Master's Thesis 2022 30 ECTS
Faculty of Science and Technology (Realtek)

Explore the Effect of Data Augmentation of Spectroscopic Data for Deep Learning Models

Talha Naveed
MS Data Science

List of Figures

List of Tables

Contents

Abstract

This study aimed to explore the effect of data augmentation techniques to improve the performance of deep learning models on data sets that contain more features than samples. The data set used as an example for this case study was Raman spectroscopic data. Deep learning models have a reputation of requiring large amounts of data and the sample size of data used in this study was small. As a way to study the effect of increasing sample size, three different augmentation techniques based on the principles of linear combinations, partial least squares and extended multiplicative scatter correction were developed and used to increase the sample size. The effect of these three augmentation techniques was studied for a convolutional neural network by training and evaluating the neural network using augmented data. The evaluation of the augmentation methods was based on the performance of the convolutional neural network. In order to study the behavior of the convolutional neural network, the performance of the neural network was compared to partial least squares regression model. Furthermore, learning curves where also used to analyze the performance of the neural network based on the sample size. The augmentation methods were used to artificially increase the sample size and the learning curves were used to see if the increase in sample size lead to improvement. The results of this study showed that using augmentation techniques to increase sample size does improve the performance of model.

1 Introduction

Spectroscopy is a technique that is used to study the structure of atoms and molecules. When light is passed through these structures using a light producing source, they absorb or emit light at hundreds of different wavelengths. This behavior makes it possible to investigate the composition and properties of any substance. Furthermore, spectroscopy is also useful as an analytical method because it can be used to find the constituents in a substance having unknown chemical composition. A typical spectroscopy experiment involves passing light from a source through any sample of interest which results in absorption or emission of light. In case of emission, the sample under study emits light at a different wavelength than the source. However, during absorption, the sample absorbs energy from the light source. A spectrometer is an instrument that is used to analyse the wavelength of electromagnetic radiations by measuring and separating the spectral components based on their physical and chemical properties. Spectrometer works by taking in light from a source, splitting it into spectral components, digitize the light signal as a function of wavelength and then finally displaying it through a computer. There are many different applications and types of spectroscopy. Vibrational spectroscopy is one type of spectroscopy where the emitted light is affected by molecular vibrations in the molecular structure, manifesting itself as peaks or overtones at various wavelengths in the spectra. Such type of spectra contains a lot of information about atoms and molecules of a sample. Raman spectroscopy is a sub type of vibrational spectroscopy that has been widely used in chemical identification in recent decade [1].

Spectroscopy have shown promising results in experimental studies however the chemical analysis method is long and time consuming [1]. Artificial intelligence is a famous principle that is now being used in many scientific fields to speed up processes. Machine learning(ML) is a subset of artificial intelligence and it provides a set of algorithms that has the ability to learn and perform specific tasks without providing explicit set of instructions. Machine learning techniques have been applied to multiple chemical problems in recent years and have shown promise in the advancement of several areas of chemistry [2]. Spectroscopic data is high dimensional and possibly highly inter correlated which means that it has high number of variables and there are correlations among the variables [3]. For spectroscopic data, the feature space can become so large that that the number of samples are left small. When the variables of the data are highly inter correlated then it is said that the data suffers from multicollinearity [3]. Multicollinearity is a problem since it undermines the statistical significance of a variable. This possess an issue for linear machine learning algorithms as they require input data features to be independent [4]. Furthermore, simple linear models do not work well when the number of features are greater than the number of samples. Multi variate analysis methods can solve these problems by extracting new orthogonal latent features that are the combination of existing features. Partial least squares regression(PLSR) is a type of ML multivariate analysis technique and is used in this study to analyze Raman spectroscopy data.

Another challenge while working with spectroscopic data is that it usually requires preprocessing. Preprocessing is defined as the manipulation of data in search of achieving enhanced performance and it is often a critical step during data modelling. All types and sub types of spectroscopy are used to study the chemical information of samples by analyzing the absorbance and emitted spectrum and this interpretation may be difficult if the data suffers from any kind of noise or effects. Raw spectra may be distorted due to high degree of noise and other effects like scattering and instrumental effects [5]. Scattering effects are usually occur to the state of the sample while instrumental effects arise due to measurement conditions that can include the condition and quality of the instrument. Raman spectra also suffers from scattering effects like fluorescence's baseline variation and shot noise [6] [1]. Baseline variations that are a part of the Raman spectrum are uncorrelated to the sample compositions and it should be removed after acquiring the spectrum to minimize adverse effects [1]. Extended multiplicative scatter correction(EMSC) is a well known method for preprocessing spectral data and this method is also adapted for the baseline variations experienced in Raman spectroscopic data due to fluorescence effects.

Machine learning methods require feature engineering or preprocessing and that is why it may not be optimal to use to these methods when the number of features are large. Deep learning is a sub field of machine learning and in recent years it is the state of the art technology that is being used in image analysis and many other domains as well [7]. Deep learning models have proved to be a a very powerful tool because of its ability to handle large amounts of data. The principle of using multiple hidden layers for modelling has surpassed the traditional techniques especially in pattern recognition.

There are different kinds of deep neural networks and one of the most popular kind is the convolution neural network (CNN) which has become a prominent tool for computer vision and text analysis [8]. CNNs are designed to extract features from an input signal with different levels of abstraction. The combinations of different types of layers in a CNN extracts features(patterns) hierarchically. Extracted features are then used to make predictions. CNNs are end to end trainable system and they offer an alternative to a pipeline in which each part is trained independently or crafted manually. CNNs have recently become popular due their large scale success in image recognition problems and because to its widespread success, researchers are motivated to used it for spectral data as well [8]. However, CNN is a data hungry model and this does not comply well with the frequent possibility of spectral data having limited samples. To account for limited sample sizes, deep learning community uses a technique called data augmentation. Data augmentation is a method for data set extension. The principle of data augmentation is to apply transformation techniques on existing original data to derive new artificially created observations called augmented observations. These observations can then be used for training models and providing them with sufficient data. However, there have not been a lot of research performed on the augmentation methods for spectral data.

Motivated by the increasing popularity of CNNs and the limited research present for the augmentation methods for spectral data, three augmentation methods for generating artificial data samples for spectroscopic data were developed and tested using a CNN model in this study. The three augmentation methods were named as linear augmentation, PLSR augmentation and EMSC augmentation. All three methods were based on different principles. The linear augmentation method works by linearly combining samples from the existing data. PLSR augmentation generates new samples by reconstructing the original data matrix with some deviations using the scores and loadings matrix of a PLSR model fitted on original data whereas the EMSC augmentation works on the principle of reverse preprocessing and generates new samples by transforming EMSC preprocessed data back to raw data with slight variations. The validity of the three augmentation methods was tested by training and evaluating the performance of CNN model on data generated using the different augmentation methods. Some of the recent studies have reported that the CNN model performs better on spectral data compared to linear models such as PLSR [9]. Therefore, in order to evaluate the performance and the effectiveness of using a CNN model, the results of CNN model were compared to the results of PLSR model that was trained and evaluated on the same data as CNN. Furthermore, to study the effect of increasing sample size by augmenting data, the technique of learning curves was used. Learning curves can act as a diagnostic tool for machine learning models. The data sample size was increased gradually and the learning curves were used to observe the performance of models with the increasing sample size. In addition the learning curves were also used to compare the performance of CNN on artificially generated data and raw data to observe if the augmentation methods lead to improvement in performance.

2 Theory and background

The theory background needed to understand the concepts and experiments used in this study spans multiple domains, and this section lays the foundation for topics and methods used later in the study.

2.1 Raman Spectroscopy

Raman spectroscopy allows a highly sensitive structural interpretation and identification of trace amounts of chemicals based on their unique vibrational fingerprints. It is considered as an inexpensive technique which is used for the analysis of biological samples[14, 15]. It measures the frequency shift of inelastic scattered light from a sample when the photon from the incident light strikes a molecule and produces a scattered photon [10, 11]. The technique is excellent for obtaining biotechnological samples as they can simultaneously measure broad chemical constituents present in the bio process by the detection of various functional groups [12, 16]. Generally, Raman measures the shift in energy of the outgoing photon. The difference in wavelength of the scattered light is dependent on the chemical composition of the molecules that are responsible for scattering. Magnitude of change in molecular polarization is proportional to the Raman scattering. Alteration in the molecular polarizability is the result of the molecular vibrations that displace the constituent atoms from equilibrium according to

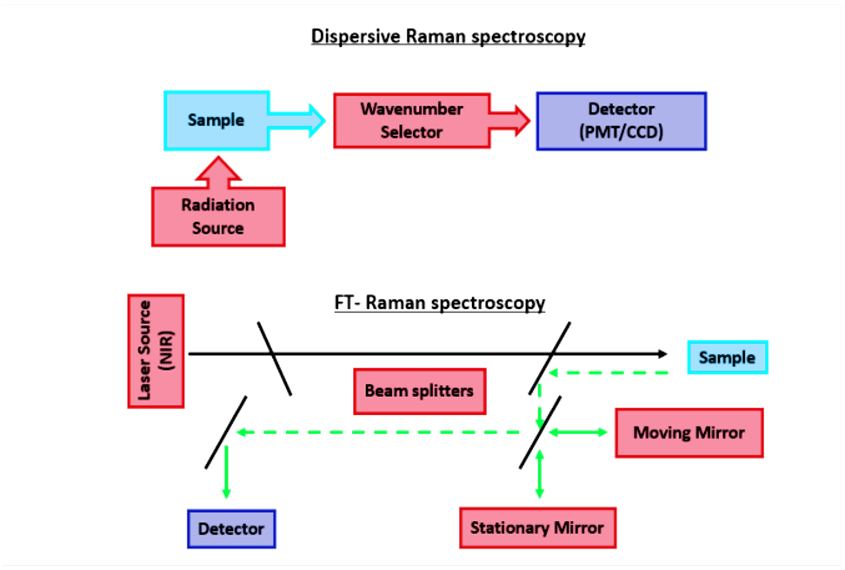


Figure 1: Two types of Raman spectrometers. The upper half of the figure shows the flowchart for collecting spectra using Dispersive Raman spectroscopy while the lower half shows the flowchart for FT Raman spectroscopy

the Raman selection rule which states that vibrations are only active if the molecular dipole moment changes during the vibration [13].

A Raman spectrometer is composed of a sample holder, detector, light source and monochromator. There are different types of spectrometer that use different methods for collecting spectrum samples. For example, Dispersive Raman and Fourier transform(FT) Raman are two types of Raman spectroscopy that use different spectrometers. Dispersive Raman uses a diffraction spectrometer to disperse the light scattered by the samples. Light is then detected by a multi channel detector and the wavelengths of light are the detected Raman spectra. FT Raman uses an interferometer that creates a path difference between the source and signal beams to create an interference pattern. From that interferogram, the Raman spectra are reconstructed. Difference between both types of Raman spectroscopy is visualized in Figure 1. The figure shows flowcharts to demonstrate the difference between Dispersive and FT Raman spectroscopy. The choice of technique is based on the sample that is under analysis as both types of spectroscopy show unique characteristics [17].

2.2 Linear Regression

Regression is a statistical method that is used to determine the mathematical relationship between a dependent variable and a series of one or more independent variables. Linear regression is a basic type of regression that estimates the relationship between an dependent variable($\mathbf{y}$) and a independent variable ($\mathbf{x}$) by using a linear model. The linear regression model is defined as:

$$\mathbf{y} = \mathbf{b_0} + \mathbf{b_1}\mathbf{x} + \mathbf{e} \tag{1}$$

In equation 1, $\mathbf{b_0}$ is the bias which predicts the value of $\mathbf{y}$ when $\mathbf{x}$ is 0 and $\mathbf{e}$ represents the error. $\mathbf{b_1}$ is the regression coefficient and this defines the expectation of change in $\mathbf{y}$ with the change in $\mathbf{x}$. Linear regression tries to find the best fit for the data by computing the regression coefficients($\mathbf{b_0}$, $\mathbf{b_1}$) that minimizes the loss function. Loss function is a measure of of difference between the predicted and actual value of the response(dependent variable). An example of loss function is the mean squared error(MSE) which is the average of the squared difference between the predicted and actual value of a variable. The mathematical representation of MSE is provided in equation 3 where $\mathbf{y_i}$ is the actual value and $\hat{\mathbf{y}}_\mathbf{i}$ is the predicted value. The measure of how well the linear model fits a certain data set

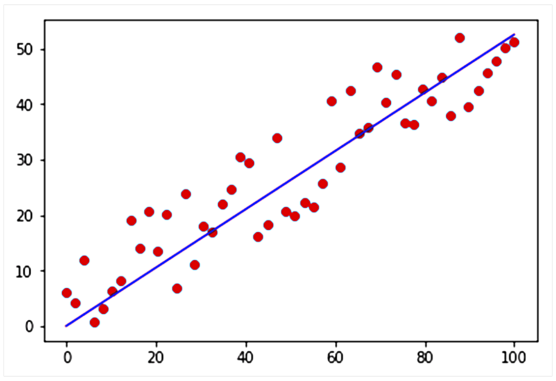


Figure 2: Graphical representation of linear regression model for a single dependent (y-axis) and independent variable (x-axis). The red dots represent the data points of the independent variable and the blue line represents a linear regression line. The regression line is adjusted to find best fit for data by minimizing the loss function

can be visualized using a regression line. Regression line is a graphical representation of the regression equation that represents the relations between the dependent and independent variables. Coefficients $\mathbf{b_0}$ and $\mathbf{b_1}$ in equation 1 represent the intercept and the slope respectively of the regression line. Linear regression tries to find the best fit for the regression line using a method called *ordinary least squares*. The ordinary least squares method tries to find the best set of regression coefficients for the model by minimizing the sum of squared distances between the actual and predicted values of the response. A linear regression model for a single dependent and independent variable is visualized in Figure 2.

$$\mathbf{MSE} = \frac{\sum_{\mathbf{i}}(\mathbf{y_i} - \hat{\mathbf{y}}_{\mathbf{i}})^{\mathbf{2}}}{\mathbf{n}} \tag{2}$$

The linear regression model can be extended to take in to account multiple independent variables instead of just one. This type of regression is called multiple linear regression and this model is defined as:

$$\mathbf{y} = \mathbf{b_0} + \mathbf{b_1 x_1} + \mathbf{b_2 x_2} ... + \mathbf{b_n x_n} + \mathbf{e} \tag{3}$$

where $\mathbf{b_{0...n}}$ are regression coefficients and $\mathbf{x_{1...n}}$ are independent variables.

A limitation of linear models is that they only work when the predictors(independent variables) are independent because if there are correlations among the predictors then the linear equation will not be meaningful anymore. When the independent variables are highly correlated to each other then this phenomena is called multicollinearity. It is problem because it undermines the statistical significance of a variable. A multivariate technique that is used when the data suffers from multicollinearity is partial least squares.

2.3 Partial Least Squares Regression (PLSR)

Partial Least Squares(PLS) method which is also sometimes referred as projection to latent structures[18] was first developed in the late 1960s to the 1980s by a Swedish economist Herman Wold[19]. However, it became popular in chemometrics due to the works of Herman's son Svante [20]. PLS relate the information present in the dependent variables(targets) and independent variables(predictors). It solves the problem of multicollinearity by calculating regression coefficients that maximizes the correlation between target and the predictors and explain as much covariance between them. Partial least squares regression(PLSR) makes prediction by extracting a set orthogonal factors with predictive power known as PLS components. Each PLS component is uncorrelated with each other. The PLS components are numbered in order of the maximum correlation with the target.

In addition to multicollinearity, an additional scenario when the linear models fail is when the number of predictors(p) are greater than the number of observations(n). In this case, there is no longer a unique least squares coefficient estimate and as a result the model fits the training data very well but fails to generalize for the new observations of the future. A solution to overcome the problem of multicollinearity and too many predictor variables is *feature selection* which is defined as the process of reducing the number of predictors to improve model performance. However, feature selection is not efficient when there are hundreds of predictors. Another approach of to tackle the defined problems is to use multivariate methods like PCA[21] and PLSR. These methods work by extracting latent variables that are the result of linearly combining existing variables in the data set. These latent variables are orthogonal to each other which solves the issue of collinearity. Furthermore, the extracted latent variables are then used in place of the original variables and as a result the number of variables are reduced significantly. PCA is a unsupervised method that extracts the latent variables based on the variation present in the data($\mathbf{X}$). With PCA, there is no guarantee that the variables chosen are relevant to the target because the chosen components are obtained to explain the variance in $\mathbf{X}$. PLSR is a supervised method that works in a similar way to PCA but instead it finds latent variables that are also relevant to the target($\mathbf{Y}$). Instead of just the $\mathbf{X}$, PLSR performs the decomposition of both $\mathbf{X}$ and $\mathbf{Y}$ simultaneously. The decomposition is done with a constraint such that the selected component explains as much covariance between $\mathbf{X}$ and $\mathbf{Y}$. The decomposition is similar to PCA but it is also followed by a regression step in which the decomposition of $\mathbf{X}$ is used to predict $\mathbf{Y}$.

The predictors($\mathbf{X}$) are decomposed as:

$$\mathbf{X} = \mathbf{TP^T} + \mathbf{E} \tag{4}$$

Similarly $\mathbf{Y}$ is decomposed as

$$\mathbf{Y} = \mathbf{TQ^T} + \mathbf{F} \tag{5}$$

In Equation[4] and Equation[5], $\mathbf{P}$ and $\mathbf{Q}$ are called the loading matrix and $\mathbf{T}$ is called the scores matrix. $\mathbf{E}$ and $\mathbf{F}$ are the error terms.

Each sample in a particular data set appears as a point in a space defined by position along an axis. After applying PLSR, the samples will now appear as point in a space defined by its position in the newly discovered PLS component axis. The new coordinates are the scores returned by PLSR method. A matrix that contains scores of all samples is called scores matrix. Although the scores show the location of the samples in the newly discovered PLS component axis but it does not show the position of that axis. In order to reconstruct the original data matrix from the PLS components, both the location of the sample relative to the PLS component axis and the location of the PLS component axis relative to the original axis must be known. The location of the PLS component axis relative to the original axis is defined by loadings.

2.4 Artificial Neural Network

Artificial neural networks(ANN) are models that belong to the domain of deep learning. ANNs use the processing of the brain as a base to develop algorithms that can be used to model complex patterns and predictions problems. Although these networks are a very simplistic representation of the workings of a human brain but still they possess the ability to solve complex real world problems.

2.4.1 History

The human brain works in extraordinary and complex ways. With the progression in knowledge and the evolution of science it was only a matter of time when human beings will try to mimic the working processes of the human brain. The task became possible with the advancements in the modern electronics. Human brain consists of a network of interconnected neurons that are used to process information coming in and out of the brain. The architecture and processes in the human brain are extremely complex and are not easy to model. Artificial Neural Networks (ANNs) are computational models that are inspired by the human brain. It is important to note that the architecture of an ANN

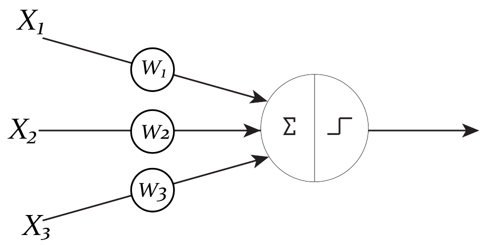


Figure 3: A visual representation of the Threshold Logic Unit. $\mathbf{X_{1,2,3}}$ represents the input, $\mathbf{w_{1,2,3}}$ represents the weights, $\sum$ represents the weighted sum of the inputs and the last symbol represents the unit step function that decides the output value

is massive simplification of the biological brain. Similar to the human brain, ANNs also consists of an interconnected network of processing units that learn from the information coming into them called neurons. Neurons are core processing units of the neural network. The first step in the invention of ANNs dates back to 1943 when Warren McCullock, a neurophysiologist, and Walter Pitts, a mathematician, published a research paper on how the neurons work[22]. They also modelled a simple neural network using electrical circuits. After their research, a book was published by Donald Hebb in 1949, named *The organization of behavior* [23]. This book focused on a concept stating that the biological neural pathways that are used by the neurons for communication are strengthened each time they are used. The concept was named *Hebbian Learning* in honor of Donald Hebb and it is an important concept used for the training of ANNs.

2.4.2 The Perceptron

Frank Rosenblatt, a neuro biologist of Cornell came up with a neural network model in 1958 called Perceptron [24]. Rosenblatt called the perceptron a "hypothetical nervous system", indicating that the perceptron was designed to represent some properties of intelligent biological systems. The architecture of the perceptron consists of a single neuron and it is based on threshold logic unit which was proposed by Warren McCulloch and Walter Pitts [24]. A visual representation of a threshold logic unit is provided in Figure 3. The figure shows the inputs coming into the perceptron model and the weights assigned to each input represented by **X** and **w** respectively. Weights are model parameters that are used to control the strength of the inputs coming into the model. Training of the perceptron involves the adjustments of weights that are assigned to each input iteratively. Initially, the weights are randomly generated and then they are continuously modified such that they gradually start reflecting the mathematical relation between the input and the output. Weight modification is done by calculating the error. The error is the difference between the predicted and actual output. The weights are updated as follows:

$$\mathbf{w_{k+1}} = \mathbf{w_k} + \alpha(\mathbf{y} - \hat{\mathbf{y}})\mathbf{x} \tag{6}$$

where $\mathbf{k}$ represents the iteration, $\mathbf{w_k}$ and $\mathbf{w_{k+1}}$ represents the weight during the current iteration and the updated new weight for the next iteration respectively. α is the learning rate coefficient and it defines the rate at which the weights will be updated. $\mathbf{y}$ and $\hat{\mathbf{y}}$ are the actual and predicted outputs respectively. $\mathbf{x}$ is the input coming coming into the neuron.

The perceptron works by computing a output by calculating weighted sum of the input data and adding a bias(intercept of the fitted line of model). After computing the output, a step function is applied which decides the output. The step function could be a simple Heaviside step function shown in Equation 7. Development of the perceptron is the foundation and an important building block in the development of ANNs.

$$\mathbf{Heaviside}(\mathbf{x}) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \tag{7}$$

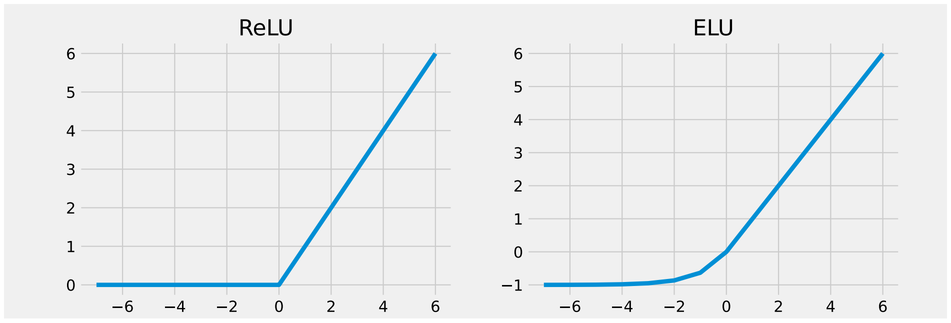


Figure 4: ReLu activation function and ELU activation function with ELU constant 1. ReLu(left) maps all negative inputs to zero while ELU(right) smooths the output curve towards the constant value

2.4.3 Deep Neural Network

Multiple perceptron stacked together forms a multi-layer neural network. Each layers in the multi-layered neural network have the ability to contain multiple neurons that are interconnected with each other and organized in a hierarchical(layered) manner. These neurons are responsible for passing a signal to other neurons based on the received input. There are different levels layers present in a multi-layer neural network. The different types of layers include the input layer, output layer and the hidden layers. Input layer is used to take in the input and the output layer provides the output that depends on the nature of problem. For example if the task is a regression task then the output of the network will be a single value. Hidden layers are present between the input layer and the output layer and are used to perform most the computations done by neural network. Neural networks with two or more hidden layers are called a deep neural networks(DNN). A visual representation of a DNN is provided in Figure 5. The DNN provided in the figure contains a input layer, 2 hidden layers and a output layer. Each neuron in the DNN is connected to the neurons of the next layer via channels. Weights are assigned to each of these channels. Data comes into the network through neurons of the input layer which computes a weighted sum based on the input and passes it to the neuron of the hidden layers. The same process is repeated in the next layers until an output is provided via the last output layer. The output computed by the neurons that is transferred to the neuron in the next layer is done by an *activation function*. Output of the activation function is based on the calculated weighted sum of the input. An activation is a non linear transformation calculated weighted sum of the input for each neuron. There are several different types of activation functions. Two types of the activation functions that are used in this study are *Relu* and *ELU*.

ReLu stands for rectified linear unit and it is defined as $\mathbf{f(x) = max(0, x)}$. It is easy to compute and the output of the ReLu activation function ranges between 0 to positive infinity. This means that ReLu produces absolute values for positive inputs and 0 for negative inputs. ELU stands for exponential linear unit and it is an alternative to ReLU. ELU behaves similar to ReLU for positive inputs however, it differs for negative inputs. Instead of mapping all the negative inputs to zero, ELU smooths the output curve towards a constant. Higher value of the constant results in negative values with higher coefficients. The ELU activation function defined as:

$$\mathbf{R(z)} = \begin{cases} z & \text{if } z > 0 \\ \varrho(e^z - 1) & \text{if } z \leq 0 \end{cases} \tag{8}$$

where ϱ represents the ELU constant. A visual representation of both ReLu and ELU is provided in Figure 4. The figure shows the difference between both activation functions for handling negative inputs.

Multi-layer neural network can also be called a feed forward network as the data in these networks flows in a sequence starting from the input layer and reaching the output layer. Details on the training process of the DNN can be found in section[2.5]

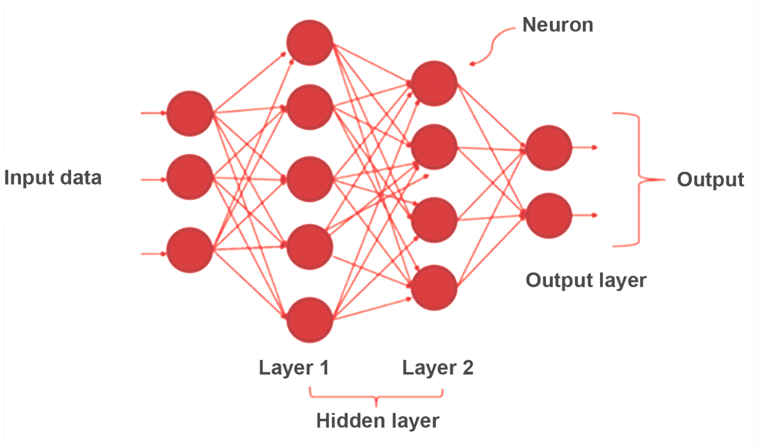


Figure 5: a deep neural network with 2 hidden layers

2.5 Training an Artificial Neural Network

The training of an ANN is an iterative process and an iteration of training process of the neural network is called an *epoch*. Each epoch is divided into two steps.

- The first step is the *forward pass* where the input is traversed through all the neurons from the first to the last layer, an output is calculated and a prediction is made.
- The second step is the *backward pass*. In the backward pass, the error is calculated and propagated backwards from the last layer to the first layers and weights are updated.

The number of weights in a deep neural network is much larger compared to a linear machine learning model. If the training data contains a large number of samples as well then the training process can become computationally and memory expensive. Therefore, a subset of the training data set is used for each training iteration of an ANN to make the training process efficient. The subset of data used for training is called *batch* or *mini-batch*.

Error in prediction is a useful measure for updating the weights because the update in weights is proportional to the minimization of a loss function. The loss function is used along with an algorithm called **optimization algorithm**. The target of the optimization algorithm is to find the local minima of the loss function. An optimization algorithms defines the direction and magnitude of the weight updates for a neural network. There are several types of optimization function and the most basic type is the *Gradient Descent*. The gradient is the generalization of the derivative to the loss functions. Gradient descent algorithm is a first order optimization algorithm. The first order means that the gradient descent tries to reach the local minima by calculating the first order derivative of the loss function. Mathematical representation of the Gradient descent algorithm is shown in equation 9. If the loss function is considered as a bowl with a base and two high ascended ends the the calculated gradient is a vector which gives the direction in which loss function has the steepest ascent. Therefore, to reach the local minima(point of steepest descent) the gradient is subtracted from the current weight resulting the weight update to be in the direction of the local minima.

$$\mathbf{w} := \mathbf{w} - \alpha \frac{\partial}{\partial \theta} \mathbf{L}(\mathbf{y}, \hat{\mathbf{y}}) \tag{9}$$

where $\mathbf{w}$ is the weight, α is the learning rate and $\frac{\partial}{\partial \theta} \mathbf{L}(\mathbf{y}, \hat{\mathbf{y}})$ is the first order derivative of loss function. $\mathbf{y}$ and $\hat{\mathbf{y}}$ represent the actual and predicted values of the response respectively. Gradient descent algorithm is simple and easy to implement however it is not the best choice when the size of the data set is large. Gradient descent algorithm updates the weights after calculating the gradient on the whole data set. So, if the data set is large than it may take a lot of time to converge to the local minima. Also, it will not be memory efficient if it calculates the gradient of the whole data set when the size of data set is large. A variant of the gradient descent algorithm is the stochastic gradient descent. It can cope with the mentioned problem for the gradient descent by frequently updating the weights after each training sample. Therefore, for stochastic gradient descent, equation 9 becomes:

$$\mathbf{w} := \mathbf{w} - \alpha \frac{\partial}{\partial \theta} \mathbf{L}(\mathbf{y}, \hat{\mathbf{y}}) \mathbf{x_j} \tag{10}$$

where $\mathbf{x_j}$ represents the training sample.

Adaptive Momentum Estimation(Adam) [25] is another type of optimization function that works by computing adaptive learning rates for each weight. Adam works in a different way then the stochastic gradient descent. The learning rate remains same for all updates of the stochastic gradient descent algorithm however the Adam optimizer computes adaptive learning rates for different weights from estimates of first and second moments of the gradients. Adam is derived by combining advantages of two extensions of stochastic gradient descent i.e. *Adaptive Gradient Algorithm (AdaGrad)* and *Root Mean Square Propagation (RMSProp)*. Adagrad and RMSProp both maintains per parameter learning rate that improves performance of the model. In addition to maintaining a per parameter learning rate RMSProp also adapts the learning rate based on the average of recent magnitudes of the gradients for the weight. Adam combines the properties of both Adagrad and RMSProp and in addition to adapting the learning rate based on the average of gradient updates it also makes use of the variance of the gradients. Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters $\beta 1$ and $\beta 2$ control the decay rates of these moving averages. The averages of the past and past squared gradients mt and vt are computed as:

$$\mathbf{m_t} = \beta_1 \mathbf{m_{t-1}} + (\mathbf{1} - \beta_1)\mathbf{g_t} \tag{11}$$

$$\mathbf{v_t} = \beta_2 \mathbf{v_{t-1}} + (\mathbf{1} - \beta_2)\mathbf{g_t^2} \tag{12}$$

mt and **vt** are estimates of the mean and the second moment variance of the gradients respectively. $\mathbf{g_t}$ represents the gradient at a time $\mathbf{t}$ for a specific weight.

2.5.1 Back-propagation Algorithm

An important algorithm that is used for training neural networks is the back-propagation algorithm. Back-propagation is used to calculate the gradient for each weight in the network model during the backward pass of an epoch. The gradient is then used by an optimization algorithm to update the model weights. The process of updating weights and finding the local minima for a loss function for a linear model is simple as they do not have high dimensional feature space for the weights. In this case, the landscape can be imagined as a big bowl visualized in figure 6. However, the process of updating the weight and to find the local minima for a multi layer neural network is challenging as it has multiple interconnected neurons which result in a large number of weights that are in a high dimensional feature space. The high dimensional feature space of the deep neural networks contain multiple local minimums for the loss function. Therefore, the target in this case is to find the global minimum of the loss function. However, the abstract high dimensional feature space of neural network can have hundreds and thousands of dimensions and there is no way to know that a founded local minima is the global minimum or even close to global minimum. Therefore, the best the network can do is to find the best local minima it can find.

Back-propagation is an efficient algorithm that works on the principle of chain rule and it is one of the most widely used algorithm when training an ANN [26]. The chain rule states that the derivative of $\mathbf{f}(\mathbf{g}(\mathbf{x}))$ is $\mathbf{f}'(\mathbf{g}(\mathbf{x}))\mathbf{g}'(\mathbf{x})$. In simple words, it helps to differentiate composite functions To understand the working of a back-propagation algorithm, consider a multi layer neural network that went through the feed forward propagation and computed an output. After computing the output, the error is propagated back from right to left i.e. from the output layer to the hidden layers. Following this step, the error of the hidden layer is calculated. After the calculation of errors for the output and the hidden layers, partial derivative of the loss function for each layer is computed. The loss function is applied to every node in each layer and the calculated loss function is accumulated for each neuron in each layer. Imagine a deep neural network with a large number of layers, then in order to calculate

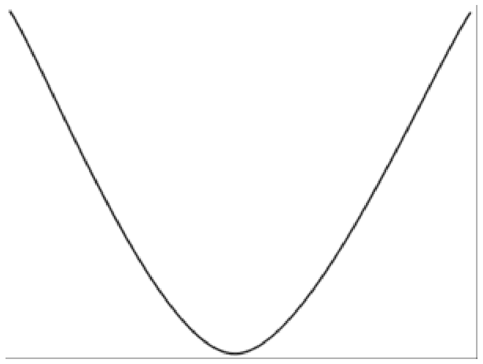


Figure 6: Representation of Gradient Descent

the weight of a neuron present the i^{th} layer the gradient of all neurons present in the $i+1...n$ layers need to be calculated and calculating the derivative with that many dimensions would be highly expensive and inefficient. Based on the principle of chain rule, the back-propagation algorithm provides a computationally efficient approach for computing the partial derivative of loss function for deep hidden layers of the network. Finally, the weights of the layers are updated based on the computed loss function.

2.6 Techniques used for training ANNs

Designing a neural network can be a challenging task. There is no general principle for this task as it solely depends upon the problem in question. In addition to defining the number of layers, number of neurons in a layer, activation function and optimization algorithm, there are other techniques as well that can be applied in the network to build more robust and generalized model. Some of the techniques are elaborated below:

2.6.1 Regularization

It is a difficult task to build a neural network that generalizes well on new and unseen data. A network cannot learn a problem when it has too little capacity for that problem. Similarly, the network can learn a problem too well when it is too complex. When the network is unable to learn a problem because of not enough capacity then the concept is called underfitting and when it learns the problem too well then the concept is called overfitting. In both cases, the network is not generalized and will not perform well on new unseen data.

The problem of underfitting can be addressed by increasing the capacity of a network. Increasing the capacity means adding more neurons to a layer or even adding more hidden layer. However, overfitting is an issue that is not straightforward to solve and it can requires applying some additional techniques to the network. Regularization methods can be used in a neural network to prevent overfitting. Regularization works by introducing a penalty term to the model which ensures small weights. The penalty term is added to the loss function which effects the updates of the weight values. L1 and L2 regularization are two types of regularization that ensures small weights by applying penalty term. Another type of regularization technique that is specifically designed for ANNs is the dropout.

Dropout Dropout[27] is another technique that is used in ANNs to prevent over fitting. The idea of dropouts was presented by Srivastava et al. in 2012 by excluding subsets of features in each iteration of the training process[27]. Preceding the concept of dropouts, regularization techniques were used to tackle the overfitting issue in neural networks. However, the dropout technique has proven to be a better choice when training a neural network. In their research paper in 2013, Wager et al. tested that dropout regularization was better than L2-regularization for learning weights for features[28].

Dropout is a method where a set of random neurons are dropped during the training process of a neural network. The visual illustration of dropouts is provided in figure 7 which shows two neurons in the fully connected(FC) layer dropped out. The neurons are dropped arbitrarily during each epoch. The dropped neurons do not contribute in the training process at all. This means they do not produce

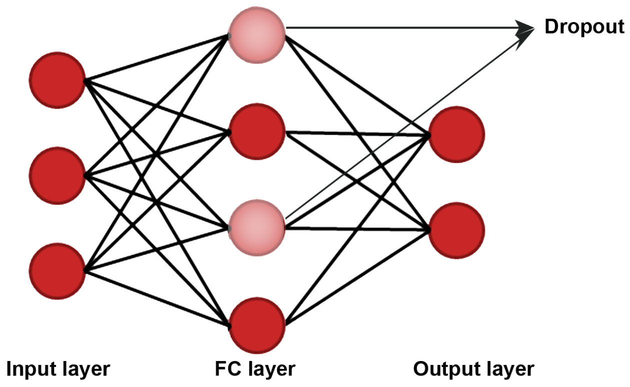


Figure 7: Dropout illustration: two random neurons are dropped out(indicated with light colors) from the training process in the FC layer. The dropout neurons will not be used for training.

activation during the forward pass and no weight update is applied during the backward pass. When the randomly selected set of neurons are dropped out, the remaining neurons will have to make the predictions which will require them to fill in for the other neurons as well. This is believed to bring about various independent internal representations being learned by the network and makes the model more robust. Dropouts are only applied in the training process. Dropout requires indicating dropout rates when it is implemented in a network. The indicated dropout rate is the probability of dropping a neuron.

2.6.2 Batch Normalization

Neural networks can have tens of layers. Training neural networks with a large number of layers is a challenging process. A reason for this challenge is that the distributions of the inputs to deeper layers may change after every mini-batch. The weights of a neural network are updated layer by layer from the last output layer to input layer. The update of a layer's weights is based on the assumption that all the weights in the prior layers are fixed. In practice, all the layers are updated simultaneously[29]. Since all layers are updated simultaneously, the change in weights will result in the output distribution of the prior layers to change. Based on the assumption that the state of prior layers do not change while updating a particular layer, the update procedure is chasing a moving target. Now, in order for a model to train on a given data, it will require lower learning rates so that the state of layers do not change significantly during every epoch. This makes it hard for a model to train on saturating non linearities and it also makes the training process slow[29]. The change in the distributions of the neurons in the layers can be referred as *internal covariate shift*[30].

Batch normalization is a method that is used in deep neural networks. This methods standardizes the outputs of layers before it is passed as an input to the next layer. Standardizing the output means that the distributions of the outputs of a layer will not change during the weight update procedure. So, using this method is a step towards achieving fixed distributions that solves the problems caused by the internal covariate shift[30]. This method will stabilizes the training process and will make it efficient and optimized[29].

2.6.3 Reduce Learning Rate on Plateau

Reduce learning rate on plateau(RDLR) is a technique that reduces the learning rate of the optimizer when the model performance does not improve for a specified number of epoch in the training process. When the learning rate is high, the weight update is done in big steps. This can be beneficial in first few epochs. But, if the learning rate is large throughout the training process, the big steps in updating the weights can have a negative effect. If the model starts going in to the wrong direction and the update in weights is large then it can become hard for the model to recover and this can result in inconsistency in performance. However, if the learning rate is small then the updates in weights is small and the model may be able to recover even if it starts following the wrong direction. So, it may be beneficial to use RDLR during training.

2.6.4 Early Stopping

Early stopping(ES) is a type of callback function that stops the training process if the performance of the model does not improve. If the model is not getting better then there is no need to train the model any longer and the training should be stopped to save time and resources. Early stopping terminates the training process if the performance of the model is not improved after a specified number of epoch.

2.7 Convolutional Neural Network

The type of ANNs depends upon the architecture of the connections between the neurons and the depth of layers. Different arrangements of inter-connected neurons and the number of layers yield different network architectural options. For example a type of ANN where each neuron of one layers is connected to all the neurons of the subsequent layer is referred to as a fully connected feed forward networks. There are various other architectural types of ANNs but for this study we will focus on the *convolutional neural network* also referred to as CNN.

Fully connected neural networks can learn global patterns from the input space because of their architecture in which each neuron is connected to all neurons in the subsequent layer. This architecture allows the model to learn complex patterns as a whole in the input but it fails to provide the ability to detect and learn local correlations that can occur in the input space at any random position [31], [35]. Theoretically, a fully connected network could learn to identify a local pattern in any position of the input space by simply adding enough neurons. However, this would likely lead to multiple neurons sharing the same weight patterns but located at various locations in the network, so they could detect the pattern at different locations in the input space [31]. This can in some cases leads to an inefficient network architecture, which increases computational cost and requires large data sets that includes samples of the pattern in all possible locations.

In 1981, a new layered hierarchical architecture of ANN was created by Kunihiko Fukushima which he called the *neocognitron* [32]. In this architecture each neuron is connected to the neurons of a small patch in the previous layer[32]. The receptive field of each layers is increased by gradually decreasing the size of spatial dimension in the deeper layers so eventually the neurons of the final layers of the neocognitron have a receptive field that covers the whole input space. It is responsive to any specific pattern in the input space irrespective of its position or size[32]. Neocognitron is recognized as the origin of deep CNNs.

Convolutions are a building block of a CNN. The term convolution refers to a mathematical combination of two functions to produce a third function. Basically, convolution means to merge two sets of information. In terms of machine learning, convolution is the application of a filter to an input to get an output. If the filter is applied repeatedly over the input then the map of the outputs is called the feature map of the inputs. This feature map indicates the locations and strengths of an input feature. The process of convolution is graphically illustrated in Figure[8]. A combination of convolutional layers is used in the neocognitron for feature extraction and down sampling(reducing the size and dimensionality of the input) of the input. This architectural design later paved the way for convolutional neural networks [31]. The CNNs of today are also based on the basic architecture of the neocognitron. CNNs are widely used in the field of image classification, image segmentation and computer vision.

A convolutional neural network can consist of different types of layers. A convolutional layer is a type of layer specifically used in CNNs that that works on the principle of convolutions. A convolutional layer consists of a filter that is applied to the whole input space with a stride. Stride defines the step size of the filter. For example if the stride is one then the filter moves one step over the input space. Similarly, if the stride in two then the filter moves two steps. This striding technique is useful when the desire is to reduce the output dimension of the layer as a larger stride results in smaller output dimensions. The size and dimension of the filter of a convolution layer can vary and depends upon the type of convolutional network. For example the dimension of filter for a 1D convolutional network is also one because the input space has one dimension. In case of a 1D convolution, a filter of size 2

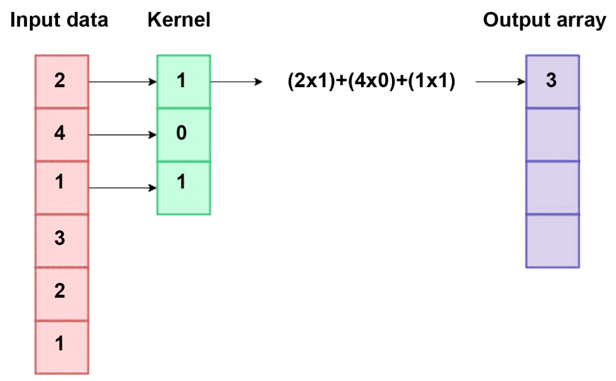


Figure 8: Illustration of convolution process

means that each pixel in the output is created by looking at a grid of 2 pixels. By reusing the same weights for the filters kernel in a sliding window across the input, a filter produces a feature map of a particular feature or pattern that it has specialized in[33].

Another type of layer that is used in CNNs is the pooling layer. The pooling layers are used for the reduction of size of output dimension[34]. The pooling has no trainable weights and its purpose is to only reduce the size of input space going into the next layer. The pooling layer results in a significant reduction of parameters because the pooling in these layers is performed with a stride equal to its size. There are different types of pooling layers. For example max pooling return the highest value of the region it passes over . A max pooling layer of size 2 will reduce the number of features by a factor of 4 and will also tend to keep the strongest activation from the input coming into it. Reducing the size of the input space may be beneficial to focus on the important parts of the input but sometimes the requirement may be to preserve the dimensions as it is. For example if the network is deep then too much dimension reduction may result in the network not capturing any meaningful features. Padding is a technique used in CNNs to prevent shrinkage of the output dimensions. It works by adding zeros at the edges of the input feature space. There are different types of padding that are used for different purpose. For example same padding is used with stride one and it evenly adds zeros to the edges of the feature space which results in no dimension reduction of the output space. A simple 1 dimensional CNN is visualized in Figure[9].

Fully Connected(FC) layers in a neural networks are those layers where all the neurons from one layer are connected to every neuron of the next layer. FC layers are often used in CNNs along with convolutional layers. The output from the convolutional layers represents high-level features in the data. While that output could be flattened and connected to the output layer, adding a fully-connected layer enables the learning non-linear combinations of these features. Essentially the convolutional layers are providing a meaningful, low-dimensional, and somewhat invariant feature space, and the fully-connected layer is learning a (possibly non-linear) function in that space. The difference between a FC layer and a convolution layer can be visualized in Figure[10].

2.8 Preprocessing

Preprocessing is an important step when training ML models. Real world data is generally inconsistent, noisy and even incomplete sometimes. Raw data is sometimes in such a condition that it becomes impossible to use it for training ML models. Preprocessing is a technique that is used to transform raw data into more understandable, useful and efficient format. Preprocessing includes steps like data cleaning, data reduction and data transformation.

In vibrational spectroscopy, the measured spectra of a compound can be affected by numerous factors other than the chemical components that are of interest. The differences in the spectra are

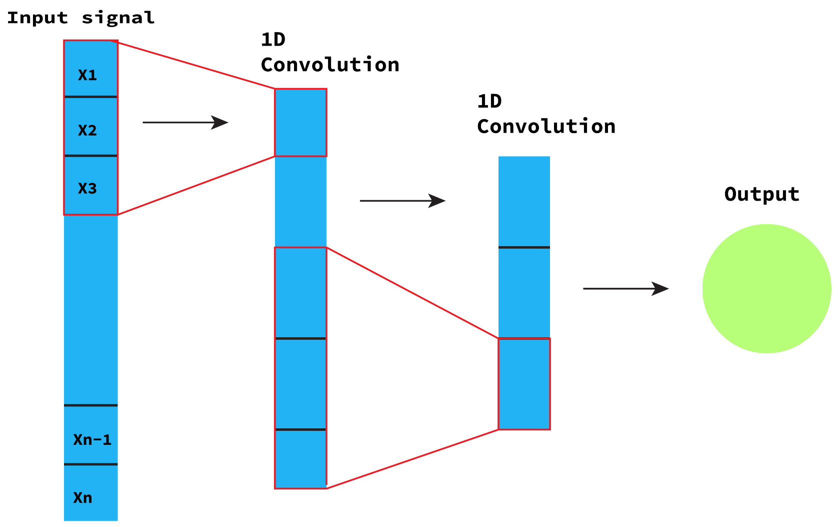


Figure 9: Simple 1D CNN with two convolution layers

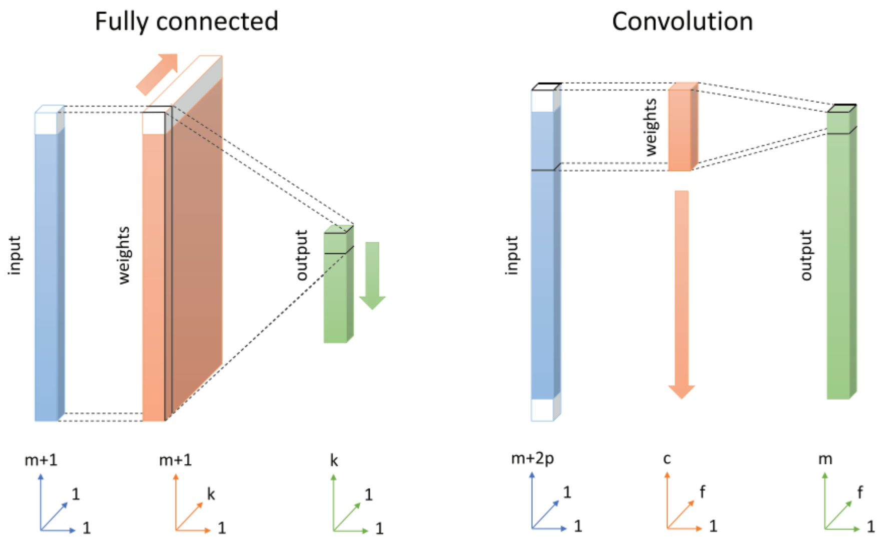


Figure 10: Difference between a Fully connected(left) and 1D convolutional(right) ANN. The Fully connected ANN consists of k neurons. The illustrated convolutional network contains f number of filters and each filter includes c weights. The white part part in the Fully connected network represents the bias whereas the white part in the convolutional network represents zero padding. The image is reprinted with permission from Helin, R, Indahl, UG, Tomic, O, Liland, KH. On the possible benefits of deep learning for spectral preprocessing. Journal of Chemometrics. 2022; 36(2)

caused by different factors such as noise and systematic errors like non linear instrument responses and effects induce from undesired chemical and physical variations. This phenomena, in which the response of a sample is affected by other factors cause unnecessary challenges during analysis. Preprocessing of spectroscopic data is vital for correcting these unwanted effects.

The choice of the preprocessing method used influences the performance of the predictive model. There are several popular preprocessing techniques for spectroscopic data like Savitzky–Golay filtering, Standard normal variate. One technique for preprocessing spectroscopic data is the extended multiplicative scatter correction (EMSC).

Extended Multiplicative scatter Correction (EMSC) : EMSC[37] has proved to be a reliable method to correct for additive baseline, multiplicative scaling and interference effects. EMSC can also be used for preprocessing Raman spectroscopy data as it is adapted to the baseline variations caused due to the fluorescence effects.

EMSC is an extension model of the *Multiplicative scatter correction(MSC)*[36]. It is an adaptive and efficient model based preprocessing method. The methods works by determining a least squares fit of a single spectra against a few profile spectra. Since it is a model based preprocessing method, model parameters are also saved and returned in addition to correcting the spectra with the desired model. The returned model parameters can provide valuable information regarding the analyzed spectral samples.

MSC was derived based on the Beer-Lambert Law law which states that an absorbance spectrum is directly proportional to the effective optical path length. Here the optical path length can be taken as the thickness of a compound. According to Beer-Lambert law the absorbance for a transparent sample containing a single light absorbing chemical substance is given by:

$$\mathbf{A}(\bar{\mathbf{v}}) = \mathbf{k}(\bar{\mathbf{v}})\mathbf{c}.\mathbf{b} \tag{13}$$

where $\mathbf{A}(\bar{\mathbf{v}})$ is the absorbance at wavenumbers $\bar{\mathbf{v}}$, $\mathbf{k}(\bar{\mathbf{v}})$ is the characteristic absorptivity of a component at a wave number $\bar{\mathbf{v}}$, $\mathbf{b}$ is the optical path length and $\mathbf{c}$ is the concentration of light absorbing chemical species in the sample. The variations in the spectra that are caused by optical path length are generally referred to as multiplicative variations.

MSC extends the Beer-Lambert model through an additive effect resulting in the model:

$$\mathbf{A}(\bar{\mathbf{v}}) = \mathbf{a} + \bar{\mathbf{x}}(\bar{\mathbf{v}})\mathbf{b} + \mathbf{e}(\bar{\mathbf{v}}) \tag{14}$$

where $\bar{\mathbf{a}}$ is an additive baseline constant, $\bar{\mathbf{x}}(\bar{\mathbf{v}})$ is the mean spectrum (or another chosen reference) and b is a multiplicative constant. Finally, $\mathbf{e}(\bar{\mathbf{v}})$ is the residual vector containing the interesting chemical differences between the samples, i.e

$$\mathbf{e}(\bar{\mathbf{v}}) = \mathbf{b}.\sum_{\mathbf{j}=\mathbf{1}}^{\mathbf{J}} \mathbf{c}_{\mathbf{j}} \boldsymbol{\Delta}\mathbf{k}_{\mathbf{j}}(\bar{\mathbf{v}}) \tag{15}$$

where $\mathbf{c}_{\mathbf{j}}$ are concentrations of the species $\mathbf{k}_{\mathbf{j}}$ and $\boldsymbol{\Delta}\mathbf{k}_{\mathbf{j}}(\bar{\mathbf{v}})$ are the specie's profile deviations from a mean profile or other reference.

For EMSC, equation 14 is extended with polynomial baseline profiles to handle more complex baseline changes from sample to sample:

$$\mathbf{A}(\bar{\mathbf{v}}) = \mathbf{a} + \bar{\mathbf{x}}.\mathbf{b} + \mathbf{d}_{\mathbf{1}}(\bar{\mathbf{v}}) + \mathbf{d}_{\mathbf{2}}(\bar{\mathbf{v}^{\mathbf{2}}}) + ... + \mathbf{d}_{\mathbf{n}}(\bar{\mathbf{v}^{\mathbf{n}}}) + \mathbf{e}(\bar{\mathbf{v}}) \tag{16}$$

where $\bar{\mathbf{v}^{\mathbf{j}}}$ are polynomials of the wavenumbers with corresponding constants $\mathbf{d}_{\mathbf{j}}$. Different EMSC models can be derived from equation 16. The EMSC model where the polynomial in equation 14 is extended up to the quadratic term is often referred to as the basic EMSC model. A further extension of the basic EMSC model above quadratic terms is often denoted polynomial EMSC. The unknown parameters are estimated using an ordinary or weighted least squares estimation, and the spectra are corrected according to equation 17:

$$\mathbf{A}_{\mathbf{corr}}(\bar{\mathbf{v}}) = (\mathbf{A}(\bar{\mathbf{v}}) - \mathbf{a} - \mathbf{d}_{\mathbf{1}}(\bar{\mathbf{v}}) - \mathbf{d}_{\mathbf{2}}(\bar{\mathbf{v}^{\mathbf{2}}}) - ... - \mathbf{d}_{\mathbf{n}}(\bar{\mathbf{v}^{\mathbf{n}}}))/\mathbf{b} \tag{17}$$

2.9 Model Evaluation

After training a machine learning model the performance of a model should be evaluated to interpret its strength and weakness. Furthermore, the performance model should also be validated on independent unseen data to determine its generalizability. This section provides details about the evaluation and validation methods used in this study.

2.9.1 Evaluation Metrices

R squared(R2) and *Root mean squared error(RMSE)* are the two evaluation metrices used to evaluate the performance of the models after training in this study. R2 also known as the coefficient of determination is a statistical measure for a regression model that tells the proportion of variance explained in a dependent variable by an independent variable. In other words, R2 tells that how well the data fits the regression model. The value of the R2 varies between 0 and 1 with 0 being the worst and 1 being the best value. It is important to note that R2 is not the measure of correctness of a model. R2 can be interpreted in terms of percentages, e.g a 0.6 value indicates that 60 percent of the data fit the regression model. A higher value indicates a better fit. R2 can be calculated by:

$$\mathbf{R2} = \mathbf{1} - \frac{\mathbf{SS_{regression}}}{\mathbf{SS_{total}}} \tag{18}$$

where $\mathbf{SS_{regression}}$ is the sum of squared of regression error which is the sum of squared differences between the actual and the predicted value. If $\mathbf{y}$ is the target and $\mathbf{\hat{y}}$ is the predicted target then $\mathbf{SS_{regression}}$ can be calculated as $\sum_\mathbf{i}(\mathbf{y_i} - \mathbf{\hat{y}_i})^\mathbf{2}$. $\mathbf{SS_{total}}$ is the sum of squared of total error that is the deviation of data points away from the mean value. It can calculated as $\sum_\mathbf{i}(\mathbf{y_i} - \mathbf{\bar{y}})^\mathbf{2}$.

Root mean squared error(RMSE) is the modified version of MSE (defined in equation 2). RMSE tells the average distance between the predicted values and the original data points. Higher value of RMSE indicates large error and a lower value indicates less error. RMSE is calculated by taking the square root of MSE:

$$\mathbf{RMSE} = \sqrt{\frac{\sum_\mathbf{i}(\mathbf{y_i} - \mathbf{\hat{y}_i})^\mathbf{2}}{\mathbf{n}}} \tag{19}$$

where $\mathbf{y_i}$ is the observed data point, $\mathbf{\hat{y}_i}$ is the predicted value and the $\mathbf{n}$ is the total number of observation.

2.9.2 Validation

When the process of tuning and training a model is completed, the generalizability of the model should be tested to interpret the performance on model on independent unseen data. This is called model validation and in order to validate a model, it needs to be tested on unseen data. Based on the validation results, statements about a model can be made i.e. if the model is overfitting, underfitting or generalized. Two validation methods used in this study are provided below:

Train Test Split is a technique in which the data set is randomly splitted into subsets e.g train and test. The idea here is that the train subset is used for training a model while the test subset is kept aside during the training process. After training the test data acts as a independent unseen data set and is used to evaluate the model performance. There can be different algorithms for splitting data in to subsets, one of which is the *Kennard stone* algorithm. The algorithm extracts a subset of data that provides uniform coverage over the data and includes samples on the boundary of the data set. Kennard stone works by selecting two samples that have the largest geometric distance between them. To select more samples, the algorithm then picks a sample that is farthest apart in terms of separation distance from the select samples. Separation distance is the distance from a sample to its closest sample. The process is repeated until the required number of samples are added to the set. For this study, the kennard-stone package version 1.1.0 from python package index was used[40].

While building models, there is always a possibility of the data being limited and of that is the case then the train test split technique is very likely to show high bias. This is because of the possibility of the model missing information about the data that was not in the randomly selected training set. Therefore, the train test approach is not always enough for validating the model. Another technique

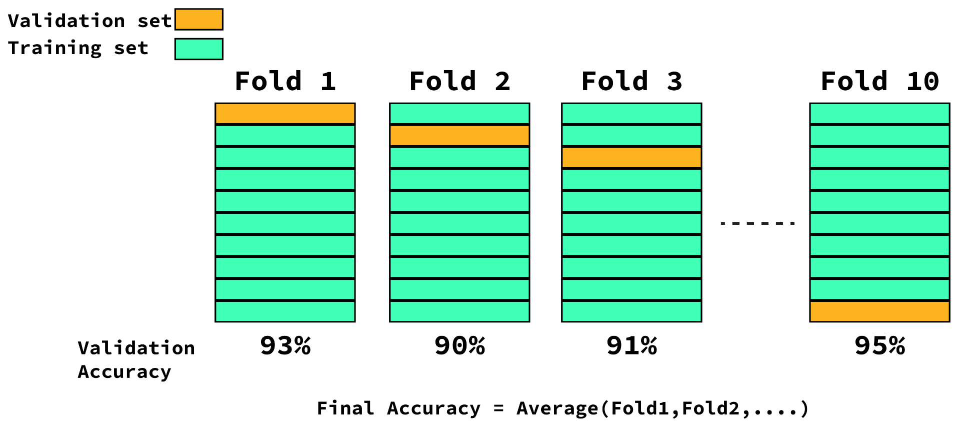


Figure 11: Kfolds cross validation

that is used for model validation is *Cross Validation*. This technique can be described as a re-sampling procedure for validating models. It works by training a model on a portion of data and keeping a portion aside from the training data. The portion of data that is kept aside is used for validation. This process is repeated several times and until the model is trained and validated on multiple portions of data. By performing cross validation, the generalizability of a model can be tested and one can get an idea of how the model will behave on unseen data in the future.

Kfolds cross validation is a type of cross validation that works by dividing the data set into K number of folds. Each fold represents a portion of the data. Kfolds cross validation works in three steps:

- Split the data randomly into **K** number of folds. The value of **K** is specified.
- The model is trained on $\mathbf{K-1}$ folds and the $\mathbf{K^{th}}$ fold is used for validation. After training the model is evaluated using a evaluation metric and the results are saved.
- Repeat until all the folds have served as the validation set. The scores that are recorded for each repetition is averaged and the averaged scores are the performance metric for the model.

Kfolds cross validation is simple and it can result in models that are less biased. The value of **K** is a parameter and is based on choice. The value of **K** should be based on the size of data. The steps of K fold cross validation are visualized in figure[11].

3 Methods and Materials

This section defines the materials and methods that are used to carry out the experimentation is this study.

3.1 Data set

The Raman spectroscopy data used in this study was the same data was used in a study to explore the effects of of model based preprocessing on Raman spectroscopy [38]. The data contains the spectral samples of milk and the variable that is used as a response is the amount of iodine. There are 2682 samples and 2979 variables in the data. The variables represented the Raman shift that ranged between 3000 - 1 cm^{-1}. Samples in the data set were based on 232 unique biological samples and multiple replicates of the 232 unique samples were analyzed and recorded in the data set. Each observation in the data set belonged to a group(232 in total) and the group was a reference to the original unique biological sample. Figure 12 shows the first 500 observations of the spectra.

Mean Data Upon visualizing the raw spectral data, it was observed that the data shows some unusual peaks that stand out and it is not according to the overall trend. These unusual peaks may be caused by a few observations in the data. It might very likely be that these records were affected

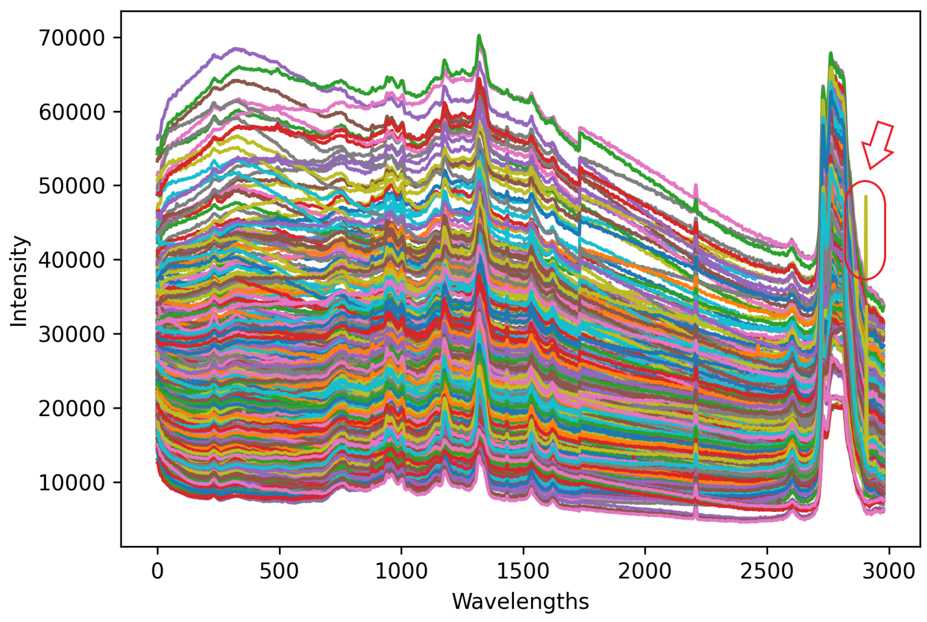


Figure 12: Data with indicated unusual peak

by some instrumental or atmospheric effects when they were recorded. An example can be visualized in figure12 which shows an unusual trend marked by a red bounding box and arrow. Machine learning models can catch these unusual trends and follow on them can diverge from the optimal path by learning these unimportant details. Therefore, in order to avoid this, the 2682 observations in the data set were grouped together based on the group number they belonged to and then averaged. This was done for the response variable as well. This resulted in a data set of 232 observations that was used for training models in this study. Averaged data is visualized in figure 13. The mean training data is treated as raw data and it is also referred as *raw* data in the upcoming sections.

3.2 Specifications

All the experiments were performed on *Orion* and Kaggle notebooks. Orion is a High-Performance Computing (HPC) cluster infrastructure owned by the Norwegian university of life science (NMBU). Kaggle is a online platform that provides free resources for their members for solving data science problems. The two main packages for building, training and evaluating models used were:

- Sklearn(scikit-learn) (0.23.2) [41]
- Tensorflow (2.4.1) [42]

3.3 Model Training Strategies

This section provides brief descriptions about the strategies and pre-training procedures that were performed for training models used in this study.

3.3.1 Validation

The data set was splitted into an 80% 20% split of train and test data respectively. After splitting, there were 185 samples in the train split and 47 samples in the test split. The train data was used for training and tuning the models while the test data was kept hidden. After the training procedure, the test data was used for evaluating the models. In order to compare results of different models, same splits of the data set were used to train and evaluate the model.

Tensorflow and specifically GPUs follow the principle of parallelism. So, while using the GPU with tensorflow, it is not possible to control the sequence of tasks and thus it is almost impossible to have reproducible deterministic results. It is possible to restrict CNN to produce same deterministic results

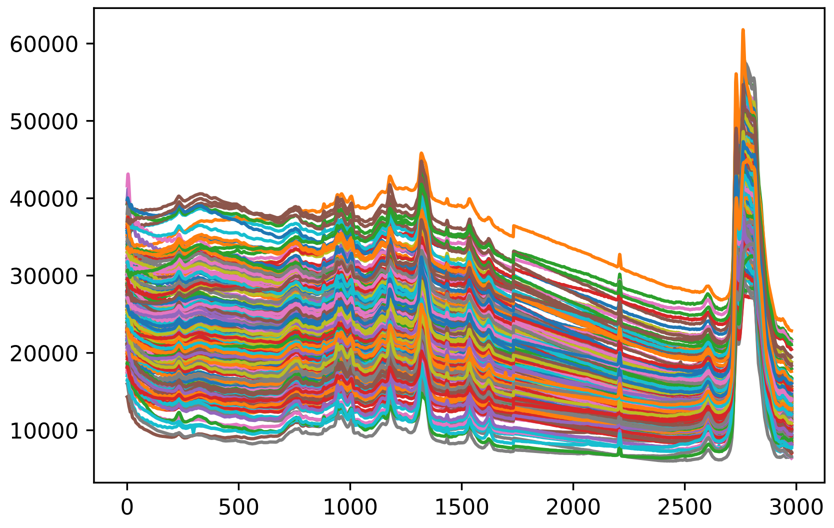


Figure 13: Mean data

by not using GPUs but this results in a significant increase in training time. Therefore, training of the CNN models was carried out using GPUs and 5 repeated experiments were executed for all CNN models. The results of the 5 experiment repetition averaged and treated as the final model result and it showed the robustness of the model.

3.3.2 Epochs

CNN models were trained on raw data for 1000 epochs. This number was chosen considering that two training methods i.e reduce learning rate on plateau and early stopping were used while training the models. Therefore, 1000 epochs should be enough for a model for convergence.

3.3.3 Batch Sizes

Three different batch sizes i.e. 8, 16 and 32 were used while selecting the benchmark model. These numbers were chosen as possible optimal options considering the size of the train data split i.e. 185. After the selection of the benchmark model, the batch size that resulted in the best scores was used for further experimentation.

3.3.4 Standardization

Standardizing a data set involves re-scaling the distribution of values so that the mean of observed values is 0 and the standard deviation is 1. The scale of the input and output used to the train the models are important factor as the weights of the neural network are updated based on the estimates of error on the training data set. Unscaled values of variables can results in slow and unstable learning process. Furthermore, unscaled target variables can result in a exploding gradients that can result in a in efficient model. Therefore the process of data standardization was performed on the input data before training on the CNN in search of making the training process faster and to make a more efficient and consistent model. The process of standardization was not performed for PLSR as it extracts its own features and the data points are mapped on a different scale and coordinate system.

3.4 Benchmark Models

This section defines the experiments and protocols that were followed to determine the benchmark models for the raw data set. These benchmarks models results will be used as benchmark scores for all the models used in different experiments.

3.4.1 PLSR model

Spectroscopic data sets contain large number of variables and fewer observations. Furthermore, spectroscopic data suffer from multicollinearity. By taking all these factors into account, PLSR was chosen as a linear benchmark model since it has the ability to deal with these problems. PLSR transforms the input data into subspace representation based on the responses which results in lower dimensional representation of the data and significant decrease in the number of variables.

When using PLSR, it is important to choose the optimal number of PLS components as the number can vary for different problems and data sets. The reason for varying number of optimal PLS components is that, first of all, the total number of meaningful PLS components that can be extracted for a particular data set can vary as the rank of the covariance/correlation matrix depends on the number of samples and the number of features and both these number can vary for different data sets. Secondly, each extracted PLS component explains a fraction of variation present in the data. The optimal number of PLS components is when the components contains the desired fraction of variation and adding more components does not add a significant amount of variation. However, if the number of selected PLS components for training model are smaller than optimal then this can lead to model underfitting as the model will not be able to learn enough variation in the data and if the number is larger then this may result in a computationally in efficient model which adds little value.

Experiment Design The experiment design for choosing the best PLSR model for the data set used in this study was divided into two steps i.e. Getting the best number of components and model evaluation. *Grid Search* is a concept that is used to compare and choose the optimal hyper parameters values while developing models. It works by looping through the predefined choice of parameters and fitting models with each possible combination of parameter options on the data. In the end, the results for each model are compared and the best set of parameters for the particular data set are determined. Grid search algorithm was used to determine the optimal number of PLS components in this experiment. A parameter list containing the possible values of number of PLS components was set and passed to grid search algorithm to get the optimal number of PLS components for training the PLSR model on raw data. The experiment was performed using the train split of the raw data and the predefined choice parameters contained list of number of PLS components that ranged up to 30. A PLSR model with each choice of number of components was trained and evaluated. In order to validate the results, K folds cross validation with 5 splits was used. At the end, the mean RMSE values for all models trained using different number of PLS components were compared to determine the best parameter value. Afterwards, a PLSR model using the determined optimal number of PLS components as parameter was evaluated on the test data split that was kept hidden for the model selection and training phases.

3.5 CNN Model

To choose an architecture for deep learning model is a challenging task. The architecture is problem dependent and often found by hit and trial method [43]. The number of layers, number of neurons in each layer, activation function and optimizer are all important factors when designing a deep learning model. Each factor can significantly effect the performance of a model. In this study, the choice of architecture was based on literature reviews and performing test experiments. A few different architectures that were selected from different articles were tested on the data to come up with the best model architecture. The model architectures that were tested are given below:

The first model architecture was taken from a research article that explored the benefits of deep learning for preprocessing spectral data[43]. The architecture of the model was simple and it consisted of a single convolutional layer followed by a batch normalization layer. After that there is a fully connected dense layer with 32 neurons. A flatten layer is added before the dense layer which changes the dimension of the 2D output from the convolutional and batch normalization layer to a 1D vector so that it may be passed as input to the dense layer. Finally, there is a dense layer with 1 neuron at the end that serves as a output layer. Relu activation function is used as an activation function. Adam optimizer is used as an optimizer with mean squared error being the loss. The full architecture is provided in Table 1.

The second architecture was taken from a research article in which the authors tried to predict the quantity of different organic materials by using CNNs on spectroscopic data[44]. This architecture

Type	No. of filters	Kernel Size	Activation Function/Attribute
Conv1D	8	9	Relu
Batch Norm	-	-	-
Flatten	-	-	-
Dense	32	-	Relu
Dense	1	-	-
Loss	-	-	MSE
Optimizer	-	-	Adam
Trainable Parameters	-	-	760,737

Table 1: Summarized CNN architecture 1

consists of six trainable layers including four convolutional layers and two fully connected layers. Every convolutional layer is followed by a max pooling layer. All convolutional layers are identical except that the number of filters for the first two layers are 32 while the number of filters for the last two layers are 64. Kernel size for convolutional layer is 3 and the activation function used is Relu. Pooling size for the last max pooling layer is 4 while pooling size is kept to 2 for all other max pooling layers. A drop out layer with a dropout percentage of 0.3 is also added after the last convolutional and max pooling layer which acts as a regularization layer to avoid overfitting. After the dropout layer, the feature map is then flattened using a flatten layer and used as input for the fully connected dense layer with 512 neurons. Finally, an output layer with 6 neurons is used to produce the output because 6 values were to be predicted in the research paper. However, to use the model for this study, the neurons of the output layer was changed to 1 because only one output will be produced for this data. Relu, Adam and MSE were used as activation function, optimizer and loss function respectively with this model. The architecture summary is provided in Table 2.

Type	No. of filters	Kernel Size	Activation Function/Attribute
Conv1D	32	3	Relu
Max Pool	-	-	size = 2
Conv1D	32	3	Relu
Max Pool	-	-	size = 2
Conv1D	64	3	Relu
Max Pool	-	-	size = 2
Conv1D	64	3	Relu
Max Pool	-	-	size = 4
Flatten	-	-	-
Dropout	-	-	0.3
Dense	512	-	Relu
Dense	1	-	-
Loss	-	-	MSE
Optimizer	-	-	Adam
Trainable Parameters	-	-	24,380,673

Table 2: Summarized CNN architecture 2

One more architecture that was tested was taken from a web article that tested the performance of CNN model on spectroscopic data[45]. The architecture of the model was based on two 1D convolutional layers, a dropout layer, a dense layer and finally a dense output layer with one neuron to produce the output. The activation function used in this network was *Relu*. The activation function used in the output layer was the linear activation function. Adadelta was used as an optimizer with a learning rate of 0.01 and the loss function chosen was mean squared error. Adadelta is a robust extension of Adagrad that adapts learning rates based on a moving window of gradient updates, instead of accumulating all past gradients. The choice of Adadelta provided an overview of using a optimizer that has a different working procedure. Full architecture of the model is given in Table 3.

Another architectural design was taken from an article that used a 1D convolutional neural network

Type	No. of filters	Kernel Size	Activation Function/Attribute
Conv1D	8	32	Relu
Conv1D	16	32	Relu
Flatten	-	-	-
Dropout	-	-	0.5
Dense	128	-	Relu
Dense	1	-	-
Loss	-	-	MSE
Optimizer	-	-	Adadelta(lr = 0.01)
Trainable Parameters	-	-	5,978,649

Table 3: Summarized CNN architecture 3

to predict the phosphorous content in soil using spectroscopic data[46]. The architecture included 10 hidden layers containing four convolutional layers, four max-pooling layers, and two fully connected layers. A max pooling layer was present after each convolutional layer. Activation function used was rectified linear unit (ReLU) for all hidden layers. Two dropout rates of 0.4 and 0.2 were used to avoid overfitting. The architecture is summarized in Table 4.

Type	No. of filters	Kernel Size	Activation Function/Attribute
Conv1D	32	20	Relu
Max Pool	-	-	size = 2
Conv1D	32	20	Relu
Max Pool	-	-	size = 5
Conv1D	32	20	Relu
Max Pool	-	-	size = 5
Conv1D	32	20	Relu
Max Pool	-	-	size = 5
Dropout	-	-	0.4
Flatten	-	-	-
Dense	100	-	Relu
Dropout	-	-	0.2
Dense	1	-	Linear
Loss	-	-	MSE
Optimizer	-	-	Adam
Trainable Parameters	-	-	84,809

Table 4: Summarized CNN architecture 4

The fifth architectural concept was taken from an article that studied the application of CNN for retinal spectroscopic data[47]. The model architecture was used as a feature extractor and consisted of all convolutional layers. The architecture was made up of three convolution layers having 128, 128, and 64 filters respectively. Each convolution layer has a kernel size of 25 and uses exponential linear unit (ELU) as an activation function. Each CNN is followed by a max pooling layer with a pooling size of 2. The output of this feature extractor model was the flattening of all 64 filters of the last convolution layer. For this study, a dense layer consisting of one neuron was added to produce an output for prediction. According to the author, the choice of the ELU has two desirable properties: producing a zero-centered distribution, which can make the training faster; and having one-sided saturation which leads to better convergence. Adam was chosen as an optimizer and finally mean squared error was opted as a loss function for optimizing the training process. The architecture of the model is summarized in Table 5.

Type	No. of filters	Kernel Size	Activation Function/Attribute
Conv1D	128	25	elu
Max Pool	-	-	size = 2
Conv1D	128	25	elu
Max Pool	-	-	size = 2
Conv1D	64	25	Relu
Max Pool	-	-	size = 2
Batch Norm	-	-	-
Dense	1	-	Linear
Loss	-	-	MSE
Optimizer	-	-	Adam
Trainable Parameters	-	-	640,513

Table 5: Summarized CNN architecture 5

3.5.1 Best Model

After observing the trends and behaviors of the experiments conducted on the 5 model architecture that are defined above, some new architectures were designed and tested. The new architectures took into account the pros and cons of the previous architectures to come up with a model architecture that would outperform the other models. Finally, after a good amount of effort spent on testing we came up with an efficient model that was consistent and produced comparable results to the PLSR benchmark model. The architecture was based on 1 convolutional layer and 1 fully connected dense layer with 64 neurons. 32 filters and a kernel size of 30 was used in the convolutional layer. A dropout layer with a dropout value of 0.3 was also added before the dense layer. ReLU activation function was used for both CNN and the dense layer. Adam optimizer was used along with the MSE loss function to train the model. The architecture is summarized in Table 6.

Type	No. of filters	Kernel Size	Activation Function/Attribute
Conv1D	32	30	Relu
Flatten	-	-	-
Dropout	-	-	0.3
Dense	64	-	Relu
Dense	1	-	-
Loss	-	-	MSE
Optimizer	-	-	Adam
Trainable Parameters	-	-	6,042,721

Table 6: Summarized best CNN architecture

3.6 Preprocessing

The effect of preprocessing was also analyzed and discussed in this study. Both benchmark PLSR and CNN models from the raw data experiment were trained and evaluated on preprocessed data as well. Results of the models that were trained on preprocessed data were compared with the results of models that were trained on raw data. The method that was used for preprocessing the data set used in this study was the EMSC method that corrects the spectra by determining a least squares fit based on a reference spectrum. Mean spectrum was used as a reference spectrum for the implemented EMSC method used in this study.

Experiment Design When preprocessing spectroscopic data with EMSC method, it is important to choose the optimal degree of polynomial [2.8]. The polynomial degree is a hyper parameter when applying EMSC method. The objective of this experiment was to find the optimal polynomial degree that results in the lowest error. Different choices of polynomial degrees ranging from 0 to 7 were used as a parameter value for the EMSC method to preprocess the raw spectral data. The preprocessed data was then used to train and evaluate the benchmark models. The results of each model trained and evaluated on data preprocessed using a different polynomial degree were compared to determine the best parameter value for EMSC method.

3.7 Data Augmentation

Data augmentation is a technique that is used for extending the data set. The main principle of this technique is to apply a range of transformations to the original data records to derive new observations for the data. The newly derived observations are called augmented observation. Deep learning models usually require huge amounts of data. As a point of reference, the database Imagenet[48] contains one million image samples. Training deep learning models on data sets that have larger sample size can result in better robust models. Therefore, deep learning community uses the data augmentation methods to increase sample size when the datasets are not sufficiently large. The augmented observation can then be used to train deep learning models. Augmented observations can either be added to the original data set or can be independently used to train models. Since the augmented observations are obtained as a result of applying transformations to the original data set, they can help the model to learn the variation in the data and hence can result in improving the performance of the model. By using this data, the model will not be learning something new however, this data can help the model to understand the variability better. In order for the augmented observations to be successful, the augmented data needs to fulfill the following requirements[5].

- Transformations should change the targets in a way that the targets correspond to the augmented sample like the targets are related to the original samples.
- Transformations should try to produce data as close as possible to the original observations.

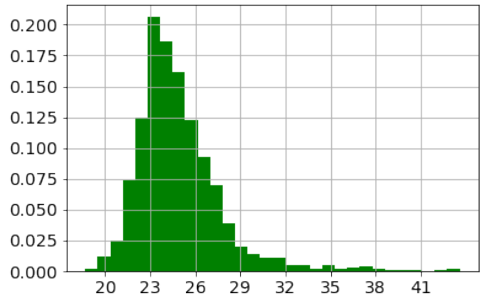
Data augmentation is a well known method in the image analysis field. Image data augmentation involves creating transformed version of images that belong to the same class as the original image. The transformations include a range of operations like flips, zooms and shifts etc. Augmentation is beneficial when producing new observations is expensive or time consuming. Spectroscopy is an example of this statement. It can be time consuming to record new readings for spectroscopic data sets because generating new data requires to go through a whole process of sampling and recording the measurements. Nevertheless it is important to note that augmentation can not introduce new chemical information in the data. Therefore, it cannot replace the effect of measuring new samples. This means that even if the results are improved by augmenting data, it is not because that the augmentation introduced some new information in data, but because the model learned how to better account for the variability existing in the data[5].

Very few studies have been performed on augmentation and deep learning in the field of spectroscopy. In this study three new methods were developed and used to augment observations for the Raman spectroscopic data set. To the best of my knowledge, these method have not been studied before for Raman spectroscopy data. A different study tested the effect of EMSC augmentation but that was on a different type of spectral data [5].

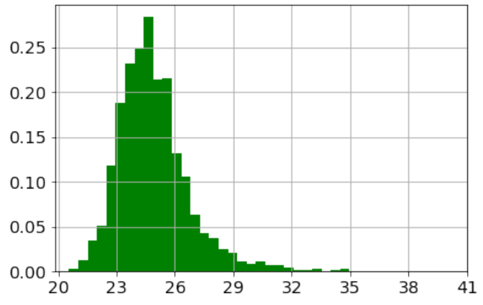
3.7.1 Linear Augmentation

The first augmentation method that was derived and tested is called linear augmentation. The method is based on the principle of linear combinations. Linear combination is a central concept of linear algebra. A linear combination is a expression constructed by multiplying a set of terms by a constant and then adding the results. In terms of vectors, a linear combination will be defined as a vector that is obtained by adding two or more vectors that are multiplied by different scalar values. By using the concept of linear combinations, new observations can be augmented as a result of linearly combining data samples from the original data set. Therefore, Linear augmentation method is based on this theory i.e. a specified number of randomly selected samples from the original data are linearly combined to generate new observations.

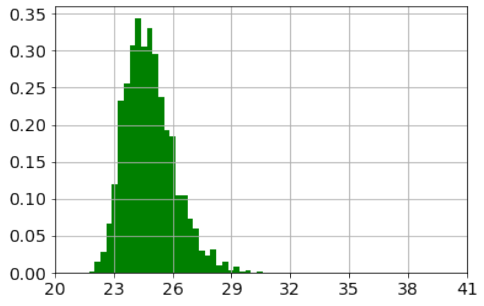
Effect of number of samples in linear combination Three different numbers i.e. 2, 5 and 10 were the numbers chosen as possible parameter options for the linearly augmentation experiment. These numbers define the number of rows that are linearly combined to augmented new samples. This



(a) Histogram showing range of response values of 2000 augmented observations when the observations were augmented by linearly combining 2 samples



(b) Histogram showing range of response values of 2000 augmented observations when the observations were augmented by linearly combining 5 samples



(c) Histogram showing range of response values of 2000 augmented observations when the observations were augmented by linearly combining 10 samples

Figure 14: Response value distribution when different number of rows are linearly combined to augment data

means that if the augmentation is performed with 10 as the parameter value, then the new augmented observation will be a result of linearly combining 10 randomly selected samples from the original data. The parameter options also shows the effect of increasing number of linearly combined samples. To explain effect of different number of samples used in linear combination, histograms of the response values of the augmented data were plotted and observed. Figure 14 shows the distribution of response values for the augmented observations when the number of linearly combined samples from raw data were 2, 5 and 10. In the figure histogram plots, it can be observed that when the number of linearly combined samples are 5 or 10, the range of the of response values is narrowed down compared to when the number is 2. In fact, the range is largest when the number of linearly combined samples are only 2. When a small number of values are randomly selected from a set of values then the mean of the selected values can be significantly different from the mean of the set of values. Furthermore, if extreme(elements that are either significantly greater or smaller than the mean) values are randomly selected from the set of values then the extreme values will have a significant effect on the mean of the randomly selected values when the number of total randomly selected values are small e.g. 2. As a result, the range of the average values for different sets of randomly selected values will be large. However, when the number of randomly selected values is greater e.g. 10, then the extreme values will not have a strong effect on the average of the randomly selected values and as a result the range of the values in this case will be narrowed down. In this case, the average values of the randomly values selected are closer to actual mean. This is the reason that the histogram in Figure[14a], when the number of linearly combined samples is 2 shows a wider response range compared to the histogram showed in figure[14b] and figure[14c].

Experiment Design The experiment for evaluating linear augmentation technique was divided into three steps i.e. Best parameters, best samples size and evaluation. The raw data was splitted into train and test set. The training set was used for the parameter selection phase(best parameters and sample size) and augmenting new data while the test data was kept hidden until the final step and was only used for evaluating the models and the augmentation method. Both PLSR and CNN models were used to evaluate the augmentation method. The details of the experiment are provided below.

The objective of first step of the experiment was to determine the ideal number of rows parameter for the linear augmentation method and the ideal number of PLS components for the PLSR model as the number of PLS components tend to change for different data sets and different sample size [49]. The method for determining the ideal number of components was similar to the benchmark model i.e. test different number of PLS components options and choose the one that performs best. All three parameter options for the linear augmentation method were used to augmented data with a constant sample size of 500. The augmented data was then used to train and evaluate the performance of the model. The models were validated using K folds cross validation and the number of folds used for the cross validation were 5. The number of rows parameter option that resulted in the lowest error for both models was determined as the best parameter for the linear augmentation method.

After determining the best set of parameters for the linear augmentation technique, the next step was to determine the ideal sample size. Sample size is an important factor as the whole point of performing augmentation is to have an ideal sample size for training machine learning models. Therefore, data was augmented with different sample sizes using the linear augmentation method to determine the ideal sample size and observe the effect of increasing sample size. 200, 500, 1000, 1500 and 2000 were the sample size options that were tested in this experiment. The validation procedure for this step of the experiment was same as parameter selection step i.e. k fold cross validation.

After determining the best parameter and the best sample size, the last step was to evaluate the augmentation method. Data augmented using the determined best sample size and parameter was used to train PLSR and CNN models. After training the models were evaluated on the original raw test set of the data. This experiment was repeated 5 time in order to get more robust estimates. Final R2 and RMSE values were the averaged values of the 5 repetition.

3.7.2 PLSR Augmentation

PLSR [2.3] decomposes the original data into scores and loadings. Equation 4 and 5 show the decomposition of data and response matrices into scores and loadings. The scores matrix represent the location of data samples in the PLS components space and the loadings matrix represent the location PLS component space with respect to the original data space. The original data matrix can be reconstructed after applying PLSR by using the scores and loadings matrix. In addition to constructing the original data matrix, the same operation can be used to augment new samples. In theory, the loadings matrix can be multiplied with a randomly generated scores matrix that lies in the same space as the original scores matrix to generate new samples. The randomly generated scores matrix will contain the original data points with some random deviations and new samples will be generated as a result of product of the original loadings and the randomly generated scores matrix. In order for this method to work it is important that the data points in the randomly generated scores matrix all lie in the same space and their position is close to the original data points. In order to ensure this, the values of each PLS score were sampled from a normal distribution which was drawn using a mean value of zero and a standard deviation of the particular PLS score. The method worked by first estimating the standard deviation for each PLS score using the maximum likelihood estimation. The new data samples were then generated by sampling each PLS score value from the normal distribution with mean zero and the standard deviation of that PLS score.

Experiment Design The outline of experiment design for PLSR augmentation method was quite similar to linear augmentation method. This experiment was also divided in to three steps with only the first step being different i.e parameter for the augmentation method. In order to augment data using PLSR augmentation a set of PLSR scores and loadings matrices are required . These matrices can be obtained by fitting a PLSR model to the data. Therefore, the parameter to determine for the PLSR augmentation method is the number of PLS components for PLSR model that will be used to generate the scores and loadings matrix for generating artificial samples. For this purpose, different number of PLS components were tested. The parameter options ranged from 20 to full rank(max observations in the data). The scores and loadings of PLSR models with different number of components were used to augment data with a constant sample size of 500 using the PLSR augmentation method. Each augmented data set was used to train models and the parameter option that resulted in the best model results was selected as a optimal number of PLS components for augmenting data. Following the determination of the best augmentation method parameter, the procedures of the remaining steps of the experiment were similar to the experiment linear augmentation method. The other steps included determining the best PLSR model for evaluating augmentation method, determining the best sample size and method evaluation. The validation procedure of the experiments were also similar to the linear augmentation method i.e. K fold cross validation.

3.7.3 EMSC augmentation

Equation 17 shows how the corrected spectrum was calculated using EMSC preprocessing method. This equation is a modified form of Equation 16 that shows the equation for the original spectra. According to these equation, the corrected spectra is basically the mean spectrum added to a residual term. Therefore, by taking these equations into account the original spectrum can be obtained by:

$$\mathbf{A}(\bar{\mathbf{v}}) = \mathbf{A_{corr}}(\bar{\mathbf{v}}).\mathbf{b} + \mathbf{a} + \mathbf{d_1}(\bar{\mathbf{v}}) + \mathbf{d_2}(\bar{\mathbf{v^2}}) + ... + \mathbf{d_n}(\bar{\mathbf{v^n}}) + \mathbf{e}(\bar{\mathbf{v}}) \tag{20}$$

In theory, the EMSC method can also be used as an approach for augmenting new data by introducing some physical variations in data. The idea here would be to first calculate the parameters of the EMSC that are based on scattering and instrumental effects and then augment new data by introducing similar physical effects. The method is based on the principle of reverse preprocessing where the EMSC preprocessing is reversed to transform the preprocessed spectra back to raw data with some slight variations. The slight variations in the data are introduced by using new random EMSC parameter values that are drawn from the same distribution as the original parameter values. Thus, in order to create new samples using EMSC augmentation, first the raw data is preprocessed and the EMSC parameters i.e. $\mathbf{a}$, $\mathbf{b}$, $\mathbf{d_1}$, $\mathbf{d_2}$ ... $\mathbf{d_n}$ are calculated for each spectrum in the spectra. Afterwards, a new set of parameters are drawn from a normal distribution that uses the respective mean and standard deviation of the measured parameters. Generation of new random parameters around the old ones helps to preserve correlation between them and thus helps to avoid generation of unnatural independent parameters. The newly generated parameters can then be used to augment a new spectrum by reversing the effect of preprocessing. The equation 20 then becomes:

$$\mathbf{A}(\bar{\mathbf{v}}) = \mathbf{A_{corr}}(\bar{\mathbf{v}}).\mathbf{b}^{'} + \mathbf{a}^{'} + \mathbf{d}^{'}_{1}(\bar{\mathbf{v}}) + \mathbf{d}^{'}_{2}(\bar{\mathbf{v^2}}) + ... + \mathbf{d}^{'}_{n}(\bar{\mathbf{v^n}}) + \mathbf{e}(\bar{\mathbf{v}})(21)$$

where $\mathbf{a}^{'}$, $\mathbf{b}^{'}$, $\mathbf{d}^{'}_{1}$, $\mathbf{d}^{'}_{2}$ ... $\mathbf{d}^{'}_{n}$ represent the new randomly generated EMSC parameters.

Experiment Design The experiment design for EMSC augmentation was identical to the linear augmentation and PLSR augmentation methods. This experiment was also divided into 3 steps. Objectives of second and third steps were exact same i.e. to determine the best augmentation sample size for models and then to evaluate the augmentation technique by using the best set of parameters and sample size determined in the previous first and second step. However, the first step was a bit different. The first step of the experiment is described below.

Polynomial degree is a required parameter for the EMSC method. The idea behind EMSC augmentation is to reverse the process of EMSC method by adding some deviations in the parameter values. These parameter values are obtained by first apply the EMSC method to the data. Therefore, in order to apply the EMSC method and then reversing the process, the polynomial degree is required. The first objective of the EMSC augmentation experiment was to determine the best polynomial degree for both PLSR and CNN models for the purpose of data augmentation.

Outline of this step was similar to the first steps of linear augmentation method and PLSR augmentation method. For this method, data was augmented using different polynomial degree options ranging from 0 to 7. After augmentation the data was evaluated for both models and the best options were determined for both models by validating the RMSE scores using K fold cross validation. In addition to the determining the best polynomial degree for augmentation, the first step of the experiment also involved determining the optimal number of PLS components for the PLSR model that will be used to evaluate the augmentation method.

3.8 Learning Curves

In general, learning curve is the representation of rate of something's growth and improvement. In machine learning, learning curves are also used for the same purpose. A learning curve is a representation of a model's learning performance evaluated over experience or time. Learning curves are used as diagnostic tool for models that learn from data sets incrementally. The model is evaluated after every increment in sample size. Finally, a plot of the measured performance for each sample size is created to show the learning curves. Observing the learning curves of the model can help to diagnose problems like underfitting and overfitting as well as whether the training and validation splits are suitable.

Learning curves were also used as a diagnostic tool in this study. The idea was to observe that increasing the number of samples in the data can actually improve the performance of the model. Furthermore, the experiment also gave an overview of the suitable number of samples that are enough to produce a generalized model. The learning curves have not been used very much with deep learning models. With deep learning models, it is interesting to see how the performance is affected with the increasing sample size. It is also interesting to observe the learning curves for raw data in this study as the raw data contains so few samples. A factor to observe is if the learning curves flattens and the model generalizes with the available sample size and also how fast it flattens out.

In order to carry out the experiment, models were trained on subsets of data and the size of data sets was increased incrementally. The training subsets ranged from 10 percent to 100 percent of the data with an increment size of 10 percent. So, the models were trained and evaluated 10 times for each subset of data. The scikit learn implementation of learning curves was used to carry out the experiment. The experiment was conducted with both PLSR and the best CNN model. Kfold cross validation was used to evaluate the performance of model with 5 being the number of folds. The scikit learn's learning curves function takes a percentage of data, trains the model and evaluates the model with cross validation. At the end, a list containing the results of the model for all sample sizes is obtained and the results are plotted for visualization of the learning curves.

4 Results

This section presents the results obtained by performing the experimentation described in the previous section of the study. The experiments conducted in this section were intended to compare the performance of CNN for different augmentation methods. The results obtained from the experiment will also be used to evaluate the different augmentation methods. Furthermore, in addition to the CNN model, the results of the PLSR model are also presented to analyze the behavior of both CNN and PLSR on a particular data set.

4.1 Benchmark Model Results

This section presents the results of the benchmark model experiments [3.4]. Results of the grid search CV experiment that was used to determine the best number of PLS components for the PLSR model are visualized in figure 15. The figure shows R2 and RMSE values for PLSR models with different number of PLS components. Observing this figure shows that the highest R2 score and the lowest RMSE is obtained when number of PLS components are 17. Therefore, a PLSR model with 17 PLS components was selected as the benchmark model for the raw data. The benchmark PLSR model was then trained and evaluated on raw data and the results showed a R2 score of 0.90 and a RMSE of 0.94. Similarly, a benchmark CNN model was also determined. The best CNN model [3.5.1] showed the best results when the batch size was 8. Therefore, best CNN model architecture with batch size 8 was determined as the benchmark CNN model. This model was used for the CNN experiments in this study. The results of the benchmark CNN model showed an average R2 score of 0.91 and a RMSE of 0.88. The average number of epochs required to train the CNN model on raw data were 484. Loss function trend for benchmark CNN model trained on raw data is visualized in figure 16. The figure shows loss function trend lines for train and validation data over the course of the training process and demonstrates the ability of CNNs to minimise the loss with increasing number of epochs. The loss trend figure shows a stable trend towards the end of the training process which implies a stable model.

After evaluation on raw data, the benchmark models were evaluated on preprocessed data to compare the results and to analyze the effect of preprocessing on Raman spectroscopy data. Benchmark PLSR model performed best when raw data was preprocessed using EMSC method with a polynomial degree of seven resulting in a R2 score of 0.97 and a RMSE of 0.52. However, the benchmark CNN model showed the best results i.e. a R2 score of 0.91 and a RMSE of 0.92 when EMSC method was

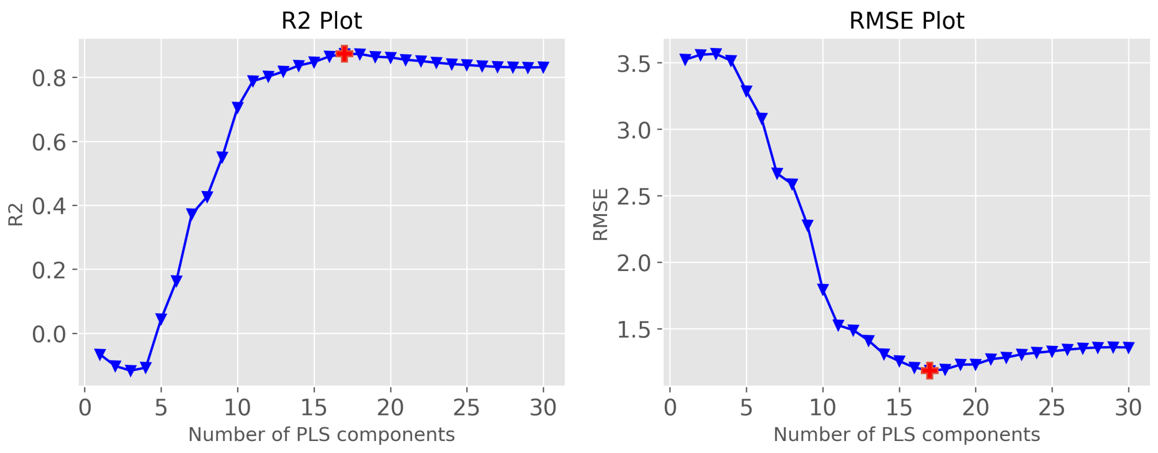


Figure 15: Grid Search CV results for PLSR model with PLS components ranging from 1 to 30. The left plot shows the R2 scores and the right shows the RMSE

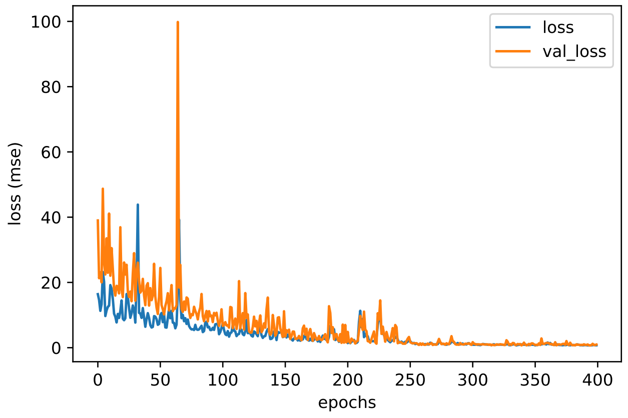
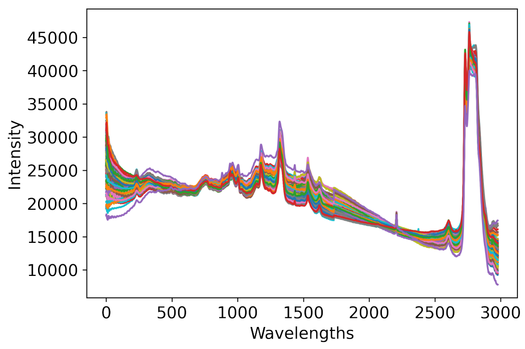
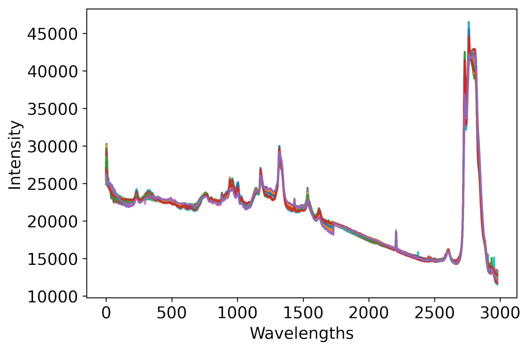


Figure 16: Loss function trend for the benchmark CNN model when trained on Raw data. The blue line shows the trend for training loss while the orange shows the trend for validation loss. First 10 epochs were trimmed for the reason of better visualizing trend because the error in the first 10 epochs was huge and it was setting the scale very high.



(a)



(b)

Figure 17: Raw data preprocessed using EMSC method with polynomial degree one and seven. a shows the spectra when raw data is preprocessed using polynomial degree one and b shows the processed spectra using seventh degree polynomial

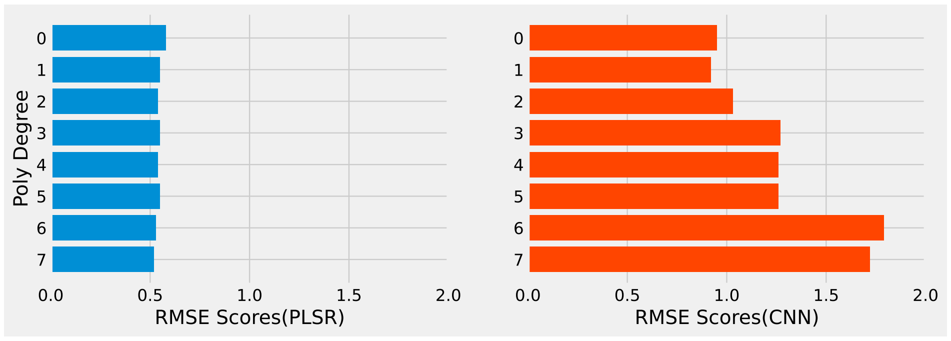


Figure 18: RMSE values for benchmark models evaluated on data preprocessed using EMSC method with polynomial degree ranging from zero to seven. The left plot with blue bars show the PLSR RMSE values while the right plot with orange bars shows RMSEs for CNN

used with a polynomial degree of one. The average number of epochs required to train the CNN model preprocessed using EMSC method with polynomial degree of one were 130. Spectra preprocessed using EMSC method with polynomial degree one and seven are visualized in figure 17a and 17b respectively. The RMSE scores for benchmark models evaluated on data preprocessed using EMSC method with polynomial degree ranging from zero to seven are visualized in figure 18. Notice that in figure 18, the RMSE values of the PLSR model with polynomial degree six(0.53) and seven(0.52) are approximately equal. Furthermore, both polynomial degrees showed the same R2 score i.e. 0.97 for PLSR model. This result is in accordance of a study performed on the same data that confirms that the PLSR model model shows a significant improvement in performance when the data is preprocessed using EMSC with a sixth degree polynomial [38].

Results of the benchmark models evaluated on raw and preprocessed data are summarized in table 7.

Model	R2	RMSE
PLSR(17) - Raw	0.90	0.94
PLSR(17) - Prep	0.97	0.52
CNN - Raw	0.91	0.88
CNN - Prep	0.91	0.92

Table 7: Summarized results of Benchmark models evaluated on raw and preprocessed data

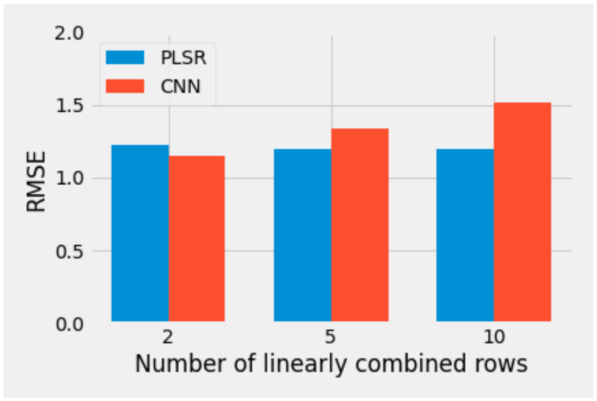


Figure 19: RMSE scores when PLSR and CNN model were trained on data augmented using linear augmentation method and different parameter options

4.2 Data Augmentation

This section shows the results of Augmentation methods that are discussed in this study. The results provided in this section will be used to evaluate and conclude the Linear augmentation, PLSR augmentation and EMSC augmentation methods.

4.2.1 Linear Augmentation

This section shows the results of the experiments that were used to evaluate the linear augmentation method. Figure 19 presents the results of the first step of the experiment i.e. best parameter selection. The figure shows the RMSE values for PLSR and CNN models based on the three opted parameter options for the linear augmentation method. The lowest RMSE obtained using the PLSR model was 1.19 and this was obtained when the number of linearly combined rows for augmenting data were 5. However, the lowest RMSE value obtained for the CNN model was 1.14 and this was obtained when 2 rows were linearly combined to generate new samples. In addition, another objective of first step of the experiment was to determine the best PLSR model for evaluating the linear augmentation method and the results showed that the PLSR gives the lowest RMSE(1.19) with 18 PLS components.

After determining the best parameter value for the augmentation method, the next step was to determine the best sample size. Since PLSR and CNN both performed best with different parameter options, the decision was made to generate different data sets for both models using the corresponding best parameter value. Therefore, for each sample size option two different data sets were generated using the linear augmentation method with different parameter values. PLSR was evaluated on data sets generated with using 5 as parameter value and CNN was evaluated on data sets generated using 2 as parameter value. The results of both models evaluated on datasets with different sample sizes are provided in figure 20. The figure shows that the lowest RMSE value of 1.19 for PLSR model is obtained when the sample size is at least 500. However, the best RMSE score for CNN model is 0.98 and this score is obtained when the number of samples were at least 1000.

The data sets used for evaluating the linear augmentation method were generated using the corresponding best parameter values and best samples sizes for PLSR and CNN model. Results of the experiment are presented in the table 8. R2 and RMSE values in table show that the CNN has outperformed the PLSR model with a significant difference. CNN showed an RMSE value of 0.67 while PLSR showed an RMSE value of 1.07.

Model	R2	RMSE
PLSR(18)	0.88	1.07
CNN	0.95	0.67

Table 8: Final results for linear augmentation method experiment

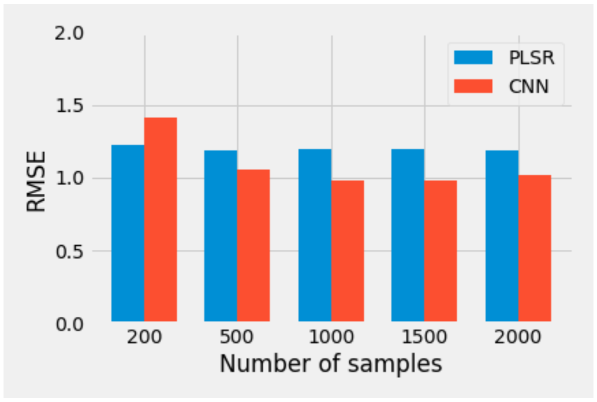


Figure 20: RMSE scores when PLSR and CNN model were trained on data augmented using linear augmentation method and different sample sizes

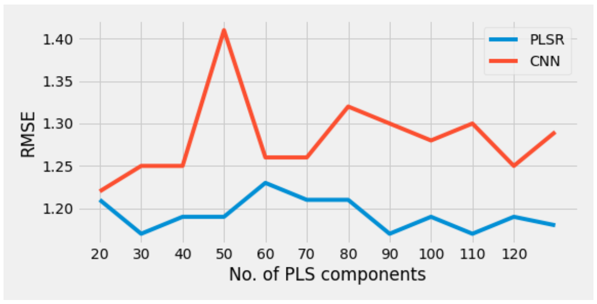


Figure 21: RMSE scores when PLSR and CNN model were trained on data augmented using PLSR augmentation with different number of PLS components

4.2.2 PLSR Augmentation

This section shows the results of the PLSR augmentation experiment. The first step was to determine the ideal parameter value i.e. the number of PLS components PLSR for augmentation method. Results visualized in and the figure 21 show that the RMSE values obtained for PLSR and CNN model when trained and evaluated on data augmented using the PLSR augmentation method with different number of PLS components. The lowest RMSE value i.e. 1.2 for the CNN model was obtained when the data set generated using 50 of PLS component as a parameter. However, PLSR model gave the best RMSE score of 1.14 on the data set generated using 110 PLS components. Furthermore, the lowest RMSE score on augmented data using PLSR model was obtained when the number of PLS components were 17. Therefore, a PLSR model with 17 number of PLS components was used for evaluating the PLSR augmentation method.

Similar to the linear augmentation method, in order to determine the best sample size value, the corresponding best parameter options for the augmentation were used for generating data set of different sample sizes for both models. Results are visualized in figure 22 which shows that the optimal sample size for PLSR model is 1500 while the optimal size for CNN model is 500. After determining the best parameter and sample size values for the augmentation method, the method was evaluated in the third step of the experiment and the result of the experiment are presented in table 9. The table shows that CNN model resulted in a RMSE of 1.11 while the PLSR showed an RMSE value of 0.94.

Model	R2	RMSE
PLSR(18)	0.90	0.94
CNN	0.87	1.11

Table 9: Final results for PLSR augmentation method experiment

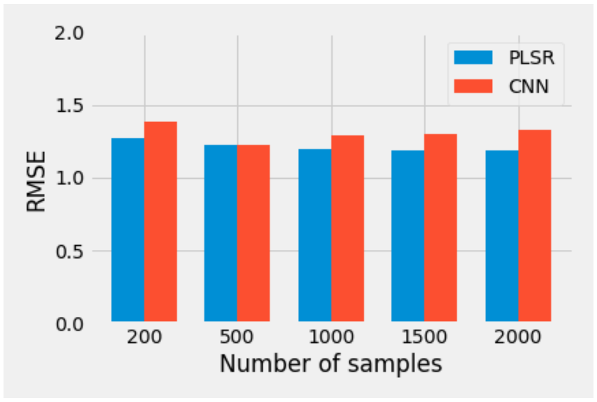


Figure 22: RMSE scores when PLSR and CNN model were trained on data augmented using PLSR augmentation method and different sample sizes

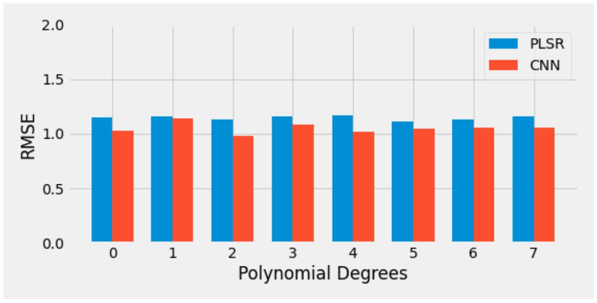


Figure 23: RMSE scores when PLSR and CNN model were trained on data augmented using EMSC augmentation with different polynomial degrees

4.2.3 EMSC Augmentation

Plot showing the RMSE values of PLSR and CNN models when trained and evaluated on data generated using EMSC augmentation method with different polynomial degrees is visualized in Figure 23. The figure shows that the lowest RMSE value(1.11) for PLSR is obtained when the polynomial degree was 5 however the lowest RMSE(0.98) for CNN is obtained when the polynomial degree was 2. Furthermore, the optimal number of PLS components that was determined using the experiment results was 21. The sample size experiment results are visualized in figure 24 which shows that the ideal data set sample size for PLSR model is 500 and for CNN model is 1000. Finally, the evaluation of the augmentation method was the final step and the results of the experiment showed that a RMSE value of 0.62 for CNN and a RMSE value of 0.75 for PLSR. The results of the evaluation experiment are also presented in table 10.

Model	R2	RMSE
PLSR(18)	0.94	0.75
CNN	0.95	0.62

Table 10: Final results for EMSC augmentation method experiment

4.2.4 Summarized results for all methods

The RMSE values of all augmentation methods, raw and preprocessed experiments are summarized in Figure 25. This figure shows varying results of PLSR and CNN models for different methods that shows the different natures of both models. PLSR model with preprocessed data was the best model and showed the lowest RMSE values compared to all other models. However, preprocessing resulted in no improvement in performance for the CNN. CNN model performed well with data augmented using

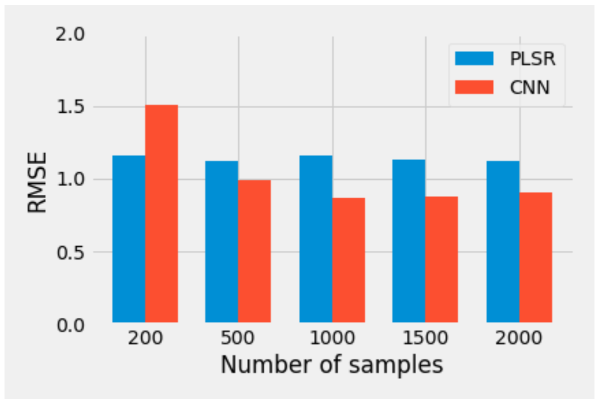


Figure 24: RMSE scores when PLSR and CNN model were trained on data augmented using EMSC augmentation method and different sample sizes

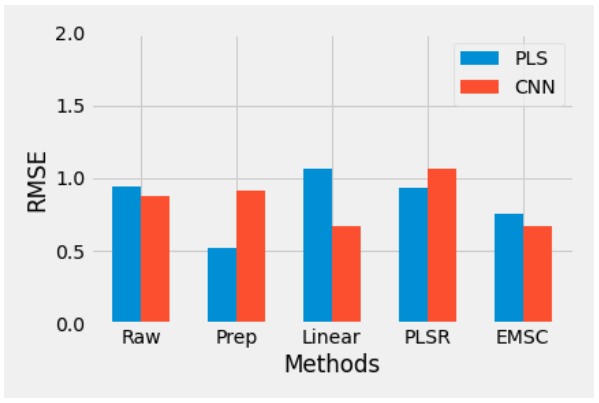


Figure 25: R2 scores for all Augmentation methods collected from the final experiment

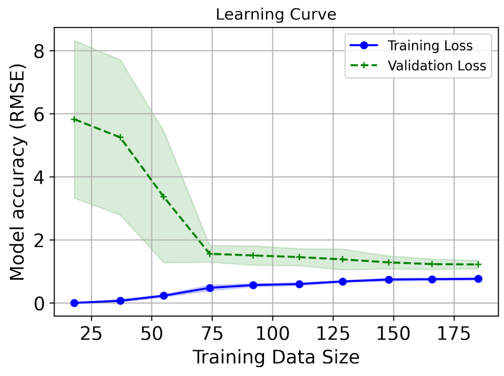
the linear augmentation method and the EMSC augmentation method and showed a RMSE value of 0.67 and 0.62 respectively. The method that worked best for both PLSR and CNN models was the EMSC augmentation method. The models trained on data augmented using the EMSC method showed an improvement in performance compared to raw data models.

4.3 Learning Curves

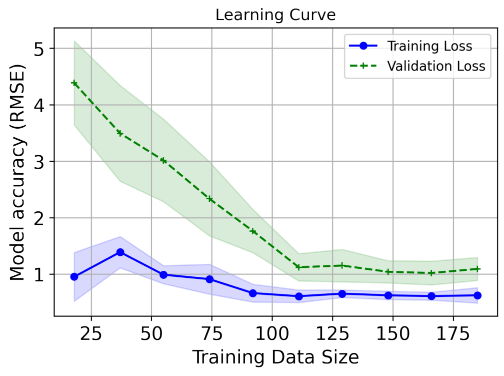
This section shows the results of the learning curve experiments.

4.3.1 Learning Curves for Raw Data

The section shows the learning curve plots for PLSR and CNN models based on raw data. Both PLSR and CNN models used in this experiment were the same models that were used in the benchmark experiments. Figure 26a shows the learning curves generated using PLSR model and figure 26b shows learning curves for CNN model. The model accuracy metric used was RMSE. Figure 26a shows a good PLSR model fit on raw data. The gap between the validation and training loss is small when the optimal sample size is reached which shows that the model has generalized well. Furthermore, both training and validation loss trends seem stable when the sample size is greater than 150 and this is a sign of the stability of the model. Figure 26b also shows good learning curve trend for the CNN model with generalization gap becoming small when the sample size reaches 110 approximately and the training loss is stabilizing towards the end. However, the validation loss is not as stable as the training loss. Furthermore, the transparent colored shaded regions around the loss trends lines in the figures show the standard deviation of the RMSE values and this appears because of the Kfold cross validation performed for each sample size. The standard deviation for both models is small which shows consistency is performance.

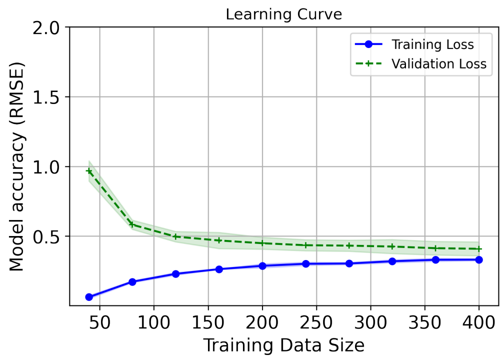


(a) PLSR

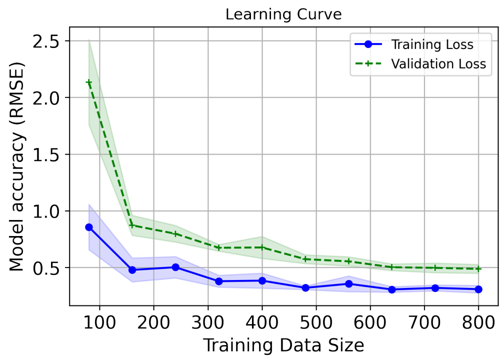


(b) CNN

Figure 26: Learning curves on Raw data



(a) PLSR



(b) CNN

Figure 27: Learning curves on data augmented using linear augmentation method

4.3.2 Learning curves for data generated using linear augmentation method

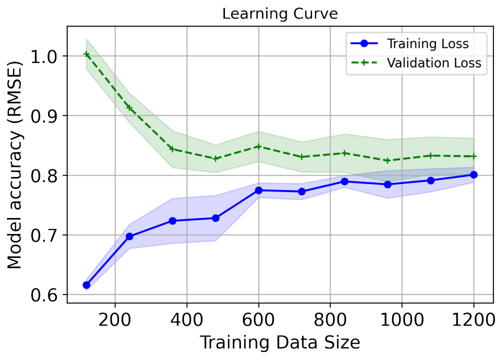
This section shows the learning curves for PLSR and CNN models when trained on data augmented using the linear augmentation method. The parameters used in this experiment for generating samples were the same parameters that were used for evaluating the linear augmentation experiment. Learning curves for PLSR model and CNN model on artificially generated data are visualized in figure 27a and figure 27b respectively. Both figures show that the loss trend seem to stabilize and the generalization gap is also small when the sample size is large enough. The standard deviations of the accuracy is also small that is a sign of consistency in performance.

4.3.3 Learning curves for data augmented using PLSR augmentation method

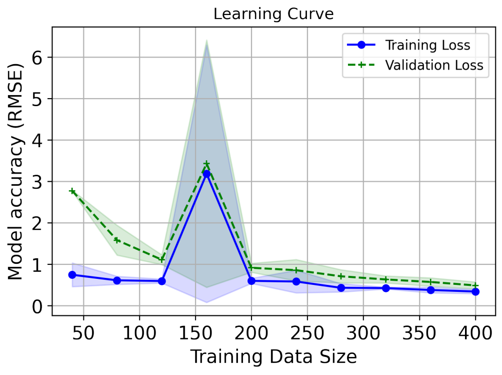
This section shows the learning curves for PLSR and CNN models when the data used for training was augmented using the PLSR augmentation method. Similar to linear augmentation method the parameters used were the best parameters that were determined in the PLSR augmentation evaluation experiment. Figure 28a and figure 28b show the learning curves for PLSR and CNN model respectively. Both learning curves figures show that the generalization gap becomes small when the sample size is large enough and the loss trend also stabilizes towards the end. An interesting behavior that can be observed in the CNN learning curve figure is the big bump when the training size is just above 150. The standard deviation is also huge for this bump. This bump is interesting because it is not according to the trend as the loss before and after this bump is completely different.

4.3.4 Learning curves for data augmented using EMSC augmentation method

The best parameters determined in the EMSC augmentation evaluation experiment were used to augment data and analyze the learning curves for PLSR and CNN model on EMSC augmented data. Figure 29a and figure 29b show the learning curves for PLSR and CNN model respectively. Both figures show good trend for the loss function. The gap between the train and validation loss trend is small which means that the model has generalized. Furthermore, the loss trend is stable towards the

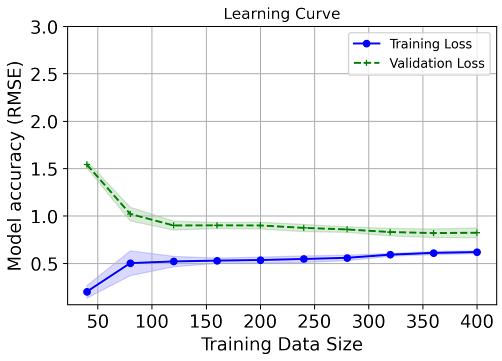


(a) PLSR

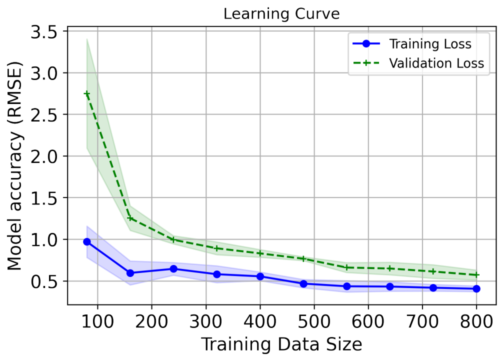


(b) CNN

Figure 28: Learning curves on data augmented using PLSR augmentation method



(a) PLSR



(b) CNN

Figure 29: Learning curves on data augmented using EMSC augmentation method

end and the standard deviation of the accuracy is small which implies that the model is consistent and stable.

5 Discussion

In this section the interpretations, implications and the limitations of the results provided in the results section are discussed in detail.

5.0.1 Small Data set

All artificially intelligent models irrespective of its type are driven by data. The performance of these models are totally dependent on the quality and quantity of the data. Applications of machine learning are growing every day and machine learning is now being used in very diverse fields. The increase in applications comes with challenges of its own and one of these challenges is the collection of data. Small sample size or insufficient data are one of the most common challenges faced when implementing machine learning solutions. Data collection can be a costly process as it may require a lot of time and effort. Spectroscopy is an example of the domain in which it is common to have a small data set size and generating new data is a costly process. Data augmentation may be good handy technique to solve the limitations of a small data set size. Applying transformations in the existing data can allow to augment new sample and reduce the operational cost of collecting more data. Although augmentation is beneficial in increasing the sample size but still it cannot replace the process of collecting new samples. This is because augmented sample generated as a result of applying transformations will fail to add any type of new information in the data. The augmented samples can help the models to generalize better on the existing data but it will not help the model to learn any anything new. Therefore if the target is to make the model perform better for existing data then augmentation can be very useful. Some of the benefits of data augmentation are listed below:

- Improving model prediction accuracy
  - Increase the amount of training data for models
  - Prevent data scarcity for better models
  - Reduce over fitting and make more generalized models
  - Help resolve class imbalance issues for classification problems
- Reduce cost of collecting and labelling data
- Enable rare event prediction

Although augmentation is beneficial but it has its own challenges as well. Some of which are:

- Domain knowledge and developing new research to create data for applications.
- Evaluation of a augmentation method and assessing the quality of data is required
- If the real data set has problems then the augmented data may also contain the same problems. For example if the original data is biased then the augmented data will also suffer from biasness. Therefore, identification of an optimal strategy for augmentation is important.

When working with small data sets the model selection and evaluation phase becomes a bit tricky. This is because they are not enough samples to train the models to learn all variation in the data and the model validation process becomes even harder. If the data set is small then splitting the data set into training and test sets for validating the results becomes complicated. While splitting data sets, it is important to have enough samples for training and leaving enough samples for validation. In this case, especially the validation process becomes really important because if there is not enough variability in the validation set then over fitting in inevitable. Furthermore, a lot of variability present in the data is also not beneficial when the sample size is small. In this case, the splitting process of data becomes extremely important. The Raman data set used in this study was quite heterogeneous as the data was divided into different groups which were based on the unique biological samples. Working with mean data may have been a bit beneficial but it resulted in a significant decrease in the sample size as well.

ANNs generally require more amounts of data for training compared to linear models. This is simply because of the fact that ANNs have to train a lot of parameters. The weight updates are performed until the gradient finds a good local minima of the loss function. Determining the best local minima is a complicated process. The problem is that the model starts off with a poor initial state and then with some gradient based optimization it converges the network to an optimal solution, which might not necessarily be the global optimum but a good local minima. So, accomplishing the task of finding the best local minima requires as much convergence as possible and this requires a larger sample size. PLSR extracts new features by maximizing the covariance between the features and the targets. All the features explain a fraction of variation in the data and so the variation present in the data can be explained by them. The feature extraction process reduces the number of features significantly with respect to the number of samples thus PLSR can work for data sets with small number of samples as well. However, the number of samples is still an important factor and increasing the number of samples can actually induce more variability and may result in a more robust model.

Having known the benefits, challenges, motivation for optimal sample size and limitations of the spectroscopy data, data augmentation was also practiced in this study and it is one of the main focus of this study. To evaluate the augmentation methods, the performance of models on raw data, pre-processed data and data generated using different three different augmentation methods are compared and discussed further in this section.

5.1 Comparison of PLSR and CNN models based on Raw data

The results provided in section 4.1 show that the CNN model performs better than PLSR model when the raw data set is used. However, the scores are not the only factors that are enough to conclude that which model is better. After all it can be argued that the recorded R2 and RMSE values for both models were not to much apart from each other.

5.1.1 Hyper parameter tuning

Tuning a model is usually the most time consuming part when performing regression or classification tasks using machine learning. Tuning is usually a trial and error process in which different set of hyper parameters are tested and compared in order to enhance model accuracy. The hyper parameters tend to change depending on a particular data set and problem. Hyper parameter tuning for models was also a time consuming task in this study. For PLSR model, number of PLS components was the hyper parameter that was tuned. Different options of number of PLS components was tested and the ideal number of PLS components was selected based on the cross validation scores.

Regarding the hyper parameter selection phase for the CNN model, parameters such as learning rate, batch size, activation function and optimizer algorithm are required to be determined. These parameters affect the speed of convergence and the value of converged loss function. Ideally the best set of hyper parameters are found by trying all sets of parameters using grid search algorithm like it was done for the PLSR model. But, this process quickly became overwhelming when trying different choices of architectures. Therefore, the best CNN model was selected based on trail and error. This is normally the case with ANNs as the architectural design is different for different natures of problems.

Tuning process for the CNN model was relatively slow because it required training a number of models. Another complication during this process was that ANNs are sensitive to weight initialization. The weights are initialized randomly at the start of training process and they tend to change for each repetition. Combining this complication with the parallelism property of ANNs on GPUs, it becomes impossible to obtain reproducible results. Furthermore, due to the mentioned properties of ANNs the credibility of the model cannot be judged based on a single run. Multiple runs have to be performed to judge the predictive power of the model. The knowledge gained through experimentation is surely beneficial and it can expected to setup new architectures for models faster but even with the experience it will still be a laborious task.

5.1.2 Efforts Required

Setting up the flow of the PLSR experiment was a straightforward task and it did not require a lot of efforts. All packages used in this experiment were available in the scikit learn library. Most of the parameters values were assumed fixed in advance as they were not required for this analysis. The only parameter that needed tuning was the number of PLS components and setting up for that was not a big task as well. Another great advantage for the PLSR model were the deterministic results (same results if run again with same conditions). This effect should not be under estimated when one also has to work with ANNs as well. It is possible to obtain consistent results with ANNs but this leads to a significant increase in the training time. However, the desire of getting deterministic results using ANNs can be argued. This is because ANNs try to find the best local minima for loss function and it may be that there might be another minima that is better than the previous one which can enhance the model performance. Therefore, if the ANN is configured to get deterministic results then the fact that there might be another better minima may not be explored.

A lot of effort had to be put into setting up experiments for the CNN model. A big part of these efforts went into testing different architectural choices for the CNN model. Different alternate architectures were tested and evaluated. The depth of model, number of layers, number of neurons and the effects of different methods like dropout, batch normalization and pooling were tested. This model selection task is also complicated because the target to achieve is to come up with a good efficient model and not just an architecture that produces good results but is computationally very expensive. Furthermore, another thing to make sure is that the models contain sufficient complexity to account for the complex structures that might be present in the data without being prone to over fitting. After spending a huge amount of time on the CNN model selection phase, finally some confidence in what choices were likely to work well with the Raman data set was developed and a model architecture was selected that checked all check boxes.

5.1.3 Model training time

Data set used for analysis in this study was not particularly large in size. Average of the original raw data set was taken based on the sample group value in order to remove unusual peaks from the data [3.1]. This process significantly reduced the sample size and the mean raw data set was based on only 232 samples.

Linear models are generally swift and they take relatively less time to train compared to the deep learning models. The CNN architecture used in this study for experimentation was relatively shallow compared to the common architectures of deep learning models. The architecture of the model was based on a single convolutional and a single fully connected dense layer. Therefore the training time for a single CNN model was particularly not time consuming at all. The average time taken for training the CNN model on raw data was 35 seconds and 484 epochs. However this time is also dependent on the way the training is executed. For example, the run time of 35 seconds was achieved by using the GPUs. However if the training was performed without GPUs then this training time might have increased significantly. Furthermore, the training time of the CNNs also tend to increase with the increase in sample size. For example, if the training time of the CNN model on raw data when the sample size was 232 is compared with the training time of the CNN model on EMSC augmented data when the sample size was 1000 then a significant increase in the number of epochs can be noticed. The CNN model with a 1000 samples took an average of 711 epochs.

It is true that the time of training for PLSR model was always significantly low compared to CNN model but it is also true that CNN outperformed PLSR in accuracy when the training was performed on raw data. It can be argued that the difference in accuracy is not significant but this statement is no different than stating that the training time difference factor is not significant as 35 seconds is also not a huge value. However, the decision of picking a model as favorite is dependent on the application and the target to achieve. If the target is to achieve the best accuracy then a compromise on the training time can be made. Similarly, if the target is efficiency then a sacrifice can be made for that small difference in the accuracy.

5.2 Preprocessing

EMSC[2.8] was the method that was used for preprocessing the Raman spectroscopic data. EMSC is a method specifically designed to preprocess spectroscopic data and it is also well suited for correcting various effects in the Raman spectroscopic data. EMSC is a model based preprocessing technique and it has the ability to store the parameter values of the corrections which can be used for further analysis. These parameters can reveal systematic variations in the samples and capture various effects that are interesting in themselves. The amount of baseline shift of each sample spectrum is an example of parameters holding interesting information about the samples.

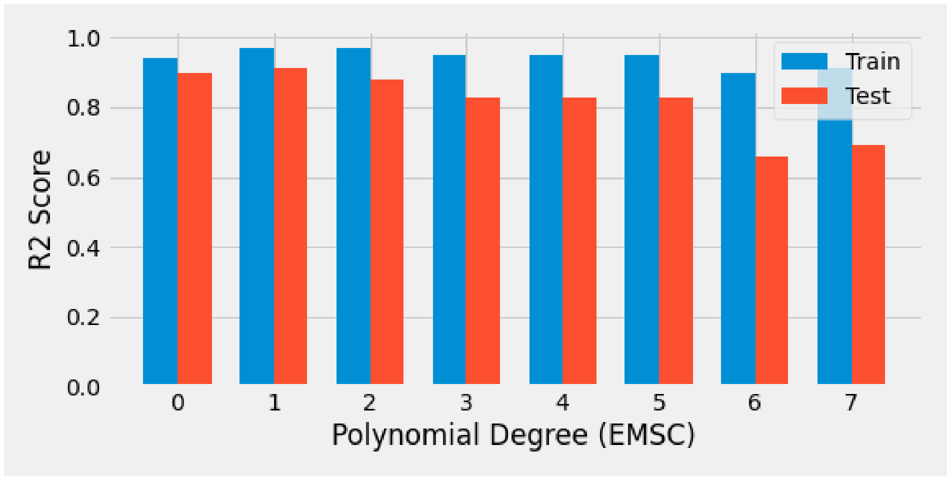


Figure 30: Performance comparison for CNN model on train and test data preprocessed using EMSC with different Polynomial Degrees

PLSR model trained on preprocessed data resulted in a significant increase in the performance. The increase in performance of the PLSR model proves that the EMSC method is well suited for sorting various effects and noise in Raman spectra and it cleanses data for visual and analytical use. Even the lowest polynomial degree results in an increase in the performance which also shows how effective the EMSC method can be.

Studies have shown that ANNs have the ability to learn important features and produce better results compared to the PLSR model even without preprocessing [9]. ANNs make their own feature space and learn in their own way. Every layer in the ANNs except the input and the output layer is basically used for transforming features to space where it is possible to capture the necessary details. Even in CNNs, all layers before the output including the convolutional layer is used for feature transformation. This is how CNN models do their own feature extraction. The results of CNN model on preprocessed data are not according to the trend showed by PLSR model. CNN show no increase in performance with preprocessed data rather there is a tiny decrease in the RMSE score. The best result is achieved when the data was preprocessed with a polynomial degree of 1 and these results were approximately equal to the raw data results. Furthermore, applying the preprocessing makes PLSR the better model than the CNN. Another interesting pattern that is shown by CNN model is that increasing the polynomial degree decreases the RMSE values. The behavior is visualized in figure 18.

A possible reason for the results displayed by CNN model on preprocessed data could be that the model is simply too complex. The CNN model used in this study for experimentation has 6,042,721 trainable parameters. The selection of the model architecture was mainly based on evaluation on raw data. The process of preprocessing with EMSC method removes most of the noisy unwanted effects from the data. So, this could be a reason that a more simpler model with less trainable parameters is enough to learn all variations in preprocessed data. In order to test this hypothesis, the performance of CNN model evaluated on preprocessed training data was compared to performance of CNN model evaluated on preprocessed test data. This comparison is visualized in figure 30. The figure shows a decreasing trend for both training and test data when the polynomial degree rises. Therefore, over fitting might not be the reason for the decreasing performance of the CNN on preprocessed data.

5.2.1 Effect of preprocessing on CNN training process

Results stated in section 4.1 show that preprocessing data using EMSC with polynomial degree 1 and training the CNN model on it produces approximately equal results as training the CNN on raw data. However, the results also showed that the average epochs required for training CNN on preprocessed data(130) is significantly smaller than the number of epochs required for training on raw data(484). It is known that EMSC removes noisy unwanted effects from the raw spectroscopic data. So, in theory EMSC preprocessing would make it easier for the model to learn the variation present in the data as the variation due to noise will no longer be present. Therefore, preprocessing may have actually made the training process less complicated.

5.3 Data Augmentation

Linear augmentation, PLSR augmentation and EMSC augmentation were the three data augmentation techniques that were tested in this study. One of the objective of the study was to evaluate the three augmentation techniques and to test if these techniques are good alternatives for producing new observations. The results of the experimentation showed that using the data augmentation and increasing the sample size is beneficial as it improves the performance of the CNN. The fact that it seems possible to improve neural networks using augmentation is interesting. Furthermore, it is important to note that neural networks might not be the best model to use on all problems even with an optimal sample size. The PLSR model with preprocessed data produced the best overall performance on the data set used in this study.

5.3.1 Linear Augmentation

First step of the experiment showed varying results for PLSR and CNN models. If the results in figure 19 are observed then a trend of increase in the RMSE value is seen with the increase in number of linearly combined rows for the CNN model. The difference in the distribution of range of response values due to different number of linearly combined rows seem to be an important factor for CNN model. The CNN model gives the best RMSE value when 2 rows are linearly combined for generating new samples. This could be because the range of response values is large when 2 rows are linearly combined and when the range of the response values is large, the chances of error by the model reduces and thus the low RMSE scores. However, as the range of response values shrinks like in the case of 5 and 10 linearly combined rows, the margin of error decreases and according to the CNN results this increases the RMSE values. However, the PLSR model results do not follow the same trend. The best RMSE value for PLSR model is obtained when the number of linearly combined rows is 5. Difference in the results trend for PLSR and CNN is interesting. Second step of the experiment show that increasing the sample size enhances the performance of the CNN model. Enhancement in performance of the CNN model also points that the linear augmentation method has the ability to be an effective method for the CNN models. However, the same cannot be said based on the results of the PLSR model. PLSR model gives the best RMSE when the sample size is 500 but the enhancement in performance is not significant compared to the CNN model. Results of the final evaluation experiment for the CNN model were quite promising. Augmenting data using the linear augmentation method and then evaluating the CNN model showed a significant increase in the performance compared to the raw data scores. This may very well be the the result of the increased number of samples as the augmentation does not introduce any new variation in the data. Furthermore, if the linear augmentation method is an effective method then the results also correspond to the theory that CNN models require larger amounts of data to excel.

Results for PLSR model on linearly augmented data are not as promising as CNN. PLSR model showed worse results for linearly augmented data when compared with raw data. This is an interesting behavior because linear augmentation does not add or remove any variation present in the data. Therefore, a plausible reason for this behavior could be the parameter choices for the linear augmentation method since the parameters were different for PLSR and CNN models. In order to better understand the behavior of the PLSR model the results of the training data were also analyzed. The model showed a R2 score of 0.96 and a RMSE value of 0.33 when evaluated on training data while the R2 and RMSE values for test data were 0.88 and 1.07 respectively. These results are significantly better than what was observed for test data. The significant difference in the results of the training and test data is a clear sign of over fitting. A possible reason of the bad performance on test data could be related to the effect of the number of samples used in the linear combinations. The cross validation experiment performed in the first step of the linear augmentation showed that the optimal number of rows for making linear combinations and generating new samples is 5 for PLSR model and 2 for CNN model. Therefore, the difference in the behavior for CNN and PLSR model and the over fitting of the PLSR model when evaluated on test may be due to the narrow range of distribution of response values as a result of linearly combining 5 rows. Furthermore, the learning curve plot visualized in figure 27a shows a validation accuracy of 0.6 which gave some confidence to the previous statement. The narrow distribution response issue can be easily solved by adding the original data samples to the artificially generated data set. Therefore, an experiment was conducted in which the PLSR model was trained on a data set that was the concatenated version of artificially generated samples and the original data samples. The concatenated data set had the same response distribution as the original data set. The PLSR was then evaluated on test data and showed a R2 score of 0.90 and a RMSE score of 0.94. The results are similar to the results observed for raw data experiment and this shows that the linear augmentation method was successful for both PLSR and CNN models.

5.4 PLSR Augmentation

Results of the experiment for the PLSR augmentation method presented in section 4.2.2 show promise for the augmentation technique when the results of the PLSR model are observed. PLSR model displayed approximately equal results for the augmentation data compared to the raw data. However, the results for CNN model were not as promising. CNN model showed worse results for augmented data compared to PLSR model and the results of CNN model on raw data. In fact, the difference in performance was quite significant. In order to analyze the performance of CNN model the results of the CNN were thoroughly analyzed. The CNN model showed an average R2 score of 0.98 and a RMSE of 0.52 when evaluated on training data and an average R2 score of 0.87 and RMSE of 1.11 on test data. There is a big difference between the train and test performance and this shows signs of over fitting. However, it is a difficult question to ask the reason for the decrease in performance for CNN model. Even though there are signs of over fitting it cannot be be claimed that only over fitting is the reason for the bad performance of the CNN model and even if over fitting is the reason then what caused the model to overfit is the real question.

There can be a number of possible reasons for the decrease in performance for the CNN model. A plausible reason for the not promising results of the CNN on test data could be the augmentation method it self. However another interesting factor to observe here is the learning curves plot for PLSR augmented data visualized in figure 28b. It was a bit surprising to see that the PLSR augmentation for CNN gave an RMSE value of 1.11 when evaluated on test data but the validation loss in the learning curve plot is approximately 0.6. This difference in these results is interesting and it might mean that the PLSR augmentation method is not that bad after all and it has potential to be a good augmentation method. However, the amount of experimentation and tests performed in this study are not enough to make claims and to state the reason of different results observed in the test data experiment and learning curves. Furthermore, the difference in the behavior of CNN and PLSR may be because of the difference in nature of both models i.e. PLSR is a linear model and CNN is a non linear model and they have their own ways of working.

5.5 EMSC Augmentation

Results of the EMSC augmentation method experiment show promise for the EMSC augmentation method. The results also show that increasing the sample size was beneficial for the models. Both PLSR and CNN model showed an increase in performance when they were trained and evaluated on data set generated using EMSC augmentation method. Both models showed better results than the results observed in the raw data experiment. However, this does not mean that EMSC augmentation add some new variation in the data. EMSC augmentation method works by reversing the effects of EMSC preprocessing using the parameter values that were recorded during the preprocessing process. New random parameters are produced and these new parameters are normally distributed based on the mean and standard deviation of the original parameters. The performance of the models on artificially generated data is dependent on the new randomly generated parameters. Sometimes the random values may result in inducing better variance in the data and sometime they may not. However, the overall performance should always be fluctuating around the values that were observed in the raw data experiment.

5.6 Learning Curves

There is a lot of information that can be extracted form the learning curves provided in section 4.3. First of all lets talk about the learning curves of the PLSR model. All the learning curves of PLSR model shows that the training RMSE starts from a point that is at zero or approximately very close to zero. As the training size increases the RMSE starts to increase. This might seem like a very strange behavior but it is actually quite logical. In each figure it can be observed that when the training RMSE is zero or very close to zero, the validation RMSE is at its highest value. At the first data point in the PLSR plots when the training RMSE is extremely small and the validation RMSE is extremely large, the size of the data set is extremely small. When the data is extremely small, the model has no problem in fitting the data so it does that perfectly or with very little error. However, when the model fitted on extremely small data set tries to make prediction on the validation set, the RMSE shoots up. This is because the model is not mature enough to make good predictions on validation set as it has not learned all the variation present in the data. The small training data set does not provide the model with all the necessary variation present in the data. However, aside from this when the data set size is increased it is observed that validation error starts to decrease and training error starts increasing. Because, now the model is able to learn the complex variation present in the data and now it is unable to fit the data perfectly even for the train data. In the last few points of the plots a nice trend is observed for each PLSR model. When the data size is large enough the difference between the train and validation score becomes small, the space between the validation and training lines is narrowed down and both train and validation loss trends become stable. This shows that at that specific sample size point, the model fits the data well and the model is generalized.

The learning curves for the CNN models show a nice trend. It is observed in all the plots that the training and validation RMSE gets better with the increase in the data size. Furthermore, the generalization gap between the validation and training loss becomes small at the end and both loss trends are approximately flat and stable near the end which shows that the models are stable and generalized. The loss trends for augmented data where better stabilized compared to the raw data loss trends which showed some fluctuations in the RMSE towards the end. This shows that the model stabilized better when the sample size was larger. In general, the learning curves plots show that all techniques were successful in enhancing the performance of CNN. Some augmentation methods require more samples to show best performance but all methods showed an approximate RMSE of 0.5 while the raw data learning curves with just 175 samples showed an approximate RMSE of just 0.9. This shows the potential of the augmentation methods and the effects of a larger sample size. Furthermore, an interesting behavior was observed in figure 28b. The figure shows that both training and validation loss experience a huge bump and the RMSE value shoots. This behavior is not according to the preceding and succeeding trend. A possible reason for this event can be the training process of the ANNs. ANNs tend to converge to optimal local minima in search of finding the global minima. As there are multiple local minima present sometimes the model might converge towards a bad local minima and may never recover. Furthermore, the use of early stopping will only make the recovery task harder for the model because if the model is not able to improve the loss within the specified number of epoch then the training will terminate. Therefore, the reason of the huge bump is loss observed in figure 28b could be due to the reason that the CNN model converged towards a bad minima and was not able to recover.

5.7 ANN and Robust Performance

One of the targets while developing artificially intelligent models is to develop a model that is robust. Robust in machine learning terms is a property of a model to produce consistent results and it characterizes how the model performs for new independent data. The performance of the ANN depends on the ability of the model to find the best local minima. The performance of ANNs vary for each training run due to the fact that there are a number of local minima present and the model may not be able to find the same minima every time. Random initialization of weights for each training run is also a factor for this varying performance. It is possible to restrict the ANNs to get deterministic results but again the question is whether it is in the best interest. ANNs are non linear models that generalize on data in high dimensional spaces. In search of the global minima, ANNs can find a optimal local minima and the convergence of the model is based on that. Due to the presence of multiple local minima the model cannot know that which minima is the best. Even though it performs well on a particular minima there might be that a better one exists and on the next training run the model may be able to converge towards it. Therefore, if the model is restricted to have deterministic results then it is not possible to have the ability to converge towards a more optimal solution. Furthermore, the non deterministic results can make it difficult to evaluate the robustness of a model. Therefore, concrete statements cannot be made on the performance of a model based on one or two evaluation results. Even in this study the results for the CNN model are based on the average of results of five repetition of the experiment.

While training ANNs, an interesting factor to discuss is that whether the fluctuating results of the model are significant or not. By significant it means that whether the difference in result is due the property of the model converging towards a different minima or it has something to do with the data. For instance a model can perform bad on the same data it performed well on during different training experiment executions. The difference in performance may be based on the possibility that there is some complex variation present in the data and the model is not complex enough to capture it. Another possible reason could be that the model was just unlucky to converge towards a bad minima and it was just not able to recover from that state. An example of this was observed in the learning curves experiment for the PLSR augmentation method.

Furthermore, the generalization of an ANN is also affected by the sample size. if the data size is not large enough then the fluctuations in the results are more because the model was not able to generalize well. This factor was also observed in this study. The results of all 5 executions of the raw data experiment for CNN model were reviewed and it was observed that the minimum RMSE out of all 5 values was 0.82 and the maximum was 1.03 (Average: 0.88). In order to test the hypothesis of the model generalizing better with a large enough sample size, the results of all 5 executions of the EMSC evaluation experiment were also observed. The minimum RMSE value observed for the augmented data experiment was 0.57 and the maximum was 0.68 (Average: 0.62). So, not only did the performance of the model increased by adding more samples through augmentation but the model generalized better as the difference between the maximum and minimum RMSE values decreased.

6 Conclusion

In this study, three different augmentation methods i.e. linear augmentation, PLSR augmentation and EMSC augmentation were developed and tested on a CNN. The purpose was to study the performance of deep learning models on data sets that contain more features than samples and to observe if increasing sample size using data augmentation leads to any improvement in performance of the models. The results of the experimentation performed in this study showed that using augmentation techniques to increase sample size did improve the performance of the CNN model. EMSC augmentation method proved to be best performing augmentation method in this study as the data augmented using this method resulted in the most enhancement of the model performance. The linear augmentation method also showed great promise and resulted in a significant enhancement of performance for the CNN. In terms of comparison, the PLSR augmentation came last out of the three methods and the evaluation results of this method showed no signs of performance enhancement rather it showed a decrease in result values. Furthermore, the effect of increasing sample size using the augmentation methods was also studied using learning curves. The learning curves plots showed that all augmentation methods were successful and managed to enhance CNN performance. It is true that there was difference in the sample size requirement for the different augmentation methods but all methods managed to show a better RMSE value for the CNN when compared to the raw data which only had 175 samples. The difference in performance observed for the PLSR augmentation in the learning curves plot and the evaluation experiment on original test data is interesting. The approximate validation RMSE value of 0.6 observed in the learning curve plot shows some hope and potential for PLSR augmentation method. The reason for the PLSR failing on the test data is not clear and requires further testing.

Based on the results observed in this study and the results reported in other studies it can be concluded that ANNs possess the ability of outperforming linear models but generalizing this statement and claiming that ANNs are always better is not correct. PLSR model with preprocessed data proved to be the best model in this study. ANNs managed to outperform PLSR for raw data but this came at the cost of longer training times and a greater amount of effort spent on model architecture and parameter selection. The choice of model depends upon the end requirements and the problem at hand. Furthermore, increasing the number of samples using augmentation can improve the performance and produce better ANN models. The performance of CNN enhanced and the model showed more stability when the sample size was larger for the augmentation methods compared to the small sample size of raw data. Data augmentation can be useful technique for increasing sample size and enhancing model performance but an in depth analysis and evaluation of the methods used for augmentation is required.

References

[1] Pan, Liangrui Zhang, Peng Daengngam, Chalongrat Peng, Shaoliang Chongcheawchamnan, Mitchai. (2021). A review of artificial intelligence methods combined with Raman spectroscopy to identify the composition of substances. Journal of Raman Spectroscopy. 10.1002/jrs.6225.

[2] Fine, J. A., Rajasekar, A. A., Jethava, K. P., Chopra, G. (2020). Spectral deep learning for prediction and prospective validation of functional groups. Chem. Sci., 11(18), 4618–4630.

[3] Bian, X. (2022). Spectral Dimensionality Reduction Methods. In Chemometric Methods in Analytical Spectroscopy Technology (pp. 209–236). Springer Nature Singapore.

[4] Sen, A., Sen, B. (2013). On Testing Independence and Goodness-of-fit in Linear Models. arXiv.

[5] Blazhko, U., Shapaval, V., Kovalev, V., Kohler, A. (2021). Comparison of augmentation and preprocessing for deep learning and chemometric classification of infrared spectra. Chemometrics and Intelligent Laboratory Systems, 215, 104367.

[6] Wu M, Wang S, Pan S, Terentis AC, Strasswimmer J, Zhu X. Deep learning data augmentation for Raman spectroscopy cancer tissue classification. Sci Rep. 2021 Dec 13;11(1):23842. doi: 10.1038/s41598-021-02687-0. PMID: 34903743; PMCID: PMC8668947.

[7] Oord, A. van den, Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K. (2016). WaveNet: A Generative Model for Raw Audio. arXiv.

[8] Liu, J., Osadchy, M., Ashton, L., Foster, M., Solomon, C. J., Gibson, S. J. (2017). Deep convolutional neural networks for Raman spectrum recognition: a unified solution. Analyst, 142(21), 4067–4074.

[9] Acquarelli, J., van Laarhoven, T., Gerretzen, J., Tran, T. N., Buydens, L. M. C., Marchiori, E. (2017). Convolutional neural networks for vibrational spectroscopic data analysis. Analytica Chimica Acta, 954, 22–31.

[10] Long, D.A. (1977), Raman Spectroscopy, New York 1-12.

[11] Das, R. S., Agrawal, Y. K. (2011). Raman spectroscopy: Recent advancements, techniques and applications. Vibrational Spectroscopy, 57(2), 163–176.

[12] Potocki L, Depciuch J, Kuna E, Worek M, Lewinska A, Wnuk M. FTIR and Raman Spectroscopy-Based Biochemical Profiling Reflects Genomic Diversity of Clinical Candida Isolates That May Be Useful for Diagnosis and Targeted Therapy of Candidiasis. Int J Mol Sci. 2019 Feb 25;20(4):988.

[13] Haynes, C. L., McFarland, A. D., Van Duyne, R. P. (2005). Surface-Enhanced Raman Spectroscopy. Analytical Chemistry, 77(17), 338 A-346 A.

[14] Kuhar, N., Sil, S., Verma, T., Umapathy, S. (2018). Challenges in application of Raman spectroscopy to biology and materials. RSC Adv., 8(46), 25888–25908.

[15] Baker, M., Trevisan, J., Bassan, P. et al. Using Fourier transform IR spectroscopy to analyze biological materials. Nat Protoc 9, 1771–1791 (2014)

[16] Kudelski, A. (2008). Analytical applications of Raman spectroscopy. Talanta, 76(1), 1–8.

[17] Zhang, X., Xiao, K., Dong, C., Wu, J., Li, X., Huang, Y. (2011). In situ Raman spectroscopy study of corrosion products on the surface of carbon steel in solution containing Cl and. Engineering Failure Analysis, 18(8), 1981-1989.

[18] Abdi H, Williams LJ. Partial least squares methods: partial least squares correlation and partial least square regression. Methods Mol Biol. 2013;930 549-579.

[19] Wold, H. (1966). Estimation of principal components and related models by iterative least squares. In P. R. Krishnajah (Ed.), Multivariate analysis (pp. 391-420). NewYork: Academic Press.

[20] Wold, S., Sjöström, M., Eriksson, L. (2001). PLS-regression: a basic tool of chemometrics. Chemometrics and Intelligent Laboratory Systems, 58(2), 109–130.

[21] Mishra, S., Sarkar, U., Taraphder, S., Datta, S., Swain, D., Saikhom, R., Panda, S., Laishram, M. (2017). Principal Component Analysis. International Journal of Livestock Research, 1.

[22] McCulloch, W.S., Pitts, W. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics 5, 115–133 (1943).

[23] Shaw, G.L. (1986). Donald Hebb: The Organization of Behavior. In: Palm, G., Aertsen, A. (eds) Brain Theory. Springer, Berlin, Heidelberg.

[24] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), 386–408.

[25] Kingma, D. P., Ba, J. (2014). Adam: A Method for Stochastic Optimization. arXiv. https://doi.org/10.48550/ARXIV.1412.6980

[26] Raschka, Sebastian and Mirjalili, Vahid (2019) *Python Machine Learning* 3rd Ed, Packt Publishing

[27] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, 15(56), 1929–1958.

[28] Wager, S., Wang, S., Liang, P. (2013). Dropout Training as Adaptive Regularization. arXiv. https://doi.org/10.48550/ARXIV.1307.1493

[29] Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep Learning. MIT Press.

[30] Ioffe, S., Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv. https://doi.org/10.48550/ARXIV.1502.03167

[31] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[32] Fukushima, K., Miyake, S. (1982). Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. Pattern Recognit., 15(6), 455–469.

[33] Aurélien Géron. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow : concepts, tools, and techniques to build intelligent systems. eng. Second Edition. Sebastopol, CA: O'Reilly, 2019. isbn: 9781492032649.

[34] R. Venkatesan and B. Li. Convolutional Neural Networks in Visual Computing: A Concise Guide. Data-enabled engineering. CRC Press, 2018. isbn:9781138747951.

[35] Engen, Martin; Sandø, Erik; Sjølander, Benjamin Lucas Oscar; Arenberg, Simon; Gupta, Rashmi; Goodwin, Morten, Farm-Scale Crop Yield Prediction from Multi-Temporal Data Using Deep Hybrid Neural Networks 2021, https://hdl.handle.net/11250/2980996

[36] Stark, E. W., Martens, H., Vegen, G., G., M., Hieftje, Hirschfeld, Tomas., , S. H., Jensen. (2017). Multivariate Linearity Transformations for Near-Infrared Reflectance Spectrometry -.

[37] Martens, H., Stark, E. (1991). Extended multiplicative signal correction and spectral interference subtraction: New preprocessing methods for near infrared spectroscopy. Journal of Pharmaceutical and Biomedical Analysis, 9(8), 625–635.

[38] Liland, K. H., Kohler, A., and Afseth, N. K. (2016) Model-based pre-processing in Raman spectroscopy of biological samples. J. Raman Spectrosc., 47: 643– 650. doi: 10.1002/jrs.4886.

[39] Kennard, R. W., L. A. Stone. (1969). Computer Aided Design of Experiments. Technometrics, 11(1), 137–148. https://doi.org/10.1080/00401706.1969.10490666

[40] kennard-stone. (n.d.). https://pypi.org/project/kennard-stone, Accessed: 2022-03-02.

[41] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B. & Varoquaux, G. API design for machine learning software: experiences from the scikit-learn project. *ECML PKDD Workshop: Languages For Data Mining And Machine Learning.* pp. 108-122 (2013)

[42] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. & Zheng, X. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015), https://www.tensorflow.org/, Software available from tensorflow.org

[43] Helin, R, Indahl, UG, Tomic, O, Liland, KH. On the possible benefits of deep learning for spectral preprocessing. Journal of Chemometrics. 2022; 36(2):e3374.

[44] Tegegn, D. D., Zoppis, F., Manzoni, S., Sas, C., Lotti, E. (2021, February). Convolutional Neural Networks for Quantitative Prediction of Different Organic Materials Using Near-Infrared Spectrum.

[45] Bjerrum, E. J. (n.d.). Deep Chemometrics: Deep Learning for Spectroscopy. https://www.cheminformania.com/deep-chemometrics-deep-learning-for-spectroscopy Accessed: 2022-02-15.

[46] Kawamura, K., Nishigaki, T., Andriamananjara, A., Rakotonindrina, H., Tsujimoto, Y., Moritsuka, N., Rabenarivo, M., Razafimbelo, T. (2021). Using a One-Dimensional Convolutional Neural Network on Visible and Near-Infrared Spectroscopy to Improve Soil Phosphorus Prediction in Madagascar. Remote Sensing, 13(8).

[47] DePaoli, D.T., Tossou, P., Parent, M. et al. Convolutional Neural Networks for Spectroscopic Analysis in Retinal Oximetry. Sci Rep 9, 11387 (2019).

[48] J. Deng, W. Dong, R. Socher, L. -J. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248-255

[49] Kvalheim, O. M., Grung, B., Rajalahti, T. (2019). Number of components and prediction error in partial least squares regression determined by Monte Carlo resampling strategies. Chemometrics and Intelligent Laboratory Systems, 188, 79–86.