VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
MODELLING AND DATA ANALYSIS MASTER'S STUDY PROGRAMME

Master's thesis

A Study into Digital Denoising of NIR Spectroscopy Measurements

Skaitmeninių triukšmo šalinimo metodų artimųjų infraraudonųjų spindulių spektroskopiniuose matavimuose tyrimas

Laima Andrejevaitė

Supervisor Viktor Skorniakov, Doc., Dr

Vilnius, 2021

ABSTRACT

Near infrared spectroscopy (NIRS) provides a non-invasive, cost effective and efficient tool for monitoring compositions of various substances. Potential applications of NIRS in medicine could simplify and improve the quality of treatment for diabetic or septic patients. The precision and reliability of any tools used in medicine are of extremely high standard since the price for an error can be immense, however, measurements of any kind inherently have errors attached and data preprocessing algorithms play a crucial role in analysing the dataset. This thesis proposes an autoencoder based approach to NIRS measurement denoising that is directly comparable to the currently used filtering functions.

To achieve this goal a two-step approach was designed to test the denoising autoencoder (DAE) and compare its results to the Butterworth "lowpass" filter (BLP) and the Savitzky-Golay filter (SVG). The DEA was trained on simulated (artificial) data with concentrations in the same range as the measurements. Filtering parameters of BLP and SVG were also optimized on the simulated dataset. Then the measured dataset was filtered three different ways and the results were compared through the mean absolute error (MAE) and the partial least squares regression (PLSR).

BLP and SVG exhibited better results when comparing the MAE, however, DAE outperformed both filters when PLSR results were compared. This offers a conclusion that the DAE distorts some parts of the dataset (spectra) in order to preserve the important features, that are later recongnised by the PLSR.

SANTRAUKA

Artimųjų infraraudonųjų spindulių spektroskopija suteikia neinvazinį, pigų ir veiksmingą būdą matuoti įvairių medžiagų sudėtį. Šio metodo taikymas medicinoje galėtų supaprastinti diabeto ir sepsio požymių aptikimą ir pagerinti gydymo kokybę. Medicininiams įrankiams yra keliami labai aukšti tikslumo ir patikimumo reikalavimai, nes klaida gali kainuoti žmogaus gyvybę, tačiau iš prigimties visi matavimai turi paklaidas, todėl duomenų paruošimas yra neatsiejama duomenų analizės dalis. Šioje tezėje aš siūlau autoenkoderiu paremtą filtravimo algoritmą, kuris prilygtų moderniems skaitmeniniams duomenų filtrams.

Tikslui pasiekti atlikau dviejų dalių eksperimentą, kuriame lyginau autoencoderį (DAE) su Butterworth („žemadažniu") filtru (BLP) ir Savitzky-Golay filtru (SVG). Apmokiau DAE ant simuliuotų (netikrų) duomenų, kurių koncentracijos buvo labai panašios į matuotąsias. BLP ir SVG filtrų parametrai taip pat buvo optimizuoti ant sintetinių duomenų. Tuomet matavimų duomenys buvo filtruojami trimis skirtingais būdais ir jų rezultatai buvo palyginti naudojant vidutinę absoliutinę paklaidą (MAE) ir dalinių mažiausių kvadratų regresiją (PLSR).

BLP ir SVG parodė geresnius rezultatus, jei vertintume tik paklaidą (MAE), tačiau autoenkoderis pademonstravo geresnius rezultatus, nei abu skaitmeniniai filtrai, kai palyginau regresijos rezultatus. Tai leidžia spręsti, jog DAE iškraipo tam tikras spektro dalis, kad išsaugotų reikalingiausius bruožus, kurie vėliau yra atpažįstami regresijos.

TABLE OF CONTENTS

Table of Figures

NOTATIONS AND DEFINITIONS

Adam	Adaptive Moment Estimation
ANN	Artificial Neural Network
BLP	Butterworth “lowpass” filter
BST	Brolis Sensor Technology
DAE	Denoising autoencoder
FDA	Functional data analysis
FT-IR	Fourier Transform
MAE	Mean Absolute Error
MLR	Multiple Linear Regression
MSC	Multiplicative Scatter Correction
MSE	Mean Squared Error
NIR	Near-Infrared
NIRS	Near-infrared spectroscopy
PCA	Principal Component Analysis
PCR	Principal Component Regression
PLS	Partial least squares
PLSR	Partial Least Squares Regression
ReLU	Rectified Linear Units
RMSEP	Root mean squared error of prediction
RMSProp	Root Mean Square Propagation
SGD	Stochastic Gradient Descent
SGD	Stochastic Gradient Descent
SNR	Signal to noise ratio
SNV	Standard Normal Variate
SVG	Savitzky-Golay filter

1. INTRODUCTION

Near infrared spectroscopy (NIRS) is the analysis of near-infrared light's interaction with a molecule. When the light hits the molecule, the bonds in the molecule absorb the energy and respond by vibrating. Each molecule exhibits characteristic vibrations and as a result absorbs specific parts of the electromagnetic spectrum creating a unique "fingerprint". Figure below depicts the absorption spectrum of liquid water. It also outlines the ranges for each part of the electromagnetic spectrum. The term near-infrared light usually refers to the light with wavelengths from 800 to 2500 nm (marked in red).

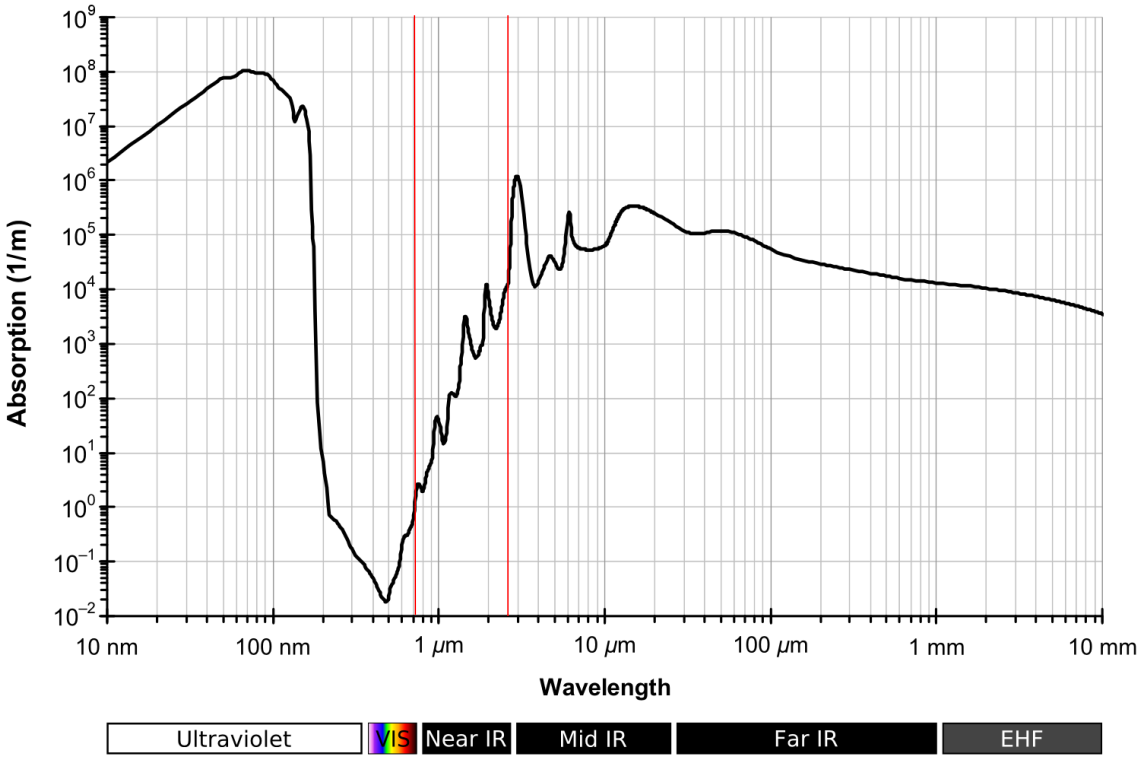


Figure 1: Absorption spectrum of liquid water (source[1])

Applications of the electromagnetic spectrum are endless. Examples in medicine include but are not limited to x-rays for imaging, gamma rays for battling cancer cells, and infrared radiation in thermal imaging. BROLIS Sensor Technology (BST) is currently developing a sensor in the NIR part of the electromagnetic spectrum that would be able to non-invasively monitor glucose for diabetic patients, lactates for sepsis management, and ethanol for screening, safety, and prevention. This option is especially attractive as NIR spectroscopy is a cost-effective, high-speed, and noninvasive material content prediction tool.

Transdermal measurements in the medical field pose a very difficult task, as the signal that we are looking for can be orders of magnitude lower than the signal that we receive. Errors in concentration prediction can cause lives and thus the requirements for measurement devices are very strict. Having

[1] Source: https://commons.wikimedia.org/wiki/File:Absorption_spectrum_of_liquid_water.png (accessed 28 Nov 2020).

the ability to reliably filter out or just reduce the noise (whether it is a setup related noise or a movement of a person) would greatly contribute to the current progress in the field.

The main issue encountered by researchers nowadays is the lack of filtering algorithms that reliably work in real life situations. There are tools available that work well in theory but pose certain limitations in real-life applications. These limitations include:

- preserving useful signal while filtering out the unwanted noise;
- being very dataset dependent and thus requiring heavy supervision; and
- being computationally expensive, thus not useful in live monitoring of analytes.

The aim of this thesis is to examine currently available noise filtering techniques in NIR spectroscopy and signal processing and develop an autoencoder for noise reduction in NIR spectroscopic and compare its performance against currently used methods.

This thesis is structured as follows:

- Part 2 of the thesis delves into the currently available literature on the topic.
- Part 3 proposes an experiment and outlines the methodology for it.
- Part 4 analyses and discusses the results achieve.
- Lastly, part 5 provides concluding remarks.

2. THEORETICAL BACKGROUND AND LITERATURE REVIEW

Part 2 discusses the theoretical background for the thesis and reviews literature on denoising algorithms currently used for preprocessing NIRS measurements. Chapter 2.1 explains the theoretical part of this thesis ("the physics behind the model"). Chapter 2.2 outlines the most popular digital filtering techniques in spectroscopy. Chapter 2.3 concentrates on machine learning algorithms in spectroscopy. Chapter 2.4 analyses the information found in the literature and provides some thoughts and insights. The aim of this section is to give the reader a good understanding of the filtering techniques in NIRS as well as other algorithms used in similar fields. This approach allows for a broader understanding of the issue at hand and means that are currently used.

2.1. Theoretical Background for the Thesis

As mentioned before, when light hits a molecule, the bonds in the molecule absorb the energy and respond by vibrating. Each molecule exhibits characteristic vibrations and as a result absorbs specific parts of the electromagnetic spectrum creating a unique "fingerprint". For illustration, a schematic picture of characteristics vibrations of water is shown below. Ethanol exhibits similar vibrations which, however, are much more complex due to the higher complexity of the molecule.

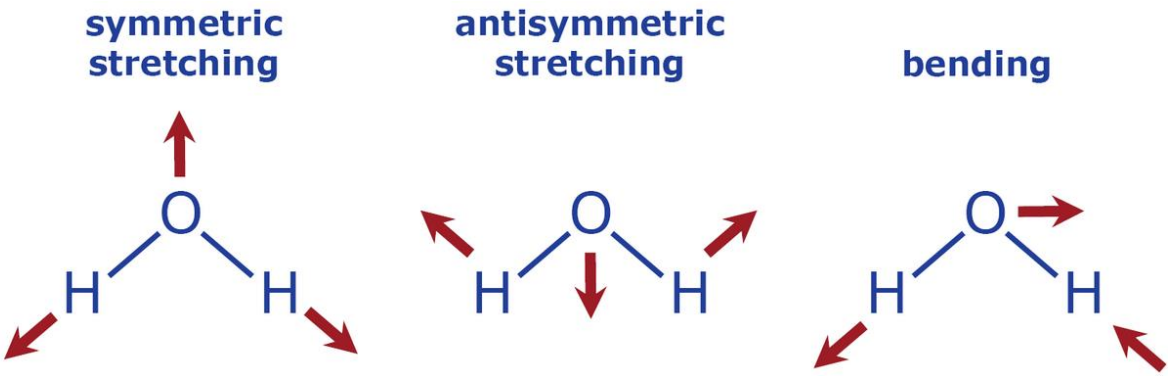


Figure 2: Characteristic vibrations of a water molecule (source[2])

This unique "fingerprint" (absorption spectrum) allows to determine the composition of samples examined to a high degree of accuracy. The complexity of determining concentrations of composites increases with the complexity of the sample examined. Additionally, the complexity changes between different measurement types. Measurement in transmission geometry is the preferred method if it is plausible. However, sometimes to measure in transmission is geometrically not feasible, so a choice to measure in reflection geometry is made. Figure 3 depicts schemes for the two measurement types. Measurements in reflection geometry are significantly more complex due to variations in path lengths that the light has traveled (the light can scatter after hitting the sample or return to the photodiode).

[2] Source: https://seos-project.eu/laser-rs/laser-rs-c07-s03-p03.html (accessed 15 Oct 2020).

For this thesis, I examine only measurements in transmission geometry (measurements of solutions of ethanol in water). The results can be further extended to measurements in reflection geometry; however, there is a significant challenge of accounting for non-linear dependencies between the explanatory variables and dependent variable in the model.

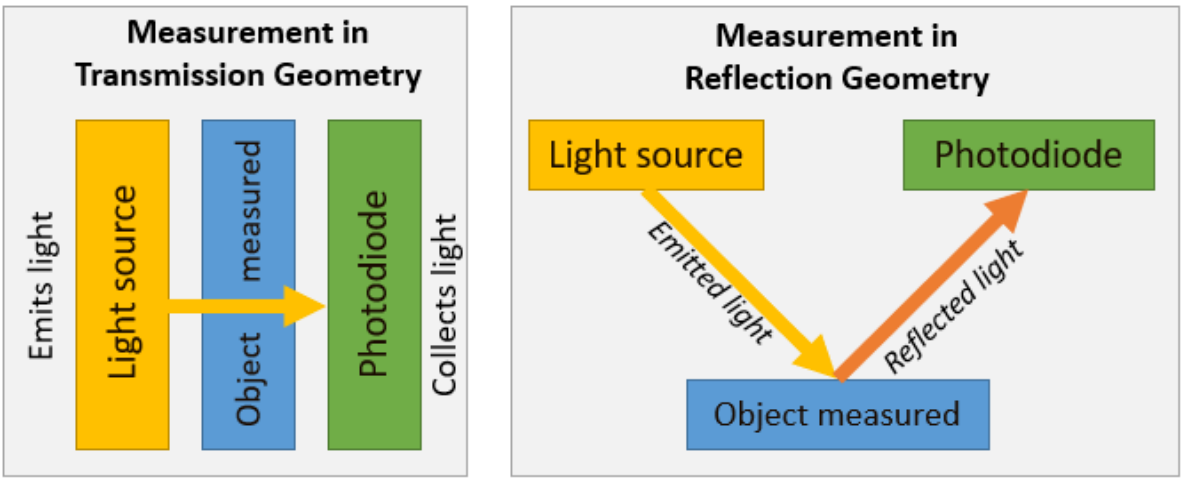


Figure 3: Schematic of spectroscopy measurement configuration in transmission and reflection geometry.

Measurements in transmission geometry follow the Beer-Lambert law which states that "*the absorptive capacity of a dissolved substance is directly proportional to its concentration in a solution*"[3]. It can be expressed as:

$$\boldsymbol{A = \varepsilon l c}.$$

Where

- ε is the absorptivity of the analyte;
- l is the optical path length; and
- c is the concentration of the analyte in the solution.

I assumed that the optical path was equal to 1mm (which is the thickness of the cuvette that the solutions were measured in). Measurements always have an error whether it is a solution mixing error, measurement error (rounding), or some error within the setup, therefore, what we measure is

$$\boldsymbol{A = l \sum_{i=1}^{N} \varepsilon_i c_i + error}.$$

Here ***N*** is the number of different analytes within the sample.

While errors in some measurements can be negligible, measurements for medical applications must be as precise as possible. This raises a need for a reliable noise filtering algorithm. There are a number of factors that can potentially cause noise in NIRS – the setup, the surroundings, the movement of the cuvette etc. It is not always possible to distinguish between the causes of noise in the measurement and thus it is difficult to mitigate for it. Figure 4 shows 4 examples of measurement of a sample in

[3] Source: https://www.britannica.com/science/Beers-law

reflection geometry. These measurements were carried out in the laboratory, by the same people and on the same subject but the noise levels and thus the spectra significantly differ.

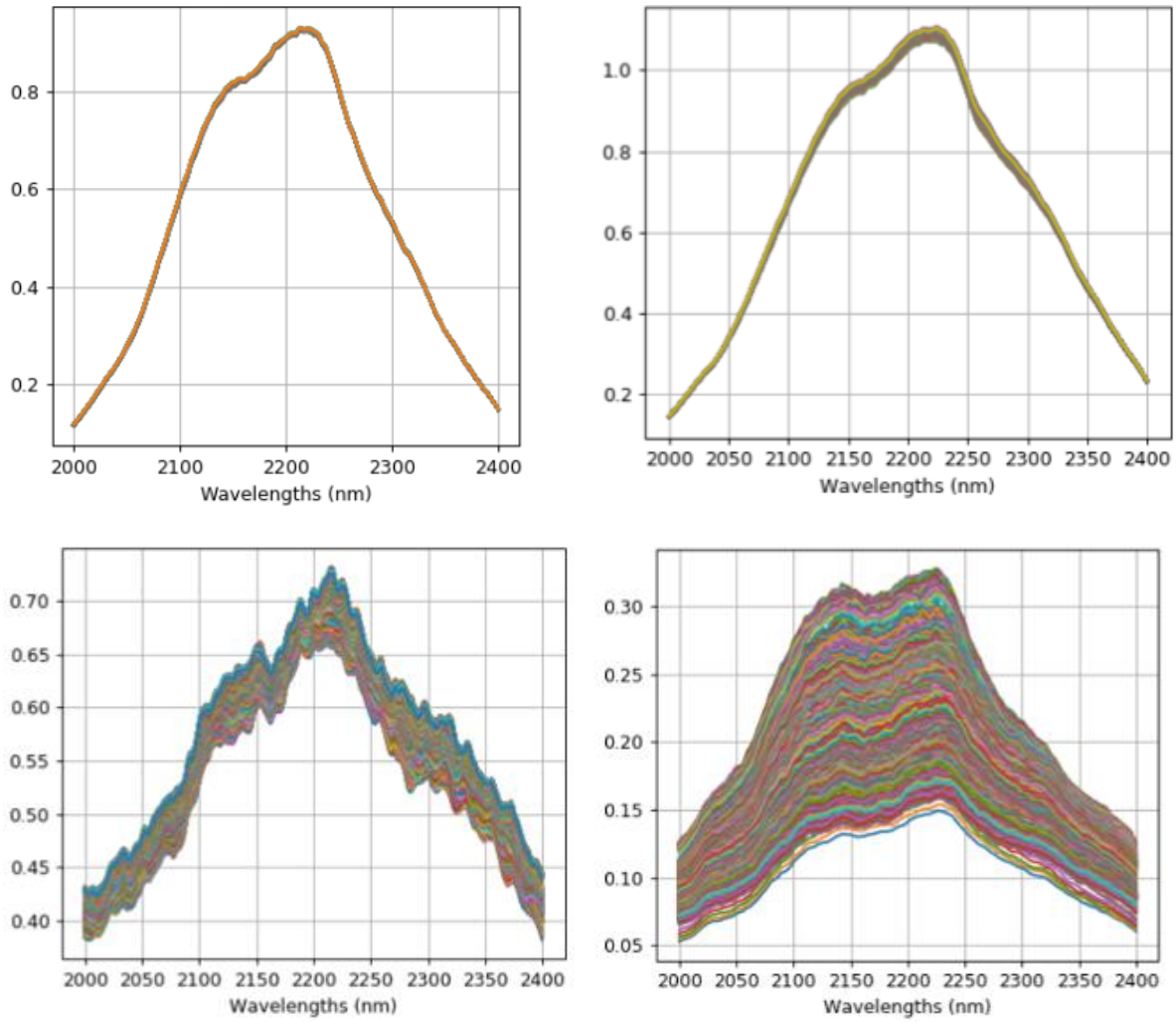


Figure 4: Measurements in transmission geometry carried out in the same laboratory.

In addition, there is also the measurement error that comes naturally with any experiment due to rounding, inaccuracies in mixing the concentrations or any other factors. These issues are also indistinguishable from the noise created by the afore mentioned sources. Therefore, for the purpose of this thesis, I will refer to any differences between the ideal spectrum and the measurement as noise.

2.2. Noise reduction techniques in spectroscopy and signal processing

NIR spectroscopy is a non-invasive way of analysing materials. It can be applied in many disciplines including but not limited to medicine, agriculture, and the food industry. Analysts and researchers in these industries use a broad toolkit for data pre-processing. The methods vary depending on the samples taken, precision required and additional information available. Rinnan et al [1] reviews the most common pre-processing techniques for near-infrared spectra. The paper examines such methods as Multiplicative Scatter Correction (MSC) and variations, de-trending, Standard Normal

Variate (SNV), and normalization as well as Norris-Williams (NW) derivatives and Savitzky-Golay (SVG) polynomial derivative filters. Other filtering methods also include Butterworth filter and Chebyshev filters (Type I and Type II). These techniques cover a few groups of methods that can be used in NIRS measurement denoising and they are successfully being applied in the field.

For this thesis I picked two methods from separate groups for comparison of the results: **Butterworth "lowpass" filter (BLP)** and the **Savitzky-Golay filter (SVG)** as they are widely used in NIRS and provide reliable results in filtering the spectra. The workings and details of these two filters are examined in part 3 of this thesis as well as the appendices. In turn, here I discuss the literature available and research carried out on these filtering techniques.

Butterworth "lowpass" filter (BLP)

The Butterworth filter is a popular noise reduction tool in signal processing. It has a frequency response and allows for several options from which "lowpass" and "highpass" filters are most common. In essence, depending on the filter settings, the filter only allows variations of a certain frequency to pass by and variations with other frequencies are flattened out. In a "lowpass" Butterworth filter all high-frequency variations (high-frequency noise) are filtered out and all slow changes remain in the dataset. Further discussion on the workings of the BLP is presented in part 3.

Pandey and Tiwari [2] used the Butterworth filter on an electrocardiogram[4] (ECG) signal. They tested lowpass, highpass, notch and combined filters. Under normal conditions, ECG graphs have a very predictable pattern, therefore, the Butterworth filter is a good choice. However, medical applications require extremely high precision, and the authors could not get an improved result from cascading the filters when compared to using them separately. In addition, the cascading of these filters was very labour intensive, as it is not automised.

Rodriguez et al [3] compared Butterworth filter to averaging techniques (an arithmetic mean and a rolling mean) for determining the peaks of oxygen availability and utilization during repeated sprint exercises. They found that the Butterworth filter is superior to the averaging techniques in NIRS measurements as it maintained the integrity of the raw dataset.

De Marchi [4][5][6][7][8][9] heavily studied various methods of predicting meat quality and other composites by means of NIRS. His usual tool for pre-processing the NIRS spectra was the Butterworth filter. While he successfully used this method for improving the predictions of his model, definite conclusions cannot be drawn for medical applications from this paper. Rather, it proves, that the filter works well for NIRS measurements.

Savitzky-Golay filter

Similarly to the Butterworth filter, Savitzky-Golay (SVG) filter is used by many researchers to smooth NIRS measurements. While it uses a different way of smoothing, it demonstrates similar performance to the Butterworth filter. SVG uses polynomial smoothing – it fits a chosen order polynomial through the dedicated window length of the spectrum. More information of the workings of the SVG filter is provided in part 3 and the corresponding appendices.

[4] An electrocardiogram (ECG) investigates the cardiac abnormalities by measuring electrical activity generated by the heart.

A paper by Alrezj et al [10] compared the results of partial least squares regression (PLSR) and principal component regression (PCR) after preprocessing the NIRS spectra with two different algorithms: SVG filter with a fixed window size and optimizing the window size for the SVG filter coupled with a Chebyshev's filter. This paper is particularly interesting as it examines the example of predicting glucose levels in a solution (one of the potential future applications of my algorithm in BST). The authors received a better result using the combined approach which again lends to the idea that current filtering in NIRS is extremely labour intensive and requires direct supervision.

Other researchers examined various combinations of SVG filter with either MSC or one of the filters from another group. Chen et al [11] examined the performance of Savitzky-Golay (SVG) filter combined with multiplicative scatter correction (MSC) and received improved results over a standard SVG. Jahani et al [12] examined various smoothing techniques in NIRS and achieved an improved result by using SVG together with a spline interpolation method. Sampaio et al [13] used Savitzky-Golay together with MSC for pre-treatment of spectra when using PLS analysis. Xie et al [14] proposed a stacked SVG approach for improvement of NIRS results and received favorable results.

Most of the current research in NIRS measurement denoising tends to concentrate on stacking or combining already existing filtering methods to improve performance. The benefits and drawback of these approaches are further discussed in chapter 2.5.

2.3. Autoencoders and other ANNs in NIRS

An autoencoder is a type of an artificial neural network (ANN), that aims to learn a representation (encoding) of the training dataset provided in an unsupervised manner. Due to its design, an autoencoder is not able to copy the dataset perfectly (as then it would not be useful). Instead, it is forced to learn the most prominent features and reconstruct them. Goodfellow et al [15] provides a great overview of the basics of autoencoders and it serves as the basis for this thesis. An autoencoder has two parts: an encoder part and a decoder part, and, in its simplest form, can contain only three layers: input, output, and a single hidden layer. A scheme of this simple ANN is presented below.

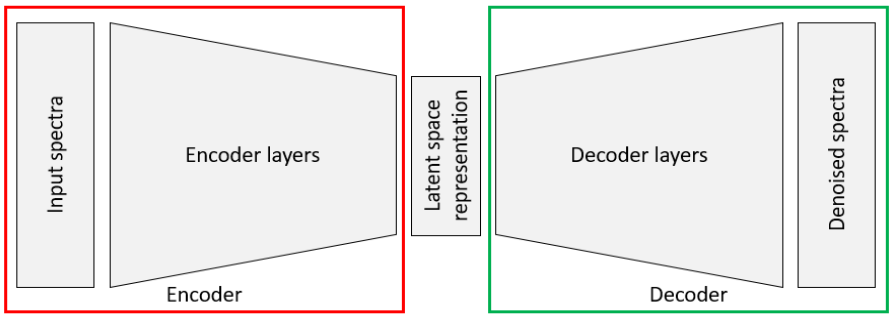


Figure 5: Basic structure of the autoencoder.

The idea of autoencoders has been part of the historical landscape of neural networks for decades (LeCun [16]; Bourlard and Kamp [17]; Hinton and Zemel [18]). Traditionally, autoencoders were used for dimensionality reduction or feature learning. Denoising autoencoders gained their popularity through picture manipulations, the most popular of which is the denoising of handwritten digits which is then used for improved classification. This is achieved by adding an additional classification layer to the already existing denoising autoencoder.

Machine learning techniques in NIRS have been used for a few years now (Madden and Ryder [19], O'connell et al [20], Zhao et al [21], Howley [22] etc). More recently researchers started applying classification algorithms used in image classification and recognition for NIR spectroscopy data. Chen et al [23] applied stacked autoencoder architectures for spectra classification. Houston et al [24] used a locally connected neural network for binary classification of Raman spectroscopy data. Milali et al [25] successfully used an autoencoder for dimensionality reduction in a mosquito parity detection study. Chao et al [26] used a stacked autoencoder and support vector machine (SVM) regression for predicting moisture content of Masson's pine seedling leaves.

These papers usually used autoencoders or other ANNs for classification after pre-processing the data using traditional filtering models (BLP, SVG, MSC or even simple averaging). This approach lends itself to inaccuracies and distortions due to filtering and it will be discussed in part 3 of this thesis.

2.4. Analysis of the Literature

Methods used in pre-processing NIRS measurements differ significantly among papers examined. This is due to variations in data available and differences in precision required. Papers that needed only a light filtering of noise to see general trends in data found Butterworth and Savitzky-Golay filters sufficient for their purposes. Additionally, papers where signal has a very clear trend (frequency) that differs from noise in the dataset used these two filters very successfully and did not have the need for additional tools.

In contrast, NIRS measurements of skin or other more complex matrices tended to look for a more advanced approach to filtering. All papers tried a combination of the currently available methods (such as cascading Butterworth filters) but found that the increase in human labour and computational power outweighs the improvement in filtering capability. This indicates the need for a better way of denoising NIR spectra.

Some papers already tried to apply autoencoders or other ANNs to NIRS measurements, however, they usually tried to classify their spectra rather than purely denoise them. While their models successfully classified the spectra, it is not clear if there was any improvement in the denoising of the dataset.

The literature analysis suggested a few topics for discussion and areas, where more attention should be paid when working with NIRS measurements:

- **Preserving the signal.** Traditional filtering models do not recognize the signal required in the general dataset. There are no safeguards against signal filtering. In addition, using a traditional pre-treatment and then putting the spectra into a neural network lends itself to further complications, where the ANN is trained on distorted datasets.
- **Improving the performance.** BLP, SVG, Chebyshev and other filters perform well, and it is difficult to directly improve their performance by stacking filters or combining them. This usually leads to significant interventions required and does not provide a viable alternative for real-world applications.
- **Unanimity of results.** There is no superior filter that performs best in most cases. In essence, filtering is very dataset specific, thus it is difficult to compare the filtered spectra without using additional methods.

3. RESEARCH METHODOLOGY

Real-life measurements exhibit unknown noise levels and patterns as the causes of the noise are not always clear. Therefore, a thorough approach to modelling and filtering noise has been taken. Firstly, all models where checked on simulated data and their abilities were evaluated. Then, real-life measurements of water and ethanol solutions in a cuvette were introduced and models' performances were compared again. Lastly, a denoising autoencoder (DAE) was trained to denoise NIR spectroscopic measurements. The code and detailed explanations for methods used can be found further in this part and in the appendices.

Chapter 3.1 explains the design of the experiments and the goals for each of them, and all other parts cover in detail various aspects of these plans. Chapter 3.2 explains in detail the datasets examined. Chapter 3.3 discusses the models used and explains the reasoning behind choosing such models. Chapter 3.4 explains how the results of all models are evaluated. Chapter 3.5 provides a short review of other filtering methods and reasons why they were not chosen.

3.1. Design of the Experiments

An experiment was designed to train the DAE on simulated (artificial) data that would have similar noise types to the ones usually encountered within real-life measurements. The reasoning behind such development was as follows:

- **Simulated datasets offer a good understanding and control of the noise levels present in the training data.** This is not always the case for NIRS measurements. Causes of noise can be varied and unknown. In addition, noise frequencies can correlate with the signal frequencies, that additionally complicates the filtering.
- **If training on simulated data proves to be successful, it would create additional opportunities for easily training the model before its use on real data no matter what noise is present.** This would allow for more flexibility in filtering algorithms in NIRS. Finding all noise or accounting even for most of it is a difficult task and sometimes it is impossible. Therefore, a trained autoencoder could solve issues that otherwise would be very time consuming and computationally expensive.
- **Training the autoencoder on measured data allows the autoencoder to learn features of the real noise present.** However, this method could not be universal as a large part of noise is setup dependent (the autoencoder would need to be retained for every setup used) or comes from a variety of factors happening around the measurement (such as changes in temperature or humidity).

While BLP and SVG methods work well in theory and on specific examples, their validity in real-world applications is limited. Therefore, to analyse the performance of the DAE in comparison to BLP and SVG, I opted to approach it in a few steps.

Figure 6 presents a flowchart of the experiment conducted (data, filtering models, DAE, and result evaluation is discussed later in more detail):

- **Step 1.** A simulated dataset was used for finding optimal fitting parameters for Butterworth "lowpass" and Savitzky-Golay filters. This was achieved by cross-validating the results within

a reasonable set of parameters. In turn, for the denoising autoencoder, the simulated dataset was split into train and test sets and the autoencoder was trained.

- **Step 2.** A measured dataset (previously unseen to any of the models) was filtered with the three models. The filtered datasets were then used for evaluation.
- **Step 3.** The mean absolute errors (MAEs) for each spectrum were compared (as a first point of evaluation).
- **Step 4.** Then the denoised spectra were put into the partial least squares regression (PLSR) and their R^2 and root mean squared errors of prediction (RMSEP) were compared (second point of evaluation).

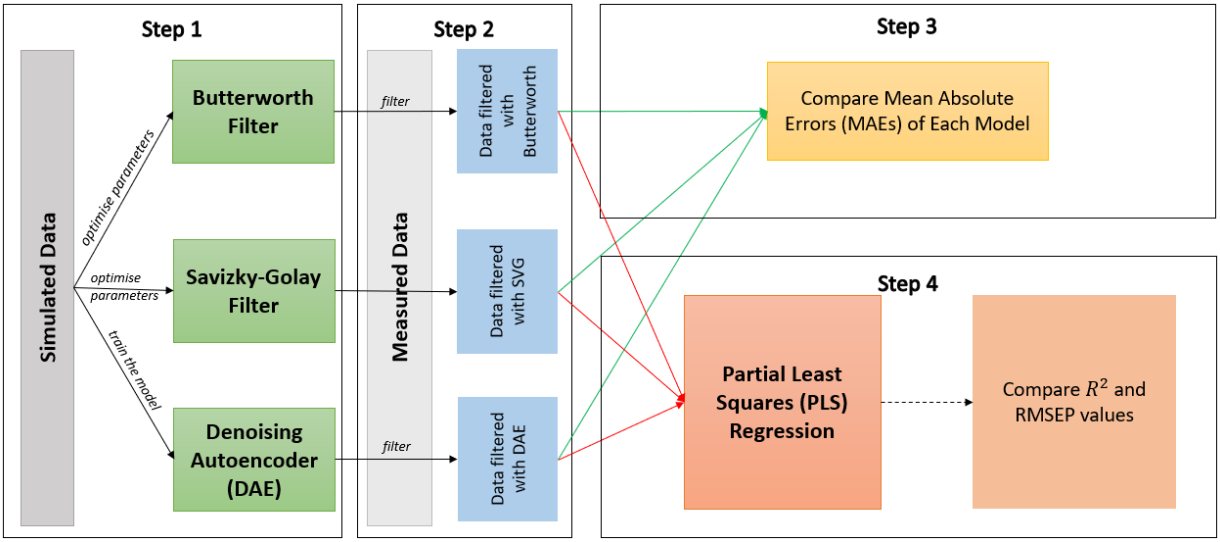


Figure 6: Flowchart of the experiment.

Part 4 provides some further insights effectiveness of each filtering algorithm. The performance failures can be very dataset specific, therefore, it is impossible to mitigate them in advance.

3.2. Data Simulation and Measurements

For better evaluation of the models a few versions of noisy spectra were examined. The simulated datasets allowed for controlled noise levels and distributions whereas the measured datasets just confirmed the results. The discussion bellow aims to shed some light on the modelling techniques used and the reasoning behind it.

Pure (Baseline) Spectra

It is difficult to obtain an exact NIRS measurement (or any real-life measurement for that matter), but for simulations some baselines were needed from which pure or reference spectra could be simulated.

Therefore, water (only water) and ethanol (a solution of 5% of ethanol and 95% of water) of various temperatures were measured with and FTIR Spectrometer[5] in the BROLIS Sensor Technology (BST) laboratory. The dataset was analysed and the relationship between the water spectrum and the temperature was interpolated over a measured temperature range (13 – 55 degrees Celsius). The same analysis was performed on the ethanol solution, with an additional step of subtracting water to obtain an ethanol-only spectrum. Figure 7 depicts the water and ethanol spectra that were later used for simulating spectra in the datasets. The analysis of spectral dependance on temperature is beyond the scope of this thesis; thus the two reference spectra are taken as the best representations available.

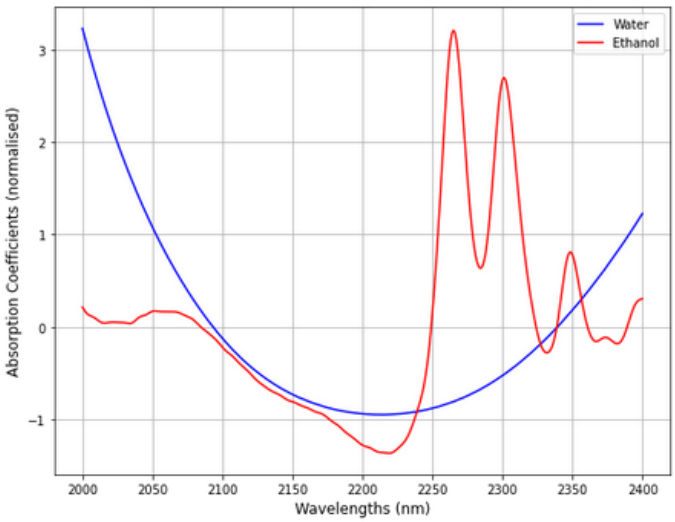


Figure 7: Pure water and ethanol spectra (2000 - 2400 nm)

Normalisation

Data normalisation is an essential part of pre-processing in machine learning, thus all data was pre-processed by the z-score method using the standard scaler function[6] from the Scikit-learn library in Python. This method transforms the dataset into a distribution with a mean (μ) equal to 0 and standard deviation (σ) equal to 1. This transformation is accomplished in the following way:

$$\boldsymbol{x_{std} = \frac{x - \mu}{\sigma}}.$$

This is also a standard practice in machine learning, therefore, not examined further.

Simulated Spectra

The simulated dataset was created based on the Beer-Lambert Law as discussed in part 2 of this thesis:

$$\boldsymbol{A = l \sum_{i=1}^{N} \varepsilon_i c_i + error}.$$

In our example N is equal to 2, thus, for better understanding the function can be re-written as:

$$\boldsymbol{simulated\ spectrum = l * (c_{ethanol} * reference\ ethanol + (1 - c_{ethanol}) * reference\ water)},$$

where $\boldsymbol{c_{ethanol}}$ is the concentration of ethanol in the sample and ***reference ethanol*** and ***reference water*** are the afore mentioned spectra measured with the FTIR spectrometer. We can write $(\boldsymbol{1 - c_{ethanol}})$ because we know that there are no other molecules in the solution and the sum of

[5] FTIR spectrometer simultaneously collects spectral data from a very wide range. As a result, it has a significant advantage over spectrometers that measure a narrow range (in other words, it is very precise).

[6] Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (last accessed 09 Jan 2021).

ethanol and water concentration is equal to 1. Concentrations of ethanol were chosen to be similar to the measured samples, as otherwise there would be no way to check if the models are performing correctly. They ranged from 0% to 5% of ethanol in water.

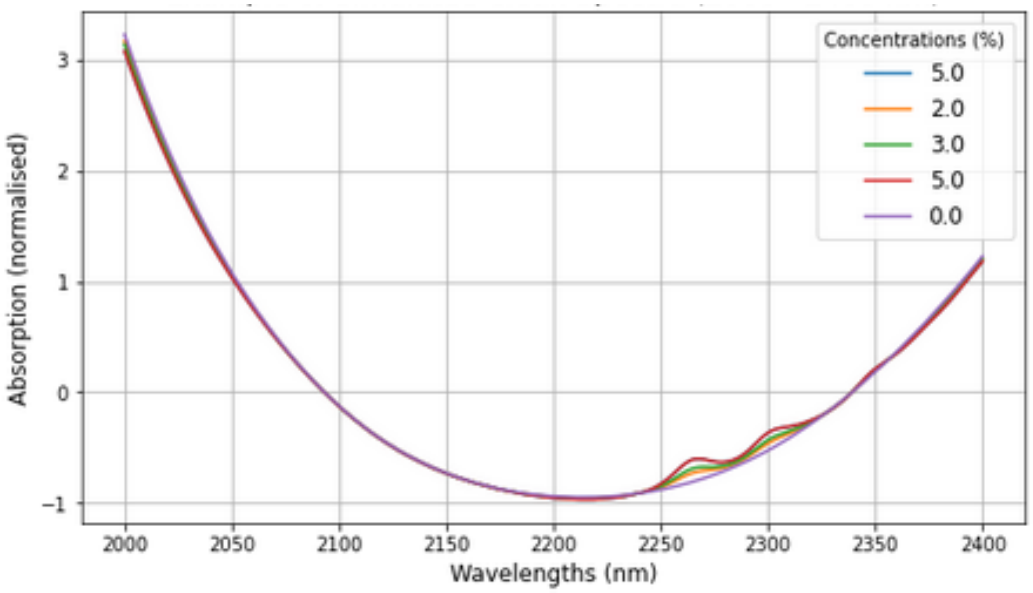


Figure 8: Example of Simulated Noise Free Spectra (2000 - 2400 nm)

Figure 8 shows a sample from the full simulated dataset. Samples with 5% ethanol concentrations have clear peaks between 2250 nm and 2350 nm. These peaks are smaller, but still clear for concentrations less than 5% as well.

We also want to ensure that the simulated spectra look like the measured ones. Figure 9 compares a part of the simulated dataset to the part of the measured dataset (spectra of the same concentrations). It is clear, that the simulated spectra have more prominent ethanol peaks than the measured spectra (marked by the boxes).

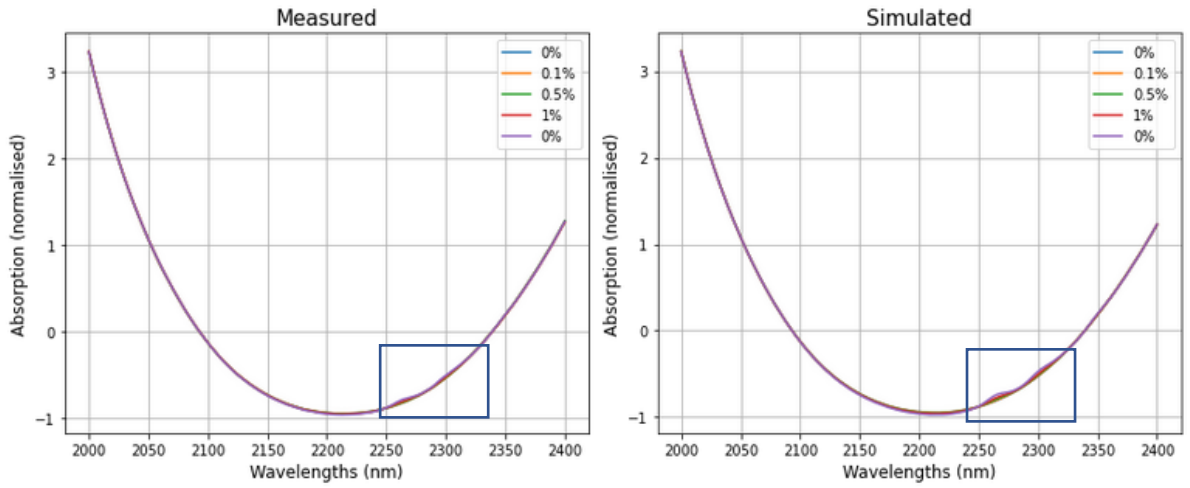


Figure 9: Comparison of measured and simulated spectra.

When filtering these datasets, we want to ensure that the peaks are preserved as much as possible. The size of the peaks is directly proportional to the concentration, therefore, the algorithm that filters well, but also flattens these peaks is not suitable for pre-processing NIRS measurements. The criteria for filters are discussed in later parts.

Noise Levels in Simulated Spectra

For a systematic evaluation of results, a few datasets with increasing and changing noise were created. For the first test, only normally distributed random noise (with μ equal to 0 and varying σ) was added onto the simulated dataset. Then the mean was varied along with the standard deviation. Various distributions of noise were examined, however, due to the algorithms applied within the filters, the distribution of noise carried no influence on the results.

Following the analysis of results from the random normally distributed noise filtering, some systematic oscillations were added to the spectra. They mimic possible interferences created by the setup when measuring. Not all setups produce this noise (and it is preferable that they do not) thus this step was added only to check how the models would deal with this type of noise.

Noise present in the measured dataset was checked to ensure that any patterns are addressed and a combination of all these noises was used for training the final model and optimizing the parameters of the Butterworth and SVG filters.

Measured Spectra

Solutions of various concentrations from 0% to 2% of ethanol in water were measured in the BROLIS Sensor Technology (BST) laboratory in October 2020 (the laser can be tuned over the wavelengths and therefore can measure a range rather than at a single wavelength). The summary of the concentrations measured is presented in appendix 1.

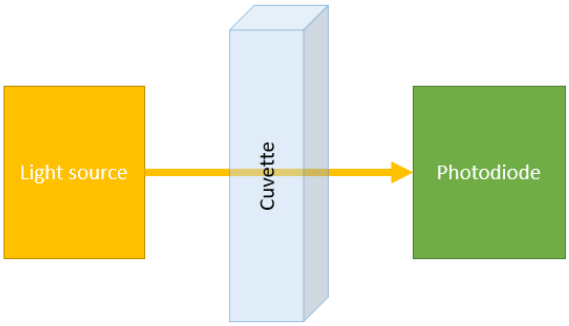


Figure 10: Measurement setup.

The scheme above provides a simple representation of how the spectra were measured. The light source (laser) emits near-infrared light that travels through the cuvette and the solution of ethanol and water in it. Some light gets absorbed by the solution and the cuvette, some light scatters, and the remaining light is collected by the photodiode on the other side of the cuvette. The latter then gives spectra that are examined in this thesis.

The measurement was set-up in the following way:

- Concentrations were prepared in advance for the whole day and put in special containers.
- Concentrations prepared included 0%, 0.1%, 0.5%, 1%, and 2%.
- The flow was alternated by switching pumps among the containers with different concentrations.

- A few iterations of the same concentration in one day were usually measured.
- There was always a gap between two concentrations when only water was flowing to ensure that no residue from higher concentrations is still present in the cuvette after switching.

This created a dataset of various concentrations measured in a continuous fashion for a full working week. A few benefits of this experiment are the elimination of labeling errors and the ability to measure 24/7 without supervision. The drawbacks include a possible inaccuracy of mixing the solutions, since relatively large quantities had to be prepared; also, a possible evaporation of higher concentrations by the end of the week (the 2% solution might be less). More information on the measurement is provided in appendix 1.

3.3. Models Tested

The aim of this thesis is to train an autoencoder for noise reduction in NIR spectroscopic measurements and get results that would be directly comparable to Butterworth filter and Savitzky-Golay filter results. These filters were chosen as a best representation of techniques currently used within the industry. In addition, they represent different groups of filters: Butterworth filter has a frequency response, whereas SVG filter uses polynomial smoothing techniques. These filters coupled with multiplicative scattering correction (MSC) form the base for data pre-processing in NIR spectroscopy.

Frequency Response Model: Butterworth “lowpass” Filter (BLP)

Butterworth [27] proposed a filter that has a frequency response: in essence, the “lowpass” filter allows all low frequency variations to pass but flattens out all high frequency variations[7] (thus the name of the filter). Butterworth filter is defined within the Scipy[8] package in Python. The user can choose the cut-off frequency (the lower the frequency chosen, the earlier the filter starts to cut the signal) and order of the filter (the higher the order, the faster the cutoff). Figure 11 depicts various parameter pairings of the Butterworth “lowpass” filter.

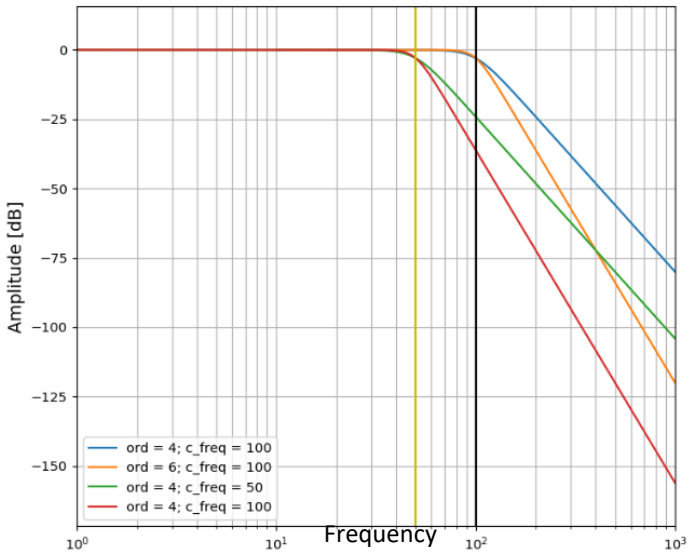


Figure 11: Buterworth "lowpass" filter's frequency response.

[7] More information can be found: https://www.electronics-tutorials.ws/filter/filter_8.html (last accessed 14 Sep 2020).

[8] Documentation: https://docs.scipy.org/doc/scipy/reference/ (accessed 09 Jan 2021).

There are also other variations of the Butterworth filter – “highpass” (that cuts-off all slow trends), “bandpass” (cascades high-pass and low-pass filters), and “bandstop” (paralell combination of the high-pass and low-pass filters). None of the alternatives seem reasonable for filtering NIR measurements, therefore, they are not examined.

BLP works well for NIR spectroscopic measurements as some noise within the measurements tends to have a much higher frequency to the signal in question. As a result, it can be filtered out by choosing correct order and cut-off parameters. To achieve the best result and make the model directly comparable to the DAE developed, the optimisation of filtering parameters was determined on simulated data. Then, with the new parameters, the measured dataset was filtered and the result was compared to those of SVG and DAE.

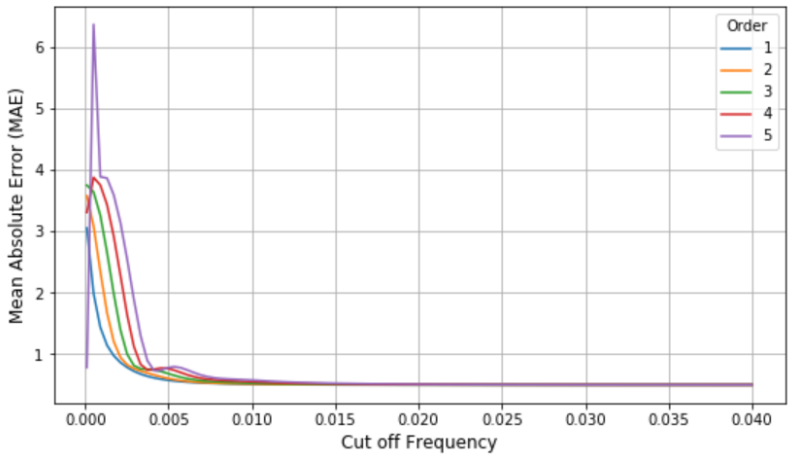


Figure 12: Optimisation of the Butterworth "lowpass" filter parameters.

Choosing the correct filtering parameters is a laborious but necessary task. It can be done one of two ways: either trying some combinations by hand or setting up a simulation through the acceptable parameter ranges (similar to Monte Carlo simulation) and looking for the smallest error provided. For best results of the model, I chose to go through a reasonable range of parameters and pick the lowest mean absolute error (MAE) for each dataset. The criteria for selection of the best filtering parameter was the mean absolute error (MAE) – the same as the loss function that was selected for the DAE, as it creates a consistent background for filter comparison. Since the simulated dataset is very close to the actual measurement, the parameters chosen will be close to the optimal ones for the measured dataset as well. This approach avoids repetition of filtering and checking the result, when working with real-world data.

Polynomial Smoothing: Savitzky–Golay Filter (SVG)

Similarly to the Butterworth filter, Savitzky-Golay filter (SVG) is defined in the Scipy library of Python and also has two parameters that do not have default values: window length and order of the polynomial fitted. This approach also has similarities to the smoothing algorithms used in functional data analysis (FDA). Its appeal lies in a simple-to-understand algorithm and strong filtering capabilities. However, it also exhibits same drawbacks as the Butterworth filter: slow trends tend to remain after filtering, and it is very easy to filter the useful signal out.

Savitzky and Golay [28] proposed a simplified least squares procedure for smoothing of data which became one of the staples of various signal processing toolkits. Wentzel and Brown [29] provide a very elegant explanation of the underlying mathematics of this method, summary of which is presented in

appendix 2. Parameter optimisation (window length and polynomial order) was also conducted on the simulated dataset. Later, the measured dataset was filtered with the chosen parameters.

Artificial Neural Networks: Denoising Autoencoder

As discussed in part 2 of this thesis, various ANNs have been used in NIRS measurements. Their validity has been proven for very specific tasks and datasets, but mainly the efforts were concentrated around classification. These approaches used preprocessing techniques that could potentially distort the signals received. Instead, here we will try to use a neural network (an autoencoder) for data preprocessing (not for classification) and then use traditional regression techniques for concentration forecasting.

We can view the noise as the corrupting factor in our analysis. Instead of copying their inputs to their output, denoising autoencoders are trained to undo a corruption that was placed on the dataset. In most general terms, an autoencoder is a parametric model with two components: the encoder f_{enc,θ_e} and the decoder f_{dec,θ_d}. Mathematically, we can express its loss function as

$$L\left(\boldsymbol{x}, g\left(f(\tilde{\boldsymbol{x}})\right)\right),$$

where $\tilde{\boldsymbol{x}}$ is a corrupted version of $\boldsymbol{x}$.

The architecture of the DAE is outlined in the table below and the choices behind this structure are explained in the next section.

Model: "sequential"			
Layer type	**Output Shape**	**Param #**	**Activation Function**
Conv1D	(None, 393, 256)	2560	ReLU
Conv1D	(None, 385, 128)	295040	ReLU
Conv1D	(None, 377, 32)	36896	ReLU
Conv1DTranspose	(None, 385, 32)	9248	ReLU
Conv1DTranspose	(None, 393, 128)	36992	ReLU
Conv1DTranspose	(None, 401, 256)	295168	ReLU
Conv1D	(None, 401, 1)	2305	Tanh
Total params: 678,209 **Trainable params: 678,209** **Non-trainable params: 0**			

Table 1: Autoencoder architecture

A significant number of variations was tested when training the autoencoder. The majority of hyperparameters were varied to determine the best fit for this task. The summary of hyperparameters that provided the best result is presented in table 1.

The input dataset was split using 80-20 split. For each epoch, the dataset was shuffled and batched into groups of 128 elements. No other transformations were performed on input data.

I have experimented with different numbers of layers: 5, 7 and 9. The model with 5 layers seemed too shallow to capture all the necessary features of the dataset. In turn, the model with 9 layers did not

perform any better than the model with 7 layers but took significantly longer to train due to the increased number of training parameters. Therefore, model with 7 layers was chosen.

Kernel size was another important feature to consider. Since it defines the spatial context the convolutional filter has oversight of. The kernel size had a significant impact on the model. The 7-layerd model produced best results with the kernel size equal to 9 hence it was selected. In addition, convolutional kernels were initialized using He uniform variance scaling initializer.

Filters were selected in a mirrored fashion as per usual in encoder and decoder architectures. The final layer's filter size was equal to one as it ensured that the input and the output dimensions matched.

All convolutional layers, but last, used the ReLU activation function as it exhibits a stable performance across various domains. The last convolutional layer uses Than activation function as it bounds the output in the range -1 to 1. As my standardised dataset ranged from -1 to more than 3, I had to multiply the values by 4 to enable the model to output values in the correct range.

The model uses Adam optimizer with a default learning rate of 0.001. Stochastic gradient descent and RMSProp were considered as well, however, they produced significantly worse results. MAE was chosen as the optimiser's loss function. Another popular option for loss function is mean squared error (MSE) but it provided worse results than MAE. In the MSE function the model smoothed out the ethanol peaks significantly more. This occurred due to the models' averseness to outliers as a result of the squared term in the loss function. Moreover, each version of the model was tested under a varying number of training epochs, ranging from 25 to 300. The chosen model did not show any improvement past 50 epochs on any of the tries, thus this number was chosen for the final model.

Summary of Hyperparameters	
Number of epochs	50
Kernel size	9
Filter sizes	256, 128, 32, 32, 128, 256, 1
Number of layers	7
Optimiser	Adam[9]
Learning rate	0.001
Loss function	MAE
Activation functions	ReLU and Tanh (last layer)
Batch size	128
Train validation split	80-20
Convolutional kernel initialisation	He uniform variance scaling initializer[10]

Table 2: Hyperparameters used in the final model.

All the model code was written using Tensorflow[11] library (version 2.4.0). Hardware was provided by Google Colab[12] where available GPU was utilised to speed up the training process.

[9] More information: https://arxiv.org/abs/1412.6980 (accessed 05 Sep 2020).
[10] More information: https://www.cv-foundation.org/openaccess/content_iccv_2015/papers/He_Delving_Deep_into_ICCV_2015_paper.pdf (accessed 09 Jan 2021).
[11] Source: https://www.tensorflow.org/ (last accessed 09 Jan 2021).
[12] Source: https://colab.research.google.com/ (last accessed 09 Jan 2021).

3.4. Evaluation of Outcomes

Since the filtering models aim to filter out the noise but at the same time preserve the signal, the evaluation of outcomes becomes a complex task. There is no straightforward metric to ensure that both conditions are met across the models. Therefore, a joint approach of using the mean absolute error (MAE) and the partial least squares regression (PLSR) is used. Coincidentally, MAE is also the loss function for the autoencoder.

Firstly, the MAE is calculated for each of the filtered spectra. Reference ("clean") spectra are available for any concentration: they are used as the baseline to calculate how different the filtered spectra are from the ideal situation. The equation used is as follows:

$$\boldsymbol{MAE} = \overline{|C - F|},$$

Where:

- C is the clean spectrum; and
- F is the filtered spectrum.

The filtered spectrum (vector) is subtracted from the clean spectrum (element-wise subtraction). Then the absolute values are calculated and, finally, the mean of these values is extracted.

$$\boldsymbol{MAE = mean(abs(clean\ spectrum - filtered\ spectrum))}$$

This results in a single digit evaluation of the goodness-of-filtering for each spectrum. While individually, these numbers do not carry a considerable amount of information, it provides a stable background for comparison across time and models. However, this evaluation method does not ensure that the signal is preserved as well as possible.

To mitigate the removal of useful signal, filtered data was checked with the PLSR. Herman O. A. Wold developed the PLSR together with his son Svante Wold (in essence it is a generalization of multiple linear regression (MLR)) around 1975. While the original applications of PLSR were in social sciences, it became a popular tool in chemometrics, bioinformatics and neuroscience. Its appeal stems from the ability to deal with possibly correlated explanatory variables as it relates not only two data matrices, **X** and **Y**, by a linear multivariate model, but also models the structures of **X** and **Y** (Wold et al [30]).

In our case, the explanatory variables are concentration, temperature, humidity, and the laser used for measuring. Since the performance of the laser can slightly differ based on the temperature in the room and the spectra are also temperature dependent, the explanatory variables are correlated, making the PLSR a perfect choice for this thesis. The corresponding R^2 and root mean squared error of prediction (RMSEP) values from the regression were compared. As for any regression, we want the R^2 to be as close to 1 as possible and RMSEP to be as low as possible. This result would ensure that there is strong linear relationship between the filtered spectra and the concentrations within the solutions measured. In addition, the results should be better than those of non-filtered spectra, otherwise the filtering loses its purpose.

<u>Comparison of the Models</u>

While the three models have different approaches to filtering, they all work well on spectroscopic data. Table 3 outlines the main benefits and drawbacks of each of the models.

Both, BLP and SVG, under default settings, do not deal well with low frequency noise (such as a slow sinus). This can be fixed by increasing the window length for SVG filter and adding the "highpass"

condition for the Butterworth filter. However, then additional issues of filtering out the signal arise. Figure 13 illustrates this issue with a simple example:

- The same spectrum was filtered in 4 different ways: SVG with a window length of 51, SVG filter with window length of 101, Butterworth lowpass filter with critical frequency of 0.2, and Butterworth lowpass filter with critical frequency of 0.05.
- The figure is a zoomed in part of the picture, where ethanol peaks are located. Therefore, we want this part to be as smooth as possible, but it is important to preserve the slow trend of ethanol peaks in this part of the spectrum.
- Butterworth lowpass filter with critical frequency of 0.2 (black line) seems to follow the data very well, but the same filter with critical frequency of 0.05 significantly flattens the signal (blue line).
- A similar situation arises with the SVG filter. When the window length is 51, the red line follows the simulated data closely, however, when the window length is increased to 101, the green line is almost flat.

In this example we can find optimal filtering parameters after a small number of trials and the spectrum can be smoothed (filtered) to a high standard. However, this is not plausible in the real world. Firstly, there are slow trends that exist in samples (they were not present in the simple example above) and they need to be filtered out. Secondly, this is only a small part of the spectrum, there is always a possibility, that other parts will be a lot worse or that by looking at a wider picture, we will not be able to determine the best fit. Increasing complexity of the sample also plays a role – when do we know that the higher peaks result from higher concentrations when are they just noise? Similar questions always arise during analysis of samples and scientists have to find alternative ways of confirming concentrations in the samples.

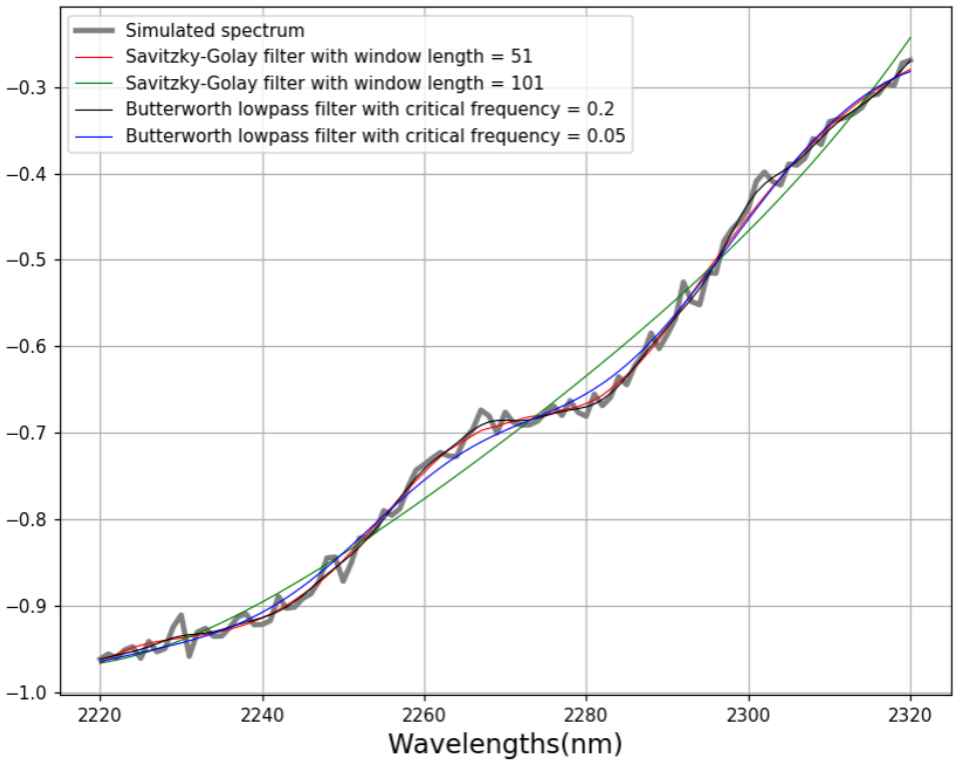


Figure 13: Comparison of Filtering Functions

Model	Benefits	Drawbacks
Butterworth "lowpass" filter	• Easy to use; • Performs very well, when the noise if off different frequency than the signal in question	• Does not filter out low frequency noise in the sample. • Easy to filter the signal out and it would not be noticed until the analysis is carried out.
Savitzky-Golay filter	• Easy to use; • Performs very well, when the noise if off different frequency than the signal in question	• Does not filter out low frequency noise in the sample. • Easy to filter the signal out and it would not be noticed until the analysis is carried out.
Denoising Autoencoder for NIR Measurements	• Can learn important features of the dataset (i.e., find the ethanol peaks in the correct place of the etahnol spectrum and preserve them). • Is able to catch slow variations in the spectra or filter out offsets.	• Takes long to reliably train. • Is not as easily manipulated as the others.

Table 3: Benefits and drawbacks of the models.

3.5. Evaluation of Alternative Methods

Various methods of denoising of datasets and evaluation of results were considered. Main merits and drawbacks of alternative methods are outlined below. The chosen methods of BLP, SVG, MAE, and PLSR cover the majority of properties measured below or are proven to perform better than the methods outlined in this chapter.

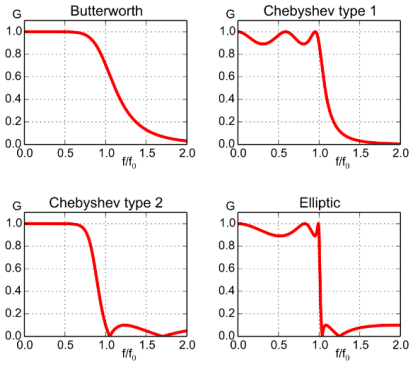


Figure 14: Comparison of the Chebyshev's. Butterworth and Elliptic filters' responses.

Averaging is a simple, powerful, and well-known tool to deal with noise in signal processing. Given a large enough sample, one could average the spectra and remove randomly distributed noise. This process is shown in the pictures below. However, this method is not able to deal with any systematic noise and requires large datasets to be successful. These issues would hinder the method's applicability (as a stand-alone method) in real life measurements; therefore, it was not examined in this thesis.

B-spline smoothing is also one of the possible ways to filter (smooth) non-periodic data. Functional data analysis (FDA) techniques provide an elegant solution to this issue. However, at the moment of writing, the FDA functions are poorly defined within Python as the package was only started in March 2020[13]. The need for switching between R and Python during the analysis would outweigh any benefits received from FDA. To compensate for this, a polynomial smoothing algorithm was chosen (SVG).

Principal component analysis (PCA) is another vastly popular tool in data analysis. This powerful dimensionality reduction tool is very useful when it can separate certain patterns in the data and express them as principal components. During previous studies, I found that PCA does not separate clearly between the components of interest and therefore is not considered in this thesis.

Chebyshev's filters (Type I and Type II) also have a frequency response (similar to the Butterworth filter). Chebyshev filters are sharper than the Butterworth filter and have unwanted ripples (see figure 14[14]).

[13] Documentation: https://fda.readthedocs.io/en/latest/ (accessed 04 Nov 2020).
[14] Source: https://en.wikipedia.org/wiki/Chebyshev_filter (accessed 23 Oct 2020).

4. ANALYSIS AND RESULTS

Part 4 analyses the results of the experiment, provides insights into the performance of the chosen models, discusses shortcomings of the approaches, and provides ideas for further studies into this topic. In chapter 4.1, PLSR results on unfiltered data are examined. Chapter 4.2 reviews Butterworth and Savitzky-Golay filter results on the same dataset. Chapter 4.3 outlines autoencoder performance under various conditions and points out the limitations discovered. Chapter 4.4 compares the results of all afore mentioned models. Chapter 4.5 provides directions for future study on this topic.

4.1. PLSR on Measured Data without Filtering

From the first glance, the measured dataset does not look noisy (figure 15). The graph below shows all spectra plotted together. Clear but small peaks of ethanol are visible between 2250 nm and 2350 nm. After a better examination, there are some differences among the spectra; the ends tend to scatter or bend inconsistently across the dataset. This is a challenge for the predictive algorithm and a potential need for filters.

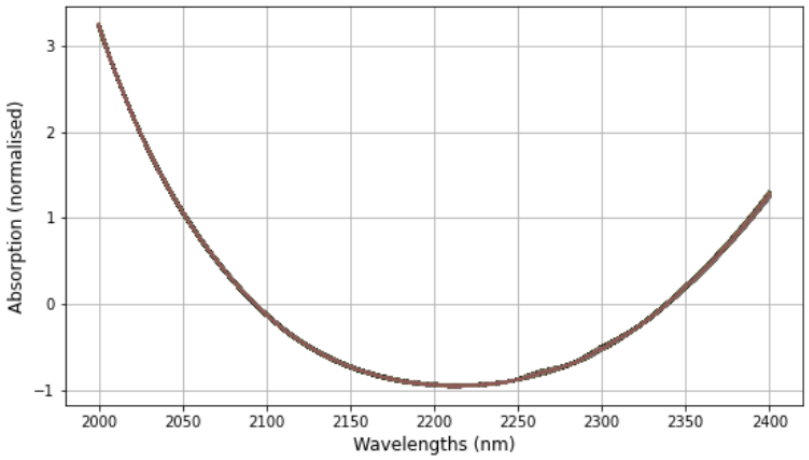


Figure 15: Measured data.

Firstly, I ran the PLSR on the measured dataset without filtering to create a reference point for the three filters. The summary of the results is presented in the table in chapter 4.4. The R^2 value of 0.7918 is too low for medical applications as it signifies, that only around 79% of the total variation is explained by the explanatory variables in our regression. In addition, the RMSEP value of 0.3336 is too high and shows that the error can be more than 35%.

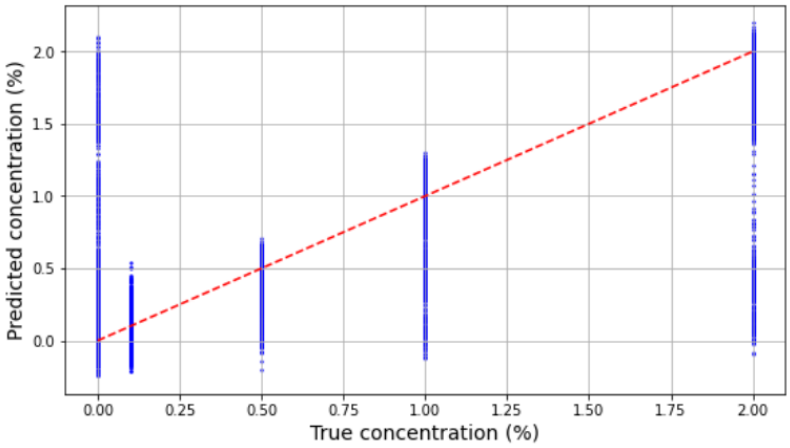


Figure 16: Regression results on unfiltered data.

The inquiry into the predictions of the model provides further insights. Figure 17 plots the predicted concentrations of ethanol against the actual concentrations. Due to the large number of samples, it is not entirely clear from the graph, but most of the points lie around points (0, 0), (0.1, 0.1), (0.5, 0.5), (1, 1), and (2, 2). However, the 0 and 2% concentrations exhibit a significant number of outliers. This can occur due to two reasons:

1. 0% concentration is sampled more often than others as it is used as a cleaning cycle between two other concentrations.
2. There might be a slight time delay between the theoretical start of the concentration and when it reaches the cuvette.

By taking these two reasons into account we can try and mitigate for at least a part of the inaccuracies. Firstly, I checked the 2D histogram to determine, where most values lie.

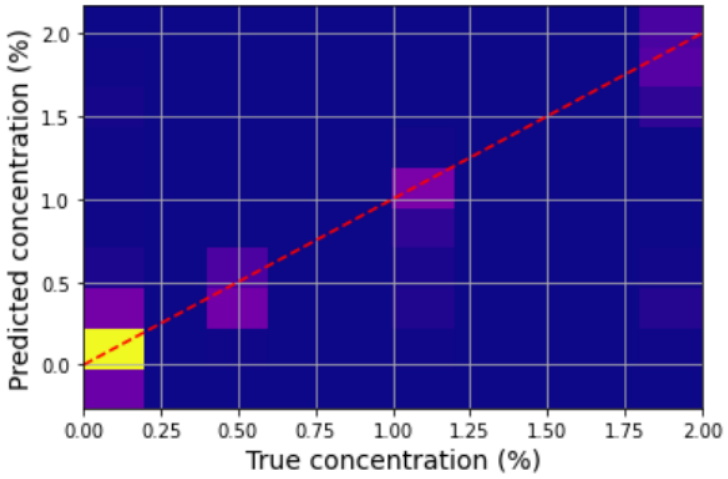


Figure 17: A heatmap of predictions of concentration.

The significantly different yellow region suggests that there are too many low concentrations taken and the sample is not balanced. I sampled 18000 spectra of each concentration from the measured dataset (total of 90000 spectra). A detailed information on sampling is provided in appendix 1. All further analysis was carried out on the balanced dataset and I will refer to it as the measurements (unless explicitly stated otherwise).

4.2. Denoising data with a Butterworth "lowpass" and Savitzky-Golay filters

Butterworth [15]and SVG [16]filters are defined in the Scipy package within Python; therefore, their use is relatively straightforward. The parameters were optimised based on the simulated dataset (as discussed in part 3) and the models were prepared for fitting the measured dataset.

Optimisation of filter parameters on simulated data provided insights into the workings of the Butterworth and SVG filters: they deal well with random noise with mean zero. Any offsets in spectra or low frequency noise are not capture by these filters. They performed well in the simplest case but exhibited weakest performance in all subsequent tests including the measured dataset.

[15] Documentation: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html (last accessed 09 Jan 2021).
[16] Documentation: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.savgol_filter.html (last accessed 09 Jan 2021).

In general, the measured datasets can exhibit a few types of noise. Figure 18 illustrates some of these possibilities.

- **Top left.** Only random normally distributed noise with mean zero is added to the simulated spectra. This is the simples form of noise and can be easily filtered out by any of the filters used in this thesis.
- **Top right.** Here random normally distributed noise with varying means is added onto the simulated spectra. Optimisation of filtering parameters revealed that BLP and SVG filters are not able to deal with the offsets created by the varying mean in the noise.
- **Bottom left.** This graph shows the spectra after a systematic slow variation is added. This type of noise is usually caused by the laser setup itself and sometimes can be fixed by hand.
- **Bottom right.** Random normally distributed noise with a varying mean was combined with the systematic noise in this example. Here, the Butterworth and SVG filters struggled to denoise the datasets correctly due to the offsets and the persistent slow trend in the spectra.

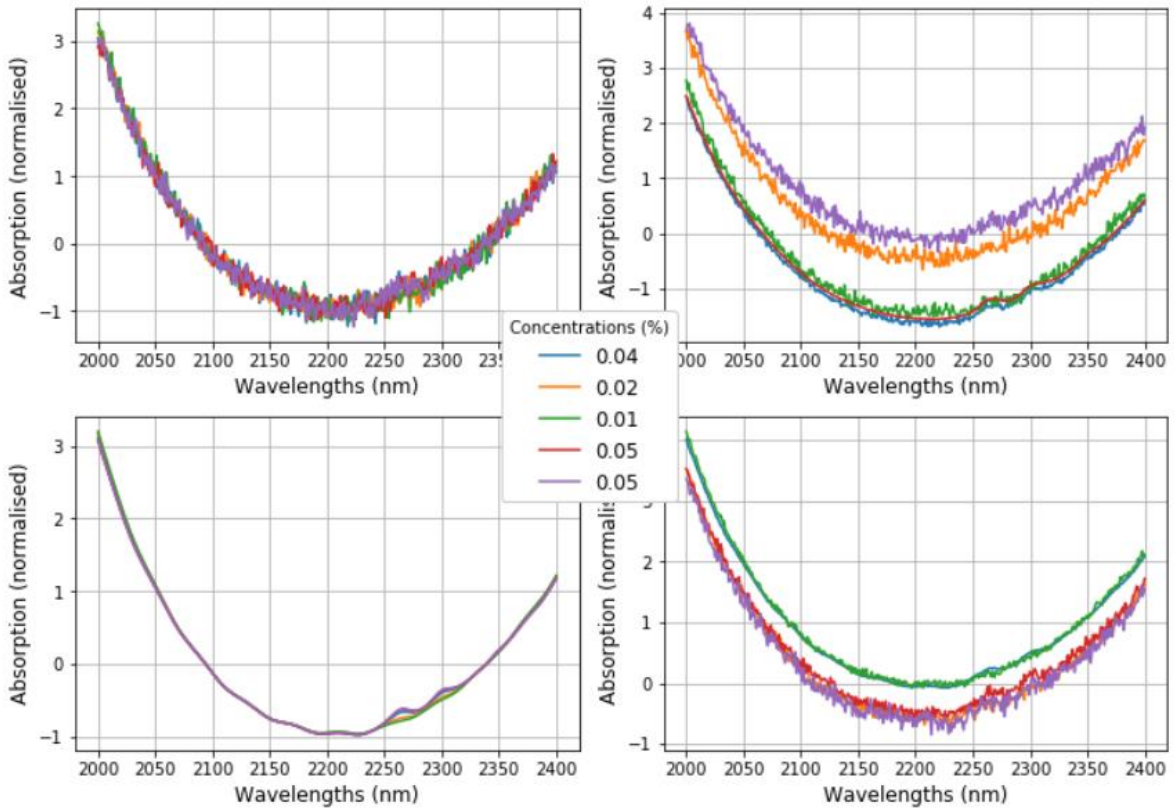


Figure 18: Examples of simulated spectra.

After optimising the parameters, the BLP and SVG filters used were as follows:

- Butterworth “lowpass”: **scipy.signal.butter(2, 0.045, btype='low').**
- Savitzky-Golay filter: **scipy.signal.savgol_filter(window_length = 107, polyorder = 3)**

These functions were then used for filtering the measured dataset. The results are discussed in part 4.4.

4.3. Denoising data with an autoencoder

As discussed in part 3, the autoencoder training process included several steps. Firstly, the autoencoder was trained on a simulated dataset (train and validation sets) and then checked on the real-life measurements (test set). Training the autoencoders on simulated data provided significant insights into the workings and possibilities of denoising datasets with artificial neural networks. Under the circumstances where only random noise is added onto the dataset, neural networks do not pose any advantages over the traditional filtering techniques apart from being unsupervised thus requiring less human intervention.

Training the autoencoder is computationally expensive especially when a differentiated sample of spectra is need. The easy option of generating training spectra of required concentrations does not always work as the researcher needs to clearly understand what noise is present in their samples. However, this is not always possible. Additionally, training on a measured dataset provides good results, but it means that the data provided to the autoencoder are set-up specific and to make a more comprehensive model, one has to spend a significant amount of time in differentiating the measurements (not only concentrations, but also set-ups, days and other conditions). This is a significant drawback of this model but with a large library of spectra created a good precision model can be developed.

With a large, simulated dataset I was able to train the autoencoder to denoise measured spectra. Below graph shows examples of some of the reconstructions of the spectra. While these spectra are not perfectly re-created, the autoencoder has preserved the most prominent features of the dataset (the impact of which will be seen in part 4.4).

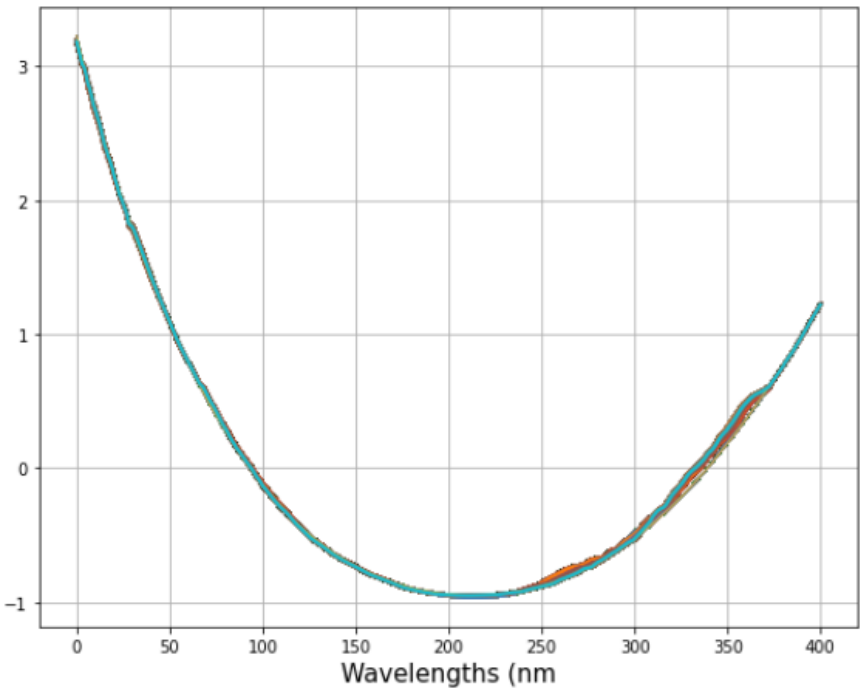


Figure 19: Reconstructions of the spectra by the DAE.

4.4. Performance Comparison

Reconstructions and filtered spectra

Graphical interpretations can be very informative when deciding on filters' performance. Figure 20 shows an example of a spectrum and its filtered/reconstructed versions from different models. It is difficult to draw conclusions from the left-hand graph as the signal does not seem to be strong. One of the possible solutions is to find a different spectrum with a higher concentration, however, then we could not directly confirm that the filters are working well for small concentrations as well. Instead, it is possible to check only the most important part of the graph – the area where ethanol peaks should be present (marked in red).

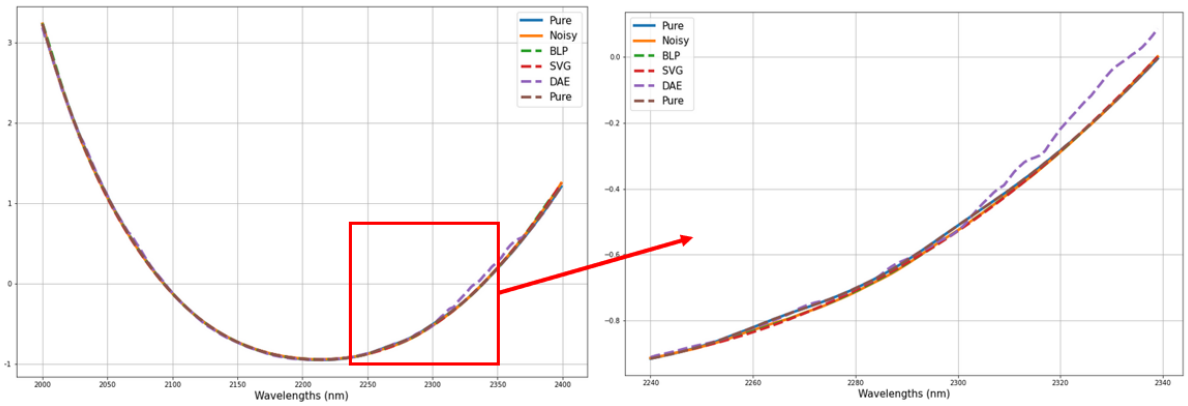


Figure 20: Comparison of the real and filtered spectra.

Graph on the right provides some insights into the differences between the models. There are small peaks of ethanol in the pure spectrum, however, they are a lot smaller in the measured spectrum (blue and orange lines respectively). As a result, the BLP and SVG filters keep the shape of the spectrum as they are not able to strengthen the signal (if the peaks are not present). In turn, the denoising autoencoder has significantly more variation in the region and additionally seems to distort the dataset on the right side of the spectrum. This result is generalized by the mean absolute errors (MAE) calculated for the full measured dataset (figure 21).

When we inputted a measured dataset into the Butterworth filter, it seemed to filter the noise correctly, however, the results were not as good. This arises from the fact that the filter only takes into account the frequency of variations within the sample and at a certain point flattens everything out. In real world applications, this technique does not provide a desired result.

Each approach had its own merits and drawbacks. For real-life NIRS measurements the denoising autoencoder worked as well or better than the Butterworth and SVG filters. The training procedure for the autoencoder did take a significantly longer time than of other filters, but when denoising the test dataset, it provided similarly good performance. The need to re-adjust the filtering parameters for each dataset for BLP and SVG filters also proves that the autoencoder is a better choice. The preservation of ethanol peaks is another advantage.

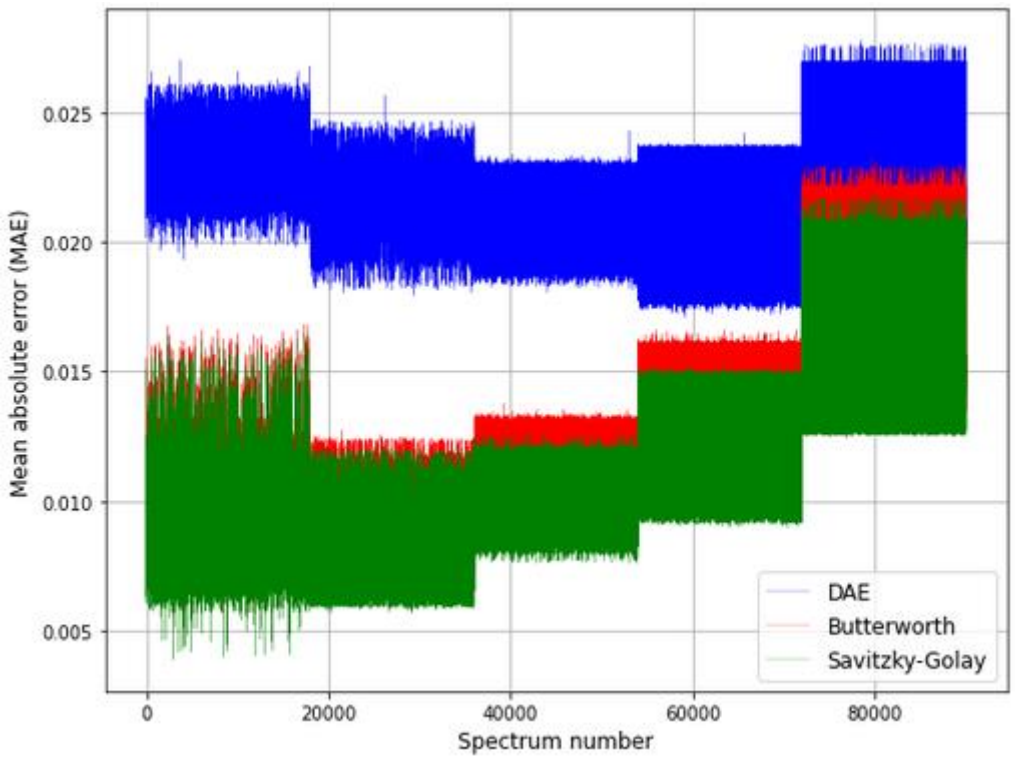


Figure 21: Mean absolute errors of the filtering models and DAE.

PLSR Results

Filtered and unfiltered datasets were tested by the PLSR. Sklearn modules cross decomposition[17] and model selection[18] were used for the regression. A Python function train_test_split from the model selection module randomly split the datasets and corresponding concentrations into train and test subsets. A histogram of the number of test and train spectra for each of the concentrations is presented in figure 22. The numbers of each concentration used in training and testing are very similar, however, the histogram combined the 0% and 0.1% concentrations into a single bar, thus making it seem that the issue of water-only spectra overflow persists (this is true for both test and train sets).

Then a partial least squares regression object with 10 components was created using the cross-decomposition module. This object was the same for all datasets. In short, PLSR takes two centered matrices $\mathbf{X} \in \mathbb{R}^{n \times d}$ and $\mathbf{Y} \in \mathbb{R}^{n \times t}$, and a number of components **K**. The algorithm [19]behind PLSRegression in the cross-decomposition package is as follows:

Set $\boldsymbol{X}_1$ to $\boldsymbol{X}$ and $\boldsymbol{Y}_1$ to $\boldsymbol{Y}$ and, then, $\forall\ k\ \in [1; \boldsymbol{K}]$:

a) Compute the first left and right singular vectors of the cross-covariance matrix $C = X_k^T Y_k$. These vectors are $u_k \in \mathbb{R}^d$ and $v_k \in \mathbb{R}^t$.They are chosen so to maximise the $Cov(X_k u_k, Y_k v_k)$.
b) Then project X_k and Y_k on the singular vectors and obtain scores: $\xi_k = X_k u_k$ and $\omega_k = Y_k v_k$.

[17] Documentation: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.cross_decomposition (last accessed 09 Jan 2021).
[18] Documentation: https://scikit-learn.org/stable/model_selection.html (last accessed 09 Jan 2021).
[19] Source: https://scikit-learn.org/stable/modules/cross_decomposition.html#cross-decomposition (last accessed 09 Jan 2021).

c) Regress:

1. X_k on ξ_k to find a vector $\gamma_k \in \mathbb{R}^d$ s.t. the rank-1 matrix $\xi_k \gamma_k^T$ is as close as possible to X_k.
2. Instead of regressing Y_k on ω_k, regress it on ξ_k (a projection of X_k) and get δ_k. This gives a slightly different loadings computation than usual.

d) Lastly, subtract the rank-1 approximations:

$$X_{k+1} = X_k - \xi_k \gamma_k^T$$

$$Y_{k+1} = Y_k - \xi_k \delta_k^T$$

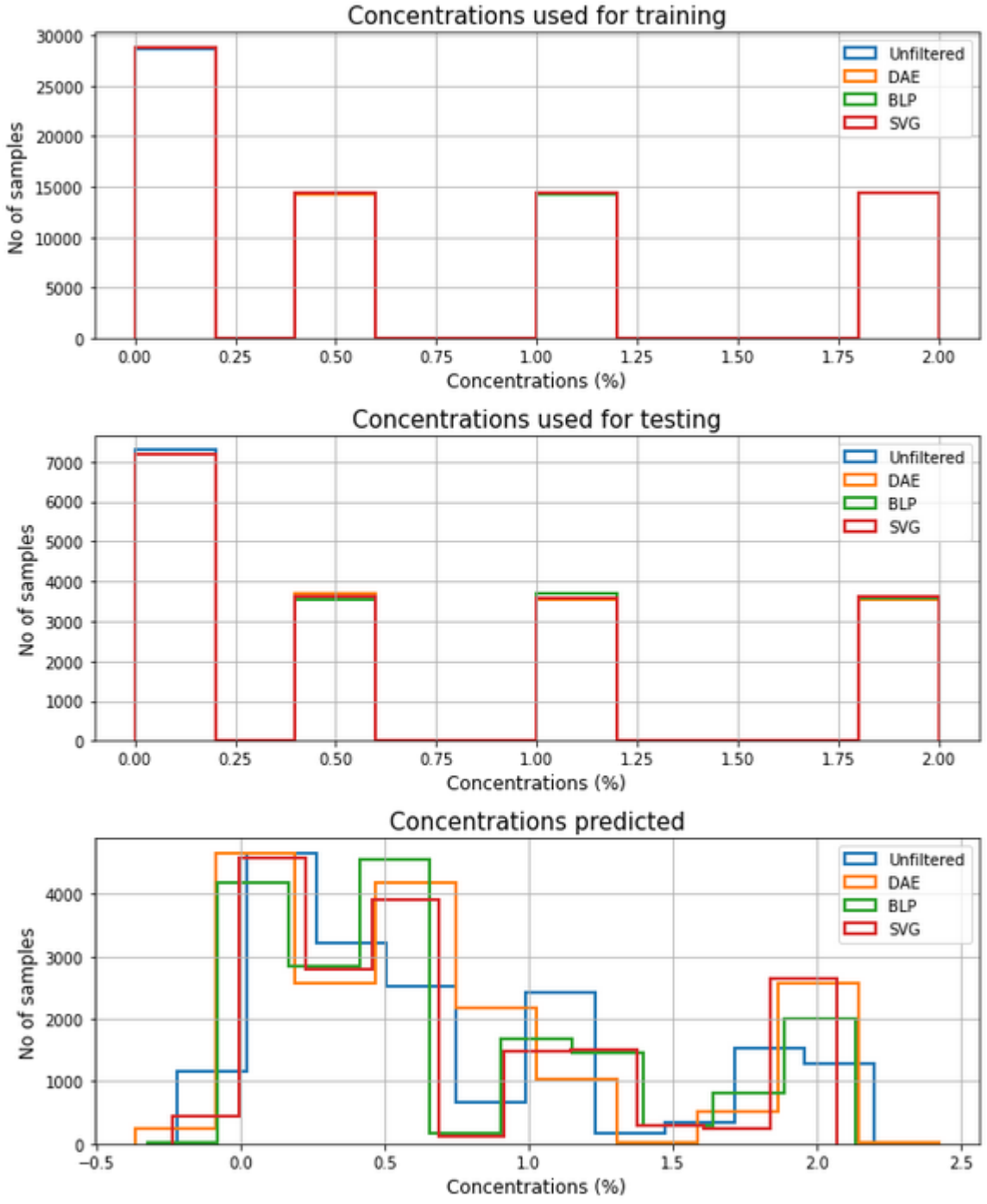


Figure 22: Comparison of model performance in PLSR.

Figure 22 further examines the predictions of concentrations of the test set. Interestingly, 3 out of 4 regressions classified concentrations that were never used for training (negative concentrations). The graph also shows that the regression tends to favor low concentrations which can be a result of relatively close 0% and 0.1% samples. From this graph it would be difficult to chose a single best model as all filters have areas, where they perform best.

Summary of the main PLSR metrics is provided in the table below. Since the measurements were relatively stable, the unfiltered dataset reached an R^2 score of 0.7918 and a RMSEP of 0.3336. In turn, the BLP and SVG filters decreased these scores to 0.7799 and 0.7757 respectively for R^2 and 0.3416 and 0.3466 respectively for RMSEP. However, the proposed denoising autoencoder slightly improved the result of the PLSR when compared to the unfiltered dataset (R^2 = 0.7949 and RMSEP = 0.3294).

In summary, the results from the two examined performance metrics suggest that the denoising autoencoder distorts certain parts of the spectra to preserve the most prominent features in a consistent way. As discussed before, the BLP and SVG filters also distort the datasets that they filter, however, the feature preservation is not always possible in these filters.

The improvement of the algorithm lies not only in the R^2, RMSEP or MAE, it also includes the possibility of filtering the data as it is measured, since a pretrained neural network does not require a full dataset to perform the filtering in a consistent way and there is no need for human intervention. This feature makes it more applicable in the real-world.

PLSR Results				
	Unfiltered	**Butterworth**	**SVG**	**DAE**
$\boldsymbol{R^2}$	0.7918	0.7799	0.7757	0.7949
RMSEP	0.3336	0.3416	0.3466	0.3294

Table 4: Summary of PLSR results.

4.5. Further Research Questions

DAE successfully reduced the noise of the measurements of simple ethanol in water solutions in a cuvette. Currently used techniques in NIRS tend to have a similar effect (especially when combinations of these techniques are used). The autoencoder provides a simple way of dealing with various types of noise in this situation and requires little human intervention and therefore is superior to the currently used filtering methods. However, its reconstructions cannot be directly comparable to the reconstructions of other models as it seems to distort some features of the spectra to preserve the others. Further research is needed to examine these distortions and possibly eliminate them.

Moreover, with linear activation functions, autoencoders have a strong relationship with the principal component analysis (PCA). PCA is a well-established tool in analysing any dataset. It provides more clarity to the user and therefore could be considered a superior technique in this case. However, the strength of autoencoder learning lies in the non-linearities of samples (Goodfellow et al [15]). This feature can prove to be very useful for further applications in transdermal sensing research as it exhibits non-linear relationships.

5. CONCLUSIONS

Currently used NIRS measurement filtering algorithms do not provide a suitable option for filtering the noise of real-life measurements under reasonable conditions. They tend to be noise dependent and require significant human monitoring and intervention. An alternative of using a denoising autoencoder for NIRS measurements was explored. A DAE for NIRS measurements of ethanol and water solutions in a cuvette was developed and its results were compared to the performance of the Butterworth “lowpass” filter and the Savizky-Golay filter.

The DAE seemed to distort some parts of the spectra to preserve the most prominent features. This was confirmed by using PLSR that demonstrated an improved results from the ones shown by the Butterwoth “lowpass” filter and the Savitzky-Golay filter. As a result of this thesis, we can conclude that the denoising autoencoder should be tested on more complex datasets with potentially non-linear relationships between independent and explanatory variables (for example skin spectra measured in reflection geography).

BIBLIOGRAPHY

[1] **Rinnan A, van den Berg F and Engelsen SB** (2009) Review of the most common pre-processing techniques for near-infrared spectra. *TrAC Trends in Analytical Chemistry, Volume 28, Issue 10, Pages 1201-1222*

[2] **Pandey H and Tiwari R** (2014) An Innovative Design Approach of Butterworth Filter for Noise Reduction in ECG Signal Processing based Applications. *Progress in Science in Engineering Research Journal.* Available here (accessed 26 October 2020).

[3] **Rodriguez RF, Townsend NE, Aughey RJ, and Billaut F** (2018) Influence of averaging method on muscle deoxygenation interpretation during repeated- sprint exercise. *Scandinavian Journal of Medicine & Science in Sports Published by John Wiley & Sons Ltd,* 2263-2271.

[4] **De Marchi M** (2013) On-line prediction of beef quality traits using near infrared spectroscopy. *Meat Science, Volume 94, Issue 4, August 2013,* 455-460.

[5] **De Marchi M, Berzaghi P, Boukha A, Mirisola M, and Gallo L** (2007) Use of near-infrared spectroscopy for assessment of beef quality traits. *Italian Journal of Animal Science, 6,* 421–423.

[6] **De Marchi M, Penasa M, Battagin M, Zanetti E, Pulici C, and Cassandro M** (2011) Feasibility of the direct application of near-infrared reflectance spectroscopy on intact chicken breasts to predict meat color and physical traits. *Poultry Science, 90,* 1594–1599.

[7] **De Marchi M, Penasa M, Cecchinato A, and Bittante G** (2013) The relevance of different infrared technologies and sample treatments for predicting meat quality traits in commercial cuts. *Meat Science, 93,* 329–335.

[8] **De Marchi M, Penasa M, Cecchinato A, Mele M, Secchiari P, and Bittante G** (2011) Effectiveness of mid-infrared spectroscopy to predict fatty acid composition of Brown Swiss bovine milk. *Animal, 5,* 1653–1658.

[9] **De Marchi M, Riovanto R, Penasa M, and Cassandro M** (2012) At-line prediction of fatty acid profile in chicken breast using near infrared reflectance spectroscopy. *Meat Science, 90,* 653–657.

[10] **Alrezj OA, Patchava K, Benaissa M, and Alshebeili SA** (2017) Pre-processing to Enhance the Quantitative Analysis of Glucose from NIR and MIR Spectra. *In: Eskola H., Väisänen O., Viik J., Hyttinen J. (eds) EMBEC & NBC 2017. EMBEC 2017, NBC 2017. IFMBE Proceedings, vol 65. Springer, Singapore.* https://doi.org/10.1007/978-981-10-5122-7_11 (accessed 04 November 2020).

[11] **Chen H, Song Q, Tang G, Feng Q and Lin L** (2012) The Combined Optimization of Savitzky-Golay Smoothing and Multiplicative Scatter Correction for FT-NIR PLS Models. *Volume 2013 |Article ID 642190*. Available at: https://doi.org/10.1155/2013/642190 (accessed on 16 Dec 2020)

[12] **Jahani S, Setarehdan SK, Boas DA, and Yücel MA** (2018) Motion artifact detection and correction in functional near-infrared spectroscopy: a new hybrid method based on spline interpolation method and Savitzky-Golay filtering. Neurophotonics, 5(1):015003.

[13] **Sampaio P, Soares A, Castanho A, Almeida AS, Oliveira J, and Brites C** (2017) Dataset of Near-infrared spectroscopy measurement for amylose determination using PLS algorithms. *Data in Brief, 15,* 389 – 396

[14] **Xie S, Xiang B, Yu L, and Deng H** (2009) Tailoring noise frequency spectrum to improve NIR determinations*. Talanta, Volume 80, Issue 2,* 895-902

[15] **Goodfellow I, Bengio Y, and Courville A** (2016) Deep Learning. *MIT Press.* Available at: http://www.deeplearningbook.org (accessed 09 Sept 2020)

[16] **LeCun Y and Fogelman FS** (1987) Modeles connexionnistes de l'apprentissage. *Intellectica Revue de l Association pour la Recherche Cognitive 2(1).* Available here (accessed 24 Oct 2020).

[17] **Bourlard H and Kamp Y** (1988) Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics, volume 59,* 291–294.

[18] **Hinton G and Zemel RS** (1994) Autoencoders, minimum description length and Helmholtz free energy. *NIPS'93: Proceedings of the 6th International Conference on Neural Information Processing Systems,* Pages 3–10.

[19] **Madden MG and Ryder AG** (2002) Machine Learning Methods for Quantitative Analysis of Raman Spectroscopy Data*. Opto-Ireland 2002: Optics and Photonics Technologies and Applications,* 1130–1140.

[20] **O'connell ML, Howley T, Ryder AG, Leger MN, and Madden MG** (2005) Classification of a Target Analyte in Solid Mixtures using Principal Component Analysis, Support Vector Machines, and Raman Spectroscopy. *Opto-Ireland2005: Optical Sensing and Spectroscopy,* 340–351.

[21] **Zhao J, Chen Q, Huang X, and Fang C** (2006) Qualitative Identification of Tea Categories by Near Infrared Spectroscopy and Support Vector Machine*. J. Pharm.Biomed. Anal.2006, 41,* 1198–1204.

[22] **Howley T** (2007) Kernel Methods for Machine Learning with Application to the Analysis of Raman Spectra. *National University of Ireland Galway.*

[23] **Chen Y, Lin Z, Zhao X, Wang G, and Gu Y** (2014) Deep Learning-based Classification of Hyperspectral Data*. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 7,* 2094–2107.

[24] **Houston J, Glavin FG, and Madden MG** (2020) Robust Classification of High-Dimensional Spectroscopy Data Using Deep Learning and Data Synthesis. Journal of Chemical Information and Modeling, Available here (accessed on 31 Aug 2020).

[25] **Milali MP, Kiware SS, Govella NJ, Okumu F, Bansal N, Bozdag S, Charlwood JD, Maia M, Ogoma SB, Dowell FE, Corliss GF, Sikulu-Lord MT, and Povinelli RJ** (2020) An Autoencoder and Artificial Neural Network-based Method to Estimate Parity Status of Wild Mosquitoes from Near-infrared Spectra. PLoS One. 2020; 15(6): e0234557.

[26] **Chao N, Zhang Y, and Wang D** (2018) Moisture Content Quantization of Masson Pine Seedling Leaf Based on Stacked Autoencoder with Near-Infrared Spectroscopy. *Journal of Electrical and Computer Engineering, Volume 2018, Article ID 8696202.*

[27] **Butterworth S** (1930) On the Theory of Filter Amplifiers*. Experimental Wireless and the Wireless Engineer, Vol. 7,* 536-541.

[28] **Savitzky MJ and Golay E** (1964) Smoothing and Differentiation of Data by Simplified Least Square Procedures. *Anal. Chem., 36,* 1627–1639.

[29] **Wentzel P and Brown C (2006)** Signal Processing in Analytical Chemistry. *In book: Encyclopedia of Analytical Chemistry.*

[30] **Wold S, Sjostrom M and Eriksson L (2001)** PLS-regression: A Basic Tool of Chemometrics. *Chemometrics and Intelligent Laboratory Systems 58(2):*109-130.

APPENDIX 1: SUMMARY OF MEASUREMENTS CARRIED OUT

All measurements were carried out in the Brolis Sensor Technology (BST) laboratory in Vilnius by the measurements team. Firstly, the water and ethanol spectra of different temperatures were measured in the FT-IR spectrometer on 14 July 2020. This provided a base for the simulations of artificial spectra.

Then a series of various concentrations in a flow cuvette were measured over a span of a full week in October 2020. Lower concentrations were chosen on purpose, as we want to develop an algorithm for transdermal sensing of ethanol in humans (where 2.5 permille is already considered a high level of alcohol in person's blood). The summary of the number of spectra measurements and concentrations is provided in the table below.

Table AX: Measurements of Ethanol Solutions in Water.

Date	Concentrations measured (%)	No of Spectra (raw)	No of Spectra (avg)
2020-10-19	0.1, 0.5, 1 and 2	18,683	753
2020-10-20	0.1, 0.5, 1 and 2	31,166	1,234
2020-10-21	0.1, 0.5, 1 and 2	26,026	1,044
2020-10-22	0.1, 0.5, 1 and 2	36,396	1,440
2020-10-23	0.1, 0.5, 1 and 2	25,324	1,006
	Total:	**137,595**	**4,577**

As discussed in the main text, the above measurement was not balanced as there were intervals of water between any two concentrations to prevent the concentrations mixing, thus, the number of water-only spectra exceeded any other concentration more than twice (see the table below). The balancing was carried out by using the sample function[20] from the Pandas package in Python.

Balancing the sample						**Total**
Concentration (%)	0	0.1	0.5	1	2	**5**
Before (no of measurements)	59532	19666	18938	18948	20502	**137586**
After (no of measurements)	18000	18000	18000	18000	18000	**90000**

Figure 23 depicts all spectra that were measured over the week. They seem very homogeneous with slight deviations on the edges and the areas were ethanol peaks should be. Unfortunately, not all measurements carried out are as good as this one.

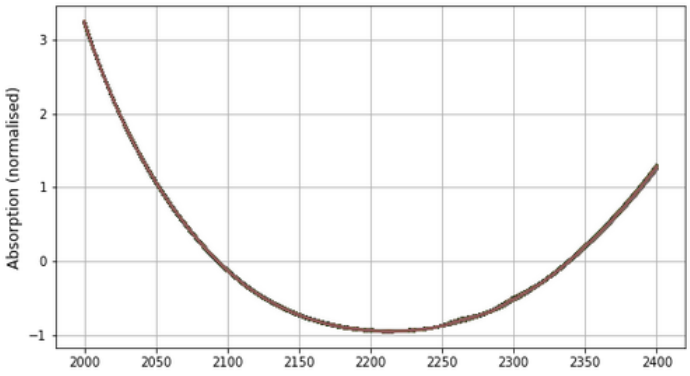


Figure 23: Measured spectra.

[20] Documentation: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.sample.html (last accessed 09 Jan 2021).

APPENDIX 2: SAVITZKY-GOLAY FILTER

This appendix discusses the mathematical background for the Savitzky-Golay filter in detail. It also explains the reasons why this filter is often used in NIRS. The information provided here comes from Wentzel and Brown [29]. Savitzky-Golay (SVG) filter uses polynomial smoothing to denoise spectra. If we consider a second order seven point polynomial smooth, then the model to be fit can be expressed as

$$\boldsymbol{y = b_0 + b_1 x + b_2 x^2}.$$

The equivalent matrix form of this is

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_{17} \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ 1 & x_4 & x_4^2 \\ 1 & x_5 & x_5^2 \\ 1 & x_6 & x_6^2 \\ 1 & x_7 & x_7^2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

We can express it as $\boldsymbol{y = Xb}$ where $\boldsymbol{X}$ is the matrix containing polynomial functions for the fit. The least squares solution for $\boldsymbol{b}$ is well known from linear algebra:

$$\boldsymbol{b = (X^T X)^{-1} X^T y = Ay}$$

Then the intercept for the fit is

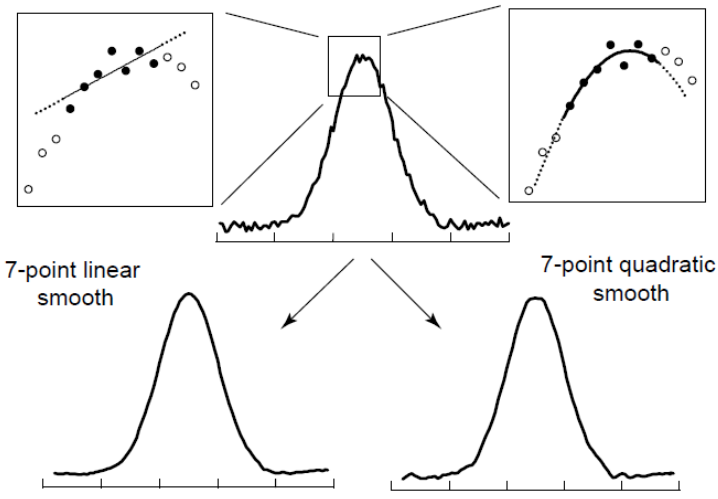
$$b_0 = \boldsymbol{a_1 y} = a_{11} y_1 + a_{12} y_2 + \cdots + a_{21} y_7$$

As *x = 0* for the central point in a 7-point sequence (because of the way this problem is setup), the following equation holds:

$$\hat{y}_4 = b_0 + b_1(0) + b_2(0)^2 = b_0 = \boldsymbol{a_1 y}$$

To simplify, the filter coefficients are first row of the matrix $\boldsymbol{(X^T X)^{-1} X^T}$.

This approach holds for SVG filter of any length and any order. The figure on the right provides a graphical explanation of the algorithm (source Wentzel and Brown [29]).



APPENDIX 3: CODE FOR THIS THESIS

```
# IMPORTS
# All packages used later are imported here.

import random
import numpy as np
import pandas as pd
from scipy import signal
from google.colab import drive
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

import keras
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Conv1D, Conv1DTranspose
from keras.constraints import max_norm

# FUNCTION DEFINITIONS
# define a functions for future use.

def calculate_temp_dependent_spectrum(temperature, coefficients):
  return temperature * coefficients[:,0] ** 2 + temperature * coefficie
nts[:,1] + coefficients[:,2]

def use_butterworth_filter(noisy_spectra, clean_spectra, order, crit_fr
eq):

    b, a = signal.butter(order, crit_freq, 'lowpass')

    filtered = []
    error_lst = []
    for i in range(noisy_spectra.shape[1]):
        y = noisy_spectra[:,i]
        output = signal.filtfilt(b, a, y)
        filtered.append(output)
        error_lst.append(np.mean(abs(clean_spectra[:,i] - output)))
    filtered = np.stack(filtered, axis = 1)
    error_lst = np.stack(error_lst)

    return filtered, error_lst
```

```
def use_svg_filter(noisy_spectra, clean_spectra, window_length, poly_or
der):

    filtered = []
    error_lst = []
    for i in range(noisy_spectra.shape[1]):
        y = noisy_spectra[:,i]
        output = signal.savgol_filter(y, window_length, poly_order)
        filtered.append(output)
        error_lst.append(np.mean(abs(clean_spectra[:,i] - output)))
    filtered = np.stack(filtered, axis = 1)
    error_lst = np.stack(error_lst)

    return filtered, error_lst

# OTHER PREPARATIONS
# Normalisation
# create a scaler object
std_scaler = StandardScaler()

# Mount Google Drive for easy access to data.
drive.mount("/content/drive")


# SIMULATING AND UPLOADING DATASETS

# Pure data upload
df = pd.read_csv('normalised_water_and_ethanol.csv', index_col = None)
wavelengths = df.WL.values
water_normalised = df.Water.values
ethanol_normalised = df.Ethanol.values

# Simulate clean spectra
conc_lst = [0, 0.01, 0.02, 0.03, 0.04, 0.05]
no_of_spectra = 100000

concentrations = []
clean_spectra = []
for i in range(no_of_spectra):
    conc = random.choice(conc_lst)
    concentrations.append(conc)
    sample = (1 - conc) * water_normalised + conc * ethanol_normalised
    clean_spectra.append(sample)
clean_spectra = np.stack(clean_spectra, axis = 1)
concentrations = np.asarray(concentrations)
```

```
# Adding noise to clean spectra
mu = 0
sigma = 0.1

noisy_spectra_1 = []
for i in range(clean_spectra.shape[1]):
    noise = np.random.normal(mu, sigma, 401)
    noisy_spectra_1.append(clean_spectra[:,i] + noise)
noisy_spectra_1 = np.stack(noisy_spectra_1, axis = 1)

noisy_spectra_2 = []
for i in range(clean_spectra.shape[1]):
    mu = np.random.uniform(low = -1, high = 1)
    sigma = np.random.uniform(low = 0, high = 0.1)
    noise = np.random.normal(mu, sigma, 401)
    noisy_spectra_2.append(clean_spectra[:,i] + noise)
noisy_spectra_2 = np.stack(noisy_spectra_2, axis = 1)

'''
Generate slow systematic noise.
'''
Fs = 401
f = 10
sample_syst = 401
x_syst = np.arange(sample_syst)
y_syst = (np.sin(2 * np.pi * f * x_syst / Fs))/50

noisy_spectra_3 = []
for i in range(clean_spectra.shape[1]):
    noise = np.random.random(401)
    noisy_spectra_3.append(clean_spectra[:,i] + noise)
noisy_spectra_3 = np.stack(noisy_spectra_3, axis = 1)

noisy_spectra_4 = []
for i in range(clean_spectra.shape[1]):
    mu = np.random.uniform(low = -1, high = 1)
    sigma = np.random.uniform(low = 0, high = 0.1)
    noise = np.random.normal(mu, sigma, 401)
    noisy_spectra_4.append(clean_spectra[:,i] + noise + y_syst)
noisy_spectra_4 = np.stack(noisy_spectra_4, axis = 1)


# OPTIMISATION
'''
Butterworth filter
'''
```

```
cut_off_freq_ranges = np.linspace(0.0001, 0.4, 100)
orders = range(1,6)
overall_errs = []
for i in orders:
    interim_errs = []
    for j in cut_off_freq_ranges:
        ds1_butter, err1_butter = use_butterworth_filter(noisy_spectra_
1, clean_spectra, i, j)
        interim_errs.append(np.mean(err1_butter))
    interim_errs = np.stack(interim_errs)
    overall_errs.append(interim_errs)
overall_errs = np.stack(overall_errs)
overall_errs_2 = []
for i in orders:
    interim_errs = []
    for j in cut_off_freq_ranges:
        ds2_butter, err2_butter = use_butterworth_filter(noisy_spectra_
2, clean_spectra, i, j)
        interim_errs.append(np.mean(err2_butter))
    interim_errs = np.stack(interim_errs)
    overall_errs_2.append(interim_errs)
overall_errs_2 = np.stack(overall_errs_2)
overall_errs_3 = []
for i in orders:
    interim_errs = []
    for j in cut_off_freq_ranges:
        ds3_butter, err3_butter = use_butterworth_filter(noisy_spectra_
3, clean_spectra, i, j)
        interim_errs.append(np.mean(err3_butter))
    interim_errs = np.stack(interim_errs)
    overall_errs_3.append(interim_errs)
overall_errs_3 = np.stack(overall_errs_3)
overall_errs_4 = []
for i in orders:
    interim_errs = []
    for j in cut_off_freq_ranges:
        ds4_butter, err4_butter = use_butterworth_filter(noisy_spectra_
4, clean_spectra, i, j)
        interim_errs.append(np.mean(err4_butter))
    interim_errs = np.stack(interim_errs)
    overall_errs_4.append(interim_errs)
overall_errs_4 = np.stack(overall_errs_4)

print(np.where(overall_errs == np.min(overall_errs)))
print(np.where(overall_errs_2 == np.min(overall_errs_2)))
print(np.where(overall_errs_3 == np.min(overall_errs_3)))
print(np.where(overall_errs_4 == np.min(overall_errs_4)))
```

```
''' Savitzky-Golay filter'''
window_length_ranges = [11, 15, 19, 23, 27, 31, 35, 39, 43, 47, 51, 55,
                        59, 63, 67, 71, 75, 79, 83, 87, 91, 95, 99, 103
, 107]
orders = range(1,6)
overall_errs_svg = []
for i in orders:
    interim_errs = []
    for j in window_length_ranges:
        ds_svg, err_svg = use_svg_filter(noisy_spectra_1, clean_spectra
, j, i)
        interim_errs.append(np.mean(err_svg))
    interim_errs = np.stack(interim_errs)
    overall_errs_svg.append(interim_errs)
overall_errs_svg = np.stack(overall_errs_svg)
overall_errs_2_svg = []
for i in orders:
    interim_errs = []
    for j in window_length_ranges:
        ds_svg, err_svg = use_svg_filter(noisy_spectra_2, clean_spectra
, j, i)
        interim_errs.append(np.mean(err_svg))
    interim_errs = np.stack(interim_errs)
    overall_errs_2_svg.append(interim_errs)
overall_errs_2_svg = np.stack(overall_errs_2_svg)
overall_errs_3_svg = []
for i in orders:
    interim_errs = []
    for j in window_length_ranges:
        ds_svg, err_svg = use_svg_filter(noisy_spectra_3, clean_spectra
, j, i)
        interim_errs.append(np.mean(err_svg))
    interim_errs = np.stack(interim_errs)
    overall_errs_3_svg.append(interim_errs)
overall_errs_3_svg = np.stack(overall_errs_3_svg)
overall_errs_4_svg = []
for i in orders:
    interim_errs = []
    for j in window_length_ranges:
        ds_svg, err_svg = use_svg_filter(noisy_spectra_4, clean_spectra
, j, i)
        interim_errs.append(np.mean(err_svg))
    interim_errs = np.stack(interim_errs)
    overall_errs_4_svg.append(interim_errs)
overall_errs_4_svg = np.stack(overall_errs_4_svg)
print(np.where(overall_errs_svg == np.min(overall_errs_svg)))
print(np.where(overall_errs_2_svg == np.min(overall_errs_2_svg)))
print(np.where(overall_errs_3_svg == np.min(overall_errs_3_svg)))
print(np.where(overall_errs_4_svg == np.min(overall_errs_4_svg)))
```

```
# DATA PREPARATION FOR THE AUTOENCODER

noisy_simulated_spectra_array = np.concatenate((noisy_spectra_1, noisy_
spectra_2, noisy_spectra_3), axis = 1)
simulated_spectra = np.concatenate((clean_spectra, clean_spectra, noisy
_spectra_3), axis = 1)
concentrations = np.concatenate((concentrations, concentrations, concen
trations))

# Check shapes of all datasets
print('Simulated spectra shape is: ' + str(simulated_spectra.shape))
print('Simulated noisy spectra shape is: ' + str(noisy_simulated_spectr
a_array.shape))

# Reshape the arrays to fit the models
simulated_spectra = simulated_spectra.reshape([1, 401, -
1]).transpose(2, 1, 0)
noisy_simulated_sp = noisy_simulated_spectra_array.reshape([1, 401, -
1]).transpose(2, 1, 0)

# Check shapes of all datasets again
print('Simulated spectra shape is: ' + str(simulated_spectra.shape))
print('Simulated noisy spectra shape is: ' + str(noisy_simulated_sp.sha
pe))

# AUTOENCODER
# Model configuration
width, height = 401, 1
input_shape = (width, height)
batch_size = 128
no_epochs = 50
max_norm_value = 2.0

X_train, X_test, y_train, y_test = train_test_split(noisy_simulated_sp,
 simulated_spectra/4, test_size=0.1)

ds_train = tf.data.Dataset.from_tensor_slices((X_train, y_train))
ds_train = ds_train.shuffle(buffer_size=len(X_train)).batch(128)

ds_validation= tf.data.Dataset.from_tensor_slices((X_test, y_test)).bat
ch(128)
```

```
# Create the model
k_size = 9
model = Sequential()

model.add(Conv1D(256, kernel_size=(k_size), kernel_constraint=max_norm(
max_norm_value), activation='relu', kernel_initializer='he_uniform', in
put_shape=input_shape))

model.add(Conv1D(128, kernel_size=(k_size), kernel_constraint=max_norm(
max_norm_value), activation='relu', kernel_initializer='he_uniform', in
put_shape=input_shape))

model.add(Conv1D(32, kernel_size=(k_size), kernel_constraint=max_norm(m
ax_norm_value), activation='relu', kernel_initializer='he_uniform'))

model.add(Conv1DTranspose(32, kernel_size=(k_size), kernel_constraint=m
ax_norm(max_norm_value), activation='relu', kernel_initializer='he_unif
orm'))

model.add(Conv1DTranspose(128, kernel_size=(k_size), kernel_constraint=
max_norm(max_norm_value), activation='relu', kernel_initializer='he_uni
form'))

model.add(Conv1DTranspose(256, kernel_size=(k_size), kernel_constraint=
max_norm(max_norm_value), activation='relu', kernel_initializer='he_uni
form'))

model.add(Conv1D(1, kernel_size=(k_size), kernel_constraint=max_norm(ma
x_norm_value), activation='tanh', padding='same'))

model.summary()

# Compile and fit data
model.compile(optimizer='adam', loss='mae')
model.fit(ds_train,
                epochs=no_epochs,
                callbacks=[keras.callbacks.TensorBoard(log_dir='logs')]
,
                validation_data = ds_validation)

# Save the weights
model.save_weights('./checkpoint')
```

```
# Upload measured data
# This dataset is already balanced
measured_spectra = pd.read_csv('/content/drive/My Drive/Data/measuremen
ts_for_masters.csv')
clean_sp = pd.read_csv('/content/drive/My Drive/Data/clean_sp_for_maste
rs.csv')

concentrations_measured = measured_spectra.conc.values
measurements = pd.DataFrame(std_scaler.fit_transform(measured_spectra.v
alues[:,:-1].T)).values
clean_for_measurements = pd.DataFrame(std_scaler.fit_transform(clean_sp
.values[:,1:-1].T)).values

# Reshape the arrays to fit the models
measured_spectra = measurements.reshape([1, 401, -
1]).transpose(2, 1, 0)
clean_for_measured = clean_for_measurements.reshape([1, 401, -
1]).transpose(2, 1, 0)

# Check shapes of all datasets again
print('Measured spectra shape is: ' + str(measured_spectra.shape))
print('Clean measured spectra shape is: ' + str(clean_for_measured.shap
e))

# Assign variables
noisy_input_test = measured_spectra
pure_input_test = clean_for_measured

# Generate reconstructions
num_reconstructions = noisy_input_test.shape[0]
samples = noisy_input_test[:num_reconstructions]
reconstructions = model.predict(samples)

# FILTERING
# Filter the dataset with Butterworth and Savitzky-Golay filters
ds1_butter, err1_butter = use_butterworth_filter(samples.squeeze().T, p
ure_input_test[:samples.shape[0],:].squeeze().T, 2, 0.045)
ds1_svg, err1_svg = use_svg_filter(samples.squeeze().T, pure_input_test
[:samples.shape[0],:].squeeze().T, 107, 3)

# Plot reconstructions of the DAE
# At the same time calculate the mean absolute errors
```

```
err_lst = []
plt.figure()
for i in np.arange(0, num_reconstructions):
  # Prediction index
  prediction_index = i
  # Get the sample and the reconstruction
  original = noisy_input_test[prediction_index]
  pure = pure_input_test[prediction_index]
  reconstruction = np.array(reconstructions[i]).reshape((width,))

  abs_err = np.mean(np.abs(pure.squeeze() -  (reconstruction.squeeze()
* 4)))
  err_lst.append(abs_err)
  plt.plot(pure.squeeze(), label = 'Pure', lw = 0.75)
  plt.plot(reconstruction.squeeze() * 4, label = 'De-noised', lw = 2)
err_lst = np.asarray(err_lst)
plt.grid(True)
plt.xlabel('Wavelengths (nm', fontsize = 15)
plt.show()

# Graph for comparing the MAEs of the three models
plt.figure(figsize = (9,7))
plt.title('MAEs of the filtering models', fontsize = 15)
plt.plot(err_lst, 'b', label = 'DAE')
plt.plot(err1_butter, 'r', label = 'Butterworth')
plt.plot(err1_svg, 'g', label = 'Savitzky-Golay')
plt.ylabel('Mean absolute error (MAE)', fontsize = 12)
plt.xlabel('Spectrum number', fontsize = 12)
plt.legend(fontsize = 12)
plt.grid(True)
plt.show()

# PARTIAL LEAST SQUARES REGRESSION (PLSR)
# Denoising Autoencoder
X_dae_1 = reconstructions.squeeze()
Y = concentrations_measured
X_train_dae_1, X_test_dae_1, Y_train_dae_1, Y_test_dae_1 = train_test_s
plit(X_dae_1, Y, test_size = 0.2)
no_of_components = 10
pls = PLSRegression(no_of_components)
pls.fit_transform(X_train_dae_1, Y_train_dae_1)
Y_pred_test_1 = pls.predict(X_test_dae_1)
r2_test_1 = r2_score(Y_test_dae_1, Y_pred_test_1)
RMSEP_test_1 = np.sqrt(mean_squared_error(Y_test_dae_1, Y_pred_test_1))
```

```
# Butterworth Filter
X_butter = ds1_butter.squeeze().T
X_train_butter, X_test_butter, Y_train_butter, Y_test_butter = train_te
st_split(X_butter, Y, test_size = 0.2)
pls.fit_transform(X_train_butter, Y_train_butter)
Y_pred_test_butter = pls.predict(X_test_butter)
r2_test_butter = r2_score(Y_test_butter, Y_pred_test_butter)
RMSEP_test_butter = np.sqrt(mean_squared_error(Y_test_butter, Y_pred_te
st_butter))

# Savitzky-Golay Filter
X_svg = ds1_svg.squeeze().T
X_train_svg, X_test_svg, Y_train_svg, Y_test_svg = train_test_split(X_s
vg, Y, test_size = 0.2)
pls.fit_transform(X_train_svg, Y_train_svg)
Y_pred_test_svg = pls.predict(X_test_svg)
r2_test_svg = r2_score(Y_test_svg, Y_pred_test_svg)
RMSEP_test_svg = np.sqrt(mean_squared_error(Y_test_svg, Y_pred_test_svg
))

# Print the results of the PLSR
print('Raw r-squared ', round(r2_test, 4))
print('Raw RMSEP ', round(RMSEP_test, 4))
print('Butterworth r-squared ', round(r2_test_butter, 4))
print('Butterworth RMSEP ', round(RMSEP_test_butter, 4))
print('Savizky-Golay r-squared ', round(r2_test_svg, 4))
print('Savizky-Golay RMSEP ', round(RMSEP_test_svg, 4))
print('DAE r-squared ', round(r2_test_1, 4))
print('DAE RMSEP ', round(RMSEP_test_1, 4))
```