several reasons including small excitation factors, high attenuation, and increased scattering ([131], page 204). Li is of particular interest in this project; AutoSim achieved the best prediction error on lithium (Li, Z=3) as shown in Figure 3.2. From the results (Appendix A.1, Figure A.1) we see that AutoSim outperformed the baselines and other neural networks on several low-Z elements (Li, Mg, Al, and K) and high-Z elements (Sn and Pb) despite these elements being outside the suitable range for the spectrometer to directly measure. Gallium (Ga) is well within range for the spectrometer but did poorly when calibrating against the K-L3 (Kα) peak directly (using OLS), whereas the multivariate models (LASSO, FCNN, and CNN) do much better and AutoSim does the best. For the remaining elements (where AutoSim was not the best model), the results were roughly tied between the competing models.

We might expect OLS, FCNN, CNN, and AutoSim to improve upon each other. FCNN is expected to outperform OLS because FCNN has more parameters and utilizes the whole spectrum (albeit, at the risk of overfitting); CNN is expected to outperform FCNN because of the regularization power of the convolution layer; and AutoSim is expected to outperform CNN because it incorporates domain knowledge. Such monotonic improvement is observed in 3 elements (Al, Ga, and Pb), similar to the one shown in Figure 3.3 (graphs for all elements are shown in Appendix A.1, Figure A.1). 6 elements (Ca, Cs, Mo, S, Sb, and Sr) performed oppositely—with FCNN’s MSE less than all other neural networks—but standard error is large, suggesting performance could have gone either way. CNN
Table 3.1: Elements where each model achieved the best MSE.

<table>
<thead>
<tr>
<th>Model</th>
<th>Best on</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLS</td>
<td>S</td>
</tr>
<tr>
<td>LASSO</td>
<td>Rb, Sb</td>
</tr>
<tr>
<td>FCNN</td>
<td>Ca, Cs, Mo, Sr</td>
</tr>
<tr>
<td>CNN</td>
<td></td>
</tr>
<tr>
<td>AutoSim</td>
<td>Al, Be, Fe, Ga, Hf, K, Li, Mg, Pb, Sn, Zr</td>
</tr>
</tbody>
</table>

did not have the best MSE on any elements, but it was the runner-up for 4 elements (Al, Ga, Pb, Sn) as can be seen in Appendix A.1.

Figure 3.2: Example where AutoSim does well but CNN does not. Results for OLS are absent because Li does not have a K-L3 (Kα) line.

Figure 3.3: Example where MSE improves with each more complex model (from left to right). Error bars are ± 1 Standard Error.

3.6 Discussion

From looking at spectral reconstructions we can gain an insight into what the model does well. Example reconstructions from AutoSim’s auto-encoder is shown in Figure 3.4. Reconstruction RMSE for the majority of samples is around the median RMSE of 2.58, such as examples B, C, and D in Figure 3.4; many of the peaks fit very well. Peaks below 5 keV rarely fit well, such as in examples A, B, and E.
Figure 3.4: Example spectra (black) and AutoSim reconstructions (red) for 5 different samples. A has the lowest root mean squared error (RMSE) and E has the highest RMSE. B, C, and D have RMSE near the median (which is 2.58). The mean RMSE is 3.01.
This may be due to interference and increased photon scattering from the rough rock surface. Improving the simulator to account for secondary fluorescence (as in a recently published XRF simulator [46]), sum peaks, diffraction peaks, escape peaks, and other artificial peaks [78, 225] may help the poor-fitting regions of the spectra and improve the reconstructions, which may then lead to improved predictions too. Implementing this functionality in a differentiable manner, as required for the method described in this chapter, would be a good way to improve upon AutoSim.

AutoSim has a number of benefits and limitations. AutoSim has the benefit of not requiring instrument parameters because it learns to model the XRF spectra from data. However, since it relies on data, AutoSim needs to be retrained if the instrument changes or if the distribution of distances, rock sizes, or mineralogy changes. AutoSim inference is fast compared to peak-fitting routines, like FitSim (Chapter 2), that run an iterative optimization routine to fit each new observed spectrum. AutoSim requires only a single pass through the Encoder module to make a prediction and it does so for all 48 simultaneously. The XRF Simulator module assumed a Lorentzian peak shape (see Section 2.3), which may lead to poor reconstructions if this is inaccurate. However, a benefit of the Encoder in AutoSim is that it learns an appropriate filter for feature extraction, whether the shape is approximately Lorentzian, Gaussian, Voigt, their derivatives, or some other shape. Lastly, a benefit of auto-encoder architectures, such as AutoSim, is the ability to train in a semi-supervised [121] fashion; the Encoder and XRF Simulator can be trained on unlabelled spectra while the whole model is trained on spectra labelled with composition provided by geochemical analysis. For example, in a mining shovel or conveyor belt application, it is expected that few samples will be sent for geochemical analysis (because large samples are more expensive) but spectra alone will be relatively cheap to obtain. Therefore, much of AutoSim can be trained on unlabelled data thus reducing the need to obtain many geochemical analyses.

Other variants of neural network architectures may also improve performance, but remain to be investigated, such as using deconvolutional layers (in the decoder) [4], variational auto-encoders [127], and others. The analysis-by-synthesis method explored in this chapter may also be applicable to other spectroscopy disciplines, assuming the XRF spectra simulator is replaced with an appropriate simulator.

### 3.7 Conclusion

The problem of XRF spectroscopy quantification where labelled data is limited and the fundamental parameters method is not directly applicable was investigated. As a proof-of-concept, an analysis-by-synthesis style auto-encoder trained on rock core samples demonstrated the potential of this method. We combined (1) learning
from limited labelled data with a neural network and (2) an XRF spectra simulator based on fundamental parameters. In experimental results, we observed improved predictions on 11 elements, 6 of which fall outside the ideal range of the XRF spectrometer. We are confident that this method can be further refined and will extend the reach of XRF to more difficult applications.
Chapter 4

Auto-Encoder using Spectra Decoder

4.1 Introduction

Simulators defined using sensor-specific attributes (such as the one in Chapters 2 and 3) are not always available, or accurate. Learning a simulator from data would allow any type of spectra to be used. For many types of spectroscopy, spectra have something in common: peak-shaped features. In this chapter, a neural network architecture is designed for spectra using domain knowledge of spectra in general.

In XRD, for instance, the expected spectrum can be expressed as a convolution between a peak function and a diffraction profile function which contains the theoretical locations and heights of all the peaks [5, 53, 205]. XRF spectra can be generated similarly. In XRF, modelling the peak as a Gaussian function is common and often sufficient, but imperfections in the detector cause deviations from this shape depending on the energy (keV) [80, 241]. Many alternatives have been proposed, such as a Voigt function—the convolution of a Lorentz distribution with a Gaussian [80]. Convolutional layers are well-suited to learn the peak function.

An AE for spectra, SpectraAE, is proposed where the decoder models a spectrum using standard neural network layers: fully-connected layers model the line profile and convolution filters model the peak function and other structures in the spectrum. While auto-encoders have been utilized for spectroscopy [149, 190, 255, 256], to our knowledge, none have tried to incorporate domain knowledge of the spectrum-generation process.

SpectraAE is tested on the rock dataset (Section 1.2.1), but the architecture is general and may be used on spectra from any type of spectroscopy. The rock
dataset contains 491 samples, 177 of which are labelled (see Section 1.2.1). All 491 samples are used in training SpectraAE with unsupervised learning. Two convolutional neural networks are also developed and a thorough evaluation scheme is employed to compare these models to standard regression models (LASSO and PLS). Models from the previous two chapters (FitSim and AutoSim) are also compared.

4.2 Background of Semi-Supervised Learning and Auto-Encoders

Supervised learning only utilizes labelled data. Semi-supervised [51, 93] learning uses both labelled data and unlabelled data. Semi-supervised models have been used in spectroscopy and chemometrics [117, 234], though these models are not neural-network based.

One method of semi-supervised learning [265] is to separately learn a representation using an unsupervised learner followed by using a supervised learner to predict labels from the learned representation. For example, one can perform dimensionality reduction (e.g., PCA) on all the samples (while ignoring any labels), then apply a regression model in the lower-dimensional space using only labelled samples. Kaneko [121] demonstrates this approach and discusses the merits of semi-supervised learning in regression analysis for chemometrics.

In this work, we use auto-encoders [22, 93] to combine unsupervised and supervised learning. Auto-encoders are neural networks trained to reconstruct their inputs by minimizing the difference between the inputs and the reconstructions of those inputs. In doing so, the auto-encoding architecture learns to project the inputs into a lower-dimensional encoding, or representation, that contains information to reconstruct the input [27]. Auto-encoders are one of many methods for representation learning. Principal component analysis (PCA) does a similar projection; a special case of the auto-encoder—namely an auto-encoder with one fully-connected layer, linear activations, and squared-error loss function—can be constructed whose weights span the same subspace as the subspace of the principal components [15, 193].

Bengio et al. [25] proposed to use an auto-encoder for semi-supervised learning using unsupervised pre-training of a neural network followed by supervised learning with the network’s weights initialized by the pre-trained network. Unsupervised pre-training has been found to prevent overfitting when the number of labelled samples is small [140] and to help find more effective local minima [26]. Examples of this in the chemometrics domain (applied to spectra from hyperspectral imaging) include stacked auto-encoders followed by a support vector
machine (SVM) [228] and a fully-connected neural network [255]. Later works expanded on this idea by training one auto-encoder network with unsupervised and supervised data simultaneously [141, 198]; we use a similar approach for spectral data.

4.3 Models
Two baseline models (LASSO and PLS) and two CNN models are developed and compared. The second CNN is used within the AE model. Additionally, the methods developed in Chapters 2 and 3 are also compared.

4.3.1 LASSO
One LASSO model (defined in Section 1.3.2) is trained per target element. It has one hyperparameter, \( \lambda \), which controls the amount of regularization and is optimized by performing a grid search over the base-10 logspace from 8e-7 to 100. The search space includes 201 values that are equally spaced over the logspace.

4.3.2 PLS
One PLS model (defined in Section 1.3.2) is trained per target element. It has one hyperparameter, the number of components (from 1 to 160), which is optimized through grid search.

4.3.3 FitSim
FitSim (Chapter 2) is used to predict the concentrations of all 48 target elements. It has one hyperparameter per target element which is the number of components (in the PLS portion of FitSim). Hyperparameters are optimized through grid search over the integers from 1 to 26.

4.3.4 CNN-1
The first convolutional neural network model, CNN-1, is shown in Figure 4.1. Its architecture is common in chemometrics [263]. The main components of this model are a convolutional layer followed by a fully-connected layer to the final outputs. The convolutional layer has 8 filters, each of size 21 \( \times \) 1. 21 was chosen because it is approximately the size of a whole peak in XRF. Zero-padding is used, which keeps the output the same size as the input. Prediction error (which is MSE) between the output and the ground truth is minimized, such that the output is the predicted concentration (in %) of each element.
4.3.5 CNN-2

The second convolutional neural network model, CNN-2, is shown in Figure 4.2. This model differs from CNN-1 in that there is an additional layer of convolutions with different filter sizes. These capture the notion that some features are narrow (like peaks) whereas others are wider (like the background continuum). The neural network architecture was initially based on “Model 2” by [263], but after hyperparameter tuning (described in Section 4.5.1) we found that 1 fully-connected layer had the lowest error. The main components are a convolutional layer (with 3 filters of size 5) followed by 3 parallel convolutional layers which differ in number of filters and filter size: (1) 3 filters of size $1 \times 3$, (2) 2 filters of size $5 \times 3$, and (3) 1 filter of size $21 \times 3$. Convolutional layers are followed by a fully-connected layer to the final outputs. Prediction error (measured by MSE) between the output and the ground truth is minimized, such that the output is the concentration (in %) of each element.

4.3.6 SpectraAE

The auto-encoder (SpectraAE) model consists of an encoder that is the same as CNN-2, and a decoder, shown in Figure 4.3. Fully-connected layers in the decoder model the notion of spectral “lines”—the location and intensity of peaks—and any
Figure 4.2: Neural network architecture of CNN-2.
other structures in the spectrum such as background continuum, Raleigh peaks, etc. Since some structures (like peaks) are narrow and some are wide (like continuum), four parallel branches of fully-connected layers are used, each with a different size. The layer with 1024 units matches the size of the spectrum and captures granular details. The layers with 512 and 341 ($\approx \frac{1024}{3}$) units capture less-granular details. The last layer, with 256 units, captures wider structures. A convolutional layer follows each fully-connected layer which creates “peaks” (or any other structure) at the locations given by the preceding fully-connected layers. The convolution filter size is 5 units to force the filter to learn simple structures (like peaks) rather than, say, copying an entire observed spectrum exactly. The outputs of the convolutional layers are upsampled, using linear interpolation\(^1\), to match the size of the spectrum. The four parallel layers are then combined by element-wise summation. The output

\(^1\)Upsampling is performed by the \texttt{tf.image.resize\_images} function in TensorFlow 1.13 with bilinear interpolation.
is a 1024-length vector (labelled “Reconstruction” in Figure 4.3). Reconstruction error (MSE between observed spectra and reconstructed spectra) is minimized with unsupervised learning such that the input spectra resemble the reconstructions as closely as possible.

The output of the encoder is called an encoding and is used as (1) the input to the decoder and (2) the predictions of element concentrations. The latter is enforced through supervised learning: prediction error (MSE between the encoding and ground truth) forces the encoding to represent element concentrations. In our implementation, the training procedure alternates between optimizing the supervised loss function (which is the prediction error) and optimizing the unsupervised loss function (which is the reconstruction error). A hyperparameter scales the unsupervised loss function to balance between the two loss functions.

4.4 Dataset

Figure 4.4: One example of an XRF spectrum from the rock dataset. In this spectrum, the primary peaks, $K\alpha_1$, of sulfur (S), potassium (K), calcium (Ca), iron (Fe), arsenic (As), rubidium (Rb), molybdenum (Mo), and silver (Ag) are labelled in the plot. Most of the time, the height of an element’s primary peak correlates to the concentration of that element in the rock sample. However, there are exceptions to this, such as matrix effects that cause an element’s peak to be higher. Also, the silver (Ag) peak is due to the silver present in the X-ray tube.
Models are trained on the rock dataset (Section 1.2.1), consisting of 177 labelled samples and 314 unlabelled samples. XRF spectra were obtained on all labelled and unlabelled samples (for example, see Figure 4.4). The labelled samples are used in training all the models, using all 48 elements of the geochemical assay. The unlabelled samples do not have any ground truth available but will be used in training the SpectraAE model.

As per standard practice, input and output variables are normalized. Since the range of each element’s values varies, values are normalized against the element’s range (mapped to the interval \([0, 1]\)) so that each element is treated equally by the training algorithm. All spectra are normalized by dividing by the maximum photon count (of the training set of each fold). This ensures the values of the input data (spectra) in the training set are in the interval \([0, 1]\).

4.5 Experiment Setup

4.5.1 Neural Network Training

For all the neural networks in this chapter (CNN-1, CNN-2, and SpectraAE):

- Models were implemented in Python 3.6 using TensorFlow (v1.13).
- The supervised loss function is mean squared error (MSE). SpectraAE also optimizes an unsupervised loss function, the reconstruction error.
- Training runs for a maximum of 30000 epochs.
- Learning rate (LR) starts at 0.001 and anneals after 2000 epochs. The learning rate at epoch \(e\) is \(0.001 \times \min(1, \frac{2000}{e})\).

The following methods are used to reduce over-fitting:

- Early-stopping and cold restarts: If the dev set loss worsens for more than 2000 epochs then (1) training is stopped, and (2) training is started again from random initial weights (known as a cold restart). A computational budget of 30000 epochs across cold restarts is enforced. The trained model with the best dev set loss, across cold restarts, is the final model.
- Dropout is applied to every fully-connected layer as well as to the input spectrum. When dropout is applied to a layer, it zeroes out the output units with some probability \([110, 217]\). The probability of dropping out starts high then gradually reduces to zero. The probability of dropping a unit at epoch
\[ e = \max(0, p - \frac{p}{E}) \] where \( p \) and \( E \) are hyperparameters: \( p \) is the initial probability and \( E \) is the number of epochs to transition to probability 0.

- Weight decay: \( L_1 \) regularization is applied to all fully-connected layers. \( L_1 \) regularization pushes layer weights toward zero, which promotes sparsity. For each layer, \( \lambda \sum_{i=0}^{N} |w_i| \) is added to the loss function, where \( N \) is the number of units in the layer, \( w_i \) is the weight of the \( i^{th} \) unit of the layer, and \( \lambda \) scales the penalty term (a higher value encourages smaller weights and 0 disables regularization).

### 4.5.2 Data Partitioning

The experiments in this chapter aim to evaluate and compare each model’s performance on average over any possible partitioning of the dataset into training and test sets. For this reason, the training and test sets are repeatedly sampled (using repeated cross-validation). Hyperparameters should be chosen to optimize a set of data outside the test set to avoid overfitting during HPO [50, 86, 199]. As shown in Figure 4.5, we reserve 20% of the labelled samples (which is 36 samples) for use in HPO as a proxy [210] for the whole dataset. HPO and evaluation use separate cross-validation schemes (shown in Figure 4.5):

- Cross-validation on the proxy set is used for optimizing model hyperparameters (Section 4.5.3).

- Cross-validation on the remaining labelled samples is used for assessing generalization performance (Section 4.5.4).

### 4.5.3 Hyperparameter Optimization (HPO)

Tuning hyperparameter values is a time-consuming and compute-intensive process. Since a separate subset of the dataset is used for tuning, HPO is only conducted once. However, the small proxy set raises two challenges.

Since the proxy set must be partitioned into a training and validation set for model training, the validation set will be small which leads to high variance performance estimates [50]. Cross-validation (6-fold) on the proxy set reduces the variance of the validation set score [50]. Between five and ten folds are typically recommended; six folds were chosen because 6 divides the 36 data points evenly.

The second challenge is that any model trained on a subset of the dataset will not perform as well as one trained on the whole dataset [199], which means that any performance estimate obtained from the proxy set is expected to be worse than one obtained from the whole dataset. However, for the purpose of picking
Figure 4.5: Diagram showing how dataset examples are partitioned and utilized. 

Hyperparameter values, the performance estimate only needs to indicate where the optimal performance lies in hyperparameter space [50]. For this study, we assume a small subset for tuning will be sufficient for the purpose of HPO.

Hyperparameter values are chosen to minimize the proxy set error, the average RMSE over 6-folds of cross-validation. The neural network hyperparameters are optimized through an iterative process of searching over all combinations of values for a handful (1 to 5) of hyperparameters and freezing the rest. Details of the search space are given in Section B.2.

4.5.4 Evaluation Phase and Statistical Testing

Once the hyperparameter configuration is chosen for each model, we switch from the HPO phase to the performance evaluation phase (see Figure 4.5) to compare performance of the models. The models are evaluated for their average performance across all the assayed elements. Repeated cross-validation (see Section 1.4) is applied, followed by statistical testing (using the paired $t$-test) to determine when one model’s cross-validation results outperform another model’s and whether or not this difference is significant. Details of repeated cross-validation and paired $t$-testing are explained in the remainder of this section.

Ten repeats of 10-fold cross-validation ($10 \times 10$-fold CV) is applied to labelled samples that are not in the proxy set (shown in the data partitioning diagram in Figure 4.5). Additionally, SpectraAE training uses all unlabelled samples in unsu-
Supervised learning. Pseudo-code in Figure 4.6 summarizes the process.

```plaintext
1 \(L\) is the set of labelled examples (177 examples)
2 \(D_{proxy}\) = select 36 examples from \(L\). // This is used in HPO.
3 \(D_{unlabelled}\) = the set of unlabelled examples (314 examples)
4 function evaluate()
   // For each CV repetition:
   for \(r\) in [1..10] do
     // Prepare data for 10-fold CV:
     \(P_{1..10}\) = shuffle \(L \setminus D_{proxy}\) and partition into 10 approx. equal sets
     // For each fold:
     for \(k\) in [1..10] do
       for \(m\) in [LASSO, PLS, CNN, SpectraAE] do
         if \(m\) is SpectraAE then
           // Only the semi-supervised model utilizes unlabelled samples:
           \(f = \text{train } m\) on \((L \setminus D_{proxy} \setminus P_k) \cup D_{unlabelled}\)
         else
           // Train on training set (consisting of labelled samples that are not in proxy or test sets):
           \(f = \text{train } m\) on \(L \setminus D_{proxy} \setminus P_k\)
       for \(e\) in 48 elements do
         \(\text{CVscores}_{m,r,k,e} = \text{prediction error of } f\) evaluated on \(P_k\) for element \(e\)
   return \(\text{CVscores}\)
```

Figure 4.6: Performance evaluation procedure. (The notation \(A \setminus B\) is the set difference.)

To allow for a better comparison between any two models, all the models in this chapter use exactly the same subsets of samples for cross-validation. That is, given an iteration of cross-validation, the same set of samples is seen by each model in training and validating the model. Since the samples are the same, any difference in RMSE between two models is due to modelling differences rather than sample selection. For each pair of models, \(m\) and \(m'\), and element, \(e\), there is a set of pairs:

\[
\{(\text{CVscores}_{m,r,k,e}, \text{CVscores}_{m',r,k,e}) | r \in [1..10], k \in [1..10]\}
\]

In statistics, this type of data is called “matched pairs.” Cross-validation with matched pairs is recommended [24] for comparing the performance of two learning algorithms. For example, Figure 4.7 shows the matched pairs from 10\(\times\)10-fold CV on the PLS and SpectraAE models (with Zr as the target).

The statistical test for matched pairs data is the paired \(t\)-test [85, 245]. It establishes whether a model is better or worse than another model by considering the distribution of differences (both the mean and the variance) between the two models’ RMSE values. The null hypothesis is that the mean of RMSE differences is
Figure 4.7: Example of matched pairs data: RMSE scores from all (100) iterations of repeated cross-validation for two models. For a given iteration of cross-validation (for a particular \( r \) and \( k \); see pseudo-code in Figure 4.6), a line connects the RMSE values from each model. The line is coloured green when SpectraAE RMSE is lower than PLS.
zero. The t-statistic is \( t = \frac{\bar{d}_n}{\sigma} \) where \( \bar{d} \) is the mean difference, \( \sigma \) is the standard deviation of the differences, and \( n \) is the number of paired observations (\( n = 100 \) for \( 10 \times 10 \)-fold CV). The t-value is compared to the critical value from the Student’s t-distribution at the desired level of significance. If the calculated t-value exceeds the critical value, we reject the null hypothesis. Rejecting the null hypothesis means the model with a lower mean RMSE is better and the result is statistically significant. Conversely, if the calculated t-value does not exceed the critical value, there is insufficient evidence to make any conclusion.

In this study, the paired t-test is calculated\(^2\) for all prediction targets (elements) and for all pairs of models. Since there are seven models, including predicting the average (PtA), the number of comparisons will be \( 48 \cdot \binom{7}{2} \) which is 1008. The standard threshold of significance is 5% ; all tests with \( p < 0.05 \) reject the null hypothesis. For \( p = 0.05 \) this means there is a 5% chance that the null hypothesis is wrongly rejected, a false positive. It is expected that 1 in 20 statistical tests conducted will reject the null hypothesis purely by chance. Since there are 1008 comparisons, there would be about 51 false positives and an approximately one in \( 10^{22} \) chance of obtaining zero false positives (which is nearly impossible).

In this study, false positives are more costly than false negatives. That is, for any reported differences between models, we want to be as certain as possible that the difference is real. False negatives, on the other hand, are somewhat self-correcting in that future practitioners will see that two models are “equal” and will have to perform their own experiments to pick between them (which will lead them to discover if the models are, in fact, different). Whereas, a false positive would lead a practitioner to believe that one model is better than another which may lead them to not test the lower-performing model at all, and thus there is no opportunity for correction.

To reduce the likelihood of false positives, we use a multiple comparison correction. The Bonferroni correction [12] is used, which sets \( \alpha \) to \( 0.05/T \) where \( T \) is the number of tests. With \( \alpha = 0.05/1008 \) (approximately 4.96e-5), the probability of zero false positives is raised to 0.95 and 1 in 20160 statistical tests will reject the null hypothesis purely by chance. It should be noted that by decreasing the rate of false positives, the rate of false negatives is increased. That is, there is a higher probability that two models report no difference in performance when there may actually be a difference. Continuing the example in Figure 4.7, the paired t-test result is \( p = 1.7 \times 10^{-14} \). Since \( p < \alpha \), the difference between RMSE distributions is statistically significant.

Results are reported in the next section. Some elements are eliminated from

---

\(^2\)The paired t-test results were calculated using \texttt{ttest\_rel} in the SciPy 1.2.1 Python package (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest\_rel.html).
the results due to low concentrations or lack of characteristic peaks in the spectra, causing all the models to make predictions no better than predicting the average concentration. This is determined by the result of a paired $t$-test between all the models and PtA. For each element, if no model achieves statistically significant improvement over PtA, then this element is not reported. For this reason, 16 of 48 elements are left out of the results.

## 4.6 Results and Discussion

### 4.6.1 Model Comparison

As expected, the CV scores (see Figure 4.8) from all models overlap significantly. The variance is expected to be high because the test sets (and training sets) are different between iterations. The variance due to the choice of the training and test sets indicates the reported test-set RMSE is sensitive to the particular samples chosen for the train and test sets. The sensitivity is likely because the dataset is small.

The mean RMSE indicates which model is best but doesn’t take into account the variance. The mean values are marked in Figure 4.8. All elements’ scores are reported in Appendix B, Table B.1. Mean RMSE (across all iterations of 10×10-fold CV) is lowest using the FitSim model (for Li) and CNN-2 (for Zr).

Since the boxplots overlap and the means are relatively close to each other, a better comparison is made by taking into account the corresponding iterations of cross-validation between any two models. For example, Figure 4.7 shows the RMSE in each iteration for the PLS and SpectraAE models (for Zr predictions), connected by a line between corresponding iterations (e.g. repetition 1, fold 1 of PLS and SpectraAE are connected). The number of times that SpectraAE’s RMSE is lower than PLS (coloured green in the figure) is 81 out of 100, meaning that in 81% of iterations, SpectraAE is better than PLS. Taking this a step further, the distribution of RMSE differences between matching pairs is shown in Figure 4.9c. Since the box for PLS is below the zero line, SpectraAE RMSE is lower than PLS in more than 50% of CV iterations. Figure 4.9 shows boxplots of differences for two elements (Li and Zr) and two models (SpectraAE and FitSim). To summarize this figure, SpectraAE RMSE is lower than the baselines (LASSO and PLS) and more-or-less on par with CNN-1 and CNN-2. For Li, FitSim achieves consistently lower RMSE than all other models tested; for Zr, FitSim beats LASSO and PLS but not any of the neural network models.

The results are further confirmed by statistical testing with the paired $t$-test. All pairs of models are compared, with the results for Li and Zr listed in Table 4.1. A
### Table 4.1: Paired t-test results for Li and Zr.

The paired t-test operates on pairs, so all pairs of models are given here. A "↓" indicates that Model 1 achieved lower mean RMSE than Model 2 and the result is statistically significant with the corresponding p-value listed in parentheses. A "↑" means Model 1’s RMSE is higher than Model 2’s RMSE. A "≈" indicates that we cannot reject the null hypothesis of equal averages.

<table>
<thead>
<tr>
<th>Model 1</th>
<th>Model 2</th>
<th>Li</th>
<th>Zr</th>
</tr>
</thead>
<tbody>
<tr>
<td>PtA</td>
<td>LASSO</td>
<td>↑ (p=8.4e-55)</td>
<td>↑ (p=1.3e-11)</td>
</tr>
<tr>
<td></td>
<td>PLS</td>
<td>↑ (p=9.7e-47)</td>
<td>↑ (p=4.1e-14)</td>
</tr>
<tr>
<td></td>
<td>FitSim</td>
<td>↑ (p=2.9e-54)</td>
<td>↑ (p=2.5e-16)</td>
</tr>
<tr>
<td></td>
<td>CNN-1</td>
<td>↑ (p=7.4e-54)</td>
<td>↑ (p=2.6e-21)</td>
</tr>
<tr>
<td></td>
<td>CNN-2</td>
<td>↑ (p=1.3e-55)</td>
<td>↑ (p=2.1e-24)</td>
</tr>
<tr>
<td></td>
<td>SpectraAE</td>
<td>↑ (p=7.4e-54)</td>
<td>↑ (p=1.7e-20)</td>
</tr>
<tr>
<td>LASSO</td>
<td>PtA</td>
<td>↓ (p=8.4e-55)</td>
<td>↓ (p=1.3e-11)</td>
</tr>
<tr>
<td></td>
<td>PLS</td>
<td>↓ (p=6.3e-06)</td>
<td>↑ (p=2.1e-07)</td>
</tr>
<tr>
<td></td>
<td>FitSim</td>
<td>≈ (p=2.3e-03)</td>
<td>↑ (p=1.6e-18)</td>
</tr>
<tr>
<td></td>
<td>CNN-1</td>
<td>≈ (p=8.8e-01)</td>
<td>↑ (p=3.3e-23)</td>
</tr>
<tr>
<td></td>
<td>CNN-2</td>
<td>≈ (p=4.6e-01)</td>
<td>↑ (p=6.8e-09)</td>
</tr>
<tr>
<td></td>
<td>SpectraAE</td>
<td>≈ (p=7.8e-01)</td>
<td>↑ (p=7.9e-18)</td>
</tr>
<tr>
<td>PLS</td>
<td>PtA</td>
<td>↓ (p=9.7e-47)</td>
<td>↓ (p=4.1e-14)</td>
</tr>
<tr>
<td></td>
<td>LASSO</td>
<td>↑ (p=6.3e-06)</td>
<td>↑ (p=2.1e-07)</td>
</tr>
<tr>
<td></td>
<td>FitSim</td>
<td>↑ (p=1.7e-09)</td>
<td>↑ (p=4.8e-08)</td>
</tr>
<tr>
<td></td>
<td>CNN-1</td>
<td>↑ (p=1.4e-06)</td>
<td>↑ (p=1.9e-16)</td>
</tr>
<tr>
<td></td>
<td>CNN-2</td>
<td>↑ (p=1.5e-05)</td>
<td>↑ (p=1.4e-07)</td>
</tr>
<tr>
<td></td>
<td>SpectraAE</td>
<td>↑ (p=7.6e-06)</td>
<td>↑ (p=1.7e-14)</td>
</tr>
<tr>
<td>FitSim</td>
<td>PtA</td>
<td>↓ (p=2.9e-54)</td>
<td>↓ (p=2.5e-16)</td>
</tr>
<tr>
<td></td>
<td>LASSO</td>
<td>≈ (p=2.3e-03)</td>
<td>↓ (p=1.6e-18)</td>
</tr>
<tr>
<td></td>
<td>PLS</td>
<td>↓ (p=1.7e-09)</td>
<td>↓ (p=4.8e-08)</td>
</tr>
<tr>
<td></td>
<td>CNN-1</td>
<td>≈ (p=4.1e-03)</td>
<td>↑ (p=2.7e-10)</td>
</tr>
<tr>
<td></td>
<td>CNN-2</td>
<td>≈ (p=6.7e-02)</td>
<td>≈ (p=2.2e-04)</td>
</tr>
<tr>
<td></td>
<td>SpectraAE</td>
<td>≈ (p=2.6e-02)</td>
<td>↑ (p=2.0e-07)</td>
</tr>
<tr>
<td>CNN-1</td>
<td>PtA</td>
<td>↓ (p=7.4e-54)</td>
<td>↓ (p=2.6e-21)</td>
</tr>
<tr>
<td></td>
<td>LASSO</td>
<td>≈ (p=8.8e-01)</td>
<td>↓ (p=3.3e-23)</td>
</tr>
<tr>
<td></td>
<td>PLS</td>
<td>↓ (p=1.4e-06)</td>
<td>↓ (p=1.9e-16)</td>
</tr>
<tr>
<td></td>
<td>FitSim</td>
<td>≈ (p=4.1e-03)</td>
<td>↓ (p=2.7e-10)</td>
</tr>
<tr>
<td></td>
<td>CNN-2</td>
<td>≈ (p=3.8e-01)</td>
<td>≈ (p=6.1e-02)</td>
</tr>
<tr>
<td></td>
<td>SpectraAE</td>
<td>≈ (p=6.8e-01)</td>
<td>≈ (p=3.5e-01)</td>
</tr>
<tr>
<td>CNN-2</td>
<td>PtA</td>
<td>↓ (p=7.4e-54)</td>
<td>↓ (p=2.1e-24)</td>
</tr>
<tr>
<td></td>
<td>LASSO</td>
<td>≈ (p=4.6e-01)</td>
<td>↓ (p=6.8e-09)</td>
</tr>
<tr>
<td></td>
<td>PLS</td>
<td>↓ (p=1.5e-05)</td>
<td>↓ (p=1.4e-07)</td>
</tr>
<tr>
<td></td>
<td>FitSim</td>
<td>≈ (p=6.7e-02)</td>
<td>≈ (p=2.2e-04)</td>
</tr>
<tr>
<td></td>
<td>CNN-1</td>
<td>≈ (p=3.8e-01)</td>
<td>≈ (p=6.1e-02)</td>
</tr>
<tr>
<td></td>
<td>CNN-2</td>
<td>≈ (p=6.3e-01)</td>
<td>≈ (p=1.1e-01)</td>
</tr>
<tr>
<td>SpectraAE</td>
<td>PtA</td>
<td>↓ (p=7.4e-54)</td>
<td>↓ (p=1.7e-20)</td>
</tr>
<tr>
<td></td>
<td>LASSO</td>
<td>≈ (p=7.8e-01)</td>
<td>↓ (p=7.9e-18)</td>
</tr>
<tr>
<td></td>
<td>PLS</td>
<td>↓ (p=7.6e-06)</td>
<td>↓ (p=1.7e-14)</td>
</tr>
<tr>
<td></td>
<td>FitSim</td>
<td>≈ (p=2.6e-02)</td>
<td>↓ (p=2.0e-07)</td>
</tr>
<tr>
<td></td>
<td>CNN-1</td>
<td>≈ (p=6.8e-01)</td>
<td>≈ (p=3.5e-01)</td>
</tr>
<tr>
<td></td>
<td>CNN-2</td>
<td>≈ (p=6.3e-01)</td>
<td>≈ (p=1.1e-01)</td>
</tr>
</tbody>
</table>
(a) Results of the lithium (Li) prediction task. (b) Results of the zirconium (Zr) prediction task.

**Figure 4.8:** Test-set results of (a) Li and (b) Zr prediction tasks. Each box gives the distribution of test-set scores (measured in RMSE, which is in the same units as the target variable) obtained by training each model 100 times (on different iterations of repeated cross-validation). Lower is better. PtA means “predict the average” (which is a naive baseline).

*How to read boxplots:* The top and bottom of the boxes are the upper and lower quartiles of the distribution with a grey line at the median and a black line at the mean. The whiskers extend from the box to show the range of the data.
Figure 4.9: Boxplots showing the distribution of RMSE differences between the model named in the subplot’s title versus the model labelled on the x-axis. (a-b) Prediction target is Li. (c-d) Prediction target is Zr.

How to read boxplots: The top and bottom of the boxes are the upper and lower quartiles of the distribution with a grey line at the median and a black line at the mean. The whiskers extend from the box to show the range of the data.
“↑” indicates that the model named under the first column (“Model 1”) achieved lower mean RMSE than “Model 2” and the result is statistically significant with the corresponding p-value listed in parentheses. A “↓” indicates the opposite model achieved lower mean RMSE. A “≈” indicates that we cannot reject the null hypothesis of equal averages, meaning that neither model significantly outperformed the other and we can conclude that there is no statistically significant difference between the two models’ scores. Table 4.1 reports that, for Li, the PLS model has the highest (worst) RMSE while all other models are on par with each other. For Zr, the three neural networks (CNN-1, CNN-2, and SpectraAE) are on par with each other, and all exhibit statistically significant reductions in mean RMSE compared to LASSO and PLS.

Overall, the best model, or set of models, is determined by counting how many times the model(s) show statistically significant improvement over the other models (this follows the approach of other work [49, 201]). As discussed in Section 4.5.4, 32 of 48 elements are reported because at least one model beats PtA for these elements. Results for these elements and all pairs of models are given in appendix Section B.1, Table B.2. The results show that a baseline (PLS or LASSO) is best, or tied for best, on 8 elements (Ba, Bi, Hf, Li, Mg, Mn, Th, and Ti). FitSim is best on In and on par with CNN-2 and SpectraAE on Na. One or both CNN’s (CNN-1 or CNN-2) are best on 10 elements: Ag, Ca, Co, Mo, Rb, S, Sb, Ta, Y, and Zn. Finally, SpectraAE is best on Sn and tied for best with either CNN-1 or CNN-2 on the remaining 11 elements: Al, As, Be, Cs, Fe, Ga, K, Pb, Sr, V, and Zr.

The model that performed best varied depending on the element. In practice, it may be sub-optimal to deploy a single model for all elements, and instead deploy a different model for each target element of interest. Why some elements perform well for some models and not others remains to be investigated.

4.6.2 Comparison to AutoSim

In Figure 4.10, a comparison is made between SpectraAE from this chapter to AutoSim, which was developed in the previous chapter. This comparison uses the 10-fold cross-validation results from the previous chapter. SpectraAE achieves lower RMSE than AutoSim on the Li and Zr prediction tasks. These results indicate, for this dataset, that the integration of domain knowledge in AutoSim may not be necessary, as the learning-based model demonstrates lower prediction error.

4.6.3 Spectrum Reconstruction

SpectraAE reconstructions are shown in Figure 4.11. As intended with the decoder design, the reconstructed spectra exhibit desired properties: they are smooth, high-
Figure 4.10: Comparison of AutoSim (Chapter 3) to SpectraAE (Chapter 4). Results use the 10-fold cross-validation scheme from Chapter 3.
(a & c): Each box gives the distribution of RMSEs from 10-fold cross-validation (from each fold’s test set). Lower is better.
(b & d): The boxplots show the distribution of differences between the RMSE of SpectraAE versus the RMSE of AutoSim from corresponding iterations of cross-validation. A negative difference indicates that SpectraAE’s RMSE is lower.
How to read boxplots: The top and bottom of the boxes are the upper and lower quartiles of the distribution with a grey line at the median and a black line at the mean. The whiskers extend from the box to show the range of the data.
frequency noise is ignored, and important peaks, like the primary Fe K-\(L_3\) (K\(\alpha\)) line (the tallest peak in the figure) and its secondary line, are fitting well. Unlike traditional neural networks, auto-encoders (such as AutoSim and SpectraAE) are able to generate data; this may be utilized to improve prediction accuracy of traditional models or to improve our understanding of a trained AE. The spectra generated by an AE may be used to gain insight into what the auto-encoder has learned, as a way of checking whether the model “understands” the input spectra. If an AE fails to reconstruct a spectrum, this may be used as a sign that its predictions should be treated with low confidence. Additionally, a trained AE may be used to generate new, synthetic, labelled samples. Synthetic data can be used to augment the dataset for training traditional supervised models, such as PLS or another neural network.

4.6.4 Other Types of Spectroscopy

The decoder in SpectraAE is not specific to XRF. While the decoder’s design reflects how XRF spectra are generated, it may also reflect how other types of spectroscopy generate spectra too, such as near-infrared (NIR) spectra. In NIR, a spectrum may be deconvolved into peaks superimposed on a baseline continuum \[123\]. The function favoured by researchers to approximate the peak has changed over time. For instance, the peak function was initially modelled as a Gaussian and then later replaced by a modified Gaussian model \[223\]. The decoder’s convolutional layers should be able to learn the modified Gaussian distribution, or any distribution that fits the data.

4.7 Conclusion

SpectraAE is designed using general domain knowledge of spectra, applicable to any type of spectra structured with peaks. A statistically significant decrease in prediction error was found using SpectraAE for 1 (Sn) out of 32 prediction targets. Reconstructions from SpectraAE show that it is able to compress XRF spectra, with a resolution of 1024, down to 48 real-valued numbers and can reconstruct the most important parts of the original spectra. The two CNN models developed were the best-performing models on the majority of elements, either outperforming or on par with the auto-encoder. SpectraAE outperformed AutoSim on Li and Zr.

Since labelling data is expensive, practical scenarios may have an abundance of spectra data without ground truth. Standard PLS and neural network models do not have a mechanism to use unlabelled data, but auto-encoder neural networks, using semi-supervised learning, can learn from both labelled and unlabelled data. Although the supervised models outperformed the semi-supervised approach on
**Figure 4.11:** Example reconstructions from SpectraAE in red and observed XRF spectra in blue. The x-axis is the channel index.
a majority of elements, it is worth emphasizing the potential of semi-supervised learning. This dataset contains 314 unlabelled samples; many more samples may improve the results.
Chapter 5

Hyperparameter Optimization

Hyperparameters of the neural network architectures developed in the preceding chapters were tuned through a combination of trial-and-error, intuition based on experience, and multiple stages of grid search over the parameter space. Automatic methods of hyperparameter optimization (HPO) [113] may obtain better hyperparameter configurations, but this is difficult when datasets are small because smaller datasets produce less accurate and noisier validation-set scores. For instance, by the law of large numbers, the mean of a small set of samples has higher variance than the mean of a large set of samples. The smaller the validation set is, the noisier the validation-set error will be. Also, the smaller the training set is when training a model, the more likely it is that the model will overfit, which results in higher validation-set error. A larger dataset would help alleviate the high variance.

We surveyed chemometrics literature for a larger spectra dataset where the goal was regression, not classification, featuring a real-valued target, and specifically seeking those with published neural network results to be used for comparison. Of the first 30 datasets found, 16 were regression problems with one or more real-valued targets. Of these, 7 are publicly available for download [9, 73, 74, 114, 188, 230, 255]. Six of them have less than 1,000 samples. And one, a dataset of mangoes, has around 11,000 samples. We chose the largest one which also has previously-published results [10, 11, 160].

With this larger dataset as a benchmark, this chapter addresses the issue of noisy validation-set scores due to neural network initialization. A further issue arises from this dataset due to the seasonal nature of growing mangoes, which is taken into account by incorporating extra domain knowledge (i.e., about the season and order in which the mangoes were grown). The work in this chapter was published [68] in the Chemometrics and Intelligent Laboratory Systems journal.
5.1 Introduction

This chapter considers how to automatically configure neural network hyperparameters such that it extrapolates for visible and near-infrared (Vis-NIR) spectroscopy. Specifically, the aim is to further streamline the process of automatic hyperparameter optimization (HPO) to produce a neural network model that generalizes, or extrapolates, well.

Two challenges in HPO are considered:

1. Overfitting to the validation set in the same way that neural network training may overfit the calibration set\(^1\) [50, 113]; this is important when extrapolating (whether to new batches, across time, or beyond the range of the calibration data) [36, 244].

2. Variance of prediction error under different random initializations of the neural network [66, 263]; this is more of an issue when datasets are small.

This study jointly considers the problem of automated HPO (from the field of machine learning) with the problem of extrapolation and robustness (from the field of chemometrics). To avoid overfitting in HPO, a combination of expert intuition [187], hand-tuning [251], and grid search [160] is most often used while automated methods are an open area of research [113].

For a dataset with multiple seasons, where the test set is a future season, our hypothesis is that a temporal relation exists between seasons and that validating on the latest samples will extrapolate better to future seasons. To test this hypothesis, we compare several choices of partitioning the data into calibration and validation sets to use in HPO. The validation sets we test are: a random 33\% (as in previous work), a semantically meaningful subset [244] (specifically, the latest season), and the first, middle, and last 33\% of samples. To address the second HPO challenge, we investigate using ensembles to reduce the variance of validation-set error during HPO; this is tested in conjunction with partitioning choices. We conduct HPO for each choice of validation set and compare HPO with and without ensemble averaging. The results in this study shed light on reproducible and automated practices for configuring and training neural networks for spectroscopy in scenarios where extrapolation, to future samples or new batches, is desired.

---

\(^1\)In machine learning, the data used to train a model is called a “training set” but in this chapter we use the term “calibration set” which is common in chemometrics.
5.2 Materials and Methods

5.2.1 Dataset

The dataset of mangoes by Anderson et al \[10, 11\] is a good example of needing to extrapolate to a new batch, in this case, a future harvest season. A harvest season refers to the time when crops are ripe and ready to be harvested. Visible and near-infrared (Vis-NIR) spectra of mango fruit from four harvest seasons (2015, 2016, 2017, and 2018) were collected. Using spectra from 3 years, the goal is to make predictions of samples in the 4th year. The dataset is publicly available \[9\]. The spectral bands range 300 – 1100 nm with approximately 3.3 nm intervals \[10\]. Near-infrared spectroscopy allows for non-invasive assessment of fruit quality. In this case, the prediction target is the percent of dry matter (DM) content. DM % is an index of total carbohydrates which is an indicator of quality in mango fruit \[10\].

Mishra and Passos \[160\] make a number of modifications to the mango fruit dataset (available online\(^2\)), specifically: (1) only a subset (684–990 nm, 3.3 nm intervals) of the available spectral bands are used, (2) outliers have been removed from non-test-set samples, (3) chemometric pre-processing techniques were applied and concatenated together, and (4) each feature is standardized separately. Standardization of a distribution entails subtracting each value by the mean of the distribution and then dividing it by the standard deviation of the distribution. Each sample in the dataset consists of DM %, as the target to predict, and the concatenation of 6 vectors (each with 103 elements) which are:

1. The raw spectrum.
2. The first derivative of the smoothed spectrum (smoothing uses a Savitzky–Golay filter with a window size of 13).
3. The second derivative of the smoothed spectrum.
4. Standardized spectrum (Standard Normal Variate, or SNV, of the spectrum).
5. The first derivative of the smoothed SNV spectrum.
6. The second derivative of the smoothed SNV spectrum.

We chose this version of the dataset because (a) it allows direct comparison to the model by Mishra and Passos \[160\], which is the state-of-the-art on this dataset, (b) it enables us to reproduce their model, and (c) it allows us to draw conclusions

about the HPO technique without worrying about whether any observed differences were caused by a difference in pre-processing.

5.2.2 Baseline Neural Network

The state-of-the-art prediction model for this dataset is a convolutional neural network (CNN), model “B” by Mishra and Passos [160], which we’ll refer to as CNN\(_B\). This model will serve as the baseline to compare our results. Its hyperparameters were optimized through a combination of multiple stages of grid search, expert intuition based on experience, and a careful analysis of overfitting. For details, readers are referred to the original paper [160] but we summarize the architecture and main hyperparameters here. The CNN\(_B\) architecture consists of a convolutional layer (1 kernel of width 21, stride 1) followed by three fully-connected layers (of size 36, 18, and 12) with exponential linear unit (ELU) activations. A final fully-connected layer with no activation produces the output (DM %). Kernel and fully-connected weights are initialized by the He normal initialization method and regularized with an \(L_2\) with a coefficient of 0.0055. Training proceeds for 750 epochs with mini-batches of size 128 and early stops when validation loss stops improving for 50 epochs. Learning rate (LR) starts at 0.005 and halves each time validation loss stops improving for 25 epochs until the minimum LR is reached (1e-6). The weights of the neural network are optimized by stochastic gradient descent using the ADAM algorithm.

Figure 5.1: Neural network architecture and hyperparameter search space. Hyperparameters are shown in blue text.
5.2.3 Repeatability

Repeatability has been highlighted as an important issue in machine learning in general as well as for spectroscopic data analysis:

“Differences between two runs under identical hyperparameters might still occur. Repeated experiments should be conducted to evaluate the model performance.” [251]

“Model repeatability and interpretation capability are critical for the further development of deep learning based spectral analysis.” [263]

Since training is stochastic (weights are initialized randomly and mini-batching randomly shuffles the data in between epochs), the final weights of the network will be different after each training run. We report the distribution of the errors given by randomly initializing and re-training, which is fairer than reporting a single sample from the distribution of errors; this improves reproducibility and reveals more about the neural network’s performance [50].

5.2.4 Hyperparameter Search Space

Neural networks can take on many different architectures (such as the number of layers), each with many possible hyperparameters (such as learning rate). In this study, architecture settings are treated as additional hyperparameters and will be optimized in conjunction with other hyperparameters [261]. Any specific assignment of all the hyperparameters is referred to as a configuration.

The space of possible architectures used in our hyperparameter search is based on typical architectures used in spectroscopy applications [2, 52, 60, 83, 148, 151, 251, 263] and CNNB [160] in particular. These neural network architectures all follow a similar pattern: one or more convolutional layers followed by one or more fully-connected layers, with an activation function after each layer. A final fully-connected layer with no activation produces the output (DM %). Our architecture and hyperparameter search space is visualized in Figure 5.1. The particular architecture and search space we use is somewhat arbitrary because the goal of the experiments is to test the HPO method under different conditions.

The search space consists of 5 hyperparameters, and we deliberately structure them to leave out conditional hyperparameters to keep it simple. More hyperparameters may also be included, with the potential of increasing prediction accuracy, but would require more computation and for the purpose of testing HPO, it is not necessary. The hyperparameters and their ranges are listed in Table 5.1 (the ranges are based on published literature and standard practice). All fully-connected layers, other than the first one, are half the size of the previous layer; this keeps the
search space smaller and simpler. For example, when there are 3 fully-connected layers and the first layer has 36 units (as in CNN$_b$), then the second and third layers will have 18 and 9 units. Search space for the convolution kernel’s width is approximately logarithmic and rounded to the nearest odd number, ranging from 3 to 29. All hyperparameter values are integers except $L_2$ regularization which is a floating-point number.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_2$ coefficient</td>
<td>$[10^{-4}, 1]$</td>
</tr>
<tr>
<td>Number of kernels</td>
<td>$[1, 13]$</td>
</tr>
<tr>
<td>Kernel width</td>
<td>3, 5, 7, 11, 15, 21, 29</td>
</tr>
<tr>
<td>Number of FC layers</td>
<td>$[1, 4]$</td>
</tr>
<tr>
<td>Number of FC units</td>
<td>$[4, 96]$ (step 4)</td>
</tr>
</tbody>
</table>

Table 5.1: Hyperparameters’ search space.

### 5.2.5 Automatic Hyperparameter Optimization

In the field of automated machine learning (Auto-ML), the state-of-the-art for automatically searching through a hyperparameter search space is Bayesian optimization. Bayesian optimization (BO) for HPO [29, 185, 206] chooses which hyperparameter configuration to try next. The first trial configuration is sampled randomly. Subsequent configurations are selected based on a mix of how well they are estimated to perform and how uncertain the estimate is. We use an implementation of Bayesian optimization called hyperopt [28].

Pseudo-code in Figure 5.2, line 5, shows the general idea: select a hyperparameter configuration (config) to try next via Bayesian optimization (line 6), train a neural network using that configuration (line 7), obtain a score (RMSE) for this configuration (line 8) using predictions on a validation set (validation set described in next section), and, finally, choose the configuration with the lowest RMSE (line 9). We stop HPO after completing approximately 2000 configurations.

We follow standard practice in automatic hyperparameter optimization (HPO); readers are referred to a tutorial on HPO for spectral modelling to learn how to perform similar experiments from scratch [187]. Even when using state-of-the-art Bayesian optimization software, many decisions must be made manually, including deciding on a search space, objective function, and validation set.
function \texttt{train}(\texttt{config, }D_{\text{split}}) \\
\text{NN} = \text{Setup NN with } \texttt{config} \text{ and initialize weights randomly.} \\
\text{Train NN weights on } D_{\text{cal}}, \text{ with ES and LR schedule based on } D_{\text{val}}. \\
\text{return NN} \\
function \texttt{HPO}(D_{\text{split}}) \\
\text{for } \texttt{config} \text{ in configurations do} \\
\text{\hspace{2em}NN} = \texttt{train}(\texttt{config, }D_{\text{split}}) \\
\text{\hspace{2em}RMSE}_{\text{config}} = \texttt{NN.predict}(D_{\text{val}}) \\
\text{return } \texttt{config} \text{ with lowest RMSE} \\
function \texttt{final.evaluation}(D_{\text{split}}) \\
\text{best}_\text{config} = \texttt{HPO}(D_{\text{split}}) \\
\text{\texttt{final.NN} = train(best}_\text{config}, D_{\text{rand}}) \\
\text{RMSE} = \texttt{final.NN.predict}(D_{\text{test}}) \\
\text{return RMSE} \\

text

\textbf{Figure 5.2:} Experiment overview (without ensembling). Pseudo-code of neural network training (\texttt{train} function) and hyperparameter optimization (\texttt{HPO} function) showing how the sets in each partitioning choice are used and how RMSE is calculated in \texttt{HPO} and \texttt{final.evaluation}. \(D_{\text{split}}\) is the dataset that has been partitioned (\texttt{split}) in some way (described in Section 5.2.6). \(D_{\text{split}}\) is the validation set and \(D_{\text{split}}\) is the calibration set for this \texttt{split}.

5.2.6 Choice of Validation Set

Pseudo-code in Figure 5.2 shows where the calibration, validation, and test sets are used in training and HPO. The test set, \(D_{\text{test}}\), is used in the final evaluation to score how well HPO performed (line 13). As in previous works [10, 11, 160], the 2018 harvest season is used as the test set for evaluation. For a predictive model in the fruit industry to be practical, it must be robust to year-to-year variations. Since the hope is that the model can extrapolate to a new season, a whole season is the right choice for the test set [244]. A calibration set (\texttt{cal}) is used for training the neural network weights (line 3) and a validation set (\texttt{val}) is used to inform the training procedure when to early-stop (ES) and when to drop the learning rate (LR).\(^3\) The

\(^3\)The epoch at which early stopping occurs and the LR schedule (epochs at which to drop the LR) could instead be learned hyperparameters, but standard practice is to use the validation set to dynamically adapt as the training proceeds. We adopted this because we are only testing HPO with a limited set of hyperparameters.
validation set (val) is also used in HPO to score configurations (line 8).

We consider a number of ways to partition harvest seasons 2015, 2016, and 2017 (i.e., all data other than the test set) into calibration and validation sets for HPO. Note that in our experiments, the partitioning of the data (split) will differ between training in HPO and training in final evaluation. In previous work [160] (and standard practice), the validation set is 33% of the non-test data, randomly sampled without replacement (which we’ll call D\text{val rand}), and the calibration set (D\text{cal rand}) is the remaining 67%. Samples can also be chosen more deliberately (e.g., grouping samples according to the target variable [35]), keeping in mind the goal to extrapolate:

1. $\mathcal{D}_{2017}$: First we select an entire season to force the model to extrapolate to a future season, just like we expect it to do for the test set. Since the test set is from 2018, we use the 2017 harvest season samples (harvest seasons do not follow the calendar year but samples are labelled with their season in the dataset [9]). The 2017 harvest season samples comprise 49% of non-test data which is considerably more validation samples than before.

2. $\mathcal{D}_3$: Secondly, we partition the data to have the same size (33% in validation) as in the partitioning used in previous work, $\mathcal{D}_{\text{rand}}$. Since $\mathcal{D}_{\text{rand}}$ is used in final model training and evaluation (see line 12), matching the size may perform better because many hyperparameters are sensitive to dataset size.

3. $\mathcal{D}_1, \mathcal{D}_2$: To test our hypothesis that there is a temporal relation between harvest seasons, we test using the first and middle 33% of samples as well ($\mathcal{D}_1$ and $\mathcal{D}_2$ respectively). If these batches are independent, we would expect them to perform similarly.

Partitioning choices, and the sizes of each set, are summarized in Table 5.2. Figure 5.3 further visualizes these choices by showing the date and DM % of each

<table>
<thead>
<tr>
<th>Cal. size</th>
<th>Val. size</th>
<th>Validation set is...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{D}_{\text{rand}}$</td>
<td>6642</td>
<td>3272</td>
</tr>
<tr>
<td>$\mathcal{D}_{2017}$</td>
<td>5045</td>
<td>4869</td>
</tr>
<tr>
<td>$\mathcal{D}_1$</td>
<td>6642</td>
<td>3272</td>
</tr>
<tr>
<td>$\mathcal{D}_2$</td>
<td>6642</td>
<td>3272</td>
</tr>
<tr>
<td>$\mathcal{D}_3$</td>
<td>6642</td>
<td>3272</td>
</tr>
</tbody>
</table>

Table 5.2: Partitioning choices and number of data points in validation set and corresponding calibration set. Test set is fixed with a size of 1448.
Figure 5.3: Calibration, validation, and test sets for each partitioning of the dataset: $D_{rand}$, $D_{2017}$, $D_1$, $D_2$, and $D_3$. Each point is a sample from the dataset coloured by assignment to calibration, validation, or test set. DM % is plotted on the y-axis with the date the sample was collected on the x-axis.
sample in each set. The final_evaluation function (in pseudo-code Figure 5.2) is run on each partitioning choice. That is, $D_{split}$ is one of $D_{rand}$, $D_{2017}$, $D_1$, $D_2$, or $D_3$. Again, we emphasize, for the purpose of comparing HPO methods, regardless of which $D_{split}$ is used for HPO (line 11), the final model configurations chosen in HPO always train using $D_{rand}$ in the final_evaluation (line 12). Thus, any observed differences will be due to HPO, and not due to training on different data.

5.2.7 Neural Network Ensemble Averaging

Neural networks are trained by minimizing an objective function, but there can be many sub-optimal weight settings after training depending on a number of factors [251] including how the weights are initialized and the order of samples seen during training. As we show in the results, a neural network’s prediction score varies under different random initializations (repeating the train function in pseudo-code Figure 5.2 results in neural networks with different weights and different prediction errors). This is problematic for HPO because the optimizer is given only a weak signal to guide it. As datasets get bigger, the variance is reduced [66, 263], but when that’s unavailable, as is often the case, another method is to use ensembles [194].

An ensemble is a set of models. The predictions of the ensemble’s models are combined to produce a final prediction. By aggregating the outputs of different models, ensembles can help mitigate the risk of overfitting [192]. There are different ensembling strategies for choosing models and for combining their predictions [66, 194, 259].

A straightforward, yet powerful [45], strategy is to train a neural network multiple times with different random seeds (random initializations) then average their predictions [172, 183], which is the approach used in this chapter (sometimes called ensemble averaging over initial conditions). More sophisticated ensemble methods exist [194], but recent literature suggests they don’t live up to expectations [13]. A resurgence of recent studies, under the name Deep Ensembles [132], continues to demonstrate the effectiveness of this approach on modern deep neural networks; optimization from random initializations more-widely explores the loss landscape leading to diverse ensemble members and, therefore, improved accuracy and out-of-distribution robustness [90].

5.2.8 Experiment

We conduct hyperparameter optimization (HPO) for 8 scenarios. First we compare HPO with and without ensembling using $D_{rand}$, $D_{2017}$, and $D_3$ (6 scenarios). Then,
we also test HPO on $\mathcal{D}_1$ and $\mathcal{D}_2$ but only with ensembling; this is to test whether any difference is observed as compared to $\mathcal{D}_3$. The final model configurations from each HPO scenario are compared.

**HPO With and Without Ensembling**

HPO without ensembling (referred to as *singles-HPO*) follows the process laid out in the pseudo-code (Figure 5.2). HPO with ensembling (referred to as *ensembles-HPO*) creates an ensemble for each configuration. In Figure 5.2, line 7 is repeated a number of times to make the ensemble’s models. Each model in the ensemble makes predictions on the validation set, as in line 8. The predictions from the models are averaged, and RMSE for the ensemble is calculated using the averaged predictions.

**Final Models With and Without Ensembling**

For each HPO scenario, the final configuration from HPO (Figure 5.2, line 11) is evaluated. We evaluate the baseline (CNN$_B$) configuration and final configurations from *singles-HPO* scenarios both as single models and as ensemble models. Since *ensembles-HPO* scenarios already utilize ensembles, we evaluate their final configurations as ensemble models only.

For single models, the final model (final NN, line 12) is obtained by training on the original partitioning of the dataset ($\mathcal{D}_{\text{rand}}$). The final model makes predictions on the test set ($\mathcal{D}_{\text{test}}$). Then a distribution of RMSE scores is generated by repeatedly training and testing (repeating lines 12-13).

For ensemble models, an ensemble is created by re-training (repeating line 12), then the final RMSE is calculated using the average of the ensemble’s models’ predictions on the test set. A distribution of RMSE scores is again generated, this time by re-training the entire ensemble a number of times.

**Compute Hardware and Software**

Source code used to train the neural networks is available from [https://github.com/skylogic004/spectroscopy-neural-network](https://github.com/skylogic004/spectroscopy-neural-network) and requires TensorFlow 2.6 and Python 3.9. Experiments are run on a high-performance compute cluster named Cedar, hosted by Simon Fraser University for Compute Canada and Digital Research Alliance of Canada, with NVIDIA V100 and P100 GPUs and Intel Xeon E5-2650 2.2GHz and Intel Xeon Silver 4216 2.1GHz CPUs (1352 GPU devices and 94528 CPU cores).

1 GPU and 6 CPUs were utilized per trial, but many were run in parallel. A total
of 177.1 GPU-days were utilized in conducting the 8 HPO experiments. A user wishing to deploy HPO only has to conduct HPO once, unlike our experiments, in which case the expected compute times are listed in Table 5.3. A user may also reduce the number of HPO trials by stopping when satisfactory performance is achieved; for instance, we observed that 500 to 1000 trials, depending on the scenario, achieved similar validation set scores to the best score after 2000 trials.

<table>
<thead>
<tr>
<th></th>
<th>Time per model</th>
<th>Number of models</th>
<th>Total (time per scenario)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPO</td>
<td>0.25</td>
<td>~2000</td>
<td>533</td>
</tr>
<tr>
<td>final_evaluation</td>
<td>0.45</td>
<td>50</td>
<td>22.5</td>
</tr>
</tbody>
</table>

Table 5.3: Average compute times of ensembles-HPO scenarios in GPU-hours. In HPO, number of models is the number of trials; in evaluation, it is the number of times the model is re-trained.

5.3 Results and Discussion

Experimental results are laid out in Figure 5.4; we explain each panel in-turn in the following sections. First we report the distributions of single models (Sections 5.3.1 and 5.3.2, panels A and B), then we report the same model configurations but evaluated as ensembles (Sections 5.3.3 and 5.3.4, panels C and D). In Section 5.3.5 and panel E, we report the results of HPO scenarios that use ensembles within HPO. Lastly, Section 5.3.8 describes a lower-bound, shown in panel F. Post-hoc analysis in Section 5.3.6 describes how well validation-set error correlates to test-set error. And, finally, Section 5.3.7 discusses the efficacy of HPO as compared to an expert. The best hyperparameter values from each HPO experiment are listed in Table 5.4. It is the combination of hyperparameter values that perform well; hyperparameters are not independent.

5.3.1 CNN\textsubscript{B} Variance

First, we obtain the distribution of error scores of the baseline, CNN\textsubscript{B}, by training 2000 times with random initializations. Scores are measured in RMSE, whose units match the prediction target: in this case, the percent of dry matter (DM) content. The distribution is shown in Figure 5.4 in panel A; it has a mean of 0.843, 95% CI [0.789, 0.896]. This means that if this model were deployed, we would expect the prediction error to be between 0.789 and 0.896 95% of the time. Mishra & Passos [160] reported 0.79 for CNN\textsubscript{B}. One can reproduce a particular training
Figure 5.4: Boxplot where each box is a distribution over RMSE scores. The results are divided into 6 panels (labelled A–F). A is explained in Section 5.3.1, B in 5.3.2, C in 5.3.3, D in 5.3.4, E in 5.3.5, and F in 5.3.8. The two groups labelled at the bottom of the figure—singles and ensembles—indicates whether the distribution shown is of single models or of ensemble models. CNN is the baseline model and CNN is the baseline model after applying ensembling. The groups labelled singles-HPO and ensembles-HPO indicates whether HPO evaluates each trial with a single model (as in standard HPO) or with an ensemble. The “lower bound” gives an optimistic idea of what the best-possible model might achieve. The legend—with names starting with D—lists the five choices of partitioning the data into calibration and validation sets to use in HPO.

How to read boxplots: The top and bottom of the boxes are the upper and lower quartiles of the distribution with a line at the median. The whiskers extend from the box to show the range of the data and the points beyond the whiskers are considered outliers.
run by seeding the random functions, but this only holds as long as nothing else changes—if you change the training data to include the test set (as you would in preparation for deployment), the resulting weights will again converge to another point in the distribution.

### 5.3.2 Singles-HPO Evaluations Without Ensembling

Figure 5.4, panel B, shows the results of the first set of hyperparameter optimization runs, singles-HPO, using $D_{\text{rand}}$, $D_{2017}$, and $D_{3}$. After obtaining the best neural network configuration from HPO (line 11 of Figure 5.2), the neural network architecture is trained on the original ($D_{\text{rand}}$) partitioning of the dataset (line 12) and evaluated on the test set (line 13); lines 12-13 are repeated 2000 times to obtain a distribution of RMSE scores. The resulting distributions are visualized in the boxplot (Figure 5.4, panel B).

HPO on $D_{\text{rand}}$ was least successful (RMSE of 1.01 on average); $D_{3}$ and $D_{2017}$ fared better with a mean around 0.88. This supports the hypothesis that HPO overfits to $D_{\text{val}}$ and that forcing HPO to extrapolate—by validating on samples in $D_{2017}$ or $D_{3}$—results in a model that is better able to extrapolate to the test set. $D_{3}$ and $D_{2017}$ HPO scenarios performed similarly to each other, except $D_{2017}$ had higher variance. Next, we look at using ensembles to reduce variance.

### 5.3.3 CNN$_B$ Evaluation With Ensembling

Before doing ensemble experiments, we need to decide how many models to include in the ensemble. Since each member of the ensemble must be trained, more computation is required for each additional model, though they are readily parallelizable if parallel computing resources are available. The number of members in the ensemble can be chosen based on the trade-off between accuracy and available computation. Using the baseline, CNN$_B$, we compute the distribution of scores for single models ($N = 1$) up to ensembles with 500 member models ($N = 500$), shown in Figure 5.5. The $N = 1$ distribution is also the distribution for CNN$_B$ reported in Figure 5.4, panel A.
Figure 5.5: RMSE of test set using CNN_B model as a function of ensemble size. Size N = 1 (purple) indicates the distribution of RMSE scores over 2000 single re-trainings (with random initialization) of the model. Each boxplot (where N > 1) is a distribution over 200 simulated ensembles, where each ensemble is formed from a unique combination of trained models (from a pre-computed set of 2000). We use ensembles with 40 member models in our experiments (green).
The distributions in Figure 5.5 are computed as follows. We first trained 2000 single models using the baseline configuration (CNN_B). Then we simulated ensembles of every size, N, from the pre-computed set of 2000. Each ensemble is a random, but unique, combination of N models. We generate 200 ensembles for each N and plot the distribution of RMSE scores.

From Figure 5.5 we observe a sharp drop in variance and test-set error as soon as N > 1. Variance continues to decrease with greater N but tapers off. The median error also drops quickly at first, but quickly converges. We use 40 ensemble members in our experiments which balances the range of error (we deemed 0.02 from min to max as sufficient) while keeping computation reasonable based on our available compute resources. An ensemble of size 40 improves the mean error to 0.790, 95% CI [0.782, 0.799]; we refer to this model as CNN_B^ensemble. We compare the results that follow to CNN_B^ensemble, whose distribution of errors is shown in Figure 5.4, panel C.

### 5.3.4 Singles-HPO Evaluations With Ensembling

Since ensembling CNN_B worked well, we also test the best configurations from each singles-HPO scenario as ensemble models. In the final_evaluation function (Figure 5.2), the best hyperparameter configuration from HPO (line 11) is re-trained 40 times (repeat line 12 40 times). Predictions from the 40 models are averaged to make the ensemble.

Prediction error from ensembles also vary due to randomness, so we repeat the process to obtain and report the distribution of RMSE scores. Using the 2000 models from Section 5.3.2, we group them into groups of 40 models which makes for 50 ensembles (50 × 40 is 2000). The resulting distributions are shown in Figure 5.4, panel D. The results show that the variance has tightened substantially and the error on average has also dropped significantly. However, CNN_B^ensemble remains better than all of these; the next experiment is to see if ensembling within hyperparameter optimization leads to an improvement.

### 5.3.5 Ensembles-HPO Evaluations

The second set of hyperparameter experiments use ensembles for each configuration in HPO and to evaluate the final configurations. For each configuration in HPO (see HPO function, line 6), neural network training is repeated 40 times (repeat line 7). Then, on line 8, the ensemble’s RMSE is calculated. The final_evaluation function proceeds with ensembling as well, as described in the previous section (Section 5.3.4). The resulting distributions of 50 ensemble models are visualized in the boxplot (Figure 5.4, panel E).
All the partitioning choices we tested do better than $D_{\text{rand}}$, but some do better than others.

**Matched calibration-set size:** $D_3$ and $D_{2017}$ are conceptually similar ($D_3$ uses the latest 33% while $D_{2017}$ uses the latest 49%), but $D_3$’s results are significantly better. The difference in prediction error may be due to the difference in size of the calibration sets. Sizes are listed in Table 5.2. The size of the calibration set is important because: (a) While training networks during HPO, networks trained using $D_3^{\text{cal}}$ see more data than when using $D_{2017}^{\text{cal}}$, so these networks perform better on the validation set which leads HPO to a better final configuration. (b) Many hyperparameters are sensitive to dataset size, so matching the sizes between HPO and final evaluation alleviates this.

**Best result:** The best ensembles-HPO result was obtained using the $D_3$ partitioning and results in a mean RMSE of 0.796, 95% CI [0.784, 0.809]. This distribution closely overlaps with the CNN$_B$ ensemble distribution. This suggests that CNN$_B$ is already very good and, indeed, Mishra & Passos [160] did an investigation into finding a model that would generalize; ensembling this model makes it even better. The combination of $D_3$ with ensembles-HPO achieved competitive performance to CNN$_B$ ensemble and this is achieved completely automatically without expert guidance.

**Temporally related versus independent:** If the batches (seasons) were independent, we would expect $D_1$, $D_2$, and $D_3$ results to be similar. Instead, the results are quite different, which is evidence that the batches are not independent. The first difference in results is in the hyperparameter values, which differ considerably between $D_1$, $D_2$, and $D_3$ (shown in Table 5.4). The second difference in results is in the test-set RMSE distributions in Figure 5.4 which shows $D_1$ is worse than $D_2$ and $D_2$ is worse than $D_3$.

Evidence of dependence may be because the distribution of DM % values is similar between the test set (which is the 2018 harvest season) and the last set used for validation (which is the 2017 harvest season) [252]. The similarity may be random or due to temporal structure. If 2017 was coincidentally very similar to 2018 (for example, in terms of climate or soil properties) then we would expect calibration sets containing 2017 season samples to lead to models that do well on the test set, and validation sets containing 2017 samples should estimate test-set error well during HPO. However:

- $D_{\text{rand}}$ and $D_1$ gave the two worst results, yet both contain many samples from 2017 in calibration.

- $D_{\text{rand}}$ and $D_{2017}$ did poorly, yet both contain many samples from 2017 in validation.
Instead, we find that $D_2$ and $D_3$ work the best. The possibility of a temporal relation should be further tested on other datasets.

**HPO with and without ensembling:** Prediction error using $D_3$ is greatly improved in *ensembles-HPO* as compared to *singles-HPO*, this may be explained by the reduced variance. Since ensembles reduce variance, the ensemble members can individually exhibit larger variance than single models would allow. That is, individual networks are given more leniency to overfit (or be less regularized) because the ensemble reduces the variance while keeping the error low. HPO then picks a configuration whose *ensemble* error is optimized, rather than the error of an individual model. $D_{rand}$ and $D_{2017}$ did not improve, however. For $D_{rand}$, the reason may be that increased overfitting is detrimental given that it already exhibits overfitting. For $D_{2017}$, the reason may be its smaller calibration set which also worsens overfitting. The main thing to note here is that ensembling alone is not sufficient to improve HPO; the choice of validation set also matters.

### Table 5.4: The best hyperparameter values found in each HPO experiment.

<table>
<thead>
<tr>
<th>HPO scenario</th>
<th>$D_{split}$</th>
<th>$L_2$ coefficient</th>
<th># of kernels</th>
<th>Kernel width</th>
<th># of FC layers</th>
<th># of FC units (in 1st layer)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Singles-HPO</td>
<td>$D_{rand}$</td>
<td>5.4391e-4</td>
<td>7</td>
<td>29</td>
<td>3</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td>$D_{2017}$</td>
<td>1.5369e-2</td>
<td>10</td>
<td>11</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>$D_3$</td>
<td>1.7251e-2</td>
<td>1</td>
<td>7</td>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>Ensembles-HPO</td>
<td>$D_{rand}$</td>
<td>3.3921e-4</td>
<td>13</td>
<td>29</td>
<td>4</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>$D_{2017}$</td>
<td>1.5128e-2</td>
<td>10</td>
<td>11</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>$D_1$</td>
<td>1.9894e-4</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td>$D_2$</td>
<td>7.3074e-3</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>$D_3$</td>
<td>2.9497e-3</td>
<td>1</td>
<td>15</td>
<td>4</td>
<td>16</td>
</tr>
</tbody>
</table>

5.3.6 Generalizability of Validation Set

Hyperparameters are chosen to minimize validation error, with hopes that the hyperparameter configuration also minimizes test set error. This assumes that validation-set error is correlated to test-set error [50]; here we check whether this is achieved.

Figure 5.6 shows validation-set and test-set error for all configurations tried during HPO and for all 8 HPO scenarios. Each point is the result of training one hyperparameter configuration on the calibration ($D_{cal_{split}}$) set then evaluating it on validation ($D_{val_{split}}$) and test sets. There are very few points with high validation-set or test-set error because HPO focuses its search on configurations with low
validation-set error. Three important observations can be made from this figure:

1. An important observation is the “V” shape in Figure 5.6 (A) (\(D_{\text{rand}}\) with singles-HPO and ensembles-HPO); as HPO progresses toward lower validation-set error the test-set error drops at first, but then increases again. This indicates overfitting. Similar to how neural network weights can overfit, overfitting in model selection is a significant problem [50]. The two \(D_{\text{rand}}\) HPO scenarios in Figure 5.6 (A) exhibit overfitting likely because the validation and calibration samples are randomly partitioned which means that for every sample in the validation set, a similar sample in the calibration set can be found. Fitting the calibration set well (or overfitting it) results in a good fit of the validation set too. However, since the test set is an entire separate season, it is not as similar to the calibration samples as the validation set is, suggesting that the models are overfitting to seasonal characteristics in the data. \(D_1\) also exhibits the “V” shape, though to a lesser degree, so the distribution of samples in its calibration set must be too similar to its validation set, allowing for overfitting. Since the validation set for this partitioning (\(D_{\text{val}}^{\text{val}}\)) contains only 2015 season samples, 2015 is not a good choice to enable extrapolation. The other partitioning choices (\(D_{2017}, D_2,\) and \(D_3\)) shown in Figure 5.6 appear to have a more linear and monotonic relationship between validation and test error. In these, minimizing validation-set error is a good proxy for minimizing test-set error.

2. Secondly, the cloud of points is tighter for all ensembles-HPO scenarios (red points in Figure 5.6) than for singles-HPO. This shows that ensembling scenarios have a better correlation between validation and test error. Furthermore, comparing \(D_{2017}\) to \(D_3\) (Figure 5.6 (B) and (E)), the cloud of points is tighter for \(D_3\); this may be because the calibration set is smaller in \(D_{2017}\), which may explain why HPO worked better for \(D_3\).

3. Lastly, the points in \(D_3\) with ensembles-HPO appear to be the most correlated, and least spread out, across all the scenarios (red points in Figure 5.6 (E)), which may explain why this scenario performed the best in the final evaluation (see Figure 5.4, panel E).
Figure 5.6: Relationship between RMSE on the test set versus validation set for each HPO scenario. Blue points: HPO configurations trained once (singles-HPO scenarios). Red points: HPO configurations trained as an ensemble of 40 models (ensembles-HPO scenarios). Plot is cutoff at 2.5 on the x-axis and 2.9 on the y-axis for readability (14 points from singles-HPO in Plot (B) are cutoff but are positioned where you’d expect by linearly extrapolating to (3.34, 4.40)).
5.3.7 Baseline Compared to HPO Configurations

In this section we consider how the baseline (CNN\(_B\)) configuration would have performed had it been a candidate configuration during HPO. Specifically, we look at validation-set error of CNN\(_B\) compared to all the validation scores seen during two HPO scenarios: \(D_{\text{rand}}\) with singles-HPO and \(D_3\) with ensembles-HPO (the worst and best scenarios).

- CNN\(_B\) achieved an average validation-set error of 0.609 (trained using \(D_{\text{rand}}\)). Compared to all the trials tested that use \(D_{\text{rand}}\) in singles-HPO (blue points in Figure 5.6 (A)), this error is approximately at the bottom of the “V” shape. This is not the minimum validation-set error, so CNN\(_B\) would not have been chosen in HPO. Rather, other configurations work better than CNN\(_B\) on the validation set. It is, however, close to where test-set error is minimized.

- When we trained CNN\(_B\)\(^{\text{ensemble}}\) on \(D_3\) (instead of \(D_{\text{rand}}\)) validation error is 0.792 on average. Compared to the trials that use \(D_3\) in ensembles-HPO (red points in Figure 5.6 (E)), this validation-set error is close to the best HPO configuration, which has a validation error of 0.774. Since HPO picked a configuration whose validation-set error is essentially on par with CNN\(_B\)\(^{\text{ensemble}}\), we can say that this HPO objective is successfully guiding the search towards configurations on par with the expertly-chosen one.

5.3.8 Lower Bound

From Figure 5.6, the model configuration with the best test-set loss may give an idea of what the lower bound is for our hyperparameter search space and how far away our models are from it. The hyperparameter configuration that minimizes test-set error is from \(D_{\text{rand}}\) with ensembles-HPO (Figure 5.6 (A)) with a test-set error of 0.766. This configuration was not chosen by HPO, however, because its validation error was not the minimum. We can’t deploy this model because it likely overfits the test set. Re-trained 50 times, it achieves a mean of 0.770 on the test set (for reference, CNN\(_B\)\(^{\text{ensemble}}\) has a mean of 0.790) and its error distribution is shown in Figure 5.4, panel F. This gives a lower bound on what we could possibly expect to achieve with further improvements.

5.4 Conclusion

The goal of this study was to elucidate hyperparameter tuning methods for building neural networks that extrapolate. Experiments showed that HPO overfit the vali-
dation set the most when the validation set was a random selection. We found that HPO worked best when trial configurations were validated on the latest 33% of samples rather than random samples. For this validation set choice, validation-set error (of HPO trials) better correlates to test-set error, thus guiding hyperparameter optimization to a model that extrapolates well. Ensembles of randomly-initialized neural networks reduced the variance and average error substantially for all HPO models and the previous state-of-the-art. Lowering the variance of the neural networks during HPO—combined with the new validation set—led to considerable improvement for hyperparameter optimization which found a hyperparameter configuration on par with CNN$_B^{ensemble}$ without requiring an expert.

For practitioners building neural networks for spectroscopy, three recommendations should be taken under consideration as a result of this study: (1) Optimize hyperparameters by validating on samples that correspond to how the model will ultimately be used. We found evidence that seasons were not independent; a validation set based on temporal structure among batches (or seasons) may lead to better HPO. (2) Reduce neural network variance and improve accuracy using ensembling, such as ensembles obtained by re-initializing the neural network many times, re-training it, and averaging all the models’ predictions. (3) Use ensembles both in deployment and in validating HPO configurations. This study used only one dataset so we can’t conclude how common it is that these recommendations apply. However, it is plausible, in this domain, that extrapolation across seasons led to the improvement. These recommendations should be considered in other domains where extrapolation may occur.
Chapter 6

Sensor Fusion

This chapter addresses the problem described in the thesis abstract of building a prediction model utilizing multiple types of spectrometers given limited labelled data and a large number of features. Integrating information from multiple sensors, known as sensor fusion, and selecting from the plethora of available methods poses a significant challenge in chemometric analysis. This chapter compares several sensor fusion methods (spanning data-level, feature-level, and decision-level fusion) based on partial least squares (PLS) and convolutional neural network (CNN) models and is the first to compare simple sensor fusion methods to the latest multiblock\(^1\) PLS models and parallel-input CNNs. The hyperparameter optimization framework established in the previous chapter (Chapter 5) is used to develop the neural network models. The work in this chapter was published [70] in the *Chemometrics and Intelligent Laboratory Systems* journal.

6.1 Introduction

Integrating information from multiple, complementary, spectroscopic techniques is particularly challenging for small datasets where ground truth is limited, as is common in spectroscopy. The challenge in chemometric analysis lies in picking an appropriate model. There are many methods for sensor fusion ranging from straightforward models to complex neural network architectures designed specifically for sensor fusion. The aim of this chapter is to investigate which methods work well for small data scenarios.

Examples of spectral sensor fusion include combining hyperspectral spectrom-

\(^1\)The term “multiblock” in chemometrics [213] refers to data organized into “blocks,” where each block contains a set of related variables. In machine learning, the terms multiview or multitable are sometimes used. In this study, there are three blocks of spectra, one block per spectrometer.
eters [33], Raman and LIBS [242], and others [14, 38, 39, 82, 142, 152, 153, 163, 164, 184, 197]. Sensor fusion methods, and multiblock methods [165] more generally, are commonly classified into three levels [48, 163], as shown in Figure 6.1. Low-level fusion refers to joining data from multiple sources together and analyzing them in the same way as for a single multi-variable source. In mid-level fusion, features are first extracted from each input and then these features are combined. High-level fusion methods combine multiple predictions derived from each input.

Figure 6.1: Sensor fusion models can be described as fitting into one of three levels, depicted here: low-level, mid-level, and high-level fusion (also known as data-level, intermediate or feature-level, and decision-level).

In this study, we compare single-sensor methods and methods from each level of sensor fusion on a small dataset. In low-level fusion, the simplest option is to concatenate (after any normalization) all the data together then train a regression model as normal. High-level sensor fusion involves training multiple prediction models and combining their outputs. We consider three high-level sensor fusion methods: a linear model that combines the outputs of (1) partial least squares (PLS) models and (2) neural network models, and (3) a recent multiblock extension to PLS. In mid-level fusion, neural networks can be arranged to first extract intermediate features from each sensor, then to combine those features together to output a prediction [115, 161]; this model can be trained end-to-end [175], which avoids loss of information that can arise when doing variable selection or dimensionality reduction to each block prior to the fusion step. Using automatic hyperparameter optimization (HPO) [88] techniques, we optimized the architecture and hyperparameters of a convolutional neural network (CNN) for mid-level sensor fusion. We compare sensor fusion techniques on two prediction tasks individually: predicting lithium (Li) concentration and predicting zirconium (Zr) concentration using three types of spectra.
6.2 Materials and Methods

6.2.1 Data and Experiment Setup

This section describes the dataset, and the training and evaluation methodology. The rock dataset (Section 1.2.1) is used in this study, consisting of 177 rock core samples. Three types of spectroscopy were used to analyze each sample: X-ray fluorescence (XRF), visible to short-wave infrared (Vis-NIR-SWIR) hyperspectral, and laser-induced breakdown spectroscopy (LIBS). For each type of spectrometer, each rock sample was scanned in four orientations with the resulting spectra averaged together. After obtaining spectral measurements, each sample was destructively analyzed using geochemical assays to determine ground truth.

As in many spectroscopy applications, resource limitations constrained the number of samples that could be studied. Each sample is manually moved into position for each sensor type and for each of 4 orientations, and a third-party assay lab was paid to perform geochemical analysis. Not only is this process time-consuming and expensive, but it also destroys the rock in the process.

Prediction Targets

Of the assayed elements (Section 1.2.1), Li and Zr were chosen as the prediction targets because they were relevant to the mineral exploration project where the rocks were collected and they represent examples where direct signals are expected (Zr) and not expected (Li) in the XRF spectrum. Neither element would have direct responses in Vis-NIR-SWIR. Both elements have the potential to show direct responses using LIBS. Since the sensors exhibit unique abilities, their responses are expected to complement each other in a sensor fusion model.

Data Partitioning and Evaluation

Data partitioning and evaluation follows best practices [76, 79, 187, 244]. The experiments in this chapter aim to evaluate and compare each model for its expected performance on samples spanning evenly across the dataset. A test set of 35 samples (20% of the dataset) is selected using the DUPLEX algorithm. DUPLEX selection ensures that the two sets have the same diversity of sensor data across the samples [214]. Raw spectra from all three sensor types are concatenated and fed to the DUPLEX algorithm. By using the sensor data, the two sets will cover approximately the same region of sensor-data-space. Figure 6.4 shows the samples selected for the test set in red; for the purpose of visualization, Li% and Zr% concentrations are plotted.
We use 10-fold cross-validation (CV, see Section 1.4) with the remaining 142 samples which partitions the data into a training set, \( T_k \), and a validation set, \( V_k \), for \( k \in [1..10] \). The partitions are illustrated in Figure 6.2.

![Figure 6.2: Visualization of the data splitting procedure. The dataset contains 177 rock samples split into one test set, 10 training sets, and 10 validation sets for cross-validation.](image)

An overview of model training, hyperparameter optimization (HPO), and evaluation is given in the pseudo-code in Figure 6.3. Root-mean-square error (RMSE) is used to score the models. Model hyperparameters are optimized to minimize the RMSE of cross-validation (RMSECV), which is the RMSE of the union of all validation sets (and is equivalent to minimizing the sum of squared errors summed over all folds, see Figure 6.3, line 6). The best model configuration from HPO (see line 7) is evaluated on the test set, and we report its error distribution as the RMSE of prediction (RMSEP) for each fold of cross-validation (calculated in line 11).

**Spectral Pre-processing**

Spectral data is often pre-processed prior to modelling. One of the most common methods is standardization using standard normal variate (SNV) [17]. SNV is applied to each spectrum by subtracting the mean of the spectrum and dividing by
function HPO(model)
    for config in configurations do
        SSE = 0 // Sum of squared errors
        for k in [1..10] do
            // Train model on training set C_k (see Figure 6.2):
            f_k = train(config, C_k)
            // Score predictions of validation set V_k:
            SSE += \sum_{<x,y> \in V_k} (f_k(x) - y)^2
        return f_1..10 of config with lowest SSE
        // Note: \text{RMSECV} = \sqrt{\frac{\text{SSE}}{n}}, where \(n = 142\) is the number of
        samples in \(V_{1..10}\)
    function final_evaluation(model)
        f_1..10 = HPO(model)
        for k in [1..10] do
            // Score predictions of test set T:
            RMSEP_k = \sum_{<x,y> \in T} \sqrt{\frac{(y - f_k(x))^2}{|T|}}
        return RMSEP_{1..10}

Figure 6.3: Pseudo-code overview of model training, hyperparameter optimization (HPO function), and final evaluation. \(C_k, V_k,\) and \(T\) are sets of \((x, y)\) pairs in training, validation, and test sets respectively (where \(k\) denotes \(k^{th}\) fold); these dataset partitions are shown in Figure 6.2.

the standard deviation of the spectrum [260]. Although originally developed for near-infrared (NIR), it is a general technique that can also be applied to other types of spectra, such as LIBS [76, 196]. We test models with and without SNV applied to all three types of spectra.

A comparison of other pre-processing methods [49] is outside the scope of the present study. Pre-processing has been found [166] to sometimes reduce performance. In preliminary testing, we found negligible difference in PLS model performance using the following pre-processing methods: the first and second derivative of the smoothed spectrum (smoothing via a Savitzky–Golay filter with a window size of 13), and the first and second derivative of the smoothed SNV spectrum. The amount of smoothing (i.e., the window size) is a hyperparameter that ought to be tuned per sensor. To keep the hyperparameter search space small and to keep the number of experiments manageable, these methods were excluded from the study. Future work is to consider the effect of pre-processing methods, and their combinations, on sensor fusion performance.
Figure 6.4: The DUPLEX algorithm selects a test set (red ×) that captures the same diversity of samples as the remaining samples.

6.2.2 PLS-based Models

Three PLS-based sensor fusion models are tested based on partial least squares (PLS, see Section 1.3.2) [247] regression. Two hyperparameters are considered: the number of components (from 1 to 40) and whether to standardize the spectra with SNV. The hyperparameters are optimized through grid search to minimize the root mean square error of cross-validation (RMSECV). The decision of whether to use SNV in the sensor fusion models is decided by the validation results of single-sensor PLS models described below.
Single-Sensor Models: with PLS

Individual PLS models are trained for each sensor type and each target. A grid search is performed to optimize the number of components from 1 to 40 and whether to use SNV. The best hyperparameter configuration is evaluated on the test set.

For Li, applying SNV to spectra in all sensor types produced a better RMSECV than using raw spectra. For Zr, the best approach was to use raw spectra from XRF and Vis-NIR-SWIR and to use SNV on LIBS. The PLS models that follow employ SNV based on these single-sensor results.

Low-level Sensor Fusion: with PLS

In low-level sensor fusion, the fusion is carried out at the data level (see fusion levels in Figure 6.1). This is achieved by concatenating spectra together then training a PLS model as usual (e.g., [33]). The predictions obtained in this way are equivalent to multiblock PLS (MB-PLS) algorithms [20]. We refer to this sensor fusion model as PLS-LL for “Partial Least Squares - Low Level.”

Whether to use raw or SNV spectra is pre-determined by the validation results of the single-sensor PLS models (from Section 6.2.2); SNV is applied prior to concatenation. However, the number of components (from 1 to 40) is again optimized through grid search.

High-level Sensor Fusion: with NNLS

In high-level, or decision-level, sensor fusion, fusion is achieved by combining “decisions,” which are the predictions made by multiple trained models, in this case, the predicted Li or Zr concentrations from single-sensor PLS models. We call this sensor fusion model PLS-HL for “Partial Least Squares - High Level.”

We use a non-negative least squares (NNLS) model to combine the prediction values. Lee et al. [142] found that this type of sensor fusion model outperformed MB-PLS. NNLS is a method used to solve linear least squares problems where the model coefficients are constrained to be positive [134]. The non-negative constraint aligns with our intuitions that the final prediction is essentially a weighted average of the predictions made by each single-sensor model. A negative weight, however, would indicate that an increase in the corresponding single-sensor model’s predicted concentration is associated with a decrease in the final predicted concentration, which, intuitively, is not what we expect.

The NNLS model predictions are weighted sums of three predictions from the three best hyperparameter-tuned single-sensor PLS models. For each fold of cross-validation, the three single-sensor models make predictions on the training set (as
per the data partitioning shown in Figure 6.2), and then the NNLS model uses these predictions to calibrate its weights.

**High-level Sensor Fusion: with ROSA**

Response-Oriented Sequential Alternation (ROSA) [147, 212] is a state-of-the-art multiblock extension to PLS. In ROSA, components are constructed sequentially, using only one block per component. For each component, whichever block reduces the variance the most is selected. A benefit of this approach is that it is invariant to block scaling.

ROSA can be used for sensor fusion. When the input blocks to ROSA are spectra pre-processed in different ways, the result is a model that automatically chooses from the available pre-processing methods [167]; when the input blocks are spectra from different sensors, ROSA effectively performs sensor fusion. We use spectra from each of the three sensor types.

Whether to use raw or SNV spectra is pre-determined by the validation results of the single-sensor PLS models (from Section 6.2.2). As in the other PLS models, the number of components (from 1 to 40) is optimized through grid search.

**6.2.3 Neural Network Models**

Two single-sensor neural networks (NN) and one sensor fusion model are developed. Each is developed in turn, as we freeze some hyperparameters after each one to reduce the size of the hyperparameter search space in subsequent models. The models are trained separately for each prediction target.

The training procedure for all the neural networks is based on standard practice [23, 60, 187]. Each input variable is standardized independently by subtracting the mean and dividing by the standard deviation [107]. The weights of the neural network are optimized by stochastic gradient descent using the ADAM [126] algorithm. All the layers (except the last) use exponential linear unit (ELU [55]) activations. Convolutional kernels and fully-connected weights are initialized by the He normal [105] initialization method. The batch size is set to the full size of the training data. The learning rate halves each time validation loss stops improving for 25 epochs, and early stopping is employed with a patience of 200 epochs; validation loss is calculated on 10% of training samples, selected randomly at the start of a new training run. Training runs for a maximum of 10000 epochs (but mean number of epochs was around 1300 epochs, due to early stopping).
Neural Network Hyperparameter Optimization

Automatic hyperparameter optimization is performed for all the neural network models following established protocols [68, 162]. There are many hyperparameters available to tune and testing all hyperparameter combinations is computationally prohibitive, even when only a subset is considered. Like in Section 5.2.5, an implementation of Bayesian optimization (BO), called hyperopt [28], is used to search through the space of hyperparameters. The BO algorithm chooses the configurations to test in line 2 of Figure 6.3 and keeps track of the configuration resulting in the lowest root mean squared error of cross-validation (RMSECV).

Our study focuses on a select number of hyperparameters, similar to previous studies [35, 68, 187], which are listed for each neural network model in the following sections. Each run of HPO performed about 700 trials. We ran this many because the best hyperparameter configuration stopped changing between 50 and 600 trials (but typically around 300), and then we ran each one longer until they all had at least 700 trials.

One challenge in HPO is that neural networks are known to produce different results each time they are trained [23], which adds noise to the objective. However, it has been demonstrated [68] that ensembling randomly-initialized neural networks is effective in reducing variance during hyperparameter optimization. The number of members in the ensemble can be selected to trade-off between variance and compute time. To determine the number of members, an analysis of the effect of ensemble size on variance was carried out (see C.1). This analysis was conducted using the initial configuration of the NN1 model, prior to hyperparameter optimization. Based on this analysis, variance is significantly reduced when the ensemble size reaches approximately 30 members. Beyond 30, both variance and mean RMSE show little sensitivity to the specific number chosen. We opted for an ensemble size of 40 members given the computational resources at our disposal.

NN1: Single-Sensor Neural Network

One-dimensional (1D) CNNs are a type of neural network commonly used for analyzing sequential data, such as spectra [2, 6, 52, 60, 75, 83, 148, 151, 160, 208, 251, 263]. CNNs for spectral analysis all follow a similar pattern: one or more convolutional layers followed by one or more fully-connected layers, with an activation function after each layer. The convolution operation is useful for spectra because it automatically learns how to transform the spectra in the same way that pre-processing techniques do (such as smoothing and taking derivatives) [35, 60, 107, 203].

CNNs can take on many possible architectures; it is recommended [187] in
Figure 6.5: NN1 is a single-sensor model with one convolutional layer and one fully-connected layer which outputs a predicted concentration per input spectrum. An NN1 model is trained per sensor and per target.

chemometrics to start with simple networks and gradually increase complexity. Furthermore, it has been suggested [162] that, in chemometrics, a single convolutional layer may be sufficient. This observation aligns with our personal experience and the experiences of other researchers [208, 246] when working with spectral data. The first neural network (NN1) we test uses one convolutional layer and one fully-connected layer (also known as a dense layer), shown in Figure 6.5. The search space selected for NN1 is:

- Whether to standardize spectra using SNV
- \( L_2 \) regularization factor (a real number sampled uniformly from \( 10^{-4} \) to 100 in log space):

  The log space is useful because a similar impact is expected when changing the value from 0.001 to 0.002 as from 10 to 20, for example, whereas 10.001 to 10.002 is likely negligible.

- Learning rate (LR) (one of 1, \( 10^{-1} \), or \( 10^{-2} \))
• Convolutional layer’s filter size (one of 3, 5, 7, 11, 15, 21, 29, 41, 57, 79, 111, 155, or 217; search space is approximately logarithmic and rounded to the nearest odd number, ranging from 3 to 217)

• Convolutional layer’s number of filters (an integer from 1 to 5)

HPO is carried out for all three sensor types and for both targets (Li and Zr). Additionally, whether to use SNV is optimized in a separate HPO run to allow analysis of the effect of standardization. Thus, 12 HPO runs are carried out for NN1.

**High-level Sensor Fusion: with NNLS**

Just like the PLS-HL model described in Section 6.2.2, the predictions from trained single-sensor neural networks may also be fused together to form a high-level sensor fusion model. We call this model NN1-HL.

**NN2: Adding a hidden layer**

The second neural network (NN2) we developed, shown in Figure 6.6, is similar to NN1 except an additional (hidden) fully-connected layer is added. This fully-connected layer is added because it potentially allows the sensor fusion neural network (described in the next section) to balance sensors via $L_2$ regularization. The additional layer also allows the model to learn more complex interactions and nonlinearities, but at the risk of overfitting. While this could have been included as a hyperparameter in NN1, doing it separately allows us to evaluate the effect of adding this layer.

To reduce the search space, some hyperparameters are frozen based on the results of NN1’s HPO (discussed in the results, Section 6.3.1, and Table 6.1). Since a fully-connected layer is added, the $L_2$ regularization factor should be re-optimized for NN2. Typically, the $L_2$ factor is the same among all fully-connected layers, but here they are separate because they will be optimized individually in the sensor fusion model. The HPO search space for NN2 is:

• $L_2$ regularization factor for the intermediate layer (a real number sampled uniformly from $10^{-4}$ to 100 in log space)

• $L_2$ regularization factor for the final layer (a real number sampled uniformly from $10^{-4}$ to 100 in log space)

• Number of units in the intermediate layer (an integer from 1 to 20)
Figure 6.6: NN2 is similar to NN1 except an additional fully-connected layer is added.

NN3: Feature-level Sensor Fusion

The third neural network is a parallel-input convolutional neural network \cite{115, 161} that we use to perform mid-level sensor fusion, which we call NN3-ML for “Neural Network 3 - Mid-Level.” The model consists of layers arranged in parallel, each with its own input. Each input, known as a “block,” is a set of spectra. There is one block per spectrometer.

The neural network architecture is shown in Figure 6.7 and is composed of three convolutional layers, one per block, each with its own set of learned filter(s). Like NN2, a fully-connected layer follows each convolution layer; these fully-connected layers output intermediate features specific to each sensor. Finally, a fully-connected layer connects these intermediate features together—“fusing” the features from each sensor type—and outputs the predicted Li or Zr concentration. Instead of one shared regularization hyperparameter, each fully-connected layer has its own $L_2$ regularization factor hyperparameter:

- **$L_2$ on intermediate fully-connected layers:** The sensor-specific branches of the model are individually regularized to enable the sensor fusion neural
Figure 6.7: NN3 is essentially three of NN2 stacked in parallel, one per sensor (or “block”). The final fully-connected layer “fuses” the features from the intermediate layer together to make a prediction.
network to balance the sensor types appropriately. The initial sampling distribution for each of these hyperparameters is based on the results of HPO from NN2. Specifically, a lognormal distribution centered at the optimal value from NN2 for each sensor and target, with a standard deviation such that 95% of samples fall within one order of magnitude.

- \( L_2 \) on final fully-connected layer: Due to an increase in the number of connections to the final output as compared to NN2, it is necessary to re-optimize the \( L_2 \) regularization factor. The initial sampling distribution in the search space is a lognormal distribution centered at the average value from NN2’s HPO runs, with a standard deviation equal to that of the values from NN2’s HPO runs.

### 6.2.4 Compute Hardware and Software

Source code used to train the models is available on GitHub\(^2\). Python 3.9, Octave 7.3, and R 4.2.2 were used in this study. We used the DUPLEX algorithm, which is available in the prospectr R package\(^3\). Neural network models were trained using TensorFlow 2.6\(^4\) in Python. PLS models were trained using the scikit-learn Python package\(^5\). ROSA models were trained using the multiblock R package\(^6\), which we ran via rpy2\(^7\). Experiments were run on a high-performance compute cluster called Cedar\(^8\), which is hosted by Simon Fraser University for the Digital Research Alliance of Canada, with NVIDIA V100 and P100 GPUs and Intel Xeon CPUs (1352 GPU devices and 94528 CPU cores). A total of 374 GPU-days were utilized in training neural networks in the HPO experiments.

### 6.3 Results and Discussion

With numerous methods available in the literature for fusing data from multiple sensors, we trained models from each level and compared against single-sensor models for each sensor. The results of neural network hyperparameter optimization

\(^3\)https://rdrr.io/cran/prospectr/man/duplex.html and https://github.com/l-ramirez-lopez/prospectr
\(^4\)https://www.tensorflow.org/versions/r2.6/api_docs/python/tf
\(^6\)https://github.com/khililand/multiblock
\(^7\)https://rpy2.github.io/
\(^8\)https://docs.alliancecan.ca/wiki/Cedar
are reported first as these are utilized in constructing NN2 and NN3-ML. Then the results of single-sensor and sensor fusion models are presented.

### 6.3.1 Neural Network HPO

Some hyperparameters in NN2 are frozen based on the HPO results on NN1, listed in Table 6.1. Where possible, the same value is used across both targets for a given sensor (when the difference in prediction error is negligible); for instance, the best number of filters for the Vis-NIR-SWIR model targeting Zr was found to be 4, whereas targeting Li it was found to be 5 but 4 was almost as good, so 4 is used for both models in NN2.

The best hyperparameter values found from NN2 HPO are listed in Table 6.2. For NN3-ML, frozen hyperparameters include the ones previously frozen and also the best-found value for the number of hidden units (per sensor and per target) from NN2 HPO. The search space, described in Section 6.2.3, for $L_2$ regularization factors is based on the best $L_2$ values from NN2. The optimized hyperparameter values for NN3-ML are listed in Table 6.2.

More hyperparameter and architecture variations are possible, but are not necessary for testing sensor fusion and would greatly expand the hyperparameter search space if included. However, we conducted some preliminary testing (not shown) of some but found negligible differences in prediction accuracy on the validation set; the methods we tried were (1) batch normalization [155] after the convolutional layer or after the fully connected layers, (2) a proximity $L_2$ norm\(^9\) [2], (3) smaller batch sizes [207], (4) learning rate scheduler hyperparameters (patience, reduction factor, and minimum learning rate), and (5) pre-processed spectra in various combinations (including first and second derivatives of smoothed spectra, for both PLS and neural networks). An open avenue of research is to establish which architecture(s) and hyperparameter configurations are best in chemometrics; narrowing down these options would greatly reduce the effort and compute time needed to develop new neural network models.

### Results of Single-Sensor Models

Resulting prediction scores of the single-sensor models on two prediction tasks are presented in Figure 6.8. Each model’s results is shown by a box representing the distribution of RMSEP scores from 10 trained instances of this model, where each

\(^9\)The proximity $L_2$ norm introduces a regularizer on the convolution filters that encourages them to be smooth; in preliminary testing, we found that this produces nice smooth filters resembling Gaussian smoothing and first derivatives, which has the potential to be advantageous in some applications and may merit further investigation.
Table 6.1: Frozen hyperparameters for NN2 as a result of NN1 HPO.

<table>
<thead>
<tr>
<th>Target Sensor</th>
<th>Standardization</th>
<th>LR</th>
<th>Conv. width</th>
<th>Num filters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Li XRF</td>
<td>SNV</td>
<td>0.01</td>
<td>111</td>
<td>1</td>
</tr>
<tr>
<td>Li Vis-NIR-SWIR</td>
<td>SNV</td>
<td>0.01</td>
<td>57</td>
<td>4</td>
</tr>
<tr>
<td>Li LIBS</td>
<td>SNV</td>
<td>0.01</td>
<td>57</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 6.2: Resulting hyperparameter configurations after HPO for all fully-connected (FC) layers in NN2 and NN3-ML

<table>
<thead>
<tr>
<th>Target Sensor</th>
<th>After NN2 HPO</th>
<th>After NN3-ML HPO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intermediate FC layer</td>
<td>Final FC</td>
<td>Sensor-specific FC</td>
</tr>
<tr>
<td># units</td>
<td>$L_2$</td>
<td>$L_2$</td>
</tr>
<tr>
<td>Li XRF</td>
<td>6</td>
<td>2.26e-02</td>
</tr>
<tr>
<td>Li Vis-NIR-SWIR</td>
<td>19</td>
<td>9.18e-04</td>
</tr>
<tr>
<td>Li LIBS</td>
<td>17</td>
<td>1.61e-03</td>
</tr>
<tr>
<td>Zr XRF</td>
<td>9</td>
<td>2.48e-03</td>
</tr>
<tr>
<td>Zr Vis-NIR-SWIR</td>
<td>8</td>
<td>5.02e-04</td>
</tr>
<tr>
<td>Zr LIBS</td>
<td>17</td>
<td>6.12e-03</td>
</tr>
</tbody>
</table>

instance was trained on a different fold of cross-validation. In order to facilitate interpretation, a naive baseline is included that outputs the average training set Li or Zr concentration (labelled “PtA” for “predict the average” and drawn as a black box in the figures). Single-sensor models are labelled with their sensor type appended to the name (e.g., NN1-XRF) and colour-coded with XRF in orange, Vis-NIR-SWIR (labelled HS, for hyperspectral, for brevity) in blue, and LIBS in red. Each sensor has different strengths and weaknesses. Sensor-specific results are discussed next.

**XRF:** Li is not directly detectable by XRF, but, based on the geology of the ore deposit, a geochemical relationship is expected between Li and Rb which XRF is well-suited to detect. A strong Zr peak is expected in XRF. Models using the XRF sensor perform the best on average for both the Li and Zr target variables. While it was expected that Zr models using XRF would do well, Li performed better than expected despite relying on indirect signals.

**Vis-NIR-SWIR:** The Vis-NIR-SWIR models perform the poorest of the three sensors. Minerals that host Li are known to show spectral responses in the SWIR region, so it was expected that these models would exploit this. The host mineral for Zr, which is zircon, generally does not show spectral features. For Li, Vis-NIR-
SWIR models were able to extract a signal, as expected, achieving much better performance than PtA. Zr models performed poorly, which is also as expected. Overall, Vis-NIR-SWIR models did worse than both the other sensor types.

**LIBS:** LIBS is expected to have difficulty with Li because the spectral range does not cover the key lines related to Li. The primary Li peak in LIBS is near 671 nm, which is outside of range, and the other Li peaks are small and only present in a few of the samples. Rb, which is correlated to Li, is also out of range. The Li models did very well considering that Li is out of range, especially with the neural network models (NN1 and NN2). Zr is within range in LIBS and the Zr models did reasonably well, albeit not as well as XRF. LIBS may not have done as well as XRF because of the difference in spot size for the LIBS system compared to the XRF and Vis-NIR-SWIR systems, which are considerably larger. Although the rocks analyzed are fine-grained and particles were not very heterogeneous, the scale of observation relative to the volume that was geochemically analyzed would have an impact on the representativity of the spectrometer response.

**Results of Sensor Fusion Models**

Sensor fusion results are shown in Figure 6.8, drawn in green and labelled with the sensor fusion level (LL, ML, or HL) appended to the name (e.g., PLS-LL for PLS utilizing low-level fusion). The results indicate that the best sensor fusion model depends on the prediction target.

**Li target:** Of the sensor fusion models and with Li as the target, ROSA-HL obtains the best RMSE on average. The next best is the high-level fusion of neural network models (NN1-HL). Considering all the models for the Li prediction task (including single-sensor models and sensor fusion models), the results indicate that the single-sensor PLS model using XRF spectra achieves the lowest RMSE. That the XRF-only approach performed best for Li was unexpected. While the XRF response was expected to exploit the relationship between Li and Rb, the Vis-NIR-SWIR data was expected to complement XRF in sensor fusion models because of the Li host minerals that display spectral responses in the SWIR region. Li predictions with Vis-NIR-SWIR and LIBS were also substantially better than PtA, which indicates that there are features in the reflectance (Vis-NIR-SWIR) and LIBS spectra related to the Li-mineralized rock type. Thus, it is surprising that the sensor fusion models were unable to leverage the different types of sensors to achieve better results than PLS with XRF alone.

**Zr target:** Of the sensor fusion models with Zr as the target, the best is PLS-HL, followed by NN1-HL, then NN3-ML. PLS-HL also outperforms all the single-sensor models, with a lower RMSE than any of the single-sensor PLS models, on which it is based. The improved performance may be because of the complemen-
Figure 6.8: Test-set results of (a) Li and (b) Zr prediction tasks. Each box gives the distribution of test-set scores (measured in RMSE, which is in the same units as the target variable) obtained by training each model 10 times (on different folds of cross-validation). Lower (left) is better. The edges of the boxes are the upper and lower quartiles of the distribution with a line at the median. The whiskers extend from the box to show the range of the data. Vis-NIR-SWIR hyperspectral is labelled HS for brevity. Model names are suffixed with the name of the sensor (XRF, HS, or LIBS) for single-sensor models or the sensor fusion level (LL: low level, ML: mid level, HL: high level) for models that use all three sensors. PtA means “predict the average” (which is a naive baseline).
tary information provided by the different sensor types.

The performance patterns for Zr are more in line with the expected results. The XRF-only approaches were the best for the single sensors, which is logical since there is a strong XRF peak expected. The Vis-NIR-SWIR-only approach was marginally better than PtA, which is not surprising since it is difficult to discern Zr-enriched rocks visually, and the host mineral (zircon) generally does not show spectral features unless the field of view is dominated only by this mineral. Prediction error for LIBS fell between Vis-NIR-SWIR and XRF, which is expected since this sensor’s response is compositionally controlled (which is good for predicting element concentrations) but the spot size is small (which reduces representativity). Sensor fusion approaches achieved as good and better results than any single sensor, thus demonstrating that integrating different types of sensor data can improve overall performances.

6.3.2 Interpreting Behaviour of High-level Sensor Fusion

A benefit of high-level sensor fusion models is the ease of interpretability; NNLS weights are easily examined and ROSA reports the block order of the “winning” blocks.

NNLS weights for PLS-HL and NN1-HL are approximately equal across the 10 folds, so each average over 10 folds (rounded to 2 decimal places) is reported in Table 6.3. The PLS-HL model for Li performed poorly. From the weights, we observe that this model favours the PLS-LIBS model predictions the most. We investigated further and found that PLS-LIBS’s training RMSE is much lower than its validation and test-set RMSE which indicates overfitting, and it fits the training data better than the other two sensors. As a result, PLS-HL for Li is weighing LIBS higher than XRF and Vis-NIR-SWIR, and is adopting a similar RMSE as the LIBS model on its own.

The remaining three NNLS models (PLS-HL for Zr and NN1-HL for Li and Zr) exhibit similar behaviours: XRF has the highest weight followed by LIBS, and the weight for Vis-NIR-SWIR is zero or almost zero. Low Vis-NIR-SWIR weight is consistent with the training and test-set predictions for Vis-NIR-SWIR which were worse than both XRF and LIBS. By not utilizing Vis-NIR-SWIR, these models are potentially missing out on complementary sensor information.

As for ROSA, in the Li prediction task, it selects LIBS in the first component and XRF for the remaining 17 components; Vis-NIR-SWIR never won in any of the iterations. For the Zr prediction task, the ROSA model has 12 components. Results vary among the 10 models from cross-validation, but to summarize, the components are mostly XRF except: in 4 models Vis-NIR-SWIR is used once (i.e., Vis-NIR-SWIR is used in 1 of 12 components), in 3 models LIBS is used
once, and in 3 models neither Vis-NIR-SWIR nor LIBS is used (only XRF). Since the ROSA models are primarily picking XRF spectra in their components, it is not surprising that their RMSEP distribution closely matches the XRF models.

In summary, results indicate that XRF data is the most influential in the sensor fusion models, and that future work is needed to inject domain knowledge into these models in order to better exploit the other sensor types.

### 6.3.3 Modelling Considerations and Limitations

Aside from prediction error, other factors should be considered when choosing a sensor fusion method. The neural network models were the most time-consuming to develop. High-level fusion using NNLS is computationally fast but requires two steps: training individual models per sensor, then training the NNLS model using their outputs. In practice, individual models are often built anyway, so adding a high-level fusion model on top (such as PLS-HL or NN1-HL) is easy and practical, so long as all individual models perform well on their own; as we observed, when one model overfits then high-level fusion overfits as well. ROSA models are also quick to train, have only one hyperparameter (the number of components), and perform well. The main downside of PLS-based models is that further testing of pre-processing is typically required, whereas the neural network models likely do not benefit from it. NN1-HL may strike a good balance between the flexibility of neural networks and the simplicity of high-level sensor fusion.

An opportunity exists for a meta-analysis of the results. For example, it may be that certain clusters of samples have poor predictions, possibly related to other less quantitative variables such as rock type. Finding out why “difficult samples” perform poorly may lead to a better understanding of each sensor’s and model’s capabilities and limitations, such as when one sensor or model is applicable versus another. A meta-analysis may reveal patterns about where sensor fusion can have the greatest impact in the mineral exploration and mining industries, which are currently undergoing rapid digitalization. Additionally, there may be grade ranges

<table>
<thead>
<tr>
<th></th>
<th>XRF</th>
<th>Vis-NIR-SWIR</th>
<th>LIBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLS-HL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Li</td>
<td>0.05</td>
<td>0.07</td>
<td>0.91</td>
</tr>
<tr>
<td>Zr</td>
<td>0.68</td>
<td>0.01</td>
<td>0.36</td>
</tr>
<tr>
<td>NN1-HL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Li</td>
<td>0.60</td>
<td>0.02</td>
<td>0.50</td>
</tr>
<tr>
<td>Zr</td>
<td>0.84</td>
<td>0.00</td>
<td>0.30</td>
</tr>
</tbody>
</table>

**Table 6.3:** Model weights for the non-negative least squares (NNLS) sensor fusion models: PLS-HL and NN1-HL.
that are more important than others. If the cut-off grade for a mine is 0.2% Li, then a sample with 0.6% Li predicted to have 0.7% Li is less relevant than a 0.1% Li sample predicted at 0.2% Li. In the first case, the rock is clearly of ore grade regardless of the prediction error. In the second case, the prediction error would result in the material being classified as ore when instead it should be treated as waste. While there was not one sensor fusion model that dominated the rest, having multiple models to choose from is useful when additional criteria arise in real-time sensing environments (e.g., sensor-based core logging and rock sorting).

Small datasets are an unfortunate reality because of the cost associated with sample collection and analysis [186]. Training neural networks with limited data poses challenges compared to large datasets, due to the risk of overfitting given the number of parameters in a neural network. In the literature, CNNs have been successful on various small datasets, including 80 NIR samples [258, 263], 192 Vis-NIR samples [255] 124 MIR samples [151], 219 NIR samples [177], and 60 NIR samples [257], among others [251]. An increase in data is expected to significantly enhance the performance of neural network sensor fusion models as compared to PLS models.

This study serves as a preliminary investigation due to the reliance on a single dataset. More studies that test diverse domains are required before definitive recommendations can be made. Datasets with the ground truth and multiple sensors are typically expensive to create, so it is even more important to have multiple studies, so expensive decisions can be made with the best evidence.

6.4 Conclusion

While the neural network models were competitive, the most effective approach for sensor fusion was achieved through high-level PLS-based methods. The ROSA model for Li prediction and the PLS-HL model for Zr prediction yielded the best results among the sensor fusion models. We compared sensor fusion models—spanning low-level, mid-level, and high-level sensor fusion—to single-sensor models and found that sensor fusion improved predictions for one out of the two prediction tasks. For the Zr prediction task, combining information from three sensors resulted in the lowest RMSE on average, effectively leveraging the data from multiple sensors. For the Li prediction task, the PLS single-sensor model utilizing XRF outperformed all the sensor fusion models. Among the single-sensor models, neural networks demonstrated better performance than PLS in two out of the six tasks. A parallel-input convolutional neural network was developed for sensor fusion. This model (NN3-ML) consistently achieved comparable RMSE to the models utilizing a single sensor (XRF), for both prediction tasks. It is important to recognize that improved performance cannot be guaranteed solely by increasing
the number of sensor types or employing a more sophisticated prediction model. And, given the small size of the dataset in the experiments, larger datasets are likely to exhibit different behaviour. Conducting multiple studies using diverse datasets is essential to gain deeper insights. This study should be seen as one of many contributing to a more comprehensive understanding of sensor fusion in spectroscopy.
Chapter 7

Conclusion

7.1 Conclusion

Extracting chemically significant information from a spectrum is difficult. Given the limited number of labelled examples available in many real-world applications, several models were developed and evaluated in this thesis for small-data scenarios in spectroscopic data analysis. These models span the gamut, as shown in Figure 7.1, ranging from those integrating domain knowledge to those reliant on data-driven learning. Some models lean heavily toward incorporating domain knowledge, such as embedding principles from physics and geology, while others lean more toward learning from data. Each model in this thesis falls somewhere in between these two extremes.

Figure 7.1: Each chapter in this thesis sits between two extremes, approximately in the order above: from reliance on incorporating domain knowledge on the left to reliance on learning purely from data (with only general domain knowledge) on the right.

Contributions in this thesis are interdisciplinary, spanning X-ray spectrometry, chemometrics, spectroscopy, and machine learning. The first two models, FitSim and AutoSim, embed domain knowledge in an XRF simulator (Chapters 2 and 3). These models lean heavily on incorporated domain knowledge, while learning from data to estimate unavailable parameters, such as instrument and environment parameters. SpectraAE, developed in Chapter 4, uses domain knowledge...
about spectra in general and is broadly applicable to spectra from any type of spectroscopy. As auto-encoders, both AutoSim and SpectraAE are capable of unsupervised (and semi-supervised) learning, lessening the need for labelled data by learning from spectra without ground truth. Neural networks have many hyperparameters, which require optimization toward an accurate estimate of performance given by the validation set. Automatic HPO is improved (in Chapter 5) through neural ensembling—ensembles built by averaging several randomly-initialized neural networks—and through the choice of validation set. The validation set is chosen based on domain knowledge of the extrapolation prediction task. Lastly, multisensor experiments (Chapter 6) tested the applicability of a variety of sensor fusion methods for combining spectral data from multiple spectroscopic techniques. Despite the small number of samples and the larger dimensionality of the input, the sensor fusion neural network with extensive HPO performed competitively.

Small-data scenarios are plentiful in many industries and applications. For instance, the rock dataset used in this thesis consisted of individual rocks, but the intended application is real-time analysis on a mining shovel or conveyor belt. In this setting, samples are large batches of rocks for which ground truth is far more expensive to acquire. It is expensive because any interruption in the mining process incurs a loss of profit (due to lost time and material) as well as equipment and logistical costs associated with diverting a sample for geochemical assay (where a sample, in this case, may be up to 120,000 kilograms). Obtaining standards is not expected to be a one-time event, because every mine can have multiple deposits and every ore deposit has a different geology. Therefore, models that can be trained with limited labelled data, as discussed in this thesis, are crucially important.

For some applications, it will be more practical to incorporate domain knowledge, while for others it may be best to use empirical data-driven approaches. This thesis explored diverse models, each with distinct merits, suitable depending on the amount of domain knowledge available and the amount of labelled and unlabelled data.

7.2 Future Work

7.2.1 Diverse Datasets

Future research can expand on the insights of this thesis by conducting multiple studies with diverse datasets. The limitation of a small number of datasets underscores a potential area for future exploration. The thesis examined four spectroscopy datasets—three from rock samples and one from mango samples—which opens an opportunity to test different types of samples and different spectroscopic
techniques to gain deeper insights.

### 7.2.2 The Problem of Limited Data

A benefit of FitSim, and other model-based fitting routines that incorporate fundamental physics, is the unique ability to make inferences with very little ground truth. An interesting avenue of research would be to label the unlabelled samples using FitSim, then proceed to train another supervised model using these newly-labelled samples. An additional avenue worth exploring involves fitting minerals, rather than elements, to an observed spectrum. This approach would allow the inclusion of mineralogical domain knowledge in the model.

Active learning is another approach to deal with limited labelled data, and has been used in spectroscopic regression [71]. In active learning, the objective is to minimize prediction error while also minimizing the number of samples needed to build the model. Starting with a small set of labelled data, additional samples are selected and added to the training set. Two common sample selection methods are (1) choosing samples that are most different from existing samples and (2) choosing samples that are most similar to already-labelled but poorly-performing samples. Samples are typically chosen sequentially. It may be impractical to use active learning to select samples during mining but could be made more practical by (1) selecting samples in an offline setting where ground truth is obtained by a slow but accurate spectral analysis method, such as FitSim, or by (2) developing a technique to select batches of samples rather than one at a time.

Data augmentation is an approach to help deal with limited data [191] by incorporating basic domain knowledge about the structure of the data [27]. Historically, the term “virtual examples” has been associated with this technique [178]. Data augmentation has been shown to improve predictive performance on NIR spectral data [35, 162, 246], LIBS [130], and others [52] but it is not known whether similar approaches would work for XRF. How best to augment the XRF dataset will require exploration, as it is not immediately obvious what invariances can be exploited. For example, in an image labelled “cat,” one can rotate, scale, or move the image and the image label will remain unchanged, but this is not applicable to XRF spectra. Adding random noise to the input spectra is one option for augmenting the dataset.

Future work is to test AutoSim and SpectraAE on more unlabelled data, and to further test the effect that unsupervised learning has on the final model. Since both of these models are auto-encoders, they are capable of generating spectra through the decoder function. Generated spectra may be useful for interpretability—to further understand what the model has learned—and for training other supervised models on the simulated data [118, 124].


A hybrid approach [124] may be able to take advantage of the domain knowledge in AutoSim with the generality of SpectraAE. For example, pre-training an auto-encoder with an XRF simulator as the decoder, as in AutoSim, first followed by swapping out the decoder for a more general one (as in the decoder in SpectraAE) and then fine-tuning it. Since AutoSim incorporates domain knowledge of the world, it is more likely to produce a trained model grounded in reality and can be trained unsupervised, without any labelled data. Then, the more-general decoder of SpectraAE may get around any modelling inaccuracies present in our XRF simulator. Other variations of this hybrid approach can also be imagined, such as training a “two-headed” architecture where the auto-encoder has two decoders that are trained simultaneously—one decoder is the XRF simulator and the other decoder is the more-general decoder from SpectraAE.

Another interesting research avenue can make use of the auto-encoder models: research out of Google Brain proposed regularizing an AE with a critic network (like in the standard GAN framework) [30]. The key idea is to interpolate between the intermediate representations of two samples, then give the critic model the reconstruction of the interpolated datapoint (given by the AE’s decoder). The AE is trained to fool the critic into thinking that its reconstructions of interpolated samples are real, non-interpolated, samples. This is an interesting idea for limited-data spectroscopy scenarios because (1) model training is entirely unsupervised, (2) it creates more data for training by using interpolations between any pair of samples, and (3) interpolation is a difficult task, which, if it is successful, would imply a deep understanding of the underlying physical processes that generate the spectra. In XRF, for instance, the encoder would have to learn to disentangle composition from matrix effects in order to produce a physically-plausible reconstruction of a mix of two samples.

A natural next step in the research direction started in the sensor fusion neural network architecture is to extend this to an auto-encoder format like AutoSim and SpectraAE. Preliminary experiments show that an auto-encoder with multiple decoders is capable of reconstructing multiple types of spectra simultaneously (shown in Figure 7.2) while sharing the intermediate representation amongst them. This is a powerful idea because it opens up the possibility of “translation” [186] between different types of spectroscopy. This is known as cross-domain disentanglement [236]. For example, one could translate an XRF spectrum into a reflectance (Vis-NIR-SWIR) spectrum, or vice versa. In this case, one may incorporate domain knowledge of the relation between compounds, such as minerals, and elements in a multiple-layer encoding that maps compounds to their elemental composition. With translation possible, one could imagine training a model not only on spectra from one dataset, but spectra from multiple. Such a “large spectral model” may allow learning from big data, rather than small data.
Pre-training a larger model on a massive dataset is a natural next step in machine learning for spectroscopy. Pre-training has been shown to improve accuracy in computer vision tasks [221]. A massive dataset need not be collected by a single individual or institution. Instead, it can be a collection of community-contributed data, simulated spectra, or some combination of these. Once a pre-trained model is available, transfer learning [93] may be used on a target dataset (for example, by fine-tuning the model on the new dataset). State-of-the-art fine-tuning methods update only a subset of weights or add an adapter layer [112] where weights are learned while the pre-trained model remains fixed.
Figure 7.2: Reconstructions from a prototype multi-sensor auto-encoder. The observed spectra are in blue. Reconstructions are in red. The first row contains XRF, the second row contains LIBS, and the last row contains Vis-NIR-SWIR spectra.
7.2.3 Hyperparameter Optimization and Ensembling

The space of hyperparameters for neural networks is virtually limitless and so may be further and more thoroughly searched to provide better neural network models. Many methods exist to automate HPO [113], and it is an active field of research to develop these methods further (called automatic machine learning, or AutoML [106]). Possible methods include evolutionary algorithms, population-based learning, Bayesian optimization, probabilistic programming, and metaheuristics. Recent developments in ensembling include proposals for how best to choose the ensemble members. HPO explores the hyperparameter search space and trains many neural network configurations, and all but one configuration is kept in standard HPO. Ensembles consisting of unique neural network configurations is one option [89]. This type of ensemble exploits the diversity of models explored in HPO.

Future work may also consider using ensembling to “identify training examples where the individual networks disagree” [132] as a measure of predictive uncertainty. Predictive uncertainty is useful in a live deployment where, when uncertainty is high, a sample may be re-scanned or a request may be made to have ground truth obtained for the sample. Existing solutions might not be applicable in the mining domain, however, because there are further peculiarities to consider. For instance, it is expensive and takes a long time to obtain ground truth. The cost associated with obtaining ground truth should be balanced with the cost of a wrong prediction.

7.2.4 Considerations in Mining Environment

On a mining shovel or conveyor belt, multiple spectra are collected over time. While beyond the scope of this thesis, an interesting avenue of research might consider how variability of these spectra may be indicative of different mineralogies. For example, some minerals are more homogeneous than others, which is not revealed when averages are taken across all the spectra.

Given the expense of geochemical assays, more creative solutions to the limited data problem may need to be considered. Since a conveyor belt is a more controlled environment, it may provide more accurate sensor readings. The sensor data from the belt could be matched with the sensor data from the mining shovel where the rocks came from, and then the shovel’s prediction model may be encouraged (through a penalty term in a neural network, for instance) to more closely match the conveyor’s model. It would also be plausible to convert between the spectra from a shovel to the spectra from a conveyor by using an auto-encoder where the input is shovel spectra and the output is conveyor spectra. If the conveyor belt also has additional spectroscopy types, this can also be utilized. The ideas given
here would require extensive tracking of the movement of rocks within the mine site. Such tracking may be intentionally developed or may come “for free” when automated self-driving hauling trucks become more common.
Bibliography


126


135


[248] S. Wold. Chemometrics; what do we mean with it, and what do we want from it? *Chemometrics and Intelligent Laboratory Systems*, 30(1):


Appendix A

Supplementary Material: Chapter 3

A.1 Full Evaluation Results

48 elements have ground truth, 30 of these were removed from the results because none of the models were able to do better than predicting the average, likely due to the low-abundance of these elements. The results from the remaining elements are shown in Table A.1 and Figure A.1. The removed elements are Ag, As, Ba, Bi, Cd, Ce, Co, Cr, Cu, Ge, In, La, Mn, Na, Nb, Ni, P, Re, Sc, Se, Ta, Te, Th, Ti, Tl, U, V, W, Y, and Zn.
<table>
<thead>
<tr>
<th>Element</th>
<th>Baselines</th>
<th>Neural Networks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OLS</td>
<td>LASSO</td>
</tr>
<tr>
<td>Al</td>
<td>3.29e+0±2.4e-1</td>
<td>2.65e+0±2.6e-1</td>
</tr>
<tr>
<td>Be</td>
<td>1.25e-7±2.2e-8</td>
<td>1.36e-7±2.6e-8</td>
</tr>
<tr>
<td>Ca</td>
<td>8.96e+0±2.5e+0</td>
<td>1.11e+1±2.6e+0</td>
</tr>
<tr>
<td>Cs</td>
<td>1.93e-4±4.5e-5</td>
<td>1.42e-4±3.0e-5</td>
</tr>
<tr>
<td>Fe</td>
<td>7.85e-1±1.6e-1</td>
<td>1.01e+0±1.3e-1</td>
</tr>
<tr>
<td>Ga</td>
<td>3.19e-7±2.2e-8</td>
<td>2.31e-7±1.8e-8</td>
</tr>
<tr>
<td>Hf</td>
<td>1.56e-7±2.1e-8</td>
<td>1.52e-7±1.8e-8</td>
</tr>
<tr>
<td>K</td>
<td>1.10e+0±1.5e-1</td>
<td>1.28e+0±1.2e-1</td>
</tr>
<tr>
<td>Li</td>
<td>1.15e-2±1.6e-3</td>
<td>1.22e-2±1.6e-3</td>
</tr>
<tr>
<td>Mg</td>
<td>1.54e+1±1.2e+0</td>
<td>7.23e+0±5.7e-1</td>
</tr>
<tr>
<td>Mo</td>
<td>2.74e-5±5.7e-6</td>
<td>2.55e-5±6.1e-6</td>
</tr>
<tr>
<td>Pb</td>
<td>3.47e-7±6.4e-8</td>
<td>3.22e-7±5.3e-8</td>
</tr>
<tr>
<td>Rb</td>
<td>1.53e-4±1.9e-5</td>
<td>1.17e-4±1.6e-5</td>
</tr>
<tr>
<td>S</td>
<td>8.46e-1±1.8e-1</td>
<td>9.27e-1±1.4e-1</td>
</tr>
<tr>
<td>Sb</td>
<td>1.13e-6±2.5e-7</td>
<td>9.89e-7±2.3e-7</td>
</tr>
<tr>
<td>Sn</td>
<td>1.51e-8±1.5e-9</td>
<td>1.50e-8±1.6e-9</td>
</tr>
<tr>
<td>Sr</td>
<td>4.79e-4±2.1e-4</td>
<td>5.45e-4±1.5e-4</td>
</tr>
<tr>
<td>Zr</td>
<td>9.42e-4±4.2e-4</td>
<td>1.35e-3±4.8e-4</td>
</tr>
</tbody>
</table>

**Table A.1**: MSE (mean squared error) and standard error reported across cross-validation test sets. The best scores for each element across all models are highlighted.
Figure A.1: MSE and standard error (error bars) for each model and each element.
Appendix B

Supplementary Material:
Chapter 4

B.1 Experiment 1: All Elements

Results for a subset of elements are shown in Chapter 4; results for all 48 elements and for both experiments are provided here. Table B.1 lists the mean RMSE across 100 iterations of cross-validation (10 times 10-fold cross-validation). Paired $t$-test results are given in Tables B.2, B.3, B.4, and B.5. In these tables, 16 elements are left out because no model could beat predicting the average for these elements; the left-out elements are Cd, Ce, Cr, Ge, P, Re, Se, Te, and U.
<table>
<thead>
<tr>
<th>Element</th>
<th>PTA</th>
<th>LASSO</th>
<th>PLS</th>
<th>FitSim</th>
<th>CNN-1</th>
<th>CNN-2</th>
<th>AE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ag</td>
<td>9.76e-6</td>
<td>9.79e-6</td>
<td>9.79e-6</td>
<td>1.19e-5</td>
<td>1.04e-5</td>
<td>9.74e-6</td>
<td>9.58e-6</td>
</tr>
<tr>
<td>Al</td>
<td>1.86e-5</td>
<td>1.86e-5</td>
<td>1.86e-5</td>
<td>1.86e-5</td>
<td>1.86e-5</td>
<td>1.86e-5</td>
<td>1.86e-5</td>
</tr>
<tr>
<td>As</td>
<td>1.75e-7</td>
<td>1.74e-7</td>
<td>1.74e-7</td>
<td>1.74e-7</td>
<td>1.74e-7</td>
<td>1.74e-7</td>
<td>1.74e-7</td>
</tr>
<tr>
<td>Ca</td>
<td>3.92e-6</td>
<td>3.92e-6</td>
<td>3.92e-6</td>
<td>3.92e-6</td>
<td>3.92e-6</td>
<td>3.92e-6</td>
<td>3.92e-6</td>
</tr>
<tr>
<td>Ce</td>
<td>2.64e-4</td>
<td>2.64e-4</td>
<td>2.64e-4</td>
<td>2.64e-4</td>
<td>2.64e-4</td>
<td>2.64e-4</td>
<td>2.64e-4</td>
</tr>
<tr>
<td>Co</td>
<td>3.15e-6</td>
<td>3.15e-6</td>
<td>3.15e-6</td>
<td>3.15e-6</td>
<td>3.15e-6</td>
<td>3.15e-6</td>
<td>3.15e-6</td>
</tr>
<tr>
<td>Cr</td>
<td>1.15e-5</td>
<td>1.15e-5</td>
<td>1.15e-5</td>
<td>1.15e-5</td>
<td>1.15e-5</td>
<td>1.15e-5</td>
<td>1.15e-5</td>
</tr>
<tr>
<td>Cu</td>
<td>6.51e-1</td>
<td>6.51e-1</td>
<td>6.51e-1</td>
<td>6.51e-1</td>
<td>6.51e-1</td>
<td>6.51e-1</td>
<td>6.51e-1</td>
</tr>
<tr>
<td>Ga</td>
<td>5.68e-6</td>
<td>5.68e-6</td>
<td>5.68e-6</td>
<td>5.68e-6</td>
<td>5.68e-6</td>
<td>5.68e-6</td>
<td>5.68e-6</td>
</tr>
<tr>
<td>Ge</td>
<td>5.16e-6</td>
<td>5.16e-6</td>
<td>5.16e-6</td>
<td>5.16e-6</td>
<td>5.16e-6</td>
<td>5.16e-6</td>
<td>5.16e-6</td>
</tr>
<tr>
<td>In</td>
<td>6.01e-6</td>
<td>6.01e-6</td>
<td>6.01e-6</td>
<td>6.01e-6</td>
<td>6.01e-6</td>
<td>6.01e-6</td>
<td>6.01e-6</td>
</tr>
<tr>
<td>K</td>
<td>1.52e-1</td>
<td>1.52e-1</td>
<td>1.52e-1</td>
<td>1.52e-1</td>
<td>1.52e-1</td>
<td>1.52e-1</td>
<td>1.52e-1</td>
</tr>
<tr>
<td>La</td>
<td>1.75e-5</td>
<td>1.75e-5</td>
<td>1.75e-5</td>
<td>1.75e-5</td>
<td>1.75e-5</td>
<td>1.75e-5</td>
<td>1.75e-5</td>
</tr>
<tr>
<td>Li</td>
<td>1.95e-1</td>
<td>1.95e-1</td>
<td>1.95e-1</td>
<td>1.95e-1</td>
<td>1.95e-1</td>
<td>1.95e-1</td>
<td>1.95e-1</td>
</tr>
<tr>
<td>Mg</td>
<td>3.77e-6</td>
<td>3.77e-6</td>
<td>3.77e-6</td>
<td>3.77e-6</td>
<td>3.77e-6</td>
<td>3.77e-6</td>
<td>3.77e-6</td>
</tr>
<tr>
<td>Mn</td>
<td>4.52e-1</td>
<td>4.52e-1</td>
<td>4.52e-1</td>
<td>4.52e-1</td>
<td>4.52e-1</td>
<td>4.52e-1</td>
<td>4.52e-1</td>
</tr>
<tr>
<td>Mo</td>
<td>8.03e-1</td>
<td>8.03e-1</td>
<td>8.03e-1</td>
<td>8.03e-1</td>
<td>8.03e-1</td>
<td>8.03e-1</td>
<td>8.03e-1</td>
</tr>
<tr>
<td>Na</td>
<td>1.09e-2</td>
<td>1.09e-2</td>
<td>1.09e-2</td>
<td>1.09e-2</td>
<td>1.09e-2</td>
<td>1.09e-2</td>
<td>1.09e-2</td>
</tr>
<tr>
<td>Nb</td>
<td>2.38e-3</td>
<td>2.38e-3</td>
<td>2.38e-3</td>
<td>2.38e-3</td>
<td>2.38e-3</td>
<td>2.38e-3</td>
<td>2.38e-3</td>
</tr>
<tr>
<td>Ni</td>
<td>2.11e-5</td>
<td>2.11e-5</td>
<td>2.11e-5</td>
<td>2.11e-5</td>
<td>2.11e-5</td>
<td>2.11e-5</td>
<td>2.11e-5</td>
</tr>
<tr>
<td>P</td>
<td>4.23e-5</td>
<td>4.23e-5</td>
<td>4.23e-5</td>
<td>4.23e-5</td>
<td>4.23e-5</td>
<td>4.23e-5</td>
<td>4.23e-5</td>
</tr>
<tr>
<td>Pb</td>
<td>6.43e-4</td>
<td>6.43e-4</td>
<td>6.43e-4</td>
<td>6.43e-4</td>
<td>6.43e-4</td>
<td>6.43e-4</td>
<td>6.43e-4</td>
</tr>
<tr>
<td>Re</td>
<td>2.02e-4</td>
<td>2.02e-4</td>
<td>2.02e-4</td>
<td>2.02e-4</td>
<td>2.02e-4</td>
<td>2.02e-4</td>
<td>2.02e-4</td>
</tr>
<tr>
<td>Si</td>
<td>1.10e-3</td>
<td>1.10e-3</td>
<td>1.10e-3</td>
<td>1.10e-3</td>
<td>1.10e-3</td>
<td>1.10e-3</td>
<td>1.10e-3</td>
</tr>
<tr>
<td>Sc</td>
<td>3.07e-5</td>
<td>3.07e-5</td>
<td>3.07e-5</td>
<td>3.07e-5</td>
<td>3.07e-5</td>
<td>3.07e-5</td>
<td>3.07e-5</td>
</tr>
<tr>
<td>Sr</td>
<td>1.18e-3</td>
<td>1.18e-3</td>
<td>1.18e-3</td>
<td>1.18e-3</td>
<td>1.18e-3</td>
<td>1.18e-3</td>
<td>1.18e-3</td>
</tr>
<tr>
<td>Ta</td>
<td>2.06e-3</td>
<td>2.06e-3</td>
<td>2.06e-3</td>
<td>2.06e-3</td>
<td>2.06e-3</td>
<td>2.06e-3</td>
<td>2.06e-3</td>
</tr>
<tr>
<td>Ti</td>
<td>1.01e-5</td>
<td>1.01e-5</td>
<td>1.01e-5</td>
<td>1.01e-5</td>
<td>1.01e-5</td>
<td>1.01e-5</td>
<td>1.01e-5</td>
</tr>
<tr>
<td>Tl</td>
<td>3.95e-5</td>
<td>3.95e-5</td>
<td>3.95e-5</td>
<td>3.95e-5</td>
<td>3.95e-5</td>
<td>3.95e-5</td>
<td>3.95e-5</td>
</tr>
<tr>
<td>V</td>
<td>2.73e-3</td>
<td>2.73e-3</td>
<td>2.73e-3</td>
<td>2.73e-3</td>
<td>2.73e-3</td>
<td>2.73e-3</td>
<td>2.73e-3</td>
</tr>
<tr>
<td>W</td>
<td>1.87e-4</td>
<td>1.87e-4</td>
<td>1.87e-4</td>
<td>1.87e-4</td>
<td>1.87e-4</td>
<td>1.87e-4</td>
<td>1.87e-4</td>
</tr>
<tr>
<td>Zn</td>
<td>3.84e-5</td>
<td>3.84e-5</td>
<td>3.84e-5</td>
<td>3.84e-5</td>
<td>3.84e-5</td>
<td>3.84e-5</td>
<td>3.84e-5</td>
</tr>
<tr>
<td>Zr</td>
<td>4.52e-5</td>
<td>4.52e-5</td>
<td>4.52e-5</td>
<td>4.52e-5</td>
<td>4.52e-5</td>
<td>4.52e-5</td>
<td>4.52e-5</td>
</tr>
</tbody>
</table>

**Table B.1:** Scores for all 48 elements. Mean RMSE (root mean squared error) and standard deviation across validation iterations are reported. The best scores for each element are highlighted.
<table>
<thead>
<tr>
<th>Model</th>
<th>Model 2</th>
<th>Ag</th>
<th>Al</th>
<th>As</th>
<th>Ba</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTA</td>
<td>LASSO</td>
<td>≈ (p=1.5e-01)</td>
<td>(p=1.2e-28)</td>
<td>(p=1.2e-02)</td>
<td>(p=1.6e-09)</td>
</tr>
<tr>
<td></td>
<td>PLS</td>
<td>↓ (p=5.3e-07)</td>
<td>↑ (p=4.8e-26)</td>
<td>↑ (p=3.8e-06)</td>
<td>(p=7.2e-03)</td>
</tr>
<tr>
<td></td>
<td>FitSim</td>
<td>≈ (p=1.5e-02)</td>
<td>(p=6.0e-29)</td>
<td>↑ (p=5.0e-16)</td>
<td>↑ (p=6.2e-06)</td>
</tr>
<tr>
<td></td>
<td>CNN-1</td>
<td>↑ (p=3.5e-05)</td>
<td>↑ (p=3.5e-38)</td>
<td>↑ (p=3.4e-25)</td>
<td>(p=4.4e-04)</td>
</tr>
<tr>
<td></td>
<td>CNN-2</td>
<td>≈ (p=1.1e-01)</td>
<td>↑ (p=1.4e-38)</td>
<td>↑ (p=6.2e-25)</td>
<td>(p=5.8e-04)</td>
</tr>
<tr>
<td></td>
<td>SpectraAE</td>
<td>≈ (p=8.4e-01)</td>
<td>(p=2.2e-36)</td>
<td>↑ (p=3.1e-20)</td>
<td>(p=3.6e-02)</td>
</tr>
<tr>
<td>LASSO</td>
<td>PTA</td>
<td>↑ (p=1.5e-01)</td>
<td>(p=1.2e-28)</td>
<td>(p=1.2e-02)</td>
<td>(p=1.6e-09)</td>
</tr>
<tr>
<td></td>
<td>PLS</td>
<td>↓ (p=6.8e-07)</td>
<td>≈ (p=3.0e-02)</td>
<td>↑ (p=1.0e-05)</td>
<td>(p=1.5e-01)</td>
</tr>
<tr>
<td></td>
<td>FitSim</td>
<td>↑ (p=5.0e-02)</td>
<td>(p=6.5e-02)</td>
<td>↑ (p=5.7e-15)</td>
<td>(p=3.2e-01)</td>
</tr>
<tr>
<td></td>
<td>LASSO</td>
<td>↑ (p=1.8e-08)</td>
<td>↑ (p=4.3e-17)</td>
<td>↑ (p=4.7e-24)</td>
<td>(p=7.1e-01)</td>
</tr>
<tr>
<td></td>
<td>CNN-1</td>
<td>↑ (p=3.5e-02)</td>
<td>↑ (p=1.0e-13)</td>
<td>↑ (p=8.1e-24)</td>
<td>(p=6.0e-01)</td>
</tr>
<tr>
<td></td>
<td>CNN-2</td>
<td>↑ (p=9.5e-01)</td>
<td>↑ (p=1.5e-17)</td>
<td>↓ (p=2.1e-01)</td>
<td>(p=1.0e-01)</td>
</tr>
<tr>
<td></td>
<td>SpectraAE</td>
<td>≈ (p=8.4e-01)</td>
<td>(p=2.2e-36)</td>
<td>↑ (p=3.1e-20)</td>
<td>(p=3.6e-02)</td>
</tr>
<tr>
<td>LASSO</td>
<td>PTA</td>
<td>≈ (p=5.5e-01)</td>
<td>(p=1.2e-28)</td>
<td>(p=1.2e-02)</td>
<td>(p=1.6e-09)</td>
</tr>
<tr>
<td></td>
<td>PLS</td>
<td>↓ (p=6.8e-07)</td>
<td>≈ (p=3.0e-02)</td>
<td>↑ (p=1.0e-05)</td>
<td>(p=1.5e-01)</td>
</tr>
<tr>
<td></td>
<td>FitSim</td>
<td>↑ (p=2.1e-02)</td>
<td>(p=6.5e-02)</td>
<td>↑ (p=5.7e-15)</td>
<td>(p=3.2e-01)</td>
</tr>
<tr>
<td></td>
<td>LASSO</td>
<td>↑ (p=1.8e-08)</td>
<td>↑ (p=4.3e-17)</td>
<td>↑ (p=4.7e-24)</td>
<td>(p=7.1e-01)</td>
</tr>
<tr>
<td></td>
<td>CNN-1</td>
<td>↑ (p=3.5e-02)</td>
<td>↑ (p=1.0e-13)</td>
<td>↑ (p=8.1e-24)</td>
<td>(p=6.0e-01)</td>
</tr>
<tr>
<td></td>
<td>CNN-2</td>
<td>↑ (p=9.5e-01)</td>
<td>↑ (p=1.5e-17)</td>
<td>↓ (p=2.1e-01)</td>
<td>(p=1.0e-01)</td>
</tr>
<tr>
<td></td>
<td>SpectraAE</td>
<td>≈ (p=8.4e-01)</td>
<td>(p=2.2e-36)</td>
<td>↑ (p=3.1e-20)</td>
<td>(p=3.6e-02)</td>
</tr>
</tbody>
</table>

Table B.2: Paired t-test results on 32 of 48 elements; 16 elements are left out because no model achieved statistically significant improvement over predicting the average (PtA). This is part 1/8. The paired t-test operates on pairs, so all pairs of models are given here. A “↓” indicates that Model 1 achieved lower mean RMSE than Model 2 and the result is statistically significant with the corresponding p-value listed in parentheses. A “↑” means Model 1’s RMSE is higher than Model 2’s RMSE. A “≈” indicates that we cannot reject the null hypothesis of equal averages.
<table>
<thead>
<tr>
<th>Model 1</th>
<th>Model 2</th>
<th>Be</th>
<th>Bi</th>
<th>Ca</th>
<th>Cu</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LASSO</td>
<td>↑ (p=7.9e-18)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLS</td>
<td>≈ (p=6.2e-02)</td>
<td>≈ (p=7.2e-04)</td>
<td>↑ (p=4.4e-19)</td>
<td>↑ (p=6.7e-06)</td>
<td></td>
</tr>
<tr>
<td>FitSim</td>
<td>≈ (p=6.2e-03)</td>
<td>≈ (p=8.3e-04)</td>
<td>↑ (p=6.6e-17)</td>
<td>↑ (p=2.4e-05)</td>
<td></td>
</tr>
<tr>
<td>CNN-1</td>
<td>↑ (p=2.3e-23)</td>
<td>≈ (p=1.1e-02)</td>
<td>↑ (p=1.4e-22)</td>
<td>↑ (p=1.2e-08)</td>
<td></td>
</tr>
<tr>
<td>CNN-2</td>
<td>↑ (p=1.9e-21)</td>
<td>≈ (p=2.4e-02)</td>
<td>↑ (p=2.3e-23)</td>
<td>↑ (p=1.4e-12)</td>
<td></td>
</tr>
<tr>
<td>SpectraAE</td>
<td>↑ (p=2.2e-22)</td>
<td>≈ (p=1.0e-02)</td>
<td>↑ (p=2.3e-17)</td>
<td>↑ (p=1.5e-08)</td>
<td></td>
</tr>
<tr>
<td>LASSO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLS</td>
<td>↓ (p=7.9e-18)</td>
<td>↓ (p=6.2e-08)</td>
<td>↓ (p=1.1e-01)</td>
<td>↓ (p=1.2e-07)</td>
<td></td>
</tr>
<tr>
<td>FitSim</td>
<td>≈ (p=2.3e-04)</td>
<td>≈ (p=9.3e-01)</td>
<td>↑ (p=2.3e-06)</td>
<td>≈ (p=1.2e-01)</td>
<td></td>
</tr>
<tr>
<td>CNN-1</td>
<td>↑ (p=4.3e-09)</td>
<td>≈ (p=7.6e-01)</td>
<td>↑ (p=1.0e-18)</td>
<td>↑ (p=4.2e-06)</td>
<td></td>
</tr>
<tr>
<td>CNN-2</td>
<td>↑ (p=2.8e-07)</td>
<td>≈ (p=5.5e-01)</td>
<td>↑ (p=2.6e-01)</td>
<td>↑ (p=5.5e-01)</td>
<td></td>
</tr>
<tr>
<td>SpectraAE</td>
<td>↑ (p=2.5e-06)</td>
<td>≈ (p=4.6e-01)</td>
<td>≈ (p=4.6e-02)</td>
<td>≈ (p=4.4e-05)</td>
<td></td>
</tr>
<tr>
<td>PLS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LASSO</td>
<td>≈ (p=3.8e-07)</td>
<td>≈ (p=2.6e-01)</td>
<td>≈ (p=7.5e-05)</td>
<td>≈ (1.1e-01)</td>
<td></td>
</tr>
<tr>
<td>FitSim</td>
<td>≈ (p=5.5e-01)</td>
<td>≈ (p=1.1e-01)</td>
<td>≈ (p=6.6e-03)</td>
<td>≈ (8.4e-02)</td>
<td></td>
</tr>
<tr>
<td>CNN-1</td>
<td>↑ (p=2.9e-14)</td>
<td>≈ (p=2.4e-01)</td>
<td>↑ (p=1.2e-11)</td>
<td>↑ (p=3.2e-04)</td>
<td></td>
</tr>
<tr>
<td>CNN-2</td>
<td>↑ (p=1.3e-13)</td>
<td>≈ (p=3.3e-01)</td>
<td>≈ (p=2.6e-01)</td>
<td>↑ (3.2e-08)</td>
<td></td>
</tr>
<tr>
<td>SpectraAE</td>
<td>↑ (p=3.2e-12)</td>
<td>≈ (p=5.6e-02)</td>
<td>≈ (p=3.0e-02)</td>
<td>≈ (5.5e-03)</td>
<td></td>
</tr>
<tr>
<td>FitSim</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNN-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LASSO</td>
<td>↑ (p=4.3e-09)</td>
<td>≈ (p=7.6e-01)</td>
<td>↓ (p=1.0e-18)</td>
<td>↑ (p=4.2e-06)</td>
<td></td>
</tr>
<tr>
<td>PLS</td>
<td>↓ (p=2.9e-14)</td>
<td>↑ (p=2.4e-01)</td>
<td>↓ (p=1.2e-11)</td>
<td>≈ (1.3e-04)</td>
<td></td>
</tr>
<tr>
<td>FitSim</td>
<td>↑ (p=3.4e-08)</td>
<td>≈ (p=7.6e-01)</td>
<td>↑ (p=1.0e-13)</td>
<td>↑ (3.2e-06)</td>
<td></td>
</tr>
<tr>
<td>CNN-2</td>
<td>↑ (p=6.3e-08)</td>
<td>≈ (p=4.9e-01)</td>
<td>≈ (p=4.1e-03)</td>
<td>↑ (6.6e-11)</td>
<td></td>
</tr>
<tr>
<td>SpectraAE</td>
<td>↑ (p=6.3e-07)</td>
<td>≈ (p=3.8e-01)</td>
<td>≈ (p=5.0e-01)</td>
<td>≈ (1.6e-04)</td>
<td></td>
</tr>
<tr>
<td>CNN-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LASSO</td>
<td>≈ (p=5.5e-01)</td>
<td>≈ (p=7.6e-01)</td>
<td>≈ (1.5e-02)</td>
<td>≈ (5.6e-02)</td>
<td></td>
</tr>
<tr>
<td>PLS</td>
<td>≈ (p=1.5e-01)</td>
<td>≈ (p=7.6e-01)</td>
<td>↓ (p=2.6e-12)</td>
<td>≈ (1.1e-01)</td>
<td></td>
</tr>
<tr>
<td>FitSim</td>
<td>≈ (p=5.5e-01)</td>
<td>≈ (p=7.6e-01)</td>
<td>≈ (1.5e-02)</td>
<td>≈ (5.6e-02)</td>
<td></td>
</tr>
<tr>
<td>CNN-1</td>
<td>≈ (p=5.5e-01)</td>
<td>≈ (p=7.6e-01)</td>
<td>≈ (1.5e-02)</td>
<td>≈ (5.6e-02)</td>
<td></td>
</tr>
<tr>
<td>CNN-2</td>
<td>≈ (p=5.5e-01)</td>
<td>≈ (p=7.6e-01)</td>
<td>≈ (1.5e-02)</td>
<td>≈ (5.6e-02)</td>
<td></td>
</tr>
</tbody>
</table>

Table B.3: Paired t-test results part 2/8.
<table>
<thead>
<tr>
<th>Model 1</th>
<th>Model 2</th>
<th>Cs</th>
<th>Fw</th>
<th>Ga</th>
<th>Hf</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLTA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LASSO</td>
<td>(p=3.1e-39)</td>
<td>(p=5.0e-21)</td>
<td>(p=5.0e-39)</td>
<td>(p=2.6e-19)</td>
<td></td>
</tr>
<tr>
<td>PLS</td>
<td>(p=6.3e-27)</td>
<td>(p=2.6e-20)</td>
<td>(p=2.6e-27)</td>
<td>(p=2.6e-14)</td>
<td></td>
</tr>
<tr>
<td>FitSim</td>
<td>(p=1.6e-37)</td>
<td>(p=2.5e-05)</td>
<td>(p=1.2e-29)</td>
<td>(p=1.4e-12)</td>
<td></td>
</tr>
<tr>
<td>CNN-1</td>
<td>(p=3.6e-41)</td>
<td>(p=5.0e-27)</td>
<td>(p=1.5e-32)</td>
<td>(p=3.5e-16)</td>
<td></td>
</tr>
<tr>
<td>CNN-2</td>
<td>(p=7.6e-41)</td>
<td>(p=1.4e-24)</td>
<td>(p=4.6e-36)</td>
<td>(p=2.2e-19)</td>
<td></td>
</tr>
<tr>
<td>SpectraAE</td>
<td>(p=4.0e-46)</td>
<td>(p=3.6e-24)</td>
<td>(p=8.8e-35)</td>
<td>(p=3.1e-16)</td>
<td></td>
</tr>
<tr>
<td>LASSO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLTA</td>
<td>(p=3.1e-39)</td>
<td>(p=5.0e-21)</td>
<td>(p=5.0e-39)</td>
<td>(p=2.6e-19)</td>
<td></td>
</tr>
<tr>
<td>PLS</td>
<td>(p=5.3e-01)</td>
<td>(p=2.1e-05)</td>
<td>(p=5.0e-05)</td>
<td>(p=1.1e-11)</td>
<td></td>
</tr>
<tr>
<td>FitSim</td>
<td>(p=3.5e-04)</td>
<td>(p=6.9e-03)</td>
<td>(p=8.4e-04)</td>
<td>(p=1.6e-08)</td>
<td></td>
</tr>
<tr>
<td>CNN-1</td>
<td>(p=3.9e-11)</td>
<td>(p=1.7e-12)</td>
<td>(p=2.6e-11)</td>
<td>(p=1.5e-13)</td>
<td></td>
</tr>
<tr>
<td>CNN-2</td>
<td>(p=2.5e-09)</td>
<td>(p=3.0e-08)</td>
<td>(p=3.4e-12)</td>
<td>(p=3.1e-16)</td>
<td></td>
</tr>
<tr>
<td>SpectraAE</td>
<td>(p=9.3e-16)</td>
<td>(p=3.8e-09)</td>
<td>(p=1.2e-13)</td>
<td>(p=1.5e-13)</td>
<td></td>
</tr>
<tr>
<td>PLS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LASSO</td>
<td>(p=5.1e-27)</td>
<td>(p=2.6e-20)</td>
<td>(p=2.6e-27)</td>
<td>(p=2.6e-14)</td>
<td></td>
</tr>
<tr>
<td>FitSim</td>
<td>(p=3.8e-04)</td>
<td>(p=6.9e-03)</td>
<td>(p=8.4e-04)</td>
<td>(p=1.6e-08)</td>
<td></td>
</tr>
<tr>
<td>CNN-1</td>
<td>(p=2.5e-12)</td>
<td>(p=2.5e-10)</td>
<td>(p=2.1e-10)</td>
<td>(p=7.2e-01)</td>
<td></td>
</tr>
<tr>
<td>CNN-2</td>
<td>(p=5.3e-09)</td>
<td>(p=3.8e-08)</td>
<td>(p=3.4e-12)</td>
<td>(p=3.1e-16)</td>
<td></td>
</tr>
<tr>
<td>SpectraAE</td>
<td>(p=3.8e-09)</td>
<td>(p=1.7e-09)</td>
<td>(p=8.8e-07)</td>
<td>(p=8.1e-04)</td>
<td></td>
</tr>
<tr>
<td>FitSim</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLS</td>
<td>(p=4.1e-07)</td>
<td>(p=2.7e-13)</td>
<td>(p=8.4e-08)</td>
<td>(p=5.8e-03)</td>
<td></td>
</tr>
<tr>
<td>CNN-1</td>
<td>(p=9.3e-01)</td>
<td>(p=5.0e-02)</td>
<td>(p=1.5e-01)</td>
<td>(p=2.1e-02)</td>
<td></td>
</tr>
<tr>
<td>CNN-2</td>
<td>(p=3.8e-01)</td>
<td>(p=2.1e-01)</td>
<td>(p=3.0e-02)</td>
<td>(p=2.1e-04)</td>
<td></td>
</tr>
<tr>
<td>SpectraAE</td>
<td>(p=3.8e-01)</td>
<td>(p=2.1e-01)</td>
<td>(p=3.0e-02)</td>
<td>(p=2.1e-04)</td>
<td></td>
</tr>
<tr>
<td>CNN-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLTA</td>
<td>(p=4.0e-46)</td>
<td>(p=3.6e-24)</td>
<td>(p=8.8e-35)</td>
<td>(p=3.1e-15)</td>
<td></td>
</tr>
<tr>
<td>LASSO</td>
<td>(p=9.3e-16)</td>
<td>(p=3.8e-09)</td>
<td>(p=1.2e-13)</td>
<td>(p=1.5e-13)</td>
<td></td>
</tr>
<tr>
<td>PLS</td>
<td>(p=3.6e-08)</td>
<td>(p=1.9e-06)</td>
<td>(p=8.4e-12)</td>
<td>(p=4.7e-02)</td>
<td></td>
</tr>
<tr>
<td>FitSim</td>
<td>(p=3.8e-07)</td>
<td>(p=3.8e-12)</td>
<td>(p=3.8e-09)</td>
<td>(p=3.8e-01)</td>
<td></td>
</tr>
<tr>
<td>CNN-1</td>
<td>(p=4.6e-02)</td>
<td>(p=2.9e-01)</td>
<td>(p=1.0e+00)</td>
<td>(p=1.2e-01)</td>
<td></td>
</tr>
<tr>
<td>CNN-2</td>
<td>(p=9.3e-02)</td>
<td>(p=3.8e-01)</td>
<td>(p=2.9e-01)</td>
<td>(p=1.2e-01)</td>
<td></td>
</tr>
</tbody>
</table>

Table B.4: Paired t-test results part 3/8.
<table>
<thead>
<tr>
<th>Model 1</th>
<th>Model 2</th>
<th>In</th>
<th>K</th>
<th>Li</th>
<th>Mg</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLA</td>
<td></td>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
</tr>
<tr>
<td>LASSO</td>
<td></td>
<td><img src="image5.png" alt="Image" /></td>
<td><img src="image6.png" alt="Image" /></td>
<td><img src="image7.png" alt="Image" /></td>
<td><img src="image8.png" alt="Image" /></td>
</tr>
<tr>
<td>PLS</td>
<td></td>
<td><img src="image9.png" alt="Image" /></td>
<td><img src="image10.png" alt="Image" /></td>
<td><img src="image11.png" alt="Image" /></td>
<td><img src="image12.png" alt="Image" /></td>
</tr>
<tr>
<td>FitSim</td>
<td></td>
<td><img src="image13.png" alt="Image" /></td>
<td><img src="image14.png" alt="Image" /></td>
<td><img src="image15.png" alt="Image" /></td>
<td><img src="image16.png" alt="Image" /></td>
</tr>
<tr>
<td>CNN-1</td>
<td></td>
<td><img src="image17.png" alt="Image" /></td>
<td><img src="image18.png" alt="Image" /></td>
<td><img src="image19.png" alt="Image" /></td>
<td><img src="image20.png" alt="Image" /></td>
</tr>
<tr>
<td>CNN-2</td>
<td></td>
<td><img src="image21.png" alt="Image" /></td>
<td><img src="image22.png" alt="Image" /></td>
<td><img src="image23.png" alt="Image" /></td>
<td><img src="image24.png" alt="Image" /></td>
</tr>
</tbody>
</table>

Table B.5: Paired t-test results part 4/8.
<table>
<thead>
<tr>
<th>Model 1</th>
<th>Model 2</th>
<th>Mn</th>
<th>Mn</th>
<th>Na</th>
<th>Pb</th>
</tr>
</thead>
<tbody>
<tr>
<td>LASSO</td>
<td>↑ (p=1.5e-15)</td>
<td>↑ (p=1.6e-26)</td>
<td>≈ (p=1.0e+00)</td>
<td>↑ (p=1.4e-15)</td>
<td></td>
</tr>
<tr>
<td>PLS</td>
<td>≈ (p=1.2e-02)</td>
<td>↑ (p=8.3e-19)</td>
<td>≈ (p=7.6e-05)</td>
<td>↑ (p=7.0e-16)</td>
<td></td>
</tr>
<tr>
<td>FitSim</td>
<td>↑ (p=6.1e-09)</td>
<td>↑ (p=9.1e-17)</td>
<td>↑ (p=1.8e-08)</td>
<td>↑ (p=3.3e-14)</td>
<td></td>
</tr>
<tr>
<td>CNN-1</td>
<td>≈ (p=6.9e-01)</td>
<td>↑ (p=1.7e-27)</td>
<td>≈ (p=1.5e-04)</td>
<td>↑ (p=3.2e-19)</td>
<td></td>
</tr>
<tr>
<td>CNN-2</td>
<td>≈ (p=7.1e-04)</td>
<td>↑ (p=3.7e-35)</td>
<td>↑ (p=1.3e-07)</td>
<td>↑ (p=1.3e-26)</td>
<td></td>
</tr>
<tr>
<td>SpectraAE</td>
<td>≈ (p=2.0e-01)</td>
<td>↑ (p=4.4e-29)</td>
<td>↑ (p=1.0e-06)</td>
<td>↑ (p=3.3e-21)</td>
<td></td>
</tr>
<tr>
<td>LASSO</td>
<td>≈ (p=1.5e-09)</td>
<td>↓ (p=1.6e-28)</td>
<td>≈ (p=1.0e+00)</td>
<td>↓ (p=1.4e-15)</td>
<td></td>
</tr>
<tr>
<td>PLS</td>
<td>≈ (p=2.6e-02)</td>
<td>↑ (p=1.1e-08)</td>
<td>≈ (p=7.6e-05)</td>
<td>↑ (p=1.0e-07)</td>
<td></td>
</tr>
<tr>
<td>FitSim</td>
<td>≈ (p=2.3e-01)</td>
<td>↑ (p=2.8e-02)</td>
<td>↑ (p=1.8e-08)</td>
<td>≈ (p=1.8e-03)</td>
<td></td>
</tr>
<tr>
<td>CNN-1</td>
<td>≈ (p=6.8e-02)</td>
<td>≈ (p=1.0e-03)</td>
<td>≈ (p=1.5e-04)</td>
<td>↑ (p=1.8e-11)</td>
<td></td>
</tr>
<tr>
<td>CNN-2</td>
<td>≈ (p=9.7e-01)</td>
<td>↑ (p=3.1e-09)</td>
<td>↑ (p=1.3e-07)</td>
<td>↑ (p=1.4e-15)</td>
<td></td>
</tr>
<tr>
<td>SpectraAE</td>
<td>≈ (p=9.2e-01)</td>
<td>↑ (p=2.3e-08)</td>
<td>↑ (p=1.0e-06)</td>
<td>↑ (p=4.7e-14)</td>
<td></td>
</tr>
</tbody>
</table>

| LASSO        | ≈ (p=1.5e-15) | ↑ (p=1.6e-26) | ≈ (p=1.0e+00) | ↑ (p=1.4e-15) |
| PLS          | ≈ (p=2.6e-02) | ↑ (p=1.1e-08) | ≈ (p=7.6e-05) | ↑ (p=1.0e-07) |
| FitSim       | ≈ (p=2.3e-01) | ↑ (p=2.8e-02) | ↑ (p=1.8e-08) | ≈ (p=1.8e-03) |
| CNN-1        | ≈ (p=6.8e-02) | ↑ (p=1.3e-05) | ↑ (p=4.6e-04) | ↑ (p=3.2e-11) |
| CNN-2        | ≈ (p=9.7e-01) | ↑ (p=7.1e-16) | ≈ (p=3.9e-06) | ↑ (p=2.3e-11) |
| SpectraAE    | ≈ (p=9.2e-01) | ↑ (p=7.1e-16) | ≈ (p=3.9e-06) | ↑ (p=2.3e-11) |

| LASSO        | ≈ (p=1.5e-15) | ↑ (p=1.6e-26) | ≈ (p=1.0e+00) | ↑ (p=1.4e-15) |
| PLS          | ≈ (p=2.6e-02) | ↑ (p=1.1e-08) | ≈ (p=7.6e-05) | ↑ (p=1.0e-07) |
| FitSim       | ≈ (p=2.3e-01) | ↑ (p=2.8e-02) | ↑ (p=1.8e-08) | ≈ (p=1.8e-03) |
| CNN-1        | ≈ (p=6.8e-02) | ↑ (p=1.3e-05) | ↑ (p=4.6e-04) | ↑ (p=3.2e-11) |
| CNN-2        | ≈ (p=9.7e-01) | ↑ (p=7.1e-16) | ≈ (p=3.9e-06) | ↑ (p=2.3e-11) |
| SpectraAE    | ≈ (p=9.2e-01) | ↑ (p=7.1e-16) | ≈ (p=3.9e-06) | ↑ (p=2.3e-11) |

| CNN-1        | ↑ (p=5.9e-01) | ↑ (p=1.1e-21) | ≈ (p=1.0e-04) | ↑ (p=3.2e-15) |
| LASSO        | ≈ (p=6.8e-02) | ≈ (p=1.0e-03) | ≈ (p=1.5e-04) | ↓ (p=1.8e-11) |
| PLS          | ≈ (p=3.6e-01) | ↓ (p=3.0e-13) | ≈ (p=6.6e-02) | ↓ (p=7.2e-09) |
| FitSim       | ≈ (p=2.2e-02) | ↑ (p=1.7e-04) | ≈ (p=2.1e-01) | ↑ (p=2.3e-13) |
| CNN-2        | ≈ (p=7.1e-03) | ≈ (p=6.1e-03) | ≈ (p=2.0e-03) | ≈ (3.2e-03) |
| SpectraAE    | ≈ (p=3.6e-01) | ≈ (p=1.0e-03) | ≈ (p=1.5e-04) | ↓ (p=1.8e-11) |

| LASSO        | ≈ (p=9.7e-01) | ↑ (p=3.1e-09) | ↓ (p=3.3e-07) | ↓ (p=1.4e-19) |
| PLS          | ≈ (p=2.0e-02) | ↑ (p=7.2e-17) | ≈ (p=5.6e-05) | ↓ (p=9.2e-16) |
| FitSim       | ≈ (p=3.6e-01) | ↑ (p=1.7e-04) | ≈ (p=2.1e-01) | ↓ (p=2.3e-13) |
| CNN-1        | ≈ (p=7.1e-03) | ≈ (p=6.1e-03) | ≈ (p=2.0e-03) | ≈ (3.2e-03) |
| CNN-2        | ≈ (p=3.6e-01) | ≈ (p=1.0e-03) | ≈ (p=1.5e-04) | ↓ (p=1.8e-11) |
| SpectraAE    | ≈ (p=3.6e-01) | ≈ (p=1.0e-03) | ≈ (p=1.5e-04) | ↓ (p=1.8e-11) |

Table B.6: Paired t-test results part 5/8.
<table>
<thead>
<tr>
<th>Model 1</th>
<th>Model 2</th>
<th>Rh</th>
<th>S</th>
<th>Sb</th>
<th>Sn</th>
</tr>
</thead>
<tbody>
<tr>
<td>LASSO</td>
<td>↑</td>
<td>(p=7.7e-48)</td>
<td>≈</td>
<td>(p=5.9e-01)</td>
<td>≈</td>
</tr>
<tr>
<td>PLS</td>
<td>↑</td>
<td>(p=4.0e-48)</td>
<td>↑</td>
<td>(p=1.0e-04)</td>
<td>↑</td>
</tr>
<tr>
<td>FitSim</td>
<td>↑</td>
<td>(p=3.2e-16)</td>
<td>↑</td>
<td>(p=1.4e-06)</td>
<td>↑</td>
</tr>
<tr>
<td>CNN-1</td>
<td>↑</td>
<td>(p=6.4e-49)</td>
<td>↑</td>
<td>(p=6.7e-10)</td>
<td>↑</td>
</tr>
<tr>
<td>CNN-2</td>
<td>↑</td>
<td>(p=8.5e-49)</td>
<td>↑</td>
<td>(p=6.7e-10)</td>
<td>↑</td>
</tr>
<tr>
<td>SpectraAE</td>
<td>↑</td>
<td>(p=5.6e-49)</td>
<td>↑</td>
<td>(p=5.4e-08)</td>
<td>↑</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model 1</th>
<th>Model 2</th>
<th>Rh</th>
<th>S</th>
<th>Sb</th>
<th>Sn</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLA</td>
<td>↓</td>
<td>(p=7.7e-48)</td>
<td>↓</td>
<td>(p=5.4e-08)</td>
<td>↓</td>
</tr>
<tr>
<td>LASSO</td>
<td>↓</td>
<td>(p=3.6e-16)</td>
<td>↓</td>
<td>(p=3.1e-22)</td>
<td>↓</td>
</tr>
<tr>
<td>PLS</td>
<td>≈</td>
<td>(p=3.4e-02)</td>
<td>≈</td>
<td>(p=1.7e-08)</td>
<td>≈</td>
</tr>
<tr>
<td>FitSim</td>
<td>≈</td>
<td>(p=5.2e-01)</td>
<td>≈</td>
<td>(p=1.5e-07)</td>
<td>≈</td>
</tr>
<tr>
<td>CNN-1</td>
<td>≈</td>
<td>(p=7.1e-01)</td>
<td>≈</td>
<td>(p=8.1e-07)</td>
<td>≈</td>
</tr>
<tr>
<td>CNN-2</td>
<td>≈</td>
<td>(p=2.6e-01)</td>
<td>≈</td>
<td>(p=8.1e-07)</td>
<td>≈</td>
</tr>
<tr>
<td>SpectraAE</td>
<td>≈</td>
<td>(p=1.4e-07)</td>
<td>≈</td>
<td>(p=1.5e-07)</td>
<td>≈</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model 1</th>
<th>Model 2</th>
<th>Rh</th>
<th>S</th>
<th>Sb</th>
<th>Sn</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLA</td>
<td>↓</td>
<td>(p=7.7e-48)</td>
<td>↓</td>
<td>(p=5.4e-08)</td>
<td>↓</td>
</tr>
<tr>
<td>LASSO</td>
<td>↓</td>
<td>(p=3.6e-16)</td>
<td>↓</td>
<td>(p=3.1e-22)</td>
<td>↓</td>
</tr>
<tr>
<td>PLS</td>
<td>≈</td>
<td>(p=3.4e-02)</td>
<td>≈</td>
<td>(p=1.7e-08)</td>
<td>≈</td>
</tr>
<tr>
<td>FitSim</td>
<td>≈</td>
<td>(p=5.2e-01)</td>
<td>≈</td>
<td>(p=1.5e-07)</td>
<td>≈</td>
</tr>
<tr>
<td>CNN-1</td>
<td>≈</td>
<td>(p=7.1e-01)</td>
<td>≈</td>
<td>(p=8.1e-07)</td>
<td>≈</td>
</tr>
<tr>
<td>CNN-2</td>
<td>≈</td>
<td>(p=2.6e-01)</td>
<td>≈</td>
<td>(p=8.1e-07)</td>
<td>≈</td>
</tr>
<tr>
<td>SpectraAE</td>
<td>≈</td>
<td>(p=1.4e-07)</td>
<td>≈</td>
<td>(p=1.5e-07)</td>
<td>≈</td>
</tr>
</tbody>
</table>

Table B.7: Paired t-test results part 6/8.
<table>
<thead>
<tr>
<th>Model 1</th>
<th>Model 2</th>
<th>St</th>
<th>Ta</th>
<th>Th</th>
<th>Ti</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LASSO</td>
<td>(p=2.3e-36)</td>
<td>(p=1.6e-18)</td>
<td>(p=4.0e-14)</td>
<td>(p=2.8e-23)</td>
<td></td>
</tr>
<tr>
<td>PLS</td>
<td>(p=1.3e-30)</td>
<td>(p=5.2e-02)</td>
<td>(p=1.6e-03)</td>
<td>(p=7.4e-06)</td>
<td></td>
</tr>
<tr>
<td>Fitsim</td>
<td>(p=4.6e-13)</td>
<td>(p=5.4e-05)</td>
<td>(p=8.8e-04)</td>
<td>(p=3.4e-05)</td>
<td></td>
</tr>
<tr>
<td>CNN-1</td>
<td>(p=1.5e-37)</td>
<td>(p=2.2e-10)</td>
<td>(p=7.8e-01)</td>
<td>(p=5.1e-05)</td>
<td></td>
</tr>
<tr>
<td>CNN-2</td>
<td>(p=2.5e-35)</td>
<td>(p=1.1e-08)</td>
<td>(p=1.8e-01)</td>
<td>(p=1.7e-01)</td>
<td></td>
</tr>
<tr>
<td>SpectraE</td>
<td>(p=5.7e-36)</td>
<td>(p=3.7e-06)</td>
<td>(p=4.6e-01)</td>
<td>(p=5.3e-01)</td>
<td></td>
</tr>
<tr>
<td>PLS</td>
<td>(p=2.6e-36)</td>
<td>(p=1.6e-18)</td>
<td>(p=4.0e-14)</td>
<td>(p=2.8e-23)</td>
<td></td>
</tr>
<tr>
<td>LASSO</td>
<td>(p=3.8e-03)</td>
<td>(p=5.7e-03)</td>
<td>(p=1.1e-01)</td>
<td>(p=1.4e-01)</td>
<td></td>
</tr>
<tr>
<td>Fitsim</td>
<td>(p=6.3e-30)</td>
<td>(p=1.6e-10)</td>
<td>(p=1.2e-01)</td>
<td>(p=1.7e-01)</td>
<td></td>
</tr>
<tr>
<td>CNN-1</td>
<td>(p=7.6e-10)</td>
<td>(p=1.7e-10)</td>
<td>(p=9.0e-02)</td>
<td>(p=2.3e-02)</td>
<td></td>
</tr>
<tr>
<td>CNN-2</td>
<td>(p=4.6e-05)</td>
<td>(p=1.6e-01)</td>
<td>(p=2.3e-02)</td>
<td>(p=4.5e-01)</td>
<td></td>
</tr>
<tr>
<td>SpectraE</td>
<td>(p=4.6e-07)</td>
<td>(p=4.6e-01)</td>
<td>(p=4.6e-02)</td>
<td>(p=5.3e-01)</td>
<td></td>
</tr>
<tr>
<td>Fitsim</td>
<td>(p=1.3e-30)</td>
<td>(p=1.6e-10)</td>
<td>(p=1.2e-01)</td>
<td>(p=1.7e-01)</td>
<td></td>
</tr>
<tr>
<td>CNN-1</td>
<td>(p=7.6e-10)</td>
<td>(p=1.7e-10)</td>
<td>(p=9.0e-02)</td>
<td>(p=2.3e-02)</td>
<td></td>
</tr>
<tr>
<td>CNN-2</td>
<td>(p=2.7e-27)</td>
<td>(p=1.1e-04)</td>
<td>(p=4.2e-04)</td>
<td>(p=1.2e-06)</td>
<td></td>
</tr>
<tr>
<td>SpectraE</td>
<td>(p=7.6e-07)</td>
<td>(p=1.2e-03)</td>
<td>(p=2.3e-03)</td>
<td>(p=3.6e-10)</td>
<td></td>
</tr>
<tr>
<td>CNN-1</td>
<td>(p=1.1e-35)</td>
<td>(p=1.5e-10)</td>
<td>(p=7.8e-01)</td>
<td>(p=5.4e-01)</td>
<td></td>
</tr>
<tr>
<td>CNN-2</td>
<td>(p=2.8e-36)</td>
<td>(p=1.6e-10)</td>
<td>(p=7.8e-01)</td>
<td>(p=5.4e-01)</td>
<td></td>
</tr>
<tr>
<td>SpectraE</td>
<td>(p=4.6e-07)</td>
<td>(p=4.6e-01)</td>
<td>(p=5.8e-02)</td>
<td>(p=5.3e-01)</td>
<td></td>
</tr>
<tr>
<td>CNN-1</td>
<td>(p=3.6e-01)</td>
<td>(p=3.2e-06)</td>
<td>(p=6.8e-01)</td>
<td>(p=5.8e-02)</td>
<td></td>
</tr>
<tr>
<td>CNN-2</td>
<td>(p=5.6e-01)</td>
<td>(p=7.0e-06)</td>
<td>(p=1.0e-00)</td>
<td>(p=6.8e-02)</td>
<td></td>
</tr>
</tbody>
</table>

Table B.8: Paired t-test results part 7/8.
<table>
<thead>
<tr>
<th></th>
<th>Model 1</th>
<th>Model 2</th>
<th>V</th>
<th>Y</th>
<th>Zn</th>
<th>Sr</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLA LASSO</td>
<td>(p=4.9e-09) ↑ ≈ (p=1.0e-01) ↑ (p=1.5e-07) ↑ (p=1.3e-11)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLA PLS</td>
<td>(p=6.6e-06) ↑ ≈ (p=3.8e-02) ↑ (p=1.6e-01) ↑ (p=4.1e-14)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLA FitSim</td>
<td>(p=1.1e-01) ↑ ≈ (p=1.6e-02) ↑ (p=2.6e-07) ↑ (p=2.5e-16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLA CNN-1</td>
<td>(p=1.6e-03) ≈ (p=2.5e-03) ≈ (p=3.1e-14) ≈ (p=2.6e-21)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLA CNN-2</td>
<td>(p=5.6e-15) ↑ (p=7.5e-14) ↑ (p=1.5e-18) ↑ (p=2.2e-24)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLA SpectraAE</td>
<td>(p=6.6e-11) ↑ (p=3.0e-05) ↑ (p=1.7e-11) ↑ (p=1.7e-20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LASSO PLA</td>
<td>(p=4.9e-09) ↓ ≈ (p=1.0e-01) ↓ (p=1.5e-07) ↓ (p=1.3e-11)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LASSO PLS</td>
<td>(p=9.1e-01) ↑ ≈ (p=3.8e-02) ↑ (p=7.6e-06) ↑ (p=2.1e-07)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LASSO FitSim</td>
<td>(p=1.6e-02) ≈ (p=1.6e-02) ≈ (p=1.3e-03) ↑ (p=1.6e-18)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LASSO CNN-1</td>
<td>(p=3.6e-01) ≈ (p=2.3e-03) ↑ (p=1.1e-09) ↑ (p=3.3e-23)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LASSO CNN-2</td>
<td>(p=1.0e-10) ↑ (p=7.5e-14) ↑ (p=6.8e-09) ↑ (p=6.8e-59)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LASSO SpectraAE</td>
<td>(p=3.0e-09) ↑ (p=3.0e-05) ↑ (p=1.1e-05) ↑ (p=1.7e-20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLS PLA</td>
<td>(p=6.6e-06) ↓ ≈ (p=3.8e-02) ↓ (p=1.6e-01) ↓ (p=4.1e-14)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLS LASSO</td>
<td>(p=9.1e-01) ↑ ≈ (p=3.8e-02) ↑ (p=7.6e-06) ↑ (p=2.1e-07)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLS FitSim</td>
<td>(p=1.6e-02) ≈ (p=1.6e-02) ≈ (p=1.3e-03) ↑ (p=1.6e-18)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLS CNN-1</td>
<td>(p=3.6e-01) ≈ (p=1.4e-04) ≈ (p=1.9e-16) ≈ (p=9.1e-01)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLS CNN-2</td>
<td>(p=5.6e-15) ↑ (p=1.5e-18) ↑ (p=2.2e-24) → (p=3.8e-23)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLS SpectraAE</td>
<td>(p=6.6e-11) ↑ (p=3.0e-05) ↑ (p=1.7e-11) ↑ (p=1.7e-20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table B.9: Paired t-test results part 8/8.
B.2 Neural Network Hyperparameter Search Space

We considered a large hyperparameter search space and experimented with many possible neural network architectures. Hyperparameters in all neural network models were tuned using a combination of manual trial-and-error and grid search. A list of hyperparameters and candidate values that were considered are given in Table B.10 with the best-found values highlighted. Further details on some of the architecture and hyperparameter choices are described below:

- We experimented with adding a second, intermediate, fully-connected layer in the CNN-2 architecture but found that the tuning set cross-validation score was better without it.

- The auto-encoder reconstructs a spectrum given the encoding. The encoding is constrained by a supervised loss function over the element assays. We experimented with adding additional, unconstrained, units to the encoding. The number of units tried is given under “number of extra encoding units” in Table B.10. These units give the network capacity to learn features about the spectra that are not included in the 48-element composition. For example, if an element shows up that wasn’t assayed or if a non-elemental feature is influencing the spectrum (e.g., a factor of the physical environment). However, we found the effect on tuning set score to be negligible. Future work may investigate this idea further.

- The auto-encoder optimizes two loss functions: the supervised loss function (MSE between prediction and ground truth) and the reconstruction loss function (MSE between spectra and reconstructed spectra). The reconstruction loss is scaled by a hyperparameter to balance between the two loss functions and is given by “weight of reconstruction loss” in Table B.10.
<table>
<thead>
<tr>
<th>Model</th>
<th>Hyperparameter</th>
<th>Candidate Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN-1</td>
<td>fully-connected layer: L1 regularization scale ($\lambda$)</td>
<td>1e-05, 3.16e-05, <strong>0.0001</strong>, 0.000316, 0.001, 0.00316, 0.01, 0.0316, 0.1, 0.316, 1.0</td>
</tr>
<tr>
<td></td>
<td>fully-connected layer: weight initialization function</td>
<td>unit normal, xavier, <strong>zero</strong></td>
</tr>
<tr>
<td></td>
<td>fully-connected layer: activation function</td>
<td>elu, <strong>none</strong>, relu, sigm, tanh</td>
</tr>
<tr>
<td></td>
<td>conv. layers: filter initialization function</td>
<td>zero, xavier, <strong>glorot uniform</strong></td>
</tr>
<tr>
<td></td>
<td>conv. layers: filter width</td>
<td>1, 3, 5, 13, <strong>21</strong></td>
</tr>
<tr>
<td></td>
<td>conv. layers: activation function</td>
<td>elu, <strong>none</strong>, relu, sigm, tanh</td>
</tr>
<tr>
<td></td>
<td>conv. layers: stride</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9, 10</td>
</tr>
<tr>
<td></td>
<td>number of fully-connected layers</td>
<td>0, <strong>1</strong>, 2</td>
</tr>
<tr>
<td></td>
<td>fully-connected layers: L1 regularization scale ($\lambda$)</td>
<td>1e-05, 3.16e-05, <strong>0.0001</strong>, 0.000316, <strong>0.001</strong>, 0.00316, 0.01, 0.0316, 0.1, 0.316, 1.0</td>
</tr>
<tr>
<td></td>
<td>fully-connected layer: weight initialization function</td>
<td>unit normal, xavier, <strong>zero</strong></td>
</tr>
<tr>
<td></td>
<td>fully-connected layer: activation function</td>
<td>elu, <strong>none</strong>, relu, sigm, tanh</td>
</tr>
<tr>
<td></td>
<td>conv. layers: filter initialization function</td>
<td>zero, xavier, <strong>glorot uniform</strong></td>
</tr>
<tr>
<td></td>
<td>conv. layers: activation function</td>
<td>elu, <strong>none</strong>, relu, sigm, tanh</td>
</tr>
<tr>
<td></td>
<td>conv. layers: stride</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9, 10</td>
</tr>
<tr>
<td>AE</td>
<td>number of extra encoding units</td>
<td>0, <strong>1,2,3,4,5,10,15,20,25,26,27,30,40</strong></td>
</tr>
<tr>
<td></td>
<td>weight of reconstruction loss</td>
<td>1, 10, 100, 1000, <strong>10000</strong>, 100000</td>
</tr>
<tr>
<td>AE's decoder</td>
<td>fully-connected layers: L1 regularization scale ($\lambda$)</td>
<td>1e-05, 3.16e-05, <strong>0.0001</strong>, 0.000316, <strong>0.001</strong>, 0.00316, 0.01, 0.0316, 0.1, 0.316, 1.0</td>
</tr>
<tr>
<td></td>
<td>fully-connected layers: weight initialization function</td>
<td>unit normal, xavier, <strong>zero</strong></td>
</tr>
<tr>
<td></td>
<td>fully-connected layers: activation function</td>
<td>elu, <strong>none</strong>, relu, sigm, tanh</td>
</tr>
<tr>
<td></td>
<td>conv. layers: filter initialization function</td>
<td>zero, xavier, <strong>glorot uniform</strong></td>
</tr>
<tr>
<td></td>
<td>conv. layers: filter width</td>
<td>1, 3, <strong>5</strong>, 13, 21</td>
</tr>
<tr>
<td></td>
<td>conv. layers: activation function</td>
<td>elu, <strong>none</strong>, relu, sigm, tanh</td>
</tr>
<tr>
<td></td>
<td>conv. layers: stride</td>
<td>1, 2, 3, <strong>4</strong>, 5</td>
</tr>
<tr>
<td>All networks</td>
<td>dropout: apply to input units</td>
<td>enabled, <strong>disabled</strong></td>
</tr>
<tr>
<td></td>
<td>dropout: initial probability ($dropout_p$)</td>
<td>1.0, <strong>0.5</strong>, 0</td>
</tr>
<tr>
<td></td>
<td>dropout: annealing duration ($dropout_n$)</td>
<td>0 (disable dropout), 2000, <strong>5000</strong>, 10000</td>
</tr>
<tr>
<td></td>
<td>learning rate (initial)</td>
<td>0.1, 0.01, <strong>0.001</strong>, 0.0001</td>
</tr>
<tr>
<td></td>
<td># epochs until annealing learning rate</td>
<td>1000, <strong>2000</strong>, 4000, inf (disable annealing)</td>
</tr>
<tr>
<td></td>
<td># epochs until early-stopping</td>
<td>1000, <strong>2000</strong>, 5000, 10000</td>
</tr>
<tr>
<td></td>
<td># cold restarts</td>
<td>0, <strong>5</strong></td>
</tr>
</tbody>
</table>

Table B.10: List of hyperparameters and candidate values that were considered, grouped by neural network model. The **best-found value** for each parameter is highlighted.
Appendix C

Supplementary Material:
Chapter 6

C.1 Ensemble Size

To choose the number of members in the ensemble, an analysis was conducted to examine the relationship between RMSE and ensemble size prior to the experiments described in this chapter. A CNN like NN1 was used for this analysis, before hyperparameter optimization was conducted.

Following the approach used previously [68], the RMSE distributions in Figure C.1 are computed by training the neural network 5000 times and simulating ensembles of different sizes. Each ensemble is a random, but unique, combination of $N$ models. We generate 200 ensembles of each size, $N$, and plot the distribution of RMSE scores. From these results, it is apparent that the variance is significantly reduced after around $N = 30$, with diminishing returns as $N$ increases further. It is important to point out that the number of members should be adjusted according to the requirements of the prediction task and the computational resources available to the practitioner.
Figure C.1: Boxplot showing how much RMSE decreases as the size of the ensemble increases. Size $N = 1$ (purple) indicates the distribution of RMSE scores over 5000 single re-trainings (with random initialization) of the neural network model. Each box (where $N > 1$) is a distribution over 200 simulated ensembles, where each ensemble is formed from a unique combination of $N$ trained models from the pre-computed set of 5000. We use ensembles with 40 member models in our experiments (green).