

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Deep Learning for Backscatter Classification

Martim Chaves

MASTER'S THESIS

Integrated Master's in Bioengineering - Biomedical Engineering

Supervisor: Jaime Cardoso

Second Supervisor: Simão Faria

July 25, 2021

Resumo

Este trabalho foi feito no âmbito da Companhia X. A Companhia X desenvolveu um aparelho baseado em fotónica que permite adquirir o sinal de retroespalhamento de amostras biológicas. Neste momento, pode ser dito que a Companhia X tem dois objetivos. O primeiro é o desenvolvimento de um modelo capaz de distinguir com exatidão amostras de indivíduos saudáveis de amostras de indivíduos que sofrem de Doença Neurodegenerativa. Isto deve-se à atual falta de suporte no ecossistema de desenvolvimento de fármacos desta doença, dado que vários ensaios clínicos não têm sido bem sucedidos ao longo dos últimos anos. O segundo objetivo é o de continuamente melhorar o método de aquisição e análise dos dados, visando uma futura contribuição para medicina de precisão. Medicina de precisão é aquela em que cada indivíduo, que necessite, tem acesso a medicina adequada para o seu organismo, que responda com eficácia tendo em conta as particularidades biológicas ao nível molecular do mesmo, garantindo um maior sucesso de tratamento. Teoricamente, o sinal de retroespalhamento interage com meios diferentes de formas diferentes, dado que a luz irá interagir com as partículas no meio e formar um sinal específico. Isto permitirá melhor compreender meios a partir do sinal de retroespalhamento.

Neste momento, a análise do sinal é feita através de aprendizagem automática, uma sub-área da inteligência artificial, em que máquinas, computadores, automaticamente desenvolvem métodos para determinar a estrutura de um tipo de dados. Recentemente, na última década, aprendizagem profunda, uma sub-área de aprendizagem automática, surgiu como uma área extremamente eficaz em diversas áreas, tendo até melhores resultados do que aprendizagem automática. A Companhia X utiliza, atualmente, métodos de aprendizagem automática para análise de dados. Consequentemente, nesta tese, houve dois objetivos principais: o desenvolvimento e exploração de métodos de aprendizagem profunda para a análise do sinal de retroespalhamento, tendo em conta o futuro da utilização da tecnologia, em medicina de precisão, e o desenvolvimento de uma *pipeline* de aprendizagem profunda para detetar sinais de indivíduos que sofram de Doença Neurodegenerativa.

Para isso, uma visão sobre os tópicos relevantes foi feita, visando a obtenção da compreensão da área, seguindo-se a apresentação da revisão da bibliografia de sinais semelhantes, pelo menos na sua estrutura, ao do retroespalhamento. Após isso, é feita uma descrição dos métodos desenvolvidos.

Finalmente, os resultados para os métodos desenvolvidos são apresentados e comparados ao melhor método de aprendizagem automática utilizado atualmente. Verificou-se que o melhor método desenvolvido tem uma performance melhor em relação a um grande número de métricas, e uma performance ligeiramente pior em relação a uma métrica, não menos relevante, contudo. Para quantificar a contribuição desta tese, calculou-se a estimativa do valor de mercado dos métodos desenvolvidos. Conclusões gerais e propostas para trabalho futuro foram feitas na conclusão.

Abstract

This work was done within the scope of Company X. Company X has developed a photonics-based device that allows for the acquisition of the backscattered signal of biological samples. Currently, it can be said that Company X has two objectives. The first one is the development of a model that can accurately distinguish between samples from healthy individuals and patients afflicted with a Neurodegenerative Disease (ND). This is due to the lack of support that can be found in the current ND drug-development ecosystem, where several drug trials fail. The second one is to continuously improve their acquisition and analysis methods, to, in the future, contribute to precision medicine - which is when medical treatments are tailored to each individual, as each individual has a different molecular fingerprint. Theoretically, different mediums will lead to different backscatter representations, as the light will interact with the mediums distinctly. Understanding the biological molecular-level composition of an individual's organism will allow for the careful tuning of administered drugs, when that is required.

Currently, the analysis of the backscattered signal is done using Machine Learning, which is a subfield of Artificial Intelligence, where machines automatically develop methods to structure data. Recently, over the last decade, a new subfield of Machine Learning (ML), Deep Learning (DL), has become very prominent, as it has allowed for state-of-the-art results in several areas. Thus, in this thesis, there were two main goals: the development and exploration of DL methods for the analysis of the backscattered signal, considering a more long term perspective, *i.e.* aiming to understand DL viability for backscatter analysis in general, and, a more short-term oriented goal, the development of a DL pipeline that can perform well on a dataset containing healthy and ND afflicted individuals.

For that, an overview of the relevant topics was made, aiming for a thorough understanding of the field. Afterward, a review of the literature for data types that were found relevant for this dissertation, *i.e.* data types that were similar to the backscatter, from, at least, a structural perspective, was carried out. Following that, a description of the methodologies that were developed for the different steps of a DL pipeline was made, where, for each step, different methods were presented.

Finally, results were obtained for the developed methods, and they were compared to the currently in use ML methods. The best model showed a better performance regarding some metrics, and a slightly lower performance regarding one metric, which is, nonetheless, not less important. In order to quantify the contribution made, the market value of the model, which, naturally, can only be applied in conjunction with Company X's photonics device, was calculated, obtaining a rough value in the order of millions of dollars, per year. General conclusions and future work proposals were made.

Acknowledgements

I would like to thank my supervisors, Professor Jaime Cardoso and Simão Faria, for their guidance, ideas, challenges, feedback, and time. This has been the largest project I have worked on yet, and I overcame several of my previous personal boundaries, further improving my skills in this area of expertise, and, ultimately, improving myself.

The Company X's Data Science team has also made very valuable, essential even, contributions to this work, through their very much appreciated feedback and ideas, so I thank you, Cristiana Carpinteiro, João Pereira, Marina Fridman, and my fellow interns, Marta Lourenço and Nuno Costa. This work is also yours.

I would also like to express my gratitude towards everyone at Company X, for their warm welcome, for allowing me to explore so many different ideas, for their great work with the technology that is at the core of this dissertation, and for providing me with resources that allowed me to carry out all of the experiments that were envisioned.

I would also like to thank two research centers, C-BER at INESC TEC, and the Insight SFI Research Centre for Data Analytics at DCU, where I first encountered this area of study, and where I developed the fundamentals in this field. There I discovered the joy of the creating and questioning cycle, met some incredibly kind people, and learned to persevere after many failed ideas.

I am also very thankful for my academic experience, for all of the like-minded individuals that I have met, trying to make the world we live in a better place. I am thankful for all of my peers, the students I worked with, that taught me, precisely because of the disagreements that would arise, that working together leads to something greater than what I could accomplish alone. Thank you also to the Professors that guided my learning, and for sharing their knowledge and enthusiasm in their fields. Thank you to everyone in the academic community. A special note to three student's groups, to BEST Porto, for allowing me to meet so many interesting students from so many different universities in Europe, to SdDUP, the University of Porto's Debating Society, for the incredible competitive experiences, and for all of the inspiring people that I met, and to CICBAS, ICBAS's choir, for being such an incredible group of people that, once in a while, likes to sing.

Lastly, on a personal note, thank you, family and friends, for everything.

Martim Chaves

*“If you want to know how much darkness there is around you,
you must sharpen your eyes, peering at the faint lights in the distance.”*

Italo Calvino

Contents

1	Introduction	1
1.1	Context	1
1.1.1	Company X	1
1.1.2	Neurodegenerative disease	2
1.1.3	Deep Learning	2
1.2	Motivation	4
1.3	Goals	5
1.4	Dissertation Structure	5
2	Background	7
2.1	Introduction	7
2.2	Particles interactions with light	7
2.2.1	Particles	7
2.2.2	Back Scattering	8
2.3	Backscatter Signal Acquisition	9
2.4	Signal Processing	10
2.4.1	Discrete-time signals: sequences	10
2.4.2	Representation of Sequences by Fourier Transforms	11
2.5	Machine Learning	12
2.5.1	Data and Structural Patterns	12
2.5.2	Supervised Learning	12
2.5.3	Unsupervised Learning	17
2.5.4	Neural Networks (NNs)	17
2.6	Deep Learning	23
2.6.1	Convolutional NNs	23
2.6.2	Residual NNs	26
2.6.3	Multi-task Learning	27
2.6.4	Self-Attention Mechanism	27
2.7	Conclusions	28
3	Literature Review	29
3.1	Introduction	29
3.2	Analysis of backscatter signals	29
3.3	DL applied to Radar and Sonar Data	31
3.4	DL applied to 1D physiological signals	32
3.4.1	ECG	32
3.4.2	EEG	34
3.4.3	EMG - often used in tandem with sensor data	37

3.5	DL applied to sound	38
3.6	Influence of the Reviewed Literature	39
3.7	Conclusions	40
4	Methodologies	41
4.1	Introduction	41
4.2	Input	41
4.3	Data Augmentation	43
4.3.1	Regular Image Transformations DA	44
4.3.2	DA Designed Specifically for Spectrograms	46
4.4	Architectures	48
4.4.1	Architectures developed for 1D input	49
4.4.2	Architectures developed for Spectrograms	52
4.4.3	Networks with mixed inputs - 1D and spectrograms	55
4.4.4	Patient Feature Networks	56
4.5	Training Processes	58
4.6	Outputs	59
4.7	Conclusion	60
5	Results and Discussion	61
5.1	Introduction	61
5.2	Dataset Split	62
5.3	Model Selection	62
5.4	Input Types	63
5.5	Data Augmentation	64
5.6	Model Architecture	67
5.7	Training Methods	69
5.8	Outputs	70
5.9	Patient Feature Networks	70
5.10	Comparison to ML methods	71
5.11	Estimate of Value Added	74
5.12	Conclusion	75
6	Conclusions	77
6.1	Overview	77
6.2	Contributions	79
6.3	Future Work	79
6.3.1	Inputs	79
6.3.2	Data Augmentation	80
6.3.3	Model Architectures	80
6.3.4	Training Processes and Outputs	80
6.3.5	Different classification tasks and feature-combining networks	80
A	Extended results	81
	References	87

List of Figures

1.1	Structure of a Neural Network.	3
2.1	Raw backscatter acquired.	9
2.2	Schema of the backscatter signal acquisition.	10
2.3	Time and spectral representation of a signal.	11
2.4	Structure of a Neural Network with 2 layers.	18
2.5	Example of the basic convolution operation with a 3x3 kernel and an image patch for the center point of the patch.	24
2.6	Representation of the intuition behind feature extraction in a CNN and the relation between low-level and high-level features.	25
2.7	Structure of a common CNN.	26
4.1	1D input.	42
4.2	Spectrogram input.	43
4.3	General Image Transformations Applied to a Spectrogram.	47
4.4	Spectrogram specific DA Applied to an Example Spectrogram.	48
4.5	Schematic of a Pre-Activation Residual Block.	53
4.6	Transformer architecture.	58
4.7	CNN plus Transformer architecture.	59
5.1	Accuracy and Loss.	72
5.2	Precision, F1-Score, Sensitivity, and Specificity.	73
5.3	ROC curve and Confusion Matrix.	74

List of Tables

3.1	DL and ML methods reviewed.	30
4.1	Large 1D CNN - ? is used to represent the batch size.	50
4.2	Small CNN for the 1D input.	50
4.3	Residual 1D CNN.	51
4.4	Large 1D CNN with Residual Blocks.	52
4.5	Main spectrogram CNN.	54
4.6	Standard spectrogram network with Residual Blocks.	54
4.7	Residual spectrogram network.	55
4.8	Multiple spectrogram network.	56
4.9	1D plus spectrogram CNN.	56
4.10	Standard CNN for feature-blocks.	57
5.1	Results for different model stages throughout training.	62
5.2	Results obtained using different types of inputs.	63
5.3	Results obtained using different general image transformations DA.	65
5.4	Results obtained using different spectrogram-specific DA.	66
5.5	Results obtained using horizontal flips DA for the 1D input (third fold).	67
5.6	Results obtained using different types of model architectures - experiments were only done for the first fold due to time constraints.	68
5.7	Results for different training methods - results that do not present standard deviation are from experiments that were only done for the first fold.	69
5.8	Results obtained for different types of classification (second dataset).	70
5.9	Results obtained using different patient feature networks. When standard deviation is not present, the results were obtained for the first fold only, due to time constraints.	71
5.10	Comparing the results obtained to the ones obtained using ML methods (SVM) - per patient metrics.	72
A.1	Results for the test set when choosing the model state according to different metrics.	81
A.2	Results for different types of spectrogram inputs per fold.	82
A.3	Results for different types of 1D inputs per fold.	82
A.4	Results for regular image transformations DA for the spectrogram input for the different folds.	83
A.5	Results for spectrogram-specific transformations DA for the spectrogram input for the different folds.	84
A.6	Results, per fold, for different output types.	85
A.7	Results, per fold, for the networks whose input was the feature-block.	85

Abbreviations and Symbols

AE	Autoencoder
ASR	Automatic Speech Recognition
AUC	Area Under the Curve
BCI	Brain Computer Interface
CNN	Convolutional Neural Network
DA	Data Augmentation
DL	Deep Learning
DPN	Deep Polynomial Network
ECG	Electrocardiogram
EEG	Electroencephalogram
EMG	Electromyogram
FN	False Negatives
FP	False Positives
FT	Fourier Transform
IND	Intermediate Neurodegenerative Disease
LSTM	Long Short Term Memory (NN)
MAP	<i>Maximum a Posteriori</i>
ML	Machine Learning
MTL	Multi Task Learning
ND	Neurodegenerative Disease
NN	Neural Network
PCA	Principal Component Analysis
ReNN	Residual Neural Network
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SG	Spectrogram
STFT	Short Term Fourier Transform
SVM	Support Vector Machine
TN	True Negatives
TP	True Positives
<i>w.r.t.</i>	With respect/regard to

Chapter 1

Introduction

1.1 Context	1
1.2 Motivation	4
1.3 Goals	5
1.4 Dissertation Structure	5

For confidentiality reasons, the name of Company X and of Neurodegenerative disease are withheld. At some points throughout this dissertation, you may find citations that, although referring to a specific Neurodegenerative Disease, ought to be considered as a general source of information for a given claim, and not directly linked to the Neurodegenerative disorder that is relevant to this work.

1.1 Context

1.1.1 Company X

This thesis has been developed within the scope of Company X's technology. Company X is a company that uses a photonics-based device to acquire the backscatter of biological samples. Company X's technology stands to become pivotal towards the development of new effective therapies for high-impact diseases as it is able to affordably and easily acquire relevant medium information from the backscatter signal, which is currently analyzed using state-of-the-art Machine Learning methods. Note that, although this device may be used for the analysis of samples of individuals with different pathologies, one of the diseases that has been the focus of the work developed by Company X is a Neurodegenerative disease, which will be used as the backdrop for this thesis - *i.e.* this thesis applies Deep Learning for backscatter analysis on data that is from healthy and Neurodegenerative disease afflicted individuals.

1.1.2 Neurodegenerative disease

Neurodegenerative disease (ND), one of the neurological disorders that is a top-ranking leading cause of death in the world [1], has been described as a “cruel disease”, since it completely robs individuals of their identity, causing immeasurable pain to afflicted individuals and their families - the costs associated with this disease are, however, measurable - the global market of Dementia, a ND, was valued at 818 thousand millions of dollars, and it was expected to increase, as this disease is more prevalent in the older population, and the current trend is towards an increase of the geriatric population [1, 2]. This disease is related to certain molecules, certain proteins, that contribute to the death of neurons, although not everything related to it is understood, as there are still new discoveries being made [3]. The behavior of light when interacting with samples from healthy individuals will differ from that of ND patients, due to the differences in the mediums’ components, which is what allows Company X’s technology to be used to discriminate between healthy and non-healthy individuals.

1.1.3 Deep Learning

Deep Learning is a subset of Machine Learning, where Machine Learning is a field that studies algorithms that automatically model, hence the learning, the structure of certain data. This is done through a process called training, where, instead of having an algorithm that contains a set of rules that may be able to interpret certain data, an algorithm is given said data and it itself creates the rules to interpret it. More specifically, the training process selects the best parameters for a model with a certain architecture in order to model the given data. An example of this is predicting if a pet is a dog or a cat from the length of their snout - a machine learning algorithm would be given lengths of snouts and understand that there was a distribution within the data, where dogs had a longer snout and cats a shorter snout. A non-machine learning algorithm would simply be a threshold, where snouts longer than said threshold would belong to dogs.

There are several Machine Learning algorithms, containing different types of training methods, and different models’ architectures - Deep Learning algorithms are those that use a specific architecture, neural networks, as their basic architectural component. Neural networks are said to be inspired by the process through which biological neurons interact with each other - it makes use of units, called neurons, which are connected between themselves, where each neuron holds a value, propagating it to other neurons through the mentioned connections, resembling synaptic behavior. Note that henceforth any method which involves neural networks is referring strictly to artificial neural networks.

Commonly, although not strictly, a neural network is used for supervised learning, where different singular elements of the data that are being modeled, samples, belonging to different groups, classes, which is known *a priori* - *i.e.* a labeled dataset, is used to train the neural network. This means that the neural network will contain input neurons, neurons that will hold input values of the different samples, and output neurons, neurons that will hold the predicted outputs of the model, the predictions of the model that for each sample ought to be equal to the *a priori* known

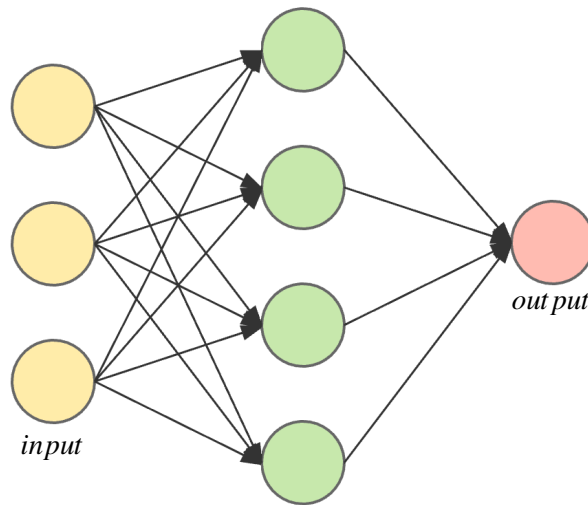


Figure 1.1: Structure of a Neural Network - inspired from [4].

classes. Between the input neurons and the output neurons, there will be a certain number of neurons that connect the input and the output, forming a network. When comparing the prediction of a sample and its true class, an error can be calculated, and this error can be linked to the connections between each neuron, which is what allows for the learning process.

The concept of a computational neuron was first proposed in 1943, by McCulloch and Pitts [5], and it was only a few years later, in 1954, that it was computationally simulated by Farley and Clark [6]. A few years afterward, research in this area stagnated, as neural networks were not believed to be capable of certain simple operations, while simultaneously requiring large amounts of computational resources - *i.e.* they were deemed ineffective and inefficient. It was only after several years that neural networks were brought back into research, as computational power became more easily accessible, and algorithmic innovations were able to provide solutions to problems that neural networks had. Nevertheless, it was only recently, in the second decade of the 21st century that neural networks started to become extremely competitive. The main factors that have led to this are the continuing increase of available computational power, the increase of available data, and further algorithmic innovation.

Given the current digital era, where there is a wide availability of high-quality sensors that can acquire high-quality data, and where everything is connected via the internet, data is easily shared and stored. We, humans, have now access to large amounts of data, far from the realities of recent past times. What was experimentally observed was that the more data available, the better the performance of a neural network, as long as the increase associated with the amount of data used is accompanied by a proportional increase of the size of the network. This latter increase has only been allowed since the available computing power has also become larger and more accessible, where, nowadays, graphics processing units, and even tensor processing units, are used in an incredibly efficient way. Furthermore, algorithmic innovation has also played a significant role, especially in areas where there is not as much data available as what is to be desired. The culmi-

nation of these factors has led to a technological conjuncture that has been favorable to the success of Deep Learning methods over more traditional Machine Learning methods. Furthermore, there are other advantages to Deep Learning that will be explored later.

Deep Learning methods have had a very high impact on many complex problems such as object detection [7], speech recognition [8], drug discovery [9], and genomics [10], often surpassing the state of the art which relies on more traditional Machine Learning methods. The models that are created sometimes perform even better than human experts [11]. Consequently, Deep Learning algorithms seem to be a tool worth exploring for many different fields and tasks.

1.2 Motivation

Regarding ND, most drugs in the market are only symptom-alleviating drugs [12], and over the past several years there have been hundreds of failed clinical trials [13], with only very few drugs being approved by regulatory bodies, having FDA recently added a an accelerated approval program that can possibly apply to NDs [14]. Clinical trials, overall, have not had a lot success, and there are some key factors that contribute to that. Currently, one of the methods used to select candidates for the trials, the lumbar puncture, is an invasive and expensive method [15], discouraging individuals from participating in the trials, and representing a serious fiscal disincentive: According to Nalls *et al.* [16], DAT scanning, often used for the diagnosis of Parkinson's Disease [17], can cost thousands of dollars, some estimates pointing at a value within the range of 2,500\$ and 5,000 \$ [18].

Considering that there are more tests that can be carried out [19], it can be considered that the average cost of a screening test for ND is between 5000\$ and 1000\$ in the United States and lower in Europe - according to Company X's sources, conservatively, in Europe the cost is closer to the lower end of the cost in the United States. Furthermore, a large percentage of the individuals are considered unsuitable after undergoing this process [20] - all these factors combined lead to strong obstacles in the way of developing drugs that can cure ND - the whole ecosystem for the development of drugs that can cure ND requires aid. Company X's technology aims to provide a solution to this problem, by utilizing the information extracted from the backscatter to easily and in a non-invasive way select possible candidates for drug trials.

Besides applications in the ND drug-development ecosystem, Company X's technology aims to provide a tool that can contribute towards the development of precision medicine, where patients with similar symptoms may have different underlying diseases, with different molecular signatures, requiring, ideally, different, more personalized, treatments. It follows then that the focus of this dissertation will be on the analysis of the backscattered signal and its information extraction, using Deep Learning methods, and the motivation is then to contribute to current data analysis methods that are used by Company X, contributing both to advancements to the ND drug-development ecosystem, and, in the future, to advancements in precision medicine.

1.3 Goals

The goals of this dissertation are to understand whether or not Deep Learning is a viable tool to model biological samples' backscatter and to explore and develop different architectures and methods and compare their performances to assess which methods may be most viable. A comparison to more traditional Machine Learning methods, using different metrics for a comprehensive and accurate one, is also to be done, in order to assess if Deep Learning is more or less viable than Machine Learning. In terms of the performance of the developed methods, note that there are two different perspectives: the first one focuses on the specific disease and use-case introduced, which is evaluating a model that is operating as a complementary tool in the ND drug-development ecosystem, and the other is a more general one, oriented towards the future goals of the company, which is to develop a system that can analyze data from individuals with different diseases that may, however, share the same symptoms, and to, instead of selecting the same treatment for both, provide insights to a more precise medicine. This will be taken into account in the discussion of the results.

1.4 Dissertation Structure

In this dissertation, besides the introduction, there are five more chapters:

- Chapter 2 includes the main concepts required for the understanding of this dissertation.
- Chapter 3 approaches the state of the art regarding deep learning applied to data whose properties are similar to backscatter.
- Chapter 4 describes the developed methodologies.
- Chapter 5 includes the results and the discussion of said results.
- Chapter 6 is the final chapter, which contains the concluding remarks.

Chapter 2

Background

2.1	Introduction	7
2.2	Particles interactions with light	7
2.3	Backscatter Signal Acquisition	9
2.4	Signal Processing	10
2.5	Machine Learning	12
2.6	Deep Learning	23
2.7	Conclusions	28

2.1 Introduction

This chapter contains the relevant topics for the understanding of this work, starting with some basic signal theory, followed by an intuition behind the physical principles of backscatter and its acquisition, and, lastly, signal processing by pattern recognition using Machine Learning and Deep Learning.

2.2 Particles interactions with light

2.2.1 Particles

Particles are small localized objects with specific physical and chemical properties, such as size and composition [21]. A dispersion occurs when particles are homogeneously suspended in a medium, where particles can be individually analyzed. There are two types of dispersions: a suspension and a colloid. The first one occurs when the particles are large enough, compared to the molecules of the surrounding medium, to the point where they sediment, whereas in the second case the particles are not, leading to an appearance of a homogeneous medium to the naked eye [22]. A dispersion can also be categorized into a monodisperse or a polydisperse one. The first one is when the particles are equal *with respect to (w.r.t.)* a certain property, whereas the

second one is when that is not the case, *i.e.* the particles are not all equal *w.r.t.* the chosen property of study [22].

2.2.2 Back Scattering

Backscatter is when light is reflected in the direction from which it was emitted. This effect occurs when a material - one onto which light is being irradiated - is heterogeneous, presents a non-stable refractive index, or has different optic properties from its surrounding material; scattering intensity is directly related, *i.e.* proportional, to how dissimilar the refractive index of the material is when compared to the one of its surrounding media [22].

In a dispersion, a particle will transmit part of the light that is being irradiated onto it and scatter the remaining. If the particles' density is large enough, the scattered light will irradiate onto other particles, a process that will repeat itself till there is no energy left. When measuring the intensity of the scattered light, it will depend, proportionally, on the detector area and, inversely squared, on the distance between the detector and the particle [22]. Small particles behave like dipoles in this scenario. When irradiated, they act as an induced dipole, with an asymmetric distribution of charge, a positive end, and a negative end, which is induced by the electromagnetic field (*i.e.*, the light that is being irradiated). The magnitude of the dipole moment is proportional to the strength of the field and the polarizability of the particle - how polarizable is the particle. In complete fairness, it is more correct to say that it is proportional to the excess polarizability, which is the polarizability of the particle minus the one of the medium. Note then, that the scattering effect is related to the dipole-like behavior of the particles under an electromagnetic field [23].

Regarding a more rigorous analysis of light scattering, the Lorentz-Mie solution to Maxwell's equations is a numerical one for homogeneous spherical particles' light scattering [23]. There are approximations according to the relation between the size of the particle and the wavelength of the light being shone, as the Lorentz-Mie one is the generalized form. The Rayleigh scattering regime is one for the case where the size of the particle is much smaller than the wavelength of the light. It states that the particle will scatter the light in all directions, similarly to an oscillating dipole, as it is homogeneously submitted to the external electric field [22]. The intensity of the scattered radiation will be inversely proportional to the square of the distance between the detector and the particle and to the fourth power of the wavelength of the light. The intensity is directly proportional to the intensity of the light being shone and to the square of the molecular weight of the particle - a particle composed of several molecules will be composed of several dipoles when under the electric field (note that this is only the case if the light being irradiated is polarized - such as a laser beam) [22].

For the case where light is being shone on several particles, the resulting scattering effect will depend on the number of particles and the distances between them amid the surrounding volume - their density. When there are few particles with a large distance between them the resulting scattering intensity can be considered the sum of the individual particle's scattering intensity. When there are several particles, and the particle density is high, then the scattering of one particle will

act as the incident light of another particle, and so it leads to multiple scattering and other effects, which will be noticeable in the collected signal [23].

Note that light has momentum - as it is an electromagnetic wave, and matter is made up of atoms with charges, the electric and magnetic field will cause a displacement towards a certain direction (sum of the electromagnetic vectors) - this is, when light reflects (or refracts) on a surface, there is a transference of momentum [24]. This affects the backscatter signal. It is clear then that the backscatter is a complex signal, dependent on many different variables, which ought to be taken into consideration when analyzing the effectiveness of pattern recognition methods.

2.3 Backscatter Signal Acquisition

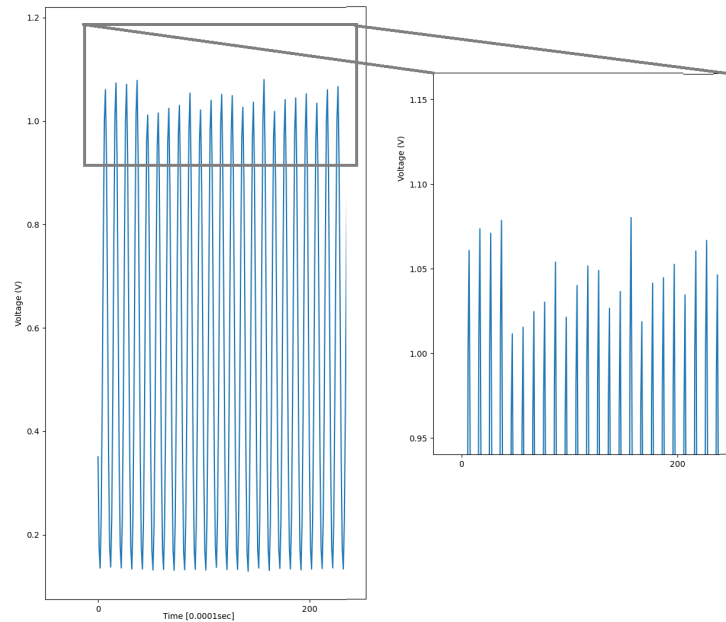


Figure 2.1: Raw backscatter acquired.

The system developed by Company X consists of utilizing a 980 nm laser, irradiating it onto a sample through an optic fiber with a specific polymerized lens. The backscattered light that results from that is sent through the optic fiber to a photodetector, which sends the acquired signal to a data acquisition board. The laser is modulated with a 1kHz signal by the same data acquisition board, which allows for less electronic noise. For each sample, the backscatter is acquired in 10 different positions, either for 30-seconds or 15-seconds in each position, leading to 300-seconds or 150-seconds, respectively, of backscatter acquisition in total for each individual. The sampling frequency of the signal can either be 10kHz or 100kHz [25]. Please see Figure 2.1 for a better idea of the signal.

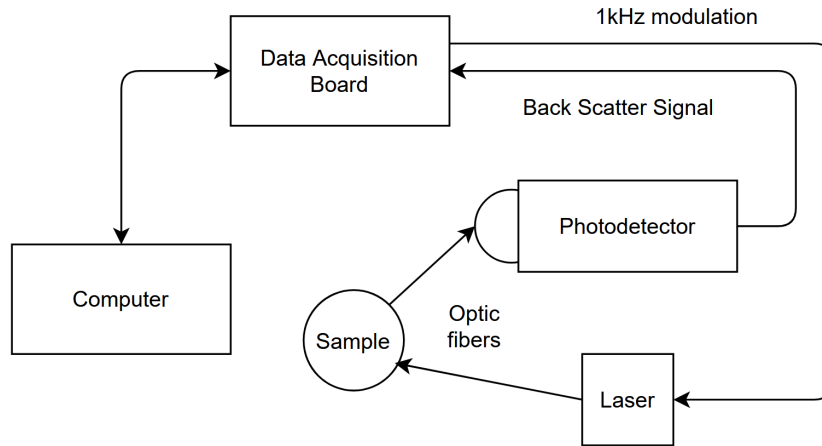


Figure 2.2: Schema of the backscatter signal acquisition - inspired from [26].

2.4 Signal Processing

Signals are generally used when conveying information about a physical system from a machine to humans [27, 28]. They can be represented mathematically as functions of one or more variables, such as time, and the information contained is usually represented as specific variations or patterns. Analog signals are continuous-time signals, *i.e.* they are defined along a continuum of time. Discrete-time signals are defined at discrete times, *i.e.* they are sequences of numbers. If they are also discrete *w.r.t.* the amplitude, then they are called digital signals [27]. The backscatter of a given sample can be represented as a signal, which can then be analyzed - hence the need to discuss signal properties and processing.

2.4.1 Discrete-time signals: sequences

Discrete-time signals are sequences of numbers. Considering a sequence x , it is formally written as $x[n]$ ¹, where n , an integer, can vary from $-\infty$ to ∞ [27]. Practically, a discrete-time signal usually results from the *periodic sampling* of an analog signal, *i.e.*, from $x_a(t)$, $x[n] = x_a(nT)$, $-\infty < n < \infty$. T is the *sampling period*, and its inverse is the *sampling frequency*, f , which represents how often the amplitude of the signal $x_a(t)$ is stored discretely in $x[n]$ [27].

A discrete-time signal can be represented in the frequency domain, instead of the time domain. Signals can be decomposed into a sum of sinusoidal signals, and the sequence of frequencies of those sinusoids corresponds to the *spectral* representation of the signal, *i.e.* its representation in the frequency domain [29]. Figure 2.3 demonstrates that for an analog signal - the spectral components can be obtained through the Fourier Transform. For a discrete signal, the same can be done with the Discrete Fourier Transform.

¹The squared brackets $[]$ are used for discrete signals, whereas parenthesis $()$ are used for continuous signals.

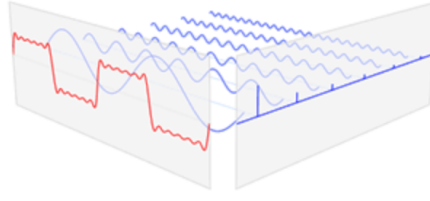


Figure 2.3: Time and spectral representation of a signal - taken from [30].

2.4.2 Representation of Sequences by Fourier Transforms

The Fourier Transform (FM) of a signal is denoted by adding a circumflex to the letter that corresponds to the initial signal. In this case, $g(t)$ will be used for the initial signal that will undergo the FT, and $\hat{g}(f)$ will represent its transform. Note that $g(t)$ is a function of time and $\hat{g}(f)$ is a function of frequencies [27].

$$\hat{g}(f) = \int_{-\infty}^{+\infty} g(t)e^{-2\pi jft} dt \quad (2.1)$$

Essentially, for a given frequency, the initial signal is multiplied by $e^{-2\pi jft}$, resulting in several rotations around the center of the complex plane by a complex vector, where the length of the vector will be equal to the magnitude of the signal for each time t - the sum of the points that are the coordinates of all of the rotating vectors will indicate whether or not that frequency is one of the spectral components of the original signal - if the value is high, then a peak will be noticed in $\hat{g}(f)$ [27]. The equation 2.1 is with respect to a continuous function, which means that it can be used for an analog signal. There is a Discrete Fourier Transform that can be used for discrete signals, which can be found in equation 2.2 - the basic principles are the same² [28].

$$X(e^{jw}) = \sum_{n=-\infty}^{+\infty} x[n]e^{-jwn} \quad (2.2)$$

The original sequence can be obtained from the frequency representation - see equation 2.3, the inverse FT. Besides the Discrete Fourier Transform, there are other relevant transforms that return the spectral component of a signal. An example of that is the discrete cosine transform, which uses cosine functions instead of sinusoids, and the Z-transform which is a broader version of the discrete FT [27]. A spectrogram is the spectrum of a signal throughout time - Fourier analysis is made for windows (segments) of the signal, resulting in a 2-dimensional plot, where frequencies are represented *w.r.t.* time [31].

$$x[n] = \frac{1}{2\pi} \int_{-\pi}^{+\pi} X(e^{jw})e^{jwn} dw \quad (2.3)$$

²There is a slight change of notation, w corresponds to the frequency.

2.4.2.1 Filters

In certain situations, it may be interesting to remove certain frequencies from a signal. For that, filters are used - they can be low-pass filters, where only frequencies lower than a threshold frequency are maintained, high-pass filters, where the opposite occurs, *i.e.* only frequencies above a certain frequency are maintained, and lastly band-pass filters where frequencies within a range of frequencies are maintained, *i.e.* within a range of two thresholds [27].

2.5 Machine Learning

2.5.1 Data and Structural Patterns

Currently, there is a lot of data available - in fact, so much so that it is unfeasible to manually analyze it all [32]. It was not always like this, ubiquitous electronics together with the world wide web have allowed for a new era of record keeping. In many cases, such data is undoubtedly a rich source of information, waiting to be transformed into knowledge - which, when applied, should improve our livelihoods [33].

Machine learning solves this issue, as it provides a set of algorithms that automatically analyze data and create models that “understand” it. These algorithms create models that, when successful, focus on the underlying structure of the data, which reveals its patterns. In fact, that is ultimately the main goal - to *recognize patterns*, as we all do when making sense of the world [33].

Recognizing patterns allows us to *predict* a certain event - if it is known that a certain event has certain prior occurring characteristics associated with it, then if those characteristics are observed it would make sense to predict that the mentioned event will indeed happen - naturally, for a lot of scenarios, this is not as clear as it may seem, but the logic behind pattern recognition stands. Note that creating algorithms to detect patterns is not necessarily machine learning - for a given task a set of rules can be created, together with their exceptions, that also describe certain data - but this, especially for more complex tasks, usually results in a lackluster performance and grueling work for the engineer or developer [34]. What distinguishes machine learning from a common pattern recognition algorithm is that the input is merely the data and the output is the set of rules that, ideally, best describe that data - machines (computers) *learn* what is relevant or not [32].

2.5.2 Supervised Learning

Machine Learning can be divided into two main parts: supervised learning and unsupervised learning (approached in section 2.5.3) [32]. In supervised learning, the data is comprised of two parts: the inputs and the outputs - *i.e.* what will be used as the input for our model and what the output should be for each input. Let us not forget that the output of the actual machine learning algorithm will be the parameters of a model - the outputs to the inputs are also referred to as labels, annotations, or targets, for the sake of clarity. Consequently, a set of data, a *dataset* D , used for supervised machine learning can be described as $D = (x_i, y_i)_{i=1}^N$, where N is the number of samples

(inputs)³, in this case annotated samples [32]. Each sample $\mathbf{x}$ ⁴ is a vector with a certain number of dimensions, usually containing features of the data - if the goal were to predict whether a person was a male or a female, then it could be as simple as a 2-dimensional vector, containing, for example, the height and the weight of a given person, or it could be a more complex structure, such as an image of a person. The label y can be a categorical one or a real-valued one [35]. In the first instance y will belong to a finite set of discrete possibilities, *i.e.* $y_i \in \{0, \dots, C\}$ where $C + 1$ will be the number of *classes* of the dataset. In the example given before, y could be 0 or 1, representing male and female. A label can also be real-valued - *i.e.* it can be any real value - for the example given before, it would be as if, instead of trying to predict the gender of an individual, the objective was to predict their age. When the labels are classes then the task can be called a *classification* problem; when they are real-values then it can be called a *regression* problem [32]. Note that the main goal, when dealing with these problems, is to create models that can predict the label of samples that have yet to be seen by the model - this is called *generalization* [32, 35]. Note also that, since this is supervised machine learning, performance metrics are easily obtained - we can test the model and calculate the distance between the given prediction and the sample's actual label [36].

2.5.2.1 Classification

A Classification task can be a *binary* one when there are only two labels - such as the example given before - or a *multiclass* classification task, when there are more than two classes - for example, predicting the breed of a dog. This definition of a multiclass problem implies that the classes are mutually exclusive, so, situations where that is not the case - for example predicting the languages that someone speaks - can be called *multi-label* classification problem - this is, however, sometimes simply viewed as several binary classification tasks in tandem [32], *i.e.* whether or not an individual speaks each language.

However, it is not unusual that ambiguous cases are found within a dataset - regarding dog breeds, for example, one dog may be a mix of two breeds. It makes sense then to return a probability, a probability of belonging to each class. When using a model with probabilistic outputs, the predicted class can be the “best guess” of the model, *i.e.* the class with the highest probability returned - this is called *maximum a posteriori estimate*, or *MAP estimate* [32, 35].

2.5.2.2 Linear Regression

In a regression task, predictions are real-valued. An example of such a task can be predicting the temperature in a room given certain features of the said room and the current weather, such as the room's volume and if it's sunny or not (realistically this model would not fare very well, the data available would most likely not be sufficient - this topic will be approached later). A type of regression that is widely used is linear regression [37].

³Note that the letter N is used to represent the number of data samples in the remaining equations, unless mentioned otherwise.

⁴When referring to a vector a bold letter will be used.

Linear regression, as the name implies, utilizes a linear combination of the input features multiplied by a parameter vector w (the f model's parameters) to calculate the label value - see equation 2.4 [34].

$$f(x, w) = w_0 + w_1x_1 + \dots w_Dx_D \quad (2.4)$$

Note that this function is not only linear *w.r.t.* the input but also *w.r.t.* the parameters as well. This can be extended, by utilizing linear combinations of nonlinear functions of the input values [34], such as:

$$f(x, w) = w_0 + \sum_{j=1}^D w_j \phi_j(x) \quad (2.5)$$

In this equation, ϕ is called a *basis function*, and it is the function that non-linearly transforms the inputs. D corresponds to the number of dimensions of the input vector x , which is manually defined. The parameter w_0 is sometimes called *bias* and acts as an offset, shifting the values in a certain direction [37]. To facilitate notation, $\phi_0(x)$ is considered to be equal to 1, which leads to the following simplified equation:

$$f(x, w) = \sum_{j=0}^D w_j \phi_j(x) = w^T \phi(x) \quad (2.6)$$

The parameter vector is sometimes referred to as θ [34]. Equation 2.6 is still considered linear since it is so *w.r.t.* the parameters. The basis function can be something as simple as the different powers of a single feature b of x , *i.e.* $x = [b^0, b^1, b^2, \dots, b^D]$ and $\phi_j(x) = x^j$, where the maximum value of j would be chosen according to the complexity of the data⁵ (naturally $j = \{0, \dots, D\}$) or more complex functions such as the Fourier basis. Nevertheless, these parameters will make up a model, function f , and it would be said that a linear regression model would be being used. Now, it is important to understand how exactly would this model *learn*. As this operation would be within the realm of supervised learning, this means that for each input $x = \{x_1, \dots, x_D\}$, the output y is known. At this point, it is pertinent to introduce the concept of *Loss* which is a measure that represents the difference between the model's output $f(x, w)$ and y , the expected output. The Loss can be formalized as a function, a *Loss function* - it becomes clear then that the goal would be to minimize the loss function, as this would lead to a model that accurately predicts the samples in the dataset [34, 32]. It is to be expected that a model that accurately predicts the samples that are in a dataset will be able to generalize - but this is not always this simple, as it will be explored in section 2.5.4.4. An example of a loss function is the mean squared error:

$$L(w) = \frac{1}{N} \sum_{i=1}^N (f(x_i, w) - y_i)^2 \quad (2.7)$$

⁵One could think that the more powers of x - from $x = [b^0, b^1, b^2, \dots, b^D]$ - the better, as the function would more easily fit all of the samples of the dataset, but this is not necessarily so - see [38].

One way to minimize the loss function is using gradient descent. The gradient of the loss function with respect to the weight vector will correspond to a D dimensional vector that can be understood as the “direction that will lead to the fastest increase and the rate of that increment” - after all, it is obtained by calculating the partial derivatives *w.r.t.* each element of the weight vector [34]. Gradient descent works in an iterative fashion - initially, the values of the weight vector w are set randomly. Afterwards, the outputs of $f(x, w)$ are calculated and so is the loss and its gradient. Finally, the gradient, multiplied by a constant, is *subtracted* to the weights vector $f(x, w)$. This is repeated a certain number of times, a certain number of *epochs*, or till the loss value is small enough, *i.e.* until $L(w) < \epsilon$, where ϵ is a pre-defined value [34, 32]. It is during this process that the model *learns*, or as it is more commonly said, that the model is *trained*.

$$w^{\tau+1} = w^{\tau} - \alpha \nabla L(w) \quad (2.8)$$

The letter τ represents the current epoch and *alpha* represents the *learning rate* - this will determine how fast the model learns. Note that, mathematically, the goal is to find a minimum of the loss function⁶ - however, if the learning rate is too high, then the weights may change drastically, to a point where they end up “far away” (euclidean norm) from the minimum [34, 37]. In this situation, it would be said that the loss is not *converging*, as it would most likely remain high during the entirety training stage. There are more reasons for a non-converging loss function, such as the non-existence of relevant minimums - in this scenario, more macro changes would have to be made, such as to the dataset or the chosen model. The gradient of the loss function is calculated according to the following:

$$\nabla L(w) = \left[\frac{\partial L(w)}{\partial w_0}, \dots, \frac{\partial L(w)}{\partial w_D} \right] \quad (2.9)$$

Where each element k , $k \in \{0, \dots, D\}$, can be calculated according to the following (assuming that the loss function used is the mean squared error and that i stands for each sample of the dataset):

$$\begin{aligned} \frac{\partial L(w)}{\partial w_k} &= \frac{\partial}{\partial w_k} \left(\frac{1}{N} \sum_{i=1}^N (f(x_i, w) - y_i)^2 \right) = \\ &= \frac{\partial}{\partial w_k} \left(\frac{1}{N} \sum_{i=1}^N (w_0 x_{i,0} + \dots + w_D x_{i,D} - y_i)^2 \right) = \\ &= \frac{2}{N} \sum_{i=1}^N (w_0 x_{i,0} + \dots + w_D x_{i,D} - y_i) x_{i,k} = \frac{2}{N} \sum_{i=1}^N (f(x_i, w) - y_i) x_{i,k} \end{aligned} \quad (2.10)$$

2.5.2.3 Logistic Regression

Logistic regression, despite its name, is actually used for classification tasks. The goal in a classification task is to associate a discrete class C_k , where $k = \{1, \dots, K\}$, to a sample vector x [32]. By imagining the features of samples within a hyper-dimensional space, we can understand that

⁶Ideally the absolute minimum, but this depends on the initial parameters.

classification implies the division of said space, the *feature space*, into *decision regions* [32]. Assuming we are using probabilistic models, *i.e.* the probability of a sample belonging to a class is returned instead of a discrete class variable, it's relevant to discuss how the outputs are represented - a common way of doing it is using *one-hot encoded* vectors, *i.e.* y will be a vector of length K such that, for the class C_j , all of its elements except for the y_j element will be 0, and y_j will be 1.

A logistic regression model is somewhat similar to a linear regression model. For the latter, a prediction will be a real value - however, for the former, a prediction that is between 0 and 1 for each class is what is desired. In order to do so, a nonlinear function g is used so that:

$$\hat{y}(x) = g(f(x, w)) \quad (2.11)$$

The function g is known as the activation function. Notice that $\hat{y}$ ⁷ here will correspond to a probability of a certain class - if this is a binary class problem, then we can consider that the $P(C_2|x) = 1 - P(C_1|x)$, and use the equation shown as is. However, if the model tackles a multi-class problem, then the weights would have to be a matrix W , with $K \times D$ dimensions (K classes and D features), and $f = x \cdot W^T$ (considering the basis function as the identity function and that $x_0 = 1$, for simplicity), returning, after being fed into the activation function, $\hat{y}$, a $1 \times K$ vector, with the predictions for each class [34]. For a logistic regression, this function will be the *logistic sigmoid*, hence the name, or simply the sigmoid function, represent by the letter σ [39]. The sigmoid function maps values from $-\infty$ to $+\infty$ between 0 and 1.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (2.12)$$

Training a logistic regression model is similar to training a linear regression one - a loss function is calculated, and gradient descent⁸ is used to find minimums. Instead of a mean squared error, it is common to use the cross-entropy error when doing logistic regression - calculated according to the following equation:

$$L(W) = \frac{1}{KN} \sum_{i=1}^N \sum_{j=1}^K (y_{i,j} \log \hat{y}_{i,j} + (1 - y_{i,j}) \log(1 - \hat{y}_{i,j})) \quad (2.13)$$

The cross-entropy loss function has a particularity that it heavily penalizes incorrect predictions since the logarithm function approaches infinity when it converges to 0 [39]. Note that this equation is functional for multi-class tasks, as the error is calculated *w.r.t.* each class. Note also that, for a given class, when the true label is 0, then the first term inside the summations is canceled, and the equivalent occurs to the second term when the true label is 1, *i.e.* the first term is canceled. Gradient descent is done similarly to how it is done in linear regression - the added layer of complexity corresponds to the activation function. However, the derivative of the sigmoid

⁷Here, $\hat{y}$ stands for prediction, and y stands for the true label.

⁸There are other optimizing algorithms that may be used, it does not have to be necessarily gradient descent.

function is a function of itself, as shown below, so it is quite manageable.

$$\begin{aligned} \frac{d\sigma}{dx} &= \frac{e^{-x}}{(1+e^{-x})^2} = \\ \frac{1+e^{-x}}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2} &= \sigma(1-\sigma) \end{aligned} \quad (2.14)$$

There is an ML model called Support Vector Machine (SVM) which is similar to logistic regression, however, an effort is made to guarantee that boundaries in the feature space are equally far apart from all of the existing classes [32]. This is the ML model that is being currently used by company X, and the best DL model developed will be compared with the SVM that is in use.

2.5.3 Unsupervised Learning

Unsupervised learning is used when the dataset is composed merely of the inputs, *i.e.* $D = x_{i=1}^N$ [32]. This means that training a model is not done using the true labels, but instead considering density estimations and clusters. Using the concept of feature space, certain *types* of inputs will be grouped, they will share a certain region of the feature space - they will form a *cluster*. The goal of unsupervised learning is to somehow discover the structure of the data [32, 40]. Unsupervised learning has the advantage that it does not require human annotation, which sometimes is very expensive (if the samples are medical images, then annotations are made by highly specialized individuals, for example). There are some very well-known algorithms, such as the K-Means algorithm, but they are outside the scope of this work - more information can be found here [32]. Some methods use unsupervised learning and supervised learning - for example, if a dataset is not fully annotated - which is very interesting, as it makes use of labels to guide the definition of boundaries in the feature space while still using the raw nature of the data to better discover its structure.

2.5.4 Neural Networks (NNs)

Neural networks were inspired by human biology, and they can be used for classification tasks (supervised learning). The main unit is a neuron, and the overall structure mimics biological connections between neurons with a graph-like structure. A neural network (NN) can be divided into 3 types of layers: the input layer, the output layer, and the layers in between those - the hidden layers [34, 41]. Let us consider a simple NN with 2 layers: the input layer, which is not counted towards the total number of layers, will have a neuron for each feature of the input vector x , and the output layer, layer 2, will have a neuron for each class⁹. The hidden layer, layer 1, will have a certain number of neurons, defined by whoever has built the NN. For h neurons, neuron 1 of layer 1 will be calculated as if it was a logistic regression model using the input neurons - *i.e.* a linear combination of weights and input features that are fed to an activation function, such as the sigmoid [34]. The weights can be visualized as links between the input neurons and neuron 1 of

⁹If it's a binary classification task then it may have a single neuron - $P(C_2|x) = 1 - P(C_1|x)$

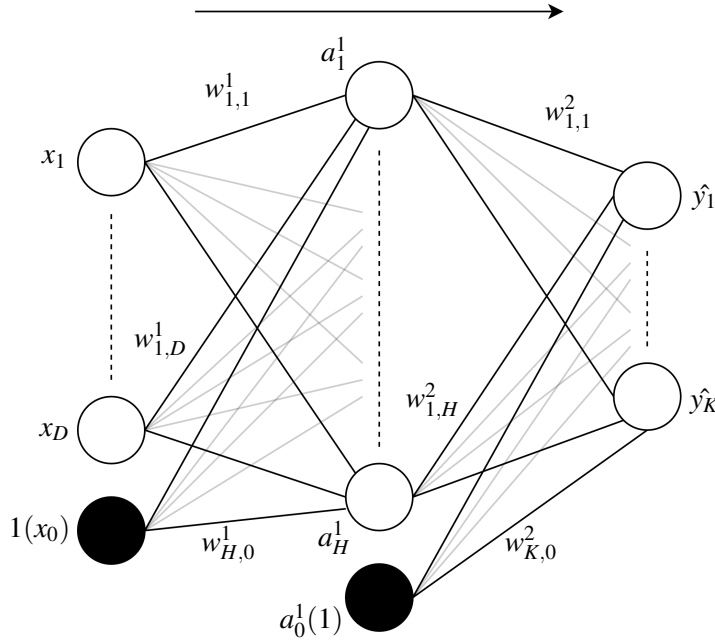


Figure 2.4: Structure of a Neural Network with 2 layers - inspired from [34].

the hidden layer. For the rest of the neurons, the same process occurs, with the exception that the weights will be different for each neuron - we can think of it as h logistic regression models in parallel!¹⁰ The neurons of the hidden layer will then be used to calculate the output neurons - assuming K classes, we can think of K logistic regression models in parallel to calculate the output layer, similarly to how the hidden layer neurons were calculated using the input neurons [41]. Note how this structure has a *feed-forward* topology - the information is gradually passed through all of the layers until it reaches the last one. Note also that non-linearity is very important - after all, if each layer was a linear combination of the previous one, then the same network could be obtained without hidden layers¹¹ [34].

Some remarks about the notation used ought to be made. Note that each weight will have the destination layer as its superscript and the destination neuron followed by the origin neuron as its subscript - *i.e.* weight $w_{1,D}^1$ will be the parameter value to apply to the input neuron D for the linear combination that will form the neuron 1 of the layer 1. Weights that originate for the neurons 0 of their respective layers correspond to the biases, and the neurons 0 will have the value 1 - this will facilitate expressing the operations as a matrix multiplication [34].

2.5.4.1 Back-propagation

So far it has been mentioned how the output values are obtained using a NN - according to a feed-forward topology, values are progressively passed from layer to layer till they reach the output

¹⁰In the literature, the concept of the perceptron is first introduced where the outputs of the neurons are binary - this concept is being skipped as it is not relevant to this work.

¹¹The composition of linear combinations is a linear combination itself: $(w_0^0 x_0 + \dots + w_D^0 x_D)w_0^1 + (w_0^0 x_0 + \dots + w_D^0 x_D)w_1^1 + \dots = w_0^1 w_0^0 x_0 + w_1^1 w_0^0 x_0 + \dots = w_0 x_0 + \dots + w_D x_D$

layer. Now, it is important to understand how the parameters of the network are updated. It is done similarly to a logistic regression model, by calculating the gradient of a loss function and applying gradient descent - however, since there are several layers, where each layer depends on the outputs of previous layers and therefore previous layers' parameters, the direct application of the chain rule is required [42]. For all of the parameters, the goal is to know the partial derivative of the loss function *w.r.t.* them. To do so, the chain rule has to be used starting from the output layer as such, considering C as the cost function (which is the same as the loss function), and L the last layer of the network:

$$\frac{\partial C(W)}{\partial w_{j,k}^L} = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial z_j^L}{\partial w_{j,k}^L} \quad (2.15)$$

Note that the notation z_j^L is used to indicate the linear combination of the previous neurons with the weights, *i.e.* $z_j^L = w_{j,0}^L + w_{j,1}^L a_1^{L-1} + \dots + w_{j,d}^L a_d^{L-1}$, assuming that the penultimate layer has q number of neurons [34, 42]. The term $\partial C / \partial a_j^L$ is dependent on the loss function used, but considering the MSE, then that term could be translated to $2(a_j^L - y_j)$, where y_j is the target for the class j . The term $\partial a_j^L / \partial z_j^L$ is simply the derivative of the activation function used, which, if it is the sigmoid function, then that would be equation 2.14. The last term, $\partial z_j^L / \partial w_{j,k}^L$ is simply $w_{j,k}^L$. So, assuming that a MSE loss function, and a sigmoid activation function, are used, equation 2.15 could be written as:

$$\frac{\partial C(W)}{\partial w_{j,k}^L} = 2(a_j^L - y_j) \cdot a_j^L (1 - a_j^L) \cdot w_{j,k}^L \quad (2.16)$$

For weights belonging to previous layers, for example for $\partial C(W) / \partial w_{j,k}^{L-1}$, then the equation 2.15 could be used, simply replacing L by $L - 1$, and taking into consideration the following:

$$\frac{\partial C(W)}{\partial a_p^{L-1}} = \sum_{j=0}^D \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial z_j^L}{\partial a_p^{L-1}} \quad (2.17)$$

Note that since a_p^{L-1} contributes to multiple activations in the last layer, it is necessary to calculate each individual contribution and then sum them. Now, all of the necessary equations have been presented which allow for the comprehension of backpropagation - as the name implies, initially the gradients of the weights of the last layer are calculated, and from those all of the other weights' partial derivatives are calculated successively, from layer $L - 1$ to $L - 2$ and so on till the input layer, obtaining the gradient of the loss [42]. The term $\partial z_j^L / \partial a_p^{L-1}$ corresponds to $w_{j,p}^L$.

The intuition with NNs with several layers is that a given layer will extract more complex features the closer it is to the output layer. For example, in a number classification task perhaps the first layer will recognize certain shapes that are commonly found in written numbers such as slight curves, straight lines, crosses, and so on. Then in the following layers more complex information is extracted such as the spatial relationship between those shapes, and perhaps further on, in *deeper* layers, how certain combinations of shapes with certain spatial relations are indicative of certain

numbers. Throughout the network different features are being extracted, different information is obtained [41].

2.5.4.2 Supervised Contrastive Loss

Note that a loss function does not have to necessarily result from the class predictions that a NN makes - take, for example, supervised contrastive loss, which is calculated according to the features that a network extracts, and was developed by Khosla *et al.* [43]. This loss penalizes a model for extracting features that are similar for samples that are from different classes, and heavily favors features that are similar for samples of the same class. This is done by comparing the similarity between two feature vectors, similarly to comparing their direction in a feature space, and obtaining a dissimilarity value - afterward doing the same chain rule is done, to obtain the loss *w.r.t.* the weights, but starting from that dissimilarity value, instead of the class prediction. The goal here is to obtain a model that extracts features that belong to as well-defined clusters as possible.

2.5.4.3 Metrics for Supervised Learning

In supervised learning the true outputs for a given input are known, so measuring the performance of a model is a straightforward process. The simplest metric is the accuracy - how many samples have been guessed correctly [36]. Naturally, the higher the accuracy the better. However, accuracy alone may not be sufficient to determine whether or not a model has learned the structure of a certain dataset - in a binary classification task, for example, if there is a large class imbalance, meaning that, perhaps, there are many negative samples and few positive samples, a high accuracy may not be adequate to understand how the model is performing *w.r.t.* the positive samples. This is because perhaps the model accurately guesses all of the negative samples but only accurately guesses a few of the positive ones, leading to an overall good accuracy but a poor performance on detecting the positives, which are, in several cases, the most important samples to accurately guess¹². It is important here to introduce the concept of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). For a binary task, TP or TN are samples that were accurately guessed as positive and negative, respectively. Conversely, FP and FN are samples that were inaccurately predicted as positive and negative, respectively as well [36, 34, 32].

To measure the ability of a model to accurately classify positive samples we can use the *sensitivity*, where the relation between TP and FN is calculated. Conversely, we can measure the relation between the TN and the FP to understand how effective the classifier is *w.r.t.* negative samples, which is called *specificity*. An average of the sum of the specificity and the sensitivity will lead to what is often called a *balanced accuracy*. If we wish to know the *precision* of a model regarding the positive samples, *i.e.* when detecting positives how often are they really positives, we have to calculate the relation between the TP and the FP [36]. Another relevant metrics is the

¹²In a medical classification task, for example, the samples that contain a pathology are the ones usually considered positive.

F1-Score, which corresponds to the harmonic mean between the precision and the sensitivity, and it aims to provide a better overall idea of how the model performs *w.r.t.* positive samples.

The formulas used to calculate the mentioned metrics follow below. Notice that these values, by definition, will range from 0 to 1, where the higher the better.

$$\begin{aligned}
 \text{Sensitivity} &= \frac{TP}{TP + FN} \\
 \text{Specificity} &= \frac{TN}{TN + FP} \\
 \text{Balanced Accuracy} &= \frac{1}{2} \left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP} \right) \\
 \text{Precision} &= \frac{TP}{TP + FP} \\
 \text{F1-Score} &= 2 \times \frac{\text{Precision} \times \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}}
 \end{aligned} \tag{2.18}$$

Usually sensitivity and specificity work on a trade-off basis - the higher the sensitivity the lower the specificity will most likely be. This is very intuitive, as the more sensible the model is to positive samples, the higher the probability of classifying a negative sample as positive. The opposite is also true. A *ROC curve*¹³ is a plot that illustrates this relationship, by plotting the sensitivity as a function of the specificity. Another measure is then introduced which is the *Area under the curve* (AUC), which reflects the overall quality of a given model - the maximum area is 1, so the closer it is to 1 the better [44].

These metrics are for binary classification tasks. However, as mentioned previously, one can consider multi-task classification as a succession of binary tasks - the way the metrics are calculated for these classification problems reflects that [32]. They can be obtained for multiclass classification tasks by considering each class *w.r.t.* to the remainder of the classes. The following is how the sensitivity is calculated in such a scenario, for a single class - the sensitivity of the whole model is the average of the sensitivities for all of the classes - the remaining metrics are calculated in a similar fashion:

$$\text{Sensitivity} = \frac{\sum_{i=1}^K TP_i}{\sum_{i=1}^K TP_i + FN_i} \tag{2.19}$$

2.5.4.4 Dataset split

When training a model, the goal is that at the end of the training process that model will be able to generalize to samples that it has yet to see. By splitting the dataset into subsets, it becomes possible to understand how a model would fair with data that it has not seen throughout the training. A common way of doing so is splitting a dataset into a training set, a validation set, and a test set [32, 34]. The training set will contain the data with which the model will be trained; the validation set will contain the data that will be used to manually adjust the model's hyper-parameters - *i.e.* model metrics will be obtained for the validation set and fine-tuning of the model's hyper-parameters will

¹³Receiver Operating Characteristic.

ensue, such as increasing or decreasing the number of hidden layers or neurons in each layer when using a NN; the test set will be used at the end of the whole model creation and training process to determine, to the best degree, what the performance of said model would be with data that it has not seen before. It is not uncommon to only use a training set and a validation set. In this case, the validation set would also be the subset that would allow for the understanding of the model's "real-world" performance - which is why, in these cases, the validation set is sometimes called the test set. The most important aspect to retain is that a subset containing different samples than the ones used for training is required to test the model, as a high accuracy for the training set may not mean a good generalizing model. Situations may happen where the model memorizes the training samples - some models may have so many tune-able parameters that, with a small training set, the gradient of the loss function will point towards a model that stores the information of the training set [32, 34]. In these cases, the accuracy for the training will be perfect, or very close to it, whereas the accuracy for the validation set will not - it would be said that the model is overfitting and that it requires regularization [38].

John Von Neumann famously said "With four parameters I can fit an elephant, and with five I can make him wiggle his trunk" [45]. Despite this being anecdotally said, the idea behind it is very important - perhaps it could be thought that the larger the model, the more parameters it would have, and the better it would be at mapping the structure of data - to the last detail, so to say. This is not true - in supervised learning, there has to be a balance between the amount of data and the number of parameters. Sometimes called the *curse of dimensionality*, if we imagine the features' hyperspace when adding a new feature, a new dimension is also being added. The increase volume-wise is proportional to the power of the number of dimensions - this means that the available samples will occupy this space in a more dispersed manner, to a point where it will be difficult to map the structure of the data and the best way to minimize the loss function will be to remember the training data [32, 34], almost like creating a marble machine, where marbles with certain colors go directly into certain slots. The opposite can also happen, where the feature space is too small, dimensionally speaking, which will not allow for the distinction between the samples of the training set - imagine using a single feature, such as the color of the hair, to determine whether a pet is a dog or a cat. In reality, overfitting and underfitting are not binary issues - NNs can slightly overfit when they are more accurate for the training set than for the validation set, which is often the case [34, 38]. NNs often have many parameters, depending on the number of hidden layers and the number of neurons in each layer, to better extract several features, from the simplest to the more complex. Regularization is required to reduce the amount of overfitting - this is done, for example, by punishing high values for the weights [34, 38]. The idea is that if the weights are very large, then the output of the network will heavily fluctuate according to the given input - however, a model that generalizes well will be responsive to evidence that is present across the training set; responding to specific characteristics of the input will most likely result in responding to the noise of the input, therefore a *regularized* network will be resistant to sample's noise, and less prone to overfitting. Another regularization method, used for NNs, is dropout, where, during the training, certain parameters are set to zero - experimentally, this method usually

leads to better results. It is thought that this reduces overfitting by enforcing more robust single-neurons [46], *i.e.* it will require all of the neurons to extract information and not allow neurons to be used to memorize data. Another very important form of regularization is data augmentation - if the dataset is not large enough, then more samples are artificially created, in order to still contain the valuable information while being slightly different, not allowing for memorization.

2.6 Deep Learning

Deep Learning is considered a subfield of machine learning - it focuses on increasing the number of layers and representations of data. Usually, the methods are related to NNs, although there is not a specific number of layers from which a model is considered deep [47].

2.6.1 Convolutional NNs

Convolutional NNs (CNNs) are a specialized form of NNs that are applied to data that has a grid-like structure - *e.g.* images, a 2-D or 3-D grid of pixels¹⁴, and time-series data, a 1-D grid (samples taken at regular intervals) [48]. As the name implies, these types of networks utilize convolutions, processing a data point with the data surrounding it, instead of all of the data available. The basic operation is represented in equation 2.20.

$$V(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i - m, j - n) K(m, n) \quad (2.20)$$

In equation 2.20, $V(i, j)$ corresponds to the value for the point (i, j) , where V is sometimes referred to as a feature map, I corresponds to the input, and K to the kernel, where each point of the kernel corresponds to a parameter, a weight [49, 47]. In this case, the input, kernel, and feature map are 2-dimensional arrays - for lower or higher dimensions, the process would be analogous. Essentially, for point (i, j) , the surrounding values are taken into consideration according to the weights (or parameters) of the kernel K to obtain the value of the feature map V at the (i, j) position - m and n can, for example, vary between -1 and 1, or -2 and 2, for a 3 by 3 (3 x 3), and a 5 x 5 kernel¹⁵ [47, 48]. This operation is done iteratively over all of the points of the input - if the input is an image, it would be akin to sliding the kernel over the image, from left to right, top to bottom, for example. In CNNs the feature map is often the result of several concatenated feature maps, resulting from several different kernels, with different parameters [47]. When the concatenated feature maps undergo a convolution operation, the kernel is 3-dimensional, *i.e.* the output of the new feature map will result from region centered at (i, j) but spanning all of the different features from the feature map. CNNs have the advantage that for a large input, such as an image, the number of parameters will be smaller than if a network with just fully-connected layers were to be used, for example. This is beneficial as it reduces the computational requirements and improves the statistical efficiency of the model [47, 49, 48]. Nevertheless, the tendency has been

¹⁴Consider pixels as data points for images.

¹⁵The kernels do not have to necessarily be squares - however, that is often the case.

to use larger and larger models, where the computational resources required are large, albeit most likely not as large as the ones required for a NN designed to accomplish the same task would most likely be.

12	11	24	56	57
10	9	36	39	40
11	7	33	34	35
5	7	28	30	32
3	6	24	26	29

 $\star$

1/3	1/3	1/3
0	0	0
0	0	0

 $=$

12	11	24	56	57
10	9	36	39	40
11	7	28	34	35
5	7	28	30	32
3	6	24	26	29

Figure 2.5: Example of the basic convolution operation with a 3x3 kernel and an image patch for the center point of the patch - inspired from [47].

The goal of the kernels is to identify certain features. Considering a given image, different kernels would *respond* differently to each region of the image. Imagining a 3 x 3 kernel, where the uppermost points would be equal to 1 and all the others would be equal to 0, then the result of the convolution of this kernel with a given patch of an image, where that patch would be represented by values from 0 to 255¹⁶, would be the highest when the uppermost values of the image patch would be the highest (within the range of available values) - as all of the others would not be relevant¹⁷ [50, 47]. Now, if instead of 0, the points other than the uppermost ones of the kernel, were to take the value -1, then the image patch that would elicit the highest response would be the one where the uppermost points of the patch would have the highest value and the others would have the lowest. This patch would resemble a line, an edge, and consequently, we could say that such a kernel would be appropriate to detect edges - in fact, we could call it a filter, as it sometimes is. This alludes to the fact that certain kernels can act as filters for certain features, such as edges and corners [50, 47]. Features can be low-level or high-level ones, such as a simple edge or an eye, respectively when classifying cats for example. The intuition behind these kernels/filters in CNNs is that they learn the relevant features, be them low or high-level ones - especially since often high-level ones are combinations of low-level ones, *i.e.* an eye can be seen as a specific spatial arrangement of a conjunction of edges - see Figure 2.6. It is clearer now why the output of a convolution is called a feature map, a group of different features obtained from different filters.

Note that for the points on the edges and corners of an image, it would not be possible to apply a standard kernel (see Figure 2.5). In these instances, several options are available [47]. The simplest one is to completely discard those points, and the result of the convolution would be smaller - for example, for an initial input with a 50 x 50 size, the result would be 48 x 48, assuming a 3 x 3 kernel size. In this case, this type of convolution would be called valid convolution, where

¹⁶8-bit grayscale image - *i.e.*, each value corresponding to a level of gray, from darker (0 - black) to brighter (255 - white).

¹⁷As the calculated value, according to equation 2.20 would be pertaining to a single point then a normalization would have to occur in order to not exceed the value range - one way of doing so would be to normalize the kernel itself so that, crudely speaking, the sum of the parameters would be 1, otherwise the output value could be higher than the highest value in the range of possible values (or lower if using negative parameters).

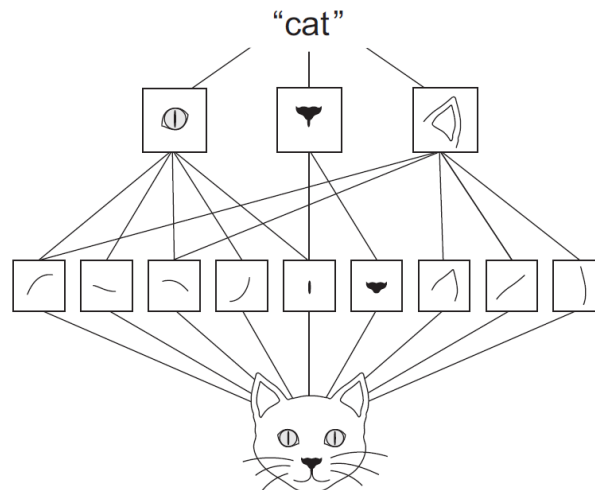


Figure 2.6: Representation of the intuition behind feature extraction in a CNN and the relation between low-level and high-level features - taken from [47].

the points except for those on the edges and corners are considered valid. However, by doing this, information would be discarded - to avoid this, padding can be done. Padding essentially means adding the necessary number of rows at the top and bottom of the input, and columns on the left and right of the input in order to allow for a standard kernel passing for the non-valid points, *i.e.* the corners and borders [50, 47]. These rows and columns can be filled with zeros - in which case the method would be called zero-padding - or other values, such as a reflection of the closest values. Padding, or the lack of it, influences the size of the convolution's result. Another important aspect of convolutions that is also relevant for the output's size is the *stride*. A stride of 1 means that the kernel is applied to each point of the data successively, whereas a stride of 2 means that the kernel is applied to every other successive point, and so on. Note that the result of a convolution will, similarly to a standard NN, be passed to an activation function. A very common one is the rectified linear unit, ReLU, that is simply $f(x) = \max(0, x)$ - this is in order to guarantee non-linearity [49].

What has been discussed so far can be called a convolutional layer - *i.e.* a convolution operation followed by the application of an activation function. The overall structure of a CNN will be made up of several layers. There is another relevant type of layer which is called a pooling layer. A pooling operation consists of downsampling a feature map. The most common example is the max pooling operation, where the maximum value for each successive exclusive region, *i.e.* starting from the top-left region and moving all the way to the right and then repeating that for the next row, without reusing points, is selected. Usually, the size of this region or window is 2×2 (the stride, in this case, would be 2 - no point is taken into consideration for two different regions), and in that case, the result of the max pooling operation would have half the size of the original feature map. Other pooling operations exist, such as the average and minimum pooling, where the average and minimum values are chosen instead, respectively. Pooling operations are usually done up to a point where the feature map is a $1 \times N$ vector, or it can be reshaped into one, and in that

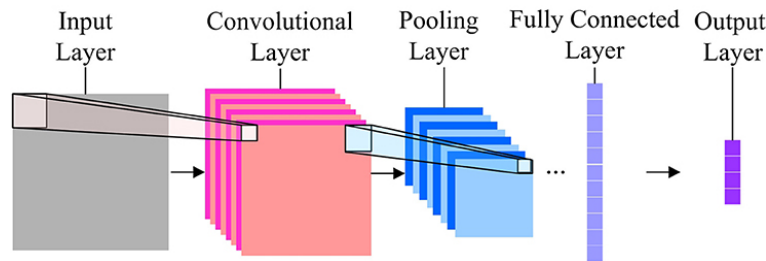


Figure 2.7: Structure of a common CNN - taken from [52].

case, a fully connected layer may be used, which is essentially a layer from a standard NN [47, 51]. Usually, the overall structure of a basic CNN would be several convolutional blocks, which can be, for example, a convolution, followed by an activation function, followed by a pooling layer, where each block would have specific hyperparameters, such as the size of the kernel or the type of padding until fully connected layers are possible. After that, it would resemble a regular NN, with the last layer containing the class predictions.

Training is done in a very similar fashion to a regular NN since the parameters of the kernels are, functionally, equal to the weights in regular NNs. Biases can also be used, where to each kernel there is a bias associated - intuitively, this should allow for the promotion of certain kernels (certain filters), and, reversely, the devaluation of others. A Loss Function is chosen, optionally having regularization terms, and a back-propagation function, such as the gradient descent one, is also chosen - after which the CNN is ready to be trained [47, 53].

When doing image classification using CNNs, sometimes an option of using pre-trained networks is available. Since, at the basic level, all images have similar features - such as edges and corners, it is viable to use the core of a pre-trained model, *i.e.* the first, or lower layers, and fine-tune the last, or the top layer - this is called transfer learning. This can naturally be done with any type of input, not necessarily images [47, 53].

2.6.2 Residual NNs

Deep NNs are considerably more difficult to train, despite being the ones currently holding the best results for a significant number of tasks. Intuitively, the basic thought is that the deeper the network, the more features, and the more relevant and complex features, it can extract. However, what is noticed is that using a simple CNN for example, if more layers are added, eventually the network will present a worse performance when compared to its counterpart with fewer layers [54]. This problem is not necessarily related to overfitting, since both the training and validation errors increase. This is rather strange as what is expected is that a deeper network will have at least the same results as a shallower network, since, after all, if it reaches a point where there are too many parameters, then some of them could simply work as an identity map, not affecting the results. However, since experimentally that does not happen, a residual learning block was introduced that explicitly allows for this identity mapping. Residual NNs (ReNNs) are networks that use shortcut connections. These connections essentially mean adding the output of a previous,

shallower, layer to the output of a latter, deeper, layer and applying non-linearity afterwards, which can be represented as $y = \sigma(F(x) + x)$, where $F(x)$ is a regular block, which can contain one or more layers, *e.g.* $F(x) = \sigma(W_t(\sigma(W_{t-1}x)))$ [54, 47].

2.6.3 Multi-task Learning

Multi-task learning basically refers to when more than one loss function is being optimized, where the network is being trained towards different tasks and only some of the parameters are shared for all of the tasks. The sharing of the parameters can be hard when they are the same, or soft when some transformation occurs. Sharing parameters has already been seen before, however in this scenario, the optimization of different tasks-solving networks occurs simultaneously - naturally, the main benefit is, besides the implicit data augmentation, a better understanding of the data structure by the network for each task [55].

2.6.4 Self-Attention Mechanism

The self-attention mechanism is a crucial component of transformers. It was first introduced by Vaswani *et al.* [56], and it has been used for several different tasks - currently, GPT-3, developed by Tom B. Brown *et al.* [57], is, most likely, the most famous transformer, being capable of remarkable feats, such as generating computer code and poetry.

The self-attention mechanism essentially operates on the basis of matrix multiplication, where a certain number of vectors are used as the input and an equal number of vectors are outputted, one for each input vector. It can be described as follows:

$$\begin{aligned}
 y_i &= \sum_j w_{ij} x_j \\
 &\text{where,} \\
 w'_{ij} &= x_i^T x_j \\
 &\text{and} \\
 w_{ij} &= \text{softmax}(w'_{ij})
 \end{aligned} \tag{2.21}$$

Softmax is a function that normalizes a vector input to a distribution of probabilities, where the highest value will correspond to the highest probability. Note that this mechanism was thought to be applied to a sequence to sequence task, *i.e.* a sequence is given as an input, and a sequence is given as an output - a simple example would be translating a sentence from one language to another: “O gato é preto” to “The cat is black”, where each sentence is a sequence of words, and, in this case, x_0 would be “O” and y_0 would be “The”. Then, each sample in the sequence would be represented by a vector, with a certain length, that would be representative of said word. By training a model with this attention mechanism, the goal would be to be able to translate sentences that were not seen before, such as “O gato é branco” to “The cat is white”, and, hopefully, not to “The cat is yellow”. Naturally, these examples are very simple, but language is often more complicated - note that using only equation 2.21, each input would have several roles: it would be

used to establish the weight for its own output, for the weights of other outputs other than itself, and as part of the weighted sum to calculate each output. An improved version does this explicitly, by having three different versions of the inputs, the queries, keys, and values, by multiplying the inputs by three different arrays, and, instead, there would be:

$$\begin{aligned}
 q_i &= W_q x_i \quad k_i = W_k x_i \quad v_i = W_v x_i \\
 &\text{where,} \\
 w'_{ij} &= q_i^T k_j \\
 &\text{and} \\
 w_{ij} &= \text{softmax}(w'_{ij}) \\
 y_i &= \sum_j w_{ij} v_j
 \end{aligned} \tag{2.22}$$

The intuition behind the self-attention mechanism is that having a very explicit relation between the different input vectors is crucial to the predictions - *i.e.*, in the case of language, using certain words in a certain order is itself relevant to the meaning of the sentence, and meaning could not be extracted by analyzing each word. An example would be “Are you yellow, McFly?” - “yellow” here has a completely different meaning than the one in “The cat is yellow”.

2.7 Conclusions

This chapter mostly focuses on machine learning and deep learning, as those will be the most relevant topics for this work. Nevertheless, some intuition behind signals and the physical principles behind the signal that has been studied is given, as that is always relevant to better understand and discuss the developed approaches within the scope of this thesis.

Chapter 3

Literature Review

3.1	Introduction	29
3.2	Analysis of backscatter signals	29
3.3	DL applied to Radar and Sonar Data	31
3.4	DL applied to 1D physiological signals	32
3.5	DL applied to sound	38
3.6	Influence of the Reviewed Literature	39
3.7	Conclusions	40

3.1 Introduction

This chapter focuses on the state of the art for tasks that use signals relatively similar to the one that is of interest - the backscatter. It was difficult to find research that specifically modeled the backscatter signal of biological samples - taking this into account, the findings may not always prove to be useful or directly applicable. Nevertheless, a broad discussion of how methodologies are applied is presented, as well as some insights that may prove useful. Table 3.1 contains a summary of the research reviewed.

3.2 Analysis of backscatter signals

As the signal that is being studied corresponds to the backscatter of a biological sample, the research that was done initially focused on that topic. However, analysis of this type of signal were not easily found - in fact, the only methods developed to which access was granted were the ones developed by Company X. They consist of using an SVM, using features extracted from the signal, after it had been processed with a high-pass filter, with a cut-off frequency of 500Hz, followed by normalization. These features are related both to the time and frequency domain of the signal - currently, 92 features in total are used - entropy, skewness, coefficients of the discrete cosine

Table 3.1: DL and ML methods reviewed.

Year	Author	Signal	Method used
2020	Paiva <i>et al.</i> [25]	Backscatter of biological samples	SVM
2018	Gower <i>et al.</i> [58]	Backscatter of particles	Kernel Ridge Regression
2018	James A. Grant-Jacob <i>et al.</i> [59]	Backscatter of pollution particles	CNN
2019	James A. Grant-Jacob <i>et al.</i> [60]	Backscatter of pollen particles	ResCNN
2019	Daniel Brodeski <i>et al.</i> [61]	Radar	U-net + NN
2019	Galusha <i>et al.</i> [62]	Sonar	CNN
2004	Stuart Perry and Ling Guan [63]	Sonar	Kalman filters + RNN
2019	Jihun Park and Dae-Jin [64]	Lofar	CNN
2017	Juhwan Kim <i>et al.</i> [65]	Sonar	Denoising AE
2018	Yong Xia <i>et al.</i> [66]	ECG	CNN
2018	Jen Hong Tang <i>et al.</i> [67]	ECG	CNN + LSTM layers
2016	Muhammad Zubair <i>et al.</i> [68]	ECG	CNN
2019	Siti Nurmaini <i>et al.</i> [69]	ECG	AE + NN
2017	Dan Li <i>et al.</i> [70]	ECG	1DCNN
2019	Fei Wang <i>et al.</i> [71]	ECG	AE
2019	Eric Ke Wang <i>et al.</i> [72]	ECG	CNN + RNN
2019	Kwon-Woo Ha and Jin-Woo Jeong <i>et al.</i> [73]	EEG	Capsule Network
2018	Xiaoyan Wei <i>et al.</i> [74]	EEG	CNN
2019	Dongye Zhao <i>et al.</i> [75]	EEG	CNN
2019	Yang Li <i>et al.</i> [76]	EEG	Multi-scale CNN
2019	Amirali Vahid <i>et al.</i> [77]	EEG	CNN
2019	Shuo Yang <i>et al.</i> [78]	EEG	Denoising AE
2018	Doborjeh <i>et al.</i> [79]	EEG	Spike NN
2019	Toraman <i>et al.</i> [80]	EEG	CNN + SVM
2019	Stefan Jonas <i>et al.</i> [81]	EEG	1DCNN
2018	Tsiouris <i>et al.</i> [82]	EEG	RNN
2019	Baiying Lei <i>et al.</i> [83]	EEG	Polynomial NN
2018	Isabell Kiral-Kornek <i>et al.</i> [84]	EEG	NN
2018	Ali Ameri <i>et al.</i> [85]	EMG	CNN
2018	Chuanjiang Li <i>et al.</i> [86]	EMG	NN + PCA
2019	Dantas <i>et al.</i> [87]	EMG	AE (1DCNN)
2019	Nodera <i>et al.</i> [88]	EMG	CNN
2020	Yi Yu <i>et al.</i> [89]	EMG	Deep Belief NN
2018	Lance Rane <i>et al.</i> [90]	Kinetic + Kinematic	NN
2015	Sojeong Ha <i>et al.</i> [91]	Sensor data	CNN
2017	Hyungui Lim <i>et al.</i> [92]	Sounds events	CNN+RNN
2018	Ahmed Humayun <i>et al.</i> [93]	Heartbeat sound	1DCNN + RNN

transform, and Daubechies 10 wavelets are an example of some of them [25]. It was found that the features that contributed the most were the ones related to the frequency domain [26].

ML and DL methods applied to other signals that were reasonably similar were found. Note the work of Gower *et al.*, where the concentration and size of particles in a media were characterized using supervised ML (Kernel Ridge Regression) on the backscatter of said media [58].

The work done by James A. Grant-Jacob *et al.* is an example of DL applied to similar signals, where DL was used on images of particles' backscatter to determine particle type [59]. This was specifically for pollution detection, and the backscatter was stored as an image - nevertheless, valuable information can be obtained, since, physically speaking, the phenomenon is the same. The network used was a traditional CNN, with 2 convolutional blocks (convolution, followed

by batch-normalisation¹, activation function, and max-pooling - occasionally, dropout was used). A total of 568 images were used, which may indicate that a reasonable, albeit not too large, data pool may be sufficient to obtain reasonable results - the developed model performed with a 97% accuracy score (acc.). Interestingly, and this is seemingly a recurring theme across networks trained on data that has frequency-related features, the learning rate used was very low - 0.0003. This may be indicative that the structure of this type of data is difficult to map since the optimized form of a model's parameters is only found with very small increments.

A year later, in 2019, James A. Grant-Jacob *et al.* continued their work, applying a ResNet-18 (residual CNN, [54]) to backscatter images (from 30 optic fibers, *i.e.* an image of the output of 30 optics fibers whose input was the backscatter signal) - the network predicted pollen types with 97% accuracy [60]. Regression tests were also made, obtaining good results as well. For this experiment, the data pool was larger - perhaps the task was more difficult requiring more data. The ResNet-18, as a residual network, allows for very large a network by adding further residual blocks, minimizing the risk of overfitting, as seen in the background section.

3.3 DL applied to Radar and Sonar Data

Other signals that were also thought to be reasonably relevant to this work were that of the Radio Detection And Ranging (Radar) and Sound Navigation and Ranging (Sonar). Since these physically operate based on reflection, some parallels may be established between the work done on them, the radar and sonar signals, and the methodologies available for backscatter.

One example found was the work of Daniel Brodeski *et al.*, where they developed a new DL network for the analysis of radar data [61]. The proposed network operates on a two-step process basis, first detecting and classifying an object, and afterward calculating relevant information, such as the azimuth angle. The first step, detection, and classification of the objects, is done using a U-Net, which has an encoder-decoder architecture - the input, in this case, is 3-dimensional. The features at the end of the encoding process and the output of the U-Net are used in the second step. This last step corresponds to a regular NN, where the inputs are a concatenation of the mentioned feature vectors. The U-Net, developed by Olaf Ronneberger *et al.* [94] has an autoencoder (AE) architecture where features from the encoding step are concatenated with features from the decoding step.

Galusha *et al.* [62] used a CNN, while augmenting the training data, to model sonar imagery. They tested several models, one of them being a 2-layer CNN, whose input was a concatenation of low and high-frequency synthetic aperture sonar imagery, followed by a pooling layer and a fully connected layer - a simple network that yielded reasonable results. The model that performed the best was a fusion model, using outputs from more than one model (essentially, an ensemble

¹When training a network, usually gradient descent is applied using the loss calculated from a batch, *i.e.* a certain number of training samples - not all. Batch-normalization normalizes the batch (*i.e.* the samples that are part of the batch) that is to be used as input to the following layer. Experimentally, it has been observed that the performance of a network that uses batch-normalization increases; theoretically, it seems to be related to maintaining the distribution of the batch, making the training process more stable [49].

method) - importantly, they prioritized results whose confidence value was high. Data augmentation focused on rotations and flips and it proved to be crucial - this served as inspiration for data augmentation used for backscatter.

Stuart Perry and Ling Guan used a two-step method to detect small objects from sonar imagery [63]. The first step corresponded to a set of Kalman filters, a specific group of filters, that extracted relevant features. These features were fed into a Recurrent NN (RNN). The created model outperformed other methods, such as traditional ML methods.

Jihun Park and Dae-Jin Jung used a CNN to determine whether or not a lofargram (similar to sonar) contained tonal frequencies [64]. The CNN had 11 convolutional blocks, followed by a fully-connected layer. The inputs of the network corresponded to the spectrograms obtained using the FT. Another recurring theme is the use of spectrograms as the input to a CNN when data has relevant frequency domain characteristics.

Juhwan Kim *et al.* [65] used a denoising AE network to treat sonar data to increase its resolution. The structure of the model was similar to that of a U-Net without the concatenation of the features from the encoding stage to the decoding stage. The model acts as a filter, focusing on the relevant aspects of the data - this is evidence that a CNN can learn sonar data relevant features.

3.4 DL applied to 1D physiological signals

Physiological signals are very useful to predict diseases or to measure the aftermath of a pathological occurrence [95]. The most commonly studied physiological signals are the electrocardiogram (ECG), the electromyogram (EMG), and the electroencephalogram (EEG). They are related to certain organs' activity, respectively, the heart, muscles, and the brain. How similar is their nature when compared to a backscatter signal is dubious - nevertheless, these signals often contain both temporal and frequency features, so the networks developed in this scope were taken into consideration.

3.4.1 ECG

The ECG signal has a very characteristic waveform. It is then evident that temporal features are very relevant to a successful analysis of the ECG signal - nevertheless, features associated with the frequency domain are also relevant, as it will be noted with the consequent reported research. Due to this, methods used for ECG analysis, specifically DL methods and the comparisons of their results with their ML counterparts, were considered relevant for this state-of-the-art study.

Yong Xia *et al.* used two different CNNs with different inputs (spectrogram and wavelets), successfully detecting atrial fibrillation in ECG signals [66]. Short Term Fourier Transform (STFT - a specific algorithm that uses the Fourier Transform (FT) concept) and stationary wavelet transform of ECG signals as short as 5 seconds were used as inputs to two different CNNs, obtaining great results (98.29% acc., 98.63% acc.). For each CNN, the input signal underwent a bandpass filter (elliptical, order 10, 0.5-50Hz) and a signal transformation (either the Fourier or wavelet

transform). The size of the window when doing the FT was found to be pivotal - a trade-off between resolution in time or resolution in frequencies was noted. The wavelet transform allowed for multiresolution analysis (a wider window for low frequencies and a narrower window for high frequencies). There were 162536 5 second segments, 38% of which belong to class 1 (pathological), and 62% of which belong to class 0 (not pathological). They fixed the class imbalance using samples from another dataset. In their case, they found that the adequate FT window size was 128. For the wavelet transform, Daubechies 5 wavelet was the mother wavelet, with the J level wavelet transform decomposition number equal to 6, as the sampling frequency was 250Hz and the frequency range of atrial activities is 4 to 9 Hz (*i.e.* somewhat "tailored" to the relevant frequencies). Unusual convolution kernel sizes (3x11) were used for the wavelet transform net, and a low learning rate was used - 0.0008.

Jen Hong Tang *et al.* used an 8-layer stacked CNN-LSTM to detect coronary heart disease in ECG signals with 99.85% acc. - this was a slight increase over a 99.62% acc., which corresponded to that of the best performing ML method [67]. Although the accuracy with more traditional methods was already quite high, a high accuracy indicates that, at least, DL methods compare with ML ones - other characteristics of the model would be interesting to analyze, such as the processing time and the computational resources required, which would allow for further comparison with traditional ML methods. The stacked CNN-LSTM also obtained better results than a conventional CNN. Circa 64k ECG segments were used, each corresponding to 5 seconds of ECG signal, each containing 1285 data points, *i.e.* with a sampling frequency of 257Hz. These segments were treated with a Daubechies 6 wavelet transform - the data used as model input actually corresponds to an altered version of the mentioned segment, which was a combination of a sinusoidal, Gaussian noise, and the signal itself after a random normalization. The model input was a reshaped version of the 1D signal, 211x24. The model is essentially a two-layer convolutional model with unusual kernel sizes, followed by three LSTM layers², the first two layers having dropout.

Muhammad Zubair *et al.* used a CNN to classify an ECG signal into 5 classes of heart pathology [68]. The data pool was quite large - 1440 minutes of data, with a 360Hz sampling frequency. Each data sample is made up of 30 minutes of data - this data was filtered (notch and band-pass, similar to the ones seen before), and used as the input (1D) to a regular CNN (3 convolutional blocks, 1 fully-connected, using a 0.001 learning rate, which is increased if the loss diminishes, and reduced by a factor of 15% otherwise), obtaining good results (92.7% acc.).

A NN was used by Siti Nurmaini *et al.* to classify diseases found in an ECG - a deep AE was used as an unsupervised way of extracting relevant features, and a deep NN was used to classify the signal using the learned features from the AE, achieving a 99.73% acc. score [69]. The model slightly outperformed conventional methods, both ML and DL ones. Using an AE allowed for a shallower network.

Dan Li *et al.* developed a 1DCNN (CNN whose input is 1-dimensional) that, using denoised ECGs, accurately (97.5%) predicted cardiovascular disease [70]. The CNN model had 5-layers, using a kernel with size 7 for the input signal (*i.e.* 1x7). The effectiveness of the size of the kernels

²A specific type of layer used in recurrent networks, which were not used in this thesis.

was studied - they concluded that there was a trade-off between kernel size and computation time, so they opted for a kernel size of 7. However, the results that were shown don't seem to indicate that the difference between the different kernels was very profound; instead, a more interesting and relevant hyperparameter was the number of kernels (feature filters) - they showed that a large number of kernels was more beneficial until a plateau was reached.

Fei Wang *et al.* used a convolutional AE to compress ECG data with a high fidelity rate - they obtained an 8,0% mean square difference [71]. The ECG signal was denoised, filtered, and used as a 1D input to an AE, which compressed the signal. That was then used as an input to another AE to obtain the original signal, creating a completely automatic end-to-end framework for data compression.

Eric Ke Wang *et al.* used a combination of a CNN and a Bi-directional RNN to extract relevant features from an ECG and classify it according to possible cardiac diseases, obtaining an accuracy that outperformed conventional CNN models and even, by a small margin, humans [72]. Essentially, for each electrode the EEG signal was denoised and filtered, being fed into a CNN core (different for each lead), which then output features to 2 RNN blocks, i.e. blocks that share information across all electrodes. The outputs of the 2 blocks are concatenated and used for the final fully connected layer. Comparisons were made: the mixed network performed the best, followed by a residual network, followed by a regular CNN.

3.4.2 EEG

The EEG signal has a less well-defined temporal shape when compared to the ECG. For that reason, more state-of-the-art methods that were used to create models to perform tasks that involved interpreting the EEG signal were analyzed.

Kwon-Woo Ha and Jin-Woo Jeong used a capsule network (which was an improvement over a regular CNN, outperforming standard DL and ML approaches for this topic) for motor imagery-based brain-computer interfaces (BCI), using the FT of the EEG signal as the input to the model [73]. Once again, the sampling frequency was 250Hz - the signal was band-pass filtered (0.5Hz to 100Hz) and notch filtered at 50Hz. These values may be relevant to understand some of the decisions taken *w.r.t.* decisions made regarding feature extraction. The traditional ML method to which the newly developed model was compared to was based on extracting features using a filter bank and using those features as inputs to a standard ML model, such as an SVM. Traditional CNNs for EEG analysis involve using the FT on EEG segments and using that as the input to regular CNNs (convolutional blocks, followed by fully connected layers). The authors used a Capsule network, which essentially means that they replaced the pooling layers with reshaping operations, transforming the feature maps into normalized vectors, which, do not incur the risk of losing information (contrary to what happens when doing a pooling operation). The EEG signal is divided into different frequency bands, so a band-pass filter from 4Hz to 38Hz is used (the remaining frequencies were not of interest in this particular case). The size of the segments used was 2 seconds, for a total of 500 data points per sample, which were converted to the frequency domain using the STFT. The actual input to the network was 3-dimensional, more than one electrode was

used. Some considerations regarding the hyperparameters used: 0.01 learning rate - this learning rate is larger than the ones previously seen, and it seems, overall, an exception regarding the learning rates used for physiological data. The best accuracy (slightly different structures were tested) was 78.44% (6.16 percentage points higher than the best ML counterpart, 2.08 percentage points higher than the best CNN-based model). In total, the training set was made up of 3600 seconds of training data and 2840 seconds of test data.

Xiaoyan Wei *et al.* used a CNN with a 3D input comprising EEG data from different channels - outperformed traditional CNN and ML methods, with an accuracy of over 90% for predicting seizures [74]. The data pool, in this case, was very large - 312 hours of EEG signal were recorded at a 500Hz sampling frequency - 2.8 hours of seizure were recorded, several seizures per patient. This translated to 36k images, 30k of which were used as training images, the remainder being split into a validation and test set. Information was being extracted from different electrodes - the input for the network was a concatenation of the signals from the different electrodes to a reshaped square 2-dimensional array of 5 seconds segments. The CNN structure itself is standard, with five convolutional blocks and two fully connected layers. The accuracy of the 3DCNN was 92.37%, higher than that of the 2DCNN, 89.91%, and of traditional ML methods, 91.25% (the difference is smaller, however, there is also an added advantage - the running time; the 3DCNN had a much shorter running time than the ML methods counterpart, 8.6s versus 85.1s).

Dongye Zhao *et al.* work focused on the task of BCI. The EEG was modeled by a CNN, using wavelet kernels and weight transfer between different individuals, obtaining better results than the baseline method [75]. The model used a 2D input: 1D filtered signal segments from different electrodes were concatenated - the input data undergoes a convolutional layer which uses specific wavelet filters, explicitly focusing on the frequency information. After this convolutional layer, regular (spatial) convolution follows, followed by a max-pooling layer and a fully connected layer. The learning rate is initially 0.003, being exponentially reduced till it reaches 0.0001. The data for each individual consisted of approximately 4.7M data points.

Yang Li *et al.* work focused on a multi-scale CNN that was used to model BCI achieving very good results (93.7% accuracy, higher than other DL methods) [76]. In this case, a multi-scale CNN essentially means using regular convolutional layers with different sized kernels for the same input and concatenating the results. They showed that different kernel sizes improved the model's accuracy by comparing the final model with more standard ones. Data augmentation was made by making slight alterations to the amplitude of the signal in the frequency domain and then using the inverse FT to obtain the original 1-dimensional signal, which, when concatenated with the signals from other electrodes, formed a 2D array, that was the input to the network.

Amirali Vahid *et al.* did very innovative work - it was the first recorded time a CNN was successfully used to detect ADHD from the EEG signal, obtaining an 83% acc. [77]. The 500Hz sampled signal was band-pass filtered from 0.5-20Hz. The input was the concatenated signals from different electrodes. The network was a regular CNN, with 3 convolution blocks, and a fully connected layer. Notably, since the data was class-imbalanced, they added a term to the loss that was the inverse of the classes' proportions, aiming towards a network that trained reasonably

equally with regards to the different classes.

Shuo Yang *et al.* developed stacked denoising AE using EEG features [78]. These were successfully used to determine the cognitive mental workload of individuals. Essentially, the stacked AEs can be described as an ensemble method, since more than one network is being used. A 92.94% acc. was obtained outperforming other classical and DL methods.

Doborjeh *et al.* were interested in studying the effect of logos' familiarity as marketing stimuli, and for that, they used a spiking NN using EEG data as its input. A good accuracy (83%) was obtained, outperforming traditional ML methods [79]. Essentially, the recorded EEG signal was converted into a 3D cube mapping neural activity, which was then used as the input to an AE, which was trained in an unsupervised manner. Afterward, the SNN (spike neural net) was used in a supervised manner, considering that certain neuronal activity is observed when there is familiarity associated with a logo (binary classification task). The network itself is basically a neural network where each neuron's activation value has to be higher than a defined threshold to propagate its value (hence the "spike") - this is used to better mimic biological neural networks, achieving good results in some scenarios, such as this one.

Toraman *et al.* wanted to understand whether or not it was possible to detect cerebral dominance applying DL to the EEG signal - for that they created several CNNs that used the concatenated spectrograms of the signals from each electrode to extract features, which were fed to an SVM layer for a classification task - the best AUC was 83% [80]. The CNNs used resulted from transfer learning - different models were tested, VGG16 had the best performance (higher than that of the VGG19 model). Transfer learning was then successfully used, even though the type of input is quite different (frequency domain images versus regular spatial domain images).

Stefan Jonas *et al.* used a 1DCNN to predict the outcome of cardiac arrest using a measured EEG signal, obtaining an AUC of 0.89 [81]. They found that the model architecture and fine-tuning of hyper-parameters did not heavily influence the result. The structure of the network itself was inspired by the VGG network, with 3 convolution blocks and a fully connected layer. They compared their results to other networks and found that they were quite similar. They also performed tests (Grad-CAM) to identify which EEG features were most relevant for the final classification and considered that they were similar to the ones that we, humans, also consider relevant.

After testing several network architectures, Tsiouris *et al.* used a two-layer LSTM network to accurately detect seizures in EEG signals, outperforming ML traditional methods [82]. Several features, which were extracted from the EEG data, were used as the input to the network. They were: cross-correlation between the different electrodes, time and frequency data, and graph theory data. Using the raw EEG data, the network massively under-performed. As the input size increased, so did the performance of the network up to a point where the difference became marginal.

Baiying Lei *et al.* used a multi-view multi-level deep polynomial network to detect relevant brain activities for motor rehabilitation from EEG signals [83]. Features are extracted from different frequency bands which are fed to a deep polynomial network (DPN). Another DPN fuses the

data from the different processed frequency bands, whose output is used to train a linear discriminant model. They tested different paradigms (text and virtual environment), obtaining a better performance than other traditional DL methods. A DPN uses a quadratic function of the input neurons to calculate the neuron value, pre-activation function, for each layer.

Isabell Kiral-Kornek *et al.* also used DL for seizure prediction, using EEG data for the model input [84]. In this case, they focused on a lighter model that runs on low-power chips - they suggest the possibility of different models for different features that are relevant to the task at hand. The model did not perform as well as models designed for each patient.

3.4.3 EMG - often used in tandem with sensor data

The EMG signal was often studied in tandem with sensor data. This means that the models developed often benefited from different sources of data, which may mean that the state of the art of EMG modeling may be the least relevant of the physiological signals. Nevertheless, intriguing material was found in the literature, which was useful for this thesis.

Interesting research work was that of Ali Ameri *et al.*, where a CNN model was used on raw EMG data for a myoelectric control scheme with results as good as traditional ML (SVM using time and frequency domain features), showing that a completely automatic learning system could be a viable alternative, although the difference was not statistically significant (p-test was done) [85]. An interesting conclusion that Ameri *et al.* made was that asserting, regarding the level of autonomy of the networks, that the CNN model is better than the SVM with chosen features is dubious, since, after all, the CNN structure is also chosen by someone. Moreover, it is also, currently, not possible to know whether a specific set of features will outperform a CNN, *i.e.* it is difficult to say that DL will eventually definitively outperform ML. Perhaps an interesting question, according to the researchers, is how would the developed models compare when applied to data obtained from different positions, *i.e.* how robust are the models. The acc. of the CNN was 91.61%, slightly higher than that of the SVM, 90.63%. Network regularization (L2) was used, as well as early stopping (if the test acc. decreased during 5 epochs, the training was terminated). The network itself is a regular CNN, 5 convolution blocks, followed by a fully-connected layer for the output, with an initial learning rate of 0.001 - the input was a 2-dimensional reshaped array of the raw signal after being filtered - 50Hz notch filter, and a 5-500Hz band-pass filter.

Chuanjiang Li *et al.* work focused on classifying force levels [86]. An EMG signal read from the wrist was used to control a prosthesis by modulating the signal with a NN, after it had been dimensionally-reduced with PCA³. The model was trained in a supervised and unsupervised manner, obtaining good results. The NN is trained using unlabeled data as an AE and afterward trained with labeled data using a second head with a fully connected layer (softmax activation). The average recognition rate is 95% (although this considers the whole method that involves other components).

³Principal component analysis - main "directions"/properties of the data [32].

Dantas *et al.* tested four different structures for an AE (CNNs, NNs, and other methods) using a data augmentation method that relied on the outputs of the decoders, concluding that decoders that use CNNs or NNs performed better than other methods [87]. The augmentation algorithm is quite interesting - by utilizing the features obtained from the decoders, more data is generated, data that is relevant to the classification task. The CNN used a 1-dimensional signal input, whose features were merged with kinematic features.

Nodera *et al.* used a CNN to successfully classify spectrograms of EMGs, especially when using transfer learning and robust data augmentation [88]. The data augmentation used was random distortions and skewing. This indicates that for frequency domain data it may be valuable to increase the data pool by creating distortions of the original data, which was taken into consideration in the work that was developed. The structure of the CNNs tested were those of VGG networks and Residual CNNs - they found that much larger networks did not necessarily yield better results; data augmentation played a pivotal role.

Yi Yu *et al.* studied the effect of fusing different data and feature level fusing methods to determine a good method to model EMG and sensor data for sign language, opting for fusing automatically extracted features from networks and manually chosen features, obtaining an 88.2% user-independent accuracy [89]. A Deep Belief Network uses some layers with undirected connections that learned in an unsupervised manner. Essentially, features extracted from the signal were concatenated with features extracted from a DBN and fed into fully-connected layers.

Lance Rane *et al.* used a deep NN to determine the magnitudes of the internal forces (EMG signal, regression problem) during gait, using kinetic, and kinematic information, obtaining better results than several competition winners [90]. A regular CNN was used, where kinetic and kinematic data was reshaped into a 2D array as the input to the CNN - the output was compared to the read EMG signal. This work may not be entirely relevant, but it is interesting in the sense that it shows that a NN can translate certain inputs into a 1-dimensional signal, so in a certain way, its parameters are related to the information of that signal and thus some level of comprehension, by the network, has been acquired.

Sojeong Ha *et al.* use a 2DCNN whose input is based on sensor data to recognize certain activities after it was shown that a 1DCNN had some success in this task [91]. The CNN architecture used was a standard one - input data was the result of concatenating different sensor data and selecting segments of data with a specific duration. They compared their results with other methods, including focusing on spectral features - they concluded that the 2DCNN outperformed other methods. The data used in this scenario was not directly related to an EMG signal, but, perhaps, to some extent it may have been indirectly so, considering that the model was designed to recognize certain motor activities.

3.5 DL applied to sound

The connection between sound and backscatter may be very shallow - nevertheless, in case some insights could be retrieved from work done with sound, the work of Hyungui Lim *et al.* and

Ahmed Humayun *et al.* is reported, as well as the work done by Daniel S. Park *et al.*.

Hyungui Lim *et al.* used a 1DCNN mixed with an RNN (with LSTM layers) to detect monophonic sound events with very good accuracy (99.93%) [92]. The proposed network uses each frequency frame of the spectrogram of the input signal as separate inputs to 1DCNNs whose inputs are transferred to an RNN. The outputs of the RNN are fed to fully connected layers. The applicability of this method may not be easily transferred to backscatter signals - monophonic sounds probably have very specific frequency characteristics (gunshot, glass breaking, ...).

Ahmed Humayun *et al.* tested different network configurations for pathological heartbeat sound classification opting for a 1DCNN, which obtained the best performance - its counterparts were other DL and ML models [93]. The dataset used had a large class imbalance which was noticed in the results obtained (the model underperformed *w.r.t.* certain classes). The 1DCNN was technically 4 different 1DCNNs cores whose features were fused into a fully connected layer. Other interesting model architectures were developed, including using features from unsupervised learning done with an RNN with other features, from ML methods, using frequency data.

Work done by Daniel S. Park *et al.* [96] showed that using specialized data augmentation on spectrograms was a powerful tool for automatic speech recognition (ASR) since it greatly improved the performance of a commonly used ASR network - they developed time and frequency masking and time warping, which proved to be very useful and a source of inspiration.

3.6 Influence of the Reviewed Literature

Regarding types of inputs, the work by Muhammad Zubair *et al.* [68] was one where the 1-dimensional input was used for a CNN, one of the main types of inputs that were tested. Others also used 1-dimensional inputs, such as Jen Hong Tan *et al.* [67], Dan Li *et al.* [70], Fei Wang *et al.* [71], Stefan Jonas *et al.* [81].

Jihun Park and Dae-Jin Jung [64] used spectrograms of a lofargram as inputs for a CNN – one of the many works reviewed that applied this concept. Others that also used spectrograms of the data as inputs to deep NNs were Yong Xia *et al.* [66]. Others, such as Kwon-Woo Ha and Jin-Woo Jeong [73], simply used the FT of the signal that they were studying (for Ha and Jeong, that was the EEG). They also concatenated information from more than one electrode – a similar idea can be found in the stacked spectrograms input, where there is the possibility of using spectrograms from different locations as a single input. This type of input was 3-dimensional, which was not new – note Xiaoyan Wei *et al.* [74], they used a 3-dimensional input as well, more specifically stacked EEG signals. There were others that also used concatenated inputs, such as Dongye Zhao *et al.* [75], Yang Li *et al.* [76], Amirali Vahid *et al.* [77], Toraman *et al.* [80] (they used specifically concatenated spectrograms) and Dantas *et al.* [87].

In the work of Galusha *et al.* [62], the method that had the best performance was the one that made use of different networks simultaneously, i.e. an ensemble network – this had an impact on the work developed, which can be seen when features extracted from two different inputs are concatenated as a single input, which can be said to have an effect similar to using two networks,

one for each input, to obtain a final output. Regarding DA, they used conventional computer vision DA, such as rotations and flips, pivotal for to a better performance, which led to those methods being tested with the available data. The work done by Paiva *et al.* [26] showed that features related to the spectral content of the signal were more relevant than the ones related to the temporal representation – this influenced the work developed, as most of the data augmentation methods that were developed were relative to the time-frequency representation of the backscattered signal. Nodera *et al.* [88] used distortions as a DA method for spectrograms – note that some of DA methods developed and tested also applied, to a certain extent, distortions. The work done by Daniel S. Park *et al.* [96] was a crucial source of inspiration for the spectrogram-specific DA.

The work by James A. Grant-Jacob *et al.* [59, 60] was a source of inspirations as they applied DL to a dataset whose physical phenomena was similar to the one that is relevant to this work – their influence can be seen in the order of magnitude of the learning rate used and in the architectures that were developed and tested, i.e. residual networks, more specifically ResNet-18. Toraman *et al.* [80] used transfer learning – to a certain extent, this may have inspired the idea of using pre-trained frozen cores for the network whose input was simultaneously the 1D input and the spectrogram.

The work by Daniel Brodeski *et al.* [61] also had a reasonable influence, which can be seen regarding the networks whose inputs were feature blocks, which resulted from the concatenation of features – both the idea of concatenating relevant information from different sources, as well as using features extracted from a DL network, can be found in their work. Others also used features extracted from NNs, such as Siti Nurmaini *et al.* [69], and Eric Ke Wang *et al.* [72].

3.7 Conclusions

In this chapter, the relevant state of the art regarding different tasks was analyzed. The findings were used throughout the development stage of this dissertation, and they are reflected in the final methodologies presented.

Chapter 4

Methodologies

4.1	Introduction	41
4.2	Input	41
4.3	Data Augmentation	43
4.4	Architectures	48
4.5	Training Processes	58
4.6	Outputs	59
4.7	Conclusion	60

4.1 Introduction

The methodologies will be presented and discussed according to the main different steps that can be found in a DL pipeline. Thus, initially, the types of inputs will be discussed, followed by the data augmentation methods used. Afterward, the architectures and training processes of the developed networks will be described, ending the chapter with the types of outputs. Note that some architectures developed were used solemnly on the features extracted from what can be considered the main networks, or the initial networks, which means that from a purely chronological perspective, they would be used at the end of the pipeline; however, for the sake of maintaining a more consistent organization, they are described in the model architectures' section.

4.2 Input

Regarding the types of inputs, there are essentially two fundamental types that were used for the developed NNs, which are the temporal and frequency domain of the backscattered signal. The remaining forms of inputs used were, in some way, combinations of these. The first one, the temporal representation, corresponds to the raw signal that is read from the photodetector, which can undergo two optional pre-processing steps, phase adjustment, and filtering. The latter uses

a high-pass Butterworth filter, with a cut-off frequency of 500Hz. The filter was thought of as a tool to possibly reduce interference from electrical sources, and the former, the phase adjustment process, as a way to, to some extent, standardize the phase, the alignment, of the signal. The key characteristic was that this input was 1-dimensional, and, henceforth it will be referred to as 1D. Regarding the characteristics of the 1D input, the original sampling frequency was either 10 kHz or 100 kHz. If it were the case that it was the latter, 100 kHz, then the input would be downsampled to 10 kHz. This was done since all of the models built and training configurations relied on data with specific dimensions, which were only possible when the original data had a 10 kHz sampling frequency. This downsampling was done by selecting every 10^{th} data point. A single acquisition could have a duration of 15 seconds or 30 seconds - to increase the number of samples that were available, different segments were obtained from a single acquisition, in a similar way to what was found in the literature to be done with physiological signals, such as the EEG or the ECG. The chosen length of these segments was 59049 data points (5.9 seconds) - this was because previous work done by Paiva *et al.* [26] showed that circa 2 seconds was sufficient to detect different materials, and, the initial network that was developed had 11 convolutional blocks with a max-pooling operation whose kernel size was 3 in the first 10 blocks, so it was appropriate to have an input whose size was a power of 3 ($59049 = 3^{10}$ - 10 convolutional blocks).

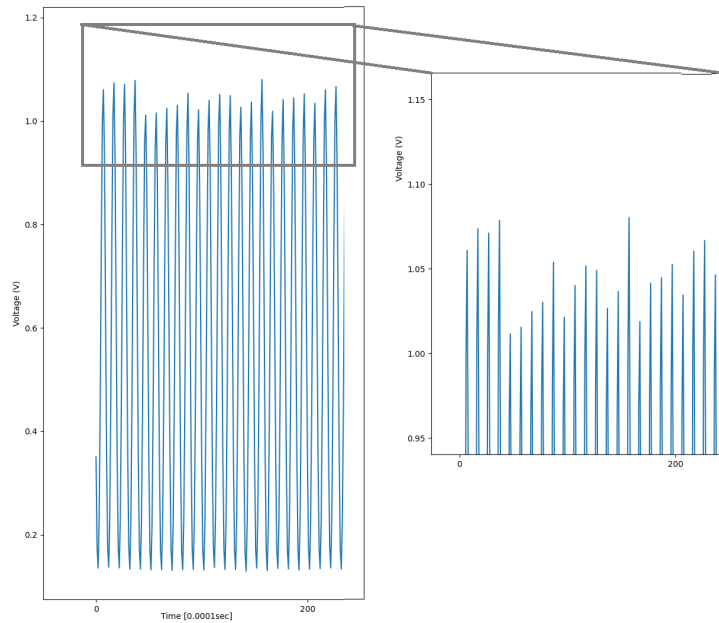


Figure 4.1: 1D input; note the periodicity of the signal is related to the modulation frequency that was used for the laser - the information can be considered to be in the peaks.

The frequency representation was obtained from these segments, by calculating their spectrogram (SG), using a Tukey window with a 0.25 taper fraction, as seen in [97]. The size of the window was determinant for the resolution of the spectrogram for different frequencies - if it was too small, then the resolution for higher frequencies would be reduced, whereas if it were too

large, then the resolution for lower frequencies would be reduced. The value that was used also had to take into account the resulting shape of the spectrogram - if the window was too large, then the width of the spectrogram would be much smaller, in proportion to the height - which, although not necessarily a problem, led to a spectrogram with an awkward shape to use, and it defeated the purpose of using the information of how the spectrum changed throughout time. Initially, a single spectrogram, a medium-sized spectrogram, was generated, with a window size of 2000 data points, or 0.2 seconds. These values were chosen to take into account the previously mentioned reasons, and window sizes that were mentioned in the literature for EEG and ECG spectrograms. Later on, two other spectrograms were calculated, with a window with half the size of the original spectrogram, *i.e.* 1000 data points, and with a window with double the size of the original, *i.e.* 4000 data points. Stacking several spectrograms, or using both the 1D input and the spectrogram simultaneously was also developed as a form of network input.

Considering all of this, there are then 5 fundamental types of inputs: 1D input, medium-sized spectrogram, 3 spectrograms with different shapes, 1D and the spectrogram simultaneously, and a block of multiple stacked spectrograms (three blocks, one for each spectrogram size)¹. Since a 1 kHz modulation frequency was used for the laser, the spectrograms had a very high amplitude for all of the 1 kHz harmonics, so, similarly to what had been done for ECG spectrograms in the literature, the logarithm of the spectrograms was used instead.

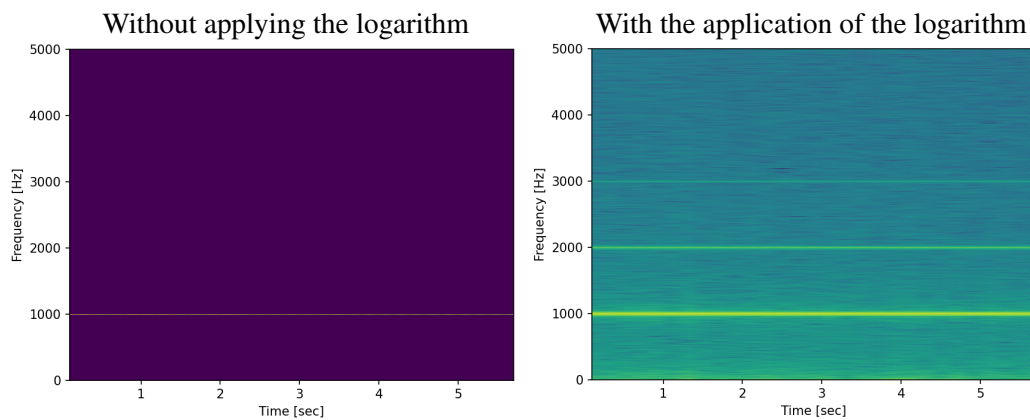


Figure 4.2: Spectrogram input; without applying the logarithm function the difference between the values is too large for a network to be able to use the data, which is why the logarithm of the spectrogram was used, following [97].

4.3 Data Augmentation

From early experiments, the 1D input was deemed to be the type of input that was most difficult to successfully use, and, consequently, the only form of data augmentation (DA) that was tested for

¹The reason why the three sized spectrograms were used for stacking was because, when the stacked input was developed, the input that seemed to lead to the best performance was the multiple spectrograms.

this input was horizontal flips. Other forms of DA were thought of as reasonable implementations, such as adding Gaussian noise, or scaling the values - however, concerns regarding the negative effects of these DA methods arose. If the signal was, by itself, difficult to model, then adding noise would likely lead to transforming the data to a point where the model would not be able to learn it. Horizontal flips seemed reasonable since the spectral content would be unchanged, which meant that it was likely that the generated data would still be useful for the model. Most of the tested and developed DA methods were used on the spectrograms of the backscatter.

The DA methods tested on the spectrograms could be split into two main categories, fundamental image transformations, and DA methods that were specifically designed for spectrograms. The former could be further divided into two sub-categories, which are DA methods whose single hyper-parameter is the probability at which they occur, *i.e.* horizontal flips, and DA whose hyper-parameters are not only how often they occur, but also, to a certain extent, their intensity - *i.e.* rotations, where not only is it required to define how often it occurs, but also the number of degrees of the rotation to be applied. The second sub-category was more difficult to explore, as several experiments would be needed to obtain a solid understanding of its effect, since if no effect, or, if a negative effect, was observed, then one could question whether or not the intensity of the transformation was not enough or too strong, respectively.

Note that, as of yet, it has not been understood why information can be extracted from spectrograms, *i.e.* what exactly is relevant in a spectrogram. Naturally, the information is related to the frequencies that are components of the signal, and, thus, DA methods that focus on maintaining the overall structure of the spectrograms, by maintaining the y-axis, the frequencies axis, unchanged seem the most reasonable. However, one could hypothesize that certain patterns, disregarding exact frequency values, may also be a source of information, and, consequently, horizontal integrity would not, necessarily, be required. Note, however, that, since the laser had a modulation frequency of 1000 kHz, the bands for this frequency and its harmonics are heavily represented in the spectrogram, and, consequently, one could also make the case that horizontal integrity is always, to a certain extent, maintained, as there are always strong reference points. Fundamentally, these questions are not trivial, and, consequently, for pragmatic reasons related to the time that it would take to run DA tests, when testing DA effects, DA designed specifically for spectrograms was given more relevance than regular image transformations, on the basis that it was, even if by a non-quantitative margin, more sound. As mentioned previously, the goal of DA was to change the values of each data point, in order to prevent the network from simply memorizing an input, while simultaneously preserving the relevant information.

4.3.1 Regular Image Transformations DA

The regular image transformations that were used for DA are the following (consider that the probability of them occurring is 0.5):

- **Rotations:** As the name implies, the effect of this is a spectrogram that has been rotated a certain number of degrees, having the center of the spectrogram as the center of rotation.

The dimensions of the spectrogram remain the same - this means that certain data points are replaced with a fill value, zero, whereas others are not used. Henceforth, whenever a fill value is mentioned, it is to be assumed that the value chosen was zero unless mentioned otherwise. The number of degrees selected was 0.5 - as the x-dimension of the spectrogram is much smaller than the y-dimension, a large rotation would have lead to a large percentage of the spectrogram being replaced by a fill value, which was deemed as too intense, from a purely observational standpoint.

- **Scaled-versions:** Spectrograms were scaled between a random factor from 0.9 to 1.1 while maintaining the same dimensions. When the scaling factor was less than 1, the borders were replaced with a fill value, almost as if padding the spectrogram, and when the scaling factor was larger than 1, then, naturally, a certain number of data points were not used, as they were “outside” of the spectrogram dimensions.
- **Shears:** This DA applies a shear parallel to the x-axis and parallel to the y-axis. The intensity of the shearing is dependent on the shear factor - the one used for the x-axis was 1 and for the y-axis 2. These values were chosen due to the difference between the height and width of the spectrogram. Since the horizontal dimension is smaller, a strong shear applied to the x-axis will lead to a large part of the spectrogram being replace by a fill value, which would reduce the amount of information found in the image.
- **Random Crops:** This effect is obtained by padding the spectrogram with a fill value and afterward selecting a crop with the initial dimensions of the spectrogram, starting at a random coordinate (within a range of coordinates that would allow for an image with the desired dimensions, *i.e.* the same as the initial spectrogram). The padding added on the left and right of the spectrogram had a size of 100, and the padding added to the bottom and top had a size of 5 - the goal of this was to guarantee that most of the image that would be the outcome of the random crop would contain relevant information.
- **Translations:** The outcome of this DA was an image that had been moved, horizontally and vertically. The dimensions of the output would be the same, and, thus, certain data points would be replaced with a fill value. The factor used for the horizontal translations was 0.05, and for the vertical translations 0.1 - this meant that considering the width of the spectrogram as w the value of horizontal shift varied between $w \times -0.05$ and $w \times 0.05$. The analogous was done with the vertical translation, using instead the factor 0.1 and the height of the spectrogram. This meant that this transformation would produce a result very similar to random crops.
- **Gaussian Noise:** DA done using this method created spectrograms that were lightly corrupted with Gaussian noise, *i.e.* noise randomly generated following a Gaussian distribution. The mean used for the noise distribution was 0 and the standard deviation was 0.1 - an array with the same dimensions as the spectrogram was created containing values according to this Gaussian curve, which was then added to the spectrogram. Whether the intensity of

the effect was appropriate or not was determined by experimental observance - several experiments would have had to be carried out in order to determine the best mean and standard deviation of the curve.

- **Gaussian Blur:** A kernel containing the discrete values of a 3-dimensional Gaussian curve with mean 0 and standard deviation 1 was used on the spectrograms as a filter, having a smoothing effect, since each pixel became a combination of itself and its surrounding pixels. The width and height of the kernel were 9 - it was set to truncate after 4 standard deviations.
- **Horizontal and Vertical Flips:** These DA effects, flipping the spectrogram using the y-axis and the x-axis as the reflection axis respectively, were applied with a probability of 0.5.

4.3.2 DA Designed Specifically for Spectrograms

These are the following DA methods that were used which were inspired by the Spec-Aug work:

- **Frequency Masking:** The result of this DA was a spectrogram with a certain frequency band replaced by a fill value. Two band sizes were tested, 20 and 40 pixels.
- **Time Masking:** Analogously to frequency masking, the result of this DA was a spectrogram with a certain time band replaced by a fill value. The two band sizes that were tested were 3 and 6 pixels.
- **Frequency Warping:** A certain randomly selected band of frequencies was replaced by the spectral information found in the frequency that was the average of the boundaries of the band, creating a warping effect, visually speaking. This DA method was very similar to frequency masking, but, instead of replacing a band with a fill value, it replaced said band with the values of a frequency instead. The same band sizes as the ones used for frequency masking were tested.
- **Time Warping:** Similar to frequency warping, but, instead of selecting a frequency band, a time band was selected. The same band sizes as the ones used for time masking were tested.
- **Frequency Warping with Interpolation:** This DA effect was very similar to frequency warping. The difference was that instead of copying the information from a single row (frequency), the main frequency, over the selected band, the values of the frequencies in the chosen band were calculated as a linear combination of the values of the main frequency and the others inside the band. The factors used differed for each frequency: the closer to the main frequency, the larger the factor that multiplied the main frequency and the smaller the factor that multiplied the non-main frequency, while the sum of the factors was 1. This led to DA that was smoother than the warping effect - due to this, a larger band size was tested, 40 pixels.

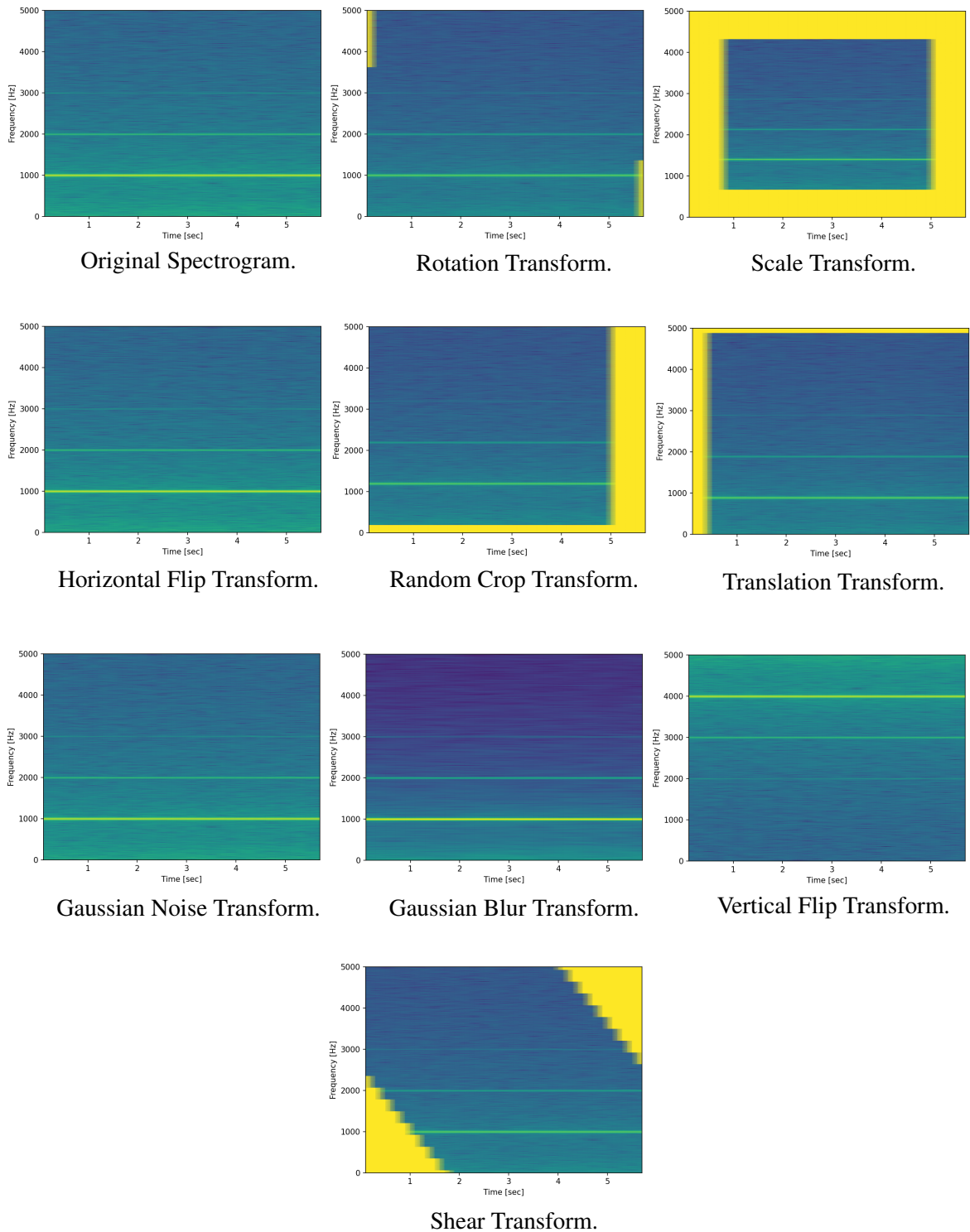


Figure 4.3: General Image Transformations Applied to a Spectrogram. Some transformations (scale transform, random crop transform, and translation transform) shown here were applied with a higher intensity in order to better show their effect.

- **Time Warping with Interpolation:** The outcome of this method was similar to frequency warping with interpolation, but, instead of being done horizontally, over a certain frequency band, it was done vertically, transforming a time band. The size of the band used was 6 pixels.

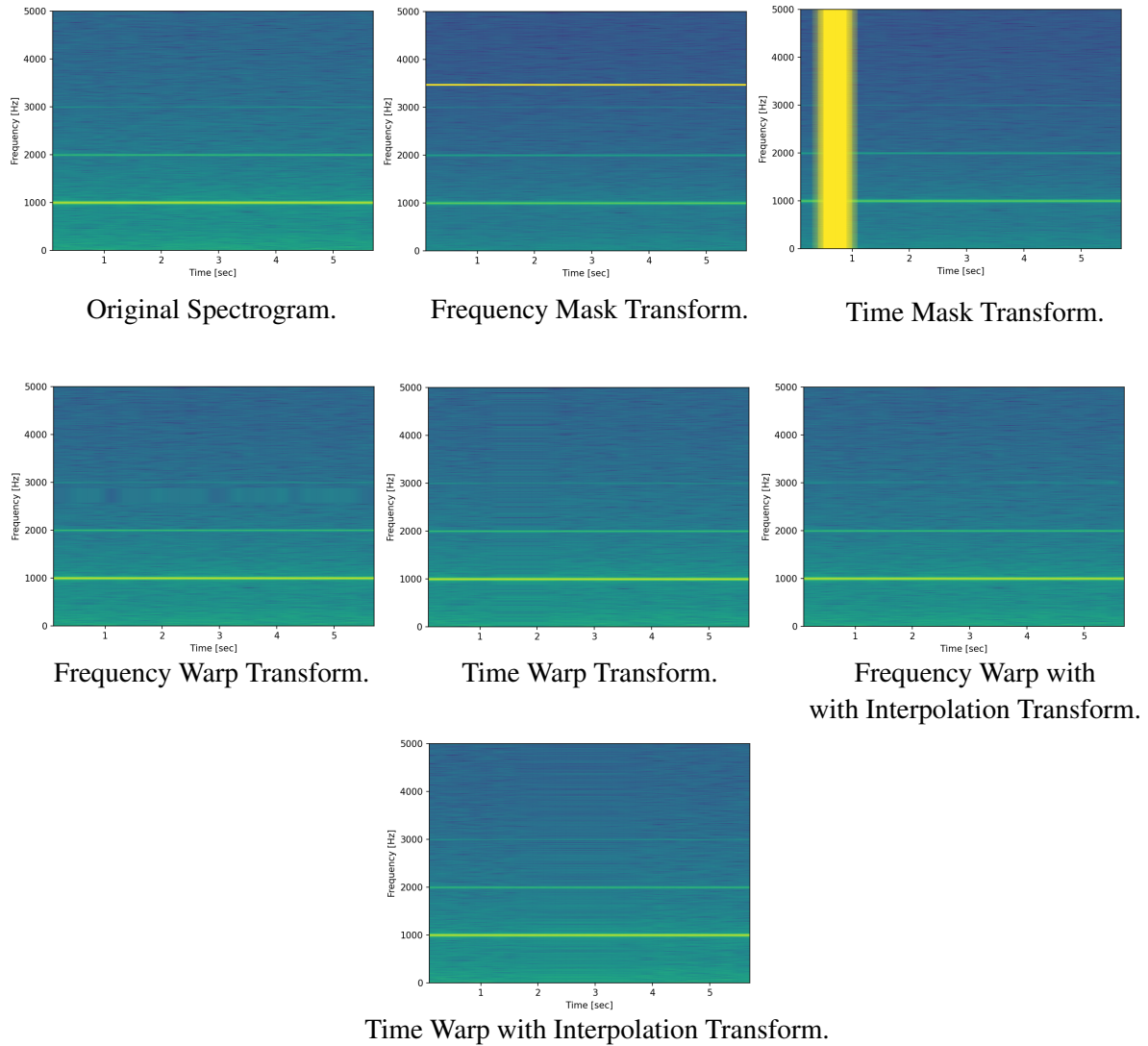


Figure 4.4: Spectrogram specific DA Applied to an Example Spectrogram. All of the transforms except for the frequency and time mask here shown were applied with a higher intensity in order to be more noticeable.

4.4 Architectures

Several different architectures were developed, and different architectures were used for different tasks. The main architectures were developed following what can be described as classic CNNs,

with initial layers that can be described as convolutional blocks, followed by some, usually three or fewer, fully-connected layers. Note that, overall, there are two main stages - the first one focuses on training a model where each input sample was a segment of the backscattered signal acquired on one location, and the second one, which was only carried out for the most promising experiments from the first stage, focused on combining, *i.e.* concatenating, features of all of the segments of a patient, creating a feature block that would itself be concatenated to the per-segment output of the initial stage, and output a single per-patient prediction.

4.4.1 Architectures developed for 1D input

4.4.1.1 Large 1DCNN

The main model built for the 1D input was a relatively large, in terms of layers, CNN, with convolutional blocks for its first eleven layers, followed by a single fully-connected layer - this model will henceforth be referred to as Large 1DCNN. The convolutional blocks contained a one-dimensional convolutional operation followed by batch-normalization - the output of this went into a ReLU activation function, and, for the exception of the last convolutional block, was max-pooled. Please consider that, unless mentioned otherwise, the activation function used for all of the layers for all of the models described in this section was the ReLU activation function. If needed, zero-padding was done for the convolutions inputs. Dropout was added to the last and sixth layers - a summary can be found in Table 4.1. The goal of this model was to understand how effective a standard convolutional model, without the latest developments regarding DL research, would perform when dealing with backscatter, and, when not using the initial optional processing (high-pass filter and phase adjustment), how would it perform with raw backscatter data. Since it was dealing with what seemed to be complex data, when designing the model the possibility of underfitting was taken into consideration, *i.e.* not having enough depth to model the data, which is why the model has several layers.

4.4.1.2 Small 1DCNN

The development of the Large 1DCNN relied on commonly used practices in DL, such as kernels with size three, and max-poolings as a form of downsampling. However, in the literature, there were some DL models developed for physiological signals that differed from the standard computer vision models, *w.r.t.* kernel sizes and stride length, which mostly tried to accommodate for the nature of the physiological signals. Following this, a CNN with larger kernel sizes and stride length was developed, to test whether or not this had an impact on the performance of the network. The increased stride length was to account for the removal of max-poolings, following, once again, previously seen work in the literature. When this model was built, the goal shifted to preventing overfitting, as initial experiments, with the Large 1DCNN, showed that after a few epochs the model's accuracy for the validation set started to decrease - consequently, it was decided that this model would have fewer layers, which is why it was called Small 1DCNN. A summary with the details can be found in Table 4.2.

Table 4.1: Large 1D CNN - ? is used to represent the batch size.

Layer	Kernel Size/Stride	Max Pool Size/Stride	Dropout	Output Shape	Number of Parameters
Convolutional Block w/o MP	3/1	-	-	?,128,19683	768
Convolutional Block w/ MP	3/1	3/3	-	?,128,6561	49.5K
Convolutional Block w/ MP	3/1	3/3	-	?,128,2187	49.5K
Convolutional Block w/ MP	3/1	3/3	-	?,256,729	99K
Convolutional Block w/ MP	3/1	3/3	-	?,256,243	197K
Convolutional Block w/ MP w/ D	3/1	3/3	0.5	?,256,81	197K
Convolutional Block w/ MP	3/1	3/3	-	?,256,27	197K
Convolutional Block w/ MP	3/1	3/3	-	?,256,9	197K
Convolutional Block w/ MP	3/1	3/3	-	?,256,3	197K
Convolutional Block w/ MP	3/1	3/3	-	?,512,1	395K
Convolutional Block w/o MP w/ D	3/1	3/3	0.5	?,512,1	788K
Fully-connected	-	-		?,Number of classes	512 x Number of classes
Total:					2.4 Million
Convolutional Block (w/o MP):	Convolution → Batch Normalization → ReLU (Activation Function)				
Convolutional Block (w/o MP w/ D):	Convolution → Batch Normalization → ReLU (Activation Function) → Dropout				
Convolutional Block (w/ MP):	Convolution → Batch Normalization → ReLU (Activation Function) → Max Pooling				
Convolutional Block (w/ MP w/ D):	Convolution → Batch Normalization → ReLU (Activation Function) → Max Pooling → Dropout				

w/o - without; w/ - with; MP - Max Pooling; D - Dropout;

Table 4.2: Small CNN for the 1D input.

Layer	Kernel Size	Stride	Output Shape	Number of Parameters
Convolutional Block	17	1	?,64,59049	1,3K
Convolutional Block	61	27	?,128,2187	500K
Convolutional Block	61	27	?,256,81	2M
Convolutional Block	61	27	?,512,3	8M
1D Max-Pool	3	1	?,512,1	-
Fully-connected	-	-	?,Number of classes	512 x Number of classes
Total:				10,5 Million

Convolutional Block: Convolution → Batch Normalization → ReLU (Activation Function)

4.4.1.3 Models with Residual Blocks - 1D input

Nevertheless, from early experiments, both models developed did not seem to be strong or large enough, in terms of parameters, to learn the data. An apparent solution was to add residual blocks

to the networks, to increase the size while preventing overfitting - this led to two new models. One was an entirely new model, a residual network, based on ResNet-18 [54], whose basic bloc was a pre-activation residual block. For this block, as the name implies, the first step was to apply the activation function to the batch-normalized input of said block. Afterward, this, the output of the activation function, would be saved, as it would later be added to itself after undergoing several operations - *i.e.* the skip connection, or shortcut, which is why this was called a residual block. If throughout those operations, the dimensions of the input were to be altered - by using a stride larger than one for the convolutions - then the shortcut would be passed to a weight-normalized convolution with a larger stride to guarantee the same dimensions. Regarding the operations themselves, they were a convolution, followed by a dropout layer and a standard, albeit reversed (since the block is pre-activated), convolution block, *i.e.* a convolution of the activated batch normalized output of the dropout layer. This would be added to the initial activated batch normalized input, obtaining the output of the residual block. A schematic of this can be found in Figure 4.5.

Table 4.3: Residual 1D CNN.

Layer	Sub-blocks	Kernel Size/Stride	Pool Size/Stride	Dropout	Output Shape	Number of Parameters
Convolutional Block w/o MP	-	51/1	-	-	?,64,59049	3.5K
Layer 1	Pre-Activation Block	51/1	-	0.5	?,64,59049	418K
	Pre-Activation Block	51/1	-	0.5	?,64,59049	418K
Layer 2	Pre-Activation Block	51/27	-	0.5	?,128,2187	1.3M
	Pre-Activation Block	51/1	-	0.5	?,128,2187	1.7M
Layer 3	Pre-Activation Block	51/27	-	0.5	?,256,81	5M
	Pre-Activation Block	51/1	-	0.5	?,256,81	6.7M
Layer 4	Pre-Activation Block	51/27	-	0.5	?,512,3	20.1M
	Pre-Activation Block	51/1	-	0.5	?,512,3	26.8M
AP	-	-	3/3	-	?,512,1	-
Fully-connected	-	-	-	-	?,Number of classes	512 x Number of classes
Total:						62 Million
Convolutional Block (w/o MP):	Convolution → Batch Normalization → ReLU (Activation Function)					
Pre-Activation Block	Please see Figure 4.5; The second compulsory convolution always has a stride of 1, the first one varies; The optional convolution is used when the stride of the first one is different than 1, taking that value as its own stride;					

w/o - without; AP - Average Pooling; D - Dropout;

The 1D Residual CNN had an initial convolutional block, followed by four layers each composed of two pre-activation residual blocks. Afterward, the output of the last layer would be average-pooled and fed to a fully-connected layer - more details can be found in Table 4.3.

The other addition was, instead of having a completely new network, to replace some layers of the original Large 1DCNN with pre-activated residual blocks - the layers that could be replaced were the convolutional blocks six through nine, and, in that case, the model would be called Large 1DCNN with Residual Blocks. These blocks would be equal to the ones of the first layer of the residual network, all having 264 feature maps, leading to a model with 28.3 million parameters -

Table 4.4: Large 1D CNN with Residual Blocks.

Layer	Kernel Size/Stride	Max Pool Size/Stride	Dropout	Output Shape	Number of Parameters
First five layers of Large 1D CNN	...	...	...	?,256,243	396K
Pre-Activation Block (w/ MP)	51/1	3/3	0.5	?,256,81	6.7M
Pre-Activation Block (w/ MP)	51/1	3/3	0.5	?,256,27	6.7M
Pre-Activation Block (w/ MP)	51/1	3/3	0.5	?,256,9	6.7M
Pre-Activation Block (w/ MP)	51/1	3/3	0.5	?,256,3	6.7M
Convolutional Block w/ MP	3/1	3/3	-	?,512,1	395K
Fully-connected	-	-	-	?,Number of classes	512 x Number of classes
Total:					27 Million
Convolutional Block (w/o MP):	Convolution → Batch Normalization → ReLU (Activation Function)				
Convolutional Block (w/ MP):	Convolution → Batch Normalization → ReLU (Activation Function) → Max Pooling				
Pre-Activation Block (w/ MP)	Pre-Activation Block (see note in Table 4.3) → ReLU → Max Pooling				

w/o - without; MP - Max Pooling;

see Table 4.4. The reason why the pre-activation residual block was the residual one used was due to it being experimentally found to be an improvement over the regular residual block, according to the literature [54].

4.4.2 Architectures developed for Spectrograms

4.4.2.1 Standard CNN

The main network used with spectrograms was a standard CNN. The input was passed to four convolutional blocks, where each block had a convolution followed by batch-normalization, activation function, and, lastly, max-pooling. Afterward, the features were sent to three fully-connected layers. The goal, at this point of development, was to create a network that was not too large to prevent overfitting, and that allowed for an input with an unusual shape. This network was used with the medium-sized spectrogram, with a 1001 by 29 shape - a detailed summary can be found in Table 4.5.

4.4.2.2 Models with Residual Blocks - SG input

Similar to what was done with the networks for the 1D input, two other networks were created for the medium spectrogram - a residual network and a network that was exactly like the main spectrogram CNN except for the last convolutional block, which was replaced by two pre-activation residual blocks. The residual blocks were the same as the ones used for the 1D residual network, with the exception that the convolutions were two-dimensional - please see Table 4.6.

The residual network for the spectrogram differed slightly from the 1D - instead of four layers with two pre-activation residual blocks, it had six and the last two, instead of having two residual

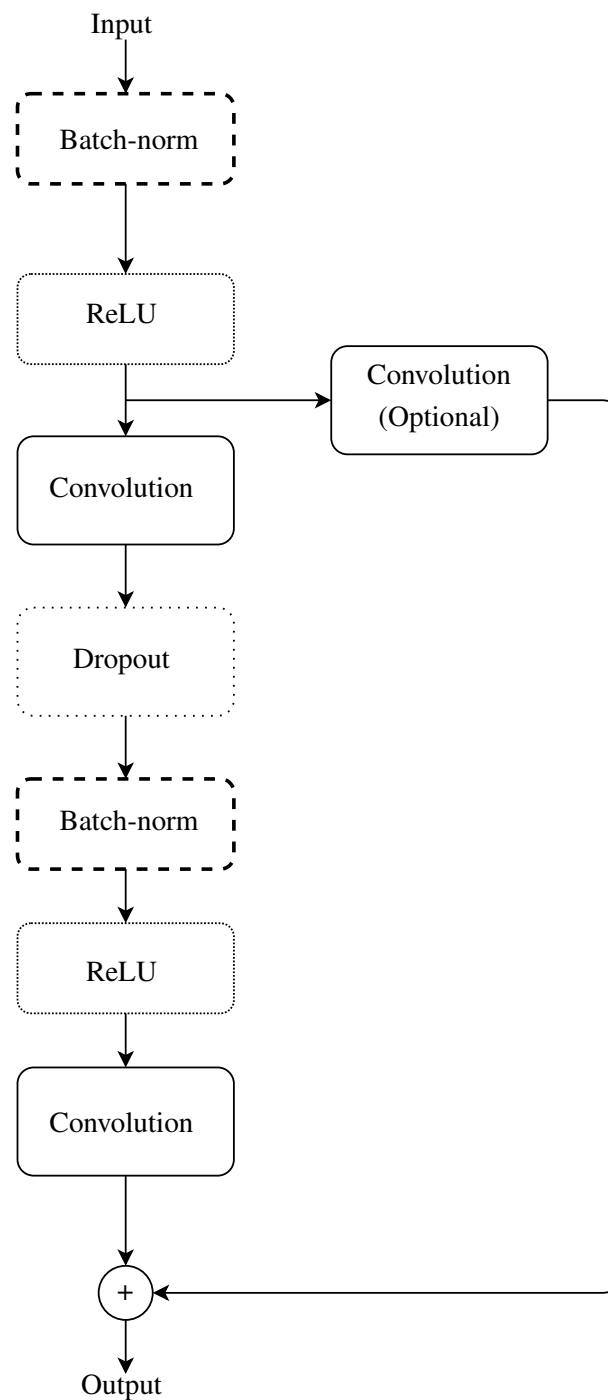


Figure 4.5: Schematic of a Pre-Activation Residual Block.

blocks, only had one. Unlike the residual 1D network, the output of the residual layers was not average pooled and instead passed directly to two fully-connected layers. A detailed description of the network can be found in Table 4.7. Note that most of the networks' hyper-parameters values and choices made regarding the number of layers, and overall structure, was essentially connected to the shape of the input, and were mostly determined experimentally.

Table 4.5: Main spectrogram CNN.

Layer	Kernel Size/Stride	Max Pool Size/Stride	Output Shape	Number of Parameters
Convolutional Block (w/ MP):	3/1	2/2	?,64,500,14	768
Convolutional Block (w/ MP):	3/1	2/2	?,128,250,7	74K
Convolutional Block (w/ MP):	3/1	2/2	?,64,125,3	73.9K
Convolutional Block (w/ MP):	3/1	2/2	?,32,62,1	18.5K
Fully-connected	-	-	?,512	1M
Fully-connected	-	-	?,64	33K
Fully-connected	-	-	?,Number of classes	64 x Number of classes
Total:				1.2 Million
Convolutional Block (w/ MP): Convolution → Batch Normalization → ReLU (Activation Function) → Max Pooling				

w/ - with; MP - Max Pooling;

Table 4.6: Standard spectrogram network with Residual Blocks.

Layer	Kernel Size/Stride	Max Pool Size/Stride	Dropout	Output Shape	Number of Parameters
First three layers of SG CNN	...	...	...	?,64,125,3	148K
Pre-Activation Block (w/ MP)	3/1	3/3	0.1	?,32,62,1	29.7K
Pre-Activation Block	3/1	-	0.1	?,16,62,1	7.4K
Fully-connected	-	-	-	?,512	508K
Fully-connected	-	-	-	?,64	33K
Fully-connected	-	-	-	?,Number of classes	64 x Number of classes
Total:					727K
Convolutional Block (w/ MP): Convolution → Batch Normalization → ReLU (Activation Function) → Max Pooling					
Pre-Activation Block	Pre-Activation Block (see note in Table 4.3) → ReLU				
Pre-Activation Block w/ MP	Pre-Activation Block (see note in Table 4.3) → ReLU → Max Pooling				

w/ - with; MP - Max Pooling;

4.4.2.3 Multiple Spectrograms Model

The last network that will be mentioned whose input was solemnly of the spectrogram type is the multiple spectrogram network. The goal of this network was to study if more spectrograms, with different shapes, *i.e.* obtained with different window sizes, could have a better performance than the one with a single spectrogram - the idea was that, perhaps, certain frequencies, which are not present in the medium-sized spectrogram, or a higher resolution in lower frequencies, would be beneficial. This network was similar to the main spectrogram CNN, whereas, instead of having a single main head, it instead had three heads, each for each spectrogram shape, whose features were concatenated and passed onto a fully-connected layer, which was itself connected to another fully-connected layer. The head for the medium-sized spectrogram input corresponded to the first four layers of the main spectrogram network. The head for the smaller spectrogram, named after the smaller window size, *i.e.* not related to its shape, was two convolutional blocks, and the head for the larger spectrogram was five convolutional blocks - these blocks were similar to the ones found in the head of the medium spectrogram, albeit with slightly different hyper-parameters, mostly regarding kernel-sizes for the smaller spectrogram. A detailed summary can be found in

Table 4.7: Residual spectrogram network.

Layer	Sub-blocks	Kernel Size/Stride	Dropout	Output Shape	Number of Parameters
Convolutional Block w/o MP	-	3/1	-	?,16,1001,29	176
Layer 1	Pre-Activation Block	3/1	0.1	?,16,1001,29	2.6K
	Pre-Activation Block	3/1	0.1	?,16,1001,29	2.6K
Layer 2	Pre-Activation Block	3/2	0.1	?,32,501,15	14.3K
	Pre-Activation Block	3/2	0.1	?,32,501,15	18.4K
Layer 3	Pre-Activation Block	3/2	0.1	?,64,251,8	57.3K
	Pre-Activation Block	3/2	0.1	?,64,251,8	73.8K
Layer 4	Pre-Activation Block	3/2	0.1	?,128,126,4	229.5K
	Pre-Activation Block	3/2	0.1	?,128,126,4	295K
Pre-Activation Block		3/3	0.1	?,64,42,2	118.9K
Pre-Activation Block		3/2	0.1	?,32,31,1	23.4K
Fully-connected	-	-	-	?,512	344K
Fully-connected	-	-	-	?,Number of classes	512 x Number of classes
Total:					1.2 Million
Convolutional Block (w/o MP):	Convolution → Batch Normalization → ReLU (Activation Function)				
Pre-Activation Block	Please see note in Table 4.3;				

w/o - without; MP - Max Pooling;

Table 4.8

4.4.3 Networks with mixed inputs - 1D and spectrograms

Another network type developed was one that had both types of inputs, both the 1D and the spectrogram. This was done by utilizing the cores of the Large 1DCNN, *i.e.* the first initial eleven convolutional blocks, and the core of the medium spectrogram CNN, *i.e.* the first four convolutional blocks. These cores outputted vectors with 512 features - these two vectors were concatenated along the second axis, forming a vector with 1024 features, which was passed to three fully-connected layers. The cores could have, instead of the regular convolutional blocks, residual blocks. Naturally, the main goal when developing this network was to assess whether or not it was possible to combine the strengths of both inputs, assuming that they were both useful. A detailed description of the network can be found in Table 4.9.

Table 4.8: Multiple spectrogram network.

Input	Layer/Operation	Kernel Size/Stride	Max Pool Size/Stride	Output Shape	Number of Parameters
Medium SG	First five layers of standard SG CNN	...	...	?,512	1.2M
Small SG	Convolutional Block (w/ MP):	5/2	2/2	?,16,500,3	448
	Convolutional Block (w/ MP):	5/2	2/2	?,16,125,1	73.9K
	Fully-connected	-	-	?,512	1M
Large SG	Convolutional Block (w/ MP):	3/1	2/2	?,32,250,29	384
	Convolutional Block (w/ MP):	3/1	2/2	?,64,125,14	18.5K
	Convolutional Block (w/ MP):	3/1	2/2	?,64,62,7	37K
	Convolutional Block (w/ MP):	3/1	2/2	?,32,31,3	18.5K
	Convolutional Block (w/ MP):	3/1	2/2	?,32,15,1	9.3K
	Fully-connected	-	-	?,512	246K
	Concatenate Features	-	-	?,1536	-
	Fully-connected	-	-	?,512	787K
	Fully-connected	-	-	?,Number of classes	512 x Number of classes
Total:					3.3 Million
Convolutional Block (w/ MP): Convolution → Batch Normalization → ReLU (Activation Function) → Max Pooling					

w/ - with; MP - Max Pooling;

Table 4.9: 1D plus spectrogram CNN.

Layer/Operation	Kernel Size/Stride	Max Pool Size/Stride	Output Shape	Number of Parameters
First eleven layers of Large 1DCNN	...	...	?,512	2.4M
First four layers of standard SG CNN	...	...	?,512	1.2M
Concatenate Features	-	-	?,1024	-
Fully-connected	-	-	?,512	525K
Fully-connected	-	-	?,64	33K
Fully-connected	-	-	?,Number of classes	64 x Number of classes
Total:				4.2 Million

4.4.4 Patient Feature Networks

Initially, the per-patient prediction was obtained from the average of the predictions for each segment of a given patient - it was noted, throughout early experiments, that the models had, in general, higher sensitivity, which led to the average of the predictions favoring class 1, in binary classification tasks. Considering a scenario where different ratios of positive to negative samples could be an important feature itself, the development of a new dataset was carried out. Each sample of this dataset consisted of firstly concatenating the features of the input segments, from the core of a network, with their predictions, obtained from the used model for that segment, forming a feature-prediction array, and, afterward, concatenating all of those features-predictions arrays of each patient to form a dataset where each patient had a single input sample - a feature-block. The order of these segments is set as the files are being loaded from the folder - since the seed is set for all experiments, the order is the same for all of the experiments that were carried out with patient feature networks. This dataset could be used to train three different networks, which were

a standard CNN, a transformer, and a network that first passed the feature-blocks to convolutional blocks, and afterward passed the output of those to a transformer - essentially, a combination of the previous networks, which was called CNN-T.

4.4.4.1 Standard CNN - Patient Feature Blocks Model

The standard CNN for the feature-blocks consisted of three convolutional layers, without max-poolings, followed by a linear layer. This network had a very small number of parameters, and, since the input was, supposedly, high-level features, it was thought that, perhaps, it would be more beneficial to not downsample the data throughout the network, which is why no poolings were done. A more detailed view can be found in Table 4.10

Table 4.10: Standard CNN for feature-blocks.

Layer	Kernel Size	Stride	Output Shape	Number of Parameters
Convolutional Block:	(3,5)	2	?,16,16,32	7.2K
Convolutional Block:	(5,7)	4	?,16,4,8	9K
Convolutional Block:	(3,3)	2	?,8,2,4	1.1K
Fully-connected	-		?, Number of classes	64 x Number of classes
Total:				17.5K
Convolutional Block:	Convolution $\rightarrow$ Batch Normalization $\rightarrow$ ReLU (Activation Function)			

4.4.4.2 Transformer

The transformer started by flattening each feature-prediction array into a vector and embedding the position of each feature in those vectors. This embedding was done to guarantee that the position of each vector, formerly a feature-prediction array, remained relevant, which, otherwise, would not be possible. The idea behind this was that possibly, in the future, DA could be added, where, for a given patient, the feature-predictions arrays could be swapped. Afterward, the embedded, flattened, inputs were passed to two consecutive transformer blocks. The input of a transformer block, x , was used as the input for a self-attention layer, being afterward added to itself, as a skip connection, and normalized, *i.e.* $x = \text{normalize}(\text{self-attention}(x) + x)$. Afterward, it was fed to two fully-connected layers, being the output of the first one activated before being sent to the second one, and then, once again, added to itself, to form another shortcut, and, finally, normalized, *i.e.* $x = \text{normalize}(\text{two-fully-connected}(x) + x)$. The mean, throughout the y-axis, of the output of the transformer blocks, was passed to a fully-connected layer, in order to output the final prediction. A description with more details can be found in Figure 4.6. This model was, due to the input shape, a slightly altered version of the initial transformer by [56].

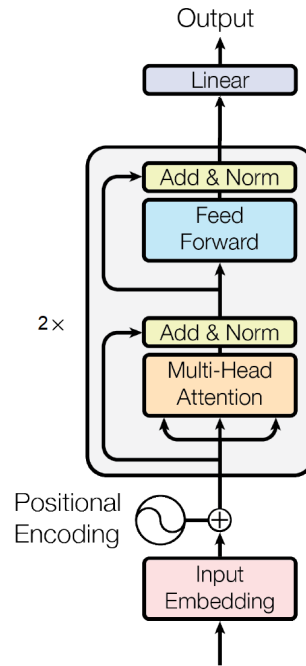


Figure 4.6: Transformer architecture - taken from [56].

4.4.4.3 Convolutional Block Followed by Transformer Block

The last network to be described is the CNN-T - the first part consisted of two standard convolutional blocks without batch-normalization and with max-pooling. The output of this was flattened and sent to a transformer similar to the one described before - positional embedding was added to the input, followed by a dropout layer, followed by two transformer blocks, similar to the ones of the transformer. Afterward, the output of the transformer blocks was normalized, and, instead of being averaged, the vectors were multiplied by a transposed version of themselves, that had been passed to a fully-connected layer and softmaxed, forming a single vector per patient, which was then passed to a fully-connected layer. This model was taken from [98] - CCT2. A more thorough understanding can be obtained from Figure 4.7.

4.5 Training Processes

Besides the standard training process, where stochastic gradient descent was done taking into account the calculated class-weighted cross-entropy loss, which was done both for regular training, *i.e.* where the inputs were segments of the several acquisitions for each patient, and for the feature-blocks networks, there were two other training methods that slightly differed from this one. The first one was supervised contrastive learning, where, during a first stage, features were obtained from the core of the networks, and the networks were trained with supervised contrastive loss, using the extracted features. Afterward, the layers that were trained using this loss were frozen and the final layers were fine-tuned, using regular training. This process was taken from [43].

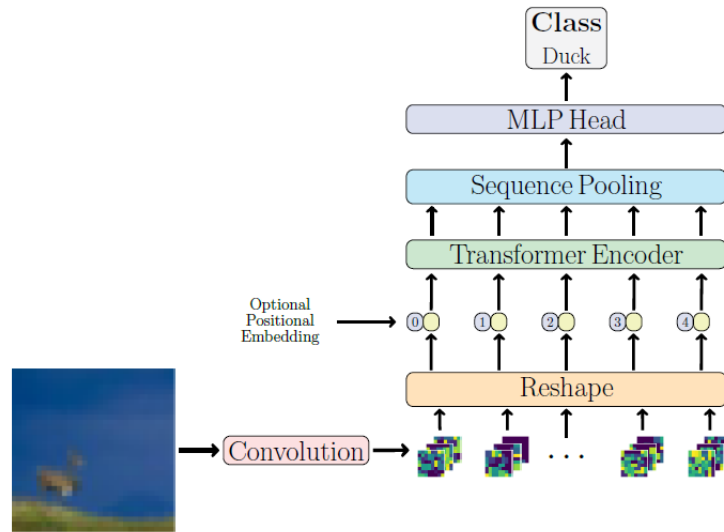


Figure 4.7: CNN plus Transformer architecture - taken from [98]. The transformer encoder is equal to the one in Figure 4.6.

The other training process that was tested was multi-task learning (MTL). This was done by training a network with two heads, for two different tasks, while sharing the same core. The goal with this was to test if there were relevant features that were shared between more than one dataset, since, if that was the case, then that would strengthen the claim that DL could be used as a viable tool for backscatter classification. The second dataset used, for the second classification task, was a dataset that contained the outcome of patients that had been afflicted with a respiratory disease - there were three available classes, which were mild symptoms, hospital stay required, hospital stay at an intensive care unit required.

4.6 Outputs

Several datasets were available throughout this thesis. Earlier datasets had three classes available, which were control patients (X- -), patients with an intermediate level of ND (-X-), and patients with ND (- -X). In this scenario, it was possible to do three types of binary classification: by either considering both the control and the intermediate class as the class 0 (001), by considering the intermediate class and ND as class 1 (011) or by not using the intermediate class and considering only the control and ND patients (0-1). Lastly, multiclass classification was possible, considering the classes independently (012). The most recent ND dataset, which had slight changes to the acquisition setup, only contained two classes, control and ND afflicted patients.

4.7 Conclusion

Throughout this chapter the methodologies developed were discussed - the same structure will be used for the discussion of the results obtained. Two main input types were used, the temporal and frequency representation of the backscattered signal, to assess which representation would be the most viable. Regarding network architectures, several models were developed, with distinct architectures, to understand which type of model would perform the best. Similarly, several DA methods were developed, as there was little information in the literature about DA methods for backscattered signals. Finally, different training processes and classification types were developed in order to assess which methods were more applicable, and how would models behave in different boundary-defining scenarios.

Chapter 5

Results and Discussion

5.1	Introduction	61
5.2	Dataset Split	62
5.3	Model Selection	62
5.4	Input Types	63
5.5	Data Augmentation	64
5.6	Model Architecture	67
5.7	Training Methods	69
5.8	Outputs	70
5.9	Patient Feature Networks	70
5.10	Comparison to ML methods	71
5.11	Estimate of Value Added	74
5.12	Conclusion	75

5.1 Introduction

Throughout this chapter, results obtained for different configurations of the different steps found in a DL pipeline, following what was mentioned in Chapter 4, will be presented and discussed. The discussion will consider the best model as the best model overall, considering that a good model has simultaneously high sensitivity and high specificity. However, considering the ND use-case mentioned in the introduction, if the goal was to detect all unhealthy individuals and detect as many healthy individuals as possible, then the best model for that scenario would be the one where the sensitivity corresponds to the maximum value possible, *i.e.* 1, and the specificity is as high as possible, without sacrificing sensitivity. Why the perspective that was used was the first one mentioned is because one of the goals of this dissertation was to explore DL methods for backscatter classification in general, *i.e.* methods that could be used for future applications - and, to that sense, it was found to be more interesting to consider the more common and general

perspective of what makes a model good or not. Nevertheless, an assessment of what model would be best for the ND drug-development ecosystem use case is done, considering the market value of the models developed. At the end of the chapter, a comparison to the currently available machine learning methods will be made.

5.2 Dataset Split

The dataset was split into three subsets: the training set, the validation set, and the test set. The model was trained on the training set, and, throughout the training epochs, metrics for the validation set were calculated. The model which had the best validation set accuracy was the one chosen to use on the test set - the reason for this will be presented shortly. Regarding the split itself, both the test set and validation set equaled 25% of the dataset, with the training set corresponding to the remaining data, *i.e.* 50%. Two datasets were used, and in the main one, the one used for most of the experiments except for the ones that tested different output scenarios there were a total of 261 patients, and for the second there were a total of 115 patients - this meant that to have a representative test set, the fraction of the dataset dedicated to this set had to be reasonably large, or, at least, larger than what is commonly used in larger datasets. After defining the test set to be 25%, the validation set was chosen to be 25% following the same logic that was used when selecting the test set fraction, and also because this allowed to split the remaining dataset, after the test set had been chosen, into three equal parts, which would be used as three different folds. Unfortunately, not all experiments were carried out for all three folds, as there was not enough time to do so - in some situations, only the results for one of the folds are presented (in that case, the fold used was chosen without any specific intent). For the main dataset, there were only two classes available, with 126 patients belonging to class 0, and 135 patients belonging to class 1. For the second dataset, there were three classes, healthy, intermediate ND (IND), *i.e.* an intermediate stage, and ND, each one having, respectively, 51, 18, and 46 patients.

5.3 Model Selection

Table 5.1: Results for different model stages throughout training.

	Accuracy	Accuracy per Patient	AUC	F1-Score	Precision	Sensitivity	Specificity
Best Accuracy	60.96±1.08	63.13±0.88	60.87±1.21	66.99±2.31	59.06±0.24	77.54±5.89	43.53±4.07
Best AUC	56.63±1.60	62.63±4.37	60.26±1.84	58.92±1.36	57.26±1.68	60.72±2.32	52.32±3.90
Best Loss	59.29±0.71	61.62±2.32	61.85±0.86	63.94±3.27	58.54±0.88	70.97±9.06	47.03±8.57

Selecting the model according to the best validation set accuracy did not always lead to the best model for the test set - however, it was found to be very consistent, and, except for the specificity, it was the one that presented the best results, which is why it was the one that was considered in the following experiments for the test set results. This choice is nevertheless debatable since the model chosen according to best AUC led to having better specificity results, but that had a significant sensitivity cost, which was the main reason against it. Another three models, *i.e.* three

supposedly different moments throughout training, were considered, the model with the best-balanced accuracy, the model with the best per-patient accuracy, and the last model - however, these models were also the model with the best AUC, which is why they are not present in the Table 5.1. In the future, it may be interesting to explore at what moment should the model be saved to be used on the test set - often some moments led to the best results for a certain fold while having the worst results for another fold - this may mean that a new metric can possibly be created and used as the best indicator for when to save the model.

5.4 Input Types

Regarding the different types of inputs, it is quite clear that the frequency domain of the signal leads to a better overall performance than the 1D input - please see Table 5.2. Although the accuracy per patient is only around 5% higher, note that the imbalance between the sensitivity and specificity when using the 1D input is quite large - in fact, when no initial processing is used, *i.e.* the raw backscattered signal is used as the input, the model mostly predicts class 1 for the test samples. Initial processing seems to be useful for the 1D, as it diminishes the gap between the sensitivity and specificity, whereas for the spectrogram it seems to have a small negative effect - notice that using the initial processing leads to, in general, slightly lower metrics, and a higher standard deviation for the per-patient accuracy - this result seems to indicate that, perhaps, filtering frequencies lower than 500 Hz may not be beneficial when using spectrograms. Why using the frequency domain of the signal leads to better results is not immediately clear - it seems that the characteristics of the 1D input do not allow for an easy extraction of features when compared to spectrograms. It can be hypothesized that this is because the 1D input is more complex and requires more data and a much larger model to learn said data - why it is more complex is also not exactly clear; perhaps, since each 1D input sample has a much larger size, in terms of data points than the spectrogram, there may be issues extracting what is most relevant, as there is, assuming a similar noise to signal ratio between both inputs, in terms of absolute values, more noise in the 1D input. Nevertheless, these results strengthen the case for inputs that result from backscatter transformations - *i.e.* it seems that it is worth exploring different inputs other than the raw backscatter, at least while the number of available samples remains in the same order of magnitude.

Table 5.2: Results obtained using different types of inputs.

	Accuracy	Accuracy per Patient	AUC	F1-Score	Precision	Sensitivity	Specificity
SG w/o IP	60.96±1.08	63.13±0.88	60.87±1.21	66.99±2.31	59.06±0.24	77.54±5.89	43.53±4.07
SG w/ IP	59.57±1.28	63.13±3.15	60.37±0.62	65.73±2.22	58.05±0.51	75.84±4.94	42.48±2.59
Multiple SG (w/o IP)	60.91±1.43	63.13±2.31	60.90±0.89	66.47±3.42	59.21±0.05	76.14±8.67	44.90±6.20
Multiple SG stacked (n=3, w/o IP)	58.41±2.37	58.08±2.32	58.99±1.61	63.96±2.26	57.51±1.83	72.07±3.31	44.06±2.86
1D w/o IP	56.31±2.23	57.07±3.50	54.30±3.87	68.81±2.54	54.20±1.10	94.30±6.24	16.40±2.42
1D w/ IP	56.40±1.80	57.58±3.03	55.24±1.13	65.43±4.16	55.08±1.26	81.48±13.66	30.08±12.92
1D+SG w/o IP	57.52±0.51	57.58±2.63	55.77±0.73	70.21±0.20	54.79±0.33	97.69±0.65	15.32±1.59

w/o - without; w/- with; IP - initial processing;

When comparing multiple spectrograms without initial processing to a single, medium-sized, spectrogram, also without said processing, there does not seem to be a very significant difference - all of the metrics are very similar, although using multiple spectrograms seems to lead to a model with a better specificity, which indicates that it might be valuable to further explore different frequencies and different resolutions for different frequency bands. When using multiple stacked spectrograms, the results are, except for the specificity, noticeably worse. The most obvious reason is that stacking n spectrograms leads to a reduction of the dataset by a factor of n , which should worsen the overfitting (in this experiment, n was equal to three). It is difficult to assess if stacking is, as a process, not useful, or if it really was just a case of having a smaller sample size. It may also be the case that a single segment of data, *i.e.* the approximately six seconds that were used, may contain enough information to model the data as accurately as possible with a given model. Nevertheless, considering the sample size that is available at the moment, the main conclusion is that it is better to not use stacked spectrograms.

Regarding combining both types of inputs, *i.e.* both the 1D and spectrogram input, without initial processing, the result was not very good - it seems that the features from the 1D input core model overwhelmed the spectrogram input core features, as the results are quite similar to the ones obtained for the 1D input without initial processing. Perhaps if the features were normalized before being concatenated, the result would have been different - this may be something to explore in the future.

Overall, all inputs tend to have a higher sensitivity and a lower specificity. The input that led to the highest specificity was the multiple spectrograms without initial processing, which was 44.90%, lower than 50%, meaning that it more often than not wrongly predicts the class of a sample that belongs to class 0. As mentioned in the introduction, if the perspective of this analysis was to value a model with a reasonable level of specificity without compromising sensitivity, then the best model would have been the one trained on the 1D input without initial processing.

5.5 Data Augmentation

Regarding general image transformations found in Table 5.3, rotations, random crops, Gaussian noise, Gaussian blur, and horizontal flips showed an improvement *w.r.t.* the per-patient accuracy - all of these DA methods showed either a slight improvement or were generally equal to the baseline *w.r.t.* most of the remaining metrics. It was only regarding the sensitivity and specificity where some of these methods did not perform as well, which usually occurred, without surprise, as a trade-off, *i.e.* if the sensitivity improved, then the specificity diminished, and the analogous for the reverse. The only DA that showed an improvement in either the sensitivity or specificity without sacrificing the specificity or sensitivity, respectively, was Gaussian noise. The methods that performed the worse were those where the gap between the sensitivity and specificity increased substantially, such as the scaled versions, the shears, and the translations.

Table 5.3: Results obtained using different general image transformations DA.

	Accuracy	Accuracy per Patient	AUC	F1-Score	Precision	Sensitivity	Specificity
No DA (Baseline)	60.96±1.08	63.13±0.88	60.87±1.21	66.99±2.31	59.06±0.24	77.54±5.89	43.53±4.07
Rotations	61.54±0.75	64.65±0.87	60.68±1.25	67.91±1.76	59.29±0.39	79.58±5.06	42.59±4.11
Scaling	59.50±0.88	60.61±1.52	59.48±1.48	64.72±1.89	58.42±0.20	72.64±4.50	45.71±2.92
Shears	57.20±7.92	60.61±10.50	62.58±5.05	52.50±28.17	55.44±6.38	59.09±40.26	55.20±26.68
Random Crop	61.16±1.26	63.64±0.00	61.44±1.19	66.82±2.77	59.36±0.16	76.64±7.03	44.90±4.84
Translations	61.51±0.73	61.62±3.81	60.11±0.76	68.33±1.52	59.22±0.28	81.18±4.09	40.84±2.84
Gauss noise	61.47±1.01	63.64±0.00	61.38±1.17	67.15±2.42	59.58±0.29	77.14±6.44	45.01±4.84
Gauss blur	60.84±1.28	64.14±0.87	62.20±0.85	67.37±1.74	58.75±0.72	79.01±4.01	41.75±2.65
Horizontal Flips	60.94±1.60	64.14±0.87	61.12±1.01	66.95±3.27	59.00±0.32	77.68±8.03	43.36±5.17
Vertical Flips	60.96±1.93	63.13±2.31	62.63±1.10	66.63±3.24	59.18±0.73	76.44±7.31	44.69±3.80

Curiously, considering the per-patient accuracy¹, the method that performed the best, for general image transformations DA, was rotations - this was the only method that performed well that also made a hard transformation regarding the orientation of the axis of the spectrogram. Notice that all of the methods with positive effects other than this one made no changes *w.r.t.* the orientation of the axis, *i.e.* they were only per-pixel transformations, except for the random crops. If it were not for the relative success of rotations and random crops, perhaps it could be said that maintaining the placement of the axis would be pivotal, but, since that is not the case, it is difficult to assess why some transformations were successful and other were not, as there does not seem to be any evident pattern. Furthermore, vertical flips only had a small negative impact on the sensitivity of the model, while having a positive impact on the AUC - this seems to indicate that either the model learns patterns placement *w.r.t.* reference points in the spectrogram, such as certain constant frequency bands (harmonics of the frequency of the modulation signal), or that the vertical orientation is not relevant to the structure of the data. A more human-bound perspective would perhaps say that it must be the former. Ultimately, it is very difficult to assess why some methods worked, whereas others did not, with the currently available information. Progressing into the future, it would be interesting to assess what the models find most relevant in a spectrogram, through existing methods, such as explainable artificial intelligence, to better interpret these DA results, and to better design and implement future DA.

Nevertheless, there were not any DA methods that showed a remarkable improvement - it was only the horizontal flips and the Gaussian noise that showed an improvement to some metrics while remaining equal regarding the remaining ones. Besides these, rotations, random crops, Gaussian Blur, and vertical flips had some improvements for some metrics while having a decrease for one other metric, which was considered acceptable, so they were, in conjunction with the horizontal flips and Gaussian noise, the general image transformations DA that were chosen to be part of the “final” data augmentation. The remaining all showed a significant decrease *w.r.t.* some metrics - the scaled versions, in particular, seem to have, consistently, the worst performance, which may indicate that changes of scale may affect the relevant patterns. Perhaps when the spectrogram is downsampled, some patterns are lost. Shears seem to be a very weak DA method, however, note

¹The per-patient accuracy seems to be a reasonable metric to take into account as the class imbalance is not too strong.

the large standard deviation for several metrics - for fold number two, the model had a behavior contrary to what was common for all other methods and the baseline, where the specificity was quite large and the sensitivity was small, less than 50% - this revealed that, at least, with the current hyper-parameters used for shearing the spectrograms, this method was not consistent, but it does not seem to be worthy of completely discarding for the future. Perhaps, that may be evidence that supports that shearing may help reduce the gap between the sensitivity and specificity and help build a model that also has high specificity. For the other folds, it actually had a very positive impact, having the highest per-patient accuracy, when compared to the remaining DA methods, for those folds. Translations were similar to random crops, so it is curious that it was one of the DA methods that did not perform very well, having a significant specificity decrease - in fact, it is especially curious how the random crops augmented data led to an improvement of the specificity with a slight decrease of the sensitivity, which is the opposite of what happened with translations. This may have been due to the difference between the hyper-parameters that defined the intensity of these operations - although they have the potential to lead to the same transformation, the factor for horizontal translation was considerably higher than the one for vertical translation, which may have not been beneficial. It seems, then, that it would be interesting to further explore the effect of different intensities for these DA methods.

Table 5.4: Results obtained using different spectrogram-specific DA.

	Accuracy	Accuracy per Patient	AUC	F1-Score	Precision	Sensitivity	Specificity
No DA (Baseline)	60.96±1.08	63.13±0.88	60.87±1.21	66.99±2.31	59.06±0.24	77.54±5.89	43.53±4.07
Freq Mask	61.11±1.64	63.64±1.52	61.76±0.85	67.40±3.01	58.99±0.46	78.81±7.28	42.52±4.31
Freq Mask +	60.99±0.91	64.14±0.87	61.07±1.03	67.37±2.26	58.92±0.48	78.88±6.36	42.20±5.25
Time Mask	61.56±0.54	62.63±2.31	60.19±0.33	67.67±1.42	59.43±0.16	78.65±4.08	43.60±3.22
Time Mask +	61.55±0.98	62.12±2.62	60.51±0.97	67.66±2.04	59.42±0.14	78.71±5.47	43.53±3.78
Freq Warp	61.19±1.37	64.14±0.87	61.46±1.15	66.86±2.93	59.37±0.18	76.78±7.37	44.82±4.92
Freq Warp +	61.72±1.19	64.14±0.87	61.37±1.18	67.71±2.10	59.59±0.52	78.51±5.26	44.09±3.53
Time Warp	61.26±1.08	63.64±1.52	61.53±1.56	67.02±2.63	59.39±0.09	77.12±6.89	44.58±5.07
Time Warp +	60.37±1.71	62.12±1.52	59.84±1.24	65.96±2.92	58.83±0.64	75.24±6.82	44.76±3.81
Freq W. I.	59.44±1.63	61.62±2.32	59.84±1.88	64.52±2.74	58.41±0.99	72.23±6.31	45.98±4.39
Time W. I.	60.01±0.96	61.11±0.87	60.16±1.54	65.07±2.41	58.86±0.27	72.94±6.24	46.44±4.71

Freq - Frequency; + - stronger intensity; W. I. - warping with interpolation;

Regarding DA methods designed specifically for spectrograms, please see Table 5.4, all of the DA methods seem to either have a, at least, small positive effect, or a slight decrease of performance. Regarding frequency masks, it seems that smaller frequency masks are better than larger ones - however, the difference is very small, which questions the validity of this claim. This is the case for all of the different intensities that were tested, for each spectrogram-specific DA. Since these methods were specifically designed for spectrograms, as they do not affect the axis, it would have been somewhat expected that they would have led to a good performance - however, it was expected that they would have led to a better performance than the regular image transformations methods, which does not seem to be the case. Although this may be because the intensity of the DA was not fine-tuned, this seems to indicate that DA may only slightly increase the performance

of the models using the medium-sized spectrogram and that it is more important to experiment with different input types, or even different spectrogram sizes - *i.e.* the potential for the current in-use spectrogram has been reached. This is, however, a somewhat bold claim, and it would require more experiments to assess. Naturally, this plateau may simply be due to the network that was used, or the general training configuration that was used, and perhaps different architectures or training configurations, that were not tested in this dissertation, may also improve the results.

When combining all of the methods for the “final DA”, the ones chosen from general image transformations DA and all of the spectrogram-specific DA, the results were not significantly better than when using only one DA method at a time - this may indicate that using several DA methods may not have a compounding effect, please see the experiment with “several DA” in Table 5.10. Note that they were not applied concurrently - *i.e.* there was a random chance to choose between any of the available DA methods.

Table 5.5: Results obtained using horizontal flips DA for the 1D input (third fold).

	Accuracy	Accuracy per-patient	AUC	F1-Score	Precision	Sensitivity	Specificity
1D w/o IP	53.79	53.03	50.22	65.88	52.98	87.09	18.82
1D w/o IP w/ DA	54.97	54.55	50.48	66.59	53.71	87.58	20.71

w/ - with; w/o - without; IP - Initial Processing; DA - Data Augmentation;

Regarding horizontal flips for the 1D input, an improvement was observed, which may indicate that DA methods that do not alter the frequency domain of the 1D input may be beneficial. Nevertheless, the model still performed rather mediocrely - please see Table 5.5.

5.6 Model Architecture

Two types of models, other than the standard CNN, were tested for both the time domain and frequency domain inputs, which were a model with residual blocks instead of certain convolutional blocks, and a model whose main block was a residual one, *i.e.* a residual network. Considering the 1D input, a smaller network was also tested. The smaller network led to an overall worse performance - all of the metrics were worse except for the specificity. The significantly worse metrics were the sensitivity and f1-score, which is a reflection of the lessening of the gap between specificity and sensitivity, so it could be said that the smaller network is actually better than the standard one, as it has a more balanced performance. These results seem to indicate that exploring unusual kernel sizes and strides, or, in other words, adapting the hyper-parameters of the network to the backscatter data may be a relevant pursuit.

Other than the sensitivity, replacing some convolutional blocks with residual blocks seems to improve the performance of the model - in this case, specificity improved while not sacrificing most of the sensitivity. It seems, then, that perhaps using different types of blocks may be beneficial - in the future, transformer blocks could be added as well. Why were residual blocks, generally, an improvement may be related with the ability of residual blocks to improve the hierarchical relation between the features, *i.e.* instead of generating several layers of abstraction as

Table 5.6: Results obtained using different types of model architectures - experiments were only done for the first fold due to time constraints.

	Accuracy	Accuracy per Patient	AUC	F1-Score	Precision	Sensitivity	Specificity
1D standard (w/o IP) (baseline)	57.08	59.09	57.92	70.08	54.51	98.1	13.99
1D small (w/o IP)	54.97	59.09	53.69	61.52	54.72	70.27	38.90
1D residual blocks (w/o IP)	58.77	60.61	58.92	69.59	55.92	92.09	23.76
1D residual network (w/o IP)	56.21	59.09	57.74	67.45	54.46	88.58	22.18
Sg standard (w/o IP) (baseline)	59.90	63.64	62.23	64.49	59.02	71.07	48.16
Sg residual blocks (w/o IP)	59.59	63.64	60.06	70.25	56.48	92.89	24.82
Sg residual network (w/o IP)	58.15	62.12	56.19	67.39	56.09	84.38	30.60

w/o - without; w - with; IP - initial processing;

convolutional layers do, they generate fewer but more relevant layers of abstraction, since if new abstract layers are created which are not useful, the block may simply focus on the initial block input, due to the shortcut connection.

The completely residual network for the 1D input had very similar results to the regular CNN. This may indicate that, for a raw backscatter input, the result for a model which uses convolutional filters may not generally differ much from these, *i.e.* very high sensitivity and low specificity - it seems then that, considering the currently available data, the case for focusing on input transformations is quite strong, as that seems to be what lead to greater differences in a model's performance.

Regarding the spectrogram input, using residual blocks leads to a widening of the gap between the sensitivity and specificity - the sensitivity becomes quite large, sacrificing some of the specificity. Even though the per-patient accuracy remains the same, there is a noticeable reduction of the AUC, which is to be expected, since the model is not as balanced as the standard convolutional network. This result seems odd since there does not seem to be any characteristic associated with residual blocks that may lead to a model having higher sensitivity and lower specificity. It may be hypothesized that the loss more easily decreases for the positive samples, which may have led the network to use the added parameters to overfit the positive samples, but that seems difficult to confirm with the currently available results.

The residual network for the spectrogram had a similar result to using residual blocks, although the difference was not as noticeable, *i.e.* the sensitivity did not increase as much as the sensitivity for the network with residual blocks, and the analogous happened with the specificity (smaller decrease when compared to the baseline). The same idea that was presented for the residual blocks may be used for the residual network - more parameters available led to the model prioritizing a certain class, assuming that said class was easier to learn - this is just a hypothesis.

5.7 Training Methods

Regarding training methods, comparing between regular training and using contrastive learning, it seems that contrastive learning led to a model that generally performed worse - the only metric where there were significant improvements was *w.r.t.* specificity. It may be that, as contrastive learning does not allow for the core of the network to be changed according to the final predictions for each sample, the features that were extracted may not have always been the most relevant - this may be clearer if it is considered that there is a possibility that some segments of data may not contain the information that would allow them to be distinguishable from other classes, which may have led the model to trying to associate those samples with other samples that do have relevant information, leading it to focus on features that are not as relevant. Later results, with the feature-blocks from the network trained with contrastive learning were somewhat better - this may further confirm this hypothesis, as features from a single segment may not be sufficient, whereas combining features may provide a better representation of an individual.

Table 5.7: Results for different training methods - results that do not present standard deviation are from experiments that were only done for the first fold.

	Accuracy	Accuracy per Patient	AUC	F1-Score	Precision	Sensitivity	Specificity
Standard Sg (w/o IP)	59.9	63.64	62.23	64.49	59.02	71.07	48.16
Contrastive learning (w/o IP)	52.61	48.48	56.14	48.32	54.75	43.24	62.46
MTL (w/o IP, per patient metrics, second dataset)	-	54.54	60	61.54	60.00	63.16	42.86
Standard Sg (w/o IP, second dataset)	60.49	56.57	62.43	70.37	80.65	63.77	49.19
1D+SG Pre-trained cores (w/o IP)	55.06±0.74	54.55±1.52	56.61±2.16	69.32±0.12	53.30±0.46	99.09±1.09	8.79±2.67
1D+SG (w/o IP)	57.52±0.51	57.58±2.63	55.77±0.73	70.21±0.20	54.79±0.33	97.69±0.65	15.32±1.59

w/o - without; IP - initial processing; MTL - multi-task learning;

Using pre-trained cores for the network that uses both the 1D and spectrogram input does not seem to lead to better results - perhaps this is still related to the fact that the features from the different cores are not normalized. However, this does not necessarily explain the really high sensitivity, and, naturally, the really low specificity - it seems that when using a 1D input class 1 is somehow much more relevant to the network than class 0. It may be because class 1 is slightly easier to model than class 0, and, since there are more class 1 samples, it may be prioritized - even though the loss is class-weighted.

Multi-task learning (MTL) seems slightly worse than regular training, which is not very surprising since the same core is being used for two different tasks (90% of the loss is calculated according to the main task, and 10% is calculated according to the secondary task, which was classifying samples from respiratory disease afflicted patients between two classes, patients that

developed mild symptoms and patients that had to be sent to an intensive care unit). However, that the model still reaches some level of learning is quite interesting, and it may indicate that future applications may benefit from pooling data from different tasks to learn relevant backscatter features. The model, to some extent, working seems to further strengthen the case that information can indeed be stored in a backscatter signal and that this technology may have a significant impact in the future. To further test this, it would be interesting to use MTL using a different dataset, *w.r.t.* its nature, such as cifar-10², for the second task.

5.8 Outputs

Table 5.8: Results obtained for different types of classification (second dataset).

	Accuracy	Accuracy per Patient	AUC	F1-Score	Precision	Sensitivity	Specificity
Binary (001)	60.49±14.32	56.57±4.63	62.43±9.36	70.37±13.70	80.65±4.08	63.77±19.94	49.19±7.37
Binary (011)	55.01±2.99	51.52±0.00	54.29±2.72	42.19±5.92	56.58±4.80	33.70±5.78	75.46±0.48
Binary (0-1)	44.69±7.61	43.59±8.00	45.46±11.21	51.39±9.39	64.42±7.79	43.33±10.60	47.68±18.32
Multiclass (012)	31.86±10.63	26.43±11.08	49.52±10.62	28.84±9.98	29.72±11.13	31.39±9.99	65.35±5.33

Regarding different outputs, it seems that when the model is doing binary classification it is preferable to remove the intermediate class. This either indicates that the model used was not sufficiently strong to establish a boundary between the two classes when the difference between them was softer, or that there was not sufficient data that would allow for such a learning. Doing binary classification considering the intermediate class and the pathology class as one class seems to lead to a better performance - this may just be related to the data that is being used, as in, a patient with a mild form of ND has a backscatter that is more similar to one that is of an ND patient than of a healthy individual. It is also clear that multiclassification is harder than binary classification - after all, instead of one boundary, the models have to delineate two boundaries, and it may not have sufficient data to do so, which is what seems more likely.

5.9 Patient Feature Networks

The goal of the patient feature networks was to combine all of the features that had been extracted from all of the acquisitions of a single patient and to obtain a prediction based on that - the idea was that, perhaps, that would lead to a more solid prediction, since it would take into consideration different information from several acquisition locations. According to Table 5.9, that does not seem to be the case: the results for the CNN and the CNN block followed by a transformer block (CNN+Transformer) both had a very weak performance. It was only the transformer that, surprisingly, had a reasonable performance. According to [98], transformers require very large amounts of data, which is why they developed the CNN+Transformer, which was the one that was tested, so the expected was that perhaps the transformer would only work if accompanied by an

²Well known dataset, contains regular images of objects such as vehicles.

early convolutional block. Thus, it does not seem to be the case that the worse performance, when compared to the baseline, may be only related to the lack of sufficient data, which probably plays a part, as there are fewer samples, but it may also be significantly related to the actual structure of the network - the transformer immediately uses the self-attention mechanism between the extracted features for each segment, to explicitly compare features from different segments, whereas that is not the case with the other networks, which may explain the observed difference in the results.

Table 5.9: Results obtained using different patient feature networks. When standard deviation is not present, the results were obtained for the first fold only, due to time constraints.

	Accuracy per Patient	AUC	F1-Score	Precision	Sensitivity	Specificity
SG w/o IP (Baseline)	63.13±0.88	60.87±1.21	66.99±2.31	59.06±0.24	77.54±5.89	43.53±4.07
Transformer	60.61±2.62	61.67±3.51	66.23±4.02	59.18±1.15	75.49±8.99	44.79±4.77
CNN+Transformer	49.49±2.31	52.67±5.69	58.99±8.14	50.63±1.46	73.53±24.07	23.96±23.45
CNN	52.53±6.12	48.00±5.57	57.16±9.05	52.88±4.67	62.75±14.80	41.67±4.77
SG w/o IP (Baseline)	63.64	62.23	64.49	59.02	71.07	48.16
CNN+Transformer (Contrastive features)	59.09	62	55.74	62.96	68.75	50.00
Transformer (Contrastive Features)	63.64	65	65.71	63.89	59.375	67.65

w/o - without; IP - initial processing;

The features from the network trained according to the supervised contrastive loss were also tested with the patient features network, for the first fold. The results from the CNN were not good, whereas the results for the CNN+Transformer, and the transformer, were reasonable - notice how using the former, when comparing it to the baseline, the results seem more balanced, with a specificity reaching 65%, while only compromising some of the sensitivity. This may mean that the most balanced results may be obtained from supervised contrastive learning followed by passing the feature blocks to a CNN+Transformer network. If the comparison were to be made *w.r.t.* the results obtained from supervised contrastive learning, then it seems that the CNN+Transformer improved the sensitivity of the network. The transformer seems to improve all of the metrics except the specificity, and, curiously, it leads to a performance similar to the baseline, with a much higher sensitivity. It seems, then, that these networks were able to extract information from the relationship between the features, as long as they are obtained from contrastive learning - note, however, that these conclusions are not very sound since the results are only for one fold. It seems then, that perhaps some relevant features were extracted from the supervised contrastive learning training method.

5.10 Comparison to ML methods

Comparing the results obtained using the standard CNN model with a single spectrogram without initial processing as its input with frequency warping DA (DL-model1) to an SVM (ML), it

Table 5.10: Comparing the results obtained to the ones obtained using ML methods (SVM) - per patient metrics.

	Sensitivity	Precision	Specificity	Accuracy	F1-Score	AUC
SVM (ML)	64.29	60.00	47.83	56.86	62.07	58.70
SG w/o IP standard CNN Frequency warping DA (DL)	81.37	61.53	45.83	64.14	69.95	61.45**
SG w/o IP standard CNN Several DA (DL)	84.31	61.00	42.71	64.14	70.75	60.08**

w/o - without; IP - initial processing; ** - not per patient;

seems that DL may indeed be a valuable tool to explore regarding the analysis and classification of backscatter, as it obtains better results for most metrics, especially for the sensitivity, which is reflected in other metrics, such as the f1-score, and, for the metrics where DL performs worse, the specificity, the decrease in performance is not drastic, although noticeable, two percentage points. Curiously, several DA methods simultaneously (rotations, random crops, Gaussian Blur, horizontal flips, vertical flips, frequency masks, time masks, frequency warps, time warps, frequency warps with interpolation, time warps with interpolation) did not have a cumulative effect - note that not all DA methods were used simultaneously for each sample, but instead, there was a random choice between the available methods, since the effect of several DA methods applied concurrently to a single sample had not been manually verified, *i.e.* there was a concern that multiple simultaneous DA methods per sample would not be viable, and, considering the number of DA methods available, there were too many possible combinations to manually confirm or deny this. The train/test split done for the ML method was 80%/20% - *i.e.* 20% of the dataset was used as the test set, which means that both test sets have a similar number of patient samples, and therefore both methods may adequately be compared. Below follows how some metrics changed throughout training for the standard CNN with the medium-sized spectrogram with initial processing as its input with frequency warping data augmentation for the third fold.

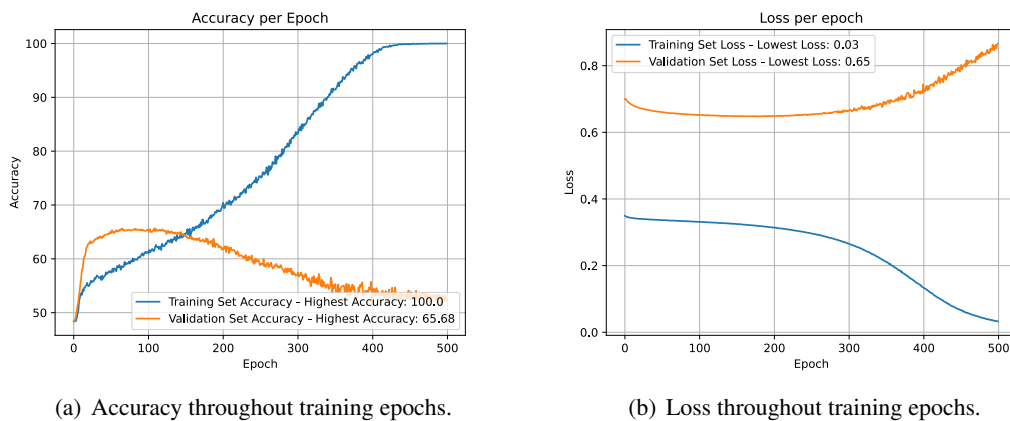


Figure 5.1: Accuracy and Loss.

Accuracy throughout training seemed reasonable - note, however the odd sharp increase during early epochs of the validation accuracy, please see Figure 5.1. This may be because the validation set contained easier samples - this was not the case for other folds, where the validation accuracy always either accompanied the training accuracy or was lower than the training accuracy. The model is capable of overfitting the training set, and, even with some DA, it is clear that there is still a lot of overfitting, indicating that, perhaps, fine-tuning DA should be one of the priorities in further experiments. Regarding the loss, the validation loss remains steadily higher than the training loss, and, at some point, it starts to increase, after decreasing, as expected. The increase of the validation set loss occurs when the training set loss most rapidly decreases, which is when memorization is most likely the strategy of the model to lower the loss.

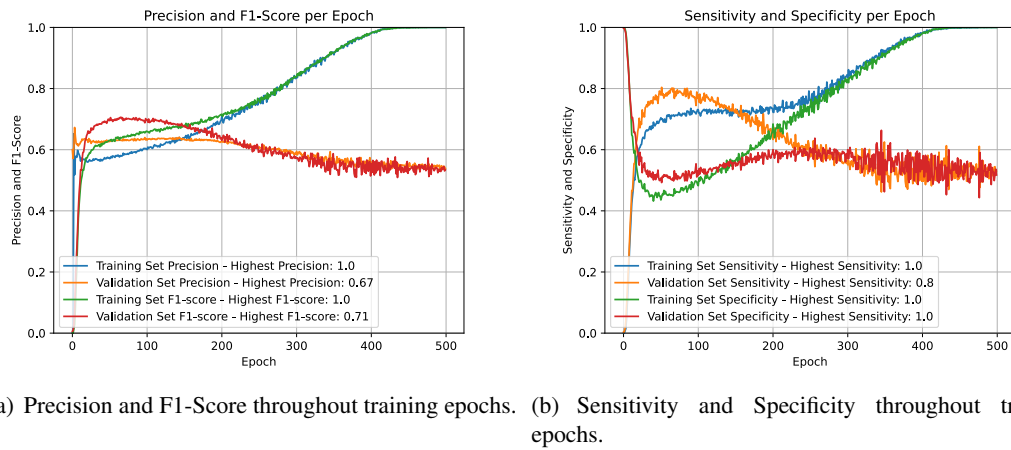


Figure 5.2: Precision, F1-Score, Sensitivity, and Specificity.

Regarding the performance of the model *w.r.t.* both classes, please see Figure 5.2, it seems that early on in the training, class 1 is favored to be more easily learned, perhaps because the class 1 samples are more easily distinguished, or easier to model. The reasons for this in terms of the data structure are not clear, however, from a biological and theoretical point of view, it does not seem so surprising, as there is a clear difference between the medium of a sample that was taken from a person with ND and the medium of samples that were taken from healthy people. Perhaps the class 0 intra-variability is larger, as a connecting point between several different healthy individuals may be more difficult to establish - consider that healthy individuals most likely have different mediums. This is, however, just a hypothesis. Towards the end of the training stage, the sensitivity and specificity converge to values slightly higher than 50%, which, although not remarkable, is an indicator that the model can distinguish both classes simultaneously, with equal performance for both.

Analyzing the ROC curve and the confusion matrix, please see Figure 5.3, it is evident the tendency of the model to, in a certain way, prioritize class 1 samples over class 0 ones. For a low false positives rate, the true positive rate is practically equal to random - a reasonably large false positive rate is required for a decent true positive rate.

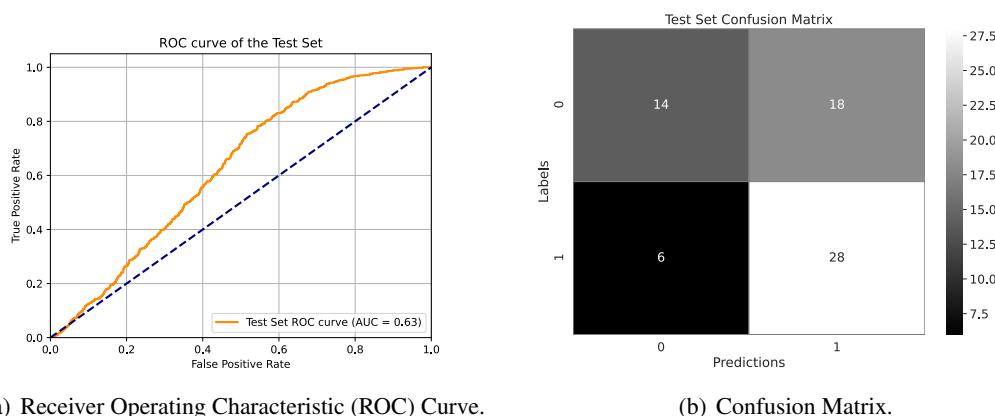


Figure 5.3: ROC curve and Confusion Matrix.

5.11 Estimate of Value Added

According to ClinicalTrials.gov, there are currently circa 700 recruiting/not yet recruiting and circa 100 active not recruiting clinical trials for ND [99, 100], totaling circa 800 trials. According to surveys done to physicians by Company X, circa 80% of individuals are not suitable for trials, meaning that each trial screens hundreds of possible candidates. Assuming that 50 individuals are required for phase 1 trials (estimate from FDA [101]), there were circa 250 possible candidates per trial, totaling 200,000 individuals for the 800 trials. Assuming that 20% of these had ND, using the DL-model1, which has 81.37% sensitivity, 32,548 of the 40,000 apt candidates could have been accurately detected. Analogously, considering the 45.83% specificity of DL-model1, 73,328 candidates of the 160,000 not apt candidates would have been detected, leaving 86,672 candidates to be ruled out by conventional methods. Assuming the cost of circa 5000\$ mentioned in the introduction for the assessment of the aptness of a candidate, using the Company X method with DL-model1, which would cost in the order of magnitude of hundreds of dollars, let us assume 500\$, the costs associated with detecting not apt candidates could have been reduced by 266 million \$ for the mentioned trials, or 35.5 million € per year, assuming that a drug trial lasts on average 7.5 years [102] - these are, of course, rough estimates. If, instead of DL-model1, the model with the 1D input without initial processing was to be used, then most of the unhealthy and apt candidates would be found, but almost none of the healthy, not apt candidates, would be ruled out using Company X's technology, which would not be very useful. As the goal here is not to diagnose individuals, a lower sensitivity could perhaps be considered, in order to have a higher sensitivity. The question would then be what would be the value of the circa 7.5K unhealthy candidates that would be considered not apt for the drug trials, despite actually being apt for the trials - perhaps it can be considered that a larger candidate pool would have to be required in order to find the ideal number of 40K candidates. Assuming the 81.37% mentioned sensitivity, and that only 20% of candidates are apt, then the candidate pool would have had to be 245,791 candidates, with 49,158 unhealthy apt candidates. In that scenario, circa 196,633 individuals

would be not apt candidates, and of those, considering the 45.83% specificity of DL-model1, 90,117 would be detected using Company X's technology, leaving 106,516 persons to be ruled out using conventional methods, an added 19,884 ones, which means an added cost of circa 99 million \$ - which would have then meant a 167 million \$ reduction of the costs of detecting not apt candidates for drug trials, or 22.3 million \$ per year.

5.12 Conclusion

In conclusion, in this chapter results from the developed methodologies and their discussion was presented. Using a standard CNN on the frequency domain of the backscatter with DA seems to be the combination that leads to a model with the best performance, considering the goal where DL was being explored and tested for backscatter classification. Considering the ND-specific goal, if the desired method would be one that detects all of the unhealthy people while removing, from the pool of possibilities, as many healthy individuals as possible, then the best model for that scenario would be the one that used the 1D input without initial processing, as it is the one with the highest sensitivity and some specificity. However, if, for the ND-specific goal, the method with the highest market value would be the desired one, then that would perhaps not be the case. For different goals, different methods arise as the most suitable - nevertheless, it seems clear that DL may be a viable tool for the analysis of backscatter data.

Chapter 6

Conclusions

6.1 Overview	77
6.2 Contributions	79
6.3 Future Work	79

6.1 Overview

In this dissertation, a literature review of the state of the art of signals similar, in their structure, to the backscattered one, such as physiological signals, was made to understand the latest DL developments for the analysis of 1-dimensional data. From there, several methodologies were developed for the main steps in a DL pipeline, *i.e.* different types of inputs, DA, model architectures, training processes, output configurations, and even different types of post-output processing networks. Most of these methods were then tested extensively, others in a shallower way, due to time constraints, to obtain results that would allow for the understanding of their effectiveness, and ultimately, in order to achieve the proposed goals in the introduction. These goals, and how they were attained, are the following:

1. Exploration and development of DL methodologies for backscattered signal analysis:

As mentioned, several methodologies were developed for each essential stage in a DL pipeline. Regarding the types of inputs that could be used, from two main types, which were the temporal and frequency representation of the backscattered signal, five setups were created, which were the 1D input, the spectrogram, the multiple spectrograms, the stacked single-size spectrograms, and the 1D input in conjunction with the spectrogram. Each of these inputs could undergo an initial processing stage - this was tested for the 1D and spectrogram input. The main conclusion here was that spectrograms are most likely a better input than the raw backscattered input. This means that, in the future, it will be interesting to explore more forms of inputs, as they seem to be a key factor.

The spectrogram input data could be augmented using fifteen different methods, nine of which corresponded to general image transformations, which are used in several computer vision tasks, and six of which were spectrogram-specific DA methods. The 1D input could be augmented using horizontal flips. Some of the general image transformations seem useful, others ought to be tested more thoroughly; all of the spectrogram-specific DA were, to a certain extent, successful, although more thorough testing also seems likely to produce a further understanding of the data and these methods. No DA method proved to be exceptionally remarkable, but some presented noticeable improvements.

Regarding model architectures, besides testing standard CNNs, CNNs with residual blocks, and residual networks were tested - the main conclusion was that CNNs seem to be a good starting point, and different model architectures have a significant, although not necessarily positive, impact on the performance of the model. Residual blocks and residual networks did not have a better performance than the standard CNN. Nevertheless, it would not be rigorous to conclude that standard CNNs are better than residual networks when using backscattered signals since not all folds were tested - still, these networks provided a possibility to further understand the data and how it can be modeled.

From the different training processes tested, it was possible to understand that currently, the standard training is the most preferred method to deploy, as different methods yielded worse results. Nevertheless, better results were not always the goal, such as when MTL was used, which allowed for the strengthening of the case for the future use of backscattered data as a way to assess a medium composition, as it demonstrated that two different datasets, obtained using the same acquisition method, may share key features.

Regarding output classification types, the main conclusions are that there is not sufficient data, at the moment, to expect a DL model to be able to define more than one boundary in the feature space. Furthermore, for the ND dataset in use, it seems that combining the intermediate class with the pathology class yields better results than combining the intermediate class with the healthy class - which, intuitively, is sensible.

Post-output, feature-combining, models did not improve on the results of the simplified outputs - the most likely reason seems to be the reduction of the size of the dataset - notwithstanding, the transformer was able to obtain results similar to the best performing model, which demonstrates that combining the information from different locations can account for a dataset size reduction. Curiously, using these feature-combining models, showed that contrastive learning may indeed be a useful learning process - more tests ought to be made to clarify this claim.

2. Development of a DL model able to detect ND patients:

The DL model that performed the best with the ND dataset outperformed, for some metrics, the ML model currently in use. For some metrics, the improvement was not very large, however, it seems that the DL model is substantially more sensitive, while not sacrificing

a lot of specificity, which allows for a better detection of non-healthy individuals. This model, after extensive testing that was not done in this dissertation, could perhaps be used in a real-world scenario, allowing for cost savings in the order of millions of dollars every year, according to rough estimates.

6.2 Contributions

The following are the contributions of this dissertation:

- Implementation of DL models that use different types of inputs for the analysis of biological samples' backscattered signals.
- Testing of several DA methods for the spectrogram input.
- Implementation of models with different architectures for backscatter classification: standard CNN, CNN including residual blocks, completely residual CNNs.
- Implementation of different training algorithms, which are, besides a standard training process, MTL, using pre-trained networks, and contrastive learning.
- Testing different classification tasks, as in, different binary tasks and a multiclassification task.
- Comparison of the best developed DL model with the currently in-use ML method, and assessment of the market value of this contribution.

6.3 Future Work

The following future work will be presented following the different steps in a DL pipeline, similar to what has been done before, as there are key future developments for most.

6.3.1 Inputs

The type of input used was shown to be absolutely determinant to how the network performed. Thus, it is pivotal to develop and test further types of inputs. Perhaps a model could be used to extract features and create a sort of spectrogram with the relevant information, enforcing an explicit focus on the frequency information of the raw backscatter. Additionally, the window size used to calculate the spectrogram should be further tested, and perhaps logistic regression could be used to determine the best window size to be used when calculating the spectrogram. Additionally, different algorithms should be tested, such as different wavelets, or using the discrete cosine transform. There are, essentially, several possibilities that seem worthy of exploration.

6.3.2 Data Augmentation

There are several forms of DA available - essentially, if it alters the input data without seemingly destroying its information, then, at this stage, it seems worthy of being tested. Of course, this would mean that there are endless forms of DA to be developed, which is not necessarily true, at least in the short-term future - further thorough testing of the developed DA seems to be an essential step to understand the limits of the current backscatter DA. Moreover, understanding what is relevant in a spectrogram will also likely aid in the development of DA that significantly improves the performance of the developed models. Furthermore, there is also the testing of combinations of DA methods, which is itself a demanding task, as there are several combinations. It seems, then, that the keyword for DA is testing.

6.3.3 Model Architectures

Several model architectures seem interesting to test, such as transformers for the initial inputs, instead of using them for the feature-blocks only - however, it seems especially interesting to test models that use unsupervised learning. These would most likely not have the best performances, but they would allow for a better understanding of the data, and they would be an interesting challenge. If a model can, in an unsupervised manner, accurately assess the different clusters in the ND dataset, then the case for the use of backscatter for pathology detection would be very strong.

6.3.4 Training Processes and Outputs

Currently, it seems interesting to further test the developed training processes. Contrastive learning on its own did not lead to great results, but when the features from contrastive learning of each patient were combined, the results were quite similar to the ones observed by the best performing models, trained in a standard manner. There is also further testing to be done regarding a better fine-tuning of the models' hyper-parameters, which is always an ongoing process.

6.3.5 Different classification tasks and feature-combining networks

Lastly, it seems clear that difficult backscatter classification tasks will require more data, which means that for the foreseeable future, it seems that focusing on binary classification using samples from healthy and non-healthy may be the preferred path until more data is acquired. The likely increase of the available dataset will also allow for a more adequate and fair testing of feature-combining networks, which, intuitively, seems advantageous, as they are combining information from different locations, which may overcome the possible issue of some locations presenting more relevant information than others. A larger dataset will also allow for the reduction of the test and validation sets fractions, which should increase the models' performance.

Appendix A

Extended results

In this appendix, a more extensive presentation of the results can be found, contained the discriminated per fold results, following the sections in Chapter 5, *i.e.* the types of inputs, DA, outputs, and patient-feature networks.

Table A.1: Results for the test set when choosing the model state according to different metrics.

Model Choice	Fold	Accuracy	Accuracy per-patient	AUC	F1-Score	Precision	Sensitivity	Specificity
Best Accuracy	Fold 0	59.90	63.64	62.23	64.49	59.02	71.07	48.16
	Fold 1	62.05	63.64	59.93	69.04	59.31	82.58	40.48
	Fold 2	60.92	62.12	60.44	67.44	58.84	78.98	41.96
	Avg.	60.96	63.13	60.87	66.99	59.06	77.54	43.53
	STD	1.08	0.88	1.21	2.31	0.24	5.89	4.07
Best AUC	Fold 0	58.46	65.15	62.39	60.21	59.11	61.36	55.41
	Fold 1	55.48	57.58	59.21	59.05	55.84	62.66	47.94
	Fold 2	55.94	65.15	59.18	57.49	56.84	58.15	53.62
	Avg.	56.63	62.63	60.26	58.92	57.26	60.72	52.32
	STD	1.60	4.37	1.84	1.36	1.68	2.32	3.90
Best Loss	Fold 0	59.89	63.64	62.23	64.48	59.01	71.07	48.15
	Fold 1	59.48	59.09	62.46	66.91	57.52	79.97	37.96
	Fold 2	58.51	62.12	60.87	60.44	59.08	61.86	54.99
	Avg.	59.29	61.62	61.85	63.94	58.54	70.97	47.03
	STD	0.71	2.32	0.86	3.27	0.88	9.06	8.57

Table A.2: Results for different types of spectrogram inputs per fold.

Input type	Fold	Max Validation Accuracy	Accuracy	Accuracy per-patient	AUC	F1-Score	Precision	Sensitivity	Specificity
SG without IP	Fold 0	55.62	59.90	63.64	62.23	64.49	59.02	71.07	48.16
	Fold 1	60.15	62.05	63.64	59.93	69.04	59.31	82.58	40.48
	Fold 2	65.57	60.92	62.12	60.44	67.44	58.84	78.98	41.96
	Avg.	60.45	60.96	63.13	60.87	66.99	59.06	77.54	43.53
	STD	4.98	1.08	0.88	1.21	2.31	0.24	5.89	4.07
SG with IP	Fold 0	55.99	58.1	60.61	60	63.18	57.46	70.17	45.43
	Fold 1	59.88	60.26	62.12	60.03	66.83	58.37	78.18	41.43
	Fold 2	65.89	60.36	66.67	61.08	67.18	58.33	79.18	40.59
	Avg.	60.59	59.57	63.13	60.37	65.73	58.05	75.84	42.48
	STD	4.99	1.28	3.15	0.62	2.22	0.51	4.94	2.59
Multiple SGs without IP	Fold 0	55.73	59.38	63.64	61.81	62.75	59.18	66.77	51.63
	Fold 1	59.72	62.21	65.15	60.04	69.46	59.26	83.88	39.43
	Fold 2	67.29	61.13	60.61	60.84	67.21	59.18	77.77	43.64
	Avg.	60.91	60.91	63.13	60.90	66.47	59.21	76.14	44.90
	STD	5.87	1.43	2.31	0.89	3.42	0.05	8.67	6.20
Multiple SGs without IP stacked (n=3)	Fold 0	56.09	60	60.61	60.84	64.86	58.97	72.07	47.31
	Fold 1	60.41	59.54	57.58	57.96	65.62	58.1	75.38	42.9
	Fold 2	64.69	55.69	56.06	58.17	61.39	55.45	68.77	41.96
	Avg.	60.40	58.41	58.08	58.99	63.96	57.51	72.07	44.06
	STD	4.30	2.37	2.32	1.61	2.26	1.83	3.31	2.86

IP - Initial Processing; Avg. - Average; STD - Standard Deviation;

Table A.3: Results for different types of 1D inputs per fold.

Input type	Fold	Max Validation Accuracy	Accuracy	Accuracy per-patient	AUC	F1-Score	Precision	Sensitivity	Specificity
1D without IP	Fold 0	57.14	57.08	59.09	57.92	70.08	54.51	98.1	13.99
	Fold 1	54.48	58.05	59.09	54.75	70.47	55.11	97.7	16.4
	Fold 2	62.4	53.79	53.03	50.22	65.88	52.98	87.09	18.82
	Avg.	58.01	56.31	57.07	54.30	68.81	54.20	94.30	16.40
	STD	4.03	2.23	3.50	3.87	2.54	1.10	6.24	2.42
1D with IP	Fold 0	58.59	57.17	57.58	56.17	69.91	54.62	97.1	15.25
	Fold 1	56.76	57.69	60.61	55.57	64.67	56.51	75.56	38.91
	Fold 2	62.6	54.35	54.55	53.99	61.7	54.11	71.77	36.07
	Avg.	59.32	56.40	57.58	55.24	65.43	55.08	81.48	30.08
	STD	2.99	1.80	3.03	1.13	4.16	1.26	13.66	12.92
1D+SG without IP	Fold 0	58.8	57.49	60.61	56.57	70.32	54.74	98.3	14.62
	Fold 1	54.27	58.05	56.06	55.58	70.32	55.15	97	17.14
	Fold 2	57.66	57.03	56.06	55.15	69.98	54.49	97.78	14.2
	Avg.	56.91	57.52	57.58	55.77	70.21	54.79	97.69	15.32
	STD	2.36	0.51	2.63	0.73	0.20	0.33	0.65	1.59

IP - Initial Processing; Avg. - Average; STD - Standard Deviation;

Table A.4: Results for regular image transformations DA for the spectrogram input for the different folds.

DA type	Fold	Accuracy	Accuracy per-patient	AUC	F1-Score	Precision	Sensitivity	Specificity
None	Avg.	60.96	63.13	60.87	66.99	59.06	77.54	43.53
	STD	1.08	0.88	1.21	2.31	0.24	5.89	4.07
Rotation	Fold 0	61.13	65.15	61.18	66.25	59.66	74.47	47.12
	Fold 1	62.41	65.15	61.61	69.75	59.34	84.58	39.11
	Fold 2	61.08	63.64	59.26	67.72	58.88	79.68	41.53
	Avg.	61.54	64.65	60.68	67.91	59.29	79.58	42.59
	STD	0.75	0.87	1.25	1.76	0.39	5.06	4.11
Scaled version	Fold 0	59.33	60.61	58.78	64.33	58.42	71.57	46.48
	Fold 1	60.46	59.09	61.18	66.78	58.62	77.58	42.48
	Fold 2	58.72	62.12	58.48	63.06	58.22	68.77	48.16
	Avg.	59.50	60.61	59.48	64.72	58.42	72.64	45.71
	STD	0.88	1.52	1.48	1.89	0.20	4.50	2.92
Shear	Fold 0	59.79	66.67	62.98	67.55	57.58	81.68	36.8
	Fold 1	48.31	48.48	57.34	20	48.26	12.61	85.8
	Fold 2	63.49	66.67	67.41	69.96	60.47	82.98	43.01
	Avg.	57.20	60.61	62.58	52.50	55.44	59.09	55.20
	STD	7.92	10.50	5.05	28.17	6.38	40.26	26.68
Random Crop	Fold 0	59.79	63.64	61.84	63.84	59.2	69.27	49.84
	Fold 1	62.26	63.64	60.11	69.33	59.38	83.28	40.17
	Fold 2	61.44	63.64	62.38	67.28	59.51	77.38	44.69
	Avg.	61.16	63.64	61.44	66.82	59.36	76.64	44.90
	STD	1.26	0.00	1.19	2.77	0.16	7.03	4.84
Translation	Fold 0	60.67	57.58	59.7	66.58	58.95	76.48	44.06
	Fold 1	62	65.15	60.98	69.16	59.19	83.18	39.75
	Fold 2	61.85	62.12	59.64	69.26	59.51	83.88	38.7
	Avg.	61.51	61.62	60.11	68.33	59.22	81.18	40.84
	STD	0.73	3.81	0.76	1.52	0.28	4.09	2.84
Gaussian Noise	Fold 0	60.31	63.64	61.63	64.5	59.53	70.37	49.74
	Fold 1	62.15	63.64	60.11	69.25	59.31	83.18	40.06
	Fold 2	61.95	63.64	62.41	67.71	59.89	77.88	45.22
	Avg.	61.47	63.64	61.38	67.15	59.58	77.14	45.01
	STD	1.01	0.00	1.17	2.42	0.29	6.44	4.84
Gaussian Blur	Fold 0	60.15	63.64	61.8	67.23	58.09	79.78	39.54
	Fold 1	62.31	65.15	63.17	69.18	59.52	82.58	41.01
	Fold 2	60.05	63.64	61.62	65.7	58.65	74.67	44.69
	Avg.	60.84	64.14	62.20	67.37	58.75	79.01	41.75
	STD	1.28	0.87	0.85	1.74	0.72	4.01	2.65
Horizontal Flip	Fold 0	59.18	63.64	62.19	63.28	58.68	68.67	49.21
	Fold 1	62.31	65.15	61	69.56	59.32	84.08	39.43
	Fold 2	61.33	63.64	60.18	68.02	59.01	80.28	41.43
	Avg.	60.94	64.14	61.12	66.95	59.00	77.68	43.36
	STD	1.60	0.87	1.01	3.27	0.32	8.03	5.17
Vertical Flip	Fold 0	58.87	63.64	61.38	63.21	58.34	68.97	48.26
	Fold 1	62.67	65.15	63.09	69.64	59.69	83.58	40.69
	Fold 2	61.33	60.61	63.43	67.05	59.5	76.78	45.11
	Avg.	60.96	63.13	62.63	66.63	59.18	76.44	44.69
	STD	1.93	2.31	1.10	3.24	0.73	7.31	3.80

Table A.5: Results for spectrogram-specific transformations DA for the spectrogram input for the different folds.

DA type	Fold	Accuracy	Accuracy per-patient	AUC	F1-Score	Precision	Sensitivity	Specificity
None	Avg.	60.96	63.13	60.87	66.99	59.06	77.54	43.53
	STD	1.08	0.88	1.21	2.31	0.24	5.89	4.07
Frequency Mask	Fold 0	59.33	63.64	62.33	64.04	58.54	70.67	47.42
	Fold 1	62.56	65.15	62.17	69.86	59.45	84.68	39.33
	Fold 2	61.44	62.12	60.79	68.29	58.99	81.08	40.8
	Avg.	61.11	63.64	61.76	67.40	58.99	78.81	42.52
	STD	1.64	1.52	0.85	3.01	0.46	7.28	4.31
Frequency Mask +	Fold 0	60.25	63.64	61.82	64.88	59.27	71.67	48.26
	Fold 1	62	65.15	61.5	69.29	59.12	83.68	39.22
	Fold 2	60.72	63.64	59.89	67.95	58.38	81.28	39.12
	Avg.	60.99	64.14	61.07	67.37	58.92	78.88	42.20
	STD	0.91	0.87	1.03	2.26	0.48	6.36	5.25
Time Mask	Fold 0	61.03	60.61	60.54	66.43	59.45	75.28	46.06
	Fold 1	62.1	65.15	59.88	69.22	59.27	83.18	39.96
	Fold 2	61.54	62.12	60.16	67.36	59.58	77.48	44.79
	Avg.	61.56	62.63	60.19	67.67	59.43	78.65	43.60
	STD	0.54	2.31	0.33	1.42	0.16	4.08	3.22
Time Mask +	Fold 0	60.87	60.61	60.64	66.01	59.47	74.17	46.89
	Fold 1	62.67	65.15	61.41	69.94	59.52	84.78	39.43
	Fold 2	61.12	60.61	59.48	67.04	59.26	77.17	44.26
	Avg.	61.55	62.12	60.51	67.66	59.42	78.71	43.53
	STD	0.98	2.62	0.97	2.04	0.14	5.47	3.78
Frequency Warp	Fold 0	59.74	63.64	61.83	63.77	59.16	69.17	49.8
	Fold 1	62.46	65.15	60.17	69.6	59.47	83.88	39.96
	Fold 2	61.38	63.64	62.38	67.21	59.48	77.28	44.69
	Avg.	61.19	64.14	61.46	66.86	59.37	76.78	44.82
	STD	1.37	0.87	1.15	2.93	0.18	7.37	4.92
Frequency Warp +	Fold 0	60.35	63.64	61.16	65.47	59.11	73.37	46.68
	Fold 1	62.51	65.15	60.31	69.63	59.51	83.88	40.07
	Fold 2	62.31	63.64	62.65	68.03	60.15	78.28	45.53
	Avg.	61.72	64.14	61.37	67.71	59.59	78.51	44.09
	STD	1.19	0.87	1.18	2.10	0.52	5.26	3.53
Time Warp	Fold 0	60.15	65.15	63.22	64.21	59.47	69.77	50.06
	Fold 1	62.31	63.64	60.14	69.41	59.4	83.43	40.06
	Fold 2	61.33	62.12	61.22	67.44	59.3	78.17	43.63
	Avg.	61.26	63.64	61.53	67.02	59.39	77.12	44.58
	STD	1.08	1.52	1.56	2.63	0.09	6.89	5.07
Time Warp +	Fold 0	59.94	62.12	60.82	64.8	58.93	71.97	47.31
	Fold 1	62.25	63.64	60.25	69.28	59.41	83.08	40.38
	Fold 2	58.92	60.61	58.45	63.8	58.15	70.67	46.58
	Avg.	60.37	62.12	59.84	65.96	58.83	75.24	44.76
	STD	1.71	1.52	1.24	2.92	0.64	6.82	3.81
Frequency Warp with Interpolation	Fold 0	59.85	63.64	61.85	63.9	59.23	69.37	49.84
	Fold 1	60.82	62.12	59.55	67.51	58.68	79.47	41.21
	Fold 2	57.64	59.09	58.12	62.14	57.31	67.86	46.89
	Avg.	59.44	61.62	59.84	64.52	58.41	72.23	45.98
	STD	1.63	2.32	1.88	2.74	0.99	6.31	4.39
Time warp with Interpolation	Fold 0	59.89	60.61	61.93	64.19	59.16	70.17	49.1
	Fold 1	61.02	62.12	59.43	67.79	58.78	80.08	41.00
	Fold 2	59.12	60.61	59.11	63.22	58.64	68.56	49.21
	Avg.	60.01	61.11	60.16	65.07	58.86	72.94	46.44
	STD	0.96	0.87	1.54	2.41	0.27	6.24	4.71
Several DA	Fold 0	60.82	63.64	60.74	66.28	59.27	75.17	45.74
	Fold 1	63.02	65.15	59.97	70.53	59.59	86.38	38.48
	Fold 2	60.2	63.64	59.53	66.97	58.25	78.77	40.69
	Avg.	61.35	64.14	60.08	67.93	59.04	80.11	41.64
	STD	1.48	0.87	0.61	2.28	0.70	5.72	3.72

+ - stronger intensity (larger kernel);

Table A.6: Results, per fold, for different output types.

Output type	Fold	Accuracy	Accuracy per-patient	AUC	F1-Score	Precision	Sensitivity	Specificity
SG 011	Fold 0	60.06	57.58	65.56	70.42	82.72	61.31	55.75
	Fold 1	75.03	60.61	69.82	84.05	83.29	84.82	41.21
	Fold 2	46.39	51.52	51.9	56.65	75.95	45.17	50.6
	Avg.	60.49	56.57	62.43	70.37	80.65	63.77	49.19
	STD	14.32	4.63	9.36	13.70	4.08	19.94	7.37
SG 001	Fold 0	56.93	51.52	55.67	46.21	59.51	37.77	75.33
	Fold 1	51.56	51.52	51.16	35.39	51.04	27.08	75.06
	Fold 2	56.53	51.52	56.05	44.96	59.18	36.24	75.99
	Avg.	55.01	51.52	54.29	42.19	56.58	33.70	75.46
	STD	2.99	0.00	2.72	5.92	4.80	5.78	0.48
SG 0-1	Fold 0	53.33	50.00	58.36	59.57	73.37	50.13	60.3
	Fold 1	41.8	46.15	38.08	53.46	59.19	48.74	26.67
	Fold 2	38.95	34.62	39.95	41.13	60.7	31.11	56.06
	Avg.	44.69	43.59	45.46	51.39	64.42	43.33	47.68
	STD	7.61	8.00	11.21	9.39	7.79	10.60	18.32
SG 012	Fold 0	40.1	31.03	61.14	35.8	38.75	38.83	69.49
	Fold 1	19.87	13.79	40.31	17.4	17.29	20.03	59.33
	Fold 2	35.62	34.48	47.1	33.32	33.11	35.3	67.22
	Avg.	31.86	26.43	49.52	28.84	29.72	31.39	65.35
	STD	10.63	11.08	10.62	9.98	11.13	9.99	5.33

Table A.7: Results, per fold, for the networks whose input was the feature-block.

Model	Fold	Accuracy (per-patient)	AUC	F1-Score	Precision	Specificity	Sensitivity
Transformer	Fold 0	59.09	58	64.94	58.14	43.75	73.53
	Fold 1	63.64	65	70.73	60.42	40.63	85.29
	Fold 2	59.09	62	63.01	58.97	50.00	67.65
	Avg.	60.61	61.67	66.23	59.18	44.79	75.49
	STD	2.62	3.51	4.02	1.15	4.77	8.99
CNN + Transformer	Fold 0	51.52	59	68.00	51.52	0	100.00
	Fold 1	50.00	51	52.17	51.43	46.88	52.94
	Fold 2	46.97	48	56.79	48.94	25.00	67.65
	Avg.	49.49	52.67	58.99	50.63	23.96	73.53
	STD	2.31	5.69	8.14	1.46	23.45	24.07
CNN	Fold 0	59.09	53	65.82	57.78	40.625	76.47
	Fold 1	51.52	49	57.89	52.38	37.50	64.71
	Fold 2	46.97	42	47.76	48.48	46.88	47.06
	Avg.	52.53	48.00	57.16	52.88	41.67	62.75
	STD	6.12	5.57	9.05	4.67	4.77	14.80

References

- [1] Neurology 2021 market analysis - proquest. Accessed: 2021-07-14. URL: <https://www.proquest.com/openview/31fc23868839899973056b3a31517c70/1?pq-origsite=gscholar&cbl=51651>.
- [2] World alzheimer report 2015 | dementia analysis. Accessed: 2021-07-14. URL: <https://www.alzint.org/u/WorldAlzheimerReport2015.pdf>.
- [3] Evelyn Sieczkowski, Ivan Milenkovic, Vivek Venkataramani, Regina Giera, Thomas Ströbel, Romana Höftberger, Paweł P. Liberski, Eduard Auff, Oliver Wirths, Thomas A. Bayer, and et al. I716f a β pp mutation associates with the deposition of oligomeric pyroglutamate amyloid- β and α -synucleinopathy with lewy bodies. *Journal of Alzheimer's Disease*, 44(1):103–114, January 2015. URL: <https://doi.org/10.3233/JAD-141524>, doi:10.3233/JAD-141524.
- [4] Arden Dertat. Applied deep learning - part 1: Artificial neural networks | by arden dertat | towards data science. Accessed: 2021-06-14 11:11:01. URL: <https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6>.
- [5] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, December 1943. URL: <https://doi.org/10.1007/bf02478259>, doi:10.1007/bf02478259.
- [6] B. Farley and W. Clark. Simulation of self-organizing systems by digital computer. *Transactions of the IRE Professional Group on Information Theory*, 4(4):76–84, September 1954. URL: <https://doi.org/10.1109/tit.1954.1057468>, doi:10.1109/tit.1954.1057468.
- [7] Zhong-Qiu Zhao, Peng Zheng, Shou-Tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 30(11):3212–3232, November 2019. URL: <https://doi.org/10.1109/tnnls.2018.2876865>, doi:10.1109/tnnls.2018.2876865.
- [8] Zixing Zhang, Jürgen Geiger, Jouni Pohjalainen, Amr El-Desoky Mousa, Wenyu Jin, and Björn Schuller. Deep learning for environmentally robust speech recognition. *ACM Transactions on Intelligent Systems and Technology*, 9(5):1–28, July 2018. URL: <https://doi.org/10.1145/3178115>, doi:10.1145/3178115.
- [9] Hongming Chen, Ola Engkvist, Yinhai Wang, Marcus Olivecrona, and Thomas Blaschke. The rise of deep learning in drug discovery. *Drug Discovery Today*, 23(6):1241–1250, June 2018. URL: <https://doi.org/10.1016/j.drudis.2018.01.039>, doi:10.1016/j.drudis.2018.01.039.

- [10] James Zou, Mikael Huss, Abubakar Abid, Pejman Mohammadi, Ali Torkamani, and Amalio Telenti. A primer on deep learning in genomics. *Nature Genetics*, 51(1):12–18, November 2018. URL: <https://doi.org/10.1038/s41588-018-0295-5>, doi:10.1038/s41588-018-0295-5.
- [11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. URL: <https://doi.org/10.1038/nature14539>, doi:10.1038/nature14539.
- [12] Approved medications | american parkinson disease assoc. Accessed: 2021-07-14. URL: <https://www.apdaparkinson.org/what-is-parkinsons/treatment-medication/medication/>.
- [13] Cns trial failure rates high as need for new drugs grows. Accessed: 2021-07-14. URL: <https://www.wcgclinical.com/insights/institute/cns-trial-failure-rates-high-as-need-for-new-drugs-grows>.
- [14] Accelerated approval program | fda. Accessed: 2021-07-14. URL: <https://www.fda.gov/drugs/information-health-care-professionals-drugs/accelerated-approval-program>.
- [15] Randolph W Evans. Complications of lumbar puncture. *Neurologic clinics*, 16(1):83–105, 1998.
- [16] Mike A Nalls, Cory Y McLean, Jacqueline Rick, Shirley Eberly, Samantha J Hutten, Katrina Gwinn, Margaret Sutherland, Maria Martinez, Peter Heutink, Nigel M Williams, John Hardy, Thomas Gasser, Alexis Brice, T Ryan Price, Aude Nicolas, Margaux F Keller, Cliona Molony, J Raphael Gibbs, Alice Chen-Plotkin, Eunran Suh, Christopher Letson, Massimo S Fiandaca, Mark Mapstone, Howard J Federoff, Alastair J Noyce, Huw Morris, Vivianne M Van Deerlin, Daniel Weintraub, Cyrus Zabetian, Dena G Hernandez, Suzanne Lesage, Meghan Mullins, Emily Drabant Conley, Carrie A M Northover, Mark Frasier, Ken Marek, Aaron G Day-Williams, David J Stone, John P A Ioannidis, and Andrew B Singleton. Diagnosis of parkinson's disease on the basis of clinical and genetic classification: a population-based modelling study. *The Lancet Neurology*, 14(10):1002–1009, October 2015. URL: [https://doi.org/10.1016/s1474-4422\(15\)00178-7](https://doi.org/10.1016/s1474-4422(15)00178-7), doi:10.1016/s1474-4422(15)00178-7.
- [17] Parkinson's disease - diagnosis and treatment - mayo clinic. Accessed: 2021-07-14. URL: <https://www.mayoclinic.org/diseases-conditions/parkinsons-disease/diagnosis-treatment/drc-20376062>.
- [18] Dat scan to diagnose parkinson's disease | university of florida. Accessed: 2021-07-14. URL: <https://movementdisorders.ufhealth.org/2012/04/11/should-i-get-a-dat-scan-to-confirm-my-parkinsons-disease/>.
- [19] Neurological diagnostic tests and procedures fact sheet | national institute of neurological disorders and stroke. Accessed: 2021-07-14. URL: <https://www.ninds.nih.gov/Disorders/Patient-Caregiver-Education/Fact-Sheets/Neurological-Diagnostic-Tests-and-Procedures-Fact>.
- [20] Hasan Aldewachi, Radhwan N. Al-Zidan, Matthew T. Conner, and Mootaz M. Salman. High-throughput screening platforms in the discovery of novel drugs for neurodegenerative

- diseases. *Bioengineering*, 8(2):30, February 2021. URL: <https://doi.org/10.3390/bioengineering8020030>, doi:10.3390/bioengineering8020030.
- [21] T. William Lambe and Robert V. Whitman. *Soil Mechanics*. 01 1991.
- [22] Renliang Xu. *Particle Characterization: Light Scattering Methods*. Kluwer Academic, 2002.
- [23] Renliang Xu. Light scattering: A review of particle characterization applications. *Particuology*, 18:11–21, February 2015. URL: <https://doi.org/10.1016/j.partic.2014.05.002>, doi:10.1016/j.partic.2014.05.002.
- [24] Onofrio M. Maragò, Philip H. Jones, Pietro G. Gucciardi, Giovanni Volpe, and Andrea C. Ferrari. Optical trapping and manipulation of nanostructures. *Nature Nanotechnology*, 8(11):807–819, November 2013. URL: <https://doi.org/10.1038/nnano.2013.208>, doi:10.1038/nnano.2013.208.
- [25] Joana S. Paiva, Pedro A. S. Jorge, Rita S. R. Ribeiro, Meritxell Balmaña, Diana Campos, Stefan Mereiter, Chunsheng Jin, Niclas G. Karlsson, Paula Sampaio, Celso A. Reis, and João P. S. Cunha. iLoF: An intelligent lab on fiber approach for human cancer single-cell type identification. *Scientific Reports*, 10(1), February 2020. URL: <https://doi.org/10.1038/s41598-020-59661-5>, doi:10.1038/s41598-020-59661-5.
- [26] Joana Paiva, Rita Ribeiro, João Cunha, Carla Rosa, and Pedro Jorge. Single particle differentiation through 2d optical fiber trapping and back-scattered signal statistical analysis: An exploratory approach. *Sensors*, 18(3):710, February 2018. URL: <https://doi.org/10.3390/s18030710>, doi:10.3390/s18030710.
- [27] Alan V. Oppenheim, Ronald W. Schaffer, and John R. Buck. *Discrete-time Signal Processing*. Prentice-Hall Inc., 02 1999.
- [28] Hwei P. Hsu. *Schaum's Outline of Signals and Systems 3ed*. 11 2013.
- [29] L. A. Zadeh. Theory of filtering. *Journal of the Society for Industrial and Applied Mathematics*, 1(1):35–51, September 1953. URL: <https://doi.org/10.1137/0101003>, doi:10.1137/0101003.
- [30] Wikipedia. Frequency domain. Accessed: 2021-02-06. URL: https://en.wikipedia.org/wiki/Frequency_domain.
- [31] Sean A Fulop and Kelly Fitz. A spectrogram for the twenty-first century. *Acoustics Today*, 2(3):26–33, 2006.
- [32] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, Cambridge, Massachusetts, first edition, 2012.
- [33] Eibe Frank Ian H. Witten. *Data Mining, Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers, San Francisco, CA, second edition, 2002.
- [34] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, NY, 2006.
- [35] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160(1):3–24, 2007.

- [36] Alexei Botchkarev. Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology. *arXiv preprint arXiv:1809.03006*, 2018.
- [37] Sanford Weisberg. *Applied linear regression*, volume 528. John Wiley & Sons, 2005.
- [38] Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM computing surveys (CSUR)*, 27(3):326–327, 1995.
- [39] Raymond E Wright. Logistic regression. 1995.
- [40] Horace B Barlow. Unsupervised learning. *Neural computation*, 1(3):295–311, 1989.
- [41] W Thomas Miller, Paul J Werbos, and Richard S Sutton. *Neural networks for control*. MIT press, 1995.
- [42] Raul Rojas. The backpropagation algorithm. In *Neural networks*, pages 149–182. Springer, 1996.
- [43] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *arXiv preprint arXiv:2004.11362*, 2020.
- [44] Peter A Flach. The geometry of roc space: understanding machine learning metrics through roc isometrics. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 194–201, 2003.
- [45] Attributed to von Neumann by Enrico Fermi, as quoted by Freeman Dyson in "A meeting with Enrico Fermi" in *Nature* 427 (22 January 2004) p. 297.
- [46] Pierre Baldi and Peter J Sadowski. Understanding dropout. *Advances in neural information processing systems*, 26:2814–2822, 2013.
- [47] François Chollet. *Deep Learning with Python*. Manning Publications Co., 10 2017.
- [48] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. Ieee, 2017.
- [49] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 11 2016.
- [50] Richard Szeliski. *Computer Vision*. Springer, 10 2010.
- [51] Eugene Charniak. *Introduction to Deep Learning*. MIT Press, 02 2021.
- [52] Min Peng, Chongyang Wang, Tong Chen, Guangyuan Liu, and Xiaolan Fu. Dual temporal scale convolutional neural network for micro-expression recognition. *Frontiers in Psychology*, 8, October 2017. URL: <https://doi.org/10.3389/fpsyg.2017.01745>, doi:10.3389/fpsyg.2017.01745.
- [53] Josh Patterson and Adam Gibson. *Deep Learning*. O'Reilly Media, Inc., 02 2021.
- [54] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

- [55] Sebastian Ruder. An overview of multi-task learning in deep neural networks, 2017. [arXiv:1706.05098](https://arxiv.org/abs/1706.05098).
- [56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. [arXiv:1706.03762](https://arxiv.org/abs/1706.03762).
- [57] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. [arXiv:2005.14165](https://arxiv.org/abs/2005.14165).
- [58] Artur L. Gower, Robert M. Gower, Jonathan Deakin, William J. Parnell, and I. David Abrahams. Characterising particulate random media from near-surface backscattering: A machine learning approach to predict particle size and concentration. *EPL (Europhysics Letters)*, 122(5):54001, July 2018. URL: <https://doi.org/10.1209/0295-5075/122/54001>, doi:10.1209/0295-5075/122/54001.
- [59] James A. Grant-Jacob, Benita S. Mackay, James A. G. Baker, Daniel J. Heath, Yunhui Xie, Matthew Loxham, Robert W. Eason, and Ben Mills. Real-time particle pollution sensing using machine learning. *Optics Express*, 26(21):27237, October 2018. URL: <https://doi.org/10.1364/oe.26.027237>, doi:10.1364/oe.26.027237.
- [60] James A Grant-Jacob, Saurabh Jain, Yunhui Xie, Benita S Mackay, Michael D T McDonnell, Matthew Praeger, Matthew Loxham, David J Richardson, Robert W Eason, and Ben Mills. Fibre-optic based particle sensing via deep learning. *Journal of Physics: Photonics*, 1(4):044004, September 2019. URL: <https://doi.org/10.1088/2515-7647/ab437b>, doi:10.1088/2515-7647/ab437b.
- [61] Daniel Brodeski, Igal Bilik, and Raja Giryes. Deep radar detector. *CoRR*, abs/1906.12187, 2019. URL: <http://arxiv.org/abs/1906.12187>, [arXiv:1906.12187](https://arxiv.org/abs/1906.12187).
- [62] A. Galusha, J. Dale, J. M. Keller, and A. Zare. Deep convolutional neural network target classification for underwater synthetic aperture sonar imagery. In *Detection and Sensing of Mines, Explosive Objects, and Obscured Targets XXIV, 11012*, volume 11012 of *Proc. SPIE*, May 2019. doi:10.1117/12.2519521.
- [63] S. W. Perry and Ling Guan. A recurrent neural network for detecting objects in sequences of sector-scan sonar images. *IEEE Journal of Oceanic Engineering*, 29(3):857–871, 2004. doi:10.1109/JOE.2004.831616.
- [64] J. Park and D. Jung. Identifying tonal frequencies in a lofargram with convolutional neural networks. In *2019 19th International Conference on Control, Automation and Systems (ICCAS)*, pages 338–341, 2019. doi:10.23919/ICCAS47443.2019.8971701.
- [65] J. Kim, S. Song, and S. Yu. Denoising auto-encoder based image enhancement for high resolution sonar image. In *2017 IEEE Underwater Technology (UT)*, pages 1–5, 2017. doi:10.1109/UT.2017.7890316.

- [66] Yong Xia, Naren Wulan, Kuanquan Wang, and Henggui Zhang. Detecting atrial fibrillation by deep convolutional neural networks. *Computers in Biology and Medicine*, 93:84–92, February 2018. URL: <https://doi.org/10.1016/j.combiomed.2017.12.007>, doi:10.1016/j.combiomed.2017.12.007.
- [67] Jen Hong Tan, Yuki Hagiwara, Winnie Pang, Ivy Lim, Shu Lih Oh, Muhammad Adam, Ru San Tan, Ming Chen, and U. Rajendra Acharya. Application of stacked convolutional and long short-term memory network for accurate identification of CAD ECG signals. *Computers in Biology and Medicine*, 94:19–26, March 2018. URL: <https://doi.org/10.1016/j.combiomed.2017.12.023>, doi:10.1016/j.combiomed.2017.12.023.
- [68] Muhammad Zubair, Jinsul Kim, and Changwoo Yoon. An automated ECG beat classification system using convolutional neural networks. In *2016 6th International Conference on IT Convergence and Security (ICITCS)*. IEEE, September 2016. URL: <https://doi.org/10.1109/icitcs.2016.7740310>, doi:10.1109/icitcs.2016.7740310.
- [69] Siti Nurmaini, Radiyati Umi Partan, Wahyu Caesarendra, Tresna Dewi, Muhammad Naufal Rahmatullah, Annisa Darmawahyuni, Vicko Bhayyu, and Firdaus Firdaus. An automated ECG beat classification system using deep neural networks with an unsupervised feature extraction technique. *Applied Sciences*, 9(14):2921, July 2019. URL: <https://doi.org/10.3390/app9142921>, doi:10.3390/app9142921.
- [70] Dan Li, Jianxin Zhang, Qiang Zhang, and Xiaopeng Wei. Classification of ECG signals based on 1d convolution neural network. In *2017 IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom)*. IEEE, October 2017. URL: <https://doi.org/10.1109/healthcom.2017.8210784>, doi:10.1109/healthcom.2017.8210784.
- [71] Fei Wang, Qiming Ma, Wenhan Liu, Sheng Chang, Hao Wang, Jin He, and Qijun Huang. A novel ECG signal compression method using spindle convolutional auto-encoder. *Computer Methods and Programs in Biomedicine*, 175:139–150, July 2019. URL: <https://doi.org/10.1016/j.cmpb.2019.03.019>, doi:10.1016/j.cmpb.2019.03.019.
- [72] Eric Ke Wang, , liu Xi, Rupei Sun, Fan Wang, Leyun Pan, Caixia Cheng, Antonia D. Dimitrakopoulou-Strauss, Nie Zhe, Yueping Li, and and. A new deep learning model for assisted diagnosis on electrocardiogram. *Mathematical Biosciences and Engineering*, 16(4):2481–2491, 2019. URL: <https://doi.org/10.3934/mbe.2019124>, doi:10.3934/mbe.2019124.
- [73] Kwon-Woo Ha and Jin-Woo Jeong. Motor imagery EEG classification using capsule networks. *Sensors*, 19(13):2854, June 2019. URL: <https://doi.org/10.3390/s19132854>, doi:10.3390/s19132854.
- [74] Xiaoyan Wei, Lin Zhou, Ziyi Chen, Liangjun Zhang, and Yi Zhou. Automatic seizure detection using three-dimensional CNN based on multi-channel EEG. *BMC Medical Informatics and Decision Making*, 18(S5), December 2018. URL: <https://doi.org/10.1186/s12911-018-0693-8>, doi:10.1186/s12911-018-0693-8.
- [75] Dongye Zhao, Fengzhen Tang, Bailu Si, and Xisheng Feng. Learning joint space–time–frequency features for EEG decoding on small labeled data. *Neural Networks*,

- 114:67–77, June 2019. URL: <https://doi.org/10.1016/j.neunet.2019.02.009>, doi:10.1016/j.neunet.2019.02.009.
- [76] Yang Li, Xian-Rui Zhang, Bin Zhang, Meng-Ying Lei, Wei-Gang Cui, and Yu-Zhu Guo. A channel-projection mixed-scale convolutional neural network for motor imagery EEG decoding. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 27(6):1170–1180, June 2019. URL: <https://doi.org/10.1109/tnsre.2019.2915621>, doi:10.1109/tnsre.2019.2915621.
- [77] Amirali Vahid, Annet Bluschke, Veit Roessner, Sebastian Stober, and Christian Beste. Deep learning based on event-related EEG differentiates children with ADHD from healthy controls. *Journal of Clinical Medicine*, 8(7):1055, July 2019. URL: <https://doi.org/10.3390/jcm8071055>, doi:10.3390/jcm8071055.
- [78] Shuo Yang, Zhong Yin, Yagang Wang, Wei Zhang, Yongxiong Wang, and Jianhua Zhang. Assessing cognitive mental workload via EEG signals and an ensemble deep learning classifier based on denoising autoencoders. *Computers in Biology and Medicine*, 109:159–170, June 2019. URL: <https://doi.org/10.1016/j.compbiomed.2019.04.034>, doi:10.1016/j.compbiomed.2019.04.034.
- [79] Zohreh Gholami Doborjeh, Nikola Kasabov, Maryam Gholami Doborjeh, and Alexander Sumich. Modelling peri-perceptual brain processes in a deep learning spiking neural network architecture. *Scientific Reports*, 8(1), June 2018. URL: <https://doi.org/10.1038/s41598-018-27169-8>, doi:10.1038/s41598-018-27169-8.
- [80] Suat Toraman, Seda Arslan Tuncer, and Ferhat Balgetir. Is it possible to detect cerebral dominance via EEG signals by using deep learning? *Medical Hypotheses*, 131:109315, October 2019. URL: <https://doi.org/10.1016/j.mehy.2019.109315>, doi:10.1016/j.mehy.2019.109315.
- [81] Stefan Jonas, Andrea O. Rossetti, Mauro Oddo, Simon Jenni, Paolo Favaro, and Frederic Zubler. EEG-based outcome prediction after cardiac arrest with convolutional neural networks: Performance and visualization of discriminative features. *Human Brain Mapping*, 40(16):4606–4617, July 2019. URL: <https://doi.org/10.1002/hbm.24724>, doi:10.1002/hbm.24724.
- [82] Kostas Mu. Tsiouris, Vasileios C. Pezoulas, Michalis Zervakis, Spiros Konitsiotis, Dimitrios D. Koutsouris, and Dimitrios I. Fotiadis. A long short-term memory deep learning network for the prediction of epileptic seizures using EEG signals. *Computers in Biology and Medicine*, 99:24–37, August 2018. URL: <https://doi.org/10.1016/j.compbiomed.2018.05.019>, doi:10.1016/j.compbiomed.2018.05.019.
- [83] Baiying Lei, Xiaolu Liu, Shuang Liang, Wenlong Hang, Qiong Wang, Kup-Sze Choi, and Jing Qin. Walking imagery evaluation in brain computer interfaces via a multi-view multi-level deep polynomial network. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 27(3):497–506, March 2019. URL: <https://doi.org/10.1109/tnsre.2019.2895064>, doi:10.1109/tnsre.2019.2895064.
- [84] Isabell Kiral-Kornek, Subhrajit Roy, Ewan Nurse, Benjamin Mashford, Philippa Karoly, Thomas Carroll, Daniel Payne, Susmita Saha, Steven Baldassano, Terence O’Brien, David Grayden, Mark Cook, Dean Freestone, and Stefan Harrer. Epileptic seizure prediction using big data and deep learning: Toward a mobile system. *EBioMedicine*, 27:103–111,

- January 2018. URL: <https://doi.org/10.1016/j.ebiom.2017.11.032>, doi: 10.1016/j.ebiom.2017.11.032.
- [85] Ali Ameri, Mohammad Ali Akhaee, Erik Scheme, and Kevin Englehart. Real-time, simultaneous myoelectric control using a convolutional neural network. *PLOS ONE*, 13(9):e0203835, September 2018. URL: <https://doi.org/10.1371/journal.pone.0203835>, doi:10.1371/journal.pone.0203835.
- [86] Chuanjiang Li, Jian Ren, Huaiqi Huang, Bin Wang, Yanfei Zhu, and Huosheng Hu. PCA and deep learning based myoelectric grasping control of a prosthetic hand. *BioMedical Engineering OnLine*, 17(1), August 2018. URL: <https://doi.org/10.1186/s12938-018-0539-8>, doi:10.1186/s12938-018-0539-8.
- [87] Henrique Dantas, David J. Warren, Suzanne M. Wendelken, Tyler S. Davis, Gregory A. Clark, and V John Mathews. Deep learning movement intent decoders trained with dataset aggregation for prosthetic limb control. *IEEE Transactions on Biomedical Engineering*, 66(11):3192–3203, November 2019. URL: <https://doi.org/10.1109/tbme.2019.2901882>, doi:10.1109/tbme.2019.2901882.
- [88] Hiroyuki Nodera, Yusuke Osaki, Hiroki Yamazaki, Atsuko Mori, Yuishin Izumi, and Ryuji Kaji. Deep learning for waveform identification of resting needle electromyography signals. *Clinical Neurophysiology*, 130(5):617–623, May 2019. URL: <https://doi.org/10.1016/j.clinph.2019.01.024>, doi:10.1016/j.clinph.2019.01.024.
- [89] Y. Yu, X. Chen, S. Cao, X. Zhang, and X. Chen. Exploration of chinese sign language recognition using wearable sensors based on deep belief net. *IEEE Journal of Biomedical and Health Informatics*, 24(5):1310–1320, 2020. doi:10.1109/JBHI.2019.2941535.
- [90] Lance Rane, Ziyun Ding, Alison H. McGregor, and Anthony M. J. Bull. Deep learning for musculoskeletal force prediction. *Annals of Biomedical Engineering*, 47(3):778–789, December 2018. URL: <https://doi.org/10.1007/s10439-018-02190-0>, doi: 10.1007/s10439-018-02190-0.
- [91] Sojeong Ha, Jeong-Min Yun, and Seungjin Choi. Multi-modal convolutional neural networks for activity recognition. In *2015 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, October 2015. URL: <https://doi.org/10.1109/smc.2015.525>, doi:10.1109/smc.2015.525.
- [92] Hyungui Lim, J. Park, K. Lee, and Yoonchang Han. Rare sound event detection using 1d convolutional recurrent neural networks. 2017.
- [93] Ahmed Imtiaz Humayun, Md. Tauhiduzzaman Khan, Shabnam Ghaffarzadegan, Zhe Feng, and Taufiq Hasan. An ensemble of transfer, semi-supervised and supervised learning methods for pathological heart sound classification. *CoRR*, abs/1806.06506, 2018. URL: <http://arxiv.org/abs/1806.06506>, arXiv:1806.06506.
- [94] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL: <http://arxiv.org/abs/1505.04597>, arXiv:1505.04597.
- [95] Oliver Faust, Yuki Hagiwara, Tan Jen Hong, Oh Shu Lih, and U Rajendra Acharya. Deep learning for healthcare applications based on physiological signals: A review.

- Computer Methods and Programs in Biomedicine*, 161:1 – 13, 2018. URL: <http://www.sciencedirect.com/science/article/pii/S0169260718301226>, doi:<https://doi.org/10.1016/j.cmpb.2018.04.005>.
- [96] Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, and Quoc V. Le. SpecAugment: A simple data augmentation method for automatic speech recognition. *Interspeech 2019*, Sep 2019. URL: <http://dx.doi.org/10.21437/Interspeech.2019-2680>, doi:10.21437/interspeech.2019-2680.
- [97] Martin Zihlmann, Dmytro Perekrstenko, and Michael Tschannen. Convolutional recurrent neural networks for electrocardiogram classification. In *2017 Computing in Cardiology Conference (CinC)*. Computing in Cardiology, September 2017. URL: <https://doi.org/10.22489/cinc.2017.070-060>, doi:10.22489/cinc.2017.070-060.
- [98] Ali Hassani, Steven Walton, Nikhil Shah, Abulikemu Abuduweili, Jiachen Li, and Humphrey Shi. Escaping the big data paradigm with compact transformers, 2021. *arXiv: 2104.05704*.
- [99] Search of: Active, not recruiting studies | neurodegenerative disorders - list results - clinicaltrials.gov. Accessed: 2021-07-14. URL: <https://clinicaltrials.gov/ct2/results?cond=neurodegenerative%5C&term=&cntry=&state=&city=&dist=&Search=Search&recrs=a&recrs=b&recrs=f>.
- [100] Search of: Active, not recruiting studies | neurodegenerative - list results - clinicaltrials.gov. Accessed: 2021-07-14. URL: <https://clinicaltrials.gov/ct2/results?cond=neurodegenerative%5C&term=&cntry=&state=&city=&dist=&Search=Search&recrs=d>.
- [101] Inside clinical trials: Testing medical products in people | fda. Accessed: 2021-06-20. URL: <https://www.fda.gov/drugs/information-consumers-and-patients-drugs/inside-clinical-trials-testing-medical-products-people>.
- [102] How long do clinical trials take? Accessed: 2021-06-20. URL: <https://www.clinicaltrialsandme.com/resources/how-long-do-clinical-trials-take.html>.