T.R.
GEBZE TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

MACHINE LEARNING TECHNIQUES FOR THE ESTIMATION OF MATERIAL PROPERTIES FROM LIGHT SPECTRUM DATA

EMRE ARDIÇ
A THESIS SUBMITTED FOR THE DEGREE OF
MASTER OF SCIENCE
DEPARTMENT OF COMPUTER ENGINEERING

GEBZE
2018

T.R.

GEBZE TECHNICAL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

MACHINE LEARNING TECHNIQUES FOR THE ESTIMATION OF MATERIAL PROPERTIES FROM LIGHT SPECTRUM DATA

EMRE ARDIÇ

A THESIS SUBMITTED FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER ENGINEERING

THESIS SUPERVISOR

ASSIST. PROF. DR. YAKUP GENÇ

GEBZE

2018

T.C.
GEBZE TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

IŞIK SPEKTRUM VERİSİNDEN MADDE ÖZELLİKLERİNİN TAHMİNİ İÇİN MAKİNE ÖĞRENMESİ TEKNİKLERİ

EMRE ARDIÇ
YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMANI
DR. ÖĞR. ÜYESİ YAKUP GENÇ

GEBZE
2018

GEBZE TEKNİK ÜNİVERSİTESİ	**YÜKSEK LİSANS JÜRİ ONAY FORMU**

GTÜ Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 20/06/2018 tarih ve 2018/32 sayılı kararıyla oluşturulan jüri tarafından 18/07/2018 tarihinde tez savunma sınavı yapılan Emre ARDIÇ'ın tez çalışması Bilgisayar Mühendisliği Anabilim Dalında YÜKSEK LİSANS tezi olarak kabul edilmiştir.

JÜRİ

ÜYE (TEZ DANIŞMANI)	: Dr. Öğr. Üyesi Yakup GENÇ
ÜYE	: Prof. Dr. Yusuf Sinan AKGÜL
ÜYE	: Dr. Öğr. Üyesi Ulaş VURAL

ONAY

Gebze Teknik Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun

...../...../..... tarih ve ...../..... sayılı kararı.

SUMMARY

The absorbance spectrum technique is as old as the first alchemists. They desired to learn and identify their potions by examining the color and opacity of solutions as different reagents were mixed, heated, and stirred. Today, it is still the most frequently used spectroscopic technique on studies of liquids and gases as it is simple, accurate, and easy to use. An absorbance spectrum can be used to identify substances or measure the concentration of a molecule in solution. In this work, partial least squares regression (PLSR), gradient boosting regression (GBR), random forests (RF), convolutional neural network (CNN) and long short-term memory (LSTM) models were trained on absorbance spectrums of different liquid solvents and concentration of a specific molecule was predicted. A large data set was collected to train and test the models. In order to increase accuracy of the models, data set was scaled and cleaned from abnormal data. Single-tasking and multi-tasking CNNs were designed and optimized. The models were compared by performance metrics of mean absolute error (MAE) and coefficient of determination. As shown in experiments, the proposed CNN models gave the best results with about 1.1% mean absolute error.

Key Words: Absorbance Spectrum Analysis, Concentration Prediction, Multi-Task Learning (MTL), Convolutional Neural Network (CNN).

ÖZET

Emilim spektrumu tekniği ilk simyacılar kadar eskidir. İksirlerin rengine, saydamlığına bakarak ve farklı karışımlar deneyerek onları öğrenmeye ve tanımaya çalışmışlardır. Bugünlerde bile basit, hassas ve kullanımı kolay olduğu için sıvılar ve gazlar üzerindeki çalışmalarda sıkça kullanılmaktadır. Emilim spektrumu tekniği sıvı kimyasal çözeltilerdeki bir maddenin tanınmasında veya bir molekülün yoğunluğunun ölçülmesinde kullanılabilir. Bu çalışmada farklı sıvı çözeltilerin emilim spektrumları üzerinde kısmi en küçük kareler, gradyan destekli regresyon, rastgele ormanlar, evrişimsel sinir ağları ve uzun kısa-süreli bellek modelleri eğitilerek belirli bir molekülün yoğunluğu tahmin edilmiştir. Modellerin eğitimi ve testi için büyük bir veri kümesi oluşturulmuştur. Modellerin doğruluğunu artırabilmek için veri kümesi ölçeklendirilip ve anormal verilerden temizlenmiştir. Tek ve çok görevli evrişimsel sinir ağları geliştirilmiş ve optimize edilmiştir. Modeller ortalama mutlak hata (MAE) ve belirlilik katsayısı performans metriklerine göre karşılaştırılmıştır. Deneylerde görüldüğü üzere önerilen evrişimsel sinir ağları yaklaşık %1,1 ortalama mutlak hata ile en iyi sonuçları vermiştir.

Anahtar Kelimeler: Emilim Spektrumu Analizi, Konsantrasyon Tahmini, Çok-Görevli Öğrenme (MTL), Evrişimsel Sinir Ağları (CNN).

ACKNOWLEDGEMENTS

I would like to express my deep and sincere gratitude to my supervisor, Assist. Prof. Yakup GENÇ, who not only shared his profound scientific knowledge with me but also taught me great lessons of life. His support, suggestions and encouragement gave me the drive and will to complete this work.

The project was completed by support of Bioelectronics Devices and System Development Laboratory in TÜBİTAK UEKAE. I would like to thank to Cihan KILIÇ for the creation of spectrum dataset, who currently is Researcher at TÜBİTAK.

I am grateful to my parents for their love and support.

TABLE of CONTENTS

	Page
SUMMARY	v
ÖZET	vi
ACKNOWLEDMENTS	vii
TABLE of CONTENTS	viii
LIST of ABBREVIATIONS and ACRONYMS	x
LIST of FIGURES	xi
LIST of TABLES	xv
1. INTRODUCTION	1
1.1. Electromagnetic Radiation	1
1.2. Absorption Spectroscopy	2
1.3. Beer-Lambert Law	4
2. RELATED WORKS	7
2.1. Predicting Compound Concentration in Tablets	7
2.2. Classification of Various Spectrums	9
2.3. Predicting Organic Acids Concentration in Digesters	10
2.4. Identification of Absorption Bumps	11
3. METHODS	13
3.1. Dataset Preparation	13
3.1.1. Preprocessing	15
3.1.2. Outlier Removal	15
3.2. Partial Least Squares Regression	18
3.2.1. Prerequisite Notions and Notations	18
3.2.2. Decompositions of Variables	18
3.2.3. Regression Procedure	19
3.3. Random Forest Regression	19
3.3.1. Classification Trees	20
3.3.2. Regression Trees	23
3.4. Gradient Boosting Regression	24
3.4.1. Function Estimation	24

3.4.2. Steepest Descent 25
3.4.3. Approximations for Finite Datasets 26
3.4.4. Regression Algorithms 27
3.4.5. Regularization 29
3.5. Convolutional Neural Networks 30
3.5.1. Core Layers 31
3.5.2. Regularization 34
3.5.3. Optimization Principles 35
3.5.4. Proposed Architectures 39
3.6. Long Short-Term Memory 43
4. EXPERIMENTS AND RESULTS 47
4.1. Partial Least Squares Regression 47
4.2. Random Forest Regression 50
4.3. Gradient Boosting Regression 55
4.4. Convolutional Neural Networks 62
4.4.1. Single-Tasking Architecture 62
4.4.2. Multi-Tasking Architecture 69
4.4.3. Multivariate Regression 73
4.5. Long Short-Term Memory 74
4.6. Analysis and Results 77
5. CONCLUSIONS 79

REFERENCES 81
BIOGRAPHY 85
APPENDICES 86

LIST of ABBREVIATIONS and ACRONYMS

Abbreviations and Acronyms		**Explanations**
1D	:	1 Dimensional
2D	:	2 Dimensional
30D	:	30 Dimensional
4D	:	4 Dimensional
ANN	:	Artificial Neural Network
CNN	:	Convolutional Neural Network
GBR	:	Gradient Boosting Regression
GerDA	:	Generalized Discriminant Analysis
GTU	:	Gebze Technical University
KNN	:	K-Nearest Neighbor
LDA	:	Linear Discriminant Analysis
LSTM	:	Long Short-Term Memory
MLP	:	Multi-Layer Perceptron
MLR	:	Multiple Linear Regression
NIR	:	Near Infrared
PCA	:	Principal Component Analysis
PLS	:	Partial Least Squares
PLSR	:	Partial Least Squares Regression
ReLU	:	Rectifier Linear Unit
RF	:	Random Forests
RVM	:	Relevance Vector Machine
SVM	:	Support Vector Machine
UV	:	Ultraviolet

LIST of FIGURES

Figure No:		Page
1.1:	Plane-polarized electromagnetic radiation on the x axis; electric component is E_y; wavelength is λ; magnetic component is H_z; A is amplitude of wave.	2
1.2:	Structure of an atom with two energy levels.	3
1.3:	An example of absorption spectrum where absorption lines are shown as vertical black lines.	4
1.4:	Absorbance spectrum measurement system; I_o is transmitted light to the sample and I_t is received light that comes from sample.	4
1.5:	Light is absorbed by sample in a cuvette where I_o is transmitted light to the sample, I_t is received light that comes from sample and l is the path length of light.	5
1.6:	Example absorption band between wavenumbers of λ_1 and λ_2; c is concentration of the absorbing material in the solution.	5
2.1:	Bjerrum's deep convolutional neural network for predicting weight of active compound in pharmaceutical tablets.	8
2.2:	Example of 1D CNN architecture for two-class classification problem with one convolution layer and two output neurons.	10
3.1:	Absorbance spectrum measurement system including spectrometer, UV-Visible light source, 1 cm path-length cuvette, cuvette holder, fibers and computer.	14
3.2:	Distribution of the entire dataset by concentration values.	15
3.3:	Scaled absorption spectrums of three samples with concentration of 0%, 103% and 140%.	16
3.4:	Average R^2 scores of PLSR model for 10-fold cross-validation sets. The best score (0.93) was achieved for 13 components.	16
3.5:	Spectral outliers are detected by PLSR model trained on the entire dataset. Blue line represents $y = x$, red and green dots indicates abnormal and normal spectrums, respectively.	17

3.6: A decision tree example for a dataset. Circular shapes are the decision nodes and rectangles are leaf nodes. The univariate decision node splits throughout one axis and consecutive splits are orthogonal to each other. After the first split, $\{x|x_1 < w_{10}\}$ is pure and is not divided further. 20
3.7: Tree construction algorithm for a classification problem. 22
3.8: Gradient boosting algorithm for least squares loss function. 28
3.9: GBR algorithm for LAD regression. 29
3.10: A three-layered feedforward neural network with an input layer, a hidden layer and an output layer. 30
3.11: An example CNN architecture with five layers for classification of MNIST images. 31
3.12: A visual representation of a convolutional layer. 32
3.13: Max and average pooling examples. The spatial extent (F) and the stride (S) parameters are 2. 33
3.14: An example of dropout operation applied on the network at left. The ignored neurons are shown as red. 34
3.15: An example of simple perceptron. $x_1, x_2, .., x_N$ are inputs. $w_1, w_2, .., w_N$ are weight of connections. f is nonlinear function and y is the output. 36
3.16: An example of neural network with a hidden layer. Inputs are $x_1, x_2, .., x_N$, outputs are $y_1, y_2, .., y_M$ and hidden layers are defined as $h_1, h_2, .., h_K$. Weight matrices are defined as W_{NxK} and W'_{KXM} respectively. 37
3.17: Single-tasking 1D CNN with one convolutional layer. 40
3.18: Multi-tasking 1D CNN with one convolutional layer. 40
3.19: An example of 1D convolution applied on a 1D vector. 41
3.20: Hard parameter sharing for multi-task learning in deep neural networks. 42
3.21: Soft parameter sharing for multi-task learning in deep neural networks. 42

3.22: Example of RNN for time series with length n. Each node represents a time t and the information at time t flows to the node at time $t+1$ 44
3.23: A common architecture of LSTM units with input i, output o and forget f gates. The input vector is x_t, output vector is h_t and memory cell is c_t at time step t. 45
4.1: Average R^2 score of the models trained by 10-fold cross-validation. The best R^2 scores were achieved for 15, 32 and 41 components. 48
4.2: Average MAE of the models trained by 10-fold cross-validation. The lowest MAE was achieved for 41 components. 48
4.3: True and predicted concentrations for the model trained for 32 components. 49
4.4: True and predicted concentrations for the model trained for 41 components. 49
4.5: Average R^2 scores of the models trained for different number of decisions trees. The best R^2 score (0.97) was achieved for 51 decision trees. 51
4.6: Average MAEs of the models trained for different number of decisions trees. The lowest MAE (2.59) was achieved for 51 decision trees. 51
4.7: True and predicted concentrations for the model trained for minimum 10 samples splits. 52
4.8: True and predicted concentrations for the model trained for maximum 50 splits. 53
4.9: True and predicted concentrations for the model trained for minimum 10 samples leaf. 54
4.10: True and predicted concentrations for the model trained for minimum 20 samples leaf. 54
4.11: True and predicted concentrations for the best model. 55
4.12: True and predicted concentrations of the model trained by using the parameters in Table 4.6, with 10-fold cross-validation. 57
4.13: Average R^2 scores of the models trained by using the parameters in Table 4.6, with 10-fold cross-validation. 58

4.14: Average R^2 scores of the models trained for learning rate 0.05 and various maximum depths, with 10-fold cross-validation. 59
4.15: Average MAE of the models trained for various maximum depths, learning rate 0.6 and 100 estimators, with 10-fold cross-validation. 60
4.16: Predicted concentrations of the best model trained on the entire dataset with the parameters in Table 4.25. Green, red and blue colors refer to chemical 1, 2 and 3, respectively. 68
4.17: Average MAE of the models trained on the entire dataset with the parameters in Table 4.25. 68
4.18: Predicted concentrations of the best model trained on the entire dataset with the parameters in Table 4.25. Green, red and blue colors refer to chemical 1, 2 and 3, respectively. 72
4.19: Average MAE of the models trained on the entire dataset with the parameters in Table 4.25. 72
4.20: Single-layered LSTM network architecture. 74
4.21: The deep network with 1D CNN and LSTM layers. 74

LIST of TABLES

Table No:		Page
2.1:	The optimized hyperparameters of the proposed neural network.	9
2.2:	Performance of CNN and PLS models on test set of standard global scaled dataset.	9
4.1:	The average MAE and R^2 scores acquired for 15, 32 and 41 components.	48
4.2:	Training parameters of the best RF model.	50
4.3:	The MAE and R^2 score of the models trained for different number of estimators.	50
4.4:	The MAE and R^2 score of the models trained for different number of minimum samples split.	52
4.5:	The MAE and R^2 score of the models trained for different number of minimum samples leaf.	53
4.6:	The parameters of the best GBR model.	56
4.7:	Average R^2 score and MAE of the model trained by using various number of estimators, learning rate 0.1 and maximum depth 3, with 10-fold cross-validation.	56
4.8:	Average MAE and R^2 score of the model trained by using different maximum depths, learning rate 0.1 and estimator count 100, with 10-fold cross-validation.	57
4.9:	Average MAE and R^2 score of the GBR model trained by using various maximum depths, learning rate 0.05 and estimator count is 100.	58
4.10:	Average MAE and R^2 score of the GBR model trained by using different maximum depths, learning rate 0.2 and estimator count is 100, with 10-fold cross-validation.	59
4.11:	Average MAE and R^2 score of the GBR model trained for different maximum depth values with learning rate 0.6 and estimator count is 100.	60

4.12: Average MAE and R^2 score of the GBR model trained by using different number of estimators, learning rate 0.1 and maximum depth 9. 61
4.13: Average MAE and R^2 score of the GBR model trained by using different loss functions, 100 estimators, learning rate 0.1 and maximum depth 9. 61
4.14: Hyperparameter search space for Bayesian optimizer. 63
4.15: Initial parameters of the shallow CNN for Bayesian optimization. 63
4.16: The hyperparameters found by Bayesian optimizer for the shallow CNN. 64
4.17: Average MAE and R^2 score of the CNN model for 50, 100 and 200 filters. 64
4.18: Average MAE and R^2 score of the CNN model for different filter lengths. 65
4.19: Average MAE and R^2 score of the CNN model for various strides and filter length 25. 65
4.20: Average MAE and R^2 score of the CNN model for various noise level in the input. 65
4.21: Average MAE and R^2 score of the CNN model for different learning rates. 66
4.22: Average MAE and R^2 score of the CNN model for various dropout probabilities. 66
4.23: Average MAE and R^2 score of the CNN model for two convolutional layers. 67
4.24: Average MAE and R^2 score of the CNN model for additional one fully connected layer. 67
4.25: The hyperparameters found by fine-tuning the parameters found by Bayesian optimizer. 67
4.26: Average MAE and R^2 scores for 50, 100 and 200 filters. 69
4.27: Average MAE and R^2 scores for various filter lengths and 50 filters. 70
4.28: Average MAE and R^2 scores for various filter lengths and 100 filters. 70

4.29: Average MAE and R^2 scores for 10, 100 and 1000 neurons in task layers. 70
4.30: Average MAE and R^2 scores for extra one convolutional layer in the shared layer. 71
4.31: Average MAE and R^2 scores for extra one convolutional layer in the task layers. 71
4.32: Average R^2 score and MAE of the multivariate regression models. 73
4.33: Initial parameters of the LSTM network. 75
4.34: Initial parameters of the LSTM layer. 75
4.35: Average MAE and R^2 score of the single-layered LSTM model for dimensions of 1, 5, 10 and 20. 76
4.36: The hyperparameters of the CNN layer. 76
4.37: Average MAE and R^2 score of the model with 1D CNN and LSTM layers for 20 and 100 units. 76
4.38: Average R^2 score and MAE of all the proposed models. 77

1. INTRODUCTION

Light is an electromagnetic radiation of any wavelength, whether visible or not. All materials absorb light energy and intensity of the absorbed energy can be expressed as a function of frequency and wavelength. Absorption spectroscopy is a technique to measure amount of energy that is absorbed by a substance. It is mainly used to determine the presence of a specific substance in a sample and quantify the amount of the substance present. It has many applications such as particle size analysis, polymer processing, trace detection of metals, ozone monitoring, analysis of composition in dairy products and clinical blood diagnostics.

In this work, various machine and deep learning models were trained on absorbance spectrums of different solvents and concentration of a specific substance in these solvents was predicted. Partial least squares, random forests, gradient boosting and convolutional neural networks were used to generate the models. Ultraviolet-visible spectroscopy technique was employed to measure absorption levels of specific wavelengths. Large data set was collected by preparing and measuring different type of solvents. Data sets were cleaned and scaled before training. Mean absolute error and coefficient of determination (R^2) were used to evaluate and compare the regression models.

1.1. Electromagnetic Radiation

Light is an electromagnetic radiation of longer and shorter wavelengths. It consists of both a magnetic and an electric component, which can be considered as plane-polarized radiation [1]. Figure 1.1 shows one photon of this kind of radiation moving on the x axis. Equation (1.1) illustrates electric and magnetic components of photons where v is frequency of photon and A is the amplitude.

$$\left.\begin{aligned} E_y &= Asin(2\pi vt - kx) \\ H_z &= Asin(2\pi vt - kx) \end{aligned}\right\} \tag{1.1}$$

The electric component of the radiation is considered as an oscillating electric field of strength E and the magnetic component is considered as an oscillating magnetic field of strength H. If E and H vectors are y and z, then the angle between

these oscillating fields is 90 degrees. Thus, the fields oscillate sinusoidally with $2\pi n$ frequency. Since k is the same for E and H components, they are in-phase.

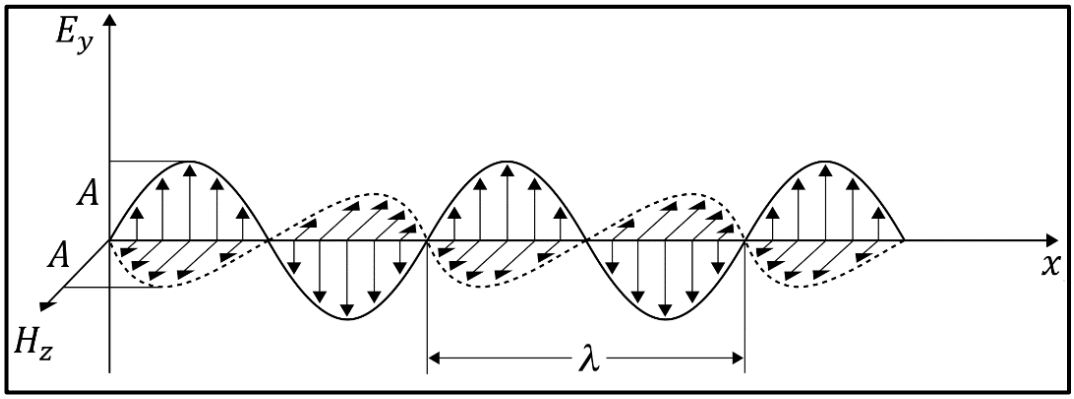

Figure 1.1: Plane-polarized electromagnetic radiation on the x axis; electric component is E_y; wavelength is λ; magnetic component is H_z; A is amplitude of wave.

The polarization plane is taken to be the plane in the direction of E and that of propagation; this is the xy plane in Figure 1.1. The reason for this selection is that interaction of matter with electromagnetic radiation is mostly through the electric component.

1.2. Absorption Spectroscopy

Atoms contain very small particles such as electrons, protons and neutrons. Electrons and protons are the negatively and positively charged particles, respectively. Neutrons are neutral particles since they have no charge.

The electrostatic attraction force between the electrons and nucleus induces electrons to turn around the nucleus. The electrons have different energy levels based on the distance from the nucleus. The electrons revolving very close to the nucleus have the lowest energy level while the electrons revolving at the longest distance from nucleus have the highest energy level. As shown in Figure 1.2, the lowest energy level is $E1$ and the next higher energy level is $E2$.

The electrons in $E1$ state needs additional energy such as light, heat or electric field to move into next $E2$ state. When the electrons in the lower energy state ($E1$) gain sufficient energy from photons, they move into next higher energy state ($E2$). After a short period, they lose their energy and fall back to the lower energy level. The

electrons in the higher energy level lose their energy by emitting photons before they fall back to the lower energy state.

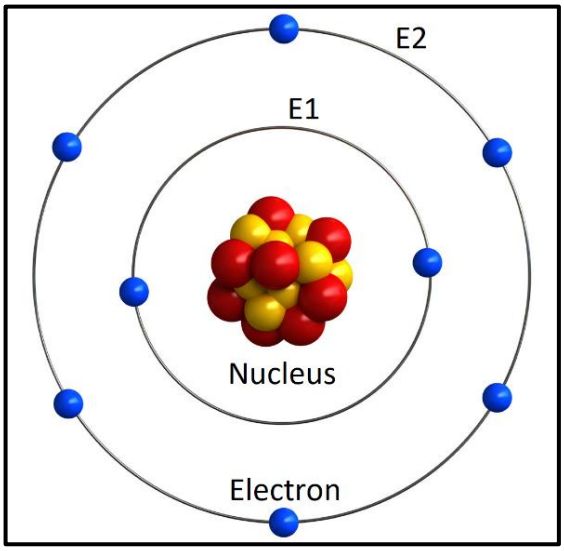

Figure 1.2: Structure of an atom with two energy levels.

The process of absorbing energy from photons is called absorption of radiation. When the electrons in $E1$ state absorb energy that is equal to the energy difference between $E1$ and $E2$, the electrons jump from $E1$ to $E2$ state. Absorption happens only if the energy of photon exactly matches the energy difference between the two electron shells or orbits.

$$hv = E1 - E2 \tag{1.2}$$

Equation (1.2) illustrates the relation between energy levels and frequency of photons where frequency of photon is v, Planck's constant is h, lower energy level is $E1$ and higher energy level is $E2$.

Absorption spectroscopy is a method to measure the energy that is absorbed by a substance. Absorption spectrum of a substance is calculated by measuring absorptions over a range of frequencies [1]. The spectrum is mostly affected by the atomic and molecular structure of a material. Light energy is absorbed at frequencies if the energy difference between two quantum mechanical states of the molecules matches. The absorption taking place because of this transition is known as absorption line as shown in Figure 1.3. An absorption spectrum contains many of it. The frequencies where absorption line is seen mostly depend on molecular structure of the

sample. They are also affected by temperature, pressure, electromagnetic fields and interactions between molecules.

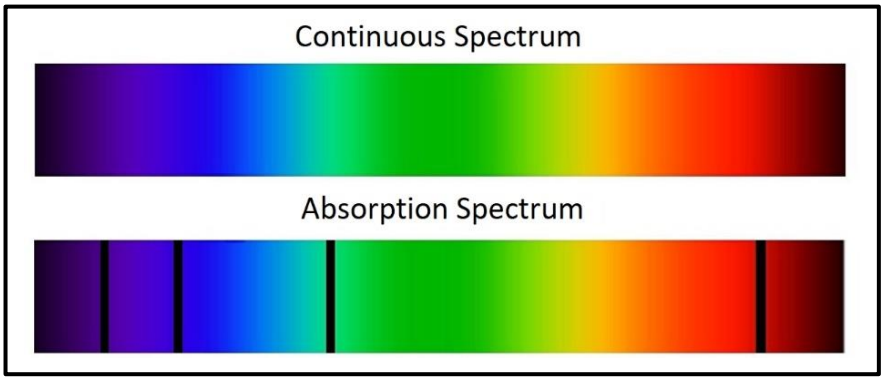

Figure 1.3: An example of absorption spectrum where absorption lines are shown as vertical black lines.

The most common case for measuring absorption spectra is to send a generated beam of radiation (I_o) to a sample and measure the intensity of the radiation (I_t) that passes through it as shown in Figure 1.4. The transmitted energy (I_o) can be used to calculate the absorption. The light source, sample arrangement and detection methods change according to the wavelengths and the type of the experiment.

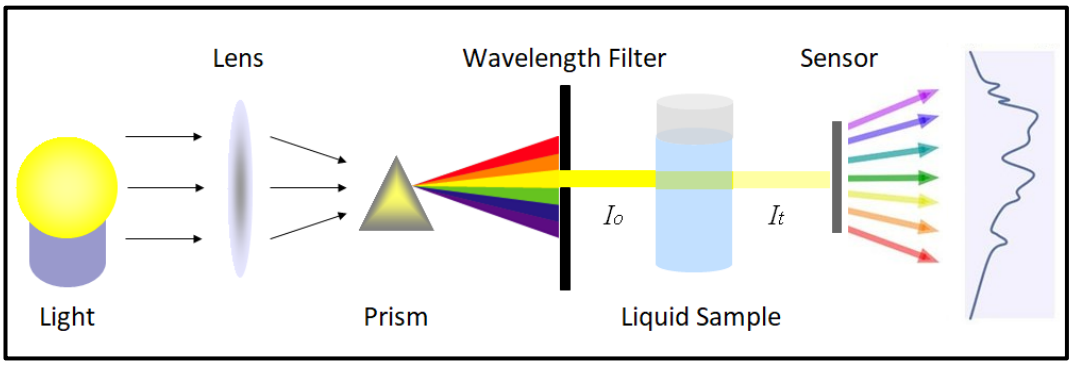

Figure 1.4: Absorbance spectrum measurement system; I_o is transmitted light to the sample and I_t is received light that comes from sample.

1.3. Beer-Lambert Law

Beer-Lambert law explains the linear relationship between absorbance and concentration of an absorbent material. Figure 1.5 demonstrates a measurement system to calculate concentration of an analyte in solution by using absorption.

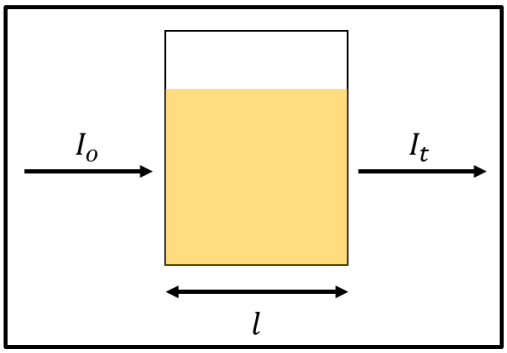

Figure 1.5: Light is absorbed by sample in a cuvette where I_o is transmitted light to the sample, I_t is received light that comes from sample and l is the path length of light.

The law is generally defined as Equation (1.3) where A is measured absorbance, $\varepsilon(\lambda)$ is wavelength-dependent absorptivity coefficient, c is concentration and l is path length of light in cuvette. Since A is dimensionless, $\varepsilon(\lambda)$ has dimensions of $mol^{-1}dm^3cm^{-1}$. Path length of light (l) and concentration (c) is directly proportional to absorbance (A) where A ranges from 0 to 1. As shown in Figure 1.6, $\varepsilon(\lambda)$ can be determined by using maximum absorbance value in a given wavelength range or integrating the area under the curve.

$$A = log_{10}(\frac{I_o}{I_t}) = \varepsilon(\lambda)cl \tag{1.3}$$

The linearity of the law is limited due to some chemical and instrumental factors such as light scattering effect caused by particulates in the sample, stray light, deviations in absorptivity coefficients at high and low concentrations.

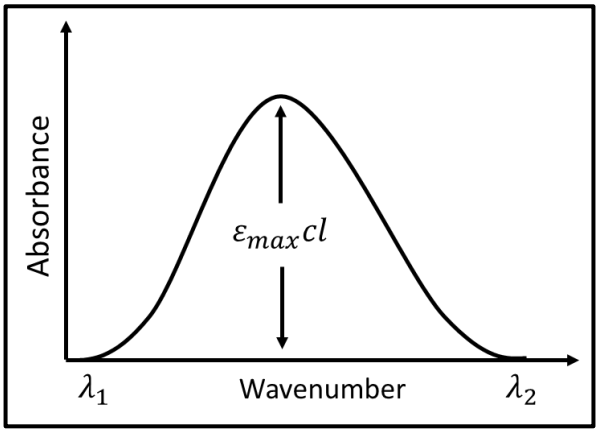

Figure 1.6: Example absorption band between wavenumbers of λ_1 and λ_2; c is concentration of the absorbing material in the solution.

Chemical interactions between the analyte and the solvent produce a substance with different absorption behaviors and this phenomenon causes nonlinearities between the absorption and concentration. For instance, pH of the solvent affects the electron arrangement of phenol molecule in it. Since UV-Visible spectroscopy is based on electrons, the change in pH of the solvent alters the absorption spectrum of the samples.

When concentration of solute increases in the solution, solute molecules induce different charge distribution on their adjacent species. Thus, absorption wavelength of the analyte is shifted. High analyte concentrations may alter refractive index of the solution and this affects measured absorbance. At very low concentrations, some molecules or ions such as methylene blue can cause deviations in absorptivity coefficients.

Stray light or false light is unintended light in an optical system and it is caused by light scattering, malfunction of spectrometer or diffraction. Stray light causes a decrease in measured absorbance and breaks the linear relationship between concentration and absorbance. It can be filtered by cut-off filters which absorbs lights in specific wavelength range and transmits higher wavelengths.

As shown in Equation (1.3), cuvettes having different path lengths causes deviations in calculated absorbance. Thus, cuvettes should have equal optical properties. All surfaces of the cuvette must be clean since fingerprints, dust and dirt may cause light scattering and add absorbance signatures of their own.

2. RELATED WORKS

Purpose of this work is to predict concentration of specific substance in different chemical solutions via absorption spectroscopy. There are various related regression and classification works based on absorption spectrums. Frequently used methods in these works are partial least squares (PLS), linear discriminant analysis (LDA), support vector machines (SVM), multi-layer perceptron (MLP), RF and k-nearest neighbor (KNN). Recent works employ CNN and other deep neural networks to deal with nonlinearity and noise explained in Section 1.3. According to related works, CNN architectures provide promising result for classification and regression tasks as they are easy to use, deal with noise and nonlinearity better, and so on.

2.1. Predicting Compound Concentration in Tablets

Bjerrum et. al. designed and optimized a deep convolutional neural network on near infrared (NIR) spectra data to predict compound concentration in tablets [2]. CNN models outperformed the PLS models for all combinations of preprocessing.

Bjerrum used the dataset containing NIR spectra from 654 pharmaceutical tablets from two spectrometers [2]. Dataset was normalized by subtracting the global mean of the training set and dividing by two times the global standard deviation of the training to make sure that values were in the range -1 to 1. Test and validation sets were created randomly by using 20% of the dataset. The training and validation sets were created by spectrums obtained with instrument one, whereas the test sets were only taken from instrument two. The wavelength region from 600 to 1798 nm was used for modelling.

Bjerrum identified spectral outliers by PLS modeling using the implementation of the NIPALS algorithm [3], without scaling, a maximum of 100.000 iterations and a tolerance of 10^{-16}. The all dataset was applied to 10-fold cross-validation for the number of principal components between 1 and 30. The number that gave the lowest average Huber loss [4] for the cross-validation sets was used to train PLS model on the entire dataset and the samples having more than 2.5 times the standard deviation of the absolute error of prediction identified as outliers.

Bjerrum designed the deep neural network as shown in Figure 2.1. The input was given to a Gaussian noise layer with standard deviation of 0.01 and then fed to 1D CNN layers with ReLU [5] activation function. The output of final CNN layer was flattened and given to a dropout layer. The output of dropout layer was fed to a fully connected dense layer with linear activation function. Final output was a single neuron with linear activation function. Huber loss [4] was used as loss function and Adadelta [6] optimizer was used to train the network.

Bjerrum used Bayesian optimization framework implemented in GpyOpt [7] package to find optimal hyperparameters for the network. Search space of kernel and filter size of CNN layers were set to $[2, 40]$ and $[5, 150]$, respectively. Search space of dropout probability and number of neurons in fully connected layer were set to $[0, 0.5]$ and $[4, 1000]$, respectively. Hyperparameter optimization was done with learning rates of 0.084 and 0.094 for 40 or 200 epochs. Batch size was set to 45 and the loss function was used as average of validation loss for the last 10 epochs of training. Maximum 40 iterations of Gaussian process optimization with expected improvement acquisition function was employed for the hyperparameter optimization. The optimized hyperparameters are shown in Table 2.1.

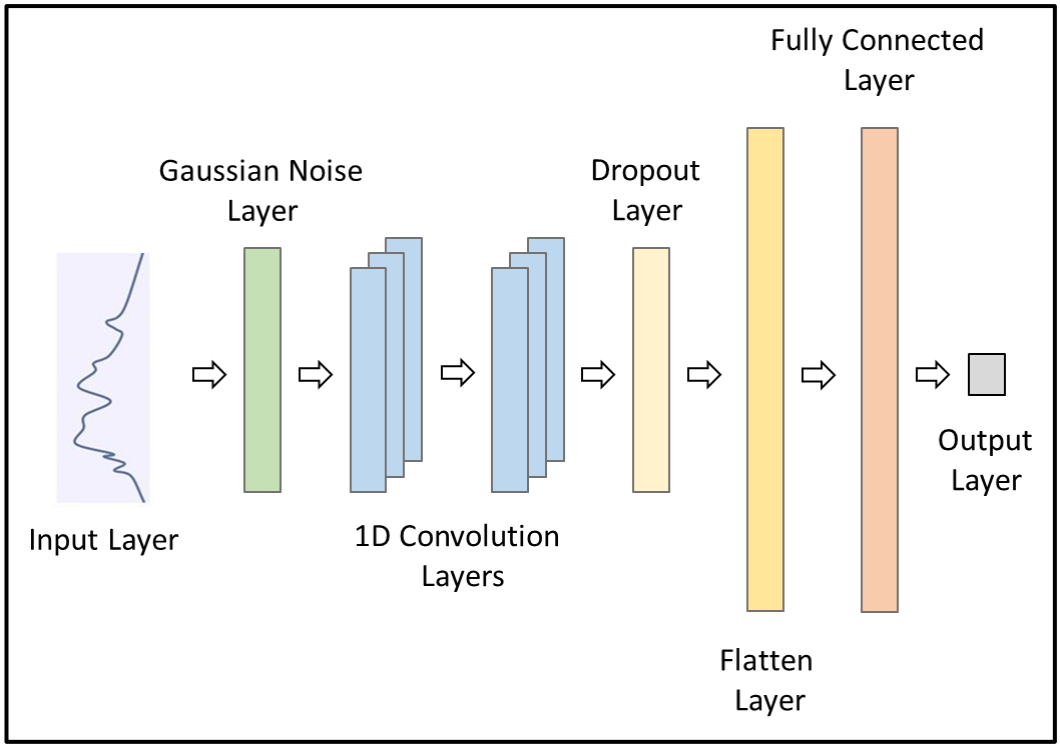

Figure 2.1: Bjerrum's deep convolutional neural network for predicting weight of active compound in pharmaceutical tablets.

Bjerrum optimized the number of principal components for PLS model by minimizing Huber loss on test set. The lowest loss was found at 5 principal components as shown in Table 2.1.

Table 2.1: The optimized hyperparameters of the proposed neural network.

Hyperparameter	Optimized Value
Conv Layer 1 Number of Kernel	14
Conv Layer 1 Filter Size	29
Conv Layer 2 Number of Kernel	30
Conv Layer 2 Filter Size	22
Dropout Probability	0.045
Dense Layer Number of Neuron	176
Principal Components	5

Table 2.2: Performance of CNN and PLS models on test set of standard global scaled dataset.

Model	R^2	RMSE	Huber
CNN	0.97	4.01	2.51
PLS	0.94	4.43	2.60

Bjerrum used coefficient of determination, root mean squared error and Huber loss metrics to measure performance of the trained CNN and PLS models. As shown in Table 2.2, CNN model outperformed PLS model without data preprocessing.

2.2. Classification of Various Spectrums

Acquarelli et. al. designed a shallow 1D CNN architecture as shown in Figure 2.2 to classify spectrums of beer, tablet, wine, coffee, olive oil, juice and meat datasets having different number of classes [8] – [15]. The proposed CNN model outperformed standard classification algorithms used in chemometrics such as PLS-LDA (Linear Discriminant Analysis), KNN and logistic regression in terms of accuracy. It achieved 86% and 96% average accuracy on raw and preprocessed test data, respectively.

Acquarelli used Glorot [16] initialization function for initial weights and Stochastic Gradient Descent (SGD) [17] algorithm for training the network. Softmax activation function was employed for output neurons to calculate probability of classes. Cross-entropy error loss with custom regularization term was used to enforce similarity between nearby features. Random Grid Search Cross-Validation framework

(RGS-CV) [18] was used to optimize parameters of the models. Various data preprocessing methods such as Savitsky-Golay [19], Polynomial detrending [20] and Robust Normal Variate transform [21] were employed to filter common data artifacts such as light scatter effects, baseline and instrumental noise [22].

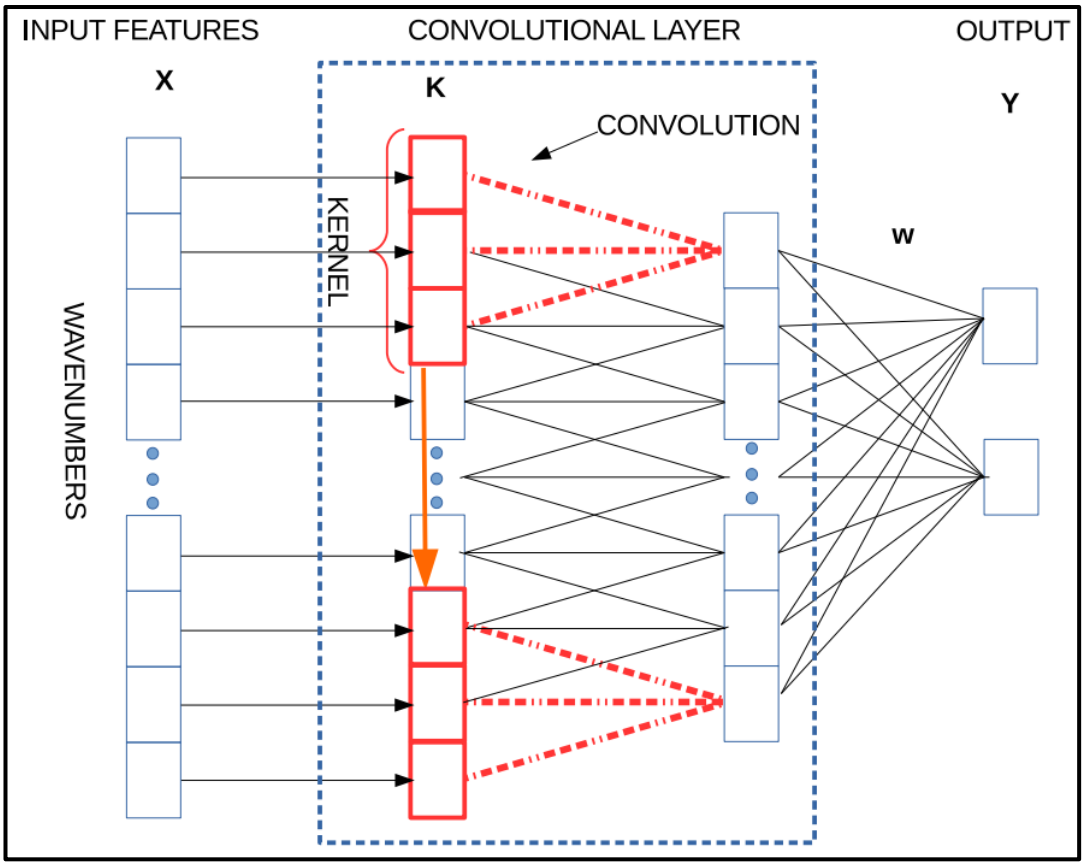

Figure 2.2: Example of 1D CNN architecture for two-class classification problem with one convolution layer and two output neurons.

2.3. Predicting Organic Acid Concentration in Digesters

Wolf et. al. developed an online measurement system using UV-Visible spectroscopic probes to predict the concentration of organic acids in anaerobic digesters by using LDA, GerDA, SVM, RVM, RF, and MLP models [23]. The models are trained on absorption spectrums from 200 to 750 nm. As a result, SVM and GerDA based classifiers outperformed other models and achieved 87% accuracy on test data.

Wolf obtained total of 4437 spectral measurements from a full-scale 1.3-MW industrial biogas plant and used them to create training and validation datasets with 3326 and 1109 samples respectively. The unnecessary spectral region from 640 to 750

nm removed from the spectrums and feature vectors with 176 dimensions were obtained. For classification problem, spectrums clustered to 5 classes such as low, low-normal, normal, normal-high, and high according to concentration. The uneven distribution of the samples between classes 4 and 5 was solved by replicating the samples in class 5 and adding them to dataset.

Wolf used LDA and GerDA methods to get 4D feature vectors followed by linear classification. RF was used for feature selection and classification on 30D feature space. SVM, MLP and RF were used for classification of 4D GerDA features. Performance of the models were compared by normalized mean misclassification rate (NMCR) calculated from confusion matrix. GerDA and RF feature extractors improved accuracy of the classifiers significantly. RF classifier based on GerDA features provided very good classification results for NMCR (12.1%) and SVM yielded the best overall performance for NMCR (12%).

2.4. Identification of Absorption Bumps

The 217.5 nm broad absorption bump is very important among the research topics about cosmic dust grains and it is believed to be highly related with some types of aromatic carbonaceous materials that are known as organic molecules in our Milk Way and other neighboring galaxies [24]-[27]. Therefore, detection of the absorption bump is crucial for understanding origin and development of the organic life. Yuan et. al. [28] designed and analyzed CNN architecture to detect the absorption bumps on a subset of spectra in Mg II catalog from SDSS Data Release 7 [29]. With data augmentation, the proposed CNN model achieved about 99% prediction accuracy for the real-world data.

Yuan applied two data augmentation methods based on curve fitting procedures to the datasets and generated total of 30K samples. Half of the samples contained the bump and half did not. The samples divided into two sets, 22K as training set and 8K as testing set. The samples were converted to 1D vectors by using only flux values.

Yuan designed fully connected and convolutional neural network architectures to identify absorption bumps in the dataset. The 1D vectors were fed to the fully connected networks with different layer and neuron configurations. The output layers were Softmax with two values indicating with or without absorption bump. Initial learning rate was selected as 0.01 and applied step decreasing policy. The best

accuracy (96.75%) was achieved by two hidden layer network with 400 neurons per layer. Another architecture was 2D CNN based on modified version of AlexNet [30] and GoogLeNet [31]. The 1D input vectors were padded and folded into 69*69 matrices to make it possible for filters in convolutional layers to detect localized patterns in flux values. Two fully connected layers were used after the convolutional layers. The activation function was ReLU in the convolutional layers. The CNN models were tested for different configurations and the best model achieved 99.404% accuracy. The best model had 4 convolution layers with 50 filters and size of the filters were 5*5, 3*3, 3*3 and 3*3 respectively.

3. METHODS

In this work, concentration of a specific substance in different chemical solutions was predicted by using RF, PLSR, GBR, CNN and LSTM models trained on the absorption spectrums of the solutions. A large dataset was created for training and testing the models. Before the training, the dataset was scaled, abnormal spectrums were cleaned and the wavelength regions where the specific substance had no effect were removed from the spectrums. Coefficient of determination, mean absolute error and mean squared error (MSE) metrics were used to train and compare the models. The parameters of the models were optimized for the highest R^2 and the lowest MAE. In this section, the details about the dataset, data preprocessing methods and the learning models were discussed.

3.1. Dataset Preparation

There are three types of chemical solutions produced by a chemical process. The color and molecular structure of each solution are different. There is a specific X substance dissolved homogeneously in these solutions and the concentration of the substance is different due to the chemical process.

Total of 6167 solutions and transparent cuvettes were prepared for all solution types and total of 31 measurement systems were developed as shown in Figure 3.1. The solutions were poured into the cuvettes with 1 cm path-length and placed on the cuvette holder. Optical lenses and cuvettes were checked and cleaned before the measurements. UV-Visible light was transferred to the solution via fiber cable. The light passed through the sample was measured by the spectrometer with 1 nm sensitivity. Temperature of the sample and the room where the measurement was taken were recorded since concentration of X substance was affected by environmental conditions as explained in Section 1.3. The actual concentration of the X substance was measured in a laboratory environment with ± 1 percent sensitivity.

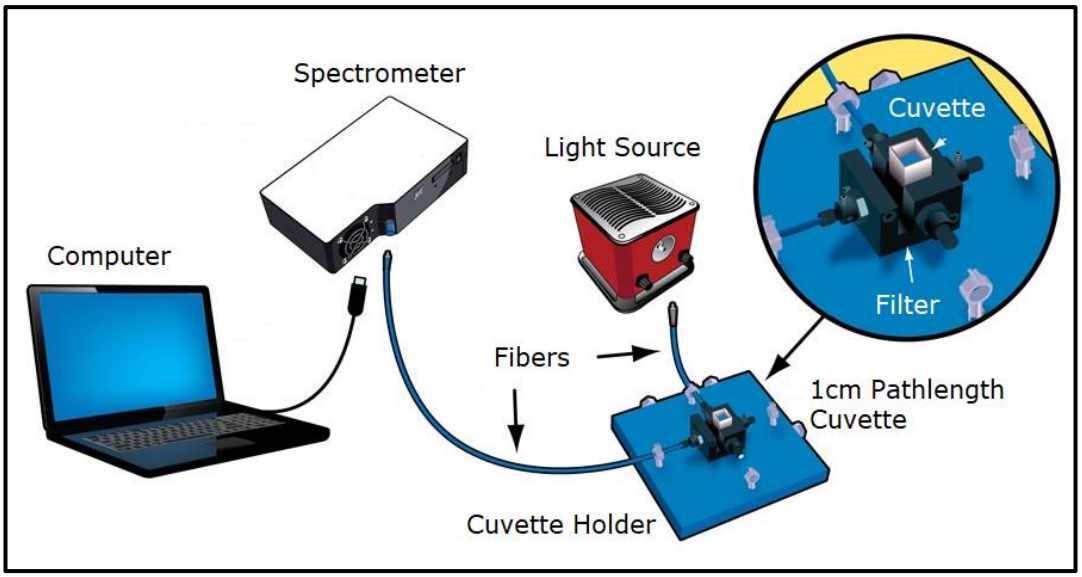

Figure 3.1: Absorbance spectrum measurement system including spectrometer, UV-Visible light source, 1 cm path-length cuvette, cuvette holder, fibers and computer.

Reference and dark measurements are required for the accurate calculation of absorption spectrums. Equation (3.1) shows the absorption calculation with dark and reference spectrums. Dark measurement (I_{dark}) is performed with the light source switched off and without cuvette. The light source is allowed to stay opened for 30 seconds to come to thermal equilibrium and then reference (I_{ref}) and sample (I_{sam}) measurements are taken with empty cuvette and filled cuvette, respectively. This technique eliminates the noises that were caused by the measurement system and environment.

$$A = log_{10}(\frac{I_{sam} - I_{dark}}{I_{ref} - I_{dark}}) = \varepsilon(\lambda)cl \tag{3.1}$$

For entire dataset, Figure 3.2 demonstrates distribution of the samples by concentration of the X substance. As seen in the figure, most of the samples have concentration between 90 and 120 due to nature of the chemical process. For each chemical type and concentration, there are about 40 different samples. The number of samples for each chemical group is almost the same. There are missing samples for some concentrations and there are not any samples with concentration above 140.

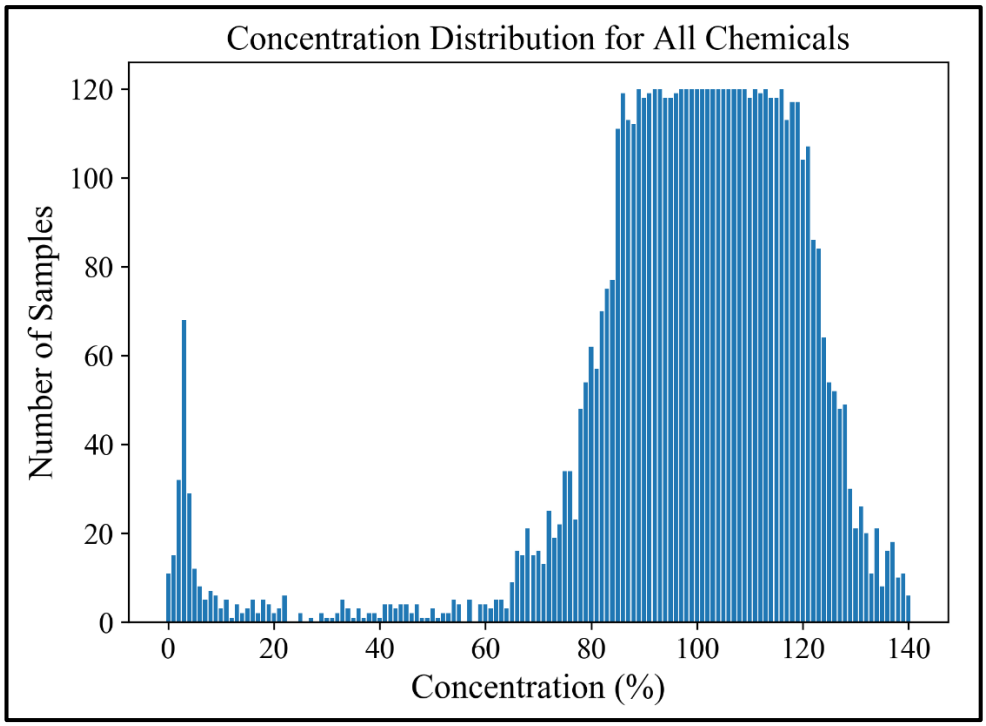

Figure 3.2: Distribution of the entire dataset by concentration values.

3.1.1. Spectrum Preprocessing

The calculated absorption spectrums were smoothed and cleaned from noises by Savitsky-Golay [19] method. The wavelength regions where the X substance had no effect were removed from the absorption spectrums leaving feature vectors with 164 dimensions. The all vectors were normalized by subtracting the global mean and dividing by the global standard deviation ensuring that the values were mostly in the range -1 to 1. This normalization was done by StandardScaler algorithm in Scikit-Learn [32]. Scaled absorption spectrums of randomly selected three samples are shown in Figure 3.3.

3.1.2. Outlier Removal

Spectral outliers were identified by PLSR modeling trained on the entire dataset without preprocessing. For the model, implementation of the NIPALS algorithm [3] in Scikit-Learn [32] was used, without scaling, a maximum of 100.000 iterations and a tolerance of 10^{-16}. The best number of principal components was achieved for 13 as shown in Figure 3.4. The all dataset was applied to 10-fold cross-validation for the number of principal components between 1 and 30. The number that gave the highest

average R^2 score for the cross-validation sets was used to train PLSR model on the entire dataset and the samples having more than $\tau = 3.7$ times the standard deviation of the absolute error of prediction identified as outliers.

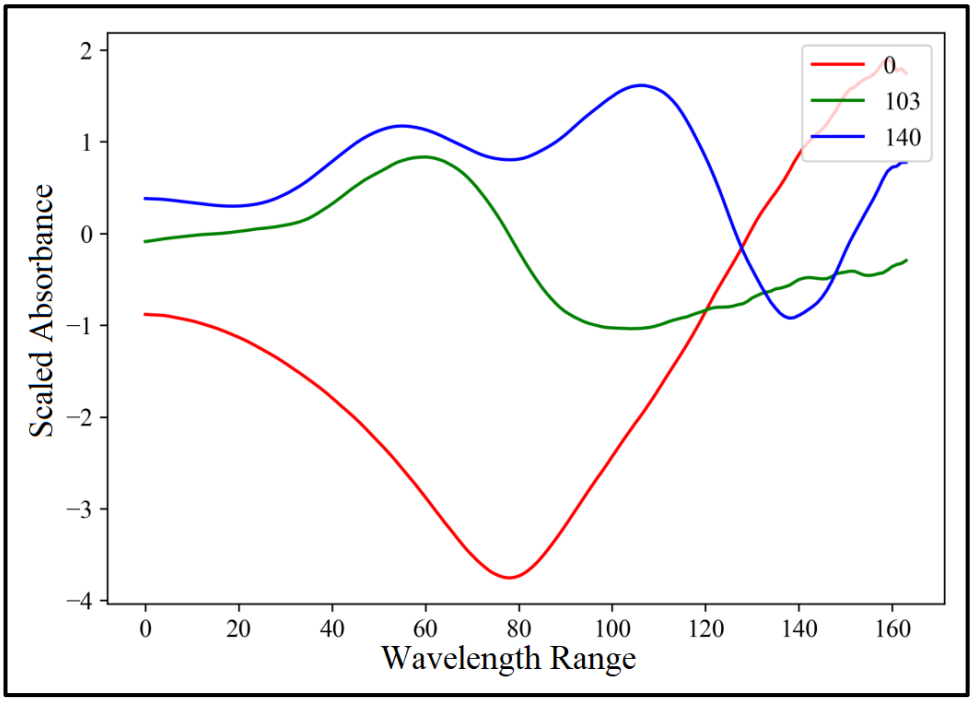

Figure 3.3: Scaled absorption spectrums of three samples with concentration of 0%, 103% and 140%.

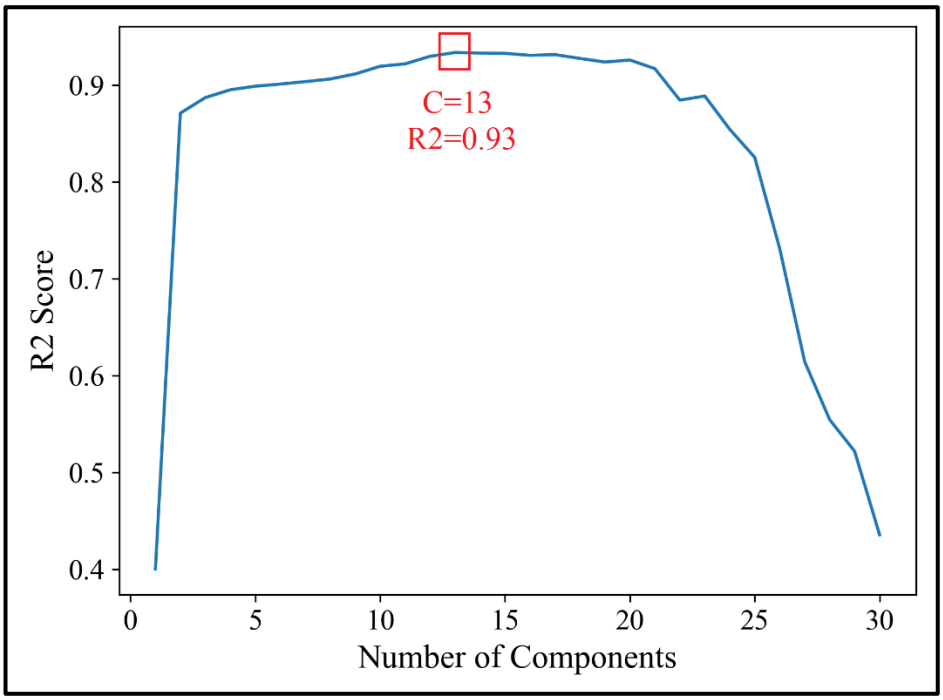

Figure 3.4: Average R^2 scores of PLSR model for 10-fold cross-validation sets. The best score (0.93) was achieved for 13 components.

Equation (3.2) shows the outlier decision function where P_s and T_s are predicted and true concentration of a specific sample S, τ is outlier boundary constant, P and T are arrays containing all predicted and true concentrations for the entire dataset, $|P - T|$ is the resulting array calculated by subtracting elements of P and T arrays at the same indexes and getting absolute values of them. $Std(|P - T|)$ is standard deviation of $|P - T|$ array whose all elements are positive.

$$\frac{|P_s - T_s|}{Std(|P - T|)} \geq \tau \tag{3.2}$$

Equation (3.2) was tested for various τ constants and the best value was found to be 3.7 for the entire dataset. For $\tau = 3.7$, total of 131 samples were identified and removed from the dataset as shown in Figure 3.5.

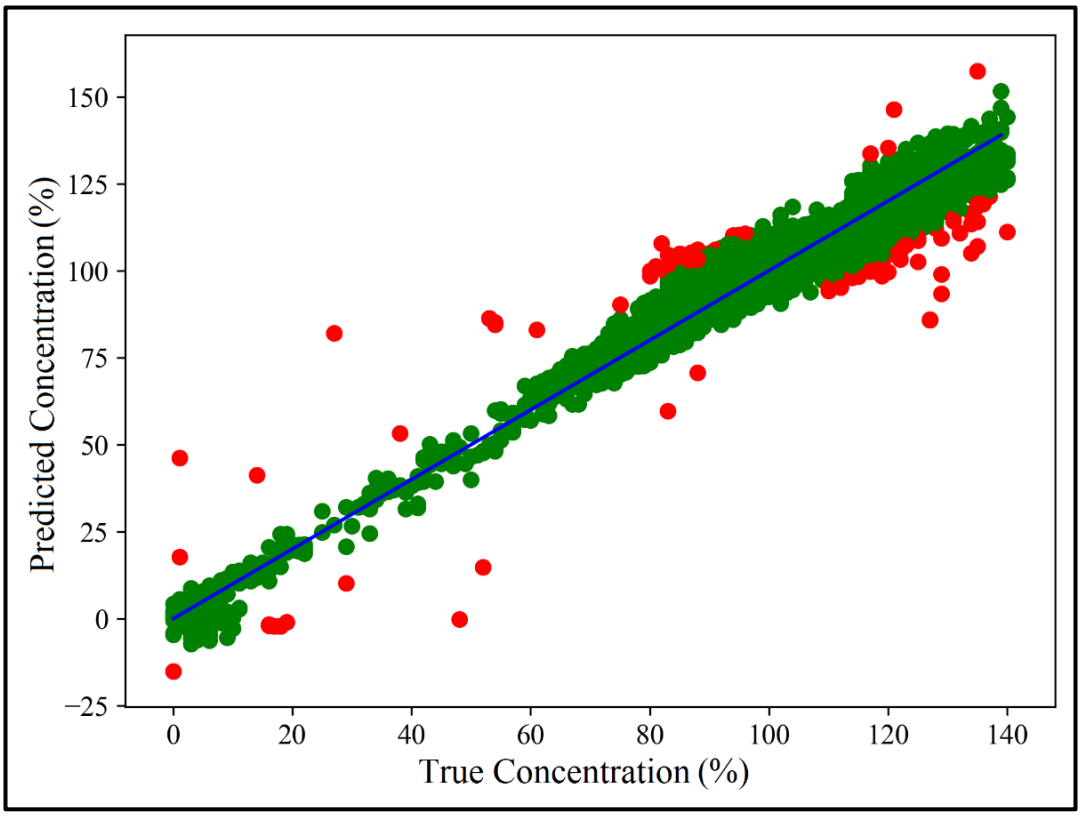

Figure 3.5: Spectral outliers are detected by PLSR model trained on the entire dataset. Blue line represents $y = x$, red and green dots indicates abnormal and normal spectrums, respectively.

3.2. Partial Least Squares Regression

Partial least squares regression is a method to reduce a collection of features to a smaller set of components uncorrelated with each other. It generalizes and associates features by using principal component analysis (PCA) and multiple linear regression (MLR) [3]. It is a good alternative to the more classical MLR and PCA methods since it is more robust and the model parameters do not vary very much for new samples. It is widely used in the drug, plastic, chemical and food industries. As explained in Section 2, it is commonly used to model the relation between spectrums having many correlated variables.

3.2.1. Prerequisite Notions and Notations

The purpose of PLSR is to predict Y_{OxD} matrix from X_{OxP} matrix where O is observations, D is dependent variables and P is predictors extracted from O observations. The matrix X is possibly singular and the regression becomes ineffective due to multicollinearity when the number predictors are larger than the number of observations. Thus, PCA is performed on X to get a smaller set of uncorrelated elements. The matrix X is decomposed via singular value decomposition which is defined as $X = U\Delta V^T$ and $U^T U = V^T V = I$ where I is identity matrix and Δ is a diagonal matrix with singular values. The columns of U and V are left and right singular vectors, respectively. The singular vectors are ordered by their related singular values that correspond to the square root of the variance of X which is explained by the singular vectors. The left singular vectors can be used to predict Y by using MLR since the orthogonality of the singular vectors prevents the multicollinearity issue.

3.2.2. Decompositions of Variables

The matrices X and Y are decomposed as $X = TP^T$ with $T^T T = I$ where I is identity matrix, T is score matrix and P is the loading matrix which is not orthogonal in PLSR. Similarly, the matrix Y is estimated by $Y' = TBC^T$ where B is a diagonal matrix with regression weights and C is a weight matrix with dependent variables. The columns of the matrix T is called latent vectors. Two sets of weights w and c are

obtained by maximizing their covariance, and then linear combination of the columns of X and Y is created by using them. The purpose is to find an initial pair of vectors that maximizing $b = t^T u$ where $t = Xw$ and $u = Yc$ with constraint of $w^T w = 1$ and $t^T t = 1$. At each step, a latent vector is found and subtracted from X and Y until X becomes null matrix.

3.2.3. Regression Procedure

The PLSR algorithm is presented below. The matrices $K = X$ and $L = Y$ are created, column centered and normalized. The vector u is initialized randomly.

- Estimate weights of X matrix by $w = K^T u$
- Estimate factor scores of X matrix by $t = Kw$
- Estimate weights of Y matrix by $c = L^T t$
- Estimate scores of Y matrix by $u = Lc$

If t converges, then $b = t^T u$ is calculated to predict Y from t and the factor loadings for X is computed by $p = K^T t$, otherwise the first step is executed again. The vector t is removed from K and L by $K = K - tp^T$ and $L = L - btc^T$. The b value is stored as a diagonal element of the matrix B and the vectors t, u, w, c and p are stored in the related matrices. The procedure continues beginning from the first step until K is a null matrix. Finally, the dependent variables are estimated by multivariate regression $Y' = TBC^T = X\overline{P^T}BC^T$ where $\overline{P^T}$ is Moore-Penrose pseudo inverse of P^T.

3.3. Random Forests Regression

Random forests regression is an ensemble learning method that fits many regression trees on randomly created subset of dataset and uses averaging to improve the predictive accuracy and control over-fitting [33]. It uses bagging technique that predictions are made by using and combining all decisions from a set of base models. All the base models are constructed separately using a various subsample of the data. RF models can capture non-linear relations between features but they are not very effective for datasets having sparse features. The predictions for new or unseen

samples x can be obtained by averaging the predictions from all the individual regression trees on x as shown in Equation (3.3) where the number of base learners is B, base learner is f and $R(x)$ is the predicted value.

$$R(x) = \frac{1}{B}\sum_{b=0}^{B} f_b(x) \tag{3.3}$$

3.3.1. Classification Trees

Regression tree is a type of decision tree that is a hierarchical model for supervised learning. A decision tree consists of decision nodes and leaves with discrete outcomes that label the branches as shown in Figure 3.6 [34]. At each node u, a specific condition is checked by $f_u(x)$ function and one of the branches is followed for a given input by starting at the root node. The function $f_u(x)$ returns a discrete output labeling the branches. This process continues recursively until a leaf node is reached. The value is stored in the leaf that determines the output. The decision tree is a nonparametric model since any parametric form for class densities is not used and the tree gets bigger by adding branches and leaves during learning.

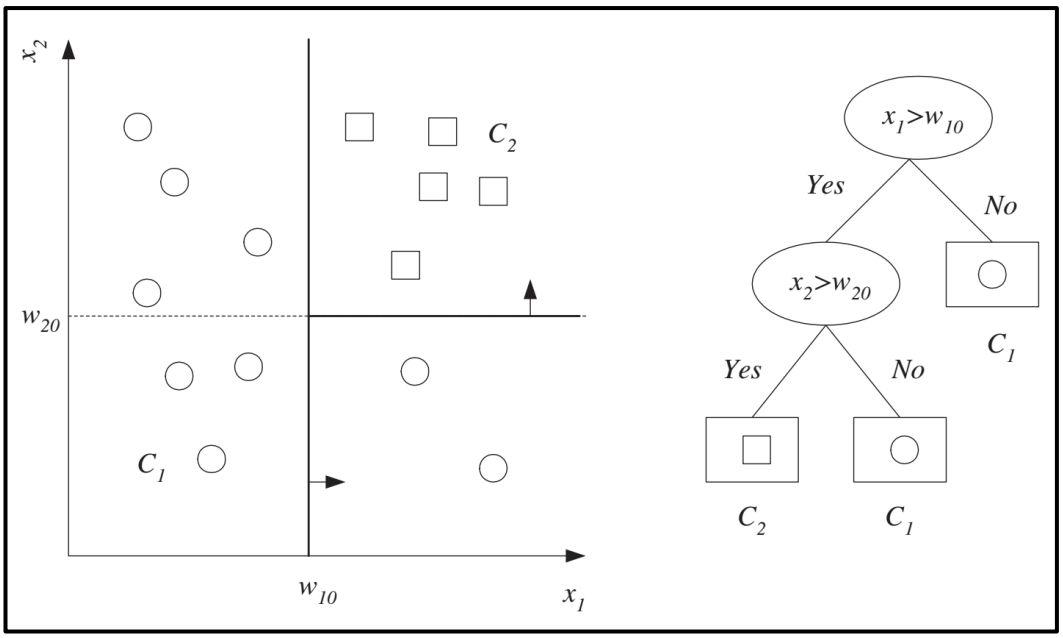

Figure 3.6: A decision tree example for a dataset. Circular shapes are the decision nodes and rectangles are leaf nodes. The univariate decision node splits throughout one axis and consecutive splits are orthogonal to each other. After the first split, $\{x|x_1 < w_{10}\}$ is pure and is not divided further.

The function $f_u(x)$ defines a separation in the d-dimensional input space and splits the input space into smaller regions by beginning from the root node. The leaves have an output label which is a class code or a numeric value for classification and regression problems, respectively. A leaf specifies a localized region in the input space. The inputs that fall in the same localized region have the similar classes for classification or numeric values for regression. The boundaries of the regions are specified by the separations in the input space.

Univariate trees use one input in each node while multivariate trees use all inputs in each node. In a univariate tree, the decision function $f_u(\cdot)$ in a node takes one input and returns n possible values. If an attribute $x_k \in \{val1, val2\}$ has two possible values then a node on that attribute has two branches. Each of the branches get one of the possible two values of the attribute. If input x_i is numeric, then decision function is $f_u(x): x_i > \alpha_{u0}$ where α_{u0} is a proper threshold value.

Quality of splits is determined by impurity measures. A split is considered pure if all the instances that go through a branch belong to the same class. Specifically, a node u is considered pure if $p_u^j = N_u^j / N_u$ is either 0 or 1 where N_u is the number of training samples that hit the node and N_u^j of N_u is the number of samples that belong to class C_j. If the node u is pure for the class C_j, then a leaf node that labeled with the class C_j can be added to the node. If the node is not pure, then the samples can be split to increase purity. Multiple split points are probable for a numeric attribute and the split that maximizes purity should be selected to create the smallest tree.

The algorithm of classification tree construction is shown in Figure 3.7 [34]. Node entropy for a node u can be defined as Equation (3.4) where K is the number of classes. For a two-class problem, if $p^1 = 1$ and $p^2 = 0$, then all the samples belong to the same class.

$$\varepsilon_u = -\sum_{i=1}^{K} p_u^i \, log_2 \, p_u^i \tag{3.4}$$

Split entropy for a node u can be defined as Equation (3.5) where n is the number of possible values for a discrete attribute, N_{uj} of N_u is the number of samples taking branch j, N_{uj}^i of N_{uj} is the number of samples that belong to class C_i and the estimate for the probability of class C_i is $p_{uj}^i = N_{uj}^i / N_{uj}$.

$$\varepsilon'_u = -\sum_{j=1}^{n} \frac{N_{uj}}{N_u} \sum_{i=1}^{K} p_{uj}^{i} \, log_2 \, p_{uj}^{i} \tag{3.5}$$

```
GenerateTree(𝒳)
    If NodeEntropy(𝒳)< θ_I
      Create leaf labelled by majority class in 𝒳
      Return
    i ← SplitAttribute(𝒳)
    For each branch of x_i
      Find 𝒳_i falling in branch
      GenerateTree(𝒳_i)

SplitAttribute(𝒳)
    MinEnt← MAX
    For all attributes i = 1,...,d
        If x_i is discrete with n values
          Split 𝒳 into 𝒳_1,...,𝒳_n by x_i
          e ← SplitEntropy(𝒳_1,...,𝒳_n)
          If e<MinEnt MinEnt ← e; bestf ← i
        Else /* x_i is numeric */
          For all possible splits
              Split 𝒳 into 𝒳_1,𝒳_2 on x_i
              e←SplitEntropy(𝒳_1,𝒳_2)
              If e<MinEnt MinEnt ← e; bestf ← i
    Return bestf
```

Figure 3.7: Tree construction algorithm for a classification problem.

The impurity scores are measured for all attributes and split positions to find the tree with minimum entropy. Until all branches are pure, the tree is constructed recursively and in parallel. Specifically, the split that gives the largest increase in purity is chosen at each step. For example, the impurity difference for a node u can be measured by the difference between the impurity of data hitting the node (ε_u) and the total entropy of data hitting its branches after split (ε'_u).

Noisy datasets may cause construction of very large trees. Since the trees are constructed until they are the purest, the trees overfit. To cope with the overfitting, construction of the trees is ended when they are pure enough. The subset of data is not split further if $\varepsilon_u < \theta_I$ and this means that p_{uj}^{i} is not exactly 0 or 1 but close enough with threshold θ_p. In a case like this, a leaf node is created and labeled with the class having the largest p_{uj}^{i}. The threshold values θ_I or θ_p is known as complexity

parameter. If the complexity parameter is high, then variance gets low and a small tree is constructed, otherwise variance gets high and the tree represents the training samples accurately by growing larger.

3.3.2. Regression Trees

Regression tree is built in a very similar way as a classification tree. The difference is that the impurity measure is replaced by a proper one for regression. The algorithm presented in Figure 3.7 can be modified to train a regression tree by replacing entropy function with MSE and class labels with averages. The partial function $b_u(x)$ is defined in Equation (3.6) where $x \in X_u$ is all samples hitting the node u. In regression, the training dataset is defined as $X = \{x^t, r^t\}$ where $r^t \in \mathbb{R}$ and $r^t = g(x^t) + \epsilon$.

$$b_u(x) = \begin{cases} 1, & if\ x \in X_u\ reaches\ node\ u \\ 0, & otherwise \end{cases} \tag{3.6}$$

Quality of a split can be measured by MSE from predicted value. If p_u is predicted value inside a node u, then the variance at the node is defined in Equation (3.7) where $N_u = |X_u| = \sum_t b_u(x^t)$.

$$\tau_u = \frac{1}{N_u} \sum_t (r^t - g_u)^2 b_u(x^t) \tag{3.7}$$

Outputs of the instances hitting a node are averaged as in Equation (3.8). If the error in a node is feasible ($\tau_u < \theta_r$) then a leaf node containing the g_u value is added, otherwise data that hits the node u is divided further such that the sum of the errors in the branches is minimum. As in classification, the attribute or split threshold for a numeric attribute that minimizes the error is searched recursively.

$$g_u = \frac{\sum_t b_u(x^t) r^t}{\sum_t b_u(x^t)} \tag{3.8}$$

The partial function $b_{uh}(x)$ is defined in Equation (3.9) where $X_u = \bigcup_{h=1}^{n} X_{uh}$ and X_{uh} is a subset of X_u that uses branch h.

$$b_{uh}(x) = \begin{cases} 1, & if\ x \in X_{uh}\ reaches\ node\ u\ and\ uses\ brach\ h \\ 0, & otherwise \end{cases} \tag{3.9}$$

Equation (3.10) defines the predicted value g_{uh} in branch h of node u and the error caused by the split is defined in Equation (3.11).

$$g_{uh} = \frac{\sum_t b_{uh}(x^t) r^t}{\sum_t b_{uh}(x^t)} \tag{3.10}$$

The decrease in error caused by a split is defined as the difference between τ_u and τ'_u which are denoted in Equation (3.7) and Equation (3.11), respectively. The split that caused the maximum decrease in error is searched recursively.

$$\tau'_u = \frac{1}{N_u} \sum_h \sum_t (r^t - g_{uh})^2 b_{uh}(x^t) \tag{3.11}$$

3.4. Gradient Boosting Regression

Gradient boosting regression is an additive ensemble learning method that optimizes many regression trees in a forward stage-wise fashion [35]. In each stage, a regression tree is added to the model that is fit on the negative gradient of a differentiable loss function. It uses boosting technique that new predictors learn from mistakes committed by previous predictors. The trees are not trained independently, but sequentially. The trees are constructed by selecting the best split points according to purity scores, and the loss is minimized. The trees in the models are not changed and the new trees are added one by one. For the new trees, a gradient descent procedure is applied to minimize the loss. The output of the new tree is added to the output of the existing series of trees to boost the final output of the model.

3.4.1. Function Estimation

Let $\{x_i, y_i\}_{i=1}^N$ denote the training set that x_i is a random input or explanatory variable and $y_i \in \mathbb{R}$ is a random output or response. Using the training set of known (y, x) values, the goal is to find $F'(x)$ of function $F^*(x)$ that maps x to y by minimizing the value of a specific loss function $L(y, F(x))$. The generic function

$h(x; a)$ in Equation (3.12) is a simple parameterized function of the input variables x, determined by $a = \{a_1, a_2, a_3, \dots\}$ whose elements change in the joint values a_m for these parameters. The function $h(x; a)$ can be a small regression tree or another model.

$$F(x; \{\beta_m, a_m\}_1^M) = \sum_{m=1}^{M} \beta_m h(x; a_m) \tag{3.12}$$

The parameterized model $F(x; P)$ is optimized by Equation (3.14) where $\phi(P) = E_{y,x} L(y, F(x; P))$ and $F^*(x) = F(x; P^*)$. Numerical optimization methods are employed to solve $\phi(P)$ for the parameters in the form $P^* = \sum_{m=0}^{M} p_m$ where p_0 is initial value and $\{p_m\}_{i=1}^{M}$ are successive steps that depends on the previous steps. The steps are defined by the optimization method.

$$P^* = arg \min_P \phi(P) \tag{3.13}$$

3.4.2. Steepest Descent

Steepest-descent is a popular numerical optimization method that defines the increments $\{p_m\}_{i=1}^{M}$. The current gradient g_m in Equation (3.14) is calculated where $P_{m-1} = \sum_{i=0}^{m-1} p_i$. The negative gradient $-g_m$ denotes steepest-descent direction and ρ_m is called "line search" along that direction. The step is defined as $p_m = -\rho_m g_m$ where $\rho_m = \arg\min_P \phi(P_{m-1} - \rho g_m)$.

$$g_m = \{g_{jm}\} = \left\{ \left[\frac{\partial \phi(P)}{\partial P_j} \right]_{P = P_{m-1}} \right\} \tag{3.14}$$

In the function space, numerical optimization is applied with a nonparametric approach. The function $F(x)$ is computed at each point x and considered as a parameter. The goal is to minimize Equation (3.15) with respect to $F(x)$.

$$\phi\big(F(x)\big) = E_y\big[L\big(y, F(x)\big)\big|x\big] \tag{3.15}$$

There is infinitive number of such parameters in function space, but only a finite number $\{F(x_i)\}_1^N$ is available in datasets. Using the numerical optimization paradigm, Equation (3.16) solves the problem where $f_o(x)$ is initial guess and $\{f_m(x)\}_1^M$ are incremental functions (steps or boosts) defined in the optimization method.

$$F^*(x) = \sum_{m=0}^{M} f_m(x) \tag{3.16}$$

In the case of steepest-descent, the incremental function $f_m(x)$ is defined as $f_m(x) = -\rho_m g_m(x)$ where the line search ρ_m and the gradient $g_m(x)$ are described in Equation (3.17) and Equation (3.18), respectively.

$$g_m(x) = E_y\left[\frac{\partial L(y, F(x))}{\partial F(x)} | x\right]_{F(x) = F_{m-1}(x)} \tag{3.17}$$

In Equation (3.17), the previous step is computed by $F_{m-1}(x) = \sum_{i=0}^{m-1} f_i(x)$. The multiplier ρ_m in Equation (3.18) is acquired by the line search.

$$\rho_m = arg \min_{\rho} E_{y,x} L(y, F_{m-1}(x) - \rho g_m(x)) \tag{3.18}$$

3.4.3. Approximations for Finite Datasets

Accurate calculation of $E_y[. | x]$ is not possible by its data value at each x_i since the dataset is composed of finite samples $\{x_i, y_i\}_{i=1}^N$. Nearby data points can be used to create a smooth solution to minimize the expected loss by using parameter optimization.

$$(\beta_m, a_m) = arg \min_{\beta, a} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \beta h(x_i; a)) \tag{3.19}$$

$$F_m(x) = F_{m-1}(x) + \beta_m h(x; a_m) \tag{3.20}$$

It is hard to solve Equation (3.19) for a specific loss function $L(y, F)$ and base learner $h(x; a)$ because of the smoothness constraint. This constraint can be applied to

the unconstrained (rough) solution by fitting $h(x;a)$ to the pseudo-responses $\{\tilde{y}_i = -g_m(x_i)\}_{i=1}^N$ where the gradient is defined in Equation (3.21). This allows least squares function minimization in Equation (3.22) to be used instead of difficult function minimization in Equation (3.19), followed by a single parameter optimization that uses the criterion in Equation (3.23). Therefore, a feasible least squares algorithm can be applied to any $h(x;a)$ to solve Equation (3.22) and any differentiable loss function can be optimized by using this approach. As a result, the update rule model $F_m(x)$ becomes as Equation (3.24).

$$-g_m(x_i) = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x) = F_{m-1}(x)} \tag{3.21}$$

$$a_m = \underset{a,\beta}{arg\,min} \sum_{i=1}^{N} [-g_m(x_i) - \beta h(x_i;a)]^2 \tag{3.22}$$

$$\rho_m = \underset{\rho}{arg\,min} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \rho h(x_i;a_m)) \tag{3.23}$$

$$F_m(x) = F_{m-1}(x) + \rho_m h(x;a_m) \tag{3.24}$$

3.4.4. Regression Algorithms

Least squares boosting regression is shown in Figure 3.8 [35]. The loss function is defined as $L(y,F) = (y-F)^2/2$. The pseudo-response is $\tilde{y}_i = y_i - F_{m-1}(x_i)$. The current residuals are fit in line 4 of the algorithm. The line search at line 5 of the algorithm yields $\rho_m = \beta_m$ where β_m is the minimizing β of line 4. Therefore, gradient boosting on squared-error loss gives the stage-wise method of iteratively fitting the current residuals.

Each of base learners is an J-terminal regression tree that has additive form defined in Equation (3.25), where $\{R_j\}_1^J$ are disjoint regions covering the space of all joint values of x. The regions are represented by the terminal nodes of related tree. The function $1(\cdot)$ return 0 for false, and 1 for true argument. The coefficients $\{b_j\}_1^J$ and values of them are the parameters of the base learner in Equation (3.25). The parameters define the boundaries of the regions and represent the splits at the

nonterminal nodes of the tree. Since the regions are disjoint, Equation (3.25) is the same as the prediction rule "if $x \in R$ then $h(x) = b_j$".

$$h\left(x; \{b_j, R_j\}_1^J\right) = \sum_{j=1}^{J} b_j 1(x \in R_j) \tag{3.25}$$

$F_0(\mathbf{x}) = \bar{y}$
For $m = 1$ to M do:
$\quad \tilde{y}_i = y_i - F_{m-1}(\mathbf{x}_i), \quad i = 1, N$
$\quad (\rho_m, \mathbf{a}_m) = \arg\min_{\mathbf{a},\rho} \sum_{i=1}^{N} [\tilde{y}_i - \rho h(\mathbf{x}_i; \mathbf{a})]^2$
$\quad F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$
endFor

Figure 3.8: Gradient boosting algorithm for least squares loss function.

The terminal nodes of the tree define the regions $\{R_{jm}\}_1^J$ at the mth iteration. The regions are needed for computation of the pseudo-responses $\{\tilde{y}\}_1^N$ by least squares. The $\{b_{jm}\}$ are coefficients of least squares where $b_{jm} = ave_{x_i \in R_{jm}} \tilde{y}_i$. The scaling factor ρ_m is the solution of "line search". The update equation is like Equation (3.26) where $\gamma_{jm} = \rho_m b_{jm}$.

$$F_m(x) = F_{m-1}(x) + \sum_{j=1}^{J} \gamma_{jm} 1(x \in R_{jm}) \tag{3.26}$$

Using the disjoint property of the regions, γ_{jm} can be calculated by using Equation (3.27). For the loss function L and the current approximation $F_{m-1}(x)$, this is the optimal constant update in each terminal node region.

$$\gamma_{jm} = arg\,\underset{\gamma}{min} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma) \tag{3.27}$$

Figure 3.9 denotes the regression algorithm of least absolute deviations (LAD) [35]. The γ_{jm} is defined by Equation (3.28) where $x_i \in R_{jm}$. It is the median of the current residuals in the jth terminal node at the mth iteration.

$$\gamma_{jm} = median\ \{y_i - F_{m-1}(x_i)\} \tag{3.28}$$

$F_0(\mathbf{x}) = median\{y_i\}_1^N$
For $m = 1$ to M do:
$\quad \tilde{y}_i = sign(y_i - F_{m-1}(\mathbf{x}_i)),\ i = 1, N$
$\quad \{R_{jm}\}_1^J = J$–terminal node $tree(\{\tilde{y}_i, \mathbf{x}_i\}_1^N)$
$\quad \gamma_{jm} = median_{\mathbf{x}_i \in R_{jm}} \{y_i - F_{m-1}(\mathbf{x}_i)\},\ j = 1, J$
$\quad F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \sum_{j=1}^{J} \gamma_{jm} 1(\mathbf{x} \in R_{jm})$
endFor
end Algorithm

Figure 3.9: GBR algorithm for LAD regression.

By the least squares criterion, a regression tree is used to estimate the sign of the current residuals $y_i - F_{m-1}(x_i)$. The median of the residuals in each of the terminal nodes is used to update the approximation. The algorithm is robust since the trees use only order on the input variables x_j and the pseudo-responses $\tilde{y}_i \in \{-1, 1\}$ have only two values.

3.4.5. Regularization

Exact fit on the training data and reducing the loss after some point can cause the overall expected loss to increase. Regularization techniques can be used to eliminate such overfitting by putting some limitation to fitting procedures. A common regularization parameter is the number of components M which can be optimized by using a separate test set.

Shrinkage method provides more promising result than restricting the number of components M [36]. Additive models defined in Equation (3.12) are built in a forward stage-wise manner as defined in Equation (3.19) and Equation (3.20), respectively. The shrinkage is defined in Equation (3.29) where λ is between 0 and 1. Updates are scaled by the learning rate parameter λ. Increasing λ decreases the best value of M and increasing M requires more computational power. Increasing λ too much can cause overfitting behaviors.

$$F_m(x) = F_{m-1}(x) + \lambda \rho_m h(x; a_m) \tag{3.29}$$

3.5. Convolutional Neural Networks

Artificial neural networks (ANNs) are computational processing systems that are heavily inspired by the human brain. ANNs are usually composed of many interconnected computational nodes or neurons as shown in Figure 3.10 [38]. The neurons operate in a distributed manner to learn from the input and optimize their final outputs. Inputs are usually loaded in a multidimensional vector form and fed to the hidden layers. The hidden layers make decisions using the information in previous layers and try to improve the final output.

DNNs are like ANNs that they have neurons that self-optimize by learning and they have multiple hidden layers stacked on each other. The neurons perform an operation such as a scalar product by using their inputs. They have learnable weights and biases. CNNs are special type of DNNs containing convolutional and subsampling layers mostly followed by fully connected layers [37], [38]. CNNs are usually used in the field of pattern recognition for images.

Traditional forms of ANN are not very effective for complex multidimensional input data. For instance, MNIST database of handwritten digits has images with dimension of 28x28 which makes a single neuron to hold 784 weights. For ANNs, this is mostly manageable, but the number of weights in a neuron is 16,384 for images with dimension of 128x128 and the network need to be much larger to deal with this input.

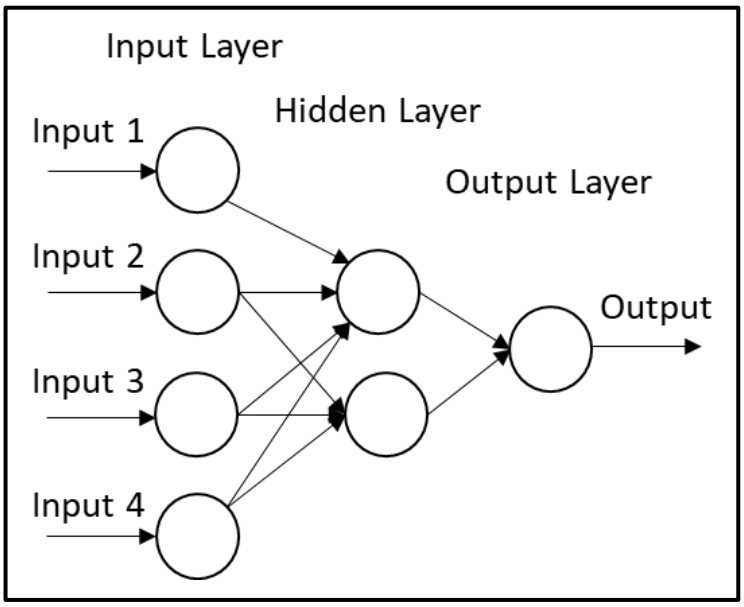

Figure 3.10: A three-layered feedforward neural network with an input layer, a hidden layer and an output layer.

3.5.1. Core Layers

A traditional CNNs have three types of layers which are convolutional, pooling and fully connected layers. A basic CNN architecture for MNIST is shown in Figure 3.11 [38]. The input layer holds pixel values of the image. The convolutional layer calculates outputs of neurons that are connected to local regions of the input image by scalar product of their weights and the connected region in the input. ReLU [5] activation function is applied in an elementwise way to output of the activation of previous layer. The pooling layer performs downscaling operation for given input and reduces the number of parameters for that activation. Fully connected layers calculate class scores from the activations. As a summary, CNNs can transform raw input to class scores for classification and regression by using many convolutional and pooling layers.

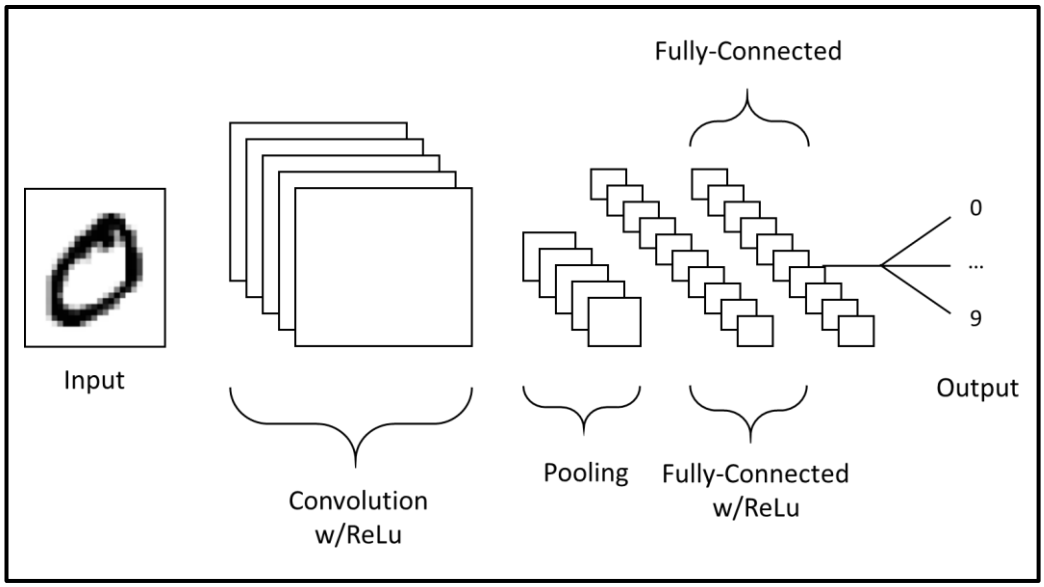

Figure 3.11: An example CNN architecture with five layers for classification of MNIST images.

Convolutional layers are very important for CNN architectures. The layer parameters are used with learnable kernels that have generally small spatial dimensions and spread along the depth of the input. The layer convolves each filter over the input and produces a 2D activation map as shown in Figure 3.12 [38]. While the kernel is slid on the input, the scalar product is calculated for each value in the kernel. In this way, the network learns kernels that activate when they detect a particular feature at a given spatial position of the input. These are generally known as

activations. Full output volume of the convolutional layer is created by the kernel with an associated activation map placed along the depth dimension.

Convolutional layers considerably reduce the complexity of the model by optimizing their outputs. These are optimized by the depth, the stride and zero-padding.

The depth corresponds to the number of filters and each of them learns how to detect something different in the input. Reducing this the depth can significantly minimize the total number of neurons in the network, but this makes the model lose the pattern recognition capabilities.

The stride is the step number for the filter movement. When the stride is 1, then the filters are moved one pixel at once. When the stride is 2, then the filters are slid 2 pixels at a time. In this way, output volumes become spatially small. Small stride values can cause the receptive fields of neurons to overlap and produce large activations.

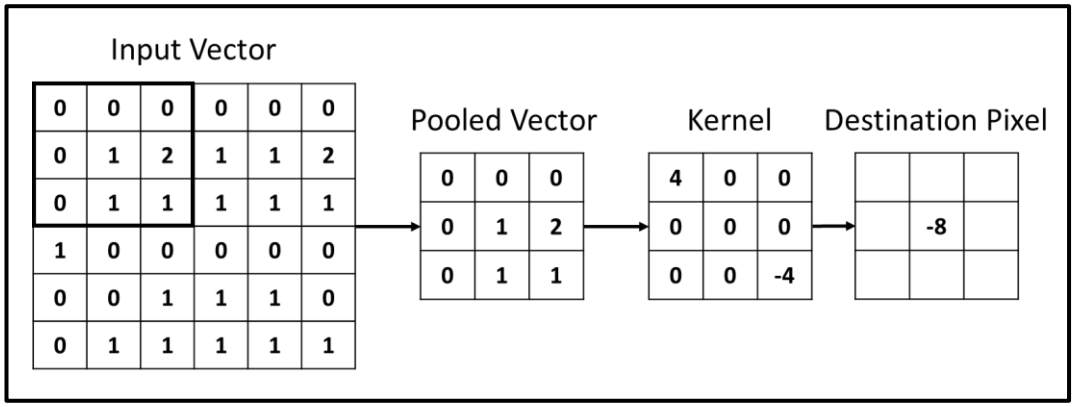

Figure 3.12: A visual representation of a convolutional layer.

The process of padding the border of the input is called zero-padding. It is a useful method to control the dimensionality of the output volumes. It is commonly used to preserve width and height of the input and output volumes.

Dimensions of the convolutional layers are calculated by Equation (3.30) where the size of input is $W_1 \; x \; H_1 \; x \; D_1$, the receptive field size is R, the number of filters is K, the amount of zero-padding is Z and the stride is S.

$$W_2 = \frac{(W_1 - R) + 2Z}{S} + 1, H_2 = \frac{(H_1 - R) + 2Z}{S} + 1, D_2 = K \qquad (3.30)$$

Parameter sharing assumes that if one feature is useful to compute at some spatial position (x, y), then it is likely to be useful to compute at another position (x', y'). This technique massively reduces the number of parameters produced by the convolutional layers since the neurons in each depth slice use the same weights and bias.

Pooling layers reduce the dimensionality of a given input. Thus, the computational complexity of the models is decreased. This layer is performed on each activation map in the input by using max or average function as shown in Figure 3.13. The pooling layer accepts a volume of size $W_1 x H_1 x D_1$, the stride S and its spatial extent F, and then produces $W_2 x H_2 x D_2$ where $W_2 = W_1 - F/S + 1$, $H_2 = H_1 - F/S + 1$ and $D_2 = D_1$. The depth of the volume is maintained.

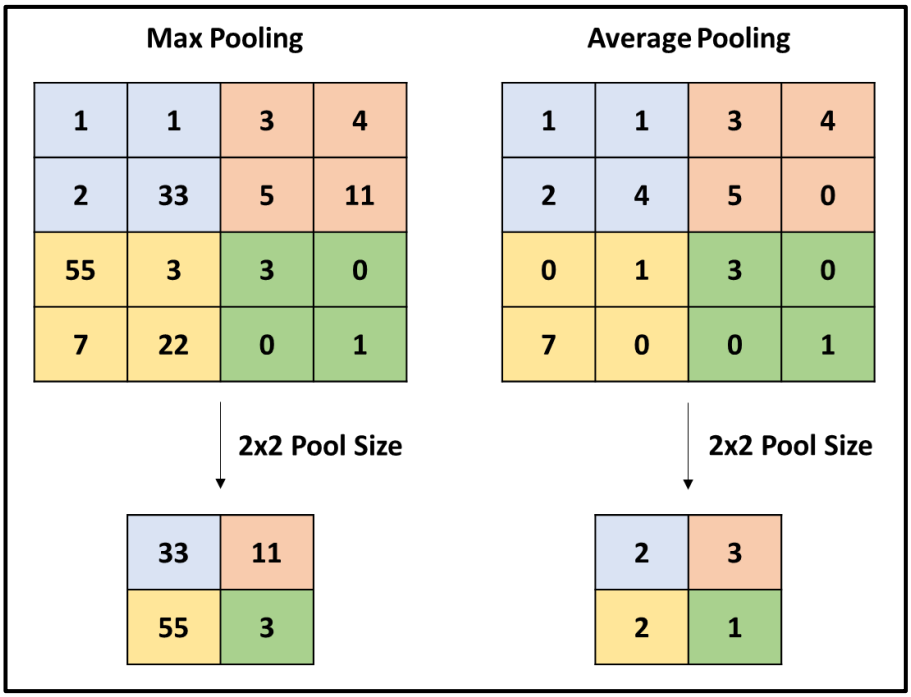

Figure 3.13: Max and average pooling examples. The spatial extent (F) and the stride (S) parameters are 2.

A fully connected layer has neurons that are fully connected to all activations in the previous layer. This is similar to the way that neurons are organized in usual forms of ANNs as shown in Figure 3.10. It is the final learning stage which maps extracted features to desired outputs. Activations of the neurons can be calculated by using a matrix multiplication, followed by a bias offset.

3.5.2. Regularization

There are several regularization techniques to deal with overfitting in CNN architectures. Some of these methods are L1/L2, dropout, noise and batch normalization. The methods are only active during training stage.

Dropout [39] is a method to prevent overfitting in CNNs. As shown in Figure 3.14, the main purpose is to randomly drop neurons and connections of them with a probability p from the neural network. It effectively prevents overfitting by reducing correlation between neurons.

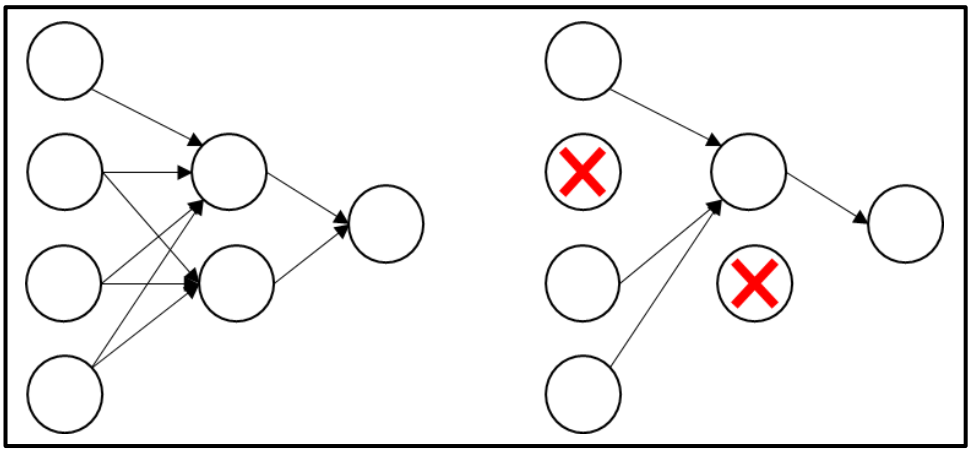

Figure 3.14: An example of dropout operation applied on the network at left. The ignored neurons are shown as red.

Batch normalization [40] makes the neural networks robust to bad weight initializations. It is generally inserted right before activation layers and it reduces covariance shift by normalizing and scaling inputs to either the range of $[0, 1]$ or $[-1, 1]$ or to mean=0 and variance=1. The scale and shift parameters are trainable to avoid losing stability of the network. It helps networks to learn faster as they do not have to adjust to covariate shift any more. It optimizes gradient flow in the network since the gradient becomes less dependent on the scale of the parameters and their initial values.

Gaussian noise layer can be used to apply additive zero-centered Gaussian noise to the input. It is useful to deal with overfitting as a form of random data augmentation.

L1 and L2 regularization are two closely related weight penalty techniques. In the objective, they are applied to all parameters to reduce overfitting. In the network, for every weight w, $L2 = \frac{1}{2}\lambda w^2$ and $L1 = \lambda|w|$ are added to the cost function where

λ is the regularization strength. L1 regularization makes the weight vectors sparse during optimization and the features that are not needed can become very close to 0.0. This is a form of feature selection. However, L2 regularization does not prune any weights by setting them to 0.0 and heavily penalizes peaky weight vectors. L2 regularization can be used for all forms of training but L1 cannot be used easily with all of them. Trial and error must be done to determine which type of regularization is better or not for a problem.

3.5.3. Optimization Principles

Loss or cost functions are crucial for optimization of the neural networks. They define the difference between predicted and ground-true values. They answer how well the models are doing their job with the current set of weight and bias. For a regression problem, Equation (3.31) denotes MSE loss function where n is the number of training data, y_j is ground-truth value and y_j' is predicted value. The lower MSE values are better for predictive power of models.

$$MSE = \frac{1}{n}\sum_{j=1}^{n}\left(y_j - y_j'\right)^2 \tag{3.31}$$

Optimization algorithms such as SGD [17] and Adam [41] minimizes or maximizes an objective function dependent on learnable parameters of a model. The parameters are used to compute the target values from the set of predictors.

Figure 3.15 shows a simple perceptron with a single output, a nonlinear function, inputs and weights. SGD algorithm can be used for the optimization of the simple perceptron. Equation (3.32) denotes output calculation of the perceptron in Figure 3.15. Weighed sum of the inputs are given to the nonlinear function f and y is determined. An additional element is usually added to the input vector that is always equal to 1 with a corresponding additional weight which behaves as a bias.

$$y = f\left(\sum_{j=1}^{N} x_j w_j\right) = f(w^T x) \tag{3.32}$$

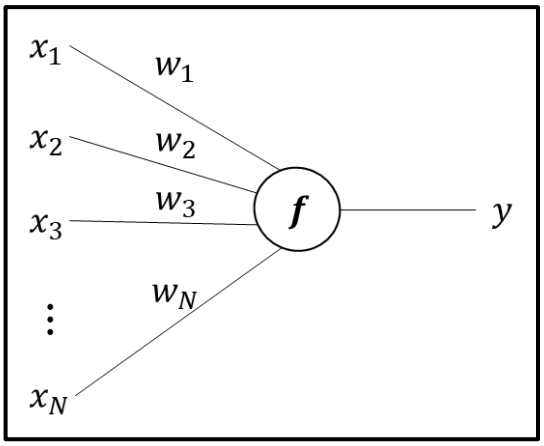

Figure 3.15: An example of simple perceptron. $x_1, x_2, .., x_N$ are inputs. $w_1, w_2, .., w_N$ are weight of connections. f is nonlinear function and y is the output.

Logistic function is a common choice for the nonlinear function f defined in Equation (3.33) and it nonlinearly binds the input and output. It is differentiable and bound between 0 and 1.

$$f(v) = \frac{1}{1 + e^{v}}, y = f(w^T x) = \frac{1}{1 + e^{w^T x}} \tag{3.33}$$

Equation (3.34) defines the derivative of the logistic function. This derivative can be used to learn the weight vector w by SGD.

$$\frac{\partial f(v)}{\partial v} = f(v) f(-v) \tag{3.34}$$

Equation (3.35) defines squared loss function that is a simple choice to measure the difference between target t and predicted output y. The aim is to minimize the loss function by finding the proper weights w.

$$L = \frac{1}{2}(t - y)^2 = \frac{1}{2}(t - f(w^T x))^2 \tag{3.35}$$

SGD optimizer updates weight parameters iteratively by using the gradient of the loss function and it continues until the minimum gradient is reached. A single data is randomly selected from the dataset and direction of the gradient is followed respect to that data. Equation (3.36) denotes gradient calculation of the loss function L with respect an arbitrary element x_i and its weight w_i.

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial v} \cdot \frac{\partial v}{\partial w_i} = (y - t) \cdot y(1 - y) \cdot x_i \tag{3.36}$$

Vector form of weight update is shown in Equation (3.37) where w^{new} and w^{old} are the new and old weights, respectively, and $\varphi > 0$ is the number of steps. The dataset is sequentially given to Equation (3.37) until the weights w converge to their optimal.

$$w^{new} = w^{old} - \varphi \cdot (y - t) \cdot y(1 - y) \cdot x \tag{3.37}$$

Figure 3.16 shows a simple neural network with one hidden layer. The output of each neuron in the output and hidden layer is calculated as single neuron example in Figure 3.15.

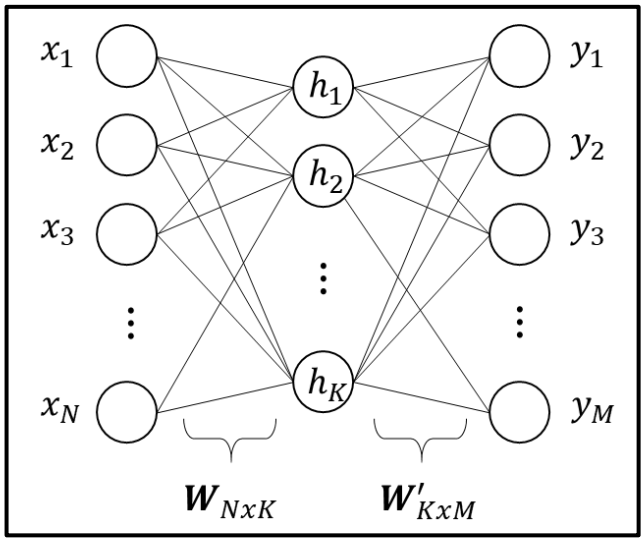

Figure 3.16: An example of neural network with a hidden layer. Inputs are $x_1, x_2, .., x_N$, outputs are $y_1, y_2, .., y_M$ and hidden layers are defined as $h_1, h_2, .., h_K$. Weight matrices are defined as W_{NxK} and W'_{KXM} respectively.

The output of a neuron in the hidden layer h_k and in the output layer y_m are calculated by Equation (3.38) where w_{ni} is $(n, k)^{th}$ element in W_{NxK}, w_{km} is $(k, m)^{th}$ element in W_{KxM}, x_n is n^{th} the input and h_k is the output of hidden layer.

$$h_k = f\left(\sum_{n=1}^{N} x_n w_{nk}\right),\ y_m = f\left(\sum_{k=1}^{K} h_k w'_{km}\right) \tag{3.38}$$

Equation (3.39) denotes the objective or loss function between target t_m and predicted output y_m in the neural network. It resembles the single neuron's loss function as in Equation (3.35).

$$L = \frac{1}{2}\sum_{m=1}^{M}(y_m - t_m)^2 \tag{3.39}$$

Weight update models are defined from the input to hidden layer and from the hidden to output layer for the optimization. The gradient of the loss function L with respect to w_{nk} and w_{km} are calculated. The gradients are computed beginning with the final output error and the error gets propagated throughout the neural network to update the weights. This process is called backpropagation. Chain rule is applied to compute $\partial L/\partial w'_{km}$ as shown in Equation (3.40). The derivative of L with respect to y_m is $\partial L/\partial y_m = y_m - t_m$ and the derivative of y_m with respect to v'_m is $\partial y_m/\partial v'_m = y_m(1 - y_m)$ by Equation (3.34). The derivative of $v'_m = \sum_{k=1}^{K} w'_{km}h_k$ with respect w'_{km} is $\partial v'_m/\partial w'_{km} = h_k$.

$$\frac{\partial L}{\partial w'_{km}} = \frac{\partial L}{\partial y_m}\cdot\frac{\partial y_m}{\partial v'_m}\cdot\frac{\partial v'_m}{\partial w'_{km}} = (y_m - t_m)\cdot y_m(1 - y_m)\cdot h_k \tag{3.40}$$

Equation (3.41) denotes update of the weight w'_{km} with the gradient computed by Equation (3.40).

$$w_{km}^{new} = w_{km}^{old} - \varphi\cdot(y_m - t_m)\cdot y_m(1 - y_m)\cdot h_k \tag{3.41}$$

The gradient of the loss function with respect to w_{nk} (between the hidden and input layer) is defined in Equation (3.42). The gradients are summed since the hidden units are connected to every output unit.

$$\frac{\partial L}{\partial w_{nk}} = \sum_{m=1}^{M}\left(\frac{\partial L}{\partial y_m}\cdot\frac{\partial y_m}{\partial v'_m}\cdot\frac{\partial v'_m}{\partial h_n}\right)\cdot\frac{\partial h_n}{\partial v_k}\cdot\frac{\partial v_k}{\partial w_{nk}} \tag{3.42}$$

Equation (3.43) is obtained by using the equations $\partial v'_m/\partial h_n = w'_{nm}$, $\partial h_n/\partial v_k = h_n(1 - h_n)$ and $\partial v_k/\partial w_{nk} = x_n$.

$$\frac{\partial L}{\partial w_{nk}} = \left(\sum_{m=1}^{M} [(y_m - t_m) \cdot y_m(1 - y_m) \cdot w'_{nm}] \right) \cdot h_n(1 - h_n) \cdot x_n \tag{3.43}$$

Equation (3.44) denotes the update model for the weight w_{nk} with the gradient computed by Equation (3.43).

$$w_{nk}^{new} = w_{nk}^{old} \varphi \left(\sum_{m=1}^{M} [(y_m - t_m) y_m(1 - y_m) w'_{nm}] \right) h_n(1 - h_n) x_n \tag{3.44}$$

As a summary, the neural networks are optimized by the algorithms such as SGD that uses the efficient weight update algorithm called backpropagation.

3.5.4. Proposed Architectures

In this work, single and multi-tasking 1D CNNs are designed to predict concentration of a substance in solutions as shown in Figure 3.17 and Figure 3.18, respectively. 1D CNNs are designed to work on 1D data and uses 1D convolution operation in the convolutional layers. They can be used for extracting local subsequences and can capture local patterns in the convolution window. Since the same transformation is applied to different parts of the input, the same pattern can be captured at different positions. This operation makes the network translation invariant. 1D CNNs are easy to manage and useful in signal analysis over fixed length signals. They work well for audio signal analysis and natural language processing.

1D convolution operation is defined in Equation (3.45) where input vector is f with length N, kernel is g with length M and the convolution of f and g is $f * g$. The kernel is slid over the input vector and each of element in the kernel is multiplied with the overlapping values in the input vector. The resulting sum of products are the output vector at the point in the input vector where the kernel is centered. An example operation is shown in Figure 3.19.

$$(f * g)(x) = \sum_{y=1}^{M} g(y).f(x - y + M/2) \tag{3.45}$$

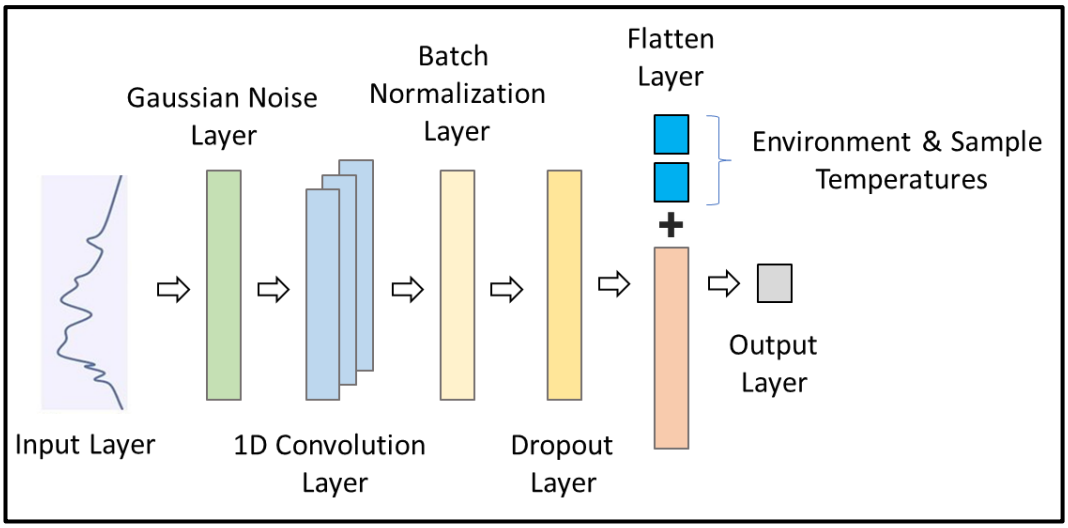

Figure 3.17: Single-tasking 1D CNN with one convolutional layer.

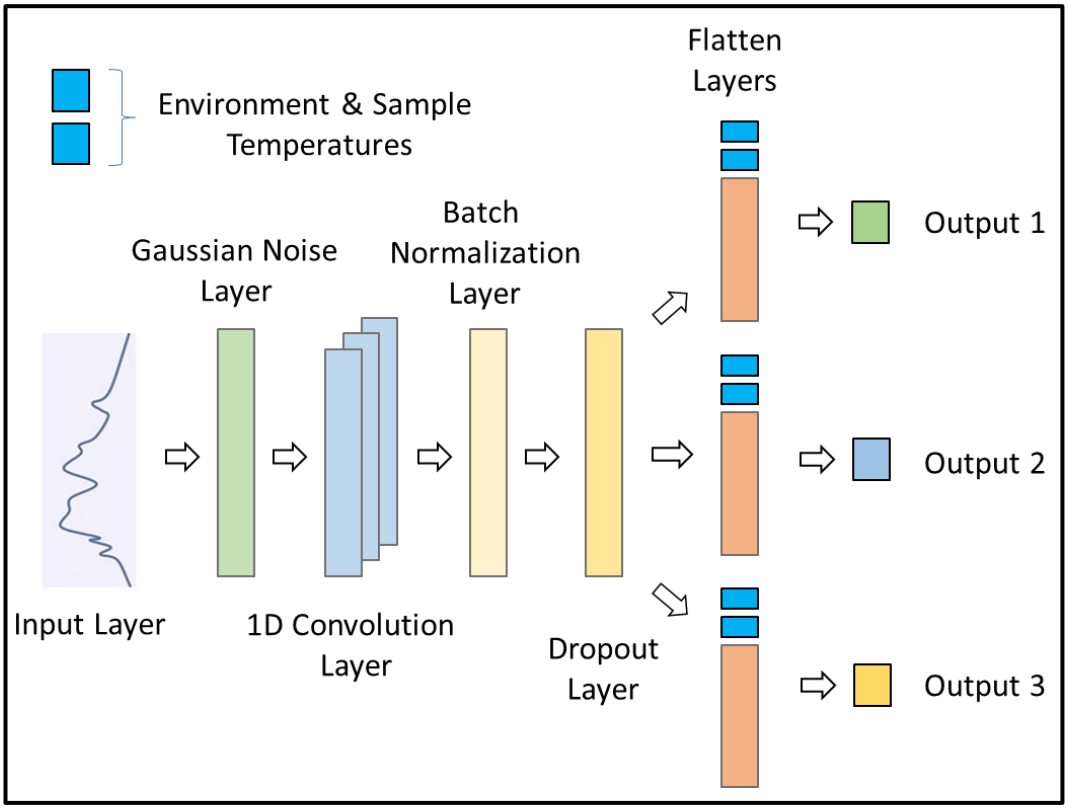

Figure 3.18: Multi-tasking 1D CNN with one convolutional layer.

Dimensions of the 1D convolutional layers are calculated by Equation (3.46) where the size of input is $W_1 \; x \; D_1$, the receptive field size or length of the kernel is R, the number of filters is K, the amount of zero-padding is Z and the stride is S.

$$W_2 = \frac{(W_1 - R) + 2Z}{S} + 1, \qquad D_2 = K \tag{3.46}$$

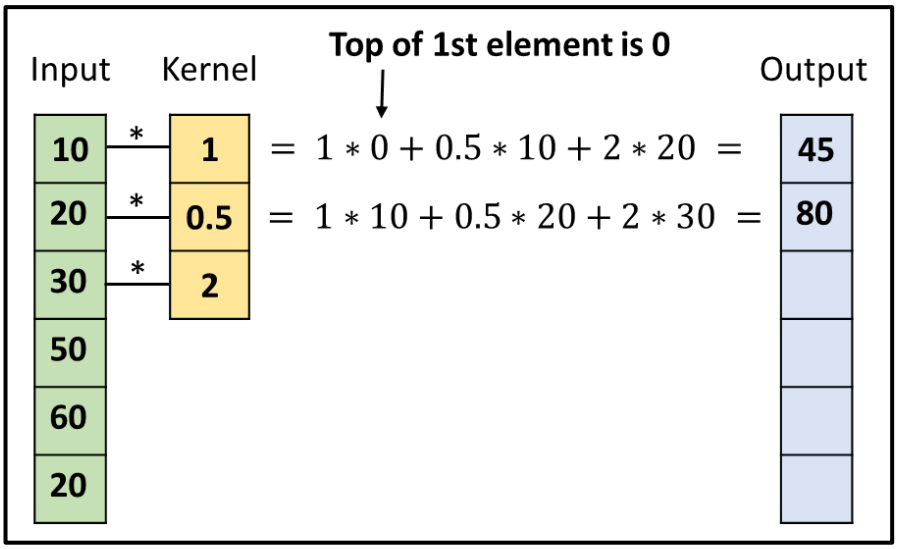

Figure 3.19: An example of 1D convolution applied on a 1D vector.

For the concentration predicting problem defined in this thesis, a traditional single-tasking 1D CNN was designed as shown in Figure 3.17. Single-tasking means that the network has one output and loss function for all types of input data. MSE loss function was employed and the network was trained by batches with size of N that were composed of absorbance spectrums that were selected randomly from training set.

Proposed multi-tasking network is shown in Figure 3.18. Multi-tasking learning (MTL) is inspired by human learning as humans use knowledge of related tasks to learn new tasks [42]. MTL has been very popular for various applications of machine learning such as natural language processing [43] and drug discovery [44]. MTL is commonly employed with either hard or soft parameter sharing of hidden layers as shown in Figure 3.20 and Figure 3.21, respectively [42].

Hard parameter sharing is the most frequently used technique. Hidden layers are shared between all tasks and several task-specific output layers are used as shown in Figure 3.20. Baxter [45] proved that the more tasks are used, the more the corresponding model has to find a representation that detects all of the tasks. Thus, it considerably reduces the chance of overfitting on original task.

Soft parameter sharing method uses different models associated for each task as shown in Figure 3.21. Each task owns a separate model with its own parameters and the distance between the model parameters is regularized to make the parameters similar to each other.

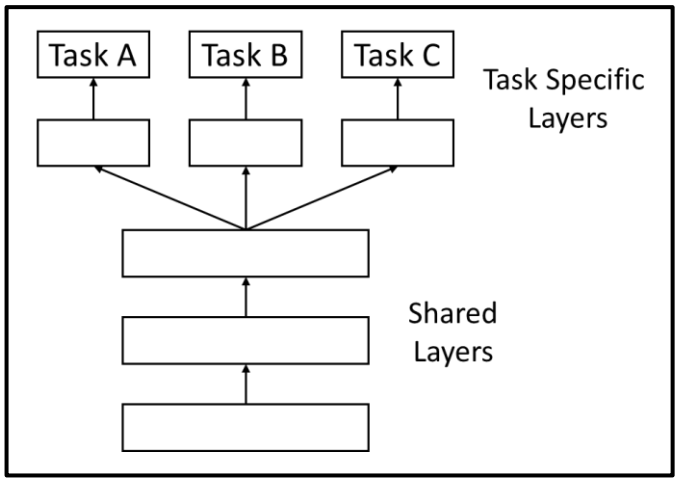

Figure 3.20: Hard parameter sharing for multi-task learning in deep neural networks.

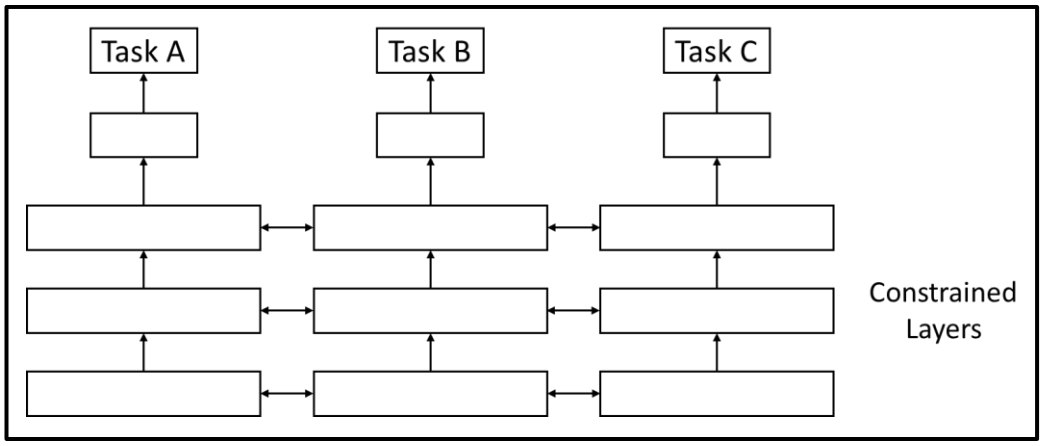

Figure 3.21: Soft parameter sharing for multi-task learning in deep neural networks.

In this work, as shown in Figure 3.18, hard parameter sharing is used because it is easy to construct and risk of overfitting is greatly reduced. There are three types of absorbance spectrums defined in Section 3.1. Each type was considered as a task and three outputs or tasks were added to the network. The output of batch normalization was flattened and the resulting one-hot vector was fully connected to the outputs separately. The MSE loss function was used for all outputs. In training, only one task was optimized and weights of other two tasks were frozen.

The network was trained with batches of size N. The batches were created in two ways. In the first way, the batches composed of the same type of spectrums. Specifically, at each iteration $i \in \{1, .., N\}$, the network was trained with the batch B_i^K where $K \in \{C1, C2, C3\}$ was task identifier. At each epoch, all batches were fed to the network in the order $\{B_1^{C1}, B_2^{C2}, B_3^{C3}, B_4^{C1}, \dots, B_N^K\}$ to help the network learn spectrum

types better. In the second way, the batches were created randomly such that each batch contained all types of spectrums not the same.

In both proposed 1D CNNs, Gaussian noise layer was added right after the input layer to prevent overfitting as Bjerrum [2] did. Then, 1D convolution layer was added with ReLU [5] activation function, L1-L2 regularization and Glorot [16] initialization. Output of the 1D convolutional layer was given to the batch normalization layer followed by a dropout layer. The output of the dropout layer was flattened to one-hot vector. Environment and sample temperatures were appended to end of the one-hot vector. The resulting vector was fully connected to output layer with linear activation. The MSE loss function defined in Equation (3.31) was employed with Adam [41] optimization.

Adam optimizer is an extension of SGD algorithm and combines advantages of both adaptive gradient algorithm (AdaGrad) and root mean square propagation (RMSProp) algorithms [41]. AdaGrad algorithm uses a learning rate for each parameter and improves performance of the problems having sparse gradients. Similarly, RMSProp uses learning rates for each parameter and the learning rates are updated by the average of recent magnitudes of the gradients. Adam optimizer adapts the per-parameter learning rates by using the average first moment as in RMSProp and uses the second moments of the gradients. In other words, an exponential moving average of the gradient and the squared gradient are calculated. The decay rates of the moving averages are adjusted by the parameters β_1 and β_2. The initial value of these parameters is recommended to be very close to 1.0.

3.6. Long Short-Term Memory

LSTM units or blocks are a type of recurrent neural network (RNN) that contains nodes connected along a sequence in a form of a directed graph as shown in Figure 3.22 [46]. This structure provides dynamic temporal behavior. RNNs process sequence of input values by using an internal memory or state. Thus, they are effective for applications, such as handwriting and speech recognition, that decisions are affected by previous information or states.

The differences between MLPs and RNNs are that MLPs only map from input to output and they do not have cyclical connections. However, RNNs map from all previous inputs to outputs and they have cyclical connections.

LSTM architectures contain subnets that are recurrently connected. These subnets are called memory blocks. The blocks consist of single or multiple memory cells that are self-connected. As shown in Figure 3.23, the blocks have three multiplicative units which are input, output and forget gates. LSTM networks have memory blocks in the hidden layer instead of summation units that classical RNNs have.

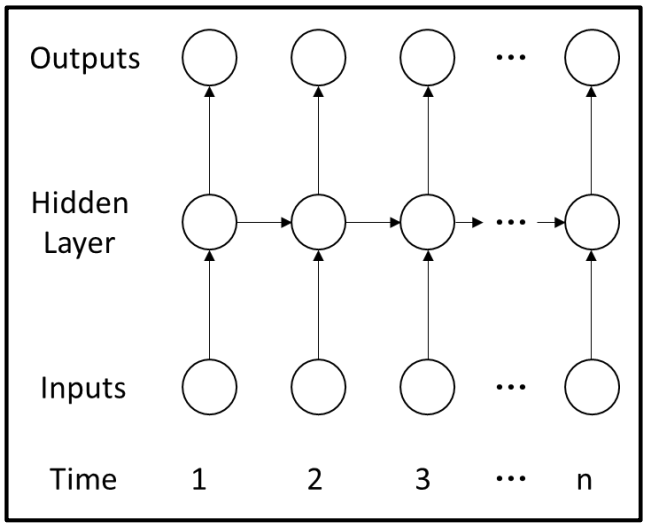

Figure 3.22: Example of RNN for time series with length n. Each node represents a time t and the information at time t flows to the node at time $t + 1$.

LSTM memory cells store and access historical information over long periods of time by the input, output and forget gates. In this way, vanishing gradient problem is reduced because the activation of the cell is not overwritten by the new inputs if the activation of input gate is close to 0. Thus, the cell activation may flow to the end of the sequence by the output gate.

Several architectures of LSTM units are present and a common one contains input, output and forget gate as shown in Figure 3.23. LSTM networks are differentiable and can be trained with backpropagation without gradients vanished. The input gate controls how the new values are transferred to the cell, the forget gate decides which values are stored in the cell and output gate determines which value in the cell is used to calculate the output of the LSTM. The gates have their own biases and weights that need to be learned during training.

As shown in Figure 3.23, the gates compute an activation of a weighted sum. At time step t, activations of the input, output and forget gates are i_t, o_t and f_t respectively. The circular shapes with $\times$ denote element-wise multiplication and the

circular shapes with S defines a differentiable function such as sigmoid to a weighted sum. The dashed arrows from the memory cell c to the gates defines the activation of the cell at time step $t-1$.

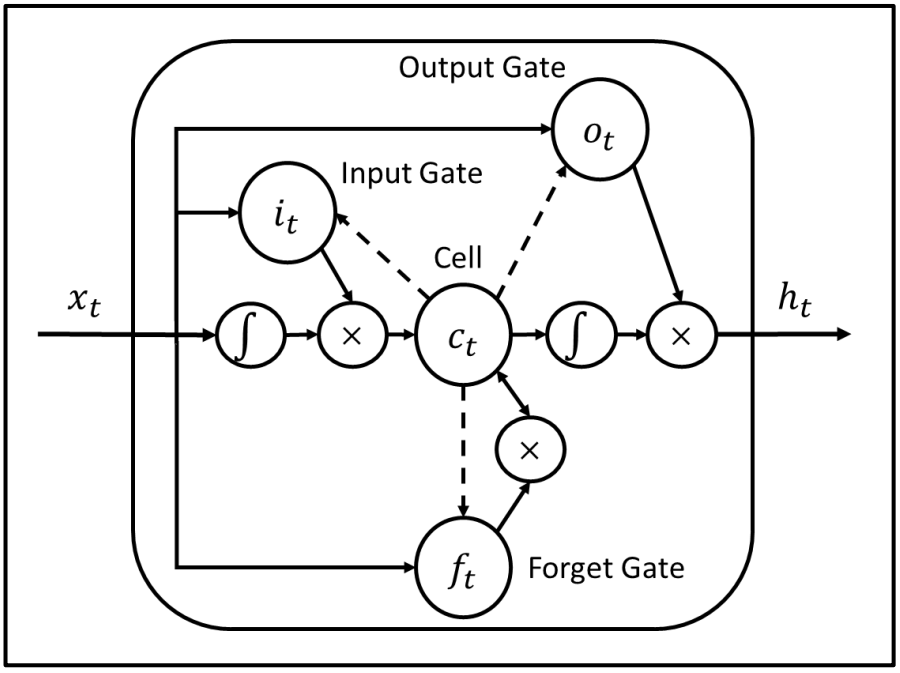

Figure 3.23: A common architecture of LSTM units with input i, output o and forget f gates. The input vector is x_t, output vector is h_t and memory cell is c_t at time step t.

The forget gate f_i^t for time step t and cell i is defined in Equation (3.47) where x_t is the input vector at time t, h_t is the hidden layer vector at time t, b^f is biases, U^f is the input weights and W^f is the recurrent weights of the gate [46]. Sigmoid activation function σ is employed to get a value between 0 and 1.

$$f_i^t = \sigma\left(b_i^f + \sum_j W_{i,j}^f h_j^{t-1} + \sum_j U_{i,j}^f x_j^t\right) \tag{3.47}$$

The state of the cell is determined by the Equation (3.48) where U, b and W are the recurrent weights, biases and input weights, respectively.

$$s_i^t = f_i^t s_i^{t-1} + g_i^t \sigma\left(b_i + \sum_j W_{i,j} h_j^{t-1} + \sum_j U_{i,j} x_j^t\right) \tag{3.48}$$

As shown in Equation (3.49), the input gate is calculated with its own parameters (b^g, U^g and W^g) like the forget gate.

$$g_i^t = \sigma\left(b_i^g + \sum_j W_{i,j}^g h_j^{t-1} + \sum_j U_{i,j}^g x_j^t\right) \tag{3.49}$$

The output of the LSTM unit h_i^t is controlled by the output gate q_i^t with sigmoid activation and its own parameters (b^o, U^o and W^o) as shown in Equation (3.50).

$$q_i^t = \sigma\left(b_i^o + \sum_j W_{i,j}^o h_j^{t-1} + \sum_j U_{i,j}^o x_j^t\right) \tag{3.50}$$

As shown in Equation (3.51), the output of the LSTM unit is calculated via hyperbolic tangent activation function and the output gate q_i^t.

$$h_i^t = tanh(s_i^t)\, q_i^t \tag{3.51}$$

Learnable parameter count of LSTM units is computed by $4(nm + n^2)$ where size of W is $n * n$ and size of U is $n * m$ and there are 4 neural network layers input, output, forget gate and memory cell. If biases are added, then parameter count is $4((n + 1)m + n^2)$.

Parameter fine tuning is not required for LSTM networks since the networks run well with parameters such as learning rate and biases. If large learning rate is used, then the output gates get close to 0 and the learning procedure is not affected badly. Update complexity of LSTM algorithms is $O(N)$ where N is the number weights. LSTMs can cope with noises, continuous values and separated representations in the input [47].

4. EXPERIMENTS AND RESULTS

In this section, the proposed regression models explained in Section 3 were tested for different parameters and compared by using MAE and R^2 score. Absorbance spectrums were loaded by using Numpy 1.14 [48] and normalized by StandardScaler algorithm in Scikit-Learn 0.19 [32].

The PLSR, GBR and RF models were constructed by using Scikit-Learn. These three models were trained with 10-fold cross-validation on the preprocessed dataset without outliers. Each of the cross-validation sets was used as a test set and concentration predictions were acquired for all spectrums in entire dataset. Thus, total of 10 models were trained. MAEs and R^2 scores of these models were averaged.

The proposed deep neural networks including CNN and LSTM layers were built by using Keras 2.1.3 [49] with Tensorflow 1.4 [50] backend. The hyperparameters of the deep networks were optimized by using Bayesian optimization framework on a random optimization dataset [7]. The result of the models was visualized by using Matplotlib 2.1.2 [51]. The models were trained and tested by a system with NVIDIA GTX 1070 graphics card with 8GB memory, Ryzen 5 1600X CPU and 16GB RAM.

4.1. Partial Least Squares Regression

The PLSR model was trained by 10-fold cross-validation and NIPALS algorithm [3] in Scikit-Learn [32], with maximum of 100.000 iterations and a tolerance of 10^{-16}. Built-in scaling operation was not used since the dataset was normalized before the training.

The number of principal components was changed from 0 to 50 to find the best model. For each of the number of components, 10 different PLSR models were trained by using 10-fold cross-validation. The MAEs and R^2 scores of these 10 models were averaged. As shown in Table 4.1, Figure 4.1 and Figure 4.2, the best MAE and R^2 score were achieved for 41 components. The highest R^2 score was 0.95 for 32 components and the lowest MAE was 3.87 for 41 components.

Table 4.1: The average MAE and R^2 scores acquired for 15, 32 and 41 components.

Number of Components	R^2 Score	MAE
15	0.94	4.45
32	0.95	3.97
41	0.94	3.87

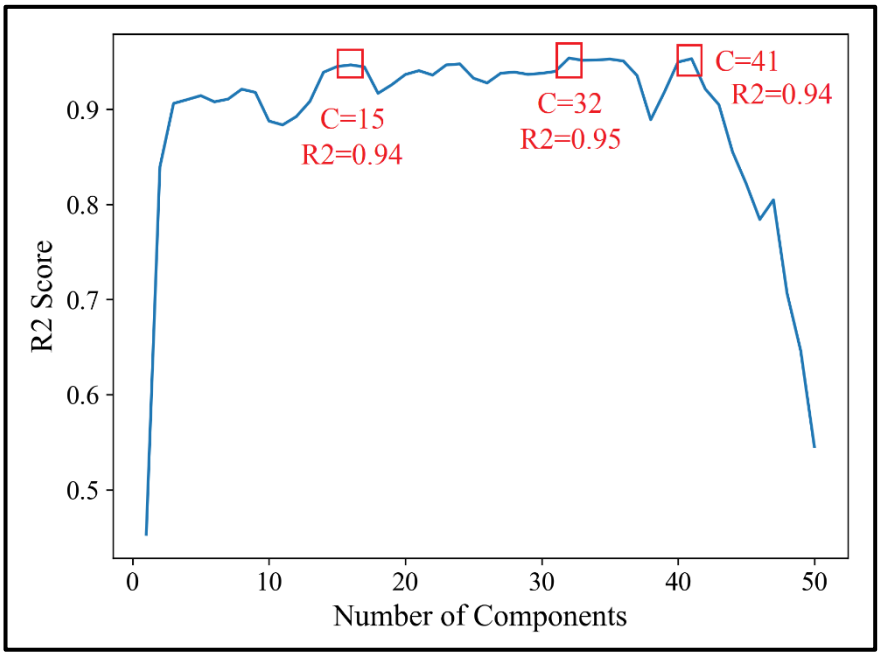

Figure 4.1: Average R^2 score of the models trained by 10-fold cross-validation. The best R^2 scores were achieved for 15, 32 and 41 components.

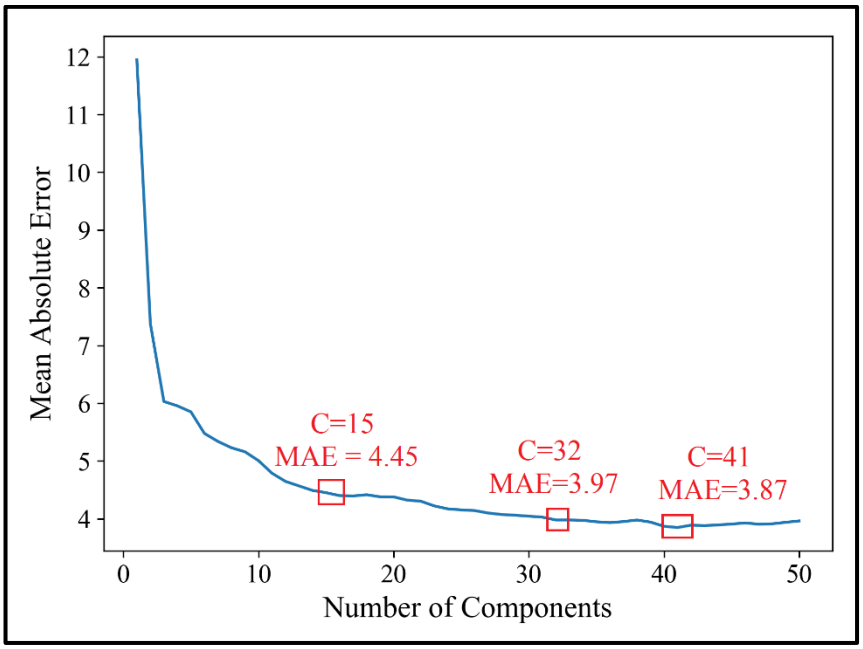

Figure 4.2: Average MAE of the models trained by 10-fold cross-validation. The lowest MAE was achieved for 41 components.

In Figure 4.3 and Figure 4.4, true and predicted concentrations for the models trained by 32 and 41 components are shown on the $x = y$ red line. When predicted values get close to the red line, R^2 score increases. Thus, the model trained for 32 components is slightly better than that of the model trained for 41 components.

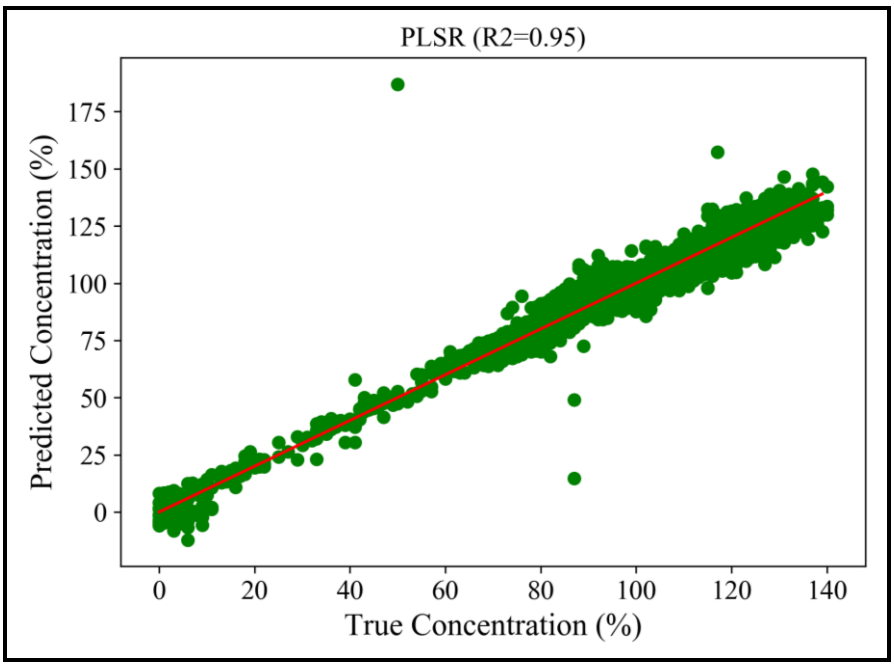

Figure 4.3: True and predicted concentrations for the model trained for 32 components.

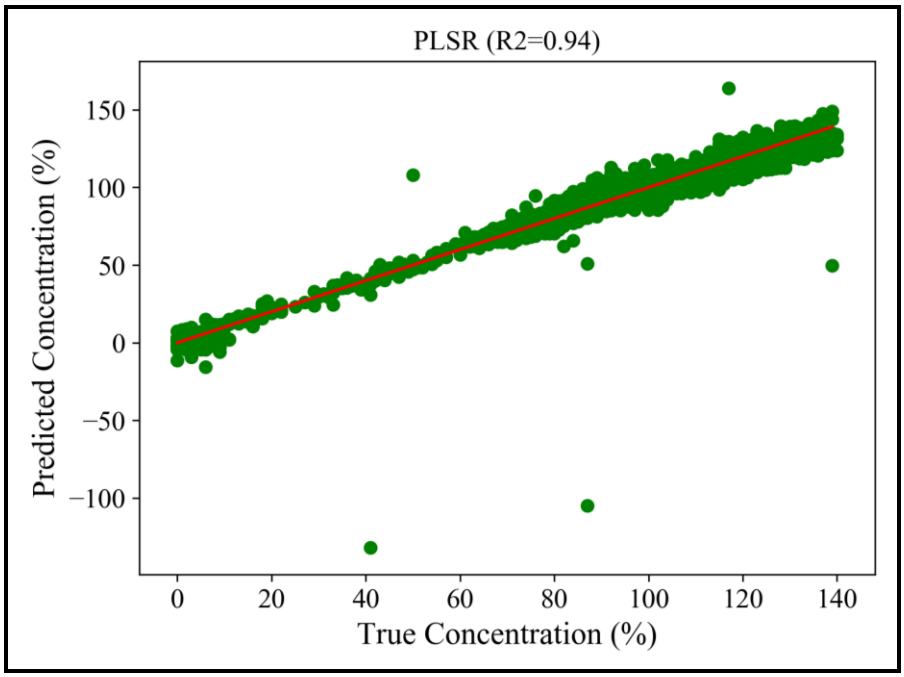

Figure 4.4: True and predicted concentrations for the model trained for 41 components.

4.2. Random Forest Regression

The RF regression model in Scikit-Learn [32] was trained with 10-fold cross-validation on the preprocessed dataset without outliers. Total of 10 models were trained and performance metrics were averaged as in the training of PLSR models.

Training parameters are shown in Table 4.2. Maximum features refer the number of features to use to find the best split. It was set to 164 which was the length of spectrum vectors. Quality of split criterion defines the function that measures split quality. It was set to MSE for regression trees. The maximum depth of the tree was set to unlimited and this means that nodes are expanded until all leaves contain less than two samples. Minimum samples split is the minimum number of samples needed to split a node and minimum samples leaf is the minimum number of samples needed to create a leaf node.

Table 4.2: Training parameters of the best RF model.

Parameter Name	Parameter Value
Number of Base Learners	51
Maximum Features	164
Quality of Split Criterion	MSE
Maximum Depth	Unlimited
Minimum Samples Split	2
Minimum Samples Leaf	1

Firstly, the number of base learners was tuned for the parameters in Table 4.2. It was changed from 0 to 60. The highest average R^2 score and the lowest average MAE for 10 cross-validation sets were achieved for 51 trees as shown in Table 4.3, Figure 4.5 and Figure 4.6.

Table 4.3: The MAE and R^2 score of the models trained for different number of estimators.

Number of Estimators	R^2 Score	MAE
1	0.93	3.89
10	0.96	2.74
30	0.97	2.61
40	0.97	2.60
51	0.97	2.59

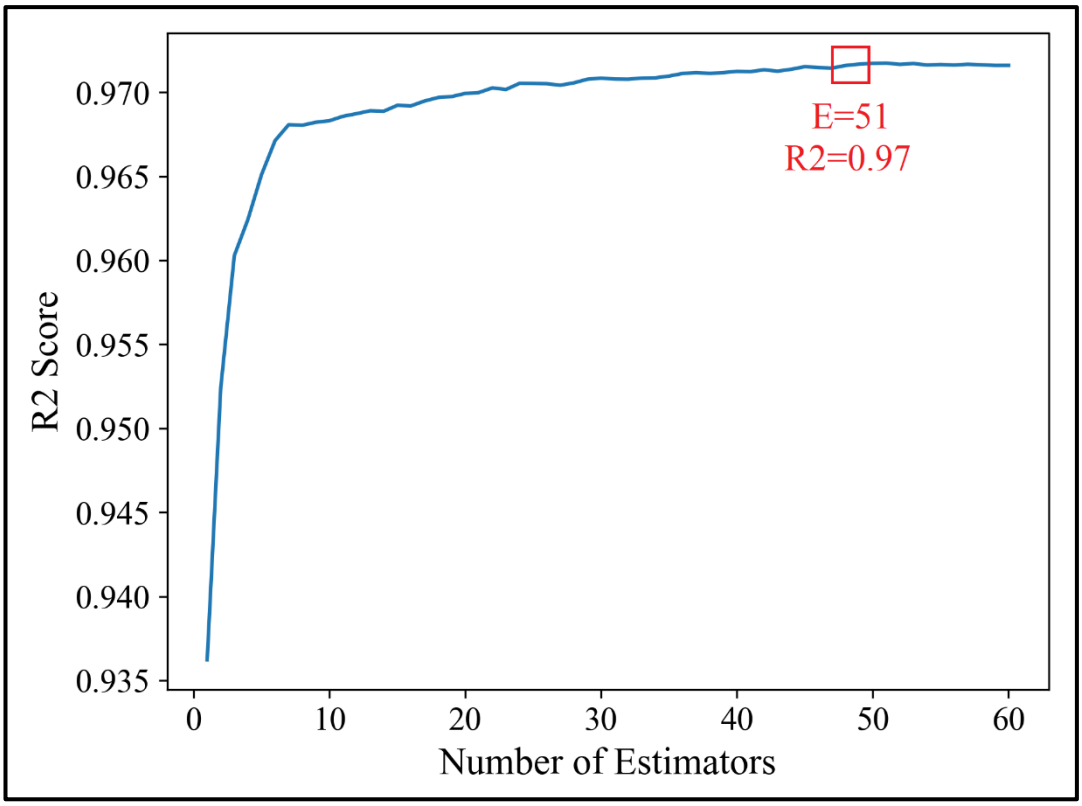

Figure 4.5: Average R^2 scores of the models trained for different number of decisions trees. The best R^2 score (0.97) was achieved for 51 decision trees.

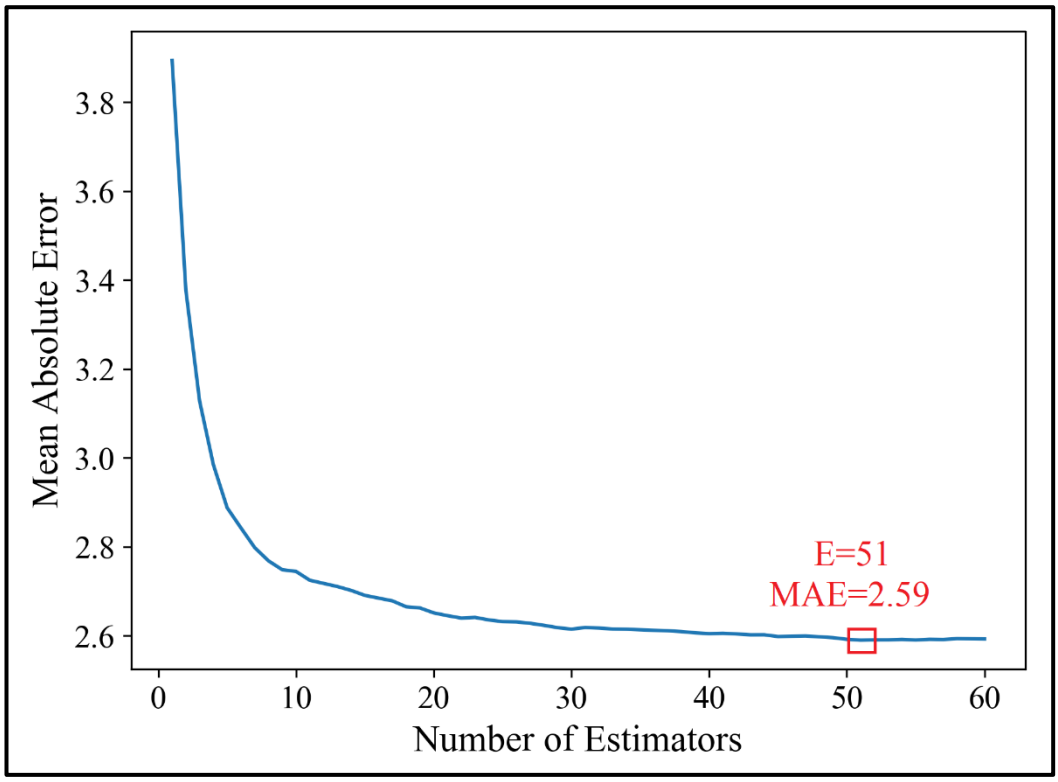

Figure 4.6: Average MAEs of the models trained for different number of decisions trees. The lowest MAE (2.59) was achieved for 51 decision trees.

After tuning the number of base learners, by using the parameters in Table 4.2, the minimum samples split was tested for 2, 10 and 50 as shown in Table 4.4. The best model was acquired by minimum 2 samples split. Predicted values for 10 and 50 are shown in Figure 4.7 and Figure 4.8, respectively. As a result, increasing the minimum samples split caused underfitting meaning that the model could not capture underlying trend of the data.

Table 4.4: The MAE and R^2 score of the models trained for different number of minimum samples split.

Minimum Samples Split	R^2 Score	MAE
2	0.97	2.59
10	0.97	2.64
50	0.96	3.31

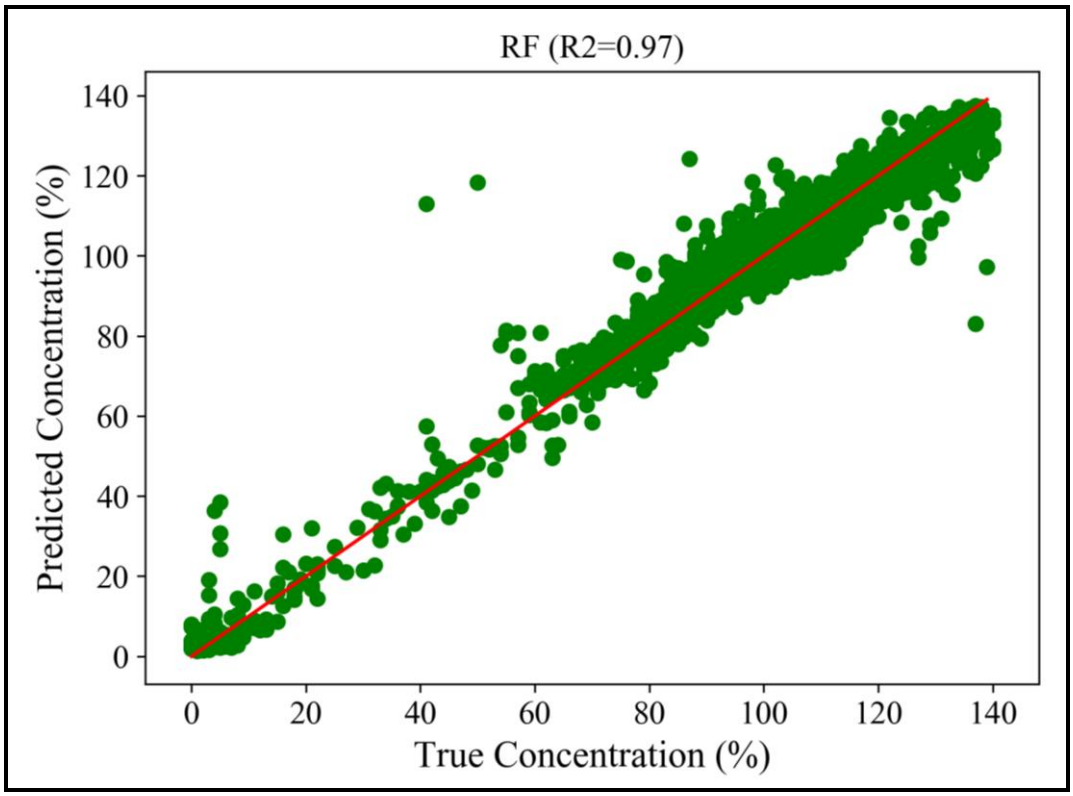

Figure 4.7: True and predicted concentrations for the model trained for minimum 10 samples splits.

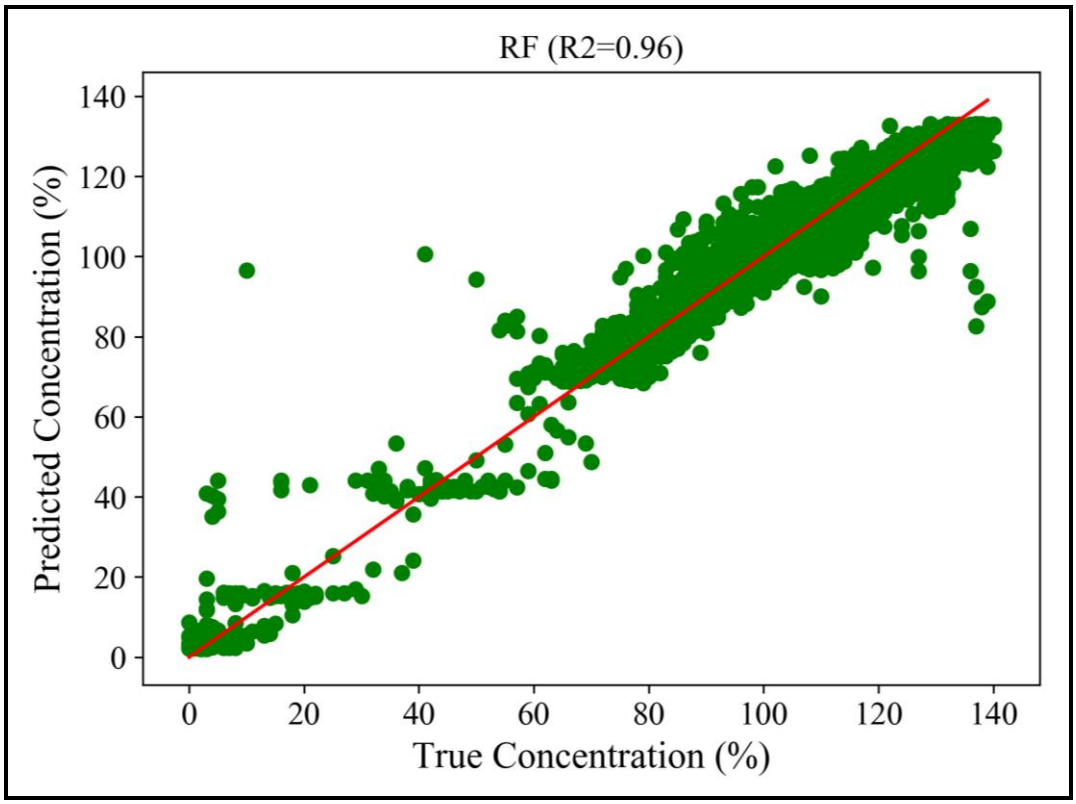

Figure 4.8: True and predicted concentrations for the model trained for maximum 50 splits.

Using the parameters in Table 4.2, the minimum samples leaf was tested for 1, 10 and 20 as shown in Table 4.5. The best model was acquired by minimum 1 samples split. Predicted values for 10 and 20 are shown in Figure 4.9 and Figure 4.10, respectively. As a result, it was observed that increasing the minimum samples leaf caused underfitting. As shown in these figures, the model failed to learn the linear relation between concentrations and spectrums. The predicted points moved away from the red line.

Table 4.5: The MAE and R^2 score of the models trained for different number of minimum samples leaf.

Minimum Samples Leaf	R^2 Score	MAE
1	0.97	2.59
10	0.96	3.05
20	0.95	3.38

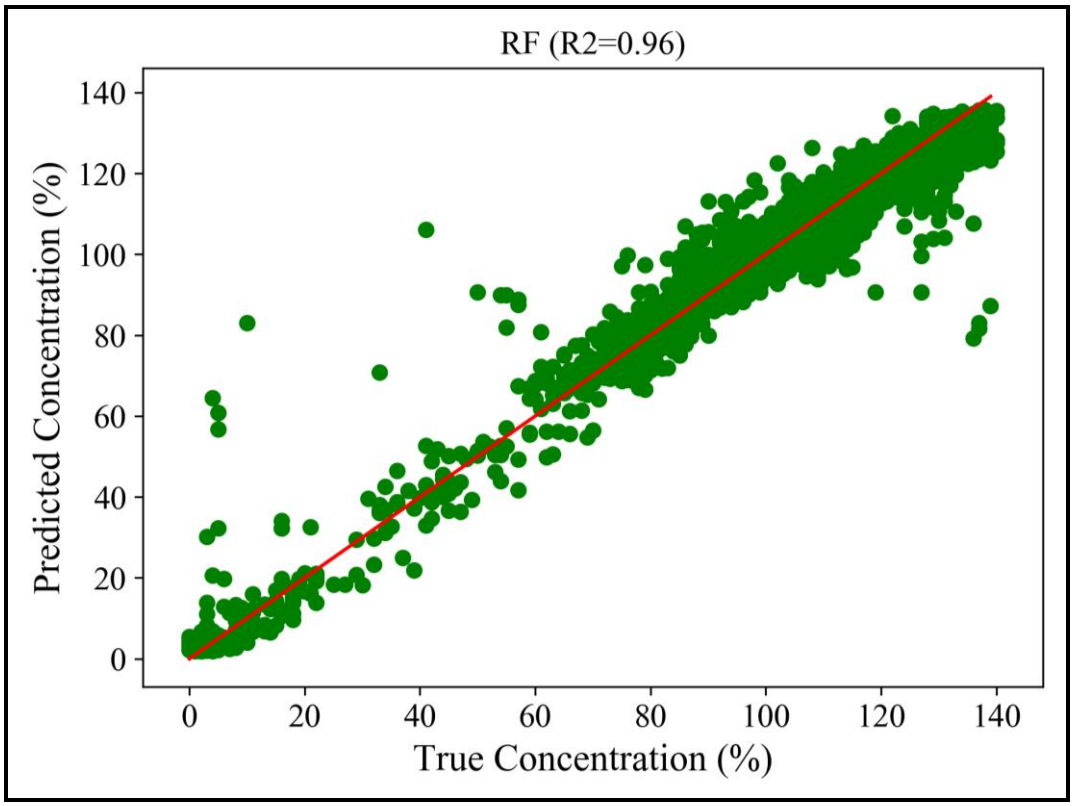

Figure 4.9: True and predicted concentrations for the model trained for minimum 10 samples leaf.

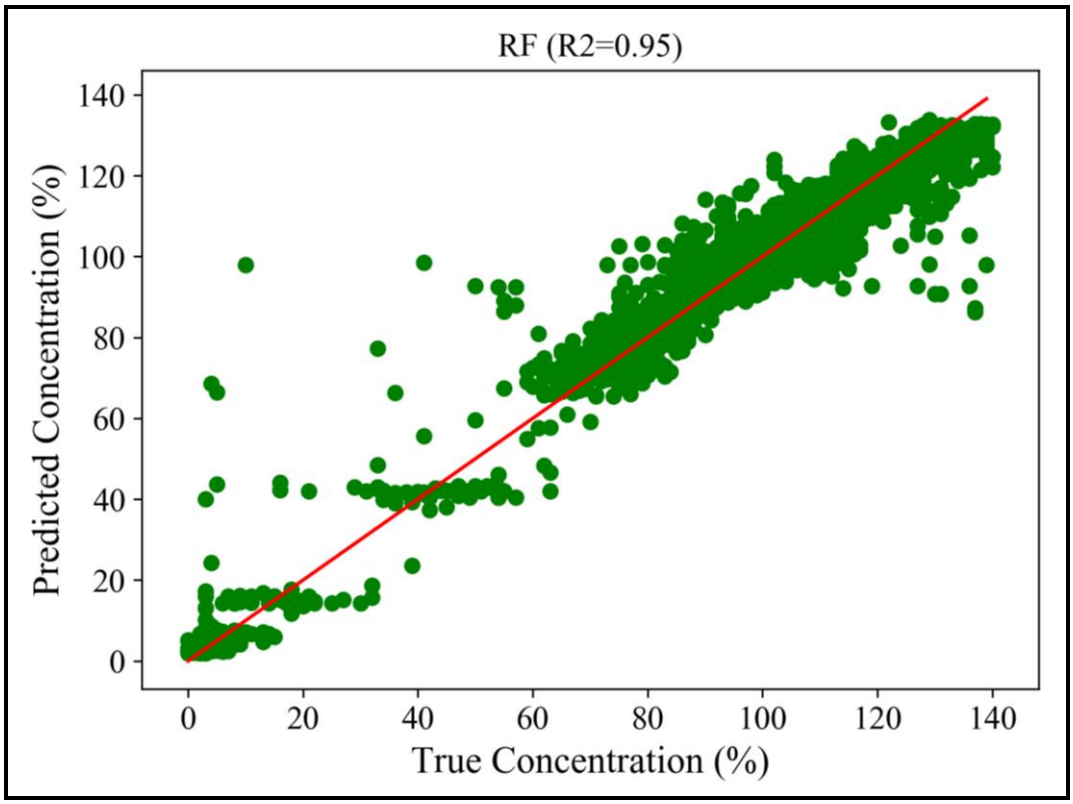

Figure 4.10: True and predicted concentrations for the model trained for minimum 20 samples leaf.

The best model with the highest R^2 (0.97) and the lowest MAE (2.59) was acquired with the parameters in Table 4.2 by tuning minimum samples split, minimum samples leaf and number of base learners. The maximum depth of the base learners was between 20 and 33. Increasing the minimum samples leaf and split caused underfitting. After number of 51 trees, the model showed overfitting behaviors. True and predicted concentrations for the best model are shown in Figure 4.11. It seems that the model managed the learn linear relation between concentrations and spectrums.

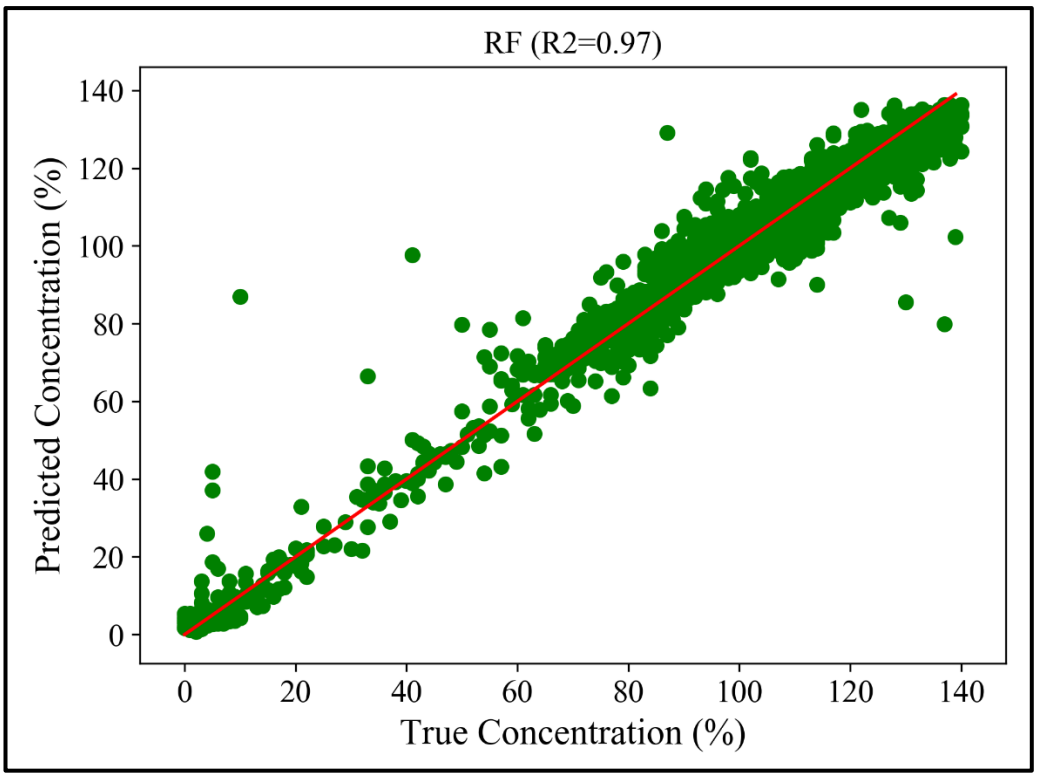

Figure 4.11: True and predicted concentrations for the best model.

4.3. Gradient Boosting Regression

The GBR model in Scikit-Learn [32] was employed with 10-fold cross-validation on the preprocessed dataset cleaned from outliers. Total of 10 models were trained and performance scores were averaged as in the training of PLSR and RF models.

The model parameters are shown in Table 4.6. Learning rate limits the contribution of each tree, the number of estimators is the number of boosting stages to do, maximum depth limits the number of nodes in the regression trees, criterion function measures the quality of a split, loss function specifies the function to be

optimized and max features is the number of features to use for the best split in trees. For the regression problem, criterion and loss function were set to MSE and least squares, respectively. Max features was set to length of spectrum vectors.

In order to find the best model, loss function, learning rate, number of estimators and maximum depth parameters were tested for various configurations. For each parameter configuration, the model trained by 10-fold cross-validation on the entire dataset, and then MAE and R^2 metrics of the 10 models were averaged. As a result, the best MAE and R^2 score were acquired by the parameters in Table 4.6.

Table 4.6: The parameters of the best GBR model.

Parameter Name	Parameter Value
Learning Rate	0.1
Loss Function	Least Squares
Number of Estimators	100-1000
Maximum Depth	9
Criterion	MSE
Max Features	164

Firstly, number of estimators was changed from 100 to 2000 with maximum depth 3 and rest of the parameters in Table 4.6. As shown in Table 4.7, the best average R^2 score and MAE were achieved by 2000 estimators, but training time increased significantly after 1000 estimators.

Table 4.7: Average R^2 score and MAE of the model trained by using various number of estimators, learning rate 0.1 and maximum depth 3, with 10-fold cross-validation.

Number of Estimators	R^2 Score	MAE
100	0.96	3.5
500	0.97	2.76
1000	0.97	2.65
2000	0.97	2.61

As shown in Table 4.8, maximum depth was changed from 3 to 15 for fixed learning rate 0.1 and 100 estimators. The best model was achieved by maximum depth 9 and increasing it much further caused overfitting and long training time. True and predicted concentrations for the best model are shown in Figure 4.12.

Table 4.8: Average MAE and R^2 score of the model trained by using different maximum depths, learning rate 0.1 and estimator count 100, with 10-fold cross-validation.

Maximum Depth	R^2 Score	MAE
3	0.96	3.45
5	0.97	2.84
7	0.97	2.55
9	0.97	2.47
11	0.97	2.51
13	0.96	2.64
15	0.96	2.79

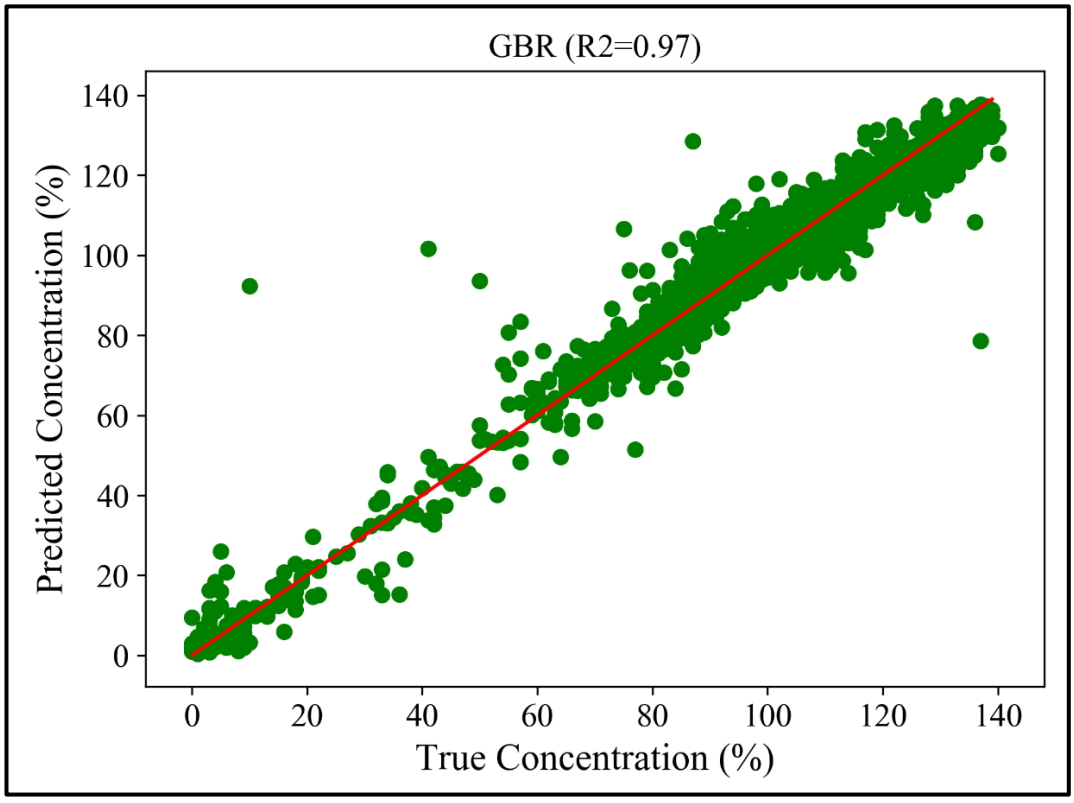

Figure 4.12: True and predicted concentrations of the model trained by using the parameters in Table 4.6, with 10-fold cross-validation.

As shown in Table 4.9, Table 4.10 and Table 4.11, while the number of estimators was fixed at 100, maximum depth was changed for learning rates 0.05, 0.2 and 0.6. As a result, decreasing and increasing the default learning rate (0.1) caused degraded performance for almost all maximum depths.

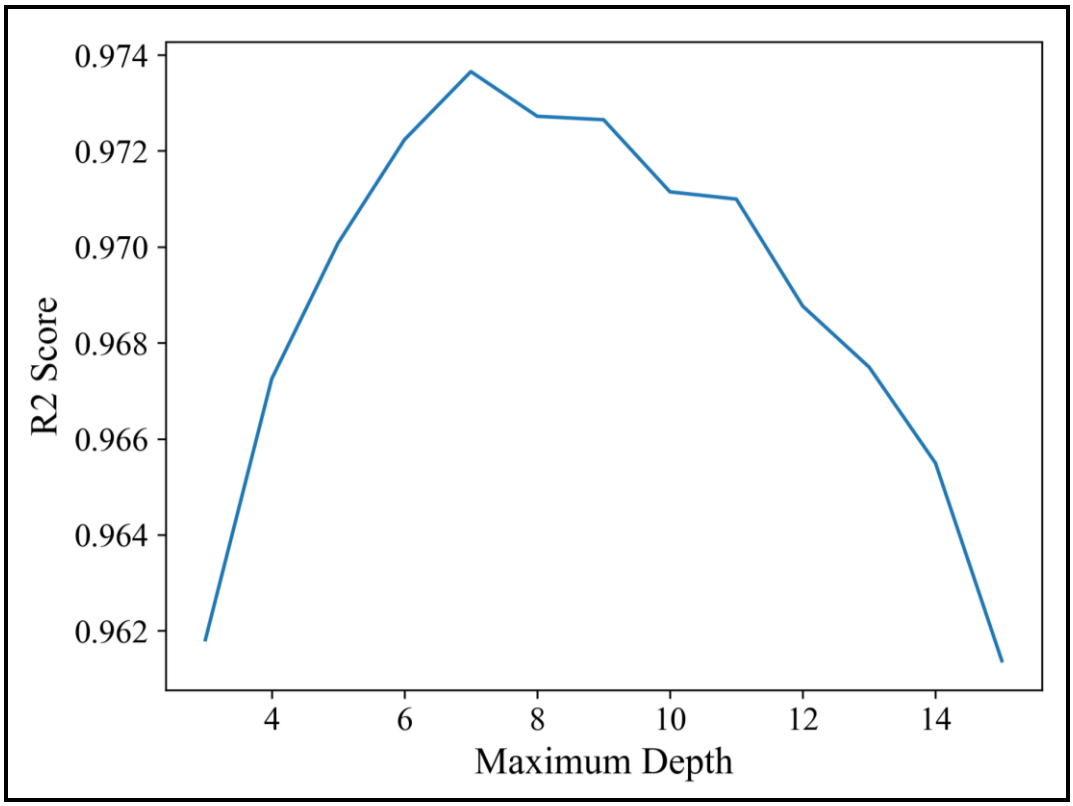

Figure 4.13: Average R^2 scores of the models trained by using the parameters in Table 4.6, with 10-fold cross-validation.

As shown in Figure 4.13, R^2 score increases smoothly until maximum depth 9 and after that depth, the model shows overfitting behaviors. The best model was achieved for maximum depth 9 as shown in Table 4.9.

Table 4.9: Average MAE and R^2 score of the GBR model trained by using various maximum depths, learning rate 0.05 and estimator count is 100.

Maximum Depth	R^2 Score	MAE
3	0.94	4.15
5	0.96	3.17
7	0.97	2.72
9	0.97	2.53
11	0.97	2.56
13	0.96	2.67
15	0.96	2.84

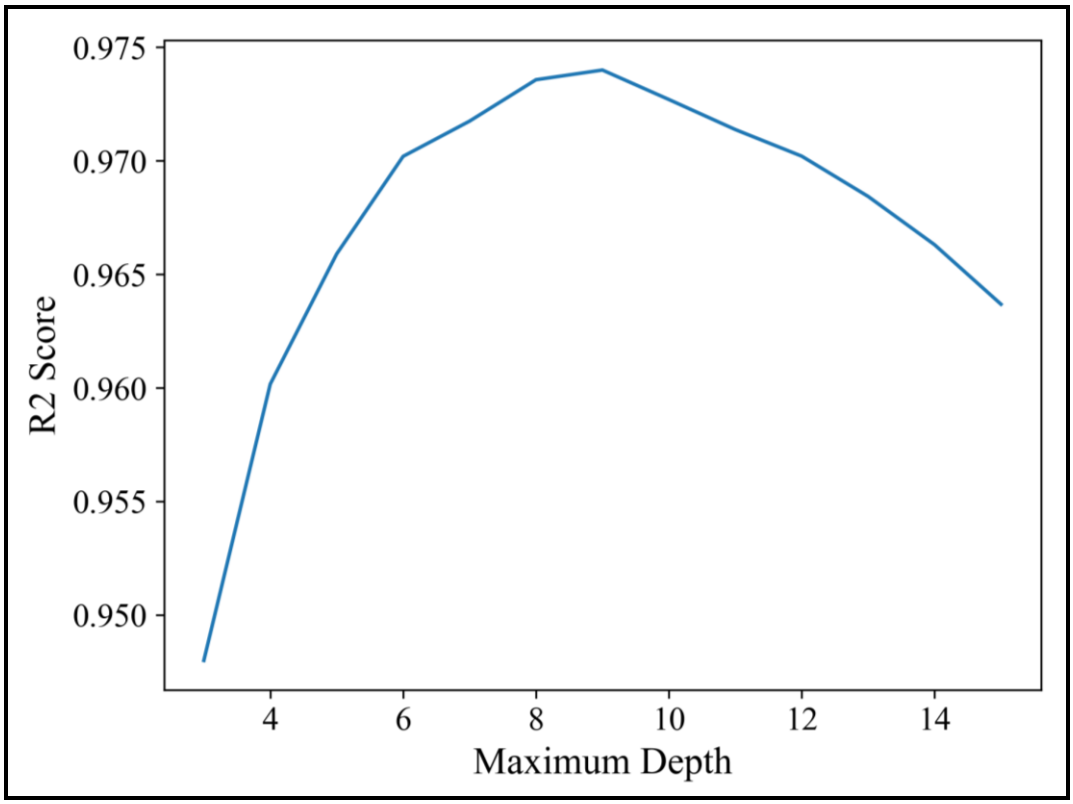

Figure 4.14: Average R^2 scores of the models trained for learning rate 0.05 and various maximum depths, with 10-fold cross-validation.

Increasing the learning rate to 0.2 provided slightly worse MAE and R^2 score for the maximum depth 9 as shown in Table 4.10.

Table 4.10: Average MAE and R^2 score of the GBR model trained by using different maximum depths, learning rate 0.2 and estimator count is 100, with 10-fold cross-validation.

Maximum Depth	R^2 Score	MAE
3	0.94	3.14
5	0.96	2.72
7	0.97	2.59
9	0.97	2.55
11	0.96	2.58
13	0.96	2.69
15	0.96	2.83

As expected, for the learning rate 0.6, the estimators were corrected more aggressively and overfit the data at each step. Thus, as shown in Table 4.11 and Figure 4.15, MAE and R^2 metrics were fluctuated while the maximum depth increased.

Table 4.11: Average MAE and R^2 score of the GBR model trained for different maximum depth values with learning rate 0.6 and estimator count is 100.

Maximum Depth	R^2 Score	MAE
3	0.96	3.30
5	0.95	3.25
7	0.95	3.21
9	0.96	3.08
11	0.95	3.03
12	0.96	2.98
13	0.95	3.02
15	0.95	3.05

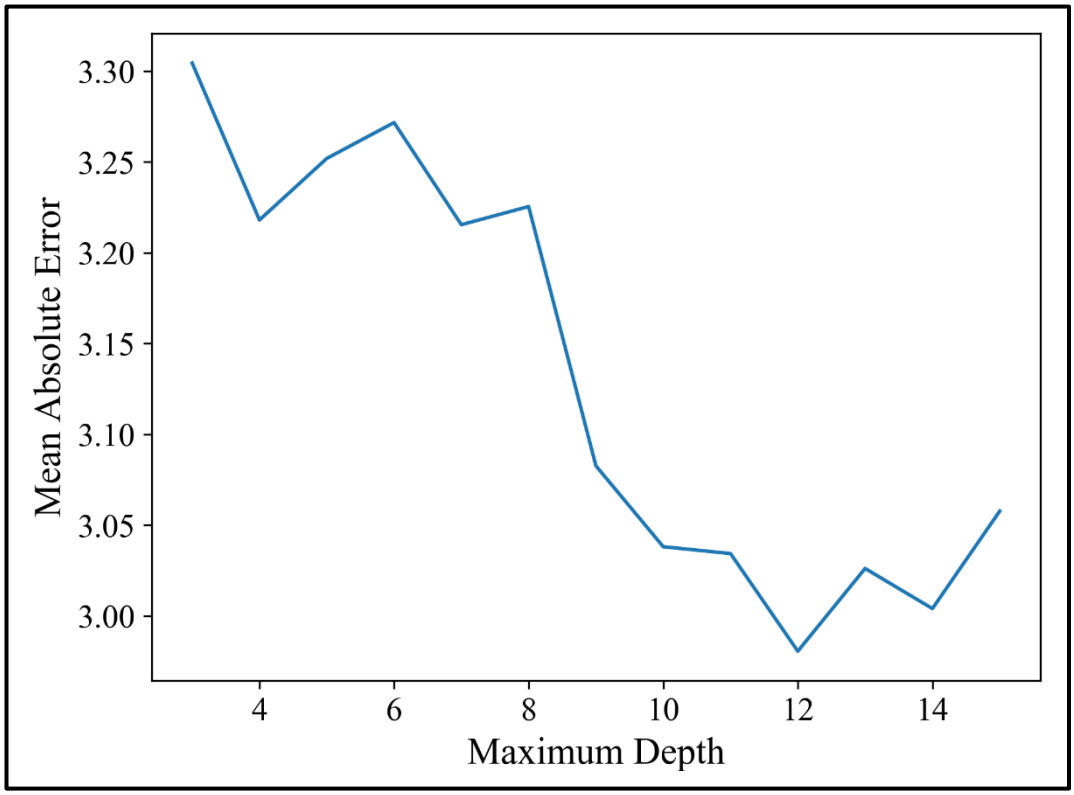

Figure 4.15: Average MAE of the models trained for various maximum depths, learning rate 0.6 and 100 estimators, with 10-fold cross-validation.

As shown in Table 4.12, the number of estimators was changed from 100 and 1500 for fixed learning rate 0.1 and maximum depth 9. Increasing estimator count much further provided slightly better results, but training time significantly increased after 1000 estimators.

Table 4.12: Average MAE and R^2 score of the GBR model trained by using different number of estimators, learning rate 0.1 and maximum depth 9.

Number of Estimators	R^2 Score	MAE
100	0.97	2.47
500	0.97	2.42
1000	0.97	2.41
1500	0.97	2.41

Finally, the loss function parameter was changed as shown in Table 4.13, with 100 estimators, learning rate 0.1 and maximum depth 9 as in Table 4.6. The best MAE and R^2 score were 2.47 and 0.97 acquired by the default loss function (least squares), respectively.

Table 4.13: Average MAE and R^2 score of the GBR model trained by using different loss functions, 100 estimators, learning rate 0.1 and maximum depth 9.

Loss Function	R^2 Score	MAE
Least Squares	0.97	2.47
Least Absolute Deviation	0.97	2.81

As a result, the best model was achieved by the parameters in Table 4.6. The best learning rate was found as 0.1, whereas lower and higher values caused underfitting and overfitting, respectively. Number of estimators was tested manually and the best model was achieved by 1000 and more estimators, as shown in Table 4.12. However, training time increased significantly for more than 1000 estimators. Maximum depth was changed between 0 and 15 for learning rates 0.05, 0.1, 0.2 and 0.6. The best model was achieved by the maximum depth 9 in most of the cases. Finally, for the maximum depth 9, learning rate 0.1 and 100 estimators, the best model was achieved by least squares loss function instead of LAD loss. As shown in Table 4.12, the best MAE and R^2 score were 2.41 and 0.97 obtained by 1000 and more estimators, respectively.

4.4. Convolutional Neural Networks

The proposed 1D CNNs in Section 3.5.4 were designed by inspiring from the works of Bjerrum and Acquarelli explained in Section 2.1 and Section 2.2, respectively. The proposed networks were built by using Keras [49] with Tensorflow [50] backend. The CNNs were optimized by the help of Bayesian optimization framework [7]. Various CNN architectures with different number of layers and parameters were devised, trained and tested in Python. The result of the models was visualized by using Matplotlib [51].

To speed up parameter optimization, the proposed CNNs were trained and tested for a small optimization dataset containing 1600 samples that were randomly selected from the preprocessed dataset with 6167 samples. The training and testing sets were created randomly by using 75% and 25% of this optimization dataset, respectively. The networks were trained by using batches with size of 60.

For each parameter configuration, the CNNs were trained and tested 10 times on the optimization dataset that was created randomly each time. The models were evaluated by performance metrics of MAE and R^2 score. Average of these metrics was calculated by using the results of 10 experiments.

Adam [41] optimizer and MSE loss function defined in Equation (3.31) were employed in the proposed CNNs. Weights of the networks were initialized by Glorot uniform initializer [16]. The initializer draws samples from uniform distribution between $[-\alpha, \alpha]$ where $\alpha = \sqrt{6/(w_{in} + w_{out})}$, w_{in} is the number of input units and w_{out} is the number of output units.

4.4.1. Single-Tasking Architecture

The proposed 1D CNN with one convolutional layer and single output is shown in Figure 3.17. In order to find optimal hyperparameters of the network that generalizes and captures linear relationship between concentration and spectrums, Bayesian optimization framework was used to get initial hyperparameters for fine tuning later. For the shallow CNN in Figure 3.17, hyperparameter search space was defined as in Table 4.14. The optimization was initialized with the parameters in Table 4.15, followed by up to 200 iterations of standard Gaussian process optimization with the

expected improvement acquisition function. In each iteration, the optimizer selected a parameter set in Table 4.14 and total of 10 networks were created from these parameters. The networks were trained for 150 epochs on 75% of the optimization dataset and each of the models was tested on 25% of the optimization dataset. Each model was trained and tested on randomly created datasets. At the end of each iteration, MAEs of the models were averaged. The aim of the optimizer was to minimize this average MAE.

Table 4.14: Hyperparameter search space for Bayesian optimizer.

Parameter	Type	Range
Learning Rate	Float	$[10^{-4}, 0.1]$
Dropout Probability	Float	$[10^{-2}, 0.3]$
Number of Filters	Float	$[1, 500]$
Filter Length	Odd Number	$[3, 25]$
Stride	Integer	$[1, 5]$

Table 4.15 shows the initial hyperparameters of the network. The loss function was selected as MAE, Adam optimizer was used with default beta values, L1-L2 regularization was applied to convolutional layer with 10^{-4} multiplier, Gaussian noise was applied to the input layer with standard deviation of 0.01 as in Bjerrum's work, the network was trained for 150 epochs, batch size was 60, number of models in each optimization iteration was 10 and weights were initialized by Glorot uniform.

Table 4.15: Initial parameters of the shallow CNN for Bayesian optimization.

Parameter	Value
Loss Function	MSE
Optimizer	Adam
L1-L2 Regularization	10^{-4}
Gaussian Noise	10^{-2}
Training Epoch	150
Batch Size	60
Number of Models	10
Weight Initializer	Glorot Uniform

The hyperparameters found by the optimizer are shown in Table 4.16. The parameters were fine-tuned by testing various parameter configurations manually. Firstly, number of filters was set to 50 and 200. After that, for these number of filters, filter length was set to 5, 10, 25 and 35. Stride, learning rate, dropout probability and

Gaussian standard deviation were changed for the best model achieved by number of filters and filter length. Additionally, one convolutional layer and one fully connected layer were added. The fully connected layer was tested for 10, 100 and 500 neurons. The two convolutional layers were tested for different number of filters configurations such as 10-10, 50-50 and 100-5. Finally, the network was trained and tested on the entire dataset with 75:25 training and testing ratio, by using the fine-tuned parameters.

Table 4.16: The hyperparameters found by Bayesian optimizer for the shallow CNN.

Parameter	Value
Learning Rate	0. 0413
Dropout Probability	0.02
Number of Filters	100
Filter Length	15
Stride	3

For the different number of filters, the average performance of the network is shown in Table 4.17. For 50 filters, the network showed the best performance. However, according to R^2 score, predicted concentrations could not form a perfect linear relationship. For more than 50 filters, the network complexity increased and failed to identify some spectrums because of possible overfitting.

Table 4.17: Average MAE and R^2 score of the CNN model for 50, 100 and 200 filters.

Number of Filters	R^2 Score	MAE
50	0.93	1.43
100	0.93	1.55
200	0.92	1.58

Since the spectrum lengths were 164, the number of filters was changed between 5 and 35 to find optimal window length of the convolutional layer. Number of filters was set to 100 for the training of the network. The average performance of the network is shown in Table 4.18. The lowest MAE was obtained by filter length 25. For filter length 5, R^2 score was decreased below 0.90 and failed to capture important regions in the spectrums.

Table 4.18: Average MAE and R^2 score of the CNN model for different filter lengths.

Filter Length	R^2 Score	MAE
5	0.87	1.60
10	0.92	1.56
15	0.93	1.55
25	0.94	1.47
35	0.93	1.50

Stride of the convolutional layer was set to 3, 5 and 7 for fixed filter length 25. Number of filters was set to 50 and 100. The performance of the network is shown in Table 4.19. The best model was achieved by 100 filters and stride 5. For stride 7, the input vectors were shrunk too much, overlapping of the windows decreased, and then MAEs began to increase. For smaller stride 3, overlapping of the windows increased and complexity of the model possibly increased.

Table 4.19: Average MAE and R^2 score of the CNN model for various strides and filter length 25.

Stride	Number of Filters	R^2 Score	MAE
3	50	0.93	1.43
	100	0.94	1.47
5	50	0.94	1.34
	100	0.95	1.33
7	50	0.94	1.36

Standard deviation of Gaussian noise in the input layer was set to 0.05, 0.01 and 0.1 for filter length 25, stride 5 and 100 filters. The performance of the network is shown in Table 4.20. As expected, increasing the noise in the input layer caused high error rates. The most convenient value was seemed to be 0.01.

Table 4.20: Average MAE and R^2 score of the CNN model for various noise level in the input.

Standard Deviation of Noise	R^2 Score	MAE
0.01	0.95	1.33
0.05	0.95	1.38
0.1	0.94	1.39

The initial learning rate of Adam optimizer was set to 0.02 and 0.08 for filter length 25, stride 5 and 100 filters. As shown in Table 4.21, the learning rate that Bayesian optimizer found was seemed to be optimal.

Table 4.21: Average MAE and R^2 score of the CNN model for different learning rates.

Learning Rate	R^2 Score	MAE
0.02	0.938	1.53
0.0413	0.95	1.33
0.08	0.897	1.74

As shown in Table 4.22, the dropout probability was increased to control overfitting for filter length 25, stride 5, learning rate 0.0413 and 100 filters. As seen in the table, increasing the dropout probability reduced the performance of the network.

Table 4.22: Average MAE and R^2 score of the CNN model for various dropout probabilities.

Dropout Probability	R^2 Score	MAE
0.02	0.95	1.33
0.1	0.945	1.46
0.2	0.928	1.67

Structure of the proposed network was changed by adding more convolutional and fully connected layers. The purpose of adding new layers is to check whether the new layers extract new features or not. Firstly, three layers having one convolutional layer with the same L1-L2 regularization as the present convolutional layer, batch normalization layer and dropout layer with 0.02 probability were added to the network after the present batch normalization layer, respectively. The network was tested with different number of filters and filter lengths. As shown in Table 4.23, number of filters (F), filter length (L) and stride (S) of the convolutional layers were tested for different configurations. As a result, adding new layer increased the complexity of the network and overfit the data.

Table 4.23: Average MAE and R^2 score of the CNN model for two convolutional layers.

Conv. Layer 1	Conv. Layer 2	R^2 Score	MAE
F100-L25-S5	F5-L3-S1	0.936	1.51
F10-L25-S5	F10-L3-S1	0.933	1.51
F50-L25-S5	F50-L3-S1	0.92	1.55

Additional one fully connected layer was added after the flatten layer and number of neurons in this layer was set to 10, 100 and 500 as shown in Table 4.24. As observed in the case of two convolutional layers, increasing the number of neurons in the network decreased the performance of the network.

Table 4.24: Average MAE and R^2 score of the CNN model for additional one fully connected layer.

Number of Neurons	R^2 Score	MAE
10	0.90	1.80
100	0.88	2.07
500	0.81	2.47

As a result, the fine-tuned hyperparameters of the network are shown in Table 4.25. The network was trained and tested 10 times on the original preprocessed dataset and the performance metrics were averaged. The distribution of the dataset by concentration is shown in Figure 3.2. Predicted concentrations are shown in Figure 4.16 and average MAE value of each chemical type is shown in Figure 4.17. As shown in the results, predicted concentrations are very close to the laboratory measurements.

Table 4.25: The hyperparameters found by fine-tuning the parameters found by Bayesian optimizer.

Parameter	Value
Learning Rate	0. 0413
Dropout Probability	0.02
Number of Filters	100
Filter Length	25
Stride	5

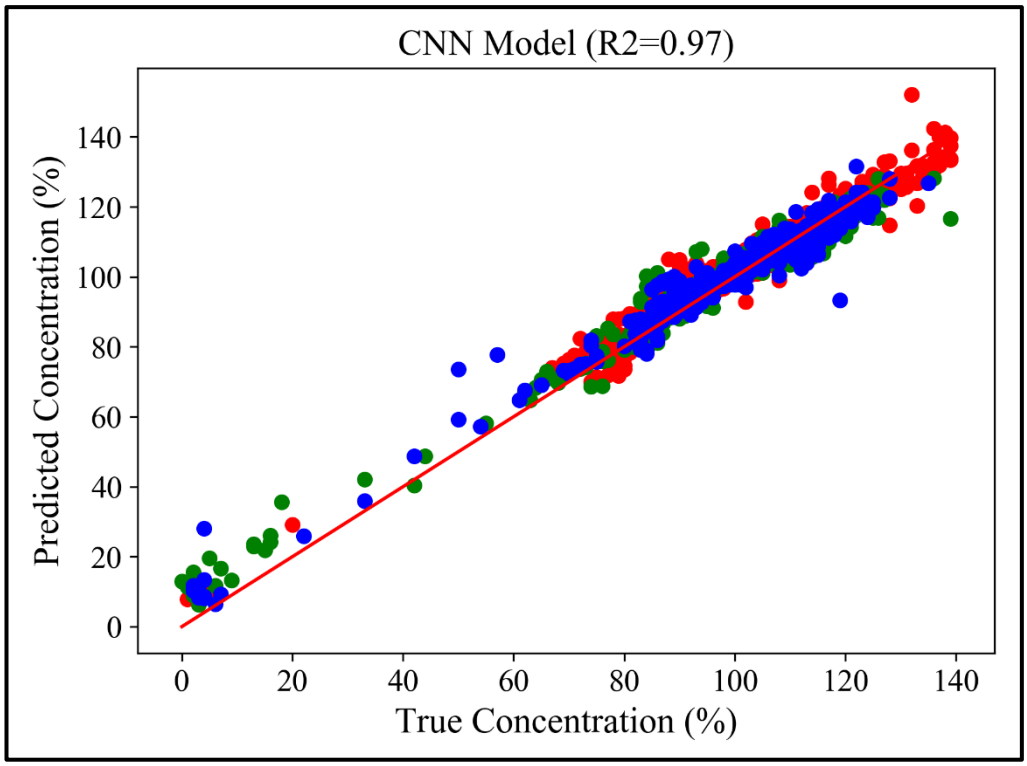

Figure 4.16: Predicted concentrations of the best model trained on the entire dataset with the parameters in Table 4.25. Green, red and blue colors refer to chemical 1, 2 and 3, respectively.

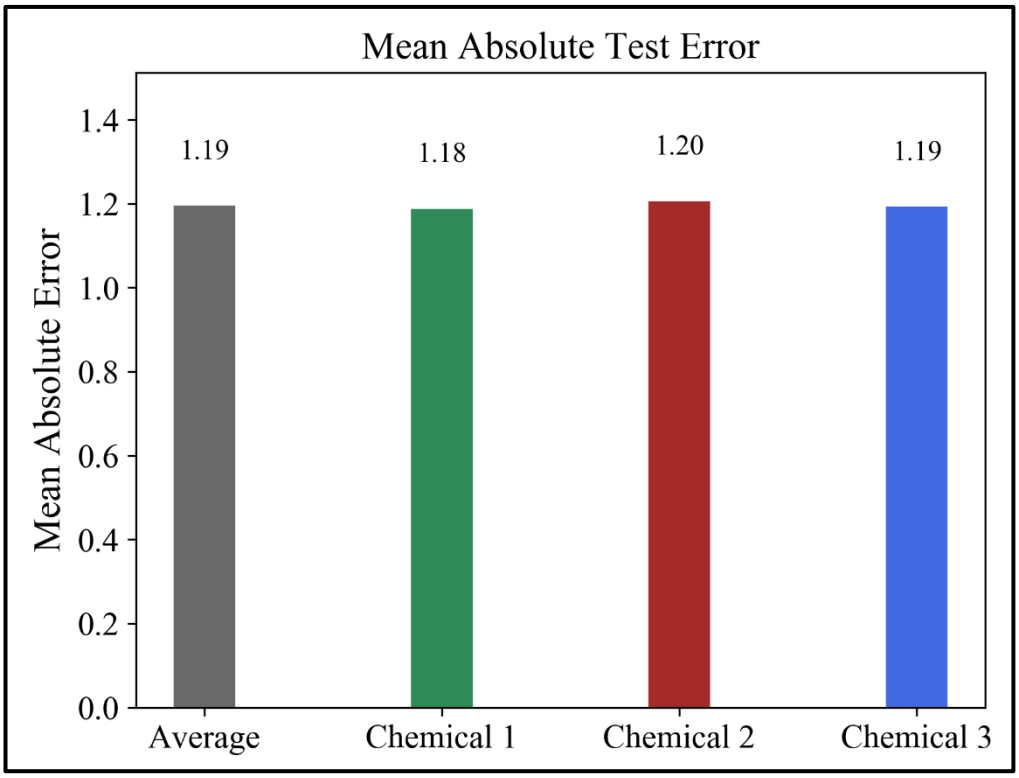

Figure 4.17: Average MAE of the models trained on the entire dataset with the parameters in Table 4.25.

4.4.2. Multi-Tasking Architecture

The proposed 1D CNN with three tasks or outputs is shown in Figure 3.18. By using the hyperparameters in Table 4.15 and Table 4.25, the optimal hyperparameters of the multi-tasking network were explored by testing various parameter configurations manually. As mentioned in Section 3.5.4, the training was done by using two different batch creation procedures. In the first one, the batches were grouped in such a way that each batch contained the same type of spectrums. In the second one, the batches contained mixed type of spectrums as in the training of single-tasking architecture. The idea of the grouped batching is to try whether the multi-tasking network captures common features better or not.

For the multi-tasking network in Figure 3.18, number of filters, filter length, stride parameters and structure of the network was modified and tested for different configurations. The learning rate and dropout probability were not changed since single-tasking and multi-tasking networks were very similar and it was proofed that they were optimal for single-tasking network. For example, after the batch normalization layer, one convolutional layer was added to the shared layer. Similarly, before the flatten layer, one convolutional layer was added to three task layers. One fully connected layer was added to the task layers after the flatten layer. The fully connected layer was tested for 10, 100 and 1000 neurons. Finally, the network was trained and tested on the entire dataset with 75:25 training and testing ratio, by using the fine-tuned parameters.

Table 4.26: Average MAE and R^2 scores for 50, 100 and 200 filters.

Number of Filters	Grouped Batches		Mixed Batches	
	R^2 Score	MAE	R^2 Score	MAE
50	0.922	1.73	0.954	1.37
100	0.933	1.56	0.947	1.37
200	0.938	1.76	0.944	1.42

Firstly, for the parameters in Table 4.15 and Table 4.25, the network was trained by 50, 100, 200 filters. As shown in Table 4.26, the best results were obtained by 50 and 100 filters in mixed batch mode. In both batch modes, the complexity of the network increased for more than 100 filters and the performance of the network reduced.

Table 4.27: Average MAE and R^2 scores for various filter lengths and 50 filters.

Filter Length	Stride	Grouped Batches		Mixed Batches	
		R^2 Score	MAE	R^2 Score	MAE
15	3	0.832	1.87	0.836	1.63
	5	0.888	1.90	0.929	1.46
25	5	0.922	1.73	0.954	1.37

Since the best model was achieved by 50 and 100 filters as shown in Table 4.26, filter length and stride of the convolutional layer was tested for various configurations as shown in Table 4.27 and Table 4.28. As a result, the network showed the best performance for the filter length 25 and stride 5 in mixed batch mode.

Table 4.28: Average MAE and R^2 scores for various filter lengths and 100 filters.

Filter Length	Stride	Grouped Batches		Mixed Batches	
		R^2 Score	MAE	R^2 Score	MAE
15	3	0.874	2.05	0.750	1.51
	5	0.899	1.74	0.934	1.51
	7	0.954	1.60	0.947	1.46
25	3	0.937	1.86	0.911	1.52
	5	0.933	1.56	0.947	1.37
	7	0.963	1.70	0.954	1.45
35	5	0.938	2.25	0.949	1.46

For each branch or task in the network, a dense layer was added before the flatten layer. The number of neurons in the dense layer was tested for 10, 100 and 1000 as shown in Table 4.29. As observed in the fine-tuning of the single-tasking network, adding more layers to the multi-tasking network reduced performance due to increased number of neurons and complexity.

Table 4.29: Average MAE and R^2 scores for 10, 100 and 1000 neurons in task layers.

Number of Neurons	Grouped Batches		Mixed Batches	
	R^2 Score	MAE	R^2 Score	MAE
10	0.920	1.79	0.937	1.51
100	0.903	1.81	0.895	1.86
1000	0.679	3.44	0.569	3.82

Table 4.30: Average MAE and R^2 scores for extra one convolutional layer in the shared layer.

Configuration		Grouped Batches		Mixed Batches	
Convolution 1	Convolution 2	R^2 Score	MAE	R^2 Score	MAE
F100-L25-S5	F5-L5-S1	0.90	1.55	0.94	1.40
	F10-L5-S1	0.89	1.64	0.92	1.41
	F30-L5-S1	0.85	2.07	0.92	1.58

Structure of the shared layer in the network was modified by adding an extra convolutional layer after the present batch normalization layer. To control overfitting and speed up learning dropout and batch normalization layers were added after the new convolutional layer. As shown in Table 4.30, number of filters (F), filter length (L) and stride (S) parameters were tested and the best performance was obtained by F5-L5-S1 configuration in mixed batch mode. For higher number of filters, performance of the network became worse.

Table 4.31: Average MAE and R^2 scores for extra one convolutional layer in the task layers.

Configuration		Grouped Batches		Mixed Batches	
Convolution 1	Convolution 2	R^2 Score	MAE	R^2 Score	MAE
F100-L25-S5	F5-L5-S1	0.93	1.57	0.91	1.55
	F10-L5-S1	0.89	1.68	0.83	1.69

Structure of the task layers was modified by adding an extra convolutional layer before the flatten layers. As in modification of shared layer above, one convolutional layer, batch normalization layer and dropout layer were added. The idea of this was to try whether increasing complexity in these branches helped the network identify the spectrums better or not. As shown in Table 4.31, the performance of the network reduced, although number of filters and filter lengths were small.

As a result, modifying the parameters in Table 4.25 such as number of filters, filter lengths, strides and number of neurons reduced test performance of the network. As observed in single-tasking network optimization, increasing complexity of the network with regularization also decreased average test performance of the network. Mixed batching provided better results than grouped one since grouped batching might cause the shared layer to adapt to each task and the network could not capture common features properly. The average MAE of the network for entire dataset is shown in Figure 4.19. For the model with best R^2 score in 10 trials, predicted concentrations are

shown in Figure 4.18. The performance of the network is slightly worse than single-tasking one and it seems that multi-tasking architecture is not suitable for this problem.

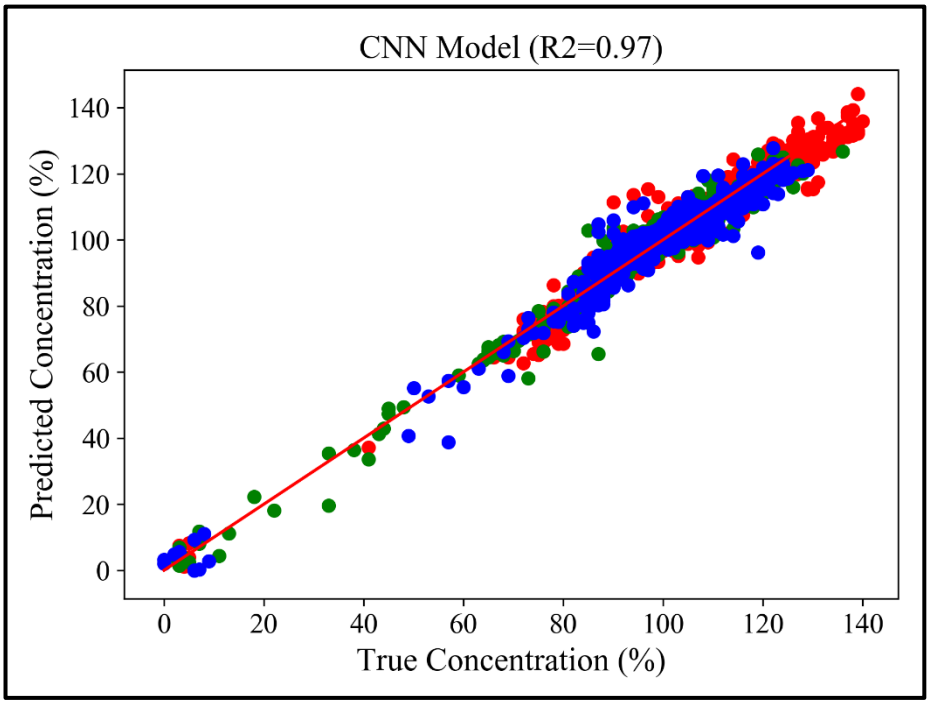

Figure 4.18: Predicted concentrations of the best model trained on the entire dataset with the parameters in Table 4.25. Green, red and blue colors refer to chemical 1, 2 and 3, respectively.

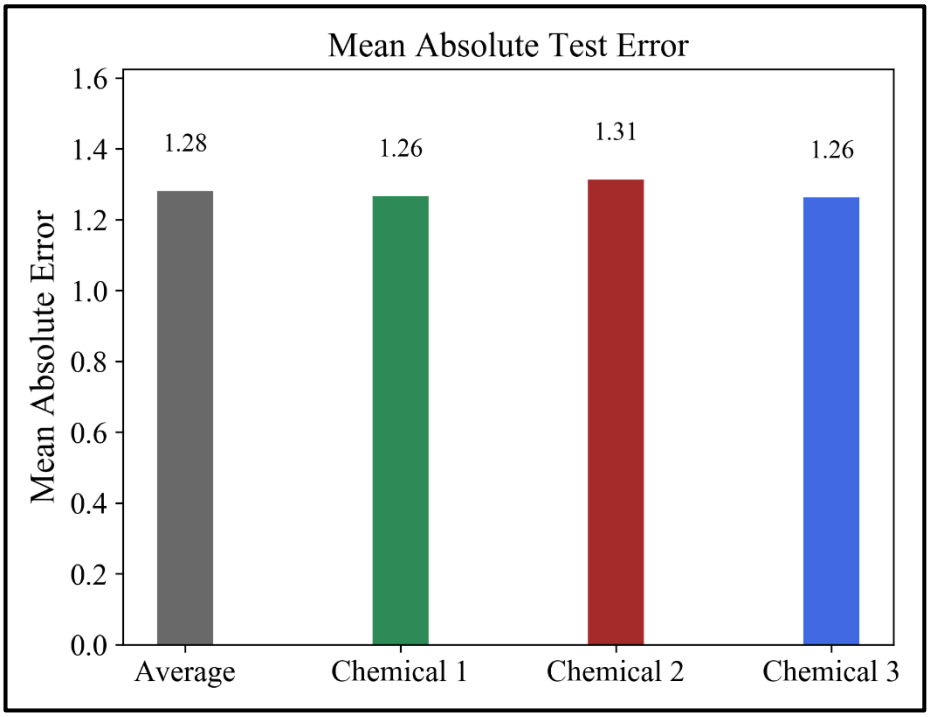

Figure 4.19: Average MAE of the models trained on the entire dataset with the parameters in Table 4.25.

4.4.3. Multivariate Regression

Multivariate regression is a technique to create a model for prediction of multiple response variables. In this work, there are three response variables since the dataset has three different spectrum types defined in Section 3.1. The response variables are concentration of X substance.

The proposed single-tasking 1D CNN architecture is shown in Figure 3.17. The network was modified to have three outputs instead of one output and tested with various hyperparameters. This architecture is an alternative to the proposed multi-tasking network. The modified network was trained with the same procedure as single-tasking network. The target concentration values were converted to the vectors of size three for multivariate regression. The three chemical types were mapped to indexes from 0 to 2 and the vectors were filled according to these indexes. For the first, second and third chemical types, the target vectors are $[C_1\ 0\ 0]$, $[0\ C_2\ 0]$ and $[0\ 0\ C_3]$ where C_1, C_2 and C_3 are corresponding concentration values, respectively. The network was trained with the fine-tuned parameters of single-tasking network denoted in Table 4.25. The network was tested 10 times and the performance metrics were averaged. The average R^2 score was -3.19 and the average MAE was 15.10 as shown in Table 4.32.

Table 4.32: Average R^2 score and MAE of the multivariate regression models.

Chemical Type	MAE
Chemical 1	8.90
Chemical 2	18.25
Chemical 3	18.14
Average	15.10

The negative value of R^2 score means that the model does not follow the trend of the data. As a result, the multivariate regression architecture could not learn from the dataset and failed to capture the linear and nonlinear relationships between spectrums and concentration values.

4.5. Long Short-Term Memory

Since the absorbance spectrums in the dataset are sequential in terms of wavelengths, two different LSTM networks are designed as shown in Figure 4.20 and Figure 4.21. By using the hyperparameters in Table 4.33, the optimal hyperparameters of the networks were explored manually by testing various parameter configurations with the entire dataset. The training and testing sets were created randomly by using 75% and 25% of the dataset, respectively. The networks were trained and tested 10 times and the performance metrics were averaged.

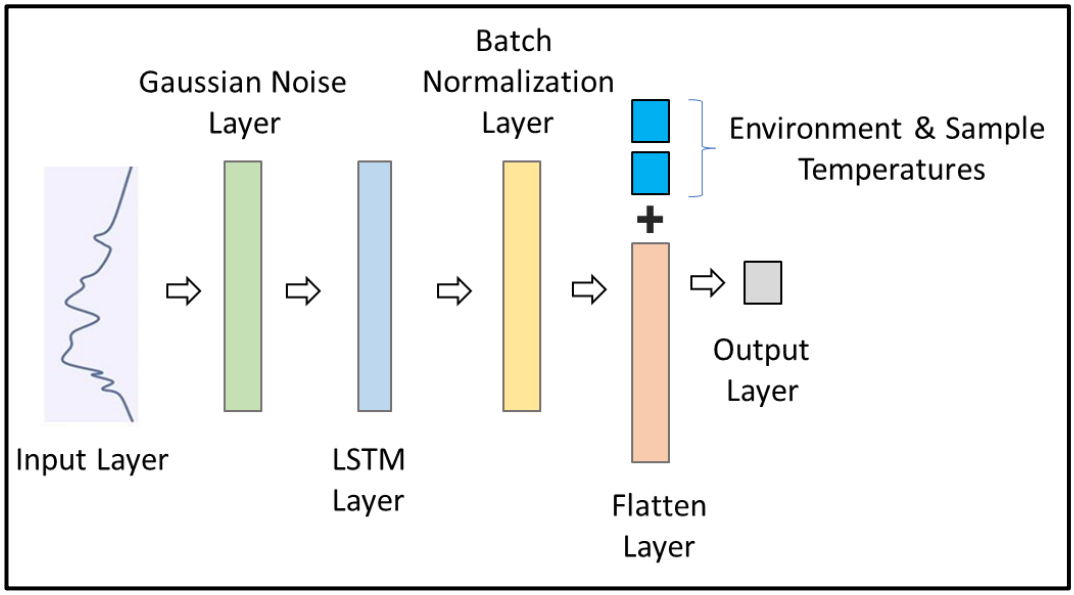

Figure 4.20: Single-layered LSTM network architecture.

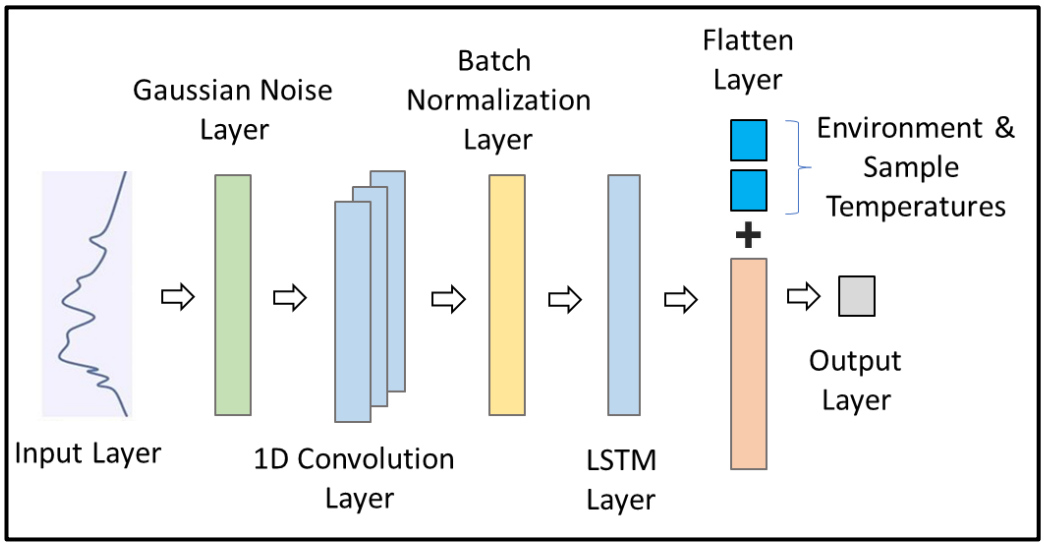

Figure 4.21: 1D CNN with LSTM network architecture.

Table 4.33: Initial parameters of the LSTM network.

Parameter	Value
Loss Function	MSE
Optimizer	Adam
Gaussian Noise	10^{-2}
Training Epoch	150
Batch Size	60
Weight Initializer	Glorot Uniform
Learning Rate	0.0413

Single-layered LSTM network with noise and batch normalization layers is shown in Figure 4.20. The hyperparameters of the LSTM layer is shown in Table 4.34. Hard sigmoid function is used for activation of the gates in LSTM since it is easy to compute compared to sigmoid function which requires computation of e^x term. Hard sigmoid is defined as $max(0,\ min(1,\ x * 0.2\ +\ 0.5))$ where x is the input. Hyperbolic tangent function is used to compute output of LSTM layer explained in Section 3.6. Dropout operation was only applied to the input of the LSTM layer with 0.02 probability. Kernel weights were initialized by using Glorot [16] uniform and recurrent weight matrices were initialized to be random orthogonal.

Table 4.34: Initial parameters of the LSTM layer.

Parameter	Value
Input Dropout	0.02
Output Activation	Hyperbolic Tangent
Recurrent Activation	Hard Sigmoid
Kernel Weight Initializer	Glorot Uniform
Recurrent Weight Initializer	Orthogonal

The single-layered LSTM network was tested by changing dimensionality of the LSTM layer to 1, 5, 10 and 20. The output of the LSTM layer for these units was $(B, 164, 1)$, $(B, 164, 5)$, $(B, 164, 10)$ and $(B, 164, 20)$, respectively. The length of the spectrums is 164 and B is the number of spectrums in each batch.

The average performances of the network for different number of units are denoted in Table 4.35. For 10 units, the trained model demonstrated the best performance and MAE of the model was better than that of machine learning models. For 20 units, R^2 score stayed the same, but MAE increased. This shows that predictions are mostly linear, but a bit far from true values.

Table 4.35: Average MAE and R^2 score of the single-layered LSTM model for dimensions of 1, 5, 10 and 20.

Number of Units	R^2 Score	MAE
1	0.83	2.44
5	0.91	1.76
10	0.94	1.34
20	0.91	1.81

As a second alternative, the network with LSTM and 1D CNN layers in Figure 4.21 was tested for various architectures and hyperparameters. The intuition behind of this architecture is that CNN extracts spatial features and LSTM captures temporal structures since the spectrums are a type of sequential data. For the training, the parameters in Table 4.33, Table 4.34 and Table 4.36 was used and the network was tested for 20 and 100 units for LSTM layer. The output of the CNN layer and input of LSTM layer was $(B, 28, 100)$ where B is size of batches. For 20 and 100 units, the output of LSTM layer was $(B, 560)$ and $(B, 2800)$, respectively.

Table 4.36: The hyperparameters of the CNN layer for the network with LSTM and 1D CNN layers.

Parameter	Value
Number of Filters	100
Filter Length	25
Stride	5

Table 4.37: Average MAE and R^2 score of the model with 1D CNN and LSTM layers for 20 and 100 units.

Number of Units	R^2 Score	MAE
20	0.94	1.37
100	0.95	1.28
150	0.94	1.33

The average performances of the network for different number of units are denoted in Table 4.37. For 20 and 150 units, the corresponding models demonstrated similar performances. The best model was achieved by 100 units and its performance was similar to the performance of the multi-tasking model. As a result, both architectures learned spectral features and demonstrated considerable performances compared to machine learning models.

4.6. Analysis and Results

The PLSR, GBR, RF, CNN and LSTM models were trained on the normalized dataset without outliers. The models were trained 10 times and the performance metrics were averaged. In this way, the parameters of the models were optimized by testing various parameter configurations. For the fine-tuned parameters, the performance of the models is shown in Table 4.38.

Table 4.38: Average R^2 score and MAE of all the proposed models.

Model	R^2 Score	MAE
Single-Task CNN	0.95	1.19
Multi-Task CNN	0.95	1.28
CNN + LSTM	0.95	1.28
Single LSTM	0.94	1.37
Gradient Boosting	0.97	2.41
Random Forests	0.97	2.59
Partial Least Squares	0.94	3.87
Multivariate CNN	-3.19	15.10

As explained in Section 1.3, chemical and instrumental factors such as scattering of light, stray light, deviations in absorptivity coefficients at high and low concentrations cause nonlinearities between concentrations of a target substance and absorbance spectrums.

The PLSR algorithm is a type of linear regression algorithm that applies MLR on the new features extracted by PCA. The PLSR model yielded the worst MAE and R^2 score among all the models since the dataset contains multiple spectrum types and nonlinear relations.

RF and GBR are ensemble learning methods based on regression trees. They are tolerant to overfitting and can capture nonlinear relations in training data. As shown in Table 4.38, the performance of the models was superior to the PLSR model. They both managed to capture nonlinear relations in data. The GBR model produced less MAE then the RF model, but training time of the GBR model was much higher.

CNNs are easy to use, noise tolerant, translation invariant and can capture local patterns in the inputs. As shown in Table 4.38, the single-tasking network demonstrated slightly better performance then multi-tasking one. They produced the lowest MAE among all the models. They both managed to identify different spectrum

types and captured nonlinear relations, but R^2 scores of the models were slightly lower than that of RF and GBR.

As an alternative to multi-tasking network, multivariate regression method was applied on the single-tasking network, but the multivariate model failed to learn spectral features and the results were disappointing as shown in Table 4.38. The possible reason of this failure may be the zeros in target vectors.

Overfitting problem is very common in deep neural network. Thus, the proposed single-tasking network was modified and trained to predict 31 devices that spectrums were measured. The intuition behind of this was to check whether the proposed deep networks memorize the dataset or not for specific devices. By using the hyperparameters of single-tasking network, Softmax and categorical cross-entropy, average accuracy of 10 runs was %12. The network was fine-tuned for device prediction problem and the best average accuracy was %45 for 10 filters in 1D CNN layer. This can be interpreted as that the chance of overfitting was weak for these networks.

LSTMs are a special type of RNN that is capable of processing sequential information such as a numerical time-series data. They have an internal memory to store historical information. They are good for capturing temporal structures and long-term dependencies. In this work, LSTMs were tested with and without 1D CNN layers and acquired significant performances as shown in Table 4.38. However, the best MAE was acquired by the single-tasking deep network.

5. CONCLUSIONS

Light is an electromagnetic radiation of longer and shorter wavelength and all materials absorb the energy of light around them. The intensity of the absorbed energy can be expressed as a function of frequency and wavelength. Absorption spectroscopy is used to measure energy that is absorbed by a substance. It is the most frequently used spectroscopic technique for liquids and gases as it is simple, accurate and easy to use. Absorption spectrum of a substance is calculated by measuring absorptions over a range of frequencies [1]. Molecular and atomic structure of a material affect the spectrum considerably. An absorbance spectrum can be used to identify substances or measure the concentration of a molecule in solution. It has many applications such as particle size analysis, polymer processing, trace detection of metals, ozone monitoring, analysis of composition in dairy products and clinical blood diagnostics. The purpose of this work is to predict concentration of a specific substance by using learning models trained on the absorbance spectrums of liquids.

There are lots of related regression and classification works based on absorption spectrums. Mostly used methods in these works are PLS, LDA SVM, MLP, RF and KNN. Recent works use deep neural networks to deal with nonlinearity and noise due to environmental and instrumental factors. According to related works, performance of the CNNs are superior to these machine learning models for classification and regression tasks since they are easy to use, noise tolerant and deal with nonlinearity better.

In this work, total of 6167 absorbance spectrums were obtained from three different type of liquid solvents containing a specific X substance by using total of 31 measurement systems. Concentration of X substance in these solutions were predicted by using learning models trained on the spectrums. Inspiring from the previous works, PLSR, RF, GBR and CNN were used as the learning models. In order to increase accuracy of the models and speed up learning, data set was scaled, smoothed and cleaned from outliers by PLSR modeling. Single-tasking and multi-tasking CNN architectures were designed and optimized. The models were compared by performance metrics of MAE and R^2 score.

The PLSR, GBR and RF models were trained with 10-fold cross-validation on the preprocessed dataset without outliers. Total of 10 models were trained and average

of MAEs and R^2 scores of these models were computed. The optimal parameters of the models were explored by testing various parameter configurations. The best PLSR model produced the worst performance since the dataset contains multiple spectrum types and nonlinear relations. The performance of the best RF and GBR models were better and demonstrated similar performances. They both managed to capture nonlinear relations. The performance of the GBR model was superior to the RF model, but training time of the GBR model was much higher.

Single-tasking, multi-tasking and multivariate 1D CNN architectures and LSTM networks were designed, trained and tested on the preprocessed dataset without outliers. Total of 10 models were trained and tested for each parameter configuration and the optimal parameters were explored. The single-tasking network demonstrated slightly better performance than that of multi-tasking network, but performance of the LSTM networks was a bit lower than that of multi-tasking network. Multivariate regression models failed to learn spectral features and produced the worst MAE and R^2 score among all the models. The proposed single-tasking 1D CNN model produced the lowest MAE among all the models. The deep learning models managed to identify different spectrum types, learned spectral features and captured nonlinear relations. However, R^2 scores of the models were slightly lower than that of RF and GBR.

In conclusion, it is shown that 1D CNNs are better than robust machine learning algorithms such as RF and GBR for the spectrum dataset containing multiple type of spectrums. In future works, more absorbance spectrums are recommended to be measured for concentrations lower than 80 and higher than 120 since the dataset contains fewer samples for this concentration range. Thus, distribution of the samples by concentration will be even and the models are most likely to learn the nonlinear relations better.

REFERENCES

[1] Hollas J. M., (2004), “Modern Spectroscopy”, 4th Edition, John Wiley & Sons.

[2] Bjerrum E. J., Glahder M., Skov T., (2017), “Data Augmentation of Spectral Data for Convolutional Neural Network (CNN) Based Deep Chemometrics”, arXiv preprint: 1710.01927.

[3] Geladi P., Kowalski B. R., (1986), “Partial least-squares regression: a tutorial”, Analytica chimica acta, 185, 1-17.

[4] Huber P. J., (1964), “Robust estimation of a location parameter”, The annals of mathematical statistics, 35(1), 73-101.

[5] Nair V., Hinton G. E., (2010), “Rectified linear units improve restricted boltzmann machines”, In Proceedings of the 27th international conference on machine learning (ICML-10), 807-814, Haifa, Israel, 21-24 June.

[6] Zeiler M. D., (2012), “ADADELTA: an adaptive learning rate method”, arXiv preprint arXiv:1212.5701.

[7] Web 1, (2018), https://github.com/SheffieldML/GPyOpt, (Date of Access: 10/04/2018).

[8] Acquarelli J., van Laarhoven T., Gerretzen J., Tran T. N., Buydens L. M., Marchiori E., (2017), “Convolutional neural networks for vibrational spectroscopic data analysis”, Analytica chimica acta, 954, 22-31.

[9] Engel J., Blanchet L., Buydens L.M., Downey G., (2012), “Confirmation of brand identity of a trappist beer by mid-infrared spectroscopy coupled with multivariate data analysis”, Talanta, 99, 426-432.

[10] Skov T., Ballabio D., Bro R., (2008), “Multiblock variance partitioning: A new approach for comparing variation in multiple data blocks”, Analytica chimica acta, 615(1), 18-29.

[11] Dyrby M., Engelsen S. B., Nørgaard L., Bruhn M., Lundsberg-Nielsen L., (2002), “Chemometric quantitation of the active substance (containing C≡N) in a pharmaceutical tablet using near-infrared (NIR) transmittance and NIR FT-Raman spectra”, Applied Spectroscopy, 56(5), 579-585.

[12] Briandet R., Kemsley E. K., Wilson R. H., (1996), “Discrimination of Arabica and Robusta in instant coffee by Fourier transform infrared spectroscopy and chemometrics”, Journal of agricultural and food chemistry, 44(1), 170-174.

[13] Tapp H. S., Defernez M., Kemsley E. K., (2003), “FTIR spectroscopy and multivariate analysis can distinguish the geographic origin of extra virgin olive oils”, Journal of agricultural and food chemistry, 51(21), 6110-6115.

[14] Rasmus A., Berglund M., Honkala M., Valpola H., Raiko T., (2015), “Semi-supervised learning with ladder networks”, In Advances in Neural Information Processing Systems, 28, 3546-3554.

[15] Al-Jowder O., Kemsley E. K., Wilson R. H., (1997), “Mid-infrared spectroscopy and authenticity problems in selected meats: a feasibility study”, Food Chemistry, 59(2), 195-201.

[16] Glorot X., Bengio Y., (2010), “Understanding the difficulty of training deep feedforward neural networks”, In Proceedings of the thirteenth international conference on artificial intelligence and statistics, 249-256, Sardinia, Italy, 13-15 May.

[17] Bottou L., (2010), “Large-scale machine learning with stochastic gradient descent”, In Proceedings of COMPSTAT-2010, 177-186, Paris, France, 22-27 August.

[18] Bergstra J., Bengio Y., (2012), “Random search for hyper-parameter optimization”, Journal of Machine Learning Research, 13, 281-305.

[19] Savitzky A., Golay M. J., (1964), “Smoothing and differentiation of data by simplified least squares procedures”, Analytical chemistry, 36(8), 1627-1639.

[20] Barnes R. J., Dhanoa M. S., Lister S. J., (1989), “Standard normal variate transformation and de-trending of near-infrared diffuse reflectance spectra”, Applied spectroscopy, 43(5), 772-777.

[21] Guo Q., Wu W., Massart D. L., (1999), “The robust normal variate transform for pattern recognition with near-infrared data”, Analytica chimica acta, 382(1-2), 87-103.

[22] Engel J., Gerretzen J., Szymańska E., Jansen J. J., Downey G., Blanchet L., Buydens L. M., (2013), “Breaking with trends in pre-processing?”, TrAC Trends in Analytical Chemistry, 50, 96-106.

[23] C. Wolf, D. Gaida, A. Stuhlsatz, T. Ludwig, S. McLoone, M. Bongards, (2013), “Predicting organic acid concentration from UV/VIS spectrometry measurements–a comparison of machine learning techniques”, Transactions of the Institute of Measurement and Control, 35(1), 5–15.

[24] Draine B. T., (2003), “Interstellar dust grains”, Annual Review of Astronomy and Astrophysics, 41(1), 241-289.

[25] Li A., Greenberg J. M., (2003), “In dust we trust: an overview of observations and theories of interstellar dust”, In Solid State Astrochemistry, 120, 37-84.

[26] Li A., Draine B. T., (2001), “Infrared emission from interstellar dust. II. The diffuse interstellar medium”, The Astrophysical Journal, 554(2), 778.

[27] Peeters E., Allamandola L. J., Hudgins D. M., Hony S., Tielens A. G. G. M., (2003), “The unidentified infrared features after ISO”, arXiv preprint astro-ph/0312184.

[28] Yuan X., Li M., Gaddam S., Li X., Zhao Y., Ma J., Ge J., (2016), ”DeepSky: Identifying Absorption Bumps via Deep Learning”, 2016 IEEE International Congress on Big Data, 214-221, San Francisco, USA, June 27 – July 2.

[29] Quider A. M., Nestor D. B., Turnshek D. A., Rao S. M., Monier E. M., Weyant A. N., Busche J. R., (2011), “The pittsburgh sloan digital sky survey mg ii quasar absorption-line survey catalog”, The Astronomical Journal, 141(4), 137.

[30] Krizhevsky A., Sutskever I., Hinton G. E., (2012), “Imagenet classification with deep convolutional neural networks”, In Advances in neural information processing systems, 2, 1097-1105.

[31] Szegedy C., Liu W., Jia Y., Sermanet P., Reed S., Anguelov D., Rabinovich A., (2015), “Going deeper with convolutions”, arXiv preprint arXiv:1409.4842.

[32] Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Vanderplas J., (2011), “Scikit-learn: Machine learning in Python”, Journal of machine learning research, 12, 2825-2830.

[33] Breiman L., (2001), “Random forests”, Machine Learning, 45(1), 5–32.

[34] Alpaydin E., (2014), “Introduction to Machine Learning”, 3th Edition, The MIT Press.

[35] Friedman J. H., (2001), “Greedy function approximation: a gradient boosting machine”, Annals of Statistics, 29(5), 1189–1232.

[36] Copas J. B., (1983), “Regression, prediction and shrinkage”, Journal of the Royal Statistical Society, Series B (Methodological), 45(3), 311-354.

[37] LeCun Y., Bengio Y., Hinton G., (2015), “Deep learning”, Nature, 521(7553), 436.

[38] O'Shea K., Nash R., (2015), “An introduction to convolutional neural networks”, arXiv preprint arXiv:1511.08458.

[39] Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R., (2014), “Dropout: A simple way to prevent neural networks from overfitting”, The Journal of Machine Learning Research, 15(1), 1929-1958.

[40] Ioffe S., Szegedy C., (2015), “Batch normalization: Accelerating deep network training by reducing internal covariate shift”, arXiv preprint arXiv:1502.03167.

[41] Kingma D. P., Ba J., (2014), “Adam: A method for stochastic optimization”, arXiv preprint arXiv:1412.6980.

[42] Ruder S., (2017), “An overview of multi-task learning in deep neural networks”, arXiv preprint arXiv:1706.05098.

[43] Deng L., Hinton G. E., Kingsbury B., (2013), “New types of deep neural network learning for speech recognition and related applications: An overview”, 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, 8599–8603, Vancouver, Canada, 26-31 May.

[44] Ramsundar B., Kearnes S., Riley P., Webster D., Konerding D., Pande V., (2015), “Massively multitask networks for drug discovery”, arXiv preprint arXiv:1502.02072.

[45] Baxter J., (1997), “A Bayesian/information theoretic model of learning to learn via multiple task sampling”, Machine Learning, 28(1), 7–39

[46] Goodfellow I., Bengio Y., Courville A., (2016), “Deep learning”, Vol. 1, Cambridge: MIT press.

[47] Hochreiter S., Schmidhuber J., (1997), “Long short-term memory”, Neural computation, 9(8), 1735-1780.

[48] Walt S. V. D., Colbert S. C., Varoquaux G., (2011), “The NumPy array: a structure for efficient numerical computation”, Computing in Science & Engineering, 13(2), 22-30.

[49] Web 2, https://github.com/fchollet/keras, (Date of Access: 10/05/2018).

[50] Abadi M., Barham P., Chen J., Chen Z., Davis A., Dean J., Kudlur M., (2016), “TensorFlow: A System for Large-Scale Machine Learning”, In OSDI-16, 265-283.

[51] Hunter J. D., (2007), “Matplotlib: A 2D graphics environment”, Computing in science & engineering, 9(3), 90-95.

BIOGRAPHY

Emre Ardıç was born in 1991 in Turkey. He graduated at the top of Computer Engineering Department from Gebze Technical University at 2014. He started his professional carrier as Software Developer at PHI Tech Bioinformatics Inc. in 2014. Since 2015, he has been a graduate student at the Computer Engineering Department of Gebze Technical University. Currently, he is working as Researcher at TÜBİTAK BİLGEM. His research interests include computer vision, artificial intelligence, machine learning and deep learning.

APPENDICES

Appendix A: The Publications about the Thesis

Ardıç E., Genç Y., (2018), "Classification Of 1D Signals Using Deep Neural Networks", 26th IEEE Signal Processing and Communication Applications Conference (SIU), İzmir, Turkey, 02-05 May.