GHENT UNIVERSITY

FACULTY OF SCIENCES

Towards a data-driven identification of the microbial “Ramanome”

Katja D’haeyer

Master dissertation submitted to
obtain the degree of
Master of Statistical Data Analysis

Promoter: Prof. dr. Willem Waegeman
Tutors: Peter Rubbens & dr. Christina Papagiannopoulou
Department of Data analysis and mathematical modelling

Academic year 2018 – 2019

The author and the promoter give permission to consult this master dissertation and to copy it or parts of it for personal use. Each other use falls under the restrictions of the copyright, in particular concerning the obligation to mention explicitly the source when using results of this master dissertation.

Katja D’haeyer
August, 2019

Foreword

This thesis is the endpoint of a supplementary course Master of Science in Statistical Data Analysis. When I graduated from the University of Ghent, I never thought I would be coming back and certainly not to study statistics. But 10 years later and following technological evolutions, the topic caught my attention. It became something I wanted to know more about, then something I wanted to apply in the context of my computational background. Returning – at least partly – to the beloved 'boerekot' made it all the better.

So, I'm absolutely grateful to my promotor prof. dr. Willem Waegeman for giving me the opportunity to work on this thesis and letting me combine my interests in life science and informatics.

Furthermore, I would like to thank prof. dr. Willem Waegeman, Peter Rubbens and dr. Christina Papagiannopoulou for their support and supervision throughout this thesis as well as for reviewing it.

Last but not least, I would like to thank my parents for their never-ending support, my friends for bringing the necessary distraction and the data science colleagues for putting things in inspiring perspectives. Without them, this work would not have been possible.

August 2019
Katja D'haeyer

Table of contents

Abstract/summary

The subject of this thesis is a data-driven identification of microbial Raman spectra, called the "Ramanome". This unique biochemical Raman fingerprint at the single-cell level represents a phenotypic profile of a live cell. Although the technique has a history starting in the '70s, the full potential of the technique could only be explored by using powerful statistical and computational methods. This led to the use of machine learning methods. Data is crucial here and a data analysis pipeline needs to be used: pre-treatment to correct non-sample dependent artefacts, pre-processing consisting of smoothing, baseline correction, normalization and data reduction followed by a final modelling step.

From literature, the impact of baseline correction on the performance of machine learning methods is known as well that a wide range of baseline correction methods are used. But this step can be avoided by applying Convolutional Neural Networks (CNNs) since they combine pre-processing, feature extraction and classification in a single architecture. Therefore, this deep learning architecture is the main focus of this thesis.

First conventional machine learning techniques were applied and assessed on two datasets: one measured stress responses on single cells of E. coli and the other dataset contained Raman spectra of human pathogens. A support-vector machine (SVM) in combination with principal components analysis (PCA) gave the best mean classification accuracy for the stressor dataset, while random forests (RFs) with PCA tuning worked best for the pathogens. These results form the baseline performance, a reference to which further results will be compared.

Continuing with CNNs we found that our custom models – based on the LeNet architecture and tuned via Bayesian Optimization – performed better than the models described in literature for Raman spectra classification. It became clear that a simple CNN without pooling performed better because it preserves the exact locations of the spectral peaks. Although, when comparing results, one needs to be attentive to the fact that machine learning experiments suffer from reproducibility issues.

Furthermore, the generalization of our models was assessed by selecting the best models on one dataset and retrain them from scratch on the other dataset. Again, the presence or absence of pooling had its effect on maintaining the peak patterns which affected the accuracy.

Finally, given the limited size of the stressors dataset, transfer learning was applied with the best performing models on the pathogen data. In this case, one uses an existing model that was trained on more data and the learned features are re-used for a new model. The results are among the lowest throughout this thesis, but may come as no surprise, since the stressor dataset benefits from simple models without pooling layers. And these were not the type of models that performed best on the pathogen data. With more data for the stressors and given the good generalization to the pathogen dataset, the reverse experiment would be interesting.

Keywords: Raman spectroscopy, Ramanome, machine learning, deep learning, Convolutional Neural Networks, transfer learning

1 Introduction

This chapter aims to introduce Raman spectroscopy. A brief overview of scientific research will be given with an emphasis on microbial applications. The full potential of the technique could only be explored by using powerful statistical and computational methods. This is where machine learning methods come into play. Data is crucial here so we will take a dive into the full data analysis pipeline covering pre-treatment, pre-processing and finally modelling. The chapter will end with a formulation of our research goals for this thesis.

1.1 Literature study

Raman spectroscopy is a technique used to characterize vibrational, rotational and other low-frequency modes of molecules. These molecules can be inorganic (e.g. minerals) as well as organic (e.g. in single bacterial cells). Raman spectroscopy relies on inelastic scattering of monochromatic light and returns a fingerprint that gives a quantitative description concerning the molecules that are present.

The potential of Raman spectroscopy for the characterization of biological samples like DNA, proteins and lipids was recognized in the early ‘70s. Nevertheless, it took until the 2000s before the potential could be fully utilized due to instrumental and computational needs. Powerful statistical and computational methods are needed in order to translate the Raman spectral signals into meaningful bio-medical information (Ryabchykov, Guo and Bocklitz 2018). On the one hand, Raman spectroscopy can be used to obtain information about single cellular components (characterization) while on the other hand, the whole Raman spectrum is a valuable asset to discriminate between bacterial species (identification) (Lorenz, et al. 2017).

In this thesis the focus will be on identification – or classification – of microbes. We will use their own unique biochemical Raman fingerprint at the single cell level, proposed as the ‘Ramanome’ (Teng, et al. 2016). The ‘Ramanome’ incorporates information coming from different kinds of molecules such as lipids, proteins and carbohydrates (Huang, et al. 2010). It represents a phenotypic profile of a live cell (Lorenz, et al. 2017). Furthermore, Raman spectroscopy requires no cultivation and is obtained in a non-disruptive manner. This gives rise to different applications, e.g.:

- Detection and identification of microorganisms in samples from the environment, water, food and clinical settings (Lorenz, et al. 2017).
- In case of infections it can help in rapid determination and decision about which antibiotic to apply without the need for time-consuming culturing (Ho, et al. 2019).
- Detect and characterize a stress response (antibiotics, heavy metals,…) depending on time and dosage (Teng, et al. 2016).

Since the 'Ramanome' represents a phenotype it also contains a downside. It is sensitive to individual variations, cellular state, environmental factors and even sample processing (Lorenz, et al. 2017); (García-Timermans, Rubbens and Kerckhof, et al. 2018). Besides, measurement conditions affect the recorded Raman spectrum and as such spectra do not solely reflect the sample. A calibration procedure is often required to reduce the introduced spectral changes, which includes wavenumber calibration and intensity calibration (Bocklitz, et al. 2015).

So in order to compare spectra – like in certain applications above - or obtain reproducibility, one needs standardized methods both in sample preparation, instrument use and data processing (García-Timermans, Rubbens and Kerckhof, et al. 2018). For the latter Ryabchykov, Guo and Bocklitz (2018) propose a data analysis pipeline: pre-treatment to correct non-sample dependent artefacts, pre-processing (smoothing, baseline correction, normalization and data reduction) and finally modelling.

Liu, et al. (2017) already applied the data analysis pipeline to mineral Raman spectra and mention the impact of pre-processing – baseline correction in particular – on the performance of machine learning systems. To avoid this separate pre-processing step, they introduce the application of Convolutional Neural Networks (CNNs). CNNs have the advantage of combining pre-processing, feature extraction and classification in a single architecture that can be trained end-to-end with no manual tuning and it achieves higher accuracy.

In combination with several machine learning methods, such as the k-nearest neighbours algorithm (KNN), gradient boosting, random forests (RFs), support vector machines (SVMs) and CNNs Liu, et al. (2017) compared six widely-used baseline correction methods: (1) modified polynomial fitting, (2) rubber band, (3) robust local regression estimation, (4) iterative restricted least squares, (5) asymmetric least square smoothing and (6) rolling ball. For the conventional classification methods, principal components analysis (PCA) was adopted to

reduce dimensionality and extract features, except for RFs where they found that PCA decreased the performance.

Liu, et al. (2017) concluded that baseline correction greatly improved the performance of all the conventional methods by 20%–40% but CNN's performance dropped by about 0.5%–2.5%. This may indicate that CNNs were able to learn a more efficient way of handling the interference and to retain more discriminant information than using an explicit baseline correction method.

Ryabchykov, Guo and Bocklitz (2018) also discuss pre-processing and highlight the modified polynomial fit, the asymmetric least squares baseline estimation and the statistics-sensitive non-linear iterative peak-clipping (SNIP) algorithm. The last one, in contrast to the others, does not lead to oscillations of the Raman baseline at the edges of the spectral interval. SNIP was applied in research by García-Timermans, Rubbens and Kerckhof, et al. (2018) while Ho, et al. (2019) used a polynomial fit of order 5 and Teng, et al. (2016) applied the baseline and normalization tools of a specific software suite for Raman spectra (Labspec 5). This indicates a wide diversity in pre-processing.

Regarding the models to classify Raman spectra, literature mentions different machine learning techniques. As already mentioned, Liu, et al. (2017) used KNN, gradient boosting, RFs, SVMs and CNNs. Besides CNNs, linear SVMs outperform the others followed by KNN with 1 neighbour and last places are for RFs and Gradient boosting. Ho, et al. (2019) applied logistic regression, SVM and an advanced CNN architecture in the form of Deep Residual learning. Logistic regression performed slightly better here than SVM.

In a broader perspective of bioinformatics problems, a ranking was made of the performance of 13 machine learning algorithms on classifying 165 datasets. It shows the strength of ensemble-based tree algorithms in generating accurate models: gradient tree boosting (first place), RFs (second), SVMs (third), logistic regression (7^{th}) and KNN (8^{th}). However, this neglects the fact that the top-ranked algorithms may not outperform others for certain problems. With this in mind, the percentage of datasets for which one algorithm outperforms another was calculated. So e.g. 33% of the times RFs outperforms SVM and 21% of the times SVM outperformed RFs. KNN outperforms RFs 8% and SVM 7% of the times while in 65% of the

cases RFs or SVM outperforms KNN. Furthermore, it is – not surprisingly - advised to tune the hyperparameters because it often improves the algorithms accuracy (Olson, et al. 2018).

1.2 Goals

This thesis will evaluate the potential to discriminate between microbial species based on their 'Ramanome'. Since the Raman spectra are accompanied with labels, this is a supervised classification problem. We will study how different conventional machine learning techniques and neural networks manage to differentiate by comparing their obtained accuracies on different datasets. The focus will be on deep learning, where different architectures and implementations are investigated. Inspired by architectures and literature models, we will try to achieve better classification rates by developing our own models. Attention will be paid to generalization to other datasets: whether it is by re-training a model on another dataset or by using the knowledge of one model and applying it onto another dataset (transfer learning).

2 Data & methods

In this chapter we will first familiarize ourselves with the two available datasets. Next, conventional machine learning techniques are briefly discussed, followed by a more in-depth overview of neural networks and deep learning. The last paragraph will explain how the evaluation of different models on both datasets will be done.

2.1 Data

Two datasets with Raman spectra of microbes will be used for this thesis: one was collected by Teng, et al. (2016) and measured stress responses on single cells of E. coli. The other dataset contains Raman spectra of human pathogens, composed by Ho, et al. (2019). Further on these data will be referred to as the 'stressor dataset' and 'pathogen dataset'.

2.1.1 Stressor dataset

The first Raman dataset contains measurements of the stress response under the influence of different stressors and durations on single cells of E. coli in wavelength range 594 – 1840,39 cm^{-1}. The data contains 5284 spectra, each measured at 1256 wavelengths. Six chemical stressors from 3 categories, i.e., antibiotics of ampicillin (Amp) and kanamycin (Kan), alcohols of ethanol (Eth) and n-butanol (n-But) and heavy metals of Cu^{2+} ($CuSO_4$) and Cr^{6+} (K_2CrO_4), were compared at 7 time points (5, 10, 20, 30, 60, 180 and 300 min). Doses of the chemicals were each set at a level that causes > 50% inhibition of growth within an 8-hour culture (Teng, et al. 2016). The dataset thus consists of a combination of 6 stressors at 7 time points. For each stressor a control group exists, also measured at 7 time points. At start time a zero measurement without stressor was done. Stress responses were measured for about 60 cells per treatment; 20 cells from each one of the three biological replicates of cell culture (Figure 2-1).

time			h3			h5			min0			min05			min10			min20			min30			min60	
rep	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
tmt																								
amp	20	20	20	20	20	20	nan	nan	nan	20	20	20	17	18	18	20	20	19	20	21	21	20	20	21
but	16	16	15	20	20	20	nan	nan	nan	25	24	24	24	24	24	19	18	19	24	24	23	20	20	21
camp	20	20	20	20	20	20	nan	nan	nan	20	20	20	19	19	18	20	20	20	20	21	21	20	20	21
cbut	17	17	16	23	24	24	nan	nan	nan	23	23	23	24	24	23	23	23	24	23	22	22	24	23	23
ccr	18	18	18	20	20	20	nan	nan	nan	20	20	20	20	20	20	20	20	20	20	20	20	20	19	19
ccu	18	18	18	20	20	20	nan	nan	nan	20	20	21	20	20	20	19	19	19	20	20	20	20	19	19
ceth	20	19	19	20	20	20	nan	nan	nan	20	20	20	20	19	19	20	21	21	21	21	21	22	22	22
ckan	20	20	20	20	20	20	nan	nan	nan	20	19	19	20	20	19	20	20	20	20	21	21	20	20	20
cr	20	20	20	20	20	20	nan	nan	nan	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
cu	20	20	20	20	20	20	nan	nan	nan	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
empty	nan	nan	nan	nan	nan	nan	54	54	54	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
eth	22	22	21	20	20	21	nan	nan	nan	20	20	20	22	22	22	22	23	23	20	20	20	20	20	20
kan	20	20	20	20	20	20	nan	nan	nan	20	20	20	23	23	23	20	20	20	20	21	21	20	20	21

Figure 2-1: Number of Raman samples per treatment (tmt), time point (time) and replicate (rep). Dataset is more or less balanced: some replicates have less than 20 cell measurements (blue) or more than 20 (red). Treatments with class 'empty' are measurements at start time without stressor. Control groups are indicated with a 'c' preceding their treatment class.

Because the control groups should show no stress response but only a natural biological evolution during time, all control groups were combined into one group per time point (including one at the start of the experiment). This results in 8 control groups. To keep the data balanced, these groups were subsampled to 20 Raman samples per replicate.

The resulting dataset contains thus samples for 6 stressors at 7 time points plus 8 control groups, which gives 50 classes, each with about 20 samples per replicate or 60 per class.

2.1.2 Pathogen dataset

The second Raman dataset (Ho, et al. 2019) was based on single cell measurements of human pathogens in wavelength range 391,98 – 1792,4 cm^{-1} and measured at 1000 wavelengths. It is a balanced dataset that consists of 30 bacterial and yeast species with 2000 spectra each. Since the focus of the paper was on human health and common antibiotics often have activity against multiple species, the species can also be arranged into a higher level grouping based on the antibiotic treatment. The idea is that the predicted species can be wrong but it is less bad if it is within the same antibiotic grouping so that at least a correct treatment is applied.

2.1.3 Baseline correction and normalization

As discussed in Chapter 1.1, raw (uncorrected) Raman spectra are preferred given the different possibilities of baseline corrections and impact of preprocessing on the data. When applying a CNN, this step can be integrated into the model itself which is a big advantage. Unfortunately, both datasets were already baseline corrected and normalized, the original data could not be obtained.

- The stressors dataset was baseline corrected and normalized via the tools of a specific software suite for Raman spectra (Labspec 5). For each spectrum, the intensities sum to 100. In Figure 2-2, an example shows the mean Raman intensities (blue line) for stress response measurements on ethanol treatment after 5 hours with indication of standard deviation (red band) and min/max measurements (green band).

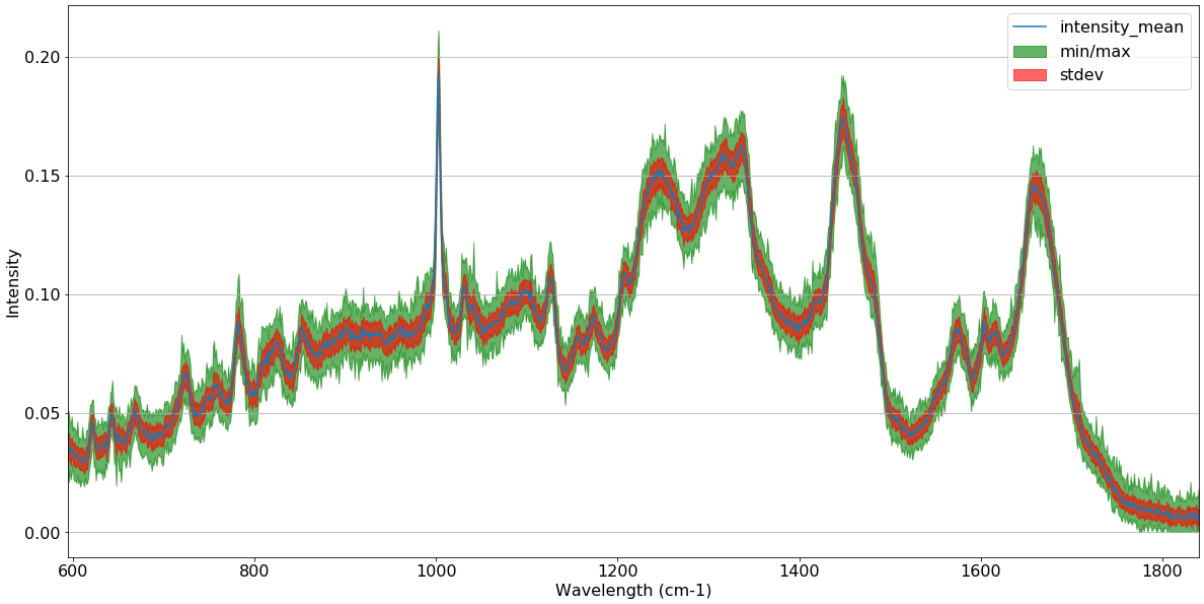

Figure 2-2: Mean Raman intensities (blue line) for stress response measurements on ethanol treatment at 5h time point with indication of standard deviation (red band) and min/max measurements (green band). The Raman spectra are baseline corrected and normalized.

- The pathogens spectra are individually background corrected using a polynomial fit of order 5 using the subbackmod Matlab function available in the Biodata toolbox. Spectra were individually normalized to run from a minimum intensity of 0 to maximum intensity of 1 within this spectral range (Ho, et al. 2019). In this case spectrum intensities do not sum to the same value but are in the range 65,8 - 636,6. In Figure 2-3 an example shows mean Raman intensities (blue line) for pathogen 'E. coli strain 1' with indication of standard deviation (red band) and min/max measurements (green band).

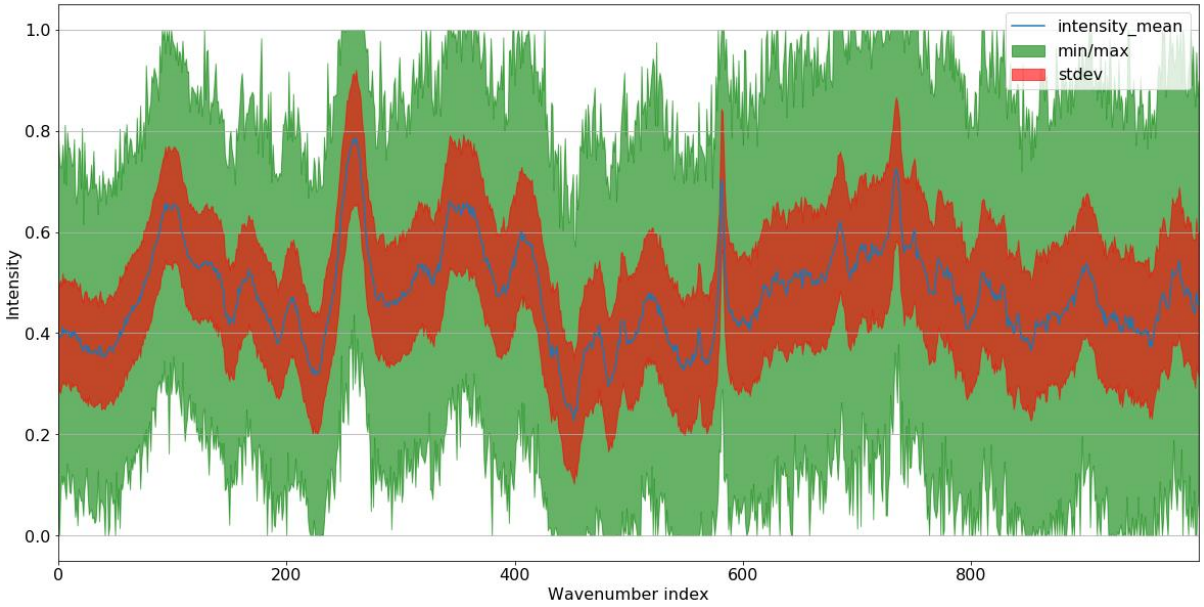

Figure 2-3: Mean Raman intensities (blue line) for pathogen 'E. coli strain 1' with indication of standard deviation (red band) and min/max measurements (green band). The Raman spectra are baseline corrected and normalized.

2.2 Machine learning techniques

In this chapter the machine learning techniques used in this thesis – based on a selection of the methods found in literature – will be discussed. With machine learning we mean systems or models that can learn from data, rather than following programmed rules. Typically, one prefers a simple model above a complex model because it generalizes better (Occam's razor) and it is easier to use, train and explain (Alpaydin 2014). Therefore conventional techniques, selected from Liu, et al. (2017), are assessed first: an optional data reduction via principal component analysis (PCA) in combination with the k-nearest neighbours algorithm (KNN), random forests (RFs) or linear support vector machines (SVMs). This is followed by an overview of the characteristics and the evolution of neural networks. Different deep learning architectures are discussed, including the LeNet variant proposed by Liu, et al. (2017) and a Residual Network architecture (ResNet) proposed by Ho, et al. (2019). This allows to compare the performance of the two datasets with different learning techniques and to make a comparison with the results from literature.

2.2.1 Conventional techniques

2.2.1.1 Dimensionality reduction

Given the high-dimensional datasets, both Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE) can be used as a technique for

dimensionality reduction with regard to the visualization of the data. Note that PCA is a linear technique and t-SNE a non-linear technique. t- SNE is more suitable for visualisation while PCA is also used for creating predictive models.

t-SNE calculates the probability of similarity of points in high-dimensional space and in the corresponding low-dimensional space. The difference between the similarities in both spaces is then minimized in order to obtain a good representation in the low-dimensional space.

PCA is an unsupervised method, which means it does not take the label into account. It rather projects a high-dimensional dataset into a lower dimensional subspace where the variance is maximally maintained or information loss is minimized. As such, PCA leads to a data reduction but retains not necessarily information relevant for the next supervised classification task. Furthermore, large feature values lead to high variances so a normalization step is required preceding PCA.

2.2.1.2 k-Nearest neighbours algorithm (KNN)

A KNN classifier is a non-parametric method and assigns a label according to the class that is most frequent among its k neighbours. Since it is a distance-based method, there is also a dependency on data transformations (such as scaling) (Guido and Müller 2016). It also means that the model needs to 'remember' all data which is costly in memory. It is also costly in computation, since it requires calculating the distance from the input to all training instances (Alpaydin 2014). KNN algorithms have low bias and high variance since these models are influenced even by small changes in the training data.

2.2.1.3 Random forests (RFs)

A decision tree is also a non-parametric model since no parametric form is assumed for the class densities nor is the tree structure fixed a priori (Alpaydin 2014). A tree with increasing depth will decrease bias at the expense of increasing variance. RFs train several decision trees, each based on a different resampling of the original training data and takes a subset of the features to split on at each node. By creating many trees and averaging them, the bias of the RFs model is only slightly increased but the variance is greatly reduced.

2.2.1.4 Linear support vector machines (SVMs)

An SVM is a discriminant-based method which cares only about the instances close to the boundary and discards those that lie in the interior. A good model has a maximal separation between the classes but to avoid overfitting, it can allow for some misclassifications (Alpaydin 2014). SVMs can handle high dimensional data but do not scale well with the number of samples and require scaled data (Guido and Müller 2016). SVMs have low bias and high variance since these models are influenced even by small changes in the training data.

2.2.2 From artificial neural networks to convolutional neural networks

2.2.2.1 Artificial neural networks (ANN)

An ANN is loosely based on biological neural networks and thus a network of connected nodes (artificial neurons), organised in different layers. The layers are fully connected, which means that each node in a layer is connected with every node in adjacent layers.

In Figure 2-4, there are three layers: the input units where the data enters the network, a layer with hidden units where the data is transformed and an output layer. In the case of Raman spectra, every output node represents the probability that an input belongs to a certain microbial class.

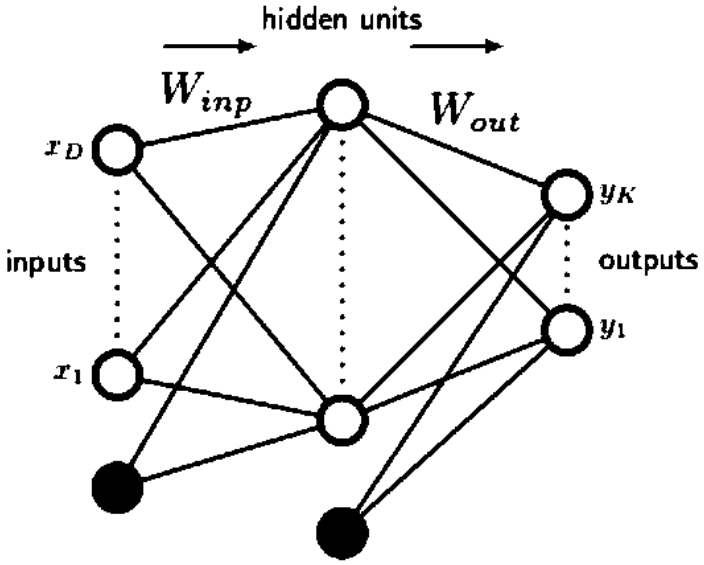

Figure 2-4: Network diagram for a two-layer neural network. The input, hidden and output variables are represented by nodes, while the weight parameters are shown as links between the nodes. Bias parameters are denoted by links coming from full black nodes (Bishop 2006).

Suppose we have Raman spectra as input of dimension D where each variable $x_1, \dots x_D$ represents a spectral measurement at a certain wavelength. We can then construct M linear combinations of the input variables, one for each of the M nodes in the hidden layer.

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \qquad with\ j = 1, \dots, M$$

The superscript (1) indicates that the corresponding parameters are in the first layer of the network. This linear combination or activation a_j is a weighted sum of the outputs of the units in the layer below with $w_{ji}^{(1)}$ representing the weights and $w_{j0}^{(1)}$ the bias. Each activation is then transformed into a hidden unit using a differentiable, non-linear activation function $h(\cdot)$ to give $z_j = h(a_j)$ e.g. rectified linear unit (ReLU) that zeros out negative values by taking the maximum in the set $(0, a_j)$. The hidden values now function as input that are again linearly combined to give K output activations

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} x_i + w_{k0}^{(2)} \qquad with\ k = 1, \dots, K$$

This transformation corresponds to the second layer of the network with weights $w_{kj}^{(2)}$ and $w_{k0}^{(2)}$ the bias parameters. The forward pass in a neural network is completed when – in case of multiclass problems – a Softmax activation function rescales the network outputs y_k to the range [0-1] with its sum being equal to 1. This can be seen as a probability distribution of an input belonging to each of the k classes.

A next step is the evaluation of the network performance for which a loss function is defined. The loss function indicates how good or bad the network output is with respect to the true label. Loss functions are chosen according to the problem at hand. Given the multiclass problem studied in this thesis, we will be using the cross-entropy, which is defined for N samples and K classes as follows:

$$L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} t_{i,k} \ln (\hat{y}_{i,k})$$

$t_{i,k}$ is 1 if observation i belongs to class k and 0 otherwise, $\hat{y}_{i,k}$ is the predicted probability that observation i is classified in class k.

The final step is the backpropagation in order to update the weights. Therefore, one needs to compute the gradient of the loss function with respect to the current weights and apply that gradient with a certain learning to the current weights. The backpropagation equation can be applied repeatedly to propagate gradients through all layers, starting from the output at the top (where the network performs its predictions) all the way to the bottom (where the external input is fed) (LeCun, Bengio and Hinton, 2015).

2.2.2.2 Deep learning

Deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher level features from raw input (Deng and Yu 2014). It starts, e.g. in image recognition, at the lower levels learning edges, textures,… while at higher levels more complex patterns are learned and allow to differentiate between an image of a cat or a dog.

Deep learning includes different architectures and each type is suitable for a different class of problems. For example, a multi-layer perceptron (MLP) connects every element of a previous layer to every element of the next layer and as such learns a global pattern. Of course, this results in lots of trainable weights, which make the network more complex to train, increasing the computational time. Furthermore, in image recognition a signal learned at one place had to be re-learned when it appeared somewhere else. This is where one looked for translation invariant methods and to learn local patterns, which led to CNNs. Due to the convolution, the same filter (and weights) is applied to every part of the input. A pattern learned at a certain position in an input can later be recognized at a different position. The weight sharing allows for a drastic reduction in trainable weights.

2.2.2.3 Convolutional neural networks architectures

The architecture of a typical CNN is structured as a series of stages. The first few stages are composed of two types of layers: convolutional layers and pooling layers. Units in a convolutional layer are organized in feature maps, within which each unit is connected to local patches in the feature maps of the previous layer through a set of weights called a filter bank. The result of this local weighted sum is then passed through a non-linearity such as a ReLU.

All units in a feature map share the same filter bank. Different feature maps in a layer use different filter banks.
Although the role of the convolutional layer is to detect local conjunctions of features from the previous layer, the role of the pooling layer is to merge semantically similar features into one. A typical pooling unit computes the maximum of a local patch of units in one feature map. Neighbouring pooling units take input from patches that are shifted by more than one row or column, thereby reducing the dimension of the representation and creating an invariance to small shifts and distortions (LeCun, Bengio and Hinton, 2015).

This combination of convolutional and pooling layers extracts features hierarchically. The extracted features go through some fully connected layers and a Softmax output layer, which can be viewed as a classifier (Liu, et al. 2017). As discussed previously in Chapter 1.1, the combination of feature extraction and classification allows for a goal-oriented implementation given the task at hand and it avoids the crucial step – certainly when working with Raman spectra – of data pre-processing.

Yann LeCun combined the ideas of CNNs and backpropagation, and applied them to the problem of classifying handwritten digits. The resulting network, LeNet-5, was used by the United States Postal Service in the 1990s to automate the reading of ZIP codes on mail envelopes (LeCun, Bottou, et al. 1998).

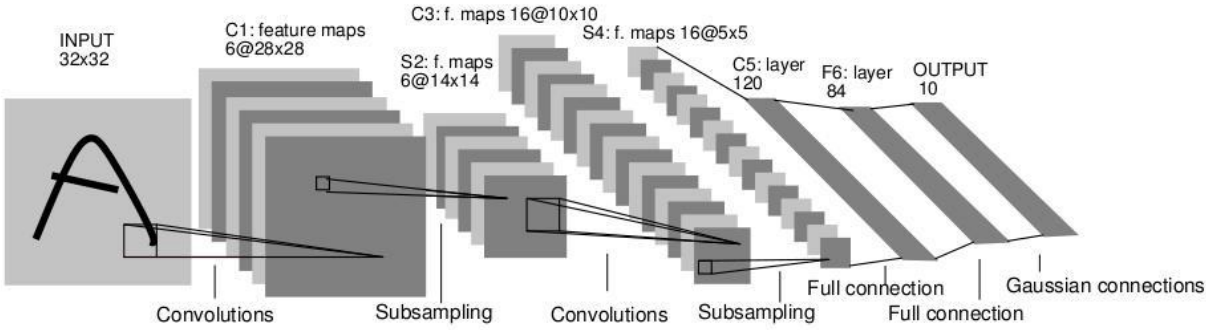

Figure 2-5: Architecture of LeNet-5, a CNN for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical (LeCun, Bottou, et al. 1998).

Figure 2-5 shows the architecture of LeNet-5, typically successive layers of convolutions and sub-sampling are alternated, resulting in a 'bi-pyramid': at each layer, the number of feature maps is increased as the spatial resolution is decreased due to the sub-sampling (LeCun, Bottou, et al. 1998). This network contains – without the input layer – 7 layers: 3 convolutional layers

with 2 average pooling layers followed by a fully connected layer at the end. At that time, it was common to use the sigmoid or tanh nonlinearity for the activation function.

Figure 2-6 shows a LeNet variant, proposed by Liu, et al. (2017) which was trained on mineral Raman spectra. Instead of two-dimensional images, it takes one-dimensional spectra as input. This CNN architecture consists of 3 sets of combined convolutional/maximum pooling layers for feature extraction and two fully connected layers for classification. Batch normalization is applied to the activations and allows to use much higher learning rates and being less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout (Ioffe and Szegedy 2015). Dropout is one of the most effective and most commonly used regularization techniques for neural networks. Dropout, applied to a layer, consists of randomly dropping out (setting to zero) a number of output features of the layer during training (Chollet 2018). As activation function, LeakyReLU is used which is an advanced ReLU. Instead of zero out negative values, a factor alpha is applied on negative values. LeakyRELU is more advantageous than sigmoid and tanh non-linearities because it allows for faster training.

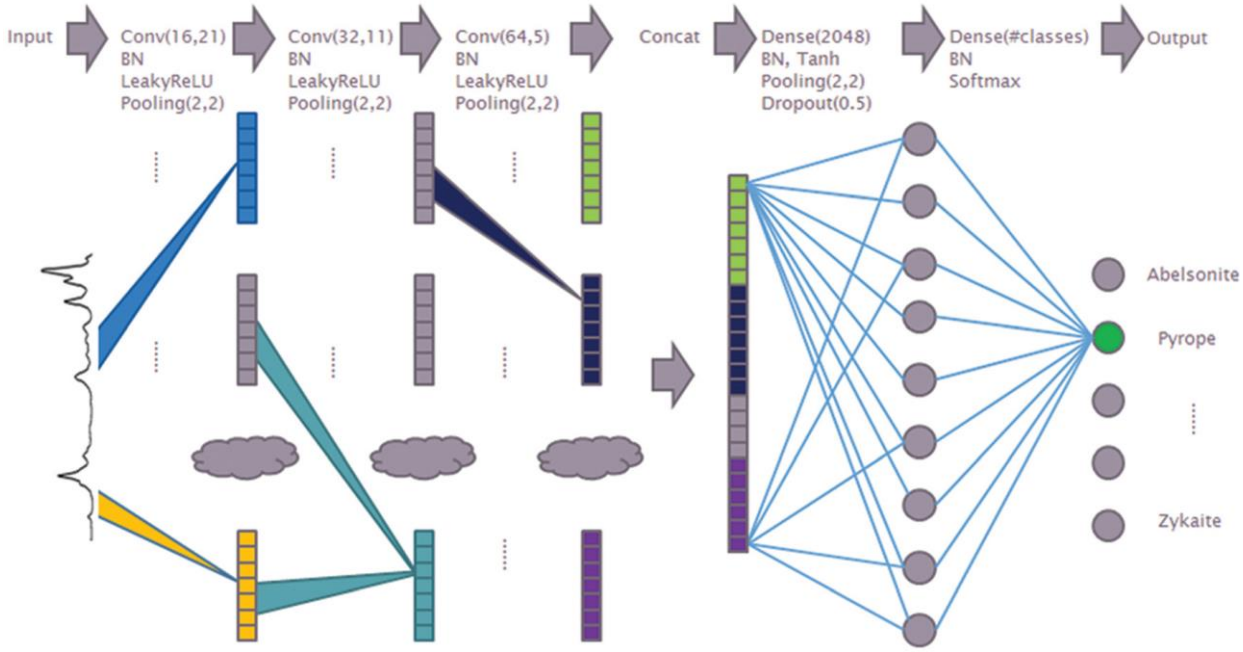

Figure 2-6: CNN architecture for Raman spectrum recognition as proposed by Liu, et al. (2017) . It consists of a number of convolutional and pooling layers for feature extraction and two fully connected layers for classification.

With the evolution of networks becoming deeper to augment the learning, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated and then

degrades rapidly. Unexpectedly, such degradation is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error (He, et al. 2015).

This problem was tackled by introducing a deep residual learning framework. A building block is shown in Figure 2-7A. Suppose $H(x)$ is an underlying mapping to be fit by a few stacked layers (at least 2), with x denoting the inputs to the first of these layers. Rather than expect stacked layers to approximate $H(x)$, we explicitly let these layers approximate a residual function $F(x) := H(x) - x$. The original function thus becomes $F(x) + x$ (He, et al. 2015).

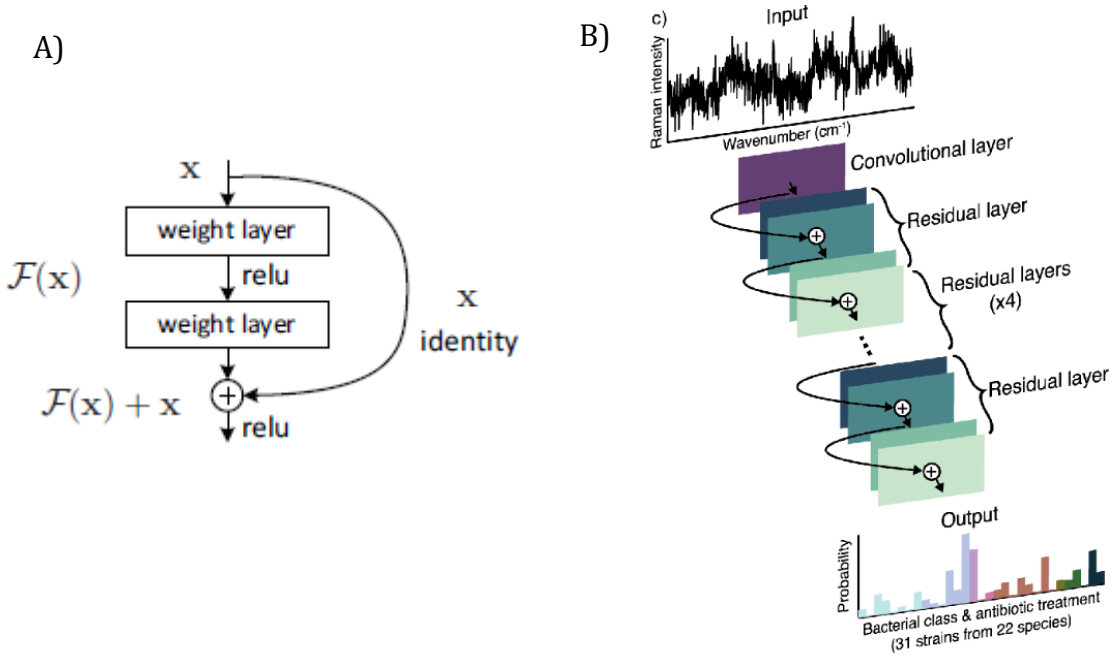

Figure 2-7: A) A residual learning building block (He, et al. 2015); B) the Residual Network (ResNet) architecture applied by Ho, et al. (2019)

This building block can be defined as:

$$y = F(x, \{W_i\}) + x$$

Here x and y are the input and output vectors of the layers considered. The function $F(x, \{W_i\})$ represents the residual mapping to be learned. The operation $F + x$ is performed by a shortcut connection and element-wise addition with a non-linearity applied after the addition. The dimensions of x and F must be equal. If this is not the case (e.g., when changing the input/output channels), a linear projection by the shortcut connection can be done to match the dimensions. Note that the shortcut connections introduce neither extra parameter nor computation complexity (He, et al. 2015).

An adaptation from the ResNet architecture (Figure 2-7B) was proposed by Ho, et al. (2019) to classify one-dimensional Raman spectra of human pathogens. Unlike previous work, no pooling layers but instead strided convolutions were used with the goal of preserving the exact locations of spectral peaks (Dumoulin and Visin 2016). A stride defines the step size of the convolutional kernel when it moves over the input. With a value more than one, it is another way to reduce dimensions.

The ResNet architecture consists of an initial convolution layer followed by 6 residual layers and a final fully connected classification layer. The residual layers contain shortcut connections between the input and output of each residual block, allowing for better gradient propagation and stable training. Each residual layer contains 4 convolutional layers, so the total depth of the network is 26 layers. The initial convolution layer has 64 convolutional filters, while each of the hidden layers has 100 filters. During training, the Adam optimizer was used with learning rate 0.001, beta1 0.5, beta2 0.999 and batch size 10 (Ho, et al. 2019).

2.2.2.4 Tuning hyperparameters: Bayesian optimization

In order to optimize the hyperparameters of deep learning models in an informed way Bayesian optimization was applied. In Bayesian optimization we define the range within which the hyperparameters need to be evaluated and describe our prior belief in the objective function via a surrogate model. This surrogate model is sequentially updated based on data coming in from different model evaluations. This is the Bayesian posterior that represents our updated beliefs given the data. A loss function indicates how well the hyperparameters work and by minimizing this function the best set of hyperparameters is selected.

The implementation was done via Hyperopt, a Python library for hyperparameter optimization that internally uses trees of Parzen estimators to predict sets of hyperparameters that are likely to work well (Chollet 2018). Whereas the Gaussian-process based approach modelled $p(y|x)$ directly, this strategy models $p(x|y)$ and $p(y)$. The tree-structured Parzen Estimator Approach (TPE) defines $p(x|y)$ using two densities:

$$p(x|y) = \begin{cases} l(x) \;\; if \; y < y^* \\ \\ g(x) \;\; if \; y \geq y^* \end{cases}$$

where $l(x)$ is the density formed by using the observations $\{x^{(i)}\}$ such that corresponding loss $f(x^{(i)})$ was less than y^* and $g(x)$ is the density formed by using the remaining observations. The TPE algorithm chooses y^* to be some quantile γ of the observed y values, so that $p(y < y^*) = \gamma$, but no specific model for $p(y)$ is necessary (Bergstra, et al. 2011).

One important issue to keep in mind when doing automatic hyperparameter optimization at scale is validation-set overfitting. Because the update of hyperparameters is based on a signal that is computed using the validation data, the hyperparameters are effectively trained on the validation data, and thus they will quickly overfit to the validation data (Chollet 2018).

2.2.2.5 Training with small datasets

Because the stressor dataset is small, data augmentation is a powerful technique for mitigating overfitting and allows to still train a CNN with sufficient data. An alternative way includes transfer learning approaches, where one uses an existing model that was trained on more data and where the learned features are re-used for a new model.

2.2.2.5.1 Data augmentation

Since the data volume per class in the stressors dataset is limited and a convolution neural network (CNN) is a data hungry model, we use augmentation. This is a very common approach for increasing the size of the training sets for CNN (Liu, et al. 2017). With their described techniques (Liu, et al. 2017), the following data augmentation procedure was applied:

- Shift a spectrum left or right a few (1-5) wavenumbers randomly. Inserted wavenumbers got a default 0 intensity value, wavenumbers that were shifted 'out of the dataframe' were dropped.
- Add random noise to a spectrum, proportional to the magnitude at each wavenumber. The intensity at each wavenumber of the spectrum was multiplied with the same random value from a uniform distribution between 0 and 1. This can be seen as a vertical shift.
- Create linear combinations of spectra belonging to the same treatment and duration. The coefficients in the linear combination were chosen at random from a uniform distribution between 0 and 1, followed by a scaling of the coefficients so that their sum becomes 1.

For each label class and replicate, 2-5 spectra were randomly sampled from the original data and a random technique (shift, random noise or linear combination) was applied to it resulting in one or more new spectra.

This way the network will never see the same input twice and has more samples to learn from in order to generalize better. However, the inputs it sees are still heavily intercorrelated, because they come from a small number of original inputs. It is not completely new information, only a remix of existing information. As such, this may not be enough to completely get rid of overfitting (Chollet 2018).

2.2.2.5.2 Transfer learning

Another way consists of using an existing model that was trained on more data (e.g. the pathogens data) and re-use the features it has learned to extract for our new model. Only the last classification layers need to be trained for the new data. This is known as transfer learning and it avoids learning from scratch because knowledge can be re-used. Besides that, it is computationally less demanding since only a fraction of the weights needs training.

One can distinct two types of transfer learning (Chollet 2018):

- feature extraction which consists of taking the convolutional base – the series of pooling and convolution layers – of a previously trained network, running the new data through it and training a new classifier on top of the output
- fine-tuning a pretrained network where one keeps only the first few (more generic) layers of the convolution base to do feature extraction. The ones higher up in the convolutional base are trained together with the classification layers e.g. if a new dataset differs a lot from the dataset on which the original model was trained.

Note that we will also assess the generalization of deep learning models. In this case the architecture and tuned hyperparameters are also re-used but the whole model is retrained with different data. In this case the model starts again from scratch and contains no previous knowledge on feature extraction.

2.3 Evaluation of model performance

After model building, the model performance needs to be evaluated. Since the performance can differ somewhat at every run, we need to assess the variability. This is done via stratified cross validation (Figure 2-8): each dataset is split in a train, validation and test set multiple times and the stratification ensures that each class is (approximately) equally represented across each fold. Furthermore, it makes sure that all data is used during testing and also during training. This results in a population of performance measures for which confidence intervals can be calculated.

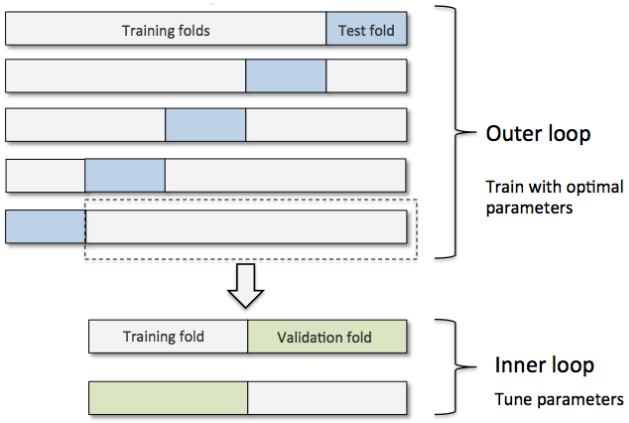

Figure 2-8: scheme of cross validation procedure: each dataset is split in a train, validation and test set multiple times and the stratification ensures that each class is (approximately) equally represented across each fold. It makes sure that all data is used during testing and also during training. (image source: https://sebastianraschka.com/faq/docs/evaluate-a-model.html)

For the conventional techniques, the inner split was done in 10 folds. These folds are used to train models, tune their parameters and do model selection via cross validation. This results in a best estimator for each corresponding outer split. The outer split consists of 5 folds with test data. The performance of the 5 best estimators on the test data will be reported via the calculation of the mean and 95% confidence intervals. Assuming that the performance measures are a random sample $X_1 \ldots X_5$ from a normal population with unknown mean μ and standard deviation σ, the exact distribution is a Student's t distribution with n−1 degrees of freedom. Because of unknown standard deviation, the sample standard deviation S is used:

$$T = \frac{\bar{X} - \mu}{S/\sqrt{5}} \sim T(5-1)$$

Thus the 95% confidence interval for the sample mean μ is:

$$P\left\{\bar{X} - t_{\frac{\alpha}{2},4} * \frac{S}{\sqrt{5}} \leq \mu \leq \bar{X} + t_{\frac{\alpha}{2},4} * \frac{S}{\sqrt{5}}\right\} = 0.95$$

Due to computational time and for comparison reasons, the implementation for deep learning models was done in a modified way. Ho, et al. (2019) followed a stratified 5-fold cross validation procedure. So for each fold, one fifth of the data was held out as unseen test data, the remaining data was split into 90/10 train and validation splits. The same strategy was used in this thesis; it is similar as for the conventional techniques except that the inner cross validation is not used here, only one train/validation split is done.

To evaluate the performance, classification accuracy will be used. It is the ratio of the number of correct predictions to the total number of inputs and works well on balanced datasets, which is the case here.

3 Results & discussion

In this chapter we will put the theory of Chapter 2 into practice. More insight will be given into how the theory was translated into implemented models. First, some data visualization will be done to get a better understanding of the data and check on the separability of the classes. Next, the results of the conventional techniques will be discussed as well as a performance breakdown at the class level for the best classification method per dataset. This section will also define the baseline performance, which we will try to improve via deep learning methods. For these methods, more details on model building, architectures and tuning of the hyperparameters via Bayesian optimization are given. The chapter ends with the results concerning the deep learning models, how well our models are able to generalize and to exchange knowledge via transfer learning.

3.1 Data visualisation

In order to get a first impression of the separability of the classes, t-distributed stochastic neighbour embedding (t-SNE) plots were made. For the stressors dataset one can see clear clusters of spectra according to the applied treatment e.g. at 5 min time point (Figure 3-1). We also notice sub-clusters per treatment, depending on the replicate. As previously described by García-Timermans, Rubbens and Kerckhof, et al. (2018) Raman spectra are sensitive to sample preparation and collection, which might explain these findings.

The control groups reveal a mixture of classes although the plot shows a separation between the early and later time points (Figure 3-2). This is in line with the findings by García-Timermans, Rubbens and Heyse, et al. (2019) that indicate that differences in growth stage are reflected in Raman data.

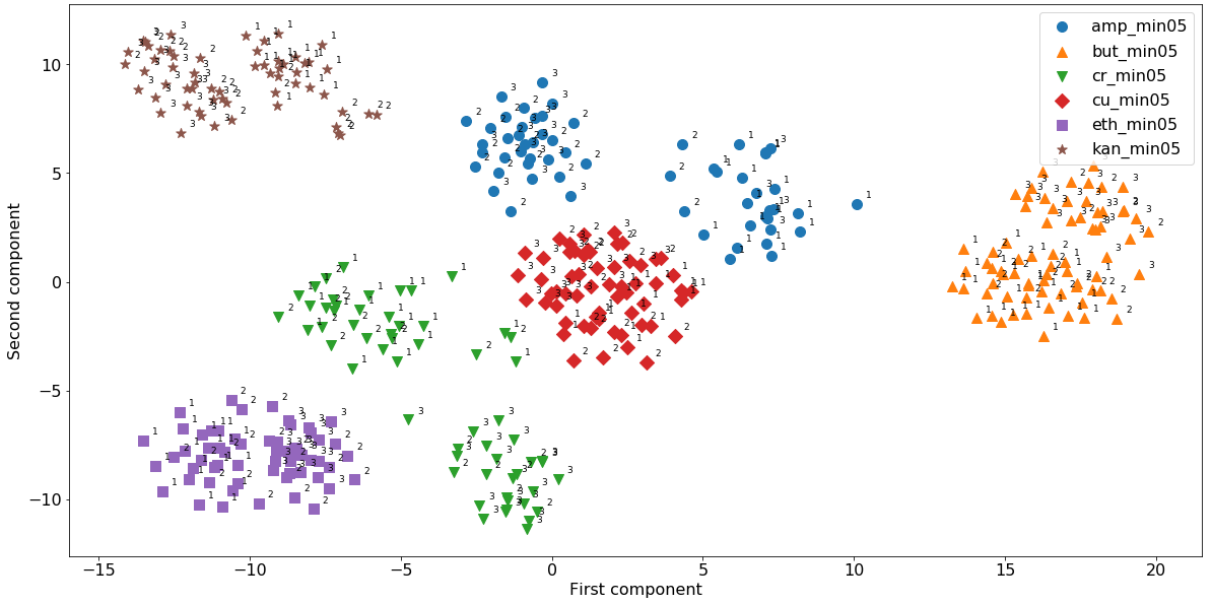

Figure 3-1: T-SNE plot of a subset of the stressors dataset, showing clear clusters for the Raman spectra with different treatments at 5 min time point. Note the sub-clusters per treatment, depending on the replicate.

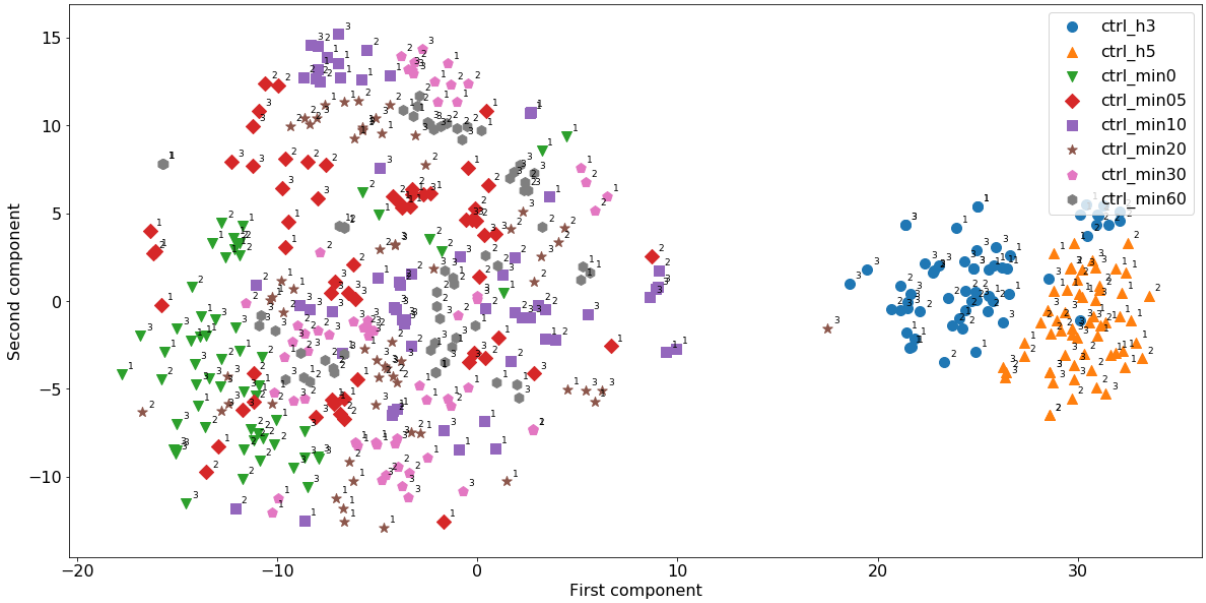

Figure 3-2: T-SNE plot of subset of stressors dataset concerning the control Raman spectra at different time points. The plot shows mixed groups of classes but has a separation between the early and later time points reflecting differences in growth stage.

For the pathogens dataset, the plot shows clear clusters as well as mixed groups e.g. for the Raman spectra of pathogens sensitive to the Vancomycin antibiotic (Figure 3-3). Methicillin-susceptible isolates of Staphylococcus aureus (MSSA) pathogens are clearly mixed while methicillin-resistant isolates of Staphylococcus aureus (MRSA) pathogens show separate clustered groups.

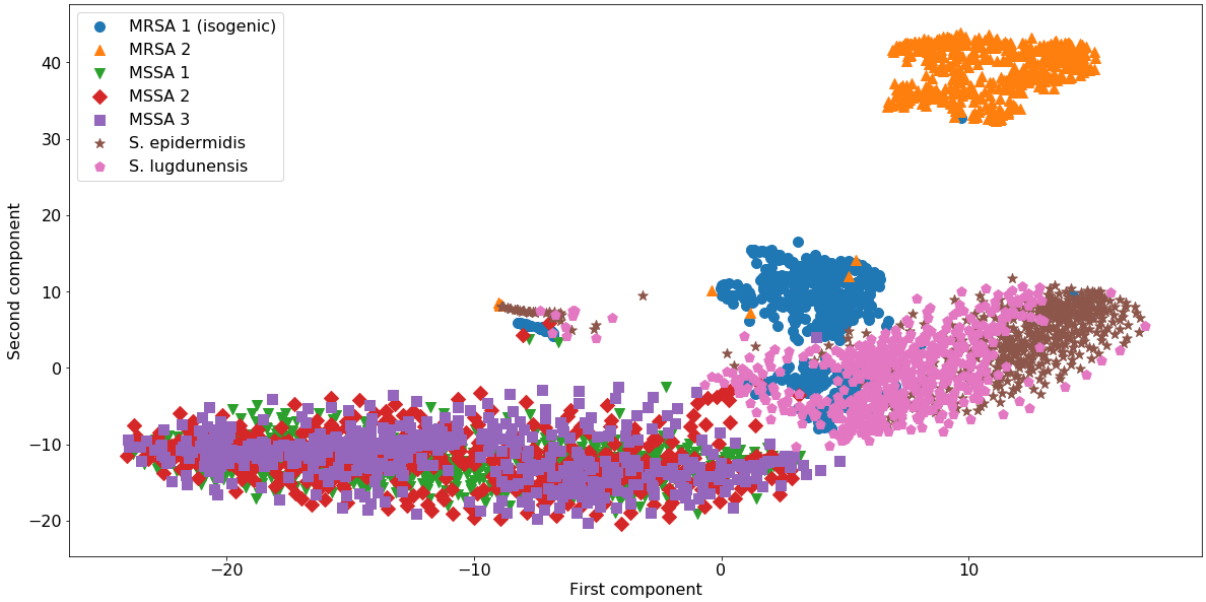

Figure 3-3: T-SNE plot of subset of pathogens dataset, showing clear clusters as well as mixed groups for the Raman spectra of pathogens sensitive to the Vancomycin antibiotic.

3.2 Baseline performance

In order to determine the baseline performance, we used the conventional algorithms as discussed earlier and applied a stratified nested cross validation with 10 inner folds and 5 outer folds (see also Chapters 2.2.1 and 2.3). For each conventional method, a pipeline was created in order to make sure that each fold is properly scaled, gets a possible data reduction and finally the algorithm is implemented. To tune the hyperparameters of the pipeline, a cross validated random search was done on an algorithm-specific hyperparameter grid.

Regarding data reduction, four strategies were tested:

1) no PCA,
2) PCA that explains 90% of the variance – a diminished value since Liu, et al. (2017) used 99.9% which is considered too high even for such uncorrelated data – ,
3) PCA where the explained variance is also a hyperparameter,
4) PCA with 20 retained components combined with SVM for comparison reasons with (Ho, et al. 2019).

Although comparison is difficult because of the use of different data, the impact of baseline corrections and performance evaluation (via leave-one-out and 50 independent runs), we list accuracies obtained by Liu, et al. (2017) – using Raman spectra of minerals – as a reference:

- KNN with 1 neighbour 77.9 ± 1.1 %
- RFs 64.5 ± 0.7 %
- Linear SVM 81.9 ± 0.4 %

More importantly Liu, et al. (2017) mentions that non-linear SVMs with a radial basis function kernel are largely used but their paper indicates higher accuracies for linear SVM. This was confirmed with our own results and thus the reason why linear SVM is applied here. Besides that, a linear SVM allowed for an alternative implementation (in scikit-learn) that scaled better to the large sample number of the pathogens dataset.

Table 3-1 summarizes the results for the conventional techniques. KNN performs worst on both datasets, although appropriate tuning – e.g. the number of neighbours – results in higher accuracies. SVM in combination with PCA retaining 90% of the variance gives the best results for the stressor dataset while RFs with PCA tuning works best for the pathogens.

Table 3-1:Summary for conventional techniques evaluated via cross validation on 5 outer folds and 10 inner folds on mean accuracy with indication of 95% confidence intervals (CI)

		Stressors			Pathogens		
		Mean	CI		Mean	CI	
KNN	Without PCA	50,24%	47,56%	52,92%	30,36%	29,97%	30,76%
	With PCA	54,61%	53,32%	55,90%	37,85%	37,11%	38,59%
	With PCA tuning	83,28%	80,50%	86,05%	82,23%	76,69%	87,78%
RFs	Without PCA	75,89%	73,50%	78,28%	74,12%	73,14%	75,09%
	With PCA	88,54%	87,83%	89,26%	86,45%	85,46%	87,44%
	With PCA tuning	88,50%	86,42%	90,58%	**88,93%**	**88,43%**	**89,42%**
SVM	Without PCA	94,32%	93,72%	94,93%	80,31%	79,97%	80,66%
	With PCA	**94,62%**	**93,57%**	**95,66%**	79,75%	79,31%	80,19%
	With PCA tuning	94,02%	93,00%	95,05%	79,79%	79,40%	80,19%
	With PCA 20	77,16%	75,92%	78,40%	69,74%	69,20%	70,29%

Note that the result of SVM with 20 retained PCA components on the pathogens data is far worse than the result of 88.7 ± 0.2 % reported by Ho, et al. (2019). Although a similar result is obtained via RFs with PCA tuning. Besides, due to high sample amount, another algorithm was used for SVM on the pathogen data: LinearSVC instead of SVC. The difference is that the first implements liblinear which is O(n) while SVC implements libsvm which is $O(n^2)$ or $O(n^3)$ with n the sample number. Despite the gain in speed, it performs less good in this case than RFs.

Applying PCA plays no role in the performance of SVM but severely limiting PCA components to 20 has a negative effect on both datasets. Too much information is removed, preventing good classifications rates.

KNN and RFs do benefit when PCA is applied on both datasets, even more when the retained variance is tuned. In all cases, except for RFs on the stressors dataset, this resulted in a much lower retained variance. For KNN on the stressors and pathogens dataset resp. 50 – 60 % and 57.5% - 65% was retained. For RFs on the pathogens dataset the retained variance was in range 57.5% - 75% while for the stressors we saw both a lower and higher retained variance (60% - 97.5%).

According to Alpaydin (2014) no gain from PCA is expected when the dimensions are not correlated. Ho, et al. (2019) points out that it can be of use to suppress the noise in the data. Given the results and the fact that Raman spectra are not expected to be much correlated, we can conclude that KNN and RFs are more sensitive to noisy spectra where SVM still manages to obtain good accuracies. Although the case with 20 retained components shows the balance between removing too much information and ameliorating a classification by removing noise.

In the next chapter, we will try to beat the upper confidence level of the best mean accuracy results: resp. 96 % for the stressors and 90% for the pathogens. But before that, we will investigate what type of mistakes our best models make on each dataset via confusion matrices.

For the stressors, already a good classification was done via SVM with 90% retained variance in PCA. Figure 3-4 shows that all errors are made on the control groups and predictions are spread over all treatment groups.

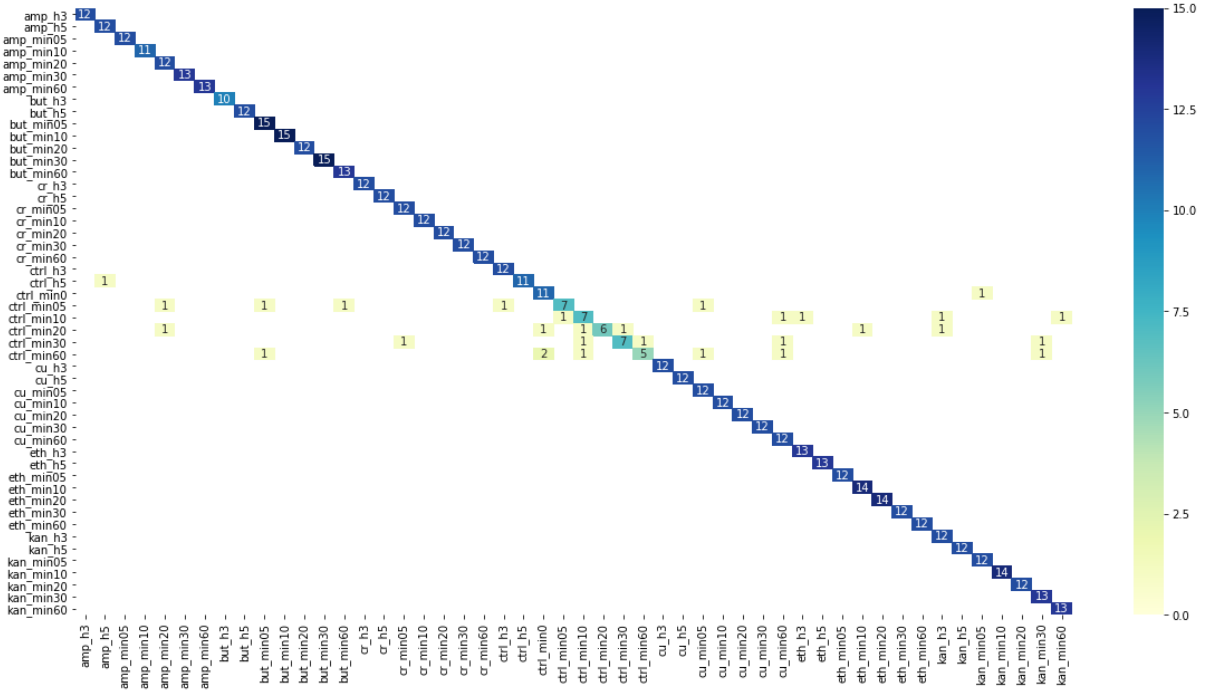

Figure 3-4: Confusion matrix for one test fold of stressor data on which SVM (with 90% variance retained via PCA) was applied. All errors are made on the controls groups and predictions are spread over all treatment groups.

The confusion matrix for Random Forest with PCA tuning on the pathogens data (Figure 3-5) shows that errors are more spread over the different classes. Note that our initial t-SNE plot already indicated less separability for MSSA and we see here misclassifications to several classes. Besides, recall that for human pathogens predicting the right antibiotic is of primary importance. Although having the best classification rate, there is certainly room for improvement on antibiotic grouping.

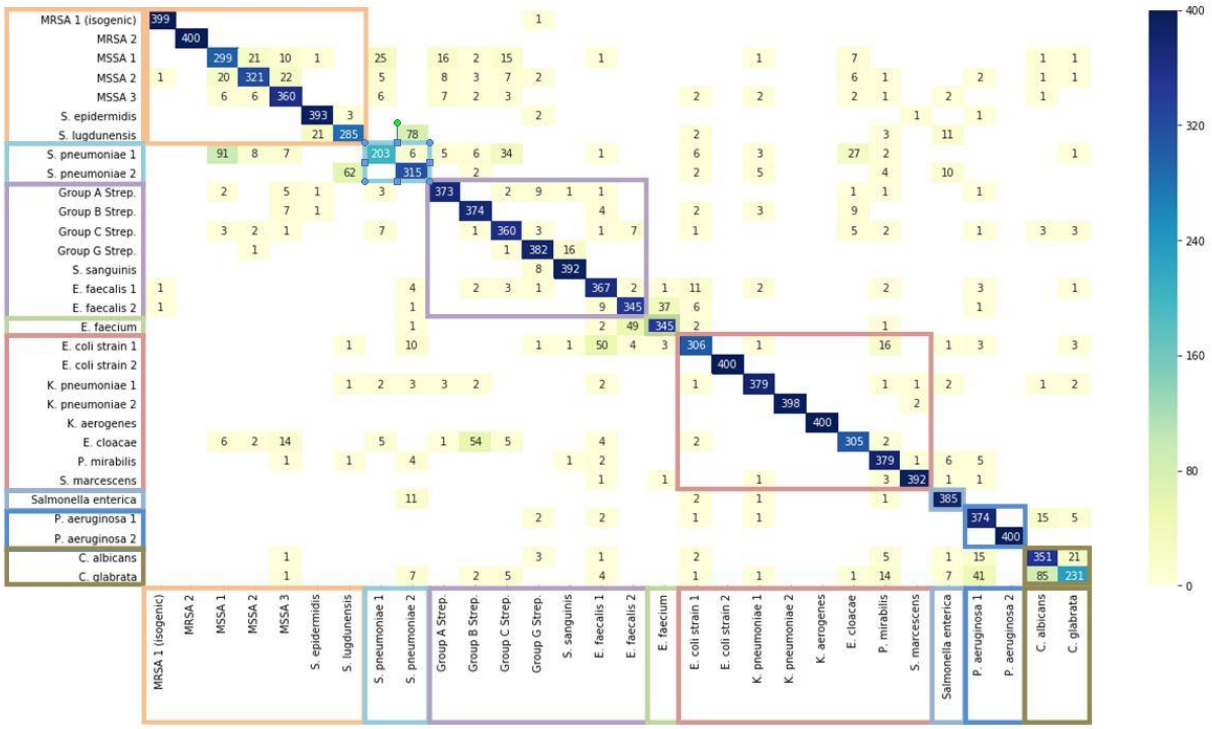

Figure 3-5: Confusion matrix for one test fold of pathogen data on which RFs with PCA tuning was applied. Colored boxes group pathogens which are sensitive to the same antibiotic. Errors are spread over the different classes and a lot are predicted in the wrong antibiotic group. Note that the right antibiotic group is of primary importance for treatment.

3.3 Deep learning

3.3.1 Model building

To start with deep learning, models were created consisting of different layers. A convolutional layer extracts patches from its input feature map and applies the same transformation to all of these patches, producing an output feature map. The output depth is a parameter of the layer (number of filters) as well as the kernel size and strides. Having more filters allows the network to learn more complex representations, but it makes the network more computationally expensive and may lead to overfitting. Strides can be used to down-sample the layer.

In LeNet architectures, convolutional layers are alternated with max pooling layers. These consist of extracting windows from the input feature maps and outputting the maximum value of each filter. Pooling is most frequently used as a way to down-sample. It reduces the number of feature-map coefficients to process, as well as to induce spatial-filter hierarchies by making successive convolution layers look at increasingly large windows (in terms of the fraction of the original input they cover) (Chollet, 2018).

Furthermore, models can include activation layers (e.g. LeakyReLU), batch normalization layers, dropout layers and dense layers; as discussed in Chapter 2.2.2.3.

Via the model architecture (see further) we decide on the amount and combination of layer types as well as on each layer settings (hyperparameters). To make the model ready for training, we need to pick three more things as part of the compilation step (Chollet 2018):

- A loss (or objective) function will measure how far the output is from what you expected. It takes the predictions of the network and the true target to compute a distance score. For multiclass classification, typically **categorical cross entropy** is used as the feedback signal, which the training phase will attempt to minimize.
- An optimizer will adjust the value of the weights at each optimization step. This is done via minibatch stochastic gradient descent in a direction that will lower the loss score for the current example. This is the implementation of the Backpropagation algorithm. Depending on the chosen optimizer, there are exact rules governing a specific use of gradient descent. Online, there are many examples using Adam or RMSprop optimizer. We follow Liu, et al. (2017) and Ho, et al. (2019) in using **Adam**.

- Metrics to monitor during training and testing, in our case **accuracy,** which is the fraction of spectra that were correctly classified.

Finally, the learning process consists of fitting the input data (and the corresponding target data) to the model and keep an eye on the validation loss as a measure of generalization. After some training, a CNN will typically start overfitting: it learns too much detail so training accuracy will still get better but on unseen data generalization will be less good. So it is more efficient to stop training when the validation loss is no longer improving for a fixed number of epochs. This can be achieved using 'Early Stopping'.

As a data preparation step, a feature-wise normalization was done so that the features are centred around 0 and have unit standard deviation. Note that the quantities used for normalizing the data sets (train, validation, test) were computed using the training data. We observed a better performance when normalizing input tensors, although both the stressors and pathogens dataset contain already values between 0 and 1 with the same unit (wavelength). This is a setting where such normalization is not necessarily needed, but we experienced that it helped in obtaining faster training times.

3.3.2 Architectures

Different architectures were tried out on both datasets: the LeNet and ResNet proposed for Raman data (see Chapter 2.2.2.3) after which we continued with some custom created models, inspired by the LeNet architecture. Starting from a single block with a final classification block, the model was gradually extended with additional blocks until a total of 3 blocks (Figure 3-6) leading to 3 models.

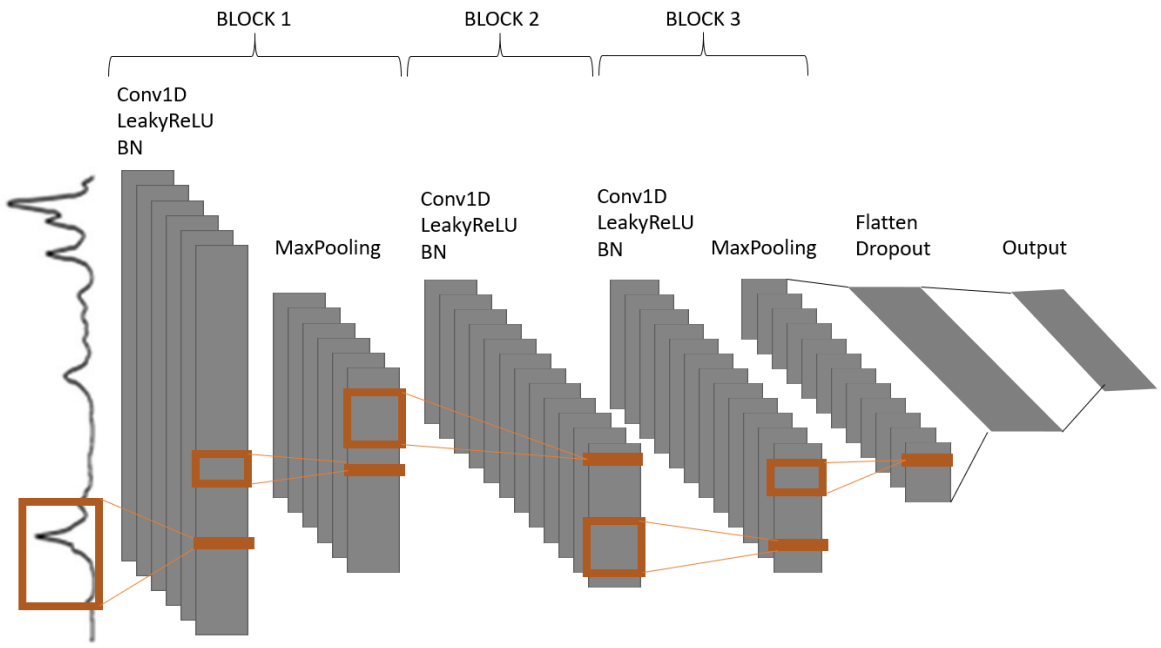

Figure 3-6: Architecture of custom models, inspired on LeNet. The models were gradually extended by more 'blocks' and all end with a final classification block.

For all architectures, training was done with the following settings:

- Training for 500 epochs
- Early stopping having a patience of 10 epochs
- Batch size of 50 (10 for ResNet, according to paper by Ho, et al. (2019))
- Adam optimizer learning rate = 0.001, beta1 = 0.9 and beta = 0.999 (beta1 = 0.5 for ResNet, according to paper by Ho, et al. (2019)

3.3.3 Bayesian optimization

In order to try to avoid overfitting and given time constraints, evaluation rounds for hyperparameter selection during this thesis were limited to 50 (stressors dataset) and 30 (pathogens dataset). The selection was done for both datasets using 1 set of training and validation data (i.e. 4/5 of the data of which 10% was used for validation). Training was done with the following settings:

- Training for 500 epochs
- Early stopping having a patience of 10 epochs
- Batch size of 50

This resulted in the hyperparameters for the custom models as listed in Table 3-2. Note that in half of the custom models, some pooling layers have a size of 1 which de facto means that no pooling takes place. Recall that Dumoulin and Visin (2016) learn that not using pooling helps in preserving the exact locations of spectral peaks.

Table 3-2: Hyperparameters for custom models tuned on the stressors and pathogens dataset

	Stressors			Pathogens		
	Custom: 1 block	**Custom: 2 block**	**Custom: 3 block**	**Custom: 1 block**	**Custom: 2 block**	**Custom: 3 block**
Convolutional1D	(32, 3, 3, 'same')	(16, 6, 3, 'same')	(4, 5, 5, 'same')	(2, 7, 1,'same')	(16, 5, 5, 'same')	(16, 6, 2, 'same')
LeakRelu	0,579706967	0,294975728	0,830237091	0,616675436	0,590875268	0,865823865
BN						
MaxPooling1D	(2, 2, 'same')	(1, 1, 'same')	(1, 1, 'same')	(6, 6, 'same')	(5, 5, 'same')	(2, 2, 'same')
Convolutional1D	\	(64, 7, 3, 'same')	(8, 3, 5, 'same')	\	(8, 4, 3, 'same')	(128, 7, 5, 'same')
LeakRelu	\	0,155238777	0,231368408	\	0,499727935	0,60372752
BN	\			\		
Convolutional1D	\	\	(8, 5, 1, 'same')	\	\	(8, 7, 1, 'same')
LeakRelu	\	\	0,211641192	\	\	0,223726302
BN	\	\		\	\	
MaxPooling1D	\	\	(2, 2, 'same')	\	\	(1, 1, 'same')
Flatten						
Dropout	\	0,722444001	\	\	0,290669996	\
Dense classification						

Besides the original hyperparameters, the model by Liu was tuned on the pathogen data in order to check on a possible improvement. A comparison between original and tuned parameters is shown in Table 3-3. Notice that the kernel sizes of the first two convolutional layers in the original model are big in comparison to the kernel sizes in the tuned version, whereas the pooling sizes are bigger, except for the second block where no pooling is done in the tuned model.

Table 3-3: A comparison between original and tuned hyperparameters on the pathogens data, using the architecture proposed by (Liu, et al. 2017).

	Liu	**Liu - Tuned**
Convolutional1D	(16, 21, 1, 'same')	(16, 6, 1, 'same')
BN		
LeakRelu	0,3	0,78365159
MaxPooling1D	(2, 2, 'same')	(7, 7, 'same')
Convolutional1D	(32, 11, 1, 'same')	(2, 6, 1, 'same')
BN		
LeakRelu	0,3	0,215979069
MaxPooling1D	(2, 2, 'same')	(1, 1, 'same')
Convolutional1D	(64, 5, 1, 'same')	(8, 7, 1, 'same')
BN		
LeakRelu	0,3	0,3984043
MaxPooling1D	(2, 2, 'same')	(4, 4, 'same')
Flatten		
Dense	(2048)	(256)
BN		
Activation (tanh)		
Dropout	0,5	0,453160418
Dense classification		

3.3.4 Model performance

As described in Chapter 2.3, a stratified 5-fold cross validation procedure was used. For each fold, one fifth of the data was hold out as unseen test data, the remaining data was split into a 90/10 train and validation set.

Our 3 custom models and 2 models from literature were evaluated on the pathogens dataset. The architecture proposed by Liu, et al. (2017) was also tuned specifically for this dataset. For the ResNet this was not needed since Ho, et al. (2019) worked with the same dataset.

As a reference and although comparison is difficult because of the use of different data, the impact of baseline corrections and performance evaluation (via leave-one-out and 50 independent runs), the accuracy obtained by Liu, et al. (2017) via CNN was 88.4 ± 0.05 %.

Looking at the results (Table 3-4), our custom model with only 1 convolutional block gave the best cross-validated results on the test data. In addition, all models perform better than the best obtained result of 90% (upper confidence boundary on mean accuracy) via RFs and PCA tuning. The model described by Liu, et al. (2017) performed better after tuning the hyperparameters on the pathogen data via Bayesian optimization. The ResNet model (94.45% accuracy) performed a little better than described in Ho, et al. (2019) where 93.8 ± 0.1% average accuracy was obtained. This difference can be due to different factors: different splits of the data, another random state for initializing the internal random number generator which defines the data splits, different initialization weights for kernel and bias on the convolutional layers, different compositions of the mini batches during training and thus different adjustment of the weights via the optimizer.

Table 3-4: Summary for deep learning models on Pathogen dataset, evaluated via cross validation on 5 outer folds with indication of 95% confidence intervals (CI)

	Pathogens		
	Mean	**CI**	
Custom: 1 block	**95,22%**	**94,90%**	**95,53%**
Custom: 2 block	90,38%	90,09%	90,68%
Custom: 3 block	94,53%	94,22%	94,85%
LeNet by Liu	91,88%	91,12%	92,63%
LeNet by Liu - Tuned	94,38%	94,06%	94,70%
ResNet by Ho	94,45%	94,20%	94,69%

Again we will investigate what type of mistakes our best model makes on each dataset via confusion matrices, in this case for the custom model with 1 block (Figure 3-7). Compared to the previous confusion matrix for the conventional techniques, the error types are similar except for less amounts resulting in a higher accuracy scoring. Still errors are spread over the different classes and are predicted in the wrong antibiotic group. Keep in mind that the right antibiotic group is of primary importance for treatment.

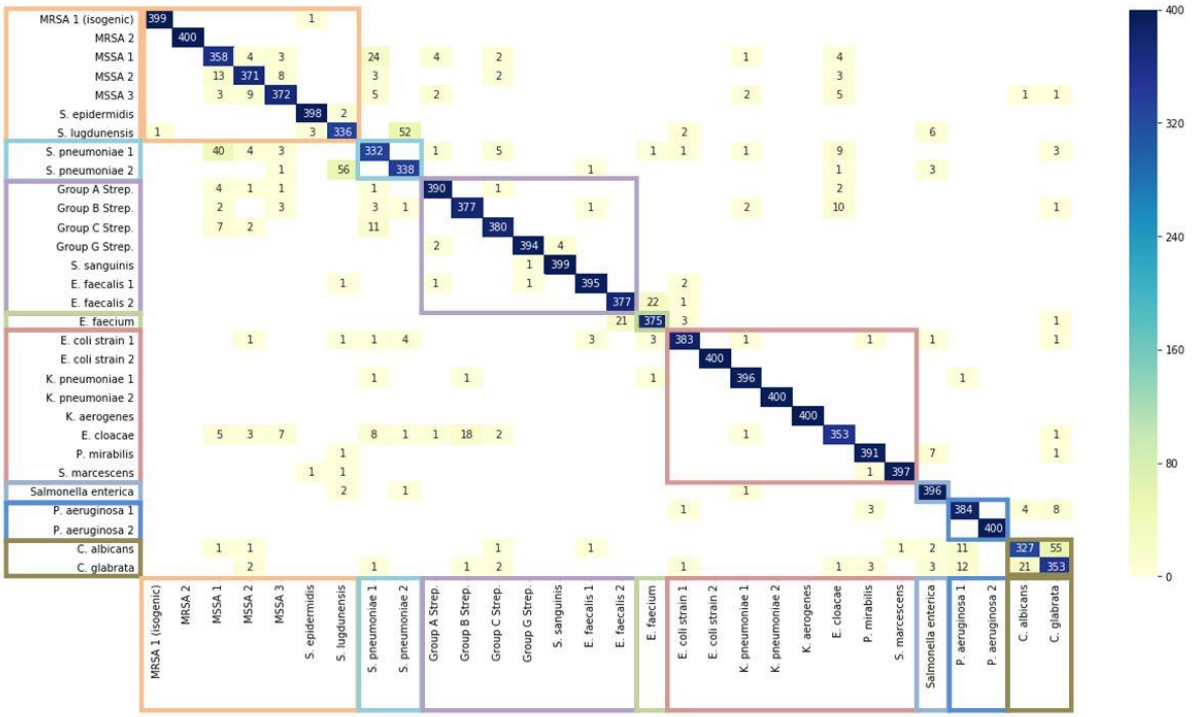

Figure 3-7: Confusion matrix for one test fold of pathogen data with the custom 1 block model. Colored boxes group pathogens which are sensitive to the same antibiotic. Error types are similar to the ones in the earlier confusion matrix for the conventional techniques except for lesser amounts resulting in a higher accuracy scoring. Still errors are spread over the different classes and are predicted in the wrong antibiotic group. Although the right antibiotic group is of primary importance for treatment.

For comparison with Ho, et al. (2019), also a confusion matrix is given for the ResNet model (Figure 3-8). Unfortunately, we see the same error pattern as for the custom 1 block model, so the statement by Ho, et al. (2019) in the sense that misclassifications are mostly within antibiotic groupings could not be reproduced. Again we can point out that this difference can be due to different factors: different splits of the data, another random state for initializing the internal random number generator that defines the data splits, different initialization weights for kernel and bias on the convolutional layers, different compositions of the mini batches during training and thus different adjustment of the weights via the optimizer.

The error on antibiotic grouping could be remediated if the network is trained for classifying both on pathogen and antibiotic grouping. This was not the case here nor was it proposed by Ho, et al. (2019).

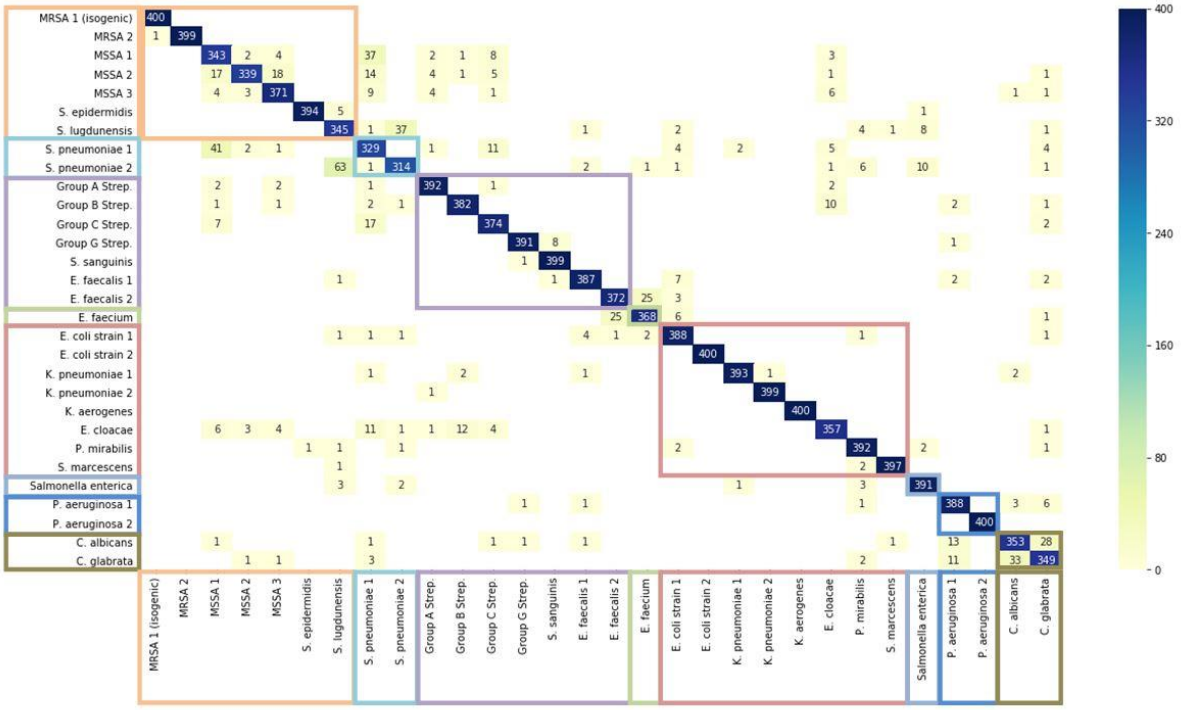

Figure 3-8: Confusion matrix for one test fold of pathogen data with the ResNet model. Colored boxes group pathogens which are sensitive to the same antibiotic. Error types are similar to the ones in the earlier confusion matrix for the conventional techniques except for lesser amounts resulting in a higher accuracy scoring. Still errors are spread over the different classes and are predicted in the wrong antibiotic group. Although the right antibiotic group is of primary importance for treatment.

Regarding the stressors dataset, we first applied data augmentation on the training data since the data volume per class in the stressors dataset is so limited that there was very little data to train on after splitting off the test and validation data. Because the augmentation is time consuming, offline augmentation was chosen and precalculated, stratified train, validation and test folds were made.

Earlier, the controls were subsampled to make balanced classes, and so, we did not use all control spectra. As part of the augmentation, the spectra that were not retained are now added again to the augmented train sets (which was then further augmented to the number of desired spectra as described earlier).

To decide on how much augmented samples were needed, we tested with an augmentation to 400 (almost no augmentation for controls groups), 500 and 1000 spectra in total per class. In

Table 3-5 one can see that our custom model with 2 convolutional blocks gives the best results for all augmentation quantities. However, when a lot of augmentation is done, results diminish and they are also slightly worse with little augmentation (except for ResNet). So further on in this thesis, the dataset with augmentation to 500 spectra per class will be used.

Table 3-5: Summary for deep learning models on Stressors dataset, evaluated via cross validation on 5 outer folds with indication of 95% confidence intervals (CI). Due to the small size of the dataset, augmentation to 400 (almost no augmentation for controls groups), 500 and 1000 spectra in total per class was used.

	Stressors 400			**Stressors 500**			**Stressors 1000**		
	Mean	**CI**		**Mean**	**CI**		**Mean**	**CI**	
Custom: 1 block	93,12%	91,86%	94,37%	94,72%	93,86%	95,58%	89,99%	88,58%	91,40%
Custom: 2 block	**96,06%**	**95,28%**	**96,84%**	**96,10%**	**95,72%**	**96,47%**	**93,17%**	**91,69%**	**94,65%**
Custom: 3 block	74,89%	71,92%	77,86%	76,27%	73,96%	78,58%	71,18%	69,50%	72,85%
LeNet by Liu	62,48%	58,39%	66,57%	65,43%	56,06%	74,80%	54,06%	46,50%	61,62%
ResNet by Ho	74,12%	69,79%	78,45%	73,46%	71,58%	75,34%	68,60%	60,56%	76,64%

Note that the custom model with 2 blocks is the only model that barely beats the baseline performance of 96% (upper confidence boundary on mean accuracy). Results are bad for the more complex models like LeNet by Liu and ResNet by Ho thus no further tuning was done on these models. Even with augmentation, the stressor dataset seems to contain too less information and/or the models from literature and their hyperparameters are not fit for this data.

The success of the custom 2 block model on this dataset can be due to the fact that it is a simple model not containing any pooling. As stated by Dumoulin and Visin (2016), using strided convolutions instead of pooling layers helps preserving the exact locations of spectral peaks. Given the data contains only Raman spectra of E. coli, one can imagine that this level of detail is needed to discriminate between the stressors. When using large convolutional kernel sizes in combination with maximum pooling like LeNet by Liu, this seems detrimental to the classification rates.

The confusion matrix for the best model is shown in Figure 3-9. In contrast to the confusion matrix for the best conventional technique, mistakes are made by predicting treatments as belonging to the control group. Besides there are still misclassifications for the control group but in lesser amount. This model has thus learnt to better classify the controls but now predicts some treatments as being part of the controls.

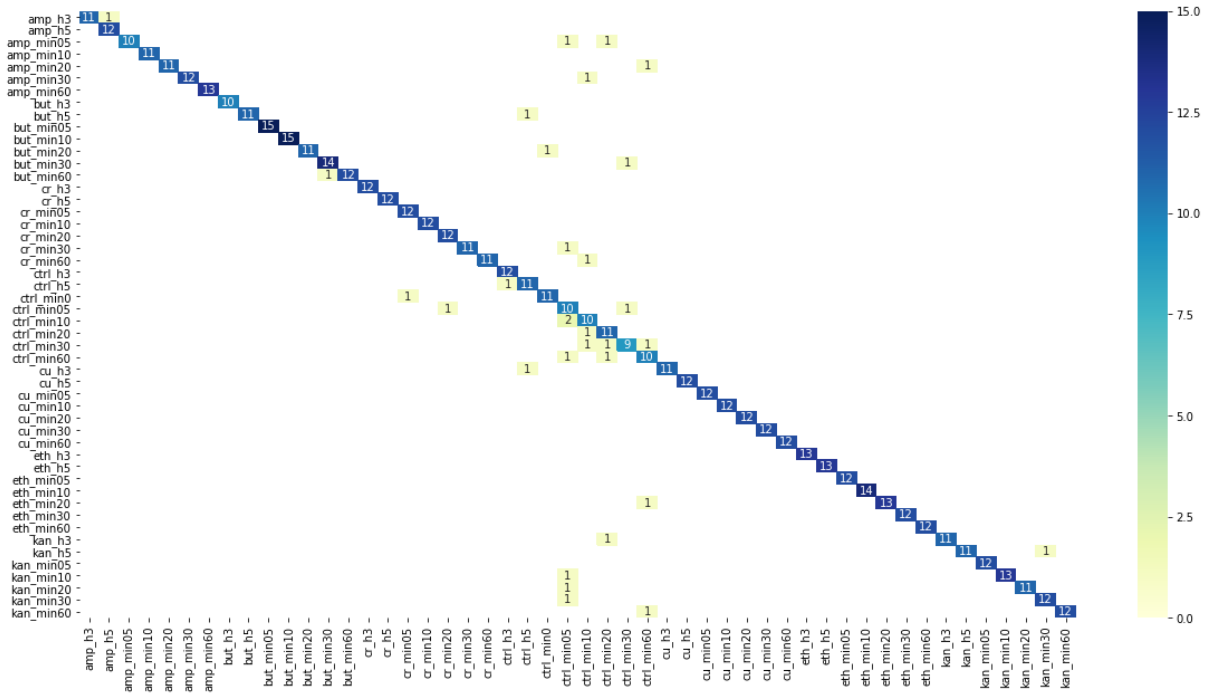

Figure 3-9: Confusion matrix for one test fold of stressor data with the custom 2 block model. Compared to the confusion matrix of the best conventional technique there are still misclassifications for the control group but in lesser amount. Instead the model now wrongly predicts some treatments as being part of the controls.

3.4 Generalization of deep learning models

The models we have trained so far perform well on one dataset. Can we reuse the architecture and tuned hyperparameters on our other dataset? We already reused the architecture and settings from Liu, et al. (2017) and Ho, et al. (2019) and obtained good – not the best – results on the pathogen data. Even a little better when we tuned the hyperparameters on the Liu model. In contrast, results were bad on the stressor dataset.

So the models for the pathogens are trained now with the stressor data (only the best set, augmented till 500 spectra per class) and vice versa. The results are listed in Table 3-6 and Table 3-7. Using the pathogens models does not give satisfying results on the stressor data, although the custom 3 block model performs best which is a model with only 1 pooling layer and a limited pool size of 2.

In contrast the best stressor model gives good results for the pathogens (second best deep learning result). As discussed in the previous section, the absence of a pooling layer helps to maintain the peak patterns, which positively affects discrimination of species.

Table 3-6: Summary for deep learning models on Pathogens dataset, evaluated on the augmented stressor dataset via cross validation on 5 outer folds with indication of 95% confidence intervals (CI)

	Pathogens			Stressors -augmented 500		
	Mean	CI		Mean	CI	
Custom: 1 block	**95,22%**	**94,90%**	**95,53%**	54,89%	49,31%	60,47%
Custom: 2 block	90,38%	90,09%	90,68%	52,91%	51,51%	54,31%
Custom: 3 block	94,53%	94,22%	94,85%	76,75%	73,46%	80,04%
LeNet by Liu	91,88%	91,12%	92,63%	65,43%	56,06%	74,80%
LeNet by Liu - Tuned	94,38%	94,06%	94,70%	58,71%	57,15%	60,27%
ResNet by Ho	94,45%	94,20%	94,69%	73,46%	71,58%	75,34%

Table 3-7: Summary for deep learning models on Stressors dataset, evaluated on the pathogen dataset via cross validation on 5 outer folds with indication of 95% confidence intervals (CI)

	Stressors -augmented 500			Pathogens		
	Mean	CI		Mean	CI	
Custom: 1 block	94,72%	93,86%	95,58%			
Custom: 2 block	**96,10%**	**95,72%**	**96,47%**	94,88%	94,48%	95,28%
Custom: 3 block	76,27%	73,96%	78,58%			
LeNet by Liu	65,43%	56,06%	74,80%	91,88%	91,12%	92,63%
ResNet by Ho	73,46%	71,58%	75,34%	94,45%	94,20%	94,69%

3.5 Transfer learning

As discussed in Chapter 2.2.2.5, transfer learning can help re-use knowledge from networks trained on more data into networks for classifying a small dataset. This way the network does not need to be trained from scratch while in fact one has too little data available.

We will apply transfer learning and specifically feature extraction on the best models that were trained on the pathogen data: the custom 1 block model and ResNet. At the same time these represent the most simple model versus the most complex model. Despite different pre-processing and spectral intensity variances, we believe that for both datasets a deep learning model should learn to recognize peak patterns that can be re-used. In that sense, we will only test feature extraction. Besides that, the Custom 1 block model is so small that it does not allow for fine-tuning.

Table 3-8 shows the results, obtained on the stressor data by re-using knowledge from the pathogen networks. The results are among the lowest throughout this thesis but may come as no surprise since the previous section indicated that the stressor dataset benefits from simple

models without pooling layers in order to maintain the peak patterns. To see the difference between all E. coli spectra and their stressor impact more detail is needed in order to discriminate between them while the pathogen data, containing a mixture of species, probably has enough with more general differences between species.

Table 3-8: Summary for transfer learning results obtained on the Stressors dataset using knowledge from the Pathogens models. Again, evaluation was done via cross validation on 5 outer folds with indication of 95% confidence intervals (CI). As a reference, the performance obtained on the pathogen data and later results for retraining the models from scratch on the augmented stressors dataset are added.

	Pathogens			**Stressors -augmented 500**			**Transfer learning**		
	Mean	**CI**		**Mean**	**CI**		**Mean**	**CI**	
Custom: 1 block	**95,22%**	**94,90%**	**95,53%**	54,89%	49,31%	60,47%	55,61%	38,97%	72,24%
Custom: 2 block	90,38%	90,09%	90,68%	52,91%	51,51%	54,31%			
Custom: 3 block	94,53%	94,22%	94,85%	76,75%	73,46%	80,04%			
LeNet by Liu	91,88%	91,12%	92,63%	65,43%	56,06%	74,80%			
LeNet by Liu - Tuned	94,38%	94,06%	94,70%	58,71%	57,15%	60,27%			
ResNet by Ho	94,45%	94,20%	94,69%	73,46%	71,58%	75,34%	39,98%	32,67%	47,30%

4 Conclusion and further research

The subject of this thesis was a data-driven identification of microbial Raman spectra, called "Ramanome". This unique biochemical Raman fingerprint at the single cell level represents a phenotypic profile of a live cell. Although the technique has a history starting in the '70s, the full potential of the technique could only be explored by using powerful statistical and computational methods. This led to the use of machine learning methods. Data is crucial here and a data analysis pipeline needs to be used: pre-treatment to correct non-sample dependent artefacts, pre-processing consisting of smoothing, baseline correction, normalization and data reduction followed by a final modelling step.

From literature, the impact of baseline correction on the performance of machine learning methods is known as well that a wide range of baseline correction methods are used. However, this step can be avoided by applying Convolutional Neural Networks (CNN) since they combine pre-processing, feature extraction and classification in a single architecture. Therefore, this deep learning architecture is the main focus of this thesis.

First, conventional machine learning techniques were applied and assessed on two datasets: one measured stress responses on single cells of E. coli and the other dataset contained Raman spectra of human pathogens. An SVM in combination with PCA gave the best mean classification accuracy for the stressor dataset, while RFs with PCA tuning worked best for the pathogens. These results form the baseline performance, a reference to which further results were compared.

Continuing with CNNs we found that our custom models – based on the LeNet architecture and tuned via Bayesian Optimization – performed better than the LeNet and ResNet models described in literature for Raman spectra classification. Regarding the pathogens dataset, all models outperformed the baseline performance of 90% and our custom model obtained the best mean accuracy of 95.2%.
For the stressor data, our custom model was the only one beating the baseline performance of 96% but only with a slight difference: 96.1%. Results were bad for the more complex models such as the ones found in literature. It became clear that this dataset was better classified with a simple CNN without pooling. By using strided convolutions instead of pooling layers, the

exact locations of spectral peaks can be preserved. A possible explanation can be that the stressor data includes only Raman spectra of E. coli and thus more detail is needed in order to discriminate between them while the pathogen data, containing a mixture of species, probably has enough with more general differences between species.

One major remark must be given on the obtained results, here and in literature. Reproducibility is an issue, even with the same model and data. This difference can be due to different factors: different splits of the data, another random state for initializing the internal random number generator that defines the data splits, different initialization weights for kernel and bias on the convolutional layers, different compositions of the mini batches during training and thus different adjustment of the weights via the optimizer,…

Furthermore, the generalization of our models was assessed by selecting the best models on one dataset and retrain them from scratch on the other dataset. The best pathogens models did not give satisfying results on the stressor data. In contrast, the best stressor model gave good results for the pathogens (second best deep learning result). Again, the presence or absence of pooling had its effect on maintaining the peak patterns which affected the accuracy.

Finally, given the limited size of the stressors dataset, transfer learning and specifically feature extraction were applied with the best pathogen models. The results are among the lowest throughout this thesis, but may come as no surprise, since the stressor dataset benefits from simple models without pooling layers. And these were not the type of models that performed best on the pathogen data. With more data for the stressors and given the good generalization to the pathogen dataset, the reverse experiment would be interesting.

Concluding this thesis, I propose some ideas for further research:

- More detailed study on the impact of amount of filter maps, kernel size, strides,… on classification results
- Explore which features were learned from the Raman spectra for each model and asses their impact on obtained accuracy.
- Study transfer learning on bigger and more datasets. Is it advantageous to learn in detail from spectra on one species (like the stressors dataset) and is the knowledge transferable to other species (e.g. pathogens)?

5 Reference list

Alpaydin, E. (2014). *Introduction to machine learning.*

Bergstra, J., Bardenet, R., Kégl, B., & Bengio, Y. (2011). Algorithms for Hyper-Parameter Optimization. *Advances in Neural Information Processing Systems*.

Bishop, C. (2006). *Pattern Recognition and Machine Learning* (Vol. 1). Springer.

Bocklitz, T., Dörfer, T., Heinke, R., Schmitt, M., & Popp, J. (2015). Spectrometer calibration protocol for Raman spectra recorded with different excitation wavelengths. *Spectrochimica Acta Part A: Molecular and Biomolecular Spectroscopy, 149*, 544–549. doi:https://doi.org/10.1016/j.saa.2015.04.079

Chollet, F. (2018). *Deep learning with Python.*

Deng, L., & Yu, D. (2014). *Deep Learning Methods and Applications.*

Dumoulin, V., & Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv*. Retrieved from https://arxiv.org/abs/1603.07285

García-Timermans, C., Rubbens, P., Heyse, J., Kerckhof, F. M., Props, R., Skirtack, A. G., . . . Boon, N. (2019). Discriminating bacterial phenotypes at the population and single-cell level: a comparison of flow cytometry and Raman spectroscopy fingerprinting. *bioRXiv*. doi:https://doi.org/10.1101/545681

García-Timermans, C., Rubbens, P., Kerckhof, F., Buysschaert, B., Khalenkow, D., Waegeman, W., . . . Boon, N. (2018, August). Label-free Raman characterization of bacteria calls for standardized procedures. *Journal of Microbiological Methods*(151), 69-75. doi:https://doi.org/10.1016/j.mimet.2018.05.027

Guido, S., & Müller, A. (2016). *Introduction to Machine Learning with Python.*

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *arXiv*. Retrieved from https://arxiv.org/abs/1512.03385

Ho, C., Jean, N., Hogan, C., Blackmon, L., Jeffrey, S., Holodniy, M., . . . Dionne, J. (2019). Rapid identification of pathogenic bacteria using Raman spectroscopy and deep learning. *arXiv*. Retrieved from https://arxiv.org/abs/1901.07666

Huang, W., Li, M., Jarvis, R., Goodacre, R., & Banwart, S. (2010). Shining Light on the Microbial World: The Application of Raman Microspectroscopy. *Advances in Applied Microbiology, 70*, 153-186. doi:https://doi.org/10.1016/S0065-2164(10)70005-8

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv*.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*, 436–444.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998, November). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278 - 2324. doi:https://doi.org/10.1109/5.726791

Liu, J., Osadchy, M., Ashton, L., Foster, M., Solomon, C., & Gibson, S. (2017). Deep Convolutional Neural Networks for Raman Spectrum Recognition: A Unified Solution. *Analyst, Vol. 142, No. 21*. doi:https://doi.org/10.1039/C7AN01371J

Lorenz, B., Wichmann, C., Stöckel, S., Rösch, P., & Popp, J. (2017, May). Cultivation-Free Raman Spectroscopic Investigations of Bacteria. *Trends in microbiology*(25(5)), 413-424. doi:http://10.1016/j.tim.2017.01.002

Olson, R., La Cava, W., Mustahsan, Z., Varik, A., & Moore, J. (2018). Data-driven advice for applying machine learning to bioinformatics problems. *arXiv*. Retrieved from https://arxiv.org/abs/1708.05070

Ryabchykov, O., Guo, S., & Bocklitz, T. (2018). Analyzing Raman spectroscopic data. *Physical Sciences Reviews*. doi:https://doi.org/10.1515/psr-2017-0043

Tang, B., Pan, Z., Yin, K., & Khateeb, A. (2019). Recent Advances of Deep Learning in Bioinformatics and Computational Biology. *Frontiers in genetics*. doi:https://doi.org/10.3389/fgene.2019.00214

Teng, L., Wang, X., Wang, X., Gou, H., Ren, L., Wang, T., . . . Xu, J. (2016). Label-free, rapid and quantitative phenotyping of stress response in E. coli via ramanome. *Scientific Reports*. doi:https://doi.org/10.1038/srep34359

Appendix A: Software implementation

All implementations in this thesis were done by using Python and Jupyter notebooks with following versions:

- Python 3.6
- Pandas 0.24.1
- Scikit-learn 0.20.3

For the implementation and tuning of deep learning models in this thesis were used:

- Keras version 2.2.4
- Tensorflow version 1.8.0
- Hyperopt version 0.2

Keras is a deep-learning framework for Python that provides a convenient way to define and train almost any kind of deep-learning model. It allows the same code to run seamlessly on CPU or GPU.
Keras is a model-level library, providing high-level building blocks for developing deep-learning models. It does not handle low-level operations such as tensor manipulation and differentiation. Instead, it relies on a specialized, well-optimized tensor library. Since Keras handles the problem in a modular way several different backend engines (TensorFlow, Theano, CNTK) can be plugged in seamlessly again without having to change anything in the code.

In order to optimize the hyperparameters of deep learning models Bayesian optimization was applied. This is an algorithm that uses a history of validation performance, given various sets of hyperparameters, to choose the next set of hyperparameters to evaluate. This can be done via Hyperopt, a Python library for hyperparameter optimization that internally uses trees of Parzen estimators to predict sets of hyperparameters that are likely to work well (Chollet 2018).

Addendum to Master thesis

Name student: Katja D'haeyer

Name promoter: Prof. dr. Willem Waegeman

Thesis title: Towards a data-driven identification of the microbial "Ramanome"

Date **pre-defense**: 4/4/19

Name and signature of one of the members of the Examination Board, testifying successful participation in the pre-defenses:

S. Vanderlandt

Date **seminar 1**: 1/04/2019

Title seminar 1: UGent Data Science Seminar - Prof. Sören Auer
https://ai.ugent.be/news/event/2019/04/01/10th-DSS.en.html

(Add details of the seminar on an extra page: speaker's name, title and abstract, time and location)
Signature of the promoter, testifying approval of and attendance of the seminar:

PETER RUYSSENS

Date **seminar 2**: 4/04/2019

Title seminar 2: Master of Statistical Data Analysis Day

(Add details of the seminar on an extra page: speaker's name, title and abstract, time and location)
Signature of the promoter, testifying approval of and attendance of the seminar:

Steven Vandenhaute