Bachelor's Thesis

Denoising of Blood-based Infrared Spectroscopic Fingerprints with Deep Learning

Department of Physics
Ludwig-Maximilians-Universität München

Corinna Elena Wegner

Munich, March 2nd, 2023

Submitted in partial fulfillment of the requirements for the degree of B. Sc.
Supervised by Dr. Kosmas Kepesidis
Co-supervisor: Prof. Dr. Ferenc Krausz

Bachelorarbeit

Rauschentfernung aus blutbasierenden infrarot-spektroskopischen Fingerabdrücken mittels Deep Learning

Fakultät für Physik
Ludwig-Maximilians-Universität München

Corinna Elena Wegner

München, 2. März 2023

Eingereicht als Teilleistung zur Erlangung des Grades B. Sc.
Betreut durch Dr. Kosmas Kepesidis
Zweitgutachter: Prof. Dr. Ferenc Krausz

Abstract

This thesis describes the development and testing of an autoencoder which denoises FTIR-spectra of human blood samples for classification of cancer and non-cancer patients. The first two chapters give a brief overview of the fundamentals behind FTIR-spectroscopy and provides an introduction to machine learning with focus on deep learning.

In the main part the process to find a model architecture adequate to denoising the spectra is described. On the way, we explore our data and investigate simple autoencoder architectures. Then we make the autoencoder reproduce and denoise the spectra in a way that the relevant information is preserved. We will try both fully-connected neural networks and convolutional models and decide for the last one eventually. At the end of this section, we will make a comparison of the autoencoder with the standard preprocessing method used upon the spectra. This shows us that the autoencoder-preprocessed spectra are classified better than the standard-preprocessed spectra.

The final evaluation is divided into two parts. In the first part, we investigate the ability of the autoencoder to denoise white noise and compare it to standard preprocessing of the spectra. The other part is about denoising measurement drifts with our model. We will see that with an engineered loss function, the autoencoder is able to align the silent regions of the spectra with an arbitrary spectrum and simultaneously reproduce the other part of the spectrum to a certain extend. This opens up possiblities for further investigation on a transformation to remove measurement drifts of the whole spectrum in the future.

Contents

List of Figures

1 Introduction

Infrared radiation (IR) was discovered 1800 by Sir William Herschel, who was investigating the spectrum of the sun by splitting it with a prism. He measured the temperature from multiple points in the spectrum and found out that the maximum temperature was surprisingly not at the brightest point, instead it was in a region that was later named infrared. But further examination of the wavelengths of this radiation had to wait until Langley invented the bolometer in 1880. With his reststrahlen method, Rubens was able to discover that infrared radiation does continuously merge with visible light in the electromagnetic spectrum. The invention of the echelette grating in the beginning of the last century formed the basis for measuring IR spectra with higher resolution. But even then, it took multiple hours to measure one single spectrum. Therefore, until this discovery was usable for spectral analysis, one had to wait until 1937, when Lehrer developed the first fully-automatic spectral photometer. From then on the development proceeded rapidly. In 1950, measuring an IR-spectrum only took about a few minutes. In the 1960's the fundamentals of infrared spectroscopy combined with fourier transformation, which allowed measuring the entire IR range from wavelengths between 780 nm and 1000 μm. With the rapid improvement of Computers in the 1980's, the success of fourier transform infrared (FTIR) spectroscopy became unstoppable (Günzler and Gremlich, 2003).

Meanwhile, researchers concerned themselves with another future success story. Artificial neural networks are machine learning models inspired by the brain. The idea sparked in 1943 from McCulloch and Pitts, who presented their model of the interaction between biological neurons. After some stagnacy, in the 80's and 90's improved training techniques led to recurring interest in this topic. Artificial neural networks are applicable for highly complex problems in machine learning and form the basis of deep learning. In the last years, the technology of deep learning has increased rapidly due to availability of large data sets for training and high computation power. Well known examples are image classification models like Google Images, speech recognition neural networks (Apple Siri) or the YouTube video recommendation algorithm (Géron, 2020). A highly debated deep learning model that has currently been released is ChatGPT (Hern, 2022), an AI that could potentially write an entire Bachelor's thesis.

In this thesis we will use a special deep learning architecture called autoencoders, to denoise FITR-spectroscopic measurements and thus increase classification performance between cancer and non-cancer patients. Autoencoders have already been successfully used to denoise electrocardiogram signals to improve cardiovascular disease detection in Chiang et al. (2019). A very impressive demonstration for the ability of autoencoders to denoise signals can also be found in Erdmann et al. (2019). This example illustrates how small signals can be recovered from high noise.

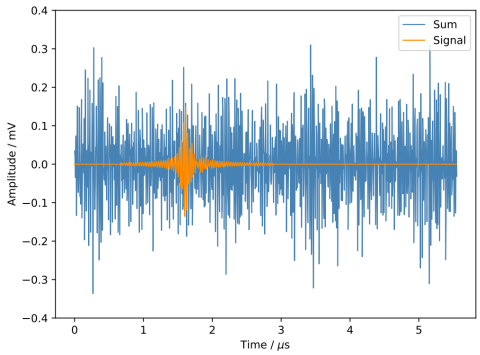


Figure 1: Detection of radio signals within a background of cosmic ray noise via deep learning (Erdmann et al., 2019)

As it has been pointed out, autoencoders have been sucessfully applied in several research fields. Therefore in Chapter 3, we successfully use them for the denoising of blood-based infrared spectra.

2 Concepts

In this chapter, the basic concepts of the project are introduced. First, the theoretical background of the measurement is explained. In the second part, the machine learning algorithms used to evaluate the results are presented. Based on the machine learning part, the idea behind deep learning is then presented in more detail.

2.1 FTIR-spectroscopy

2.1.1 Electromagnetic waves

Waves are oscillations that propagate through space. They are described by various parameters. The length between two maxima of the wave is called wavelength λ. For a fixed point in space, the time T between two maxima is called period length. Its inverse is the frequency ν, i.e. the number of oscillations per time. Commonly used is the angular frequency $\omega = 2\pi\nu$ (Demtröder, 2018).

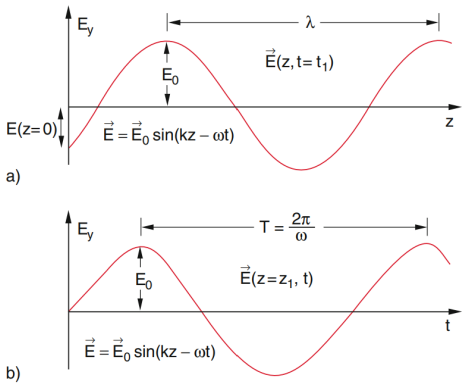


Figure 2: Representation of a wave a) in space b) in time (Demtröder, 2018)

The velocity of propagation is related with the angular frequency through the wavenumber k via $v = \frac{\omega}{k}$. The parameters are related with the velocity of propagation via (Griffiths, 2018, chapter 9)

$$k = \frac{\omega}{v} = \frac{2\pi}{\lambda} \tag{1}$$

Electromagnetic waves in vacuum are described by the maxwell equations (Griffiths, 2018, chapter 9):

$$\begin{aligned} \vec{\nabla} \cdot \vec{E} = 0, \quad \vec{\nabla} \times \vec{E} = -\frac{\partial \vec{B}}{\partial t}, \\ \vec{\nabla} \cdot \vec{B} = 0, \quad \vec{\nabla} \times \vec{B} = \mu_0 \epsilon_0 \frac{\partial \vec{E}}{\partial t}. \end{aligned} \tag{2}$$

Here, $\vec{E}$ is an electric field, $\vec{B}$ is a magnetic field, $\mu_0 = 4\pi \cdot 10^{-7} \frac{N}{A^2}$ is the permeability of free space and $\epsilon_0 = 8.85 \cdot 10^{-12} \frac{C^2}{Nm^2}$ is the vacuum permittivity. $\vec{\nabla} = \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{pmatrix}$ is the nabla operator (Griffiths, 2018). Deformation of the equations in 2 leads to the wave equations (Griffiths, 2018, chapter 9):

$$\begin{aligned} \vec{\nabla}^2 \vec{E} &= \mu_0 \epsilon_0 \frac{\partial^2 \vec{E}}{\partial t}, \\ \vec{\nabla}^2 \vec{B} &= \mu_0 \epsilon_0 \frac{\partial^2 \vec{B}}{\partial t} \end{aligned} \tag{3}$$

Solving the wave equations yields (Griffiths, 2018, chaper 9):

$$\begin{aligned} \vec{\tilde{E}}(z,t) &= \vec{\tilde{E}_0} e^{i(kz - \omega t)} \\ \vec{\tilde{B}}(z,t) &= \vec{\tilde{B}_0} e^{i(kz - \omega t)} \end{aligned} \tag{4}$$

with $\vec{E}$ and $\vec{B}$ being the real parts of $\vec{\tilde{E}}(z,t)$ and $\vec{\tilde{B}}(z,t)$.

Equations 4 describes an electromagnetic wave that is propagating in the z-direction. From equation 3 one can deduce that in vacuum, electromagnetic waves propagate with a velocity of (Griffiths, 2018)

$$c = \frac{1}{\sqrt{\mu_0 \epsilon_0}} = 3.0 \cdot 10^8 \frac{m}{s} \tag{5}$$

Electromagnetic waves are transversal waves, meaning that the directions of the fields are perpendicular to the direction of propagation. $\vec{E}$ and $\vec{B}$ are also perpendicular (see Figure 3). The amplitudes $\vec{B_0}$ and $\vec{E_0}$ are connected via (Griffiths, 2018, chapter 9)

$$\vec{B_0} = \frac{1}{c} \vec{E_0} \tag{6}$$

Therefore the electric field is much stronger than the magnetic field.

2.1.2 Infrared radiation and molecule oscillation

In the following section a specific type of electromagnetic waves and its application to the scientific project is introduced.

FTIR-spectroscopy works with infrared radiation. These are electromagnetic waves whose wavelength λ ranges from 700 nm to 100 μm (Demtröder, 2019).

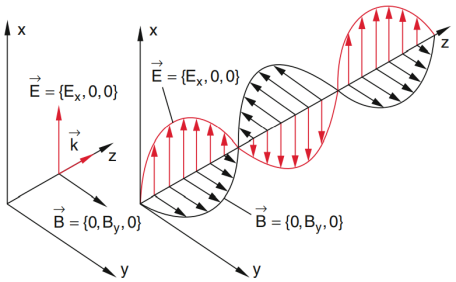


Figure 3: Electromagnetic waves (Demtröder, 2017)

In quantum mechanics, the energy of a system is described by its wave function ψ and Hamiltonian $H = T + V$, with T being the kinetic energy and V the potential energy. The energy E is then the expectation value $E = \langle\psi|H|\psi\rangle$. To find the corresponding ψ one has to solve the Schrödinger equation (Parson, 2009):

$$H\psi = i\hbar\frac{\partial\psi}{\partial t} \tag{7}$$

Diatomic molecules can be described by the quantum harmonic oscillator (Brooks, 2013). The potential V for the harmonic oscillator is given by (Parson, 2009)

$$V(x) = \frac{1}{2}kx^2 \tag{8}$$

,
where $x = 0$ is the rest position of the system. k is a constant that describes the force $F = -kx$ that acts on the system if it is ouside of the rest position. Solving the Schrödinger equation yields the discrete energy levels of the quantum harmonic oscillator (Parson, 2009):

$$E_n = (n + \frac{1}{2})h\nu \tag{9}$$

with the frequency $\nu = \sqrt{\frac{k}{m}}\frac{1}{2\pi}$ and the Planck constant h. (Parson, 2009).

A molecule with n atoms can also be has a total of 3N degrees of freedom, three for translation, three for rotation and the rest for internal vibration. In figure 4 an exemplary vibrating molecule is drafted. Simplified, the total vibrational potential is given by (Parson, 2009)

$$V_{vibration} = \frac{1}{2}\sum_{i}^{3N-6} k_i x_i^2 \tag{10}$$

, with normal coordinates x_i. Analogously to the quantum mechanical harmonic oscillator, one obtains the discrete vibrational energy levels (Parson, 2009)

$$E_{vibration} = \sum_{i=1}^{3N-6} (n_i + \frac{1}{2})h\nu_i \tag{11}$$

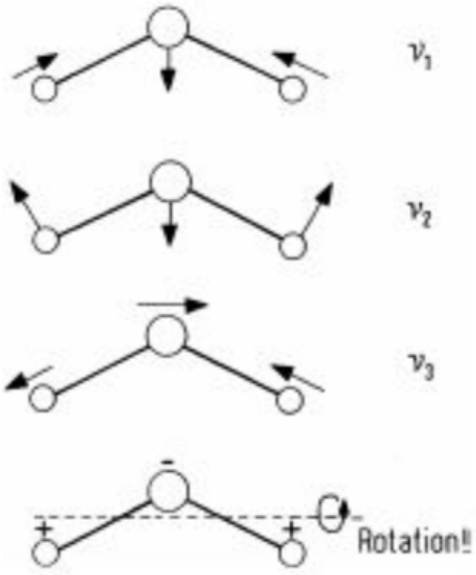


Figure 4: Oscillations of an angulate three-atomic molecule (Günzler and Gremlich, 2003)

with the frequency $\nu_i = \frac{\sqrt{k/m_r}}{2\pi}$. n is the quantum number and is 1 in ground state, but can be any natural number via excitation. (Parson, 2009).

When exposing a molecule to electromagnetic waves with frequency $\nu = \frac{\triangle E}{h}$, where $\triangle E$ is the energy difference from two vibrational energy levels, the molecule can be excited by the wave to the upper energy state. Measuring those absorptions of energy can provide information about the molecular structure. The frequencies required for excitations of molecules are located mostly in the mid infrared spectral region (Parson, 2009).

2.1.3 Structure of the FTIR-spectrometer

The main part of the FTIR-spectrometer is the michelson interferometer (Lambert et al., 2012). The structure of a michelson interferometer can be seen in figure 5. A beam of electromagnetic waves coming from a source E_I is directed towards a beam splitter BS, from where it is split and sent to mirror M_1 , covering the distance s_1, and to the movable mirror M_2 via the variable length s_2. At the mirrors, the beams are reflected and redirected to BS, passing again s_1 and s_2 respectively. Both beams are split again at BS and are partly redirected towards E_1 and the detector B (Demtröder, 2019).
On the way from BS to B (and BS to E_1), the overlapping beams interfere. If $s_1 = s_2$, both beams propagated through the same distance as they reach the detector, leading to positive interference (maximal signal enhancement). If mirror s_2 is moved by the distance δ away from that state, one beam has to go a $x = 2\delta$ longer distance. This leads to a phase difference between the two beams (Lambert et al., 2012).
A sample of substance is then inserted between BS and B (Bienz et al., 2012). Comparing the transmitted intensity with (I) and without (I_0) the sample, one can obtain the resulting absorbance A via (Lambert et al., 2012):

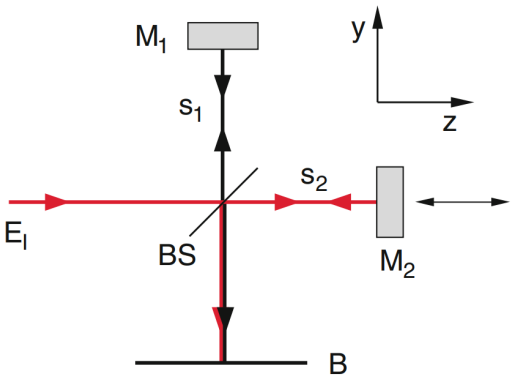


Figure 5: Michelson Interferometer (Demtröder, 2018)

$$T = \frac{I}{I_0}, \; A = log_{10}\frac{1}{T} \tag{12}$$

T is called "transmission". The absorption is proportional to the concentration c of the substance and the length d that the beam passes through the sample. The proportionality factor is the extinction coefficient ϵ and the corresponding equation is the Lambert-Beer-Bouguer-Law (Lambert et al., 2012)

$$A = \epsilon c d \tag{13}$$

Thermal sources with a Temperature of 2000 K are the origin for infrared radiation (Demtröder, 2019). In infrared spectroscopy, the beam is usually produced by radiators which can be described by Planck's law (Günzler and Gremlich, 2003). For a given frequency ν and temperature T, the black-body radiation is given by (Andrews, 2010)

$$B_\nu(T) = \frac{2h\nu^3}{c^2(exp[h\nu/k_bT] - 1)} \tag{14}$$

.
k_b is the Boltzmann constant, h the Planck constant (Andrews, 2010). The emitted spectrum of frequencies is therefore continuous (Günzler and Gremlich, 2003).
While the measurement of the detector is taken, the mirror M_2 is shifted in one direction (Lambert et al., 2012). The obtained measurement is an interferogram (equation 15), which is then converted to an infrared-spectrum $B(k)$ via fourier-transform (Günzler and Gremlich, 2003):

$$I(\delta) = \int_{-\infty}^{\infty} B(k)\exp(-2\pi ik\delta)d\tilde{\nu}. \tag{15}$$

There are three major advantages from FTIR above other infrared spectroscopic methods. First, through measuring all frequencies simultaneously, time is saved. Second, the signal-to-noise-ratio is relatively good (Jacquinot-advantage). Besides, the precision of wavenumbers is high due to a special gauge method (Connes-advantage) (Bienz et al., 2012)).

2.2 Classical Machine Learning algorithms

Via machine learning it is possible to make computers solve problems without providing a predetermined solution. In model-based learning, an algorithm trains a model based on a given dataset. A model is a function that provides an answer to a problem. For example, if the question is how happy people are depending on the gross domestic product (GDP) of their country, a possible model could be (Géron, 2020, chap. 1):

$$happiness = \theta_0 + \theta_1 \times GDP \tag{16}$$

θ_0 and θ_1 are calles the parameters of the model. The algorithm delivers appropriate values for the parameters (Géron, 2020, chap. 1).

In the previous example, the model is chosen under the assumption that the relation of the two variates is linear, as equation 16 is a linear function. In general, one first needs to choose between many possible models based on their assumptions for the data. If a chosen model is too complex for the problem, it might be that the model learns meaningless features. This occurrence is called overfitting. Overfitting can also appear if the data is noisy, i.e. contains information that are not helpful to solve the problem (Géron, 2020, chap. 1).

One distinguishes between various machine learning systems, depending on the labelling of the data used in the learning procedure. If each observation in the dataset is labelled, this is called supervised learning. If there are no labels included, one speaks of unsupervised learning. Else, semi-supervised learning means learning with partially labeled data (Géron, 2020, chap. 1).

2.2.1 Support Vector Machine

Support Vector Machine (SVM) is a technique from machine learning used, among other things, to classify data. The SVM returns a classification boundary (support vectors) that is optimized to seperate data points of different classes (support Vectors). However, there can also be observations within the street (bounded by the dashed lines in figure 6) or on the wrong side of the classification boundary. This is due to the conflicting goals of the SVM to firstly increase the width of the street and secondly try to classify many data points correcly. If the SVM follows both goals, the support vectors are usually not data points at the edge of the classes - allowing errors in classification. In this case, one speaks of soft-margin-classification. The width of the street also depends highly on the scale of the data (Géron, 2020, chap. 5).

A linear SVM returns a straight as classification boundary. The decision function $h(\mathbf{x})$ is defined as (Géron, 2020, chap. 5)

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = w_1x_1 + ... + w_nx_n + b \tag{17}$$

$\mathbf{w}$ is the n-dimensional weight vector that is multiplied with the data point $\mathbf{x}$. $h = 0$ marks the center of the classification boundary. At $h = \pm 1$ are the margins of that boundary.

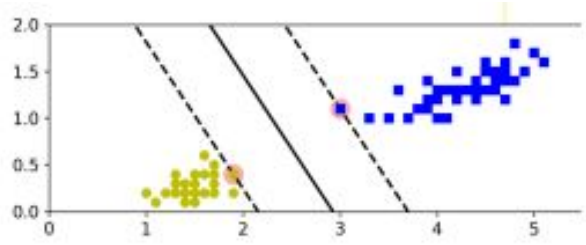


Figure 6: Classification via SVM (Géron, 2020, chap. 5)

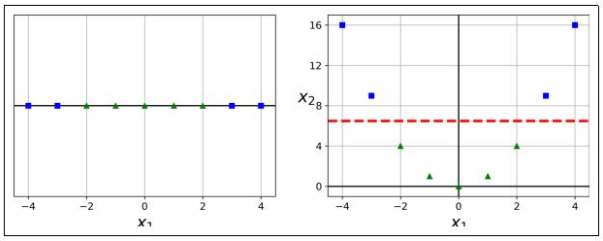


Figure 7: Transformation of data for SVM (Géron, 2020, chap. 5)

If for a new data point $\mathbf{y}$ we get $h(\mathbf{y}) > 1$, it is assigned to class 1, if $h(\mathbf{y}) < -1$, it is class 2 (Géron, 2020, chap. 5). To increase the margin, the values of $\mathbf{w}$ are trained such that the norm $||\mathbf{w}||$, is minimized. The conditions can be summarized in the following optimization problem which is addressed during training (Géron, 2020, chap. 5):

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{m}\zeta^{(i)} \\ &\text{under the condition} && t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - \zeta^{(i)} \\ &\text{and} && \zeta^{(i)} \geq 0 \ \forall i = 1, 2, ..., m. \end{aligned} \tag{18}$$

C is a hyper parameter that controls the balance between both goals, $\zeta^{(i)}$ is the so called Slack-variable, which determines the importance for the data point i to stay outside the margin of the street. $t^{(i)}$ is either 1 or -1 and ensures that each data point contributes equally (positive) to the problem (Géron, 2020, chap. 5).

Sometimes, a dataset cannot be split by a straight without performing a transformation on the data first. An example can be seen in 7. Here, the observations x_1 are first squared and can then seperated using linear SVM (Géron, 2020, chap. 5).
There are lots of possible transformations to seperate the data points in a way that a linear SVM is possible. Another example is the gaussian radial basis function (RBF), which is defined as (Géron, 2020, chap. 5)

$$\phi_\gamma(\mathbf{x}, l) = \exp(-\gamma||\mathbf{x} - l||^2) \tag{19}$$

2.2.2 Principal Component Analysis

Principal Component Analysis (PCA) is a method to reduce the dimensionality of n-dimensional data while keeping as much of the information as possible (Géron, 2020, chap. 8).

Via PCA the data is projected into a hyperplane with dimensionality $d < n$ whose axis are chosen such that the maximum of the variance is kept in the representation. In the below figure, a $n = 2$-dimensional dataset is depicted. The axis c_1 and c_2 are the so-called "principal components". On the right plot, the data is projected onto the subspace of the principal components. The principal components are ordered based on how much variance the data in this projection has (Géron, 2020, chap. 8).

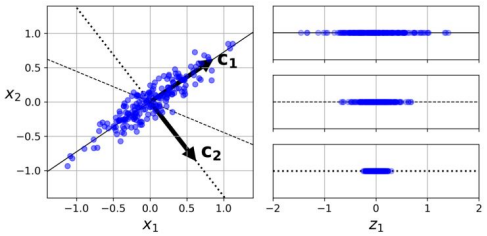


Figure 8: Principal Component Analysis (Géron, 2020, chap. 8)

The procedure how the principal components for a specific set of data is found is called "Singular value decomposition (SVD)". It finds a matrix representation $\mathbf{V}$ for a given data matrix $\mathbf{X}$, such that $\mathbf{V}$ contains the principal components and $\mathbf{X}$ can be decomposed in the following way (Géron, 2020, chap. 8):

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathbf{T}} \tag{20}$$

In order to find a SVD for a $(m \times n)$-matrix $\mathbf{X}$ one has to find an orthonormal basis of eigenvectors $(\vec{v_1}, \vec{v_2}, ..., \vec{v_n})$ from $\mathbf{C} = \mathbf{X}^{\mathbf{T}}\mathbf{X}$. This is done by first solving the eigenvalue problem (Knabner and Barth, 2013):

$$\mathbf{C}\vec{v} = \lambda\vec{v} \tag{21}$$

The eigenvalues $\lambda_1, \lambda_2, ...$ can be obtained by solving the equation (Knabner and Barth, 2013)

$$det(\mathbf{C} - \lambda\mathbf{1}) = 0 \tag{22}$$

.
Then, from equation 21 one can derive $(\vec{v_1}, \vec{v_2}, ..., \vec{v_n})$. One must normalize the vectors. $\mathbf{V}$ is then given by $\mathbf{V} = (\vec{v_1}, \vec{v_2}, ..., \vec{v_n})$. $\mathbf{U}$ is obtained analogously from $\mathbf{X}\mathbf{X}^{\mathbf{T}}$. The eigenvectors also yield $\mathbf{\Sigma} = diag(\sigma_1, ..., \sigma_n)$ via $\sigma_i = \sqrt{\lambda_i}$ and $\lambda_i := 0$ for $rank(A) < i \leq n$ (Knabner and Barth, 2013). The projection of the data onto the hyperplane is then done

by extracting the matrix W_d with the first d principal components from $\mathbf{V}$ and calculating the representation X_d via (Géron, 2020):

$$\mathbf{X_d} = \mathbf{X}\mathbf{W_d} \tag{23}$$

The eigenvalues also predicate how much of the information or variance is stored in one principal component. Dividing the eigenvalue by the number of dimensions and then cumulating over all principal components yields the proportion of information from the extracted principal components (Backhaus et al., 2021).

2.2.3 t-Distributed Stochastic Neighbor Embedding

t-distributed stochastic neighbor embedding (t-SNE) is a method to visualize data of high dimensionality by mapping the data points into two or three-dimensional space. It has the ability to not only capture the local structure of the data, but also depict the structure of the whole dataset (van der Maaten and Hinton, 2008).

The main idea of t-SNE is to interpret the distances of points x_i and x_j in the high-dimensional space as a probability p_{ij}, which denotes the similarity of the points. In the map, the representations of the points are denoted as y_i and y_j and have a probability q_{ij}. To make the map as similar to the data as possible, the goal is to make q_{ij} equal to p_{ij}. Hence, the function that is minimized from the algorithm is the Kullback-Leibler divergence (van der Maaten and Hinton, 2008):

$$KL(P||Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}}. \tag{24}$$

The probability p_{ij} is defined as follows: a normal distribution with variance σ is imagined around x_i, then p_{ij} is the value of that normal distribution at the position of x_j. Mathematically, it is defined as (van der Maaten and Hinton, 2008):

$$p_{ij} = \frac{exp(-||x_i - x_j||^2/2\sigma^2)}{\sum_{k \neq l} -||x_k - x_l||^2/2\sigma^2}. \tag{25}$$

σ is optimized by t-SNE based on the probability distribution P_i it generates over all data points. When there are many points in the region, a smaller σ could be more appropriate because the points are more similar (van der Maaten and Hinton, 2008).

The perplexity $Perp(P_i)$ is a parameter that describes the "smoothness" of the region and has to be chosen manually. It is a function of the Shannon entropy $H(P_i)$ and is given by (van der Maaten and Hinton, 2008):

$$\begin{aligned} Perp(P_i) &= 2^{H}(P_i) \\ H(P_i) &= -\sum_j p_{j|i} log_2 p_{j|i}. \end{aligned} \tag{26}$$

Since there is much fewer space in the two-dimensional space, it is difficult to precisely map all distances of the higher dimensional data. This problem is referred to as "crowding problem". To decline this problem, instead of using a gaussian distribution for the low-dimensional mappings as well, the distribution put around y_i is a Student t-distribution with one degree of freedom (van der Maaten and Hinton, 2008):

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-}1}{\sum_{k \neq l}(1 + ||y_i - y_l||^2)^{-}1} \tag{27}$$

Thus moderately distant points in the high-dimensional space are not extremely far away through the change in scaling in the map (van der Maaten and Hinton, 2008).

2.3 Deep learning methods

Deep learning is a subcategory of machine learning (Erdmann et al., 2021).

The smallest part of a Neural network is the neuron. The first artificial neuron was developed in 1943 by McCulloch and Pitts. Later, the Threshold Logic Unit was developed. It is a node that performs an affine mapping: it takes values $x_1, x_2, ...$ as input, multiplies them with weights $w_1, w_2, ...$, computes the sum $z = w_1x_1 + w_2x_2 + ... + b$ with an added bias b. z is then forwarded to a non-linear activation function h (Géron, 2020). The most commonly used activation functions can be seen in figure 9.

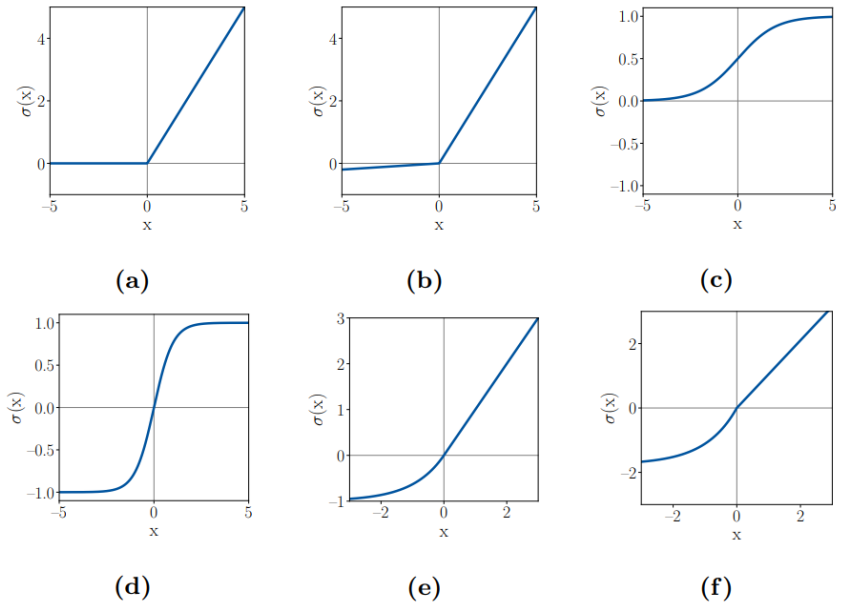


Figure 9: Activation functions: a) ReLU b) leaky ReLU c) sigmoid d) hyperbolic tangent e) ELU f) SELU (Erdmann et al., 2021)

A network is built up of connected nodes. They are arranged in different layers. The first layer is the input layer, followed by hidden layers and one output layer. Such an

architecture can be seen in figure 10 (Erdmann et al., 2021). If a neural network has multiple hidden layers, it is called a deep neural network (Géron, 2020).

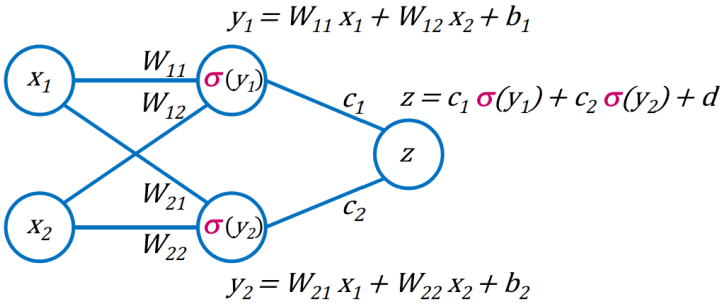


Figure 10: Structure of a neural network (Erdmann et al., 2021)

Deep learning models can be designed to perform different tasks, such as regression and classification. The "Universal approximation theorem" states that for any physics-related problem with a corresponding function, that function can be approximated by a neural network "to arbitrary precision" (Erdmann et al., 2021).

The key to training a neural network is the objective function. The objective function is the measure to evaluate the prediction of the network. There are multiple standard objective functions, that are also referred to as "loss functions". For regression tasks, the most important one is the "mean squared error". For two dimensional data, network predictions $\vec{(f)}$ and target values y, it is defined by (Erdmann et al., 2021):

$$\mathcal{L} = MSE = \frac{1}{k}\sum_{i=1}^{k}\sum_{j=1}^{m}(f_j(x_i) - y_j(x_i))^2 \tag{28}$$

The goal of the training is to find a minimum of $\mathcal{L}$ by adapting the weight and bias parameters. This is achieved via a method called gradient descent. The gradients are found via backpropagation, where the partial derivatives of the loss function are calculated (Erdmann et al., 2021):

$$z_1 = Wx, z_2 = b + z_1, z_3 = \sigma(z_2), \mathcal{L} = (y - z_3)^2 \tag{29}$$

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial z_3} \cdot \frac{\partial z_3}{\partial z_2} \cdot \frac{\partial_2}{\partial z_1} \cdot \frac{\partial z_1}{\partial W} \tag{30}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial z_3} \cdot \frac{\partial z_3}{\partial z_2} \cdot \frac{\partial_2}{\partial b} \tag{31}$$

In stochastic gradient descent, the gradients are calculated and averaged over k training events (Erdmann et al., 2021):

$$E[\frac{\partial \mathcal{L}}{\partial W}] = \frac{1}{k}\sum_{i=1}^{k}(\frac{\partial \mathcal{L}}{\partial W})_i$$
$$E[\frac{\partial \mathcal{L}}{\partial b}] = \frac{1}{k}\sum_{i=1}^{k}(\frac{\partial \mathcal{L}}{\partial b})_i \tag{32}$$

After each training iteration t, the parameters are then updated, depending on a hyper-parameter α, the "learning rate", as follows (Erdmann et al., 2021):

$$W_{t+1} = W_t - \alpha E[\frac{\partial \mathcal{L}}{\partial W}]$$
$$b_{t+1} = b_t - \alpha E[\frac{\partial \mathcal{L}}{\partial b}] \tag{33}$$

The learning rate has to be chosen large enough that a minimum of $\mathcal{L}$ can be reached within the training, but if α is too large, $\mathcal{L}$ might leap over the minimum. There are various methods to improve the parameter update method (Erdmann et al., 2021).

During training, the training data is processed multiple times, where each time the whole training data is used is called an epoch. But not the whole trainset is used for adapting the network parameters. Instead, a subset is sampled from the training data during each epoch. This subset is called "minibatch" (Erdmann et al., 2021).

The parameters are initialized before training from a gaussian distribution to ensure symmetry breaking. This is important to avoid that the nodes all learn the same features (Erdmann et al., 2021).

A major difficulty when working with deep learning is an appropriate choice of the model. It must be neither too complex (e.g. too many nodes and layers) nor too simple. If it is too simple, the data might be too complex for the model to describe (see figure 12). On the other hand, if it is too complex, that might lead to overfitting (Erdmann et al., 2021).

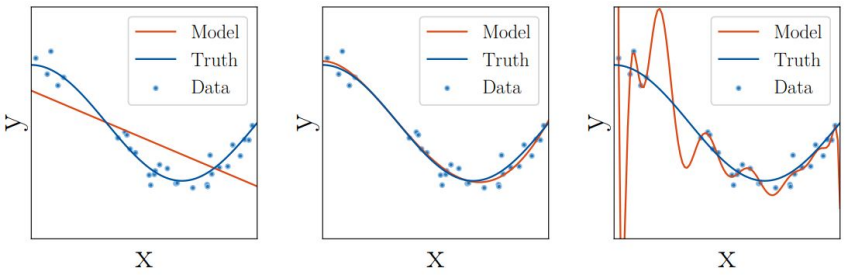


Figure 11: Left: A too simple model, Middle: an adequate model, Right: a Too complex model (Erdmann et al., 2021)

To avoid overfitting, it is useful to monitor the training. Therefore, the total available data is split into three sets: the training data set, the validation set and the test set. The trianing set is used to update the parameters as described. The validation set is used within an epoch to see if a minimum has been reached. Only after all epochs, the test set is used to review the model's performance. The difference between training loss and validation loss from one epoch is the generalization error. If the validation loss increases, this is a sign of overfitting. There are regularization methods that reduce or stop the training process early when a minimum is already reached (Erdmann et al., 2021).

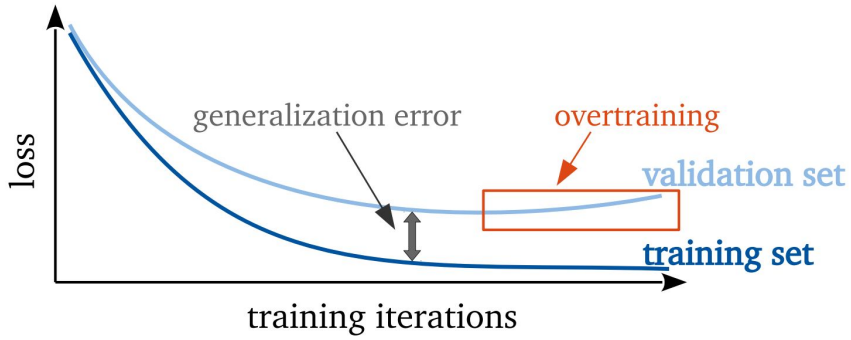


Figure 12: The training loss and validation loss during training (Erdmann et al., 2021)

2.3.1 Convolutional Neural Networks

An important model architecture is the convolutional neural network (CNN). They are superior to fully connected neural networks when dealing with data with inherent symmetries, such as images, because fewer parameters are needed and thus overfitting can be prevented. One referres to the spatial domain when speaking about the data itself, i.e. the pixel values. The feature space, on the other hand, is a term for the meaning of patterns and structures of the data. An example would be that pixels that are all gray and arranged in a circle are representing a tire. The interpretation of the pixel set is its "semantic meaning" (Erdmann et al., 2021).

The idea behind a CNN is going from a local view of neighbored data points to an overall view. With layers of "filters" that pass through the data, information of a neighborhood is collected and passed to another layer. Those filters have the weights and biases, thus they are equal for every segment of the data. This concept is referred to as "weight sharing" and makes the convolutional operation "translational invariant". Due to weight sharing, the total number of network parameters does not depend on the data dimensions, in contrast to a fully connected neural network. The computation performed by the filters are similar to that of the fully connected nodes, but with the neighborhood $\mathcal{N}$ (Erdmann et al., 2021) :

$$\begin{aligned} x_i &= \sum_{x_j \in \mathcal{N}_|} W_j \cdot x_j + b \\ y_i &= \sigma(x_i) \end{aligned} \tag{34}$$

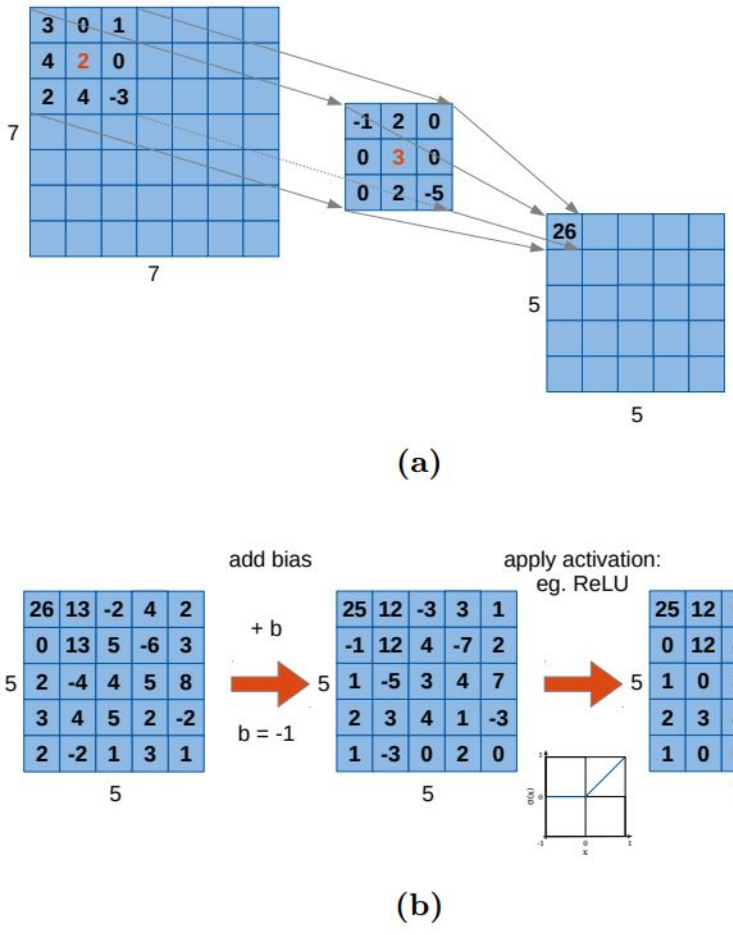


Figure 13: a) first computation of equation 34 b) second computation of 34 (Erdmann et al., 2021)

The size of the neighborhood that is covered by each filter of a layer at one time is called its "receptive field of view". The increasing receptive view of following layers allows "hierarchy learning": A layer processes the low-complex local information from preceding layers and identifies more global structures. Thus, a complex problem is divided among many layers, each of them performing not so complex tasks (Erdmann et al., 2021).

It is also possible to (re-)increase the dimensionality of data via CNN. The "deconvolution"-operation can thus be seen as the inverse operation to the convolution described above. An example of a transposed convolution can be seen in figure 14 (Erdmann et al., 2021)

2.3.2 Autoencoders

Autoencoders are used to denoise data or reduce dimensionality. In the latter case, it is especially useful when dealing with high-dimensional data, as other methods such as t-SNE require relatively low-dimensional data. The output of the autoencoder is less detailed as the input, meaning that the compression of the data results in a loss of information (Chollet, 2016).

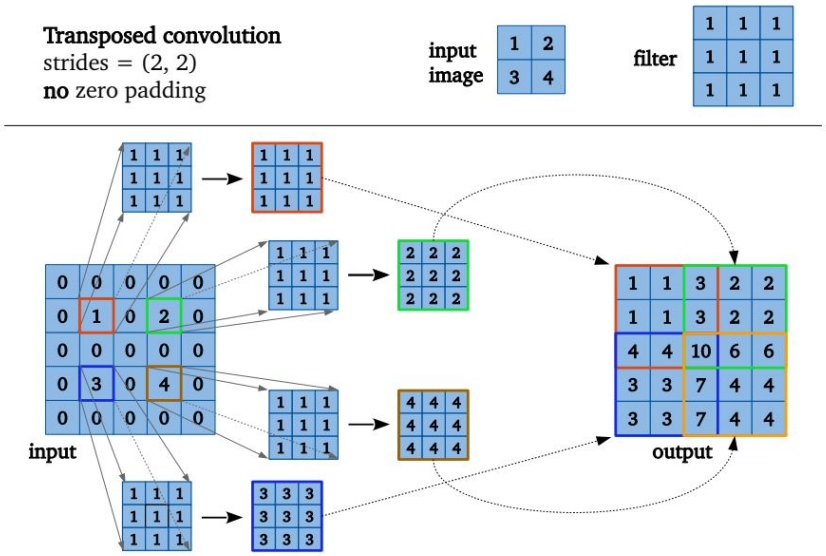


Figure 14: Each input pixel is multiplied with the filter, resulting in a filter-sized image. Then, the new images are added together to the output. (Erdmann et al., 2021)

Autoencoders mainly consist of an encoder and a decoder. It reconstructs an input **x** at the output **x'**. To ensure that the autoencoder not only replicates **x**, it is conducted through a lower-dimensional layer, the latent space. Since the training data is not labeled, autoencoders learn in an unsupervised manner (Erdmann et al., 2021).

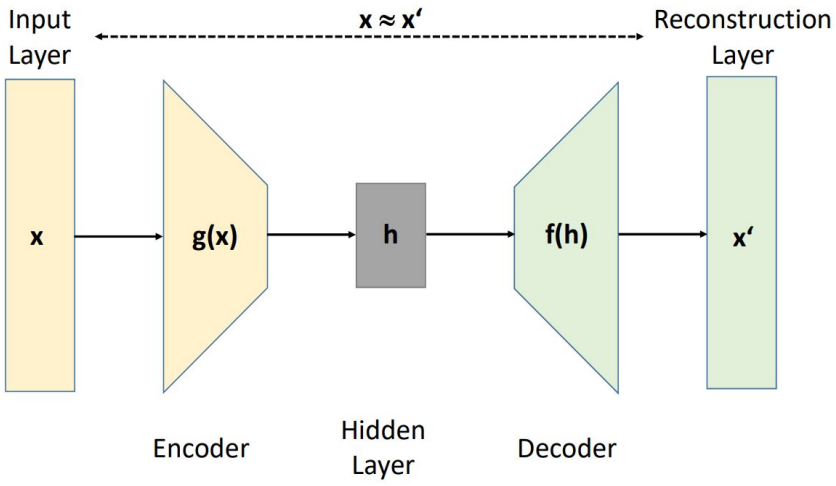


Figure 15: Structure of an autoencoder (Erdmann et al., 2021)

3 Development of the autoencoder architecture

In this chapter, we describe the process of developing an autoencoder architecture to denoise spectra. In the beginning, we investigate the existing data. Then we formulate problems of increasing complexity and find a sufficient model architecture that can solve the problem. The models are adapted step-by-step until we eventually obtain a final architecture.

The data used for the project originates from a clinical study described in more detail in Huber et al. (2021). In the study, cohorts of lung cancer (LuCa), prostate cancer (PrCa), bladder cancer (BlCa), breast cancer (BrCa) patients and healthy individuals for reference (NSR), are included (Huber et al., 2021).

To ensure numerical stability of the neural network, the input data should be preprocessed properly. There are some standard methods that are used frequently with machine learning. First, it might be useful to normalize the data before passing it to the neural network. Another beneficial preprocessing method is standard scaling, where the data is transformed with its mean $\langle x_i \rangle$ and its standard deviation σ_i as follows (Erdmann et al., 2021):

$$x_i\prime = \frac{x_i - \langle x_i \rangle}{\sigma_i} \tag{35}$$

For this work we have set some standard procedures. First, for all machine learning algorithms the data is standard scaled before forwarded to the model. For training, we apply ADAM as Optimizer for gradient descent (see Erdmann et al. (2021)). We furthermore always use the Callbacks Earlystopping, which stops the training when an optimum for the loss function is found, and ReduceLROnPlateau, which recognizes stagnating learning and declines the change of parameters in that case (Keras Team, n.d.). Besides, we choose to train the models for 50 epochs by default with a batchsize of 128 and use ten percent of the training set for validation.

3.1 Exploring the data

The project started by investigating how the spectra of quality Controls (QC's) look like. These are mixtures of human serum from multiple samples (Huber et al., 2021). To visualize the data we apply PCA and t-SNE. In figure 16 we plotted the measurements of all QC's along with the mean values.

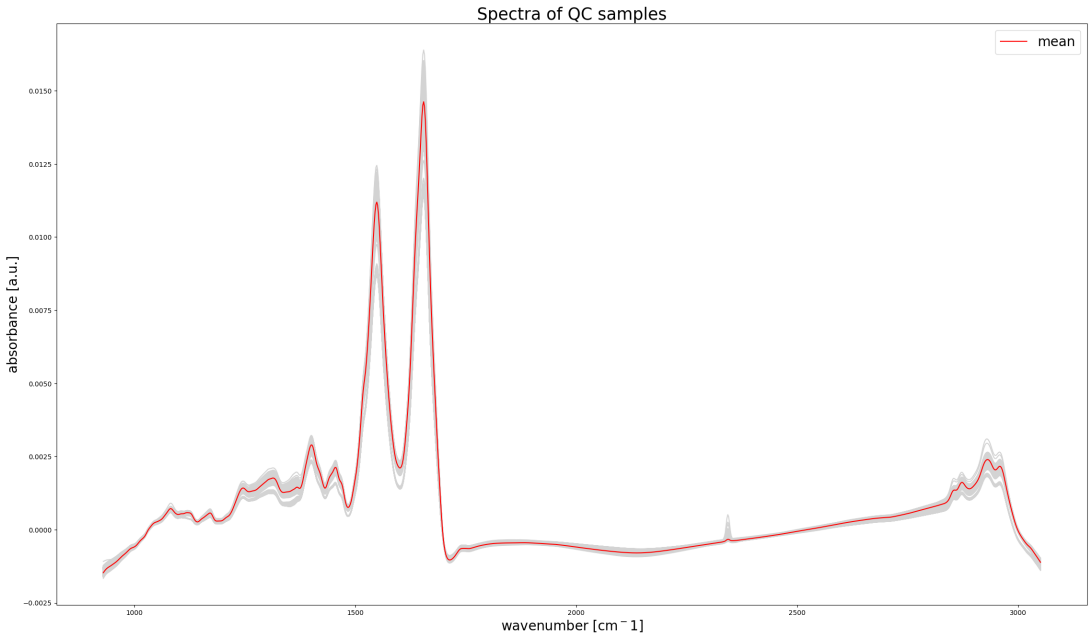


Figure 16: Absorption spectra of quality controls

Secondly, we did a PCA of the QC's.

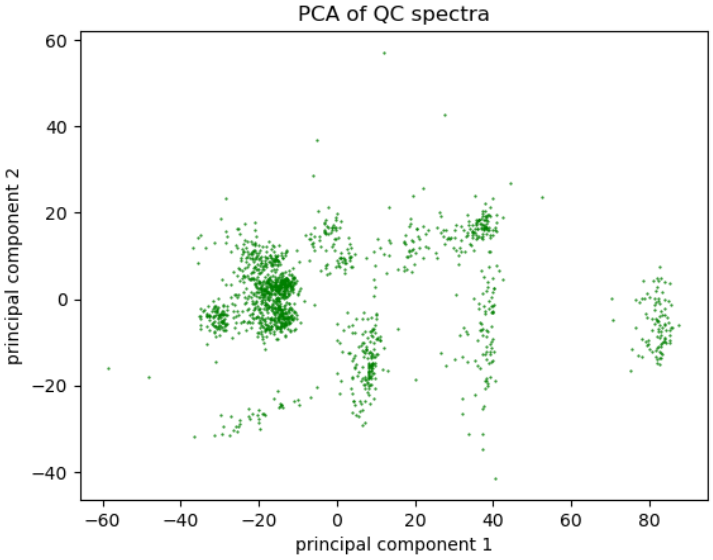


Figure 17: PCA of the QC spectra

In the t-SNE plot in figure 18 one clearly sees that there are different groups of patterns (drifts) in the spectra, which are collocated in clusters in the plot.

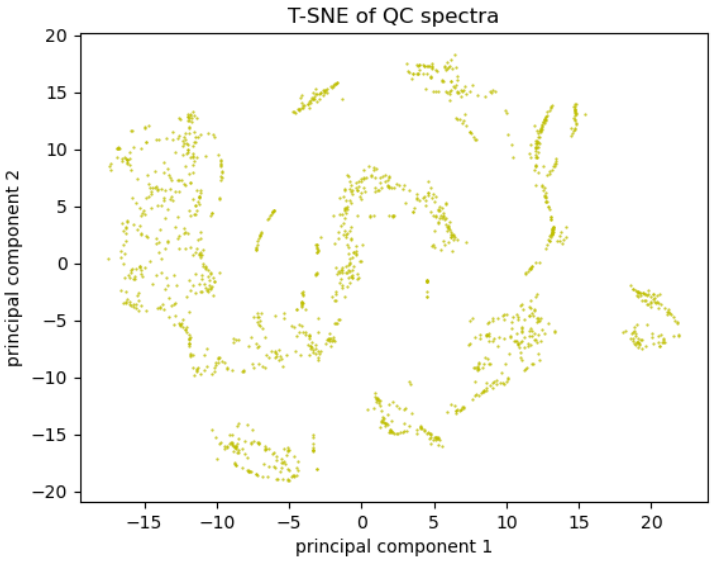


Figure 18: t-SNE of the QC spectra

3.2 Simulating spectra

To be able to train deep learning models, the measurement data is not sufficient. Therefore, in the process of developing an autoencoder we frequently use a spectral generator ()Eissa2022. It is a function written in python that produces an arbitrary amount of linear combinations from input spectra. Also, it provides the option to differentiate between cancer cases and control cases and produce spectra of those groups seperately. Another feature of the spectral generator is the possibility to generate spectra with artificial white noise. This noise is generated from a random normal distribution with mean 0 and standard deviation eps_{std}, which is the parameter that can be adjusted. We will make use of the last property in section 4. To analyse the spectral simulator, we produce some spectra from the QC's. A few of them can be seen in figure 19.

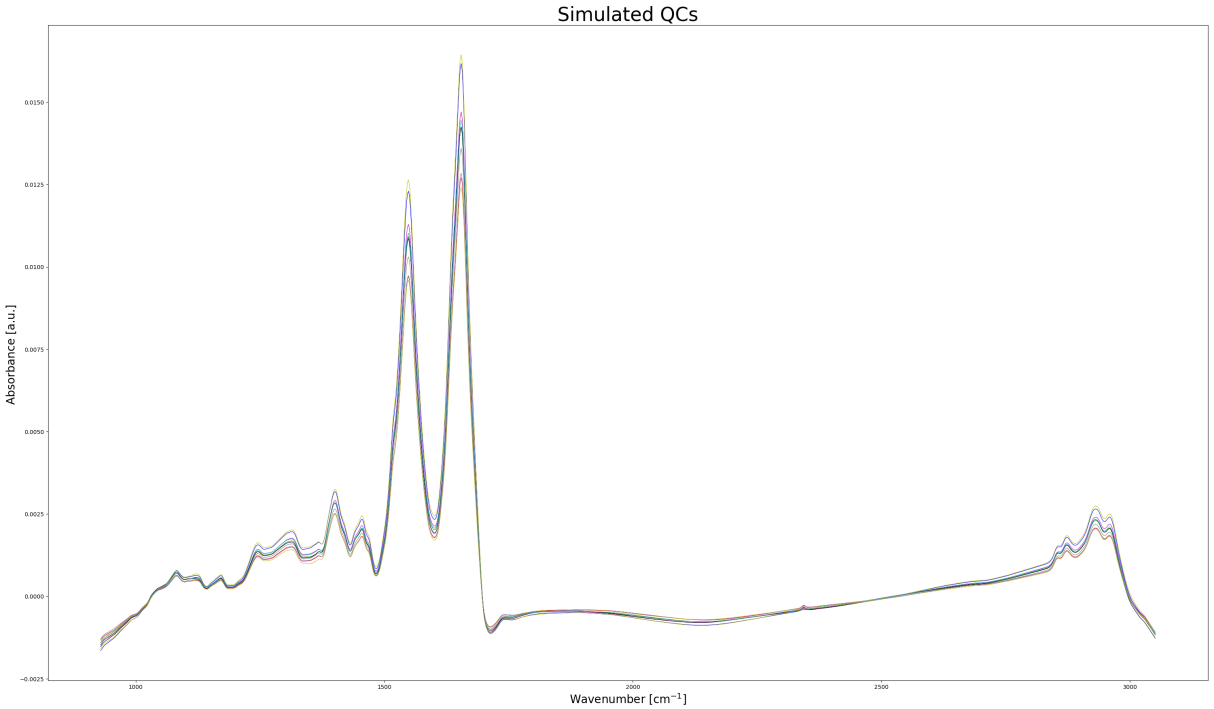


Figure 19: Simulated QC spectra

For comparison of real and generated QC's, we perform a PCA on the simulated spectra. The results can be seen in figure 20. The variance in the simulated data is much higher than in the QC's. Also, there are no visible clusters. This is because the simulated spectra are linear combinations of the QC's and thus spectra from all clusters potentially contribute to a simulated spectrum. Similar results as mentioned for the PCA can be observed in the comparison of the t-SNE plots (see figure 21).

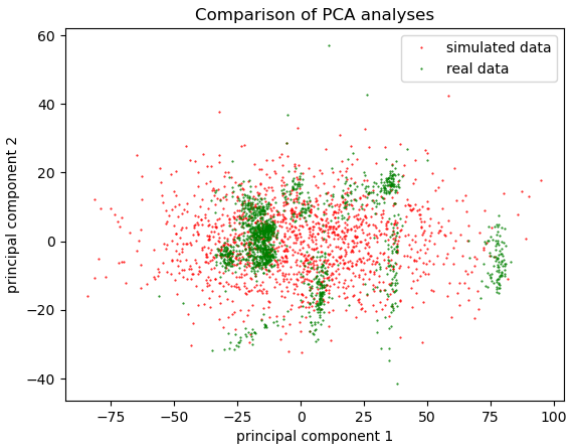


Figure 20: PCA of real and simulated QC's compared

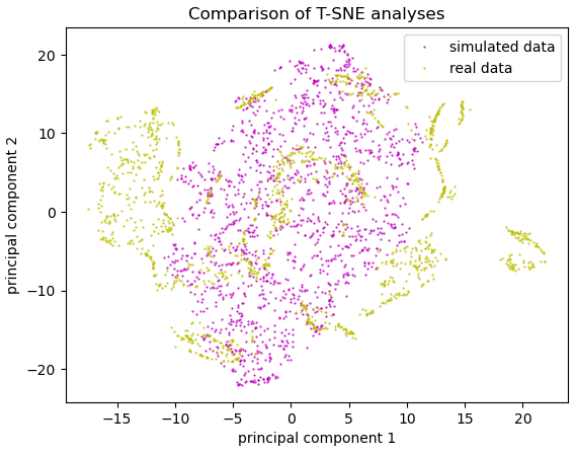


Figure 21: t-SNE of real and simulated QC's compared

For further investigation of the spectral generator, we implement a classifier based on a SVM to classify between the real and simulated data. Then we improve the perfor-

mance of the model step-by-step by doing GridSearch, a method where all combinations of hyperparameters of a given grid are tested (Erdmann et al., 2021), and optimizing the parameters around the results. The code around the classifier has also not been written as part of the thesis, but has been provided by the group. The parameter C for the SVM is determined via gridSearch. For training the classifier, Stratified K fold cross validation is used. This method splits the training set into k partitions which contain the same ratio of real and simulated spectra. One of the partitions is then used as a test set, while the rest is used for training. This is repeated k times, so that each of the k folds is used as a test set once (Zeng and Martinez, 2000). To determine C we use $K = 5$ splits and for the predictions we use $k = 5$, thus we perform a nested cross validation. This classifier structure is used for every other classification tasks.

The standard method for evaluating classifications in this thesis is the Receiver Operating Characteristic (ROC). It is based on the conflict between sensitivity (Correct identification of positive cases) and specificity (Correct negative result for truly negative cases) of a classifier. One usually has to set a threshold, which determines how important each of the conflicting concepts are considered. The ROC is a curve in a diagram with the falsely positive classified results on the abscissa and the truly positive predictions on the ordinate. For each threshold, the corresponding values are calculated and depicted in the diagram. Thereby the ROC curve is obtained. A measurement for the quality of the classifier is the AUC, the area that the ROC encloses with the abscissa and the right border of the diagram. The higher it is (with a maximum of 1), the better a classifier is rated. (Backhaus et al., 2021). For the classification between real and simulated Spectra, with real spectra being considered as positive classifications, we obtain the following ROC (figure 22):

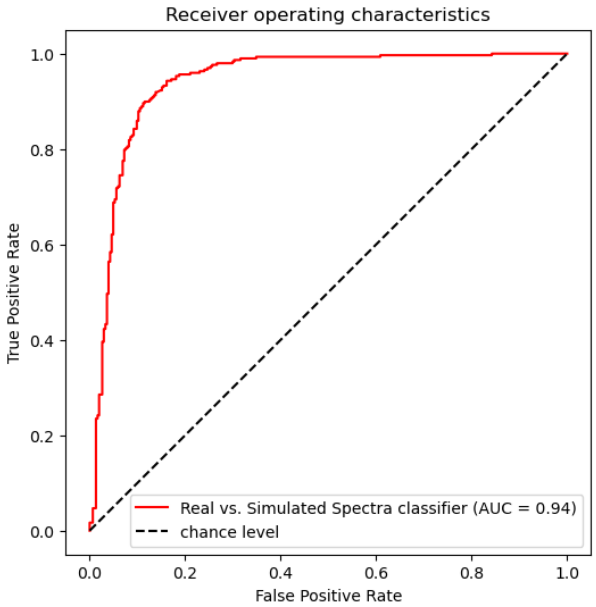


Figure 22: ROC for the spectra classifier

The AUC of the ROC is 0.94 ± 0.04.

3.3 Approximating PCA with an autoencoder

We want to compare the encoder of an autoencoder with Principal Component Analysis. According to Goodfellow et al. (2016), an autoencoder with linear activations and MSE can be used for PCA. If nonlinear activations are used, an autoencoder can perform a nonlinear reduction of dimensionality superior to PCA (Goodfellow et al., 2016).
Precisely, we build a full autoencoder which produces a copy of the original input (generated QC's). We can then use the encoder to produce 2D-representations of the data. Only for this experiment we take a model architecture with a number of latent space nodes to 2 and linear activation.

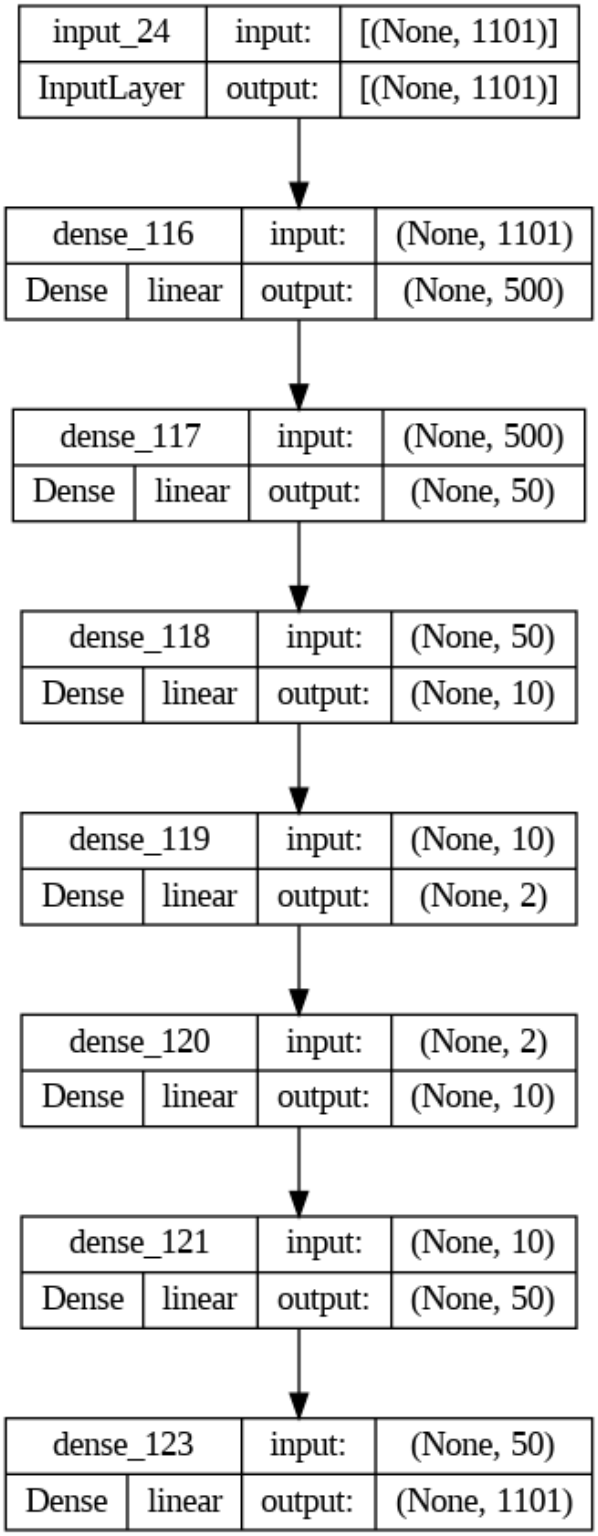


Figure 23: Autoencoder with linear activations serving for PCA

The result can be viewed in figure 24.

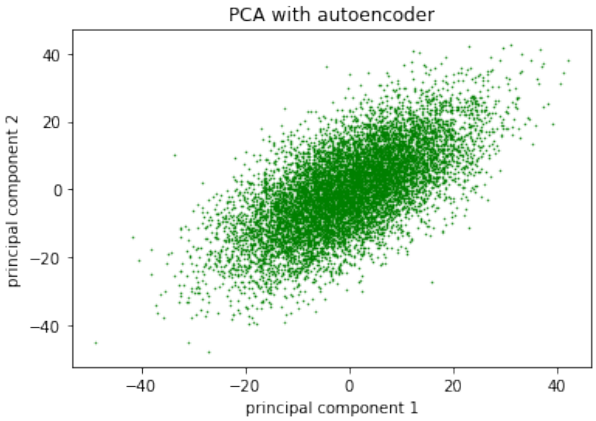


Figure 24: Latent space representation of the QC

Again we see a cloud because in the generated QC's all properties are mixed. Since the PCA from the autoencoder matches the PCA in figure 20, we consider the linear autoencoder to be able to approximate PCA.

3.4 Parameter optimization

To start with our actual aim, to develop an autoencoder which denoises the spectra, we increase the number of layers from the previous model and determine suitable hyperparameters. We are analyzing the performance of the model for different hyperparameters. Due to lack of computing power, an extensive gridSearch was not possible. Therefore we investigated if the performance of the model changes if we modify one parameter at a time. Our criterium for improved performance is the validation loss after the last epoch. We again reproduce generated spectra from the QC's.

1. First, we variate the Batch size:

Batch size	Validation loss
32	0.4371951675415039
128	0.43719440937042237
256	0.43719438552856443

   Since the validation losses for the batch sizes are all the same for the first (meaningful) digits, we assume that this parameter does not affect the models performance significantly.

2. Now we look for the best selection of the activation function:

Activation	Validation loss
hyperbolic tangent	0.43726702928543093
elu	0.00010671236959751696
relu and linear output	0.00010903983755270019

We see hat elu performs best (similar to relu with linear output activation). Therefore we use elu from now on. Now we want to see if more or fewer neurons work better.

3. Now we want to see which number of latent space neurons works best for the autoencoder

Number of neurons	Validation loss
10	0.00010828426806256175
25	0.00010824832395883277
50	0.00010777310322737321
100	0.00010955033067148 18

It seems like the model with 50 neurons performs best. However, the values are really similar.

4. Lastly, we want to see if a dropout layer in the model can improve its performance. Dropout is a regularization technique that sets some random weights during training to zero, cutting off nodes. This forces the model to learn features in multiple ways (Erdmann et al., 2021).

Dropout rate	Validation loss
0	0.00010549005877692252
0.1	0.00012050992052536457
0.3	0.00012944667367264628
0.8	0.00011758754961192608

Apparently dropout doesn't have much influence, because the order of magnitude of the validation losses is the same for every model.

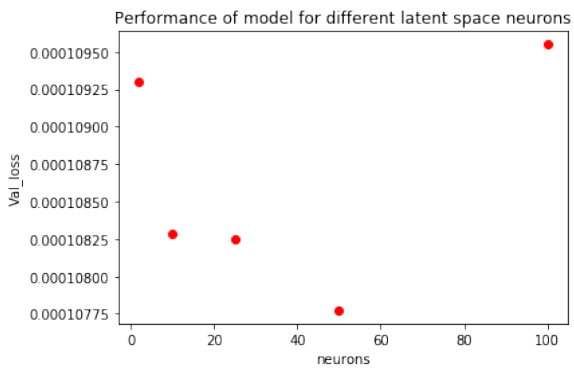


Figure 25: Validation loss for different latent space neurons

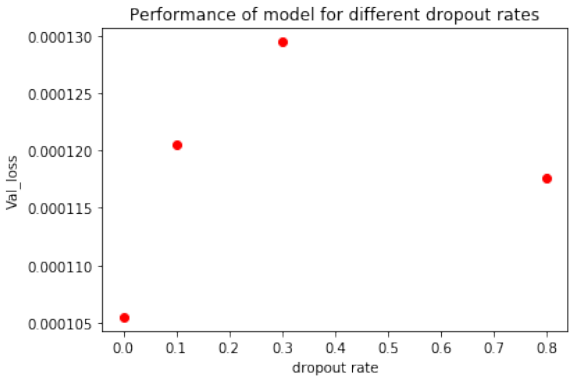


Figure 26: Validation loss for different dropout rates

We see that only the activation function made a significant difference. It seems that the other regulations become important only for more complex deep learning applications. The autoencoder with which we continue looks like this:

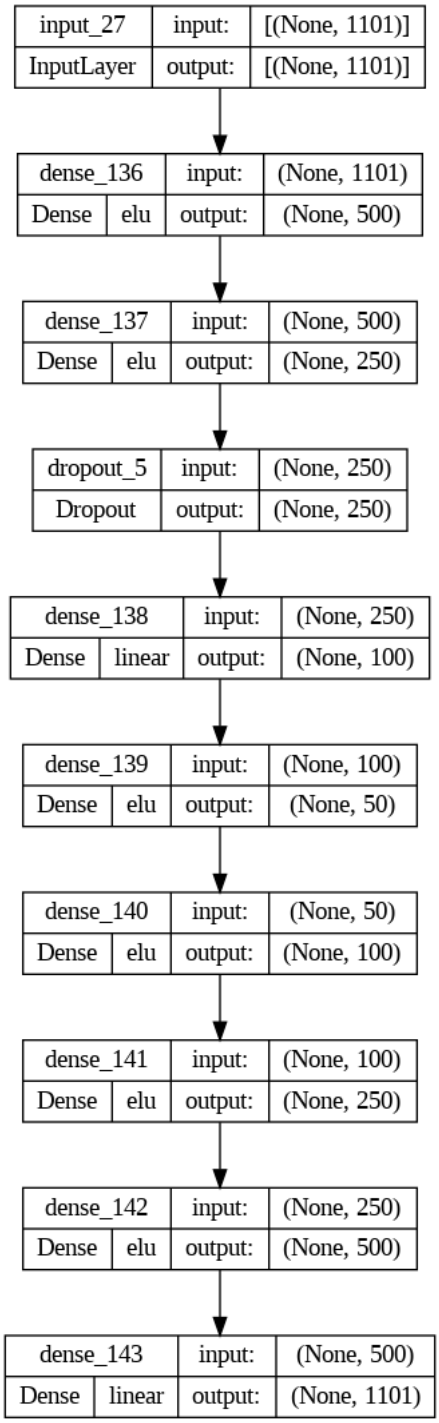


Figure 27: Structure of the optimized fully connected neural network

3.5 Reproducing Spectra with an autoencoder

Now we want to first see if our autoencoder, based on the previous optimization of hyperparameters, can reproduce spectra accurately. The data used for this experiment is a set of 10000 generated spectra with the non-QC data as original data input. Two examples of original and denoised spectra can be seen in figure 28 .

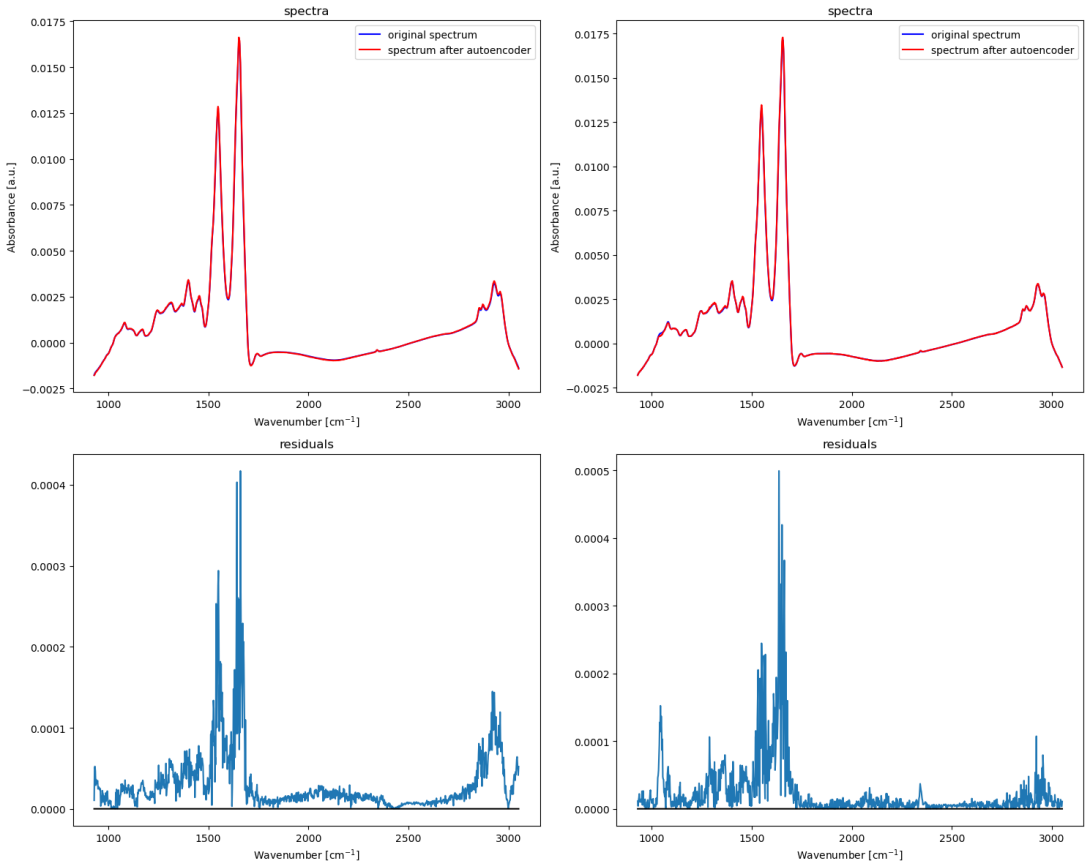


Figure 28: Examples of reproduced spectra

We want to see whether the model is able to reproduce the properties of the spectra in a way that they are distinguishable from other spectra. We perform a PCA on the real data (excluding QC's) and on the reproduced spectra of the real data. Then we plot 10 real spectra examples and their reproduced versions in the PCA-plot and see if we can relate the spectra pairs to each other. If the important informations are conserved in the reproduced spectra, we expect them to be closer to each other than to other pairs in the plot.

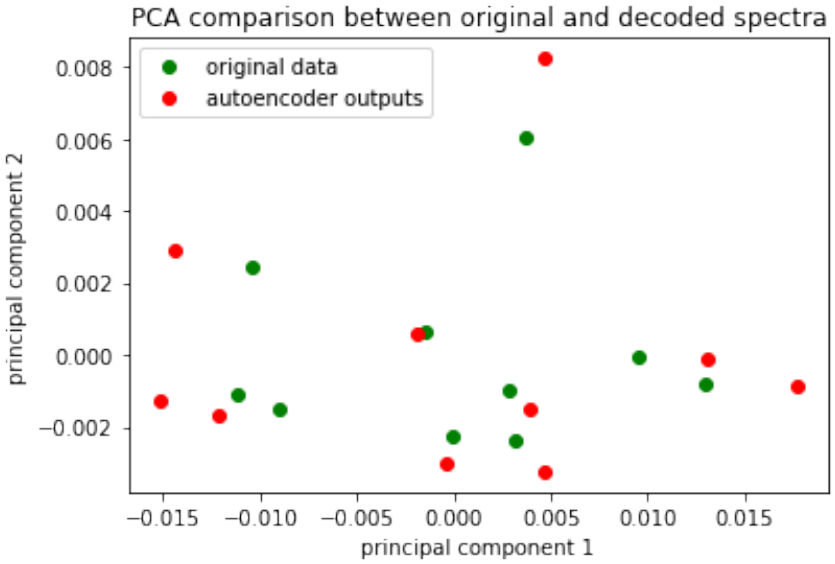
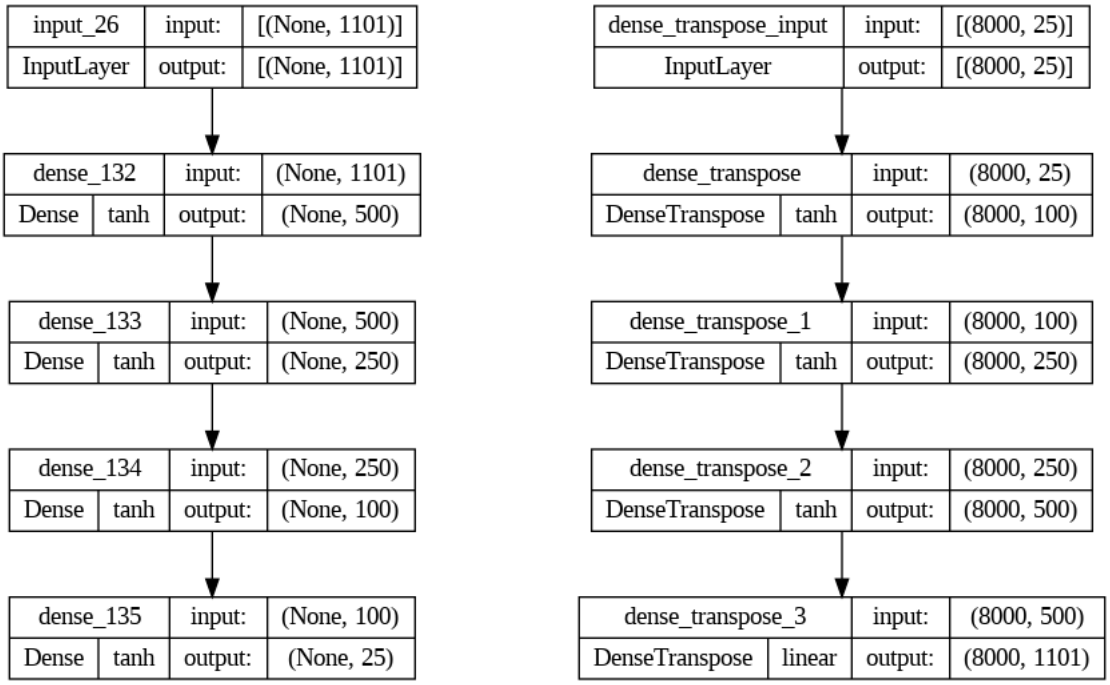


Figure 29: Comparison of original and reproduced spectra

As it can be seen in figure 29, the information stored in the spectra can be reproduced and pairs of original and decoded spectra can be assigned. Just reproducing spectra has shown to be a quite easy problem for autoencoders, because no regulation was needed.

3.6 Autoencoder with tied weights

For symmetric autoencoders it is possible to tie the weights from opposite layers. This reduces the number of hyperparameters and thus can prevent overfitting. Besides, the training process is also fastened (Géron, 2020, Chapter 17). Therefore we tried to implement a layer class, based on the class DenseTranspose proposed in Géron (2020), and compare the performance of a Tied-Weights autoencoder with the other version. Since this code is not working for the latest Keras versions, we have updated the class and tried to implement it. By trial and error we found out that the structure of the architecture illustrated in figure 30 leads to the best results. It only has 705,826 trainable parameters, compared to 1,413,851 parameters of the normal autoencoder. By reason of overfitting risk, a reduced amount of hyperparameters is preferable.



(a) Encoder of autoencoder with tied weights (b) Decoder of autoencoder with tied weights

Figure 30: Structure of the autoencoder with tied weights

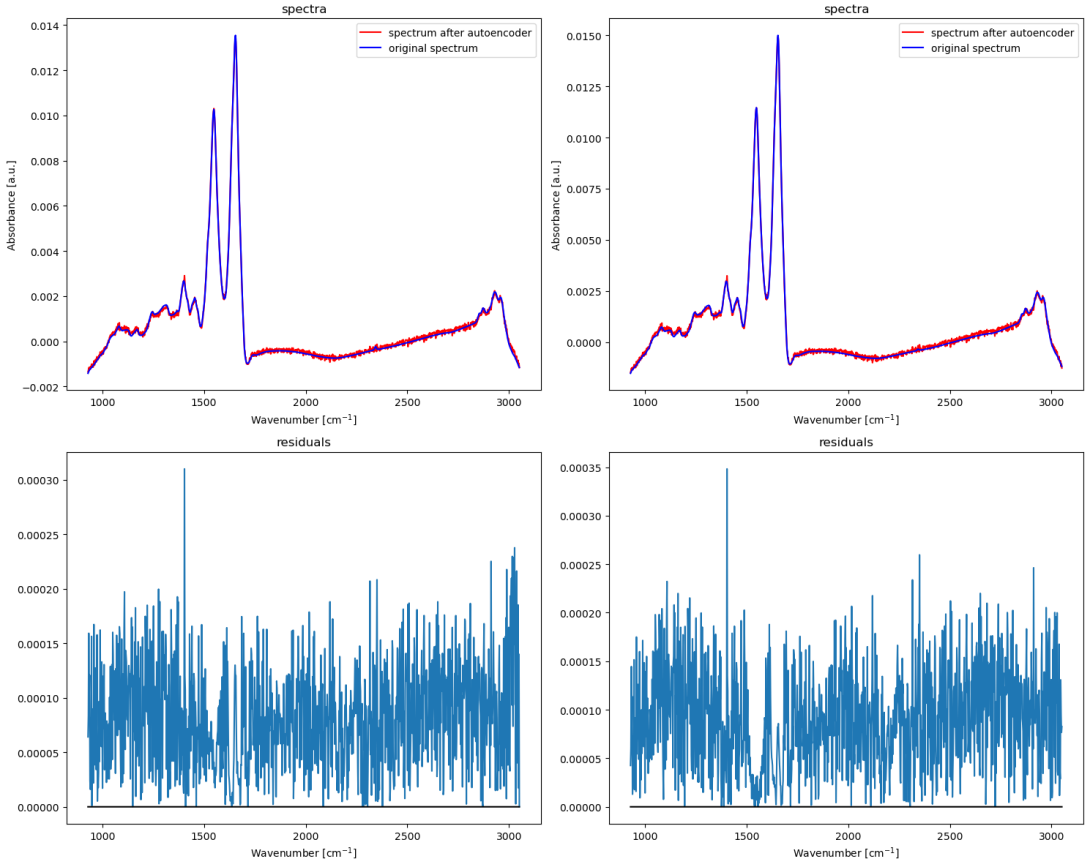


Figure 31: Examples of reproduced spectra for tied weights

Apparently the autoencoder does not denoise the spectra, but injects noise. Some unknown issues in training make the autoencoder not working properly. In general, tying weights seems to be a reasonable method for symmetric autoencoders. However, the performance of this autoencoder was very unstable and due to lack of time, there was no possibility to investigate the problem further.

3.7 Developing a convolutional neural network model

We are now beginning to use a CNN in addition to a fully connected neural network and compare them through the next tasks. Since not all regions of the spectra are correlated, it is reasonable to reduce the amount of parameters by using a CNN instead. We assume that a Convolutional neural network can reproduce the data better.
Our initial CNN architecture is based on the model from Vijay (2020), while some features are transfered from the parameter optimization. The CNN now only has 29,729 parameters. Its structure can be seen in figure 32.

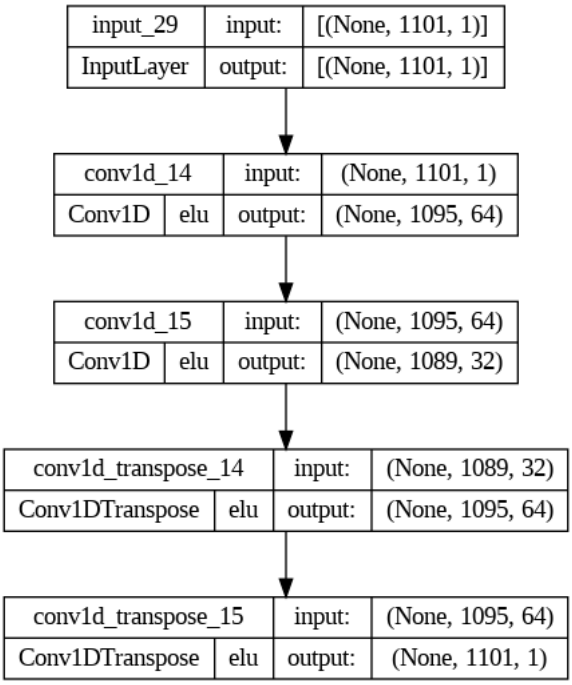


Figure 32: Structure of the initial CNN model

First, we made the CNN reproduce some generated QC's. Two examples can be seen in figure 33.

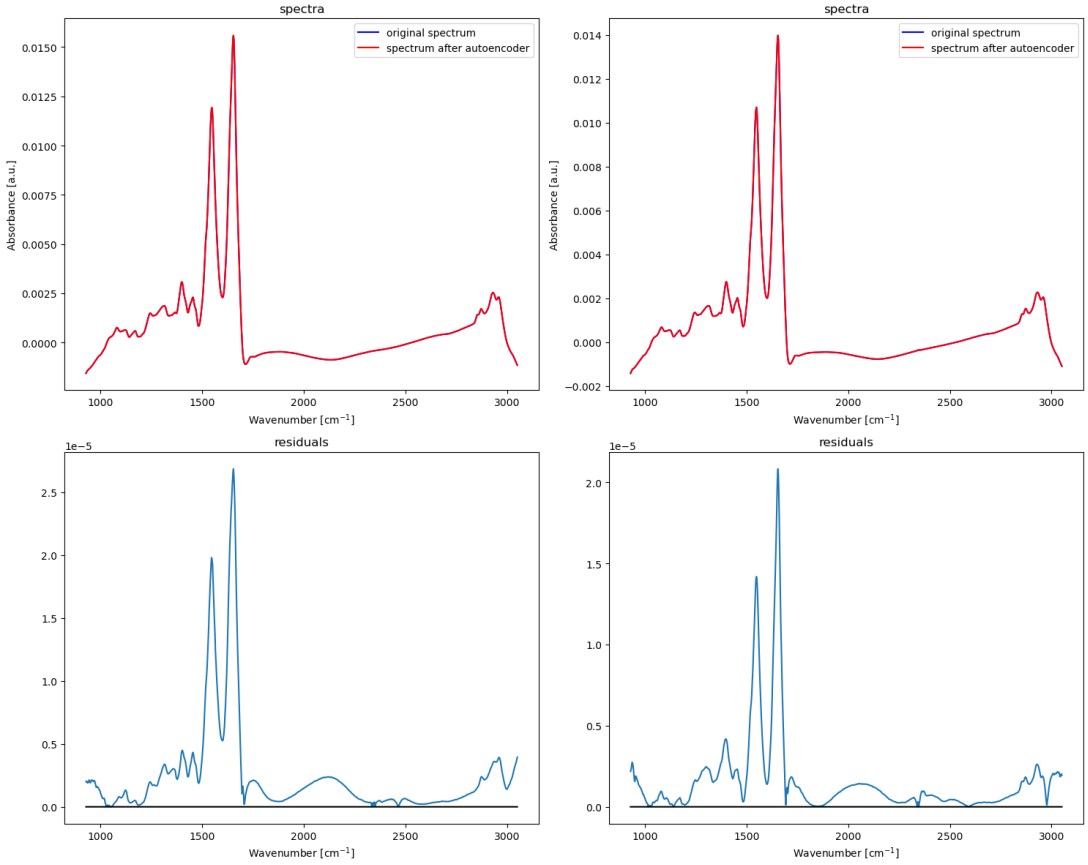


Figure 33: Examples of reproduced spectra for CNN

For the next task we changed our CNN architecture to a more complex one, because the previous version was not sufficient (a figure such as 36 for the simple CNN is attached in the appendix). The enhancement of the CNN architecture has been made via trial and error, until the results in figure 37 were at its best. The updated model is structured as in figure 34. Furthermore, striding is implemented in the model. This is a method useful for large input data and means that the filters are applied only on every nth value of the spectrum (Erdmann et al., 2021). It has a total of 1,153,745 parameters.

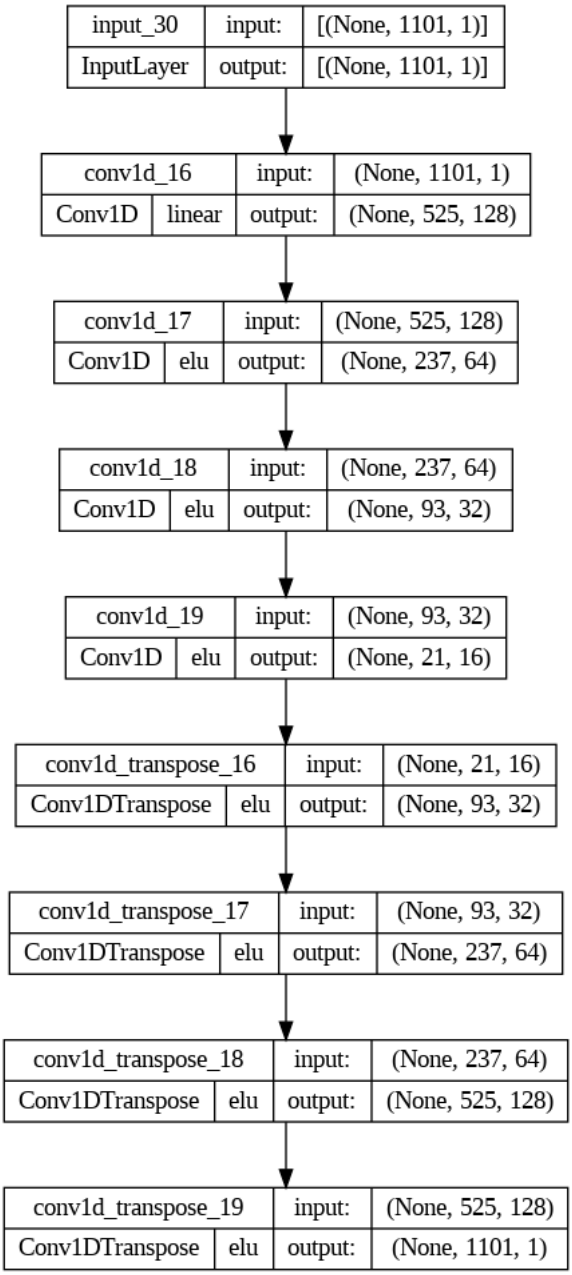


Figure 34: Structure of the optimized CNN

Again, a more systematic approach to evaluate the ability of reproducing the information of the spectra is to compare the original and reproduced spectra in a PCA. Ten example pairs can be seen in figure 35. Comparing to the fully connected neural network (see figure 29), the performance is similar.

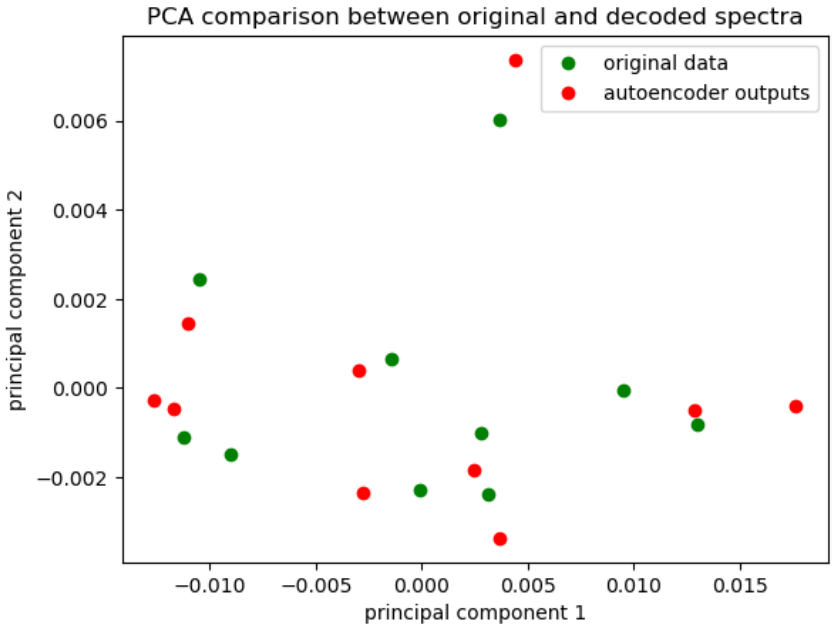


Figure 35: Comparison of original and reproduced spectra for a CNN

3.8 White noise filtering

We are examining whether the autoencoders can be trained to filter white noise. To do this, we have generated pairs of spectra with various strengths of artificial white noise and without noise from the QC's. The noisy data serves as input to the autoencoder, while the not noisy data is used as true values in the loss function. We have performed this experiment on both a fully connected neural network, as well as on CNN's.
In figure 36 are some examples for the denoised spectra from the fully-connected model.

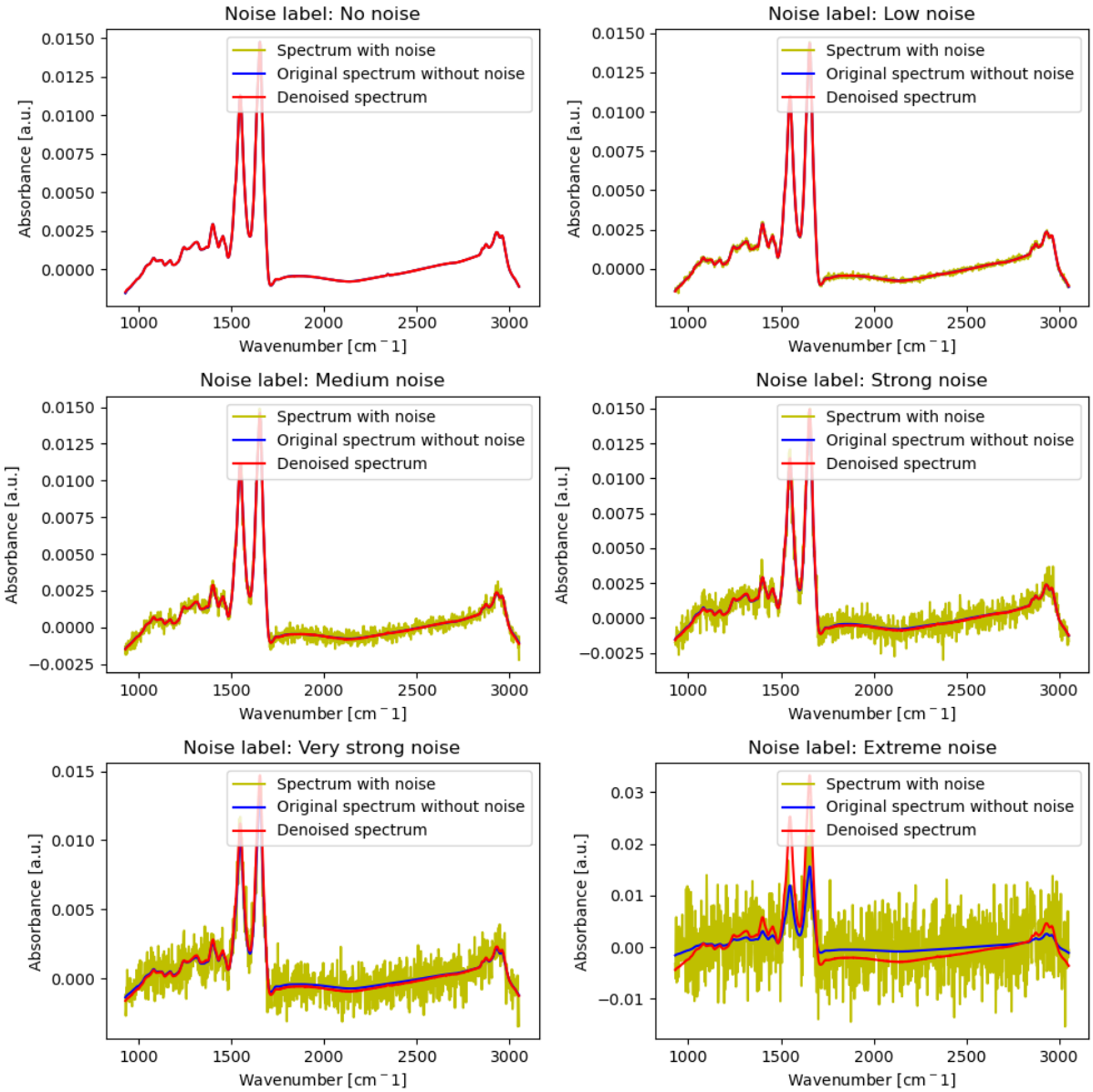


Figure 36: Removing artificial white noise with the fully connected neural network

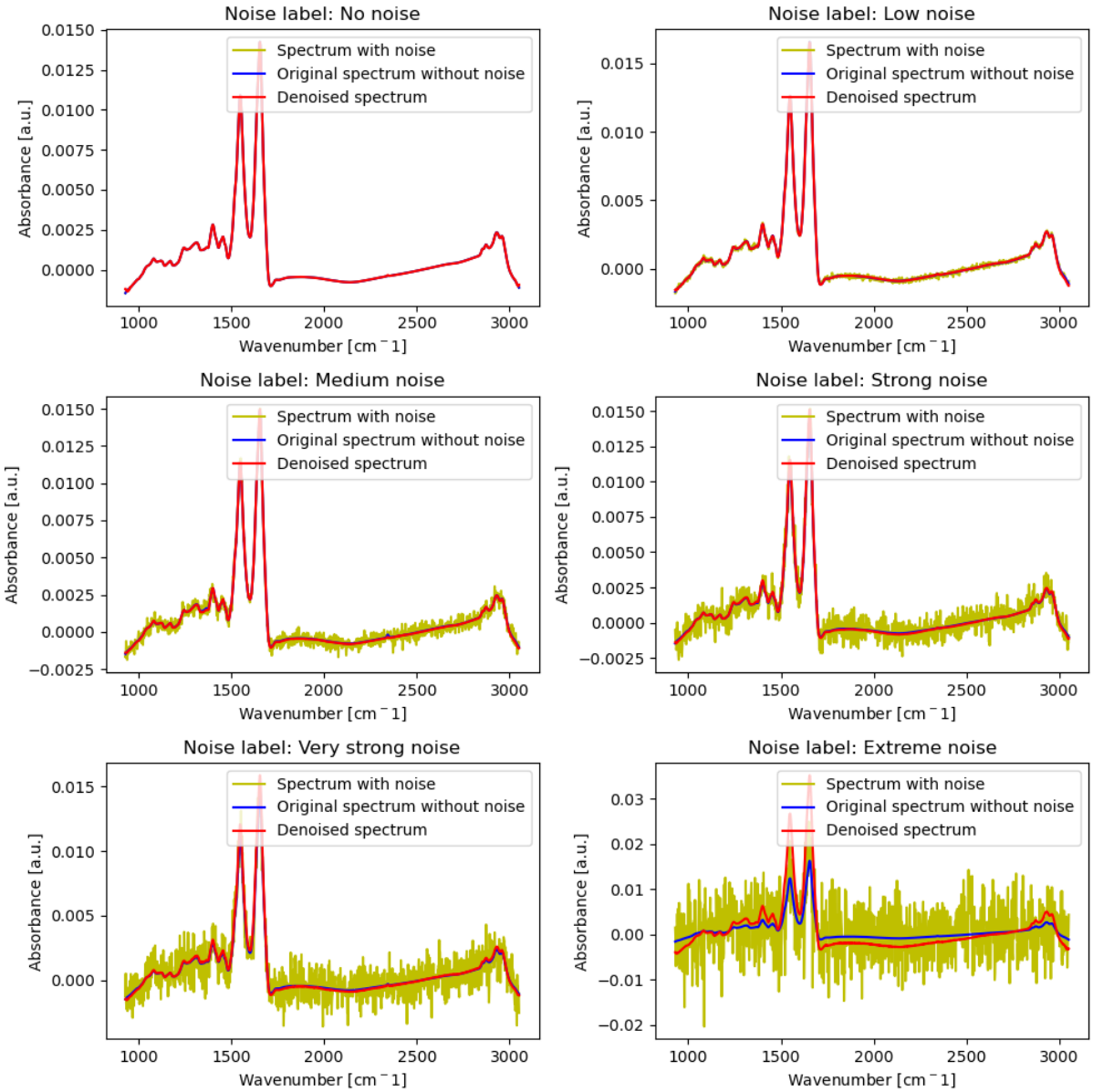


Figure 37: Removing artificial white noise with the CNN

We see that to some extend the denoising works fine, but when the strength of the noise gets too high, both models fail to reproduce the spectra properly. To obtain a more quantitative relation between noise strength and reproduction performance we are calculating the summed mean squared errors from all reproduced spectra of the epoch compared to their original spectrum. We then perform a polynomial regression on the achieved points. For the fully-connected neural network the regression yields a MSE:

$$MSE_{\text{total}}(x) = 39.6x^3 + 2.6 \cdot 10^{-1}x^2 - 2.5 \cdot 10^{-4}x + 2.5 \cdot 10^{-8}$$

For the CNN we obtain:

$$MSE_{\text{total}}(x) = 37.0x^3 + 2.9 \cdot 10^{-1}x^2 - 2.8 \cdot 10^{-4}x + 2.8 \cdot 10^{-8}$$

x is the noise parameter eps_{std} of the spectral generator. Since both regressions are similar, we conclude that the performance to denoise artificial white noise increases equally with increasing noise for both fully connected neural network and CNN. For smaller noise levels, the filtering of noise works very good and the reproduced spectra look nearly like the original ones without noise. With increasing noise levels the decline of performance, measured via the total MSE, can be estimated with a polynomial function of degree 3 for values up to $eps_{std} \approx 0.01$.

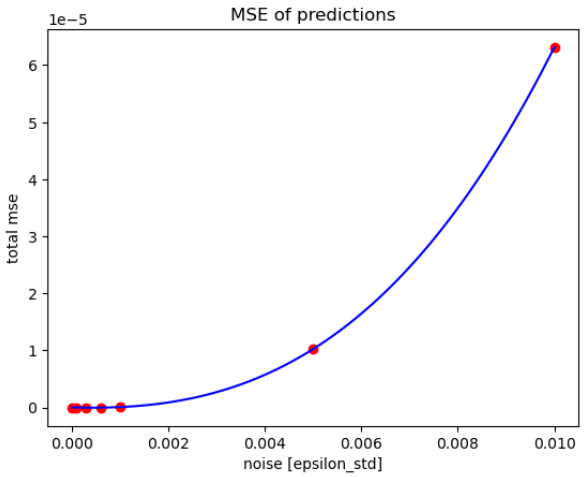


Figure 38: Summed MSE for different noise levels (fully connected)

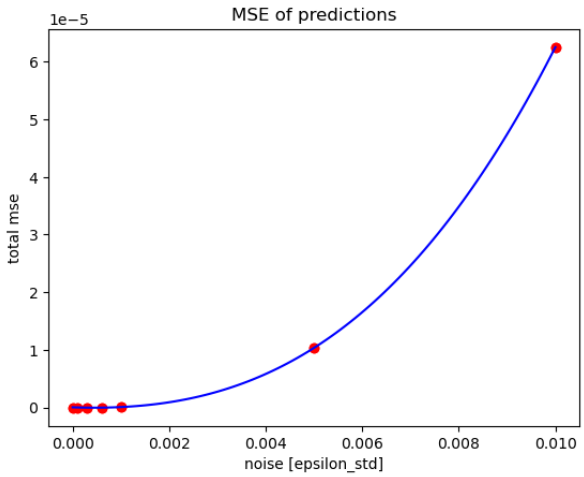


Figure 39: Summed MSE for different noise levels (CNN)

Comparing, the results of both tests for CNN and fully connected neural network are quite similar. Pairs of points that belong to the same spectrum can be assigned in most cases for both models. White Noise is removed similarly. Apparently both architectures are able to solve the so-far selected problems. With a similar performance, the CNN has fewer parameters. Therefore, we choose to continue our project with the CNN.

3.9 Comparison of preprocessing

After developing an adequate autoencoder, we want to compare its performance of preprocessing spectra compared to the standard preprocessing method in use.
The standard preprocessing method for the spectra is described in Leonardo (2021). The samples used for spectra measurement are liquid. The first step is therefore to subtract an infrared spectrum of pure water from the spectra. Since the relevant absorption bands are not located in the silent region (wavenumbers between 1800 cm^{-1} and 2750 cm^{-1}), this region is removed from the spectra. Lastly, the spectra are normalized (Leonardo, 2021).

We use the Data LuCa vs. NSR, presented in Huber et al. (2021), for classification and a set of generated spectra from this data to train the autoencoder architecture. All data is standard scaled. The classifier is a SVM with linear Kernel. Just like in the classification before, we do grid search to determine the optimal parameter C for the SVM. Then

we again do a cross-validation prediction for all data sets (not preprocessed, standard preprocessing method and preprocessed via the autoencoder). It is worth mentioning that the set used to train the autoencoder is not fully independent from the data set used for the classification, because it is generated from that. This might constitute a methodical error causing a bias. The results are summarised in the ROC in figure 40.

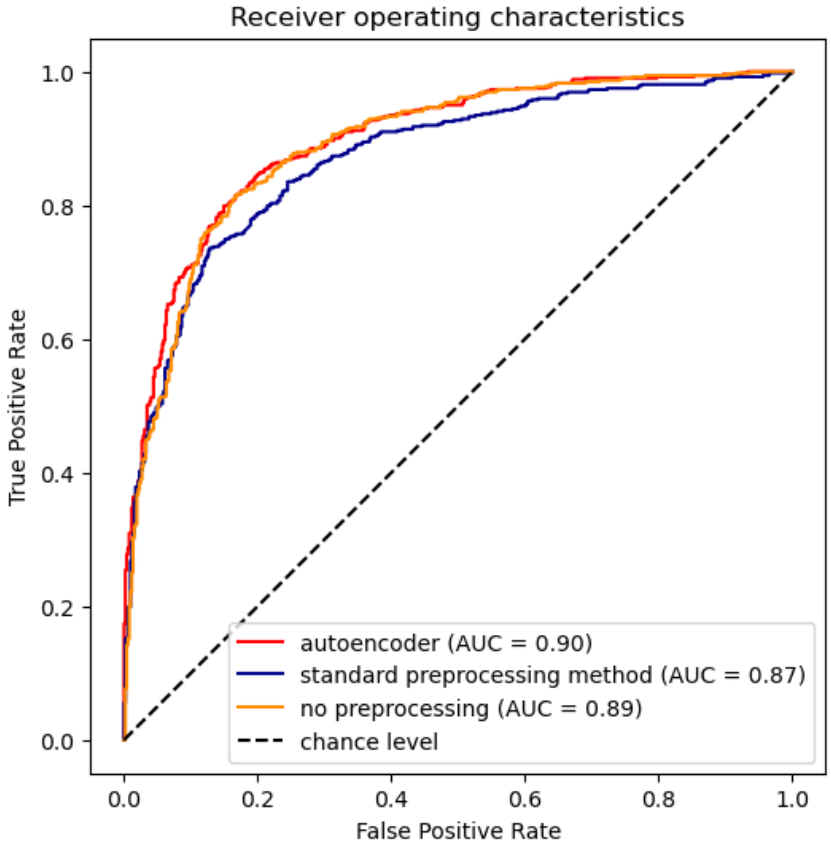


Figure 40: Classification performance comparison for different preprocessing methods

The AUC's are:

Preprocessing method	**AUC**
Autoencoder	0.90 ± 0.03
No preprocessing	0.89 ± 0.03
Standard preprocessing method	0.87 ± 0.03

We see a slightly higher AUC (0.90) for the autoencoder than for the standard preprocessing method and no preprocessing of the data. This is an indication that autoencoders are suitable tools for such tasks. From now on, we do terminal evaluations on our ultimate autoencoder.

Lastly, we want to take a closer look to the silent regions of the spectra. Figure 41 shows a PCA of them. We see that the standard preprocessed silent regions are more spread across the PCA, while the silent regions preprocessed by the autoencoder are more close

to each other than the original data. This is an indication that some denoising happens through the autoencoder.

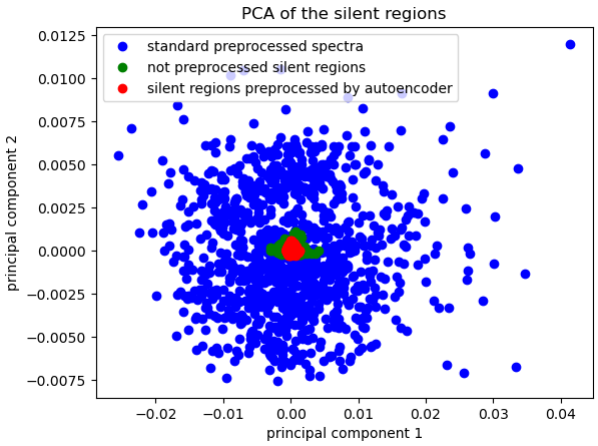


Figure 41: PCA of the silent regions

4 Results

In the last section we have developed our autoencoder for spectra denoising. Now we are completing the project by evaluating the model. First, we want to compare the denoising of white noise via autoencoder with the normalization. After that, we introduce a custom loss function to remove drift patterns from the data.

4.1 White noise removal using an autoencoder

In previous experiments we have denoised artificial white noise from QC spectra. So far, we have seen that our developed autoencoder architecture is able to preprocess and denoise the spectra from white noise with an adequate result while reproducing the biological information of the data. To finalize this investigation we now examine the noise reduction of non-artificial white noise from the QC's. Therefore we reproduce the data with an autoencoder of the final form, which is trained on generated spectra from the QC data. We want to answer the questions: Does the Autoencoder reduce the variance of the QC data in the PCA (means denoising of white noise)? Does normalization of the data yield the same results? Is the autoencoder superior to normalization of the spectra?

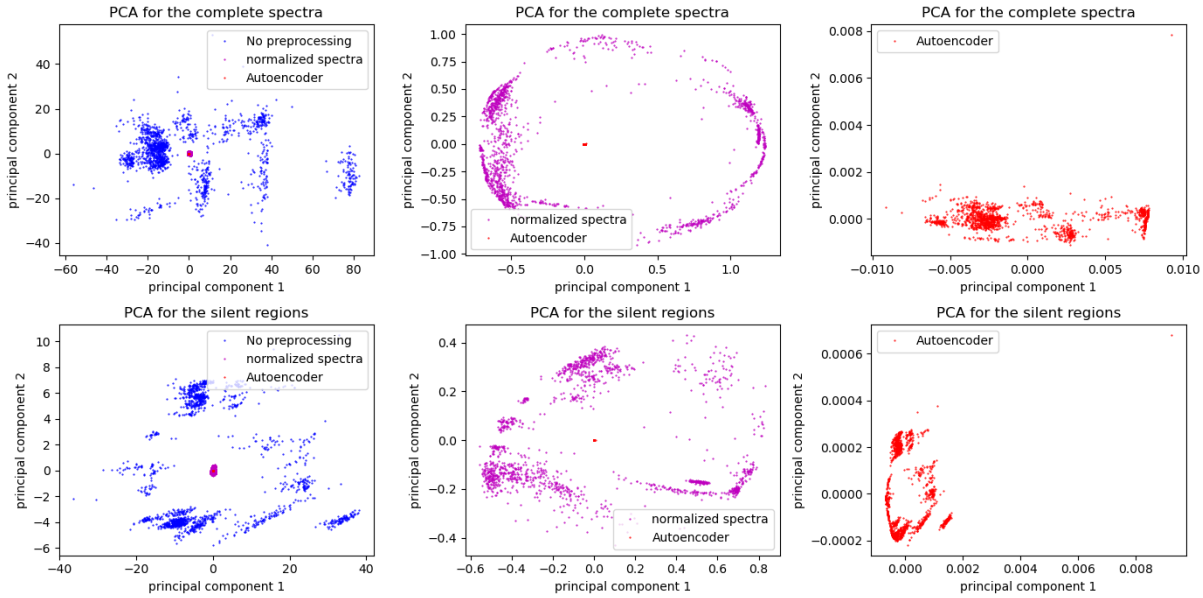


Figure 42: PCA of the QC data for different preprocessing methods

In the PCA of the three cases the variance of the normalized spectra is very low compared to the variance of the only standard scaled spectra. The variance of the autoencoder spectra is even smaller. Therefore we see a clear advantage of using the autoencoder over normalizing the spectra. This holds for both the complete spectra, as well as when only looking at the silent regions. Since the variance of the data in the PCA is much lower, the autoencoder is very suitable to denoise spectra from white noise.

4.2 Dealing with drifts using a denoising autoencoder

Now we want to use the model architecture and make it denoise the drift patterns in the spectra.

4.2.1 Alignment of the spectra

As a preparation for our last task, we want to see if the autoencoder can map spectra onto another spectrum, which is the average of all QC's. We use the data from lung cancer and prostate cancer versus healthy cohorts for this experiment. From figure 43 it becomes obvious that the spectra are all too similar to distinguish between them without further analysis.

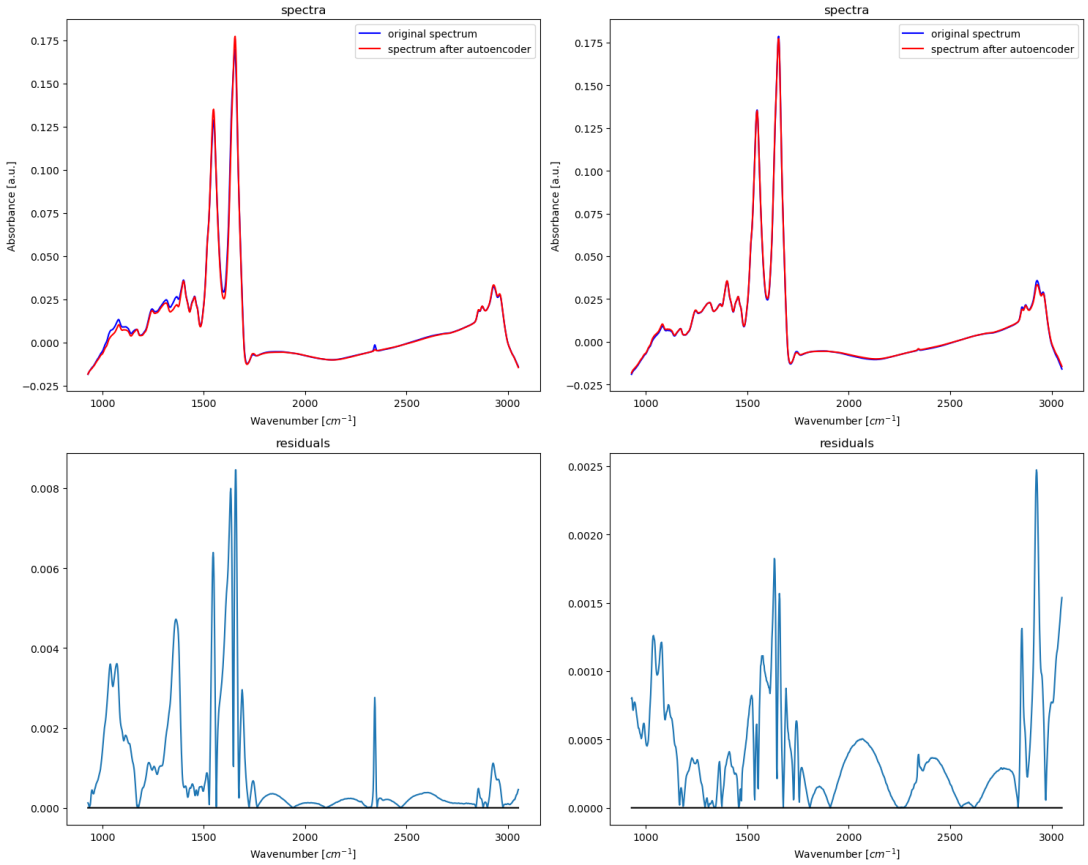


Figure 43: Examples of spectra that are mapped onto a fixed spectrum

We have evaluated the silent region and the whole spectrum using PCA and t-SNE. The silent region only gives us more information about how good the drifts can be transitioned, because of the predominance of the information from the drifts. In the PCA in figure 44 it is very well illustrated that the variance of the silent region has become very small after the autoencoder. This is a sign that the silent regions from the spectra are well aligned with the average QC. However, the t-SNE of that region (figure 45) reveals that there are still some differences in the drifts. The mapping is therefore not perfect. The same holds for the whole spectra as one can convince himself in figure 46 and 47. We conclude that

the autoencoder is able to map the whole spectrum as well as only the silent region onto another spectrum.

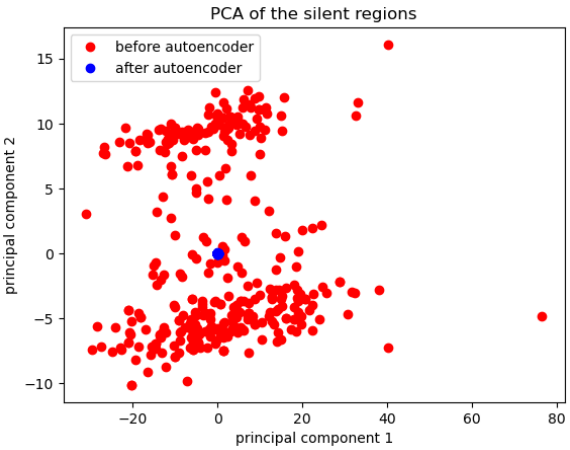


Figure 44: PCA of mapping onto single spectrum (silent regions)

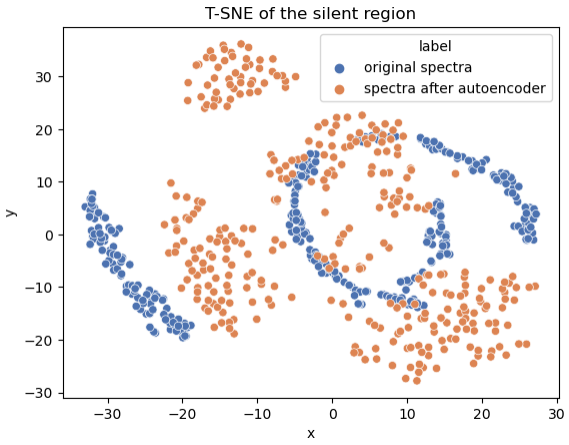


Figure 45: t-SNE of mapping onto single spectrum (silent regions)

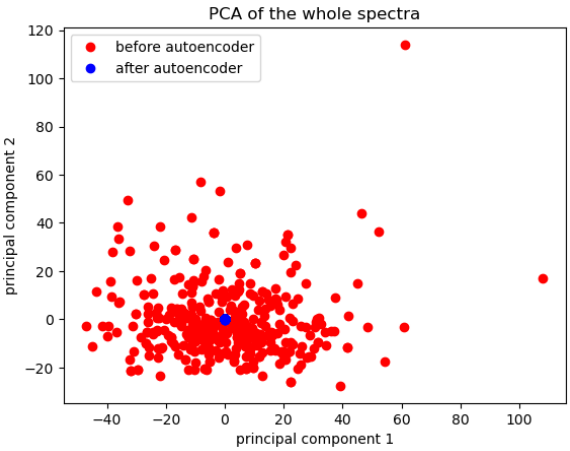


Figure 46: PCA of mapping onto single spectrum

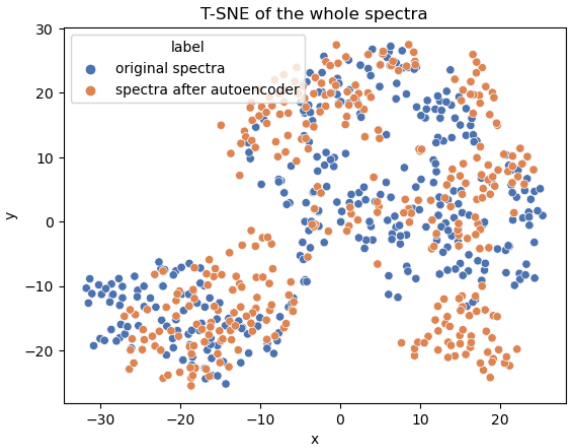


Figure 47: t-SNE of mapping onto single spectrum

4.2.2 Custom loss function

Now that we know the model can map the data onto a specific spectrum, we want to combine this with the ability of the model architecture to reproduce spectra. Our aim is to remove the drifts from the spectra while also keep the individual information in the spectra. To achieve this, we define a custom loss function that makes the model reproduce the spectra in the relevant region, while simultaneously mapping the silent region, where no relevant information is stored, to an arbitrary, fixed spectrum. In this analysis, again, we use the average of all QC spectra as fixed spectrum. The data used for the analysis is the LuCa vs. NSR set.
The custom loss function is based on the mean squared error. For network predictions $\vec{f}$, fixed spectrum $\vec{\tilde{y}}$ and target spectrum $\vec{y}$ it is defined as follows:

$$\mathcal{L}_{custom} = \begin{cases} \lambda \cdot \frac{1}{k} \sum_{i=1}^{k} \sum_{j=1}^{m} (f_j(x_i) - \tilde{y}_j(x_i))^2 & , \text{ silent region} \\ \frac{1}{k} \sum_{i=1}^{k} \sum_{j=1}^{m} (f_j(x_i) - y_j(x_i))^2 & , \text{ else} \end{cases} \tag{36}$$

The total loss is then the sum of both regions. λ is an amplification parameter that increases the importance of the loss for the silent region. Probably because of the values of that region are by trend smaller and more similar than the values of, for example, the peaks in the spectra, we observed that before introducing λ, the model paid less attention to mapping the silent region as desired. An amplification parameter of $\lambda = 1000$ led to the best results out of the tested amplification parameters. Additionally to the preprocessing we have always done, we normalized the spectra before forwarding them to the model. The performance is evaluated by doing t-SNE on the relevant region of the spectrum and both PCA and t-SNE on the silent region for the information.

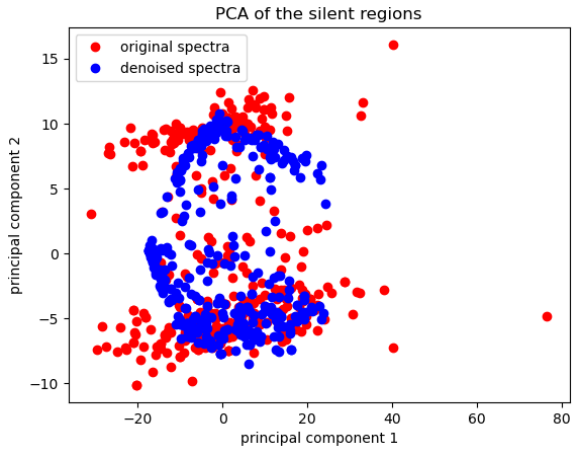


Figure 48: PCA of the silent regions with engineered loss function

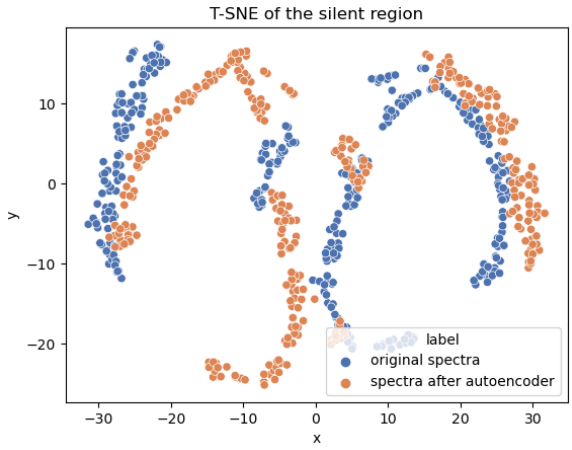


Figure 49: t-SNE of silent regions with engineered loss functions

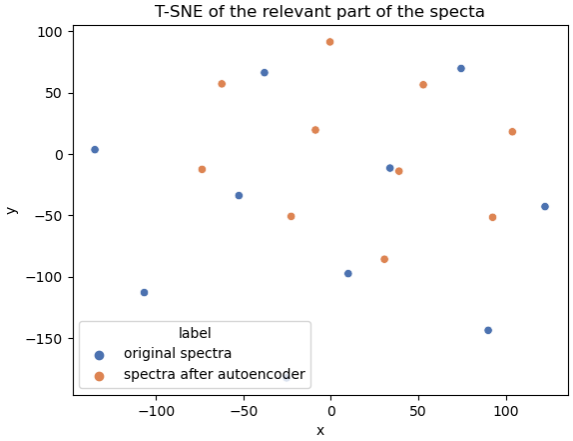


Figure 50: t-SNE of relevant region with engineered loss function

In the t-SNE of the relevant part the randomly selected spectra pairs are clearly distinguishable from one another. For the silent region the t-SNE plot looks ambiguous. There

is still some structure recognizable for the spectra after the autoencoder, meaning that the silent regions are not identical to the average QC spectrum. However, when looking at the PCA of the silent region, one can observe that the variance of the denoised data is much smaller than before. There is obviously some adjustment happening. To improve the results further, we decided to leave out the part of the spectrum with higher wavenumbers than the silent region from the optimization. Thus we were able to achieve a better assignability of the spectra pairs (see figure 58). The variance of the silent region could not be improved by this, because it depends on the amplification parameter which is fixed. In general, a higher amplification parameter can reduce the variance of the silent regions in the PCA even more, at the expense of reproducability of the relevant part.

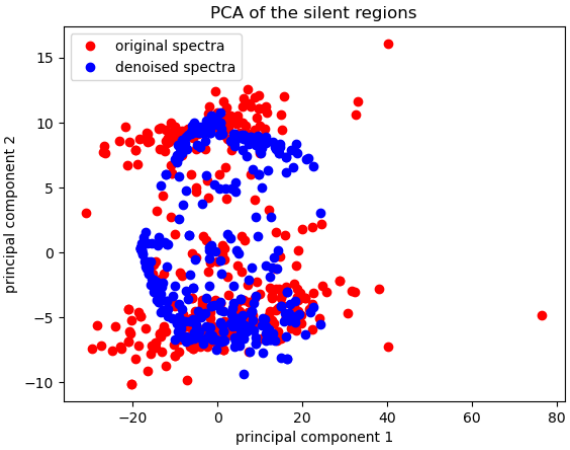


Figure 51: PCA of the silent regions with improved engineered loss function

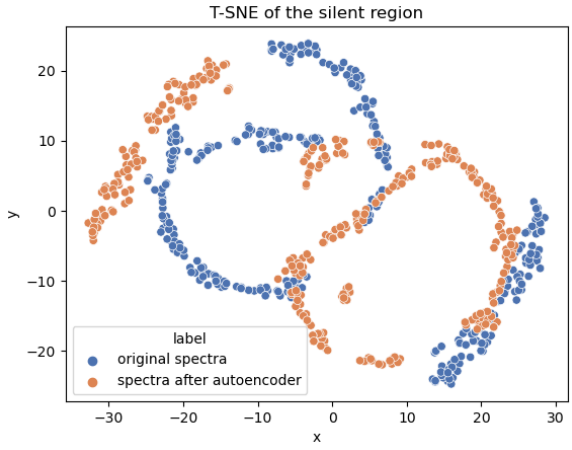


Figure 52: t-SNE of silent regions with improved engineered loss functions

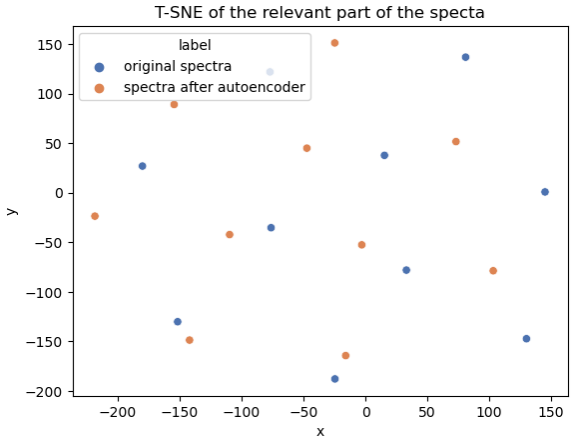


Figure 53: t-SNE of relevant region with improved engineered loss function

Eventually, we want to see if preprocessing with the autoencoder and custom loss yields better disease detection than with the standard preprocessing method. Therefore we use the LuCA vs. NSR data for classification and a set of generated spectra from this data to train the autoencoder architecture. All data is standard scaled. The classifier is a

SVM with linear Kernel, we do Grid search to determine the optimal C-parameter for the SVM. Then we do a Cross-validation Prediction for all test sets (not preprocessed, standard preprocessing method and preprocessed via the autoencoder). The results are summarised in the ROC of figure (54):

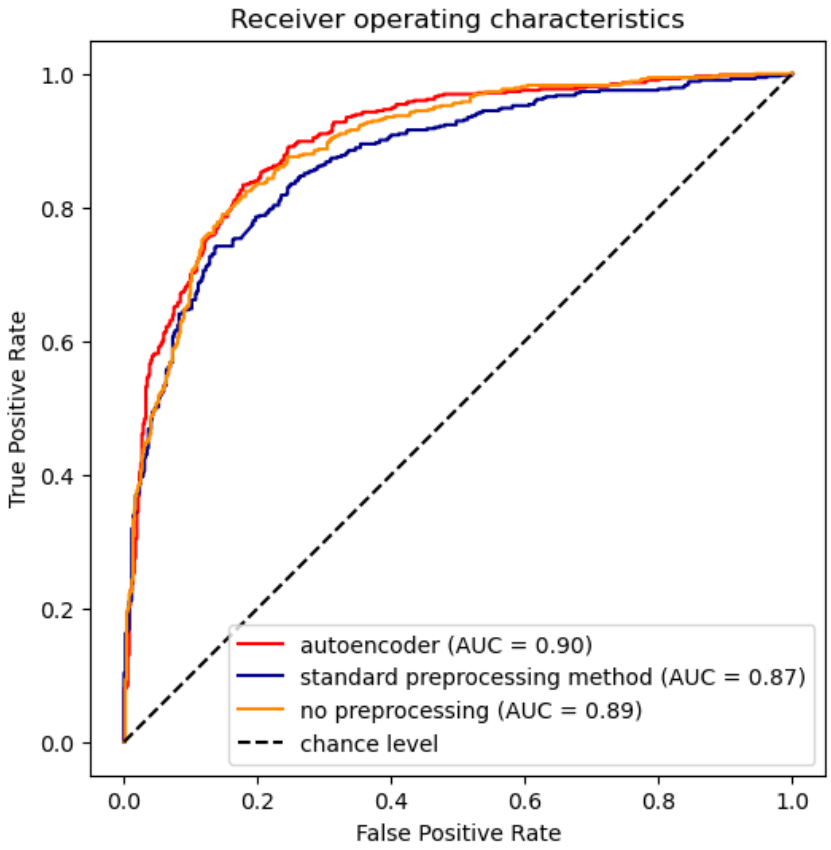


Figure 54: ROC for the classifier with denoising autoencoder and standard preprocessing method

The AUC's are:

Preprocessing method	**AUC**
Autoencoder	0.90 ± 0.03
No preprocessing	0.89 ± 0.03
Standard preprocessing method	0.87 ± 0.03

With the $AUC_{autoencoder}$ being by about 3 higher than the AUC for the standard preprocessing method $AUC_{standard} = 0.87$, we can conclude the project with a method that outperforms the current preprocessing of spectra for classification. Note that the standard preprocessing method in other examinations, for example in Yun (2022), has reached AUC's up to 0.89 for specific data sets. Therefore we assume a bias in our case that leads to worse performance of the standard preprocessing method.

5 Conclusion

Chapter 2 provided an insight into modern FTIR spectroscopy and the physical background behind it. It also introduces the basic ideas of Machine learning and especially Deep learning.

In section 3 the process of developing an autoencoder that can denoise the spectra measured with an FTIR-spectrometer was described. On the way, we have demanded the autoencoder to solve problems of increasing complexity, which required us to improve our model step by step. Starting with a fully-connected neural network, we chose to change to a CNN during the project. We have found an architecture that outperforms the prevailing preprocessing method when it comes to classification of cancer and non-cancer patients.

However, the final evaluations of the model have shown that the alignment of the drifts is not perfect. If the tuning of the hyperparameters would be more systematic, perhaps it is possible to achieve a better performance in both alignment and reproduction. Time and computing power were the limiting factors here. A proposal for an in-depth optimization of the model would be to use Grid Search to determine factors such as activation function, nodes and layer sizes, training regulations and the introduced amplification parameter.

With the autoencoder, we have found a transformation that can possibly remove the drift pattern from the silent region to a certain extend while reproducing the other part of the spectrum, such that the relevant information encoded in the data can be retrieved from the model output. This result can provide a contribution for further research on pattern recognition in spectra. An important open question is whether the transformation learned by the autoencoder does reduce the influence also in the relevant part of the spectrum.

A Appendix

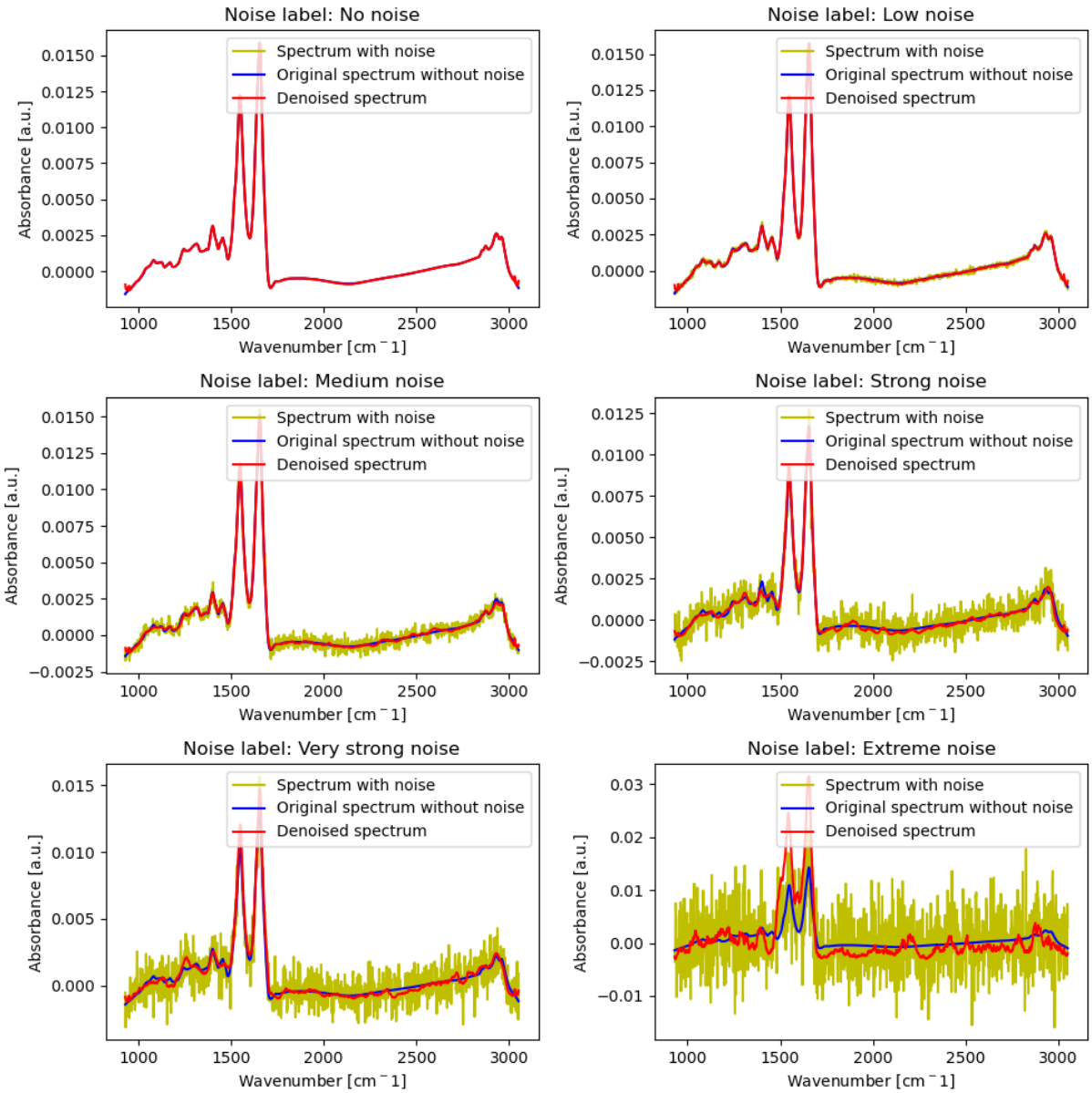


Figure 55: Removing artificial white noise with the simple CNN

Some evaluations with $\lambda = 1000$, but mistakenly chose the wrong axis to average the QC's:

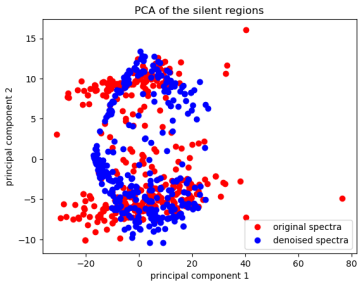


Figure 56: PCA of the silent regions with improved engineered loss function

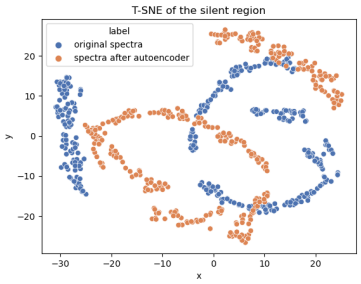


Figure 57: t-SNE of silent regions with improved engineered loss functions

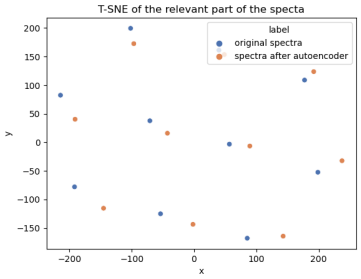


Figure 58: t-SNE of relevant region with improved engineered loss function

B Electronic appendix

The metafiles for the cancer cohorts used in this thesis are stored in a repository of the BIRD group at the Chair of Experimental Physics at Ludwig-Maximilians-Universität München. These are available upon reasonable request from Dr. Kosmas Kepesidis. The spectal generator is also available on git. The python scripts behind this thesis can be viewed in another repository (see `https://github.com/corinnawegner/denoising_fingerprints_code`).

References

(n.d.).

Andrews, D. G. (2010). *An Introduction to Atmospheric Physics*, 2 edn, Cambridge University Press.

Backhaus, K., Erichson, B., Plinke, W. and Weiber, R. (2021). *Multivariate Analysemethoden*, Springer Gabler Wiesbaden.

Bienz, S., Bigler, L., Fox, T. and Meier, H. (2012). *Spektroskopische Methoden in der organischen Chemie*, Thieme.

Brooks, R. L. (2013). *The Fundamentals of Atomic and Molecular Physics*, Springer.

Chiang, H.-T., Hsieh, Y.-Y., Fu, S.-W., Hung, K.-H., Tsao, Y. and Chien, S.-Y. (2019). Noise reduction in ecg signals using fully convolutional denoising autoencoders, *IEEE Access* **7**.

Chollet, F. (2016). Building autoencoders in keras.
URL: *blog.keras.io/building-autoencoders-in-keras.html*

Demtröder, W. (2017). *Experimentalphysik 2 - Elektrizität und Optik*, Springer Spektrum.

Demtröder, W. (2018). *Experimentalphysik 1 - Mechanik und Wärme*, Springer Spektrum.

Demtröder, W. (2019). *Electrodynamics and Optics*, Springer.

Erdmann, M., Glombitza, J., Kasieczka, G. and Klemradt, U. (2021). *Deep Learning for Physics Research*, World Scientific Publishing Company.

Erdmann, M., Schlüter, F. and Šmída, R. (2019). Classification and recovery of radio signals from cosmic ray induced air showers with deep learning, *JINST* **14**.

Goodfellow, I., Bengio, Y. and Courville, A. (2016). *Deep learning*, Cambridge.

Griffiths, D. J. (2018). *Elektrodynamik (4th ed.)*, Pearson.

Géron, A. (2020). *Praxiseinstieg Machine Learning mit Scikit-Learn, Keras und TensorFlow*, O'Reilly.

Günzler, P. D. H. and Gremlich, D. H.-U. (2003). *IR-Spektroskopie: Eine Einführung*, Wiley.

Hern, A. (2022). Ai bot chatgpt stuns academics with essay-writing skills and usability, `https://www.theguardian.com/technology/2022/dec/04/ai-bot-chatgpt-stuns-academics-with-essay-writing-skills-and-usability`. Accessed: 2023-02-01.

Huber, M., Kepesidis, K. V., Voronina, L., Fleischmann, F., Fill, E., Hermann, J., Koch, I., Milger-Kneidinger, K., Kolben, T., Schulz, G. B., Jokisch, F., Behr, J., Harbeck, N., Reiser, M., Stief, C., Krausz, F. and Zigman, M. (2021). Infrared molecular fingerprinting of blood-based liquid biopsies for the detection of cancer, *eLife* **10**: e68758.
URL: *https://doi.org/10.7554/eLife.68758*

Keras Team (n.d.). Keras api reference, `https://keras.io/api/`.

Knabner, P. and Barth, W. (2013). *Lineare Algebra*, Springer Spektrum Berlin.

Lambert, J. B., Gronert, S., Shurvell, H. F. and Lightner, D. A. (2012). *Spektroskopie*, Pearson.

Leonardo, C. (2021). *Infrared spectroscopy of blood-based biofluids*, PhD thesis, Ludwig–Maximilians–Universität, München.

Parson, W. W. (2009). *Modern Optical Spectroscopy*, Springer.

van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne, *Journal of Machine Learning Research* **9**.

Vijay, P. (2020). Timeseries anomaly detection using an autoencoder.

Yun, J. (2022). *Reliability of infrared molecular fingerprinting for the detection of cancer under realistic conditions*, Master's thesis, Ludwig-Maximilians-Universität, München.

Zeng, X. and Martinez, T. R. (2000). Distribution-balanced stratified cross-validation for accuracy estimation, *Journal of Experimental & Theoretical Artificial Intelligence* **12**(1): 1–12.
URL: *https://doi.org/10.1080/095281300146272*

Declaration of authorship

I hereby declare that the report submitted is my own unaided work. All direct or indirect sources used are acknowledged as references. I am aware that the thesis in digital form can be examined for the use of unauthorized aid and in order to determine whether the report as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future Theses submitted. Further rights of reproduction and usage, however, are not granted here. This paper was not previously presented to another examination board and has not been published.

Location, date

Name