UNIVERSITY
OF OSLO

Master's thesis

Classification and feature Regression for Multi-Phase Flow Regimes

A novel application of Deep Learning methods on Acoustic Emissions from cylindrical pipes

Daniel Johan Aarstein

60 ECTS study points

Department of Physics
Faculty of Mathematics and Natural Sciences

Spring 2023

Daniel Johan Aarstein

Classification and feature Regression for Multi-Phase Flow Regimes

A novel application of Deep Learning methods on Acoustic Emissions from cylindrical pipes

Supervisors:
Atle Jensen, prof. of Mechanics
Morten Hjort-Jensen, prof. of Physics
Anis Awal Ayati, Principal Flow Assurance Researcher, Equinor

Abstract

Detection and classification of flow-regimes are needed for improvements in the oil and gas sector, as well as in nuclear power plants. This thesis work presents a novel combination of convolutional neural networks applied to acoustic emissions from pipes containing multi-phase flow.

A novel dataset is constructed from experimental data, and automatically labeled with video analysis.

The proposed model classifies four distinct classes as well as performing a regression on the velocity and length of slugs appearing in the pipe. Both 2D and 1D transformations are tested in combination with different scaling methods. In addition the effectiveness of using a singular microphone is tested. The highest classification accuracy obtained on previously unseen data was 98.5%. The highest R2 score for slug velocity regression was 0.787 and the highest R2 score for slug length regression was 0.428.

Acknowledgements

I would like to thank Professor Atle Jensen for his patience this year. I dare say our weekly meetings has taught me more about the scientific method and rigour than all previous education combined.

I would like to thank Professor Morten Hjorth-Jensen for introducing me both to the field of Computational Physics, as well as giving me the tools necessary for the analysis performed in this thesis. The three courses you have lectured has laid the foundation for a technical skill set I'm sure will be utilized for the rest of my life.

I would like to thank Dr. Anis Awal Ayati for first introducing me to Atle, and as an extension, the interesting field of Multi-Phase Flow itself.

I would like to thank Olav Gundersen for his amazing work in constructing the lab environment where all experiments took place. Without you expertice, none of this would have been possible.

Finally I would like to thank everyone who has contributed to my five years at the University of Oslo, be it friends, colleagues or family. Special thanks to my fellow students at the Computational Science: Physics programme.

Contents

Abstract i

Acknowledgements iii

List of Figures xii

List of Tables xiv

I Introduction 1

1 Introduction 3
- 1.1 Turbulent pipe flow . . . 3
- 1.2 Flow Regime identification with Machine Learning . . . 4
  - 1.2.1 Acoustic Emission . . . 4
- 1.3 Contribution . . . 5
- 1.4 Thesis Structure . . . 5
- 1.5 Additional resources . . . 6

II Theory 7

2 Slug Flow 9
- 2.1 Flow Regimes . . . 9
- 2.2 Slugs . . . 9
  - 2.2.1 Aerated slug . . . 10
  - 2.2.2 Plug flow . . . 11

3 Machine Learning 13
- 3.1 The basics . . . 14
- 3.2 Feed-Forward Neural Networks . . . 15

3.2.1 Activation Functions . . . 17
3.2.2 Mathematics behind the forward pass . . . 18
3.2.3 Loss functions . . . 22
3.2.4 Update weights and biases; Gradient descent . . . 25
3.3 Convolutional Neural Networks . . . 31
3.3.1 Convolution . . . 31
3.3.2 Trainable kernel . . . 32
3.3.3 Data stream . . . 33

III Methodology & Implementation 39

4 Laboratory setup 41
4.1 Layout . . . 41
4.2 Hardware . . . 41

5 Data 45
5.1 Data Treatment . . . 46
5.1.1 Audio . . . 46
5.1.2 Video and Automated Labeling . . . 47
5.2 The Constructed Dataset . . . 56

6 Specific Application of the CNN 59
6.1 Architecture . . . 59
6.1.1 Custom loss function . . . 61
6.2 Data Pre-Processing . . . 61
6.2.1 Scaling . . . 62
6.2.2 Transformations . . . 62
6.3 Implementations . . . 62
6.3.1 1D . . . 62
6.3.2 2D . . . 63
6.4 Metrics . . . 64

IV Results & Discussions 65

7 Results & Discussion 67
7.1 Multi-channel . . . 67
7.1.1 2D . . . 67
7.1.2 1D . . . 71
7.2 Single Channel . . . 74

8 Conclusion **79**
8.1 Future Work 80

A MNIST Audio Case Study **87**

List of Figures

2.1 Illustrated flow regimes in horizontal gas-liquid two-phase flow. The arrow indicates the order from most liquid to least liquid. The illustration is taken from Holland and Bragg [25]. 10

2.2 Examples of the nose and tail for an aerated slug. Note the diffused light from the underlying LED light, as described in chapter 4. 11

2.3 Examples of the nose and tail for a plug slug. Note how there is practically no diffusion of the light when compared with fig. 2.2. . . . 12

3.1 Illustration of a simple Feed-Forward Neural Network, containing two hidden layers. 16

3.2 Crude categorization of Artificial Intelligence frameworks. Note that the borders are somewhat blurred, this definition is in alignment with Goodfellow et al. [30]. 17

3.3 Illustration of the softmax function applied to a rank 1 tensor. Note how the elements of the tensor softmax($\hat{\mathbf{y}}$) sum to 1. 23

3.4 Illustration of how the Cross Entropy Loss function acts on a rank 1 tensor which has undergone a softmax transformation. 24

3.5 Example of a 2D convolution applied to an image in order to extract some feature. In this case the Canny edge detection algorithm has been performed, with the Sobel kernel. Note the lack of coffee present in the mug. 32

3.6 Illustration of 2D convolution on a rank 2 tensor. The kernel is a 3x3 rank-2 tensor, such that the resulting tensor is a 3x3 rank-2 tensor. In this example, the stride of the convolution kernel is equal to 1. 35

3.7 Illustration of how the ReLU activation function first works on a 5x5 rank-2 tensor, and then how the MaxPool2d function extracts the max value using a 2x2 kernel. In this example the stride of the MaxPool kernel is equal to its size, that is stride equal to 2. If the kernel would be out of bounds wrt. the tensor, the column or row in question is omitted. This is chosen deliberately as to accurately mimic how these funtions work by default in the PyTorch framework. 36
3.8 Illustration of how the dimensionality of a tensor changes when its passed through multiple convolution layers. In this case, each convolution has three kernels, such that the number of output channels is equal to the number of input channels times three. 37

4.1 Schematics describing the experimental setup used for audio and video sampling. The horizontally striped area by the inlet indicates the honeycomb mechanism which reduces turbulence where the gas-liquid boundary is formed. 42

5.1 5 minute data sample from run number 59. 46
5.2 Sample of a data segment which contains one event. This event in particular corresponds to the eight peak in fig. 5.1. 47
5.3 First frame of the mp4 file for run number 59, with the marked pixel column. 48
5.4 Illustration of how the values for a given column can be shown as an image with time along the x-axis and the height in pixel along the y-axis. In this example the column of pixel 1500 was chosen. 49
5.5 Time evolution at pixel column 1500 for the first event in run number 59. 50
5.6 Time evolution at pixel column 1500 for the first event in run number 59, with the row-wise mean subtracted from each row. 51
5.7 Intensity of the slug, used for further information extraction. Note that the intensity is dimensionless. 52
5.8 Slug nose and tail estimation method for plug slugs. 53
5.9 Slug nose and tail estimation method for aerated slugs. 54
5.10 Examples of velocity and length estimations, both for the plug slug and the aerated slug case. 55
5.11 Region of which the intensity was averaged over for aerated slug/breaking wave classification. 56
5.12 Count of the different classes in the final data set. Note that the number of noise events is equal to the number of aerated slug events and number of breaking wave events combined. 58

6.1 Sketch of the final convolutional neural network layout used in both the 1D and 2D cases. The three final feed-forward neural networks are independent from each other, such that the loss from one will not affect the others. This is indicated by the disconnected blue regions. In contrast, all convolution layers are connected, and are thus grouped in the same region. 60

7.1 The accuracy score for the proposed model when the data has undergone a spectrogram transformation followed by normalization. 69
7.2 The R2 scores for the proposed model when the data has undergone a spectrogram transformation followed by normalization. 70
7.3 The accuracy score for the proposed model when the data has undergone a spectrogram transformation followed by normalization. 71
7.4 The R2 scores for the proposed model when the data has undergone a spectrogram transformation followed by normalization. 72
7.5 The accuracy score for the proposed model when the data has undergone a Fast Fourier Transformation followed by normalization. 73
7.6 The R2 scores for the proposed model when the data has undergone a Fast Fourier Transformation followed by normalization. 74
7.7 The confusion matrices for the normalized Short-Time Fourier Transform at epoch 7. 75
7.8 The confusion matrices for the normalized Fast Fourier Transform at epoch 20. 76
7.9 The accuracy score for the proposed model when the data from microsphone 2 has undergone a Short Time Fourier Transform followed by normalization. 77
7.10 The R2 scores for the proposed model when the data from microsphone 2 has undergone a Short Time Fourier Transform followed by normalization. 77
7.11 The accuracy score for the proposed model when the data from microsphone 2 has undergone a Short Time Fourier Transform followed by normalization. 78
7.12 The R2 scores for the proposed model when the data from microsphone 2 has undergone a Short Time Fourier Transform followed by normalization. 78

A.1 Number of speakers for the AudioMNIST dataset, sorted by their country of origin. Note: The double category count (Spain/Spanien) is a consequence of the meta data file provided by the Audio-source. . . . 88

A.2 Example of data and its spectrogram transformation, file `7_20_23.wav` was used. 89
A.3 The accuracy of the model as a functions of epochs trained, given as a fraction. Note that testing was done after each epoch, meaning that even for epoch = 1 the model had seen every data point and updated its weights and biases accordingly. 91
A.4 Confusion Matrix for True labels vs. Predicted labels for train and test data in epoch 12. 91
A.5 Confusion Matrix for True labels vs. Predicted labels for train and test data in epoch 22. 92

List of Tables

3.1 Definitions and derivatives for some of the most commonly used activation functions. 18

5.1 Overview of measurements used in the construction of the final data set. Note that "Noise" is equivalent to stratified or stratified-wavy flow, but no occurrences of slugs or breaking waves. 45
5.2 Chosen statistics about the events. Note that the runs 54, 55 and 58 contained noise data, thus no event is present. The count column for these runs reflect the number of segments each run was split into. . . 57

6.1 Kernel sizes and number of kernels (channels out) for each convolution layer. 61
6.2 Kernel sizes and number of kernels (channels out) for each convolution layer. 61

7.1 Performance metrics for the proposed 2D model with different data scaling, on multiple channels. The * indicates that the scaling takes place before a transformation, that is, on the raw audio data itself. The boldface highlights the best metric for the given column. 68
7.2 Performance metrics with data subjected to channel-wise and full standardization after the spectrogram transformation. The boldface highlights the best metric for the given column. 68
7.3 Performance metrics with data subjected to channel-wise and full normalization after the Short Time Fourier Transformation. The boldface highlights the best metric for the given column. 70
7.4 Performance metrics for the proposed 1D model with different scaling. The boldface highlights the best metric for the given column. 72
7.5 Performance metrics for the proposed 1D model with a Discrete Fourier Transform and different scaling. The boldface highlights the best metric for the given column. 73

7.6 Performance metrics for the proposed model when only subjected to data from one microphone. The boldface highlights the best metric for the given column. 75

A.1 Structure of the tested convolutional neural network, with layer-wise explanations. 90

Part I

Introduction

Chapter 1

Introduction

1.1 Turbulent pipe flow

Instabilities in multi-phase flow in pipes can lead to a phenomena known as slugs, which are intermittent regions of one phase occupying the cross-section of the pipe. In horizontal pipes, it is proposed that this behaviour is explainable through the Kelvin-Helmholtz instability [1], and the ceiling of the pipe in combination with the internal wave amplitude. Knowing why slug flow occurs does not make the problem trivially avoidable. Thus, slugs remain a headache in industrial purposes where multi-phase fluids are transported through pipes. Most notably, this occurs in the oil and gas sector and nuclear power plants.

Slug flow in oil and gas pipes are problematic for a multitude of reasons, the most prominent of which is equipment damage. The sudden change in composition is potentially damaging for the separators which receive the flow at the outlet of the pipe. An incoming slug could not only result in poor separation of the fluids, but, for severe enough slugging, could result in flooding [2, 3]. Moreover, severe slugging could cause corrosion and further damage on other equipment [4], which in turn leads to expensive maintenance and potential pauses in production. Both the cost of maintenance and the opportunity cost of not extracting from the wells lead to substantial economic losses. These losses in turn call for more efficient operations, which is dependent on proper identification of flow regimes, a task which machine learning models perform with an astonishing accuracy.

In the context of nuclear power plants, studies of multi-phase flow regimes through pipes are relevant for a multitude of reasons. Slug flow can occur, most prominently, in the steam generators, coolant transport pipes and within the liquid cooling itself. If these systems were to fail, the nuclear power generation could become unstable, which in turn could have devastating, long lasting effects. The accuracy of fluid

and heat-transport models depend heavily on flow-regimes, making these predictions important within the field of nuclear thermal-hydralics[5].

1.2 Flow Regime identification with Machine Learning

Using machine learning and/or deep learning methods on data from pipelines in order to classify flow regimes is not a novel idea, and machine learning is currently the mainstream, state-of-the-art method for flow regime identification [5]. Starting as early as 1991 Baba et al. applied artificial neural networks on data from flow-rate meters [6]. At the time, this was considered a massive success with an accuracy of 81%. Researchers were quick to perform similar experiments but with several variations when it came to the flow regimes themselves, source of the experimental data, and type of machine learning.

Measurement methods for gathering experimental data include, but are not limited to, pressure sensors [7–10], ultrasonic doppler [11, 12], gamma rays [13–15], and not surprisingly, cameras [16–19]. Recently the use of camera has increased in popularity, which is evident from the studies done by, Yang et al. [16], Hobold and da Silva [17], Du et al. [18] and Shen et al.[19], which all used cameras and were conducted within the last five years. The reason for this shift to video analysis is the relative ease and low operation cost of implementing a camera setup.

When it comes to machine learning, the earlier implementations relied more on the basic architectures, such as the feed-forward neural network. Moving on from Baba et al. with their 81%, Åbro et al. [13] managed in 1999 to classify horizontal gas-liquid multi-phase flow, with an accuracy of 97% with three classes. Similarly, Wu et al. [9] managed in 2001 an accuracy of 92.6% also for three distinct classes but this time on an oil-water multi-phase flow. Progress continued, and in 2017 Yang et al. [16] applied the convolutional neural network on data from a high speed camera. Yang's network had 7 convolution layers and classified experimental data from a pipe with an internal diameter of 4mm. Four classes were classified all with an accuracy above 92%. In 2019, Du et al. [18] compared the three popular convolutional neural network architectures LeNet-5 [20], AlexNet [21] and VGG-16 [22]. Again, the classification was between three classes, and the results ranged from 57.7% to 99.4% depending on the combination of network and flow regime.

1.2.1 Acoustic Emission

Acoustic emission is a promising candidate for analysis of multi-phase flow-regimes. The method is non-intrusive, such that it could be applied to an existing system as

is, and removed without damage after necessary analysis has taken place. It is in addition cheap, and could potentially be applied in circumstances which does not allow for larger and/or more fragile equipment. All these factors combined makes the use of acoustic emission interesting for the oil and gas sector as probes can be attached to existing pipelines, without damaging or compromising any existing structures.

Despite the need for flow regime classification, few attempts have been made when it comes to analysis by way of acoustic emissions. At least one study has previously been conducted, in 2002 Yen and Lu [23] used a feed-forward neural network in order to classify four different flow-regimes. It was in 2009 found by Al-lababidi et al. [24] that the measurement of acoustic emission is sufficient in order to determine void fraction. Despite these findings, analysis of acoustic emission is not the norm when it comes to analysis of multi-phase flow-regimes.

1.3 Contribution

This thesis aims to expand the domain of machine learning applied to experimental data from gas-liquid multi-phase flow. A novel combination of convolutional neural networks and acoustic emission is studied. Furthermore, the proposed model does not only classify multi-phase flow regimes, but provides predictions for the slug velocity and slug length. The model is trained on a new, unique dataset with acoustic emissions from a horizontal pipe with an inner diameter of 10cm. It is hypothesised that machine learning techniques can be applied to acoustic emission from pipes in order to extract characteristics of the internal multi-phase flow in a cost effective, efficient, and non-intrusive manner.

1.4 Thesis Structure

The necessary definitions for different flow regimes are presented, but no mathematical framework is used, and hence omitted. The theory section on machine learning aims to be self sufficient, and complete with respect to the tools used in this thesis work. With the exception of deriving the back-propagation algorithm for the convolutional kernels. The experimental setup used for gathering both audio and video data is presented and explained, allowing for proper recreations of the experiments. The data treatment process is described and discussed, and the final constructed dataset is presented. The specific applications of the network are presented along with justifications for which parameters were studied. The relevant metrics are briefly presented before, finally, our results and findings are presented and discussed, and a final conclusion is drawn.

1.5 Additional resources

The code used for this thesis work, and the constructed dataset, are available at the github repositories danaars/MSc and danaars/Pipesound respectively.

Part II

Theory

Chapter 2

Slug Flow

By definition, a slug is an intermittent region in which the entire cross section of the pipe consists of a single fluid. The slug is said to have a nose and a tail, corresponding to the front and back of the slug, relative to the direction of travel. The length of a slug is then the distance from the nose to the tail, and the velocity of the slug is defined as equal to the velocity of the slug nose. Due to the evolution of both the slug nose and tail throughout time, the characteristics of the gas-liquid boundary is destined to change somewhat during the slugs lifetime. Difficulties arise in accurately determining the lengths and velocities of slugs. The dynamic boarder introduces ambiguity in the parameters needed for length and velocity calculations.

2.1 Flow Regimes

Even though a slug itself is defined, subcategories are distinct enough to motivate further definitions. For horizontal pipes and a gas-liquid two-phase flow, the following flow regimes are recognised by Holland and Bragg [25]; Bubbly, Plug, Stratified, Wavy, Slug, Annular and Spray. The flow regimes are presented in descending order when it comes to liquid volume, bubbly flow consisting of mostly liquid with bubbles of gas intermittently throughout the pipe. In contrast, spray flow consisting of mostly gas with either mist, or small, dispersed liquid regions present. Different flow regimes are illustrated in fig. 2.1.

2.2 Slugs

By definition, and as can be seen from fig. 2.1, slug flow occurs when the gas-liquid boundary reaches the ceiling of the pipe. From the slug's inception it will continue to travel as a continuum throughout the pipe. During the slugs journey, it may, or

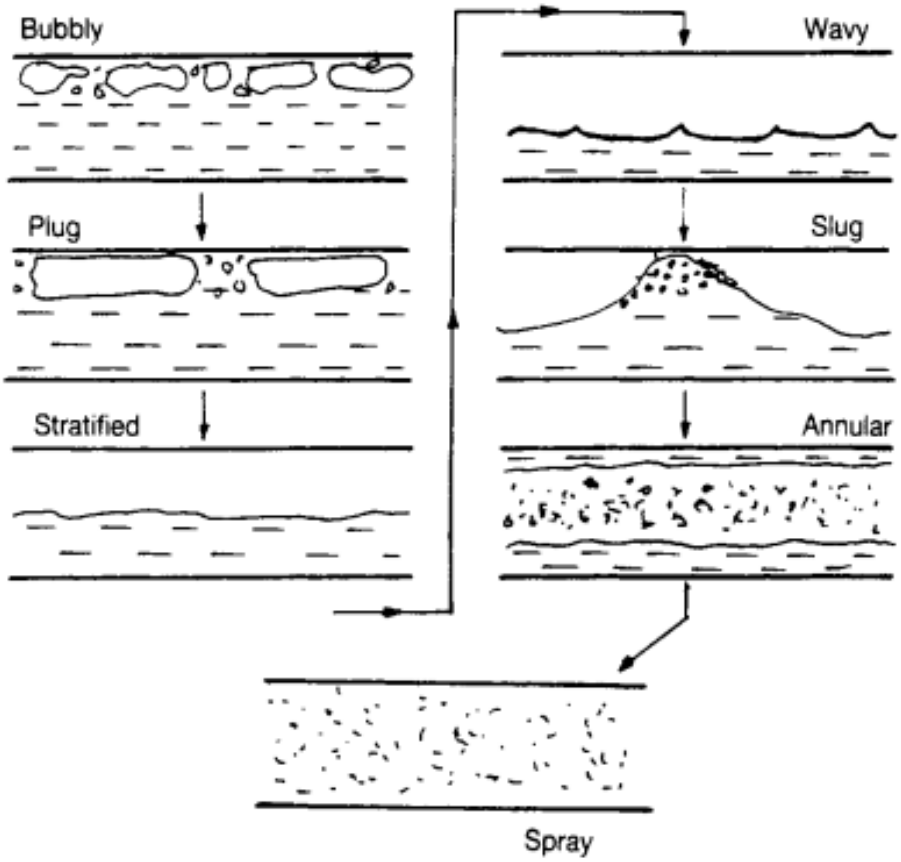


Figure 2.1: Illustrated flow regimes in horizontal gas-liquid two-phase flow. The arrow indicates the order from most liquid to least liquid. The illustration is taken from Holland and Bragg [25].

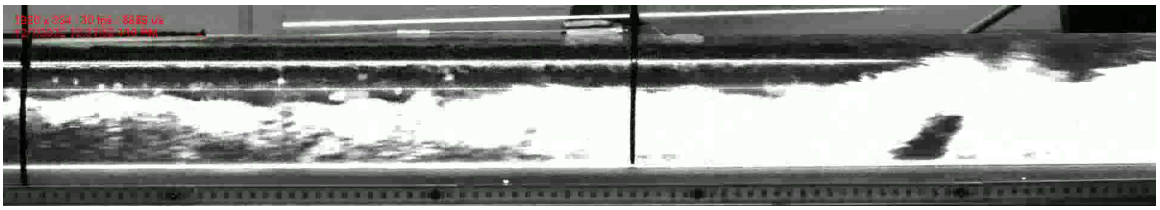
may not disintegrate, depending on the surrounding pressures and the specific flow regime experienced in the pipe.

2.2.1 Aerated slug

When a slug partly consists of dispersed air bubbles, it is said to be aerated. For this to occur, not only must the conditions for slug formation be met, but in addition the gas velocity must be sufficiently high. An aerated slug is shown in fig. 2.2, where the aerated nature is evident from the light diffraction. Due to the generally higher gas velocity, the aerated slugs tend to traverse faster through the pipes than the non-aerated alternative; plug slugs. This could be problematic with regard to fluid seperators, due to the difficulties in seperating mixed media, the incoming velocity,

and the contrast in composition from a temporal perspective.

(a) The nose of an aerated slug moving from left to right.

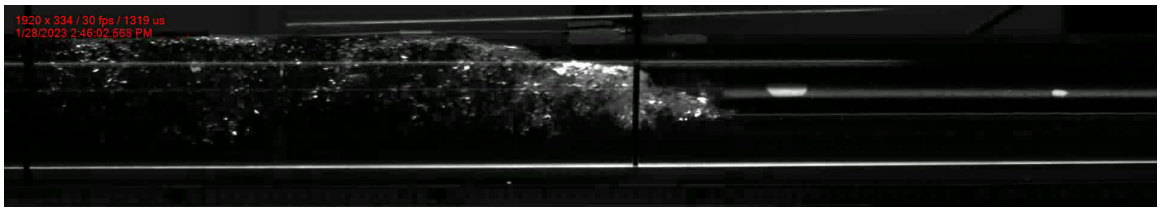


(b) The tail of an aerated slug moving from left to right.

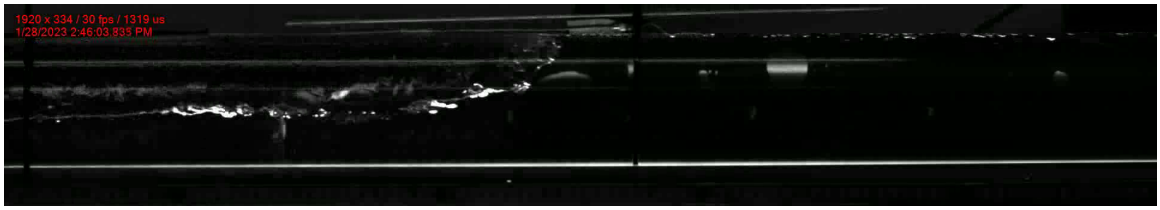
Figure 2.2: Examples of the nose and tail for an aerated slug. Note the diffused light from the underlying LED light, as described in chapter 4.

2.2.2 Plug flow

For lower gas-velocities, gas is not mixed into the liquid and thus the slug is almost entirely liquid. In fig. 2.1 the plug flow illustrated shows the plug itself as a region of gas encapsulated by liquid at the nose and tail, in addition to a thin film of liquid between the gas plug and pipe boundary. Alternatively, the plug could consist of the liquid fluid. Some minor caveats: In this case there is no film separating the plug from the boundary, and dependent on the pipe dimensions, pressures and fluid velocities, the region of gas between liquid plugs could be quite long. Hereby plug slug will be used to describe liquid plugs, as shown in fig. 2.3.



(a) The nose of a plug slug moving from left to right.



(b) The tail of a plug slug moving from left to right.

Figure 2.3: Examples of the nose and tail for a plug slug. Note how there is practically no diffusion of the light when compared with fig. 2.2.

Chapter 3

Machine Learning

> In the kingdom of the blind, the one-eyed man is king.
>
> *Desiderius Erasmus*
> (1466 - 1536)

Mankind has always drawn inspiration from nature, and this inspiration is reflected in the tools we utilize. Ancient tribes used shamanistic masks depicting animal faces in religious ceremonies, the clothes worn by early humans were usually made from animal fur or hide, and even today the influence of nature's design is apparent.

For instance, the kingfisher is a clade (group of species with common ancestor) that contain species which engage in terrestrial foraging, and species that hunt by way of plunge-diving. By studying the beaks of the respective species, scientists were able to deduce that the plunge-diving species had a smaller deceleration when passing the water barrier, both experimentally and by way of computation [26].

These results have been applied in the design of modern, Japanese bullet trains. Whenever a high-speed train enters a tunnel, it goes from a low pressure, low resistance environment into an environment with higher resistance, much like how a kingfisher travels from low-density air into high-density water. By designing the front of the train in a matter which resembles the plunge-diving kingfishers beak, the air pressure in the tunnel was reduced by 30%, and the use of electricity was reduced by 15% even though the train experienced a speed increase of 10% [1][27].

Not all problems require a physical alteration or tool, however, and the solutions to complicated problems or systems are seldom obvious. Nature's solution to the

[1]The nose of the train was not directly modelled after the kingfishers beak, but data analysis of bullets entering pipes concluded that the ideal shape is almost identical to that of the kingfishers beak

predicament of complexity was to evolve a brain; a collection of neurons which activate in specific pathways by way of electrical signals. The brain remains a scientific mystery, but the basic neuron itself is somewhat understood. The complex behavior of out own mind emerges from the connections between somewhat simple agents.

Drawing inspiration from this structure, researchers McCulloch and Pitts laid the foundation from which the biggest innovations of the 21st century would emerge. In 1943, the perceptron was born, and it is now known as the McCulloch-Pitts neuron (MP Neuron) [28].

As a consequence of the neurological inspiration, the MP Neuron only received boolean values as input, mimicking how neurons either fire a signal or don't. In addition, the weights used in the calculation were manually implemented. The MP Neuron would then be able to binary classify an input of length n by testing whether $f(\mathbf{x}, \mathbf{w})$ was a positive or negative number, with $f(\mathbf{x}, \mathbf{w})$ given by

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{x} \cdot \mathbf{w}, \tag{3.1}$$

$$f(\mathbf{x}, \mathbf{w}) = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n. \tag{3.2}$$

This was later expanded upon, most notably by Rosenblatt (which first coined the term perceptron) in 1958 [29]. His perceptron accepted not only boolean values, but any real number along with having learnable weights [30]. The perceptron started seeing some use at this point, but it could still only approximate linear solutions. Due to this, direct inspiration from the brain halted. Scientists were not finished with the perceptron, and evolution has since been rapid.

In recent times machine learning in the form of artificial intelligence has garnered societal attention with tools such at ChatGPT and Dall·E 2 which build upon technologies such as the transformer [31], and advances within stable diffusion [cite ramesh], respectively. Artificial intelligence and machine learning are here to stay, and if we are to believe Google's keynotes presented May 2023 [2], will be implemented into more existing technologies in the years to come. The understanding of artificial intelligence and machine learning is more important now than ever, so lets start at the basics.

3.1 The basics

Machine learning is a regression method, capable of making predictions, much like linear regression. Unlike linear regression, machine learning models are able to make more accurate predictions on data which is not suited for other classical regression methods.

[2] https://io.google/2023/program/396cd2d5-9fe1-4725-a3dc-c01bb2e2f38a/

The machine learning *model* is capable of improving its predictions through *training*. The training is done on collected data, referred to as *training data*. In order to validate the performance of the model, *validation data/test data* is used. This is data which is previously unseen by the model.

Training and testing data are important as we risk the model simply remembering the correct features of the training data, rather than gaining insight into the overarching trends in the features. This can lead to a problem within machine learning known as *overfitting*, where the model is substantially better optimized for predictions on the training data than the test data.

When the model trains using the training data it is said to undergo one *epoch* when it has seen every datasample from the training dataset once.

3.2 Feed-Forward Neural Networks

The Feed-Forward Neural Network is the simplest way of orchestrating a neural network, and is achieved by having the neurons (hereby node) structured in layers. This kind of network is usually illustrated by having the nodes represented by circles, and the connections between them represented by arrows indicating the direction the information travels.

The initial layer of the network is known as the input layer. This is the layer that receives the information which the network utilizes in its further calculations and is represented in orange in figure 3.1. A distinguishing feature of the input layer is that it does not receive information from any nodes, in contrast to the hidden layer as well as the output layer.

The hidden layer (or layers) is responsible for the intermediate calculations made by the network. If a neural network contains more than one hidden layer, we say that the network utilizes deep learning, a sub-category of Machine Learning as shown in figure 3.2. In figure 3.1 the hidden layers are represented in green. The hidden layers are where the magic of the neural network takes place. Partly because the connections to and from the hidden layer compose the bulk of the network itself, but perhaps most importantly, because the nodes in the hidden layers are affected by activation functions. Activation functions and their consequences will be expanded upon later in this chapter.

Lastly, the output layer returns the values calculated by the model. Depending on the specific implementation, the output layer could be prone to an activation function, but this is usually not the case. When no activation function is used and the model should solve a classification problem, the model returns a so-called logit. For the statisticians, this is an abuse of naming, as the logit (logistic unit) is defined to be a function which maps probabilities from $\mathbb{R}[0, 1]$ to $\mathbb{R}(-\infty, \infty)$. Following Berkson's

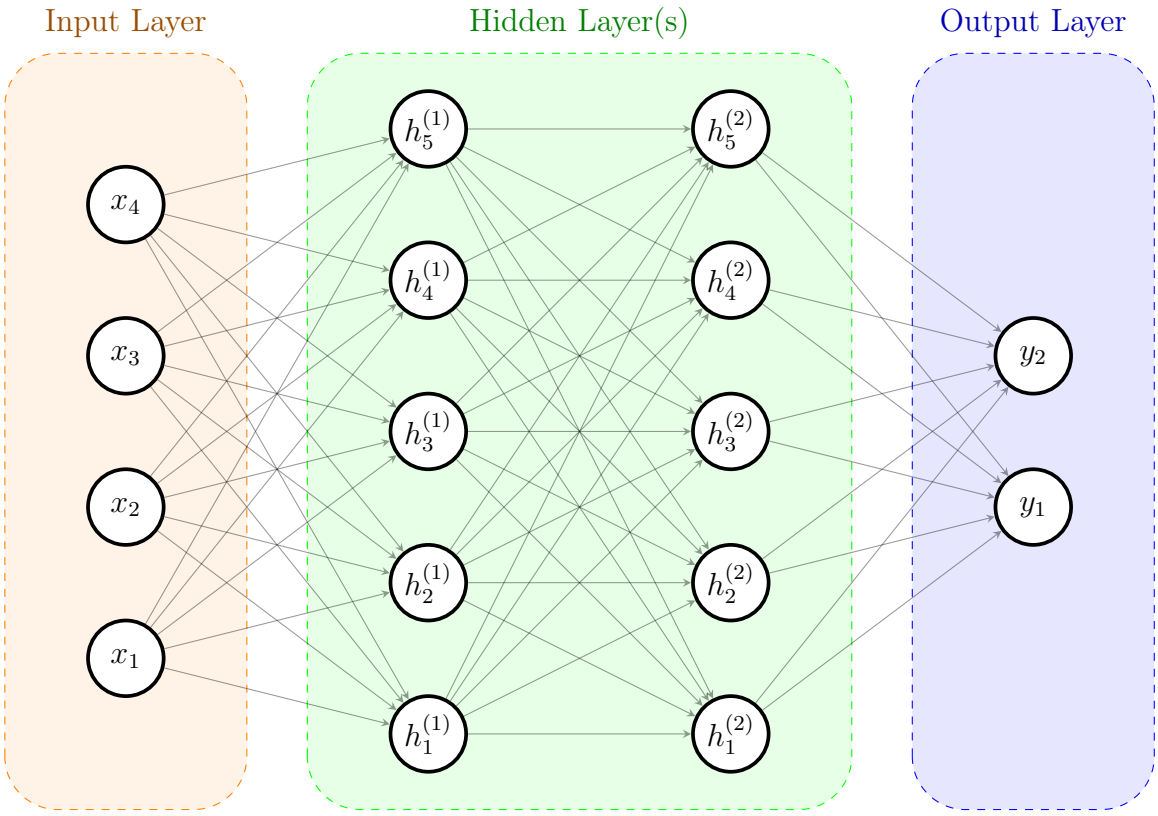


Figure 3.1: Illustration of a simple Feed-Forward Neural Network, containing two hidden layers.

definition from 1944 [32], we have that the logit is given by the taking the logarithm of the odds; $\text{logit}(p) = \ln\left(\frac{p}{1-p}\right)$.

This mapping was motivated by the ability to use linear regression methods on a domain where the output is a probability. At this point the connection somewhat reveals itself, as the neural network provides "probabilities" for which class the provided data belongs to. Of course, the neural network is a non-linear regression method, and the raw logits violate the Kolmogorov axioms of probability [33], but the terminology still remains.

With that small digression taken care of, we can turn our attention to why the regression is non-linear in the first place; the implementation and use of activation functions.

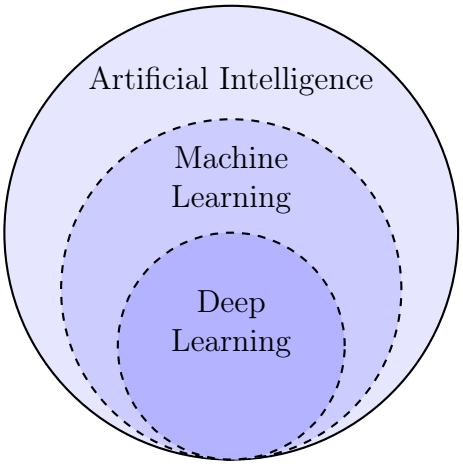


Figure 3.2: Crude categorization of Artificial Intelligence frameworks. Note that the borders are somewhat blurred, this definition is in alignment with Goodfellow et al. [30].

3.2.1 Activation Functions

One of the parameters which determine the final performance of a neural network is the activation function which works on each of the nodes in a layer. Usually the activation function is the same throughout the layer, but in theory this need not be the case. In practice, however, this is heavily encouraged, and is standard across the main frameworks for machine learning. Denoting the activation function as f, we have that the output of a single neuron is given by

$$f(\mathbf{x} \cdot \mathbf{w} + b), \tag{3.3}$$

where, as usual, $\mathbf{x}$ is the output of the previous layer, $\mathbf{w}$ are the accompanying weights, and b denotes the bias.

There are multiple choices for f, a sample of the most common is given in table 3.1. An important property of the activation function is that its derivative is known. This property is used in the back-propagation algorithm, thus the derivative of the activation function is also provided. In practice, however, it is not required that the activation function has a defined derivative at all inputs. Using numerical differentiation, for instance, in the form of autograd, the derivative need not even be implemented manually, sidestepping having to know the derivative of the activation function in its entirety.

There are not many requirements an activation function must adhere to, but it is worth noting that the activation function should be monotonic. In addition, for non-linear regression there must be at least one layer of non-linear activation functions.

Activation Function	**Definition,** $f(x)$	**Derivative,** $\frac{\mathrm{d}f}{\mathrm{d}x}$
Sigmoid/Logistic	$\frac{1}{1+e^{-x}}$	$\frac{e^{-x}}{(1+e^{-x})^2} = f(x)(1-f(x))$
Rectified Linear Unit	$\begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$	$\begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$
Linear/Identity	x	1
Tanh	$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$1 - \frac{\left(e^{x}-e^{-x}\right)^2}{(e^{x}+e^{-x})^2} = 1 - f^2(x)$

Table 3.1: Definitions and derivatives for some of the most commonly used activation functions.

It is this implementation of non-linear activation functions which is the reason our neural network can approximate not only linear functions, but, in fact, any function! This is summarized in the Universal Approximation Theorem.

Universal Approximation Theorem

The Universal Approximation Theorem states that the error of an approximation provided by a neural network that follows the feed-forward architecture, can be made arbitrarily small. There are two ways this can be achieved; namely the *arbitrary width* case, and the *arbitrary depth* case.

The Arbitrary width case was initially proven by Cybenko in 1989 [34] who showed that the feed-forward neural network is a universal approximator when using the Sigmoid activation function. Later Hornik, Stinchcombe and White showed that a network with one hidden layer and a finite amount of nodes is sufficient for the network to be a universal approximator [35].

For the depth case, progress was a bit more slow. Work continued none the less, and after a series of papers it was shown that the feed-forward neural network is a universal approximator for a range of activation functions [36–39].

These proofs rely primarily on representation and measurement theory and is outside the scope of this thesis. None the less, the results are still valid, providing Neural Networks with the theoretical foundation required for rigorous use.

3.2.2 Mathematics behind the forward pass

With an understanding of the components of the neural network, we may now dive into how predictions are made. For the Feed-Forward Neural Network, this is known as a forward pass. As a quick reminder; the input layer has no previous layer, and

the output layer has the linear activation function. That is, we do not set restrictions on the predictions provided by the network.

A single node in a fully connected feed-forward neural network (often referred to as MultiLayer Perceptrons, MLP) receives as its input all the outputs from its preceding layer in the network multiplied by some tune-able weights. In addition to the weighted previous output, the input has a tune-able bias added as well. Introducing the notation z_i^l as the input to node i in layer l, and a_i^l as the output (activation of input), we have

$$z_i^l = \left(\sum_{j=1}^{N_{l-1}} w_{ij}^l a_j^{l-1} \right) + b_i^l, \tag{3.4}$$

and

$$a_i^l = f^l(z_i^l), \tag{3.5}$$

where it is assumed that all nodes in the same layer have the same activation function, f^l. Moreover, w_{ij}^l is the weight connecting node j from layer $l-1$ to node i in layer l, a_j^{l-1} is the output, or activation, from node j in layer $l-1$, b_i^l is the bias added to node i in layer l, and N_{l-1} is the number of nodes in layer $l-1$. The data returned from the input layer will be the same as the data provided to the input layer. That is $a_i^0 = x_i$ where $\mathbf{x}$ is the input.

Let us now consider a fully connected neural network with $L+1$ layers. The final output, or prediction, of the model is then usually denoted by $\hat{y}$ as to follow the convention of $f(x) = y$, in combination with hat denoting that this is a prediction. The final prediction must then be given by

$$\hat{y}_i = f^L\left(z_i^L\right) \tag{3.6}$$

$$= f^L\left(\left(\sum_{j=1}^{N_{L-1}} w_{ij}^L a_j^{L-1} \right) + b_i^L \right) \tag{3.7}$$

$$\vdots \tag{3.8}$$

$$\hat{y}_i = f^L\left(\sum_{j=1}^{N_{L-1}} w_{ij}^L f^{L-1}\left(\sum_{k=1}^{N_{L-2}} w_{jk}^{L-1} f^{L-2}\left(\dots f^1\left(\left(\sum_{n=1}^{N_0} w_{mn}^1 a_n^0 \right) + b_m^1 \right) \dots \right) + b_j^{L-1} \right) + b_i^L \right), \tag{3.9}$$

where f^L usually is the linear function, as described in table 3.1.

The same calculations can also be described in terms of linear algebra. By now letting the input and output be given as vectors, we have

$$\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l \tag{3.10}$$

where

$$\mathbf{a}^l = \begin{bmatrix} f^l(z_1^l) \\ f^l(z_2^l) \\ \vdots \\ f^l(z_{N^l}^l) \end{bmatrix}. \tag{3.11}$$

$\mathbf{W}^l$ denotes the matrix containing the weight from layer $l-1$ to layer l, and $\mathbf{b}$ denotes the bias. Performing the calculation between layer $l-1$ containing N nodes, and layer l containing M nodes

$$\mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \dots & w_{1N}^l \\ w_{21}^l & w_{22}^l & \dots & w_{2N}^l \\ \vdots & & \ddots & \vdots \\ w_{M1}^l & w_{M2}^l & \dots & w_{MN}^l \end{bmatrix} \begin{bmatrix} a_1^{l-1} \\ a_2^{l-1} \\ \vdots \\ a_N^{l-1} \end{bmatrix} + \begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_M^l \end{bmatrix} \tag{3.12}$$

$$= \begin{bmatrix} w_{11}^l a_1^{l+1} + w_{12}^l a_2^{l+1} + \cdots + w_{1N}^l a_N^{l+1} + b_1^l \\ w_{21}^l a_1^{l+1} + w_{22}^l a_2^{l+1} + \cdots + w_{2N}^l a_N^{l+1} + b_2^l \\ \vdots \\ w_{M1}^l a_1^{l+1} + w_{M2}^l a_2^{l+1} + \cdots + w_{MN}^l a_N^{l+1} + b_M^l \end{bmatrix} \tag{3.13}$$

$$= \begin{bmatrix} \left(\sum_{j=1}^N w_{1j}^l a_j^{l+1}\right) + b_1^l \\ \left(\sum_{j=1}^N w_{2j}^l a_j^{l+1}\right) + b_2^l \\ \vdots \\ \left(\sum_{j=1}^N w_{Mj}^l a_j^{l+1}\right) + b_M^l \end{bmatrix} \tag{3.14}$$

$$= \mathbf{z}^l. \tag{3.15}$$

By using the notation $f \circ \mathbf{v}$ for a function applied piece-wise to elements in a matrix or vector, defined as

$$(f \circ \mathbf{A})_{i,j} = f(\mathbf{A}_{ij}), \tag{3.16}$$
$$(f \circ \mathbf{v})_i = f(\mathbf{v}_i), \tag{3.17}$$

we have that

$$f^l \circ \left(\mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l\right) = f^l \circ \mathbf{z}^l \tag{3.18}$$
$$= \begin{bmatrix} f^l\left(\left(\sum_{j=1}^{N_{l-1}} w_{1j}^l a_j^{l-1}\right) + b_1^l\right) \\ f^l\left(\left(\sum_{j=1}^{N_{l-1}} w_{2j}^l a_j^{l-1}\right) + b_2^l\right) \\ \vdots \\ f^l\left(\left(\sum_{j=1}^{N_{l-1}} w_{Mj}^l a_j^{l-1}\right) + b_M^l\right) \end{bmatrix}, \tag{3.19}$$

such that

$$\left(f^l \circ \left(\mathbf{W}^l \mathbf{x}^{l-1} + \mathbf{b}^l\right)\right)_i = \left(f^l \circ \mathbf{z}^l\right)_i \tag{3.20}$$
$$= f^l\left(\left(\sum_{j=1}^{N_{l-1}} w_{ij}^l a_j^{l-1}\right) + b_i^l\right), \tag{3.21}$$

which we recognise as eq. 3.5.

Using this notation further, we can again express the final prediction as

$$\hat{\mathbf{y}} = f^L \circ \left(\mathbf{W}^L f^{L-1} \circ \left(\mathbf{W}^{L-1} f^{L-2} \circ \left(\dots f^1 \circ \left(\mathbf{W}^1 \mathbf{x} + \mathbf{b}^1\right)\dots\right) + \mathbf{b}^{L-1}\right) + \mathbf{b}^L\right), \tag{3.22}$$

with $\mathbf{x}$ being the input to our fully connected feed forward neural network.

From the universal approximation theorem, we know that the prediction $\hat{\mathbf{y}}$ can be made arbitrarily precise for valid values of f, but how can we achieve this? Note that $\mathbf{x}$ is a fixed value and the activation functions are predefined and hence not subject to change. Thus the only options are to change the weights and biases. In order to change the weights and biases in a favourable manner, we need to measure the error made by the model. This is achieved through the use of loss functions.

3.2.3 Loss functions

Choosing the metric from which to judge the performance of our neural network is instrumental when it comes to the final performance. This is a mathematical consequence of how the model improves itself, or trains. The loss function, or cost function (used interchangeably), will provide some metric as to how close the prediction provided by the model is to some predefined label. The label will then function as the ground truth.

Ideally, the difference between prediction and label is zero, corresponding to a perfect prediction. Thus the loss function should be minimized, in order to ensure best possible predictions.

There exists a myriad of loss functions, each reflecting the specific purpose of the neural network in question. In our case, it is natural to sort them into two main groups. These are; Classification and Regression.

For classifications, the output is often provided as a vector in which each element correspond to one of the possible classes. Both the vector returned from the neural network, and the label are on the form $\mathbf{y} \in \mathbb{R}^N$ for N distinct possible classes. Depending on whether there are one or more correct classes, the label will have one or more of its elements equal to 1, corresponding to the correct label. The rest off the labels will be zeros, as depicted in fig. 3.4. This kind of label is often referred to as one-hot encoding, or a one-hot vector.

In the regression case, we are free to use any method for measuring distance in $\mathbb{R}^N$. Drawing inspiration from the standard linear regression, the mean squared error is most commonly used.

In this thesis work two loss-functions were utilized, one for classification and one for regression.

The loss function used for classification is the so-called Cross Entropy Loss function, and is defined as

$$L_{CE}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{1}^{N} y_i \log(\hat{y}_i), \tag{3.23}$$

where the y_i's are the labels, and $\hat{y}_i$ is the prediction from the model, with some treatment. This treatment is the softmax function, defined as

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}, \tag{3.24}$$

which scales the logit output from the model element-wise. The scaled output is now a vector of the same dimensionality, but with its elements in the closed range

$[0, 1]$. This is a necessary step as logits from the model could be negative, which would certainly make the calculations in L_{CE} end rather disastrously. Moreover, if $\hat{y}_i > 1$ then $\log(\hat{y}_i) > 0$ which could result in $L_{CE} < 0$. Seeing as the model aims to minimize L_{CE}, this would guide the model into predicting completely erroneous values that would still provide the smallest loss metric. However, as the values used in the loss calculation are now in the region $[0, 1]$, $\log(\hat{y}_i) \leq 0$, thus ensuring $L_{CE} \geq 0$.

An example of the application of the softmax function, and an example calculation of the Cross Entropy Loss are shown in figs. 3.3 and 3.4.

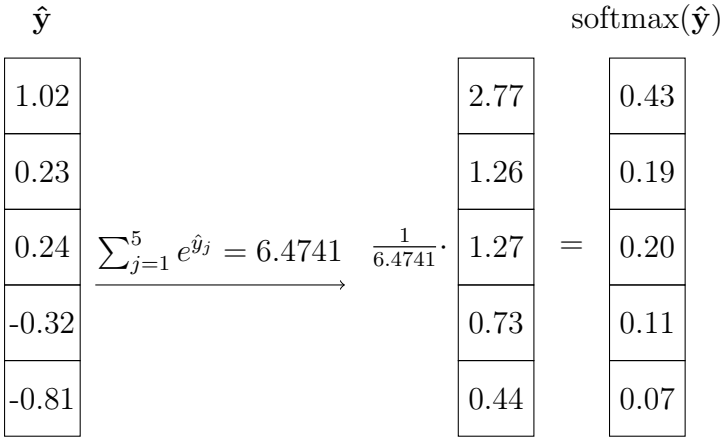
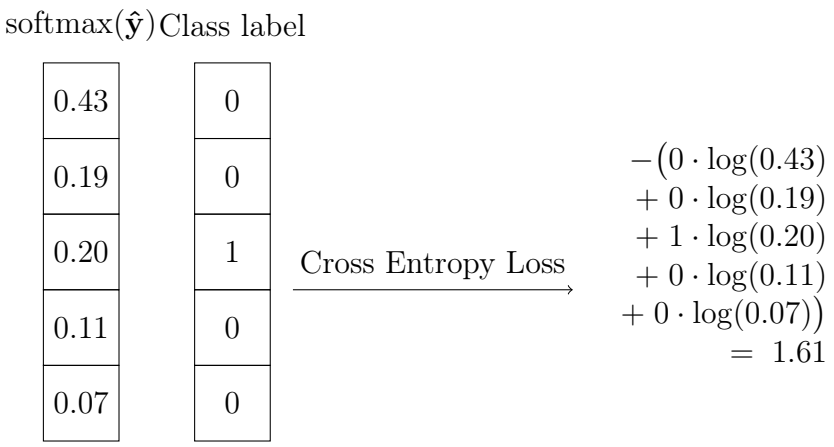
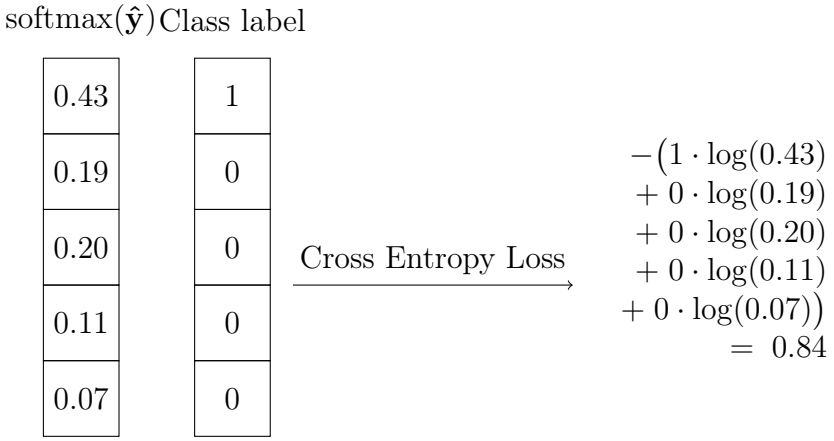


Figure 3.3: Illustration of the softmax function applied to a rank 1 tensor. Note how the elements of the tensor softmax($\hat{\mathbf{y}}$) sum to 1.



(a) Cross Entropy Loss with a weak prediction.



(b) Cross Entropy Loss with a stronger prediction.

Figure 3.4: Illustration of how the Cross Entropy Loss function acts on a rank 1 tensor which has undergone a softmax transformation.

In the regression case, the Mean Squared Error was used as a loss function, and is defined as

$$\text{MSE}(\mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{i=1}^{N} (x_i - y_i)^2. \tag{3.25}$$

Seeing as the intention of this thesis work is to regress a value, and not a function, the MSE is reduced to $(y - \hat{y})^2$. Similarly to the classification case, the loss function for regression has a lower bound of zero as the predicted values are real. That is $\text{MSE} \leq 0$ because $y, \hat{y} \in \mathbb{R}$.

3.2.4 Update weights and biases; Gradient descent

Before embarking on what is, perhaps, the most crucial algorithm in classical nerual networks, a summary is in order. Thus far we have seen how data enters a neural network, and how the values in the next layers are calculated, by the use of weights, biases and activation functions. Finally, the prediction is evaluated by some metric which is defined through a loss function.

The structure of the network, activation functions and loss functions are defined, and thus not subject to change. Yet we want a model which improves for each iteration. We are left with no other choice than to change the weights and biases in order to improve the precision of the predictions. This is done through backward propagation, and was first introduced by Rumelhart et al. [40].

The Backpropagation Algorithm

In order to change the weights and biases such that the loss is minimized, we need to know how it changes with respect to each weight and bias. As explained by Nielsen in [41], there are four essential equations which are needed in order to calculate these values.

Let a neural network consist of $L+1$ layers, $l = 0, 1, \ldots, L$. Let δ_j^l denote the *error* at node j in layer l, defined $\delta_j^l = \frac{\partial C}{\partial z_j^l}$, where C is the value of the loss function (here denoted C because of its interchangeable name; cost function, and in order to avoid confusion with layer number). Let f denote the loss function and f' its derivative. For the final layer of the network, we have the error done by node j given as

$$\delta_j^L = \frac{\partial C}{\partial z_j^L}. \tag{3.26}$$

Using the chain rule, and the definition of a_k^L as given in 3.5, we have that

$$\delta_j^L = \sum_k \frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L} \tag{3.27}$$

$$= \frac{\partial C}{\partial a_j^L} f'(z_j^L). \tag{3.28}$$

Calculating both $\frac{\partial C}{\partial a_j^L}$ and $f'(z_j^L)$ is straight forward, especially when the derivative of the cost function and activation function is known.

We can then write 3.28 as a vector δ^L. By writing the first factor as $\nabla_{a^L} C$ and the second factor as $f' \circ z^L$, the element wise product would be equal to $\delta.^L$. This can be

achieved by the Hadamard product, denoted by $\odot$, and defined such that $(A \odot B)_i = A_i B_i$. The error from the final layer L is then given as $\delta^L = \nabla_{a^L} C \odot \left(f' \circ z^L\right)$.

We now need to find the error from the previous layers in the network, which can be done by expressing the error for a given layer as the error in the following layer. By definition we have

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \tag{3.29}$$

and, again, by the chain rule we have

$$\delta_j^l = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} \tag{3.30}$$

$$= \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l}. \tag{3.31}$$

Using that $z_k^{l+1} = \left(\sum_j w_{kj}^{l+1} a_j^l\right) + b_k^{l+1} = \left(\sum_j w_{kj}^{l+1} f(z_j^l)\right) + b_k^{l+1}$, we have $\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} f'(z_j^l)$. The error for an arbitrary layer l is thus expressed by the error its following layer as

$$\delta_j^l = \sum_k \delta_k^{l+1} w_{kj}^{l+1} f'(z_j^l). \tag{3.32}$$

Again, we may express this as a vector

$$\delta^l = \left(\left(W^{l+1}\right)^T \delta^{l+1}\right) \odot \left(f' \circ z^l\right). \tag{3.33}$$

We are now able to calculate the error for any layer in the network by first calculating the error in the final layer, then iterating backward. We can then find the final partial derivatives we are after, the change in cost given by the change in weights, and the change in bias.

Starting with the bias case, again using definition 3.26 in combination with the chain rule:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \tag{3.34}$$

$$= \frac{\partial C}{\partial b_j^l} \frac{\partial b_j^l}{\partial z_j^l} \tag{3.35}$$

$$= \frac{\partial C}{\partial b_j^l} \frac{1}{\frac{\partial z_j^l}{\partial b_j^l}}. \tag{3.36}$$

Using $z_j^l = \left(\sum_k w_{jk}^l a_k^{l-1}\right) + b_j^l$, it is easy to calculate $\frac{\partial z_j^l}{\partial b_j^l} = 1$, thus $\frac{\partial C}{\partial b_j^l} = \delta_j^l$.

Similarly, by observing that $\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1}$, we can calculate

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \tag{3.37}$$

$$= \frac{\partial C}{\partial w_{jk}^l} \frac{\partial w_{jk}^l}{\partial z_j^l} \tag{3.38}$$

$$= \frac{\partial C}{\partial w_{jk}^l} \frac{1}{\frac{\partial z_j^l}{\partial w_{jk}^l}} \tag{3.39}$$

$$= \frac{\partial C}{\partial w_{jk}^l} \frac{1}{a_k^{l-1}}, \tag{3.40}$$

thus $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$, and we have our final expression needed for the full backpropagation algorithm, which is given as algorithm 1.

Algorithm 1 The backpropagation algorithm, as explained by Nielsen [41]. Note that the omission of subscript indicates vector.

Require: x ; Input
Require: W, b; Initial weights and biases
1: $a^0 \leftarrow x$
2: **for** $l = 1, 2, \ldots, L$ **do**
3: $\quad z^l \leftarrow w^l a^{l-1} + b^l$
4: $\quad a^l \leftarrow f \circ z^l$
5: **end for**
6: $\delta^L \leftarrow \nabla_{a^L} C \odot \left(f' \circ z^L\right)$
7: **for** $l = L-1, L-2, \ldots, 1$ **do**
8: $\quad \delta^l \leftarrow \left(\left(W^{l+1}\right)^T \delta^{l+1}\right) \odot \left(f' \circ z^l\right)$
9: **end for**
10: **return** Gradient of the loss function given by $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$ and $\frac{\partial C}{\partial b_j^l} = \delta_j^l$.

If we could now solve for $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l = 0$ and $\frac{\partial C}{\partial b_j^l} = \delta_j^l = 0$, we would have found a local (or potentially global) minima for the weights and biases. Sadly, closed form expression for these partial derivatives rarely exist. This motivates the use of other optimization methods.

Gradient Descent methods

The idea behind gradient descent is simple. Start at some point in $\mathbb{R}^D$, and take a step in the direction which lowers the value the most. This direction is by definition the negative gradient of the function one wishes to minimize.

In the following derivations the weight and biases will be grouped together in the common variable θ. Calculating the gradient can then be done as described in subsection 3.2.4. With the direction in which to step now known, the length of the step can be put into the spotlight. In general, the *step length* is denoted by the greek letter η, and thus we have everything needed to take a step in the parameter space θ. The weights and biases for the next time step $t + 1$ are then given by

$$\theta_{t+1} \leftarrow \theta_t - \eta \nabla_\theta C(\theta_t). \tag{3.41}$$

In this fashion, gradient descent will converge towards a position in parameter space where the gradient is zero. There is, however, no guarantee that the θ for which $\nabla_\theta C(\theta) = 0$ is the same θ as $\underset{\theta}{\operatorname{argmin}}\, C(\theta)$. Which is to say that there is no way to distinguish a local minima from a global minima, based on θ and $\nabla_\theta C(\theta)$. In order to handle this, two modifications are made. The introduction of stochasticity, and momentum.

Stochastic Gradient Descent

In contrast with the classic gradient descent, we no longer calculate the gradient for the entire data set. At least, not all at once. By splitting the dataset into multiple *batches* containing random samples from the dataset, we can calculate the gradient in parameter space for a single batch, $\nabla_\theta C_i(\theta)$, and then update the weights and biases accordingly. The idea is that the local minima will now change in parameter space, depending on which samples are currently in the batch. Hopefully, with the stochastic element now introduced, the path explored through the parameter space sweeps a larger "area", increasing the probability that the global minima is found. Or, at least, a better minima than the one found strictly by the full-batch gradient descent, which has one batch consisting of the entire dataset, making it equal to traditional gradient descent.

Momentum

The other method introduced is, as mentioned, momentum. In this case, the overarching idea is that the inertia, or momentum, of the descent should be taken into

account in order to overcome local minima and potentially noisy or oscillating gradients. This is quite analogous to momentum in classical mechanics, where a ball rolling down a hill could have enough kinetic energy to roll out of a local minima.

This is achieved by calculating the momentum for a the next iteration, $t+1$, as

$$\mathbf{m}_{t+1} = \gamma \mathbf{m}_t + \eta \nabla_\theta C_i(\theta_t), \tag{3.42}$$

where γ is a parameter which dictates how much of the prior momentum should be included in the current. The parameters are then updated as

$$\theta_{t+1} = \theta_t - \mathbf{m}_{t+1}. \tag{3.43}$$

The process of gradient descent with momentum is given in algorithm 2.

Algorithm 2 Stochastic gradient descent with momentum.

Require: η
Require: $\gamma \in [0, 1)$
Require: $C(\theta)$
Require: θ_0
$\mathbf{m}_0 \leftarrow \mathbf{0}$
$t \leftarrow 0$
while not converged **do**
  $\mathbf{m}_{t+1} \leftarrow \gamma \mathbf{m}_t + \eta \nabla_\theta C_i(\theta_t)$
  $\theta_{t+1} \leftarrow \theta_t - \mathbf{m}_{t+1}$
  $t \leftarrow t + 1$
end while
return θ

Adaptive Momentum

In this thesis work a stochastic gradient descent method with adaptive momentum was used. The method was published late in 2014, by Kingma & Ba [42], and was given the name ADAM. With over 140 000 citations, the optimizer has proven itself to be a rather popular choice. As to why this is the case, the authors explanation is sufficiently detailed;

"(...) an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method

is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters."[42]

The steps can, again, be summarized in a set of equations

$$\mathbf{g}_t = \nabla_\theta C_i(\theta_{t-1}), \tag{3.44}$$
$$\mathbf{m}_t = \gamma_1 \mathbf{m}_{t-1} + (1-\gamma_1)\mathbf{g}_t, \tag{3.45}$$
$$\mathbf{v}_t = \gamma_2 \mathbf{v}_{t-1} + (1-\gamma_2)\mathbf{g}_t^2, \tag{3.46}$$
$$\hat{\mathbf{m}}_t = \mathbf{m}_t / \big(1-\gamma_1^t\big), \tag{3.47}$$
$$\hat{\mathbf{v}}_t = \mathbf{v}_t / \big(1-\gamma_2^t\big), \tag{3.48}$$

and the algorithm itself is provided in algorithm 3.

Algorithm 3 The adaptive momentum algorithm, as provided by Kingma & Ba [42]. Note that $\mathbf{g}_t^2$ indicates the element-wise square $\mathbf{g}_t \odot \mathbf{g}_t$. Suitable default settings are $\eta = 0.001$, $\gamma_1 = 0.9$, $\gamma_2 = 0.999$ and $\epsilon = 10^{-8}$. The ϵ is included in order to avoid division by zero errors. All operations are done element-wise.

Require: η
Require: $\gamma_1, \gamma_2 \in [0, 1)$
Require: $C(\theta)$
Require: θ_0
  $\mathbf{m}_0 \leftarrow \mathbf{0}$
  $\mathbf{v}_0 \leftarrow \mathbf{0}$
  $t \leftarrow 0$
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $\mathbf{g}_t \leftarrow \nabla_\theta C_i(\theta_{t-1})$
    $\mathbf{m}_t \leftarrow \gamma_1 \mathbf{m}_{t-1} + (1-\gamma_1)\mathbf{g}_t$
    $\mathbf{v}_t \leftarrow \gamma_2 \mathbf{v}_{t-1} + (1-\gamma_2)\mathbf{g}_t^2$
    $\hat{\mathbf{m}}_t \leftarrow \mathbf{m}_t / (1-\gamma_1^t)$
    $\hat{\mathbf{v}}_t \leftarrow \mathbf{v}_t / (1-\gamma_2^t)$
    $\theta_t \leftarrow \theta_{t-1} - \eta \hat{\mathbf{m}}_t / \big(\sqrt{\hat{\mathbf{v}}_t} + \epsilon\big)$
  end while
  return θ

3.3 Convolutional Neural Networks

Convolutional neural networks aim to combine technology used in classical image treatment, and neural networks. The convolutional neural network is arguably the most common framework for deep learning, and because of its connection to image analysis applications range from text mining, spam detection, image classifications, audio and speech processing ans natural language processing [43].

3.3.1 Convolution

Convolution in general is a mathematical operation which takes two functions and produces a third. The operation in denoted by a star, $*$, and is defined for two continuous functions as

$$(f * g)(\chi) = \int_{-\infty}^{\infty} f(x)g(\chi - x)dx. \tag{3.49}$$

The applications are wide-spread, mostly within different forms of signal analysis. This is due to the operations ability to *combine*, or superimpose the two functions.

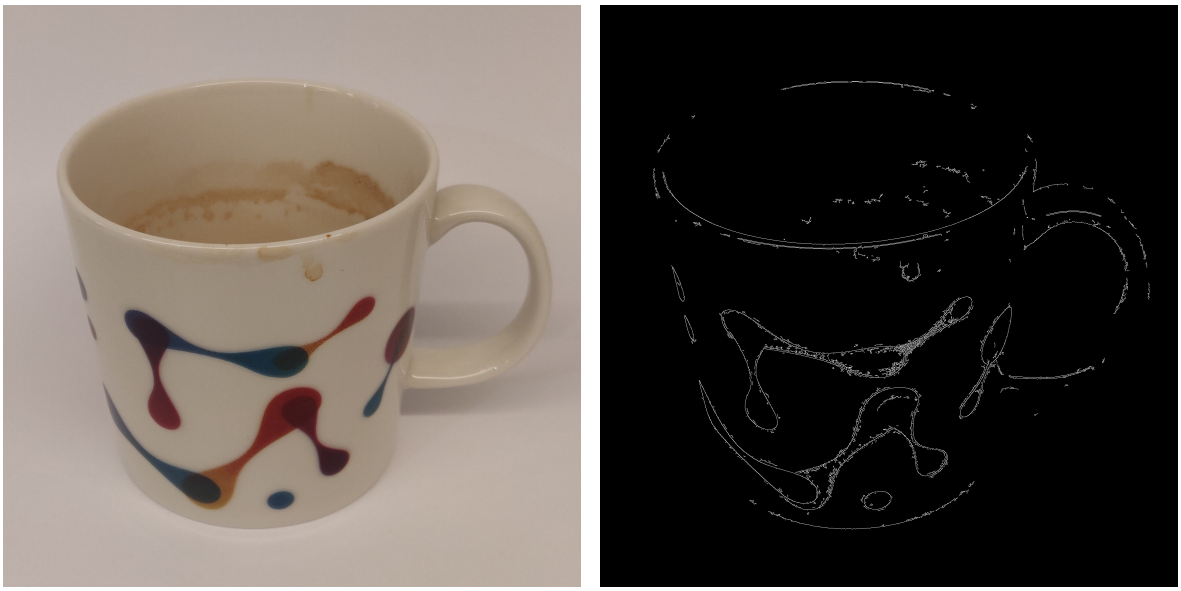
For applications on series of discrete values, which is the case for all digitally treated data, the discrete convolution is used, and is defined as

$$(f * g)[n] = \sum_{m=-\infty}^{m=\infty} f[m]g[n - m]. \tag{3.50}$$

Even though it is possible to take the convolution of two different series containing data, the application which is used in the setting of convolutional neural networks is letting one of the series be a so-called kernel. The kernel is generally a smaller series, containing specific values which alter the original series, such that their convolution has a new, desired feature. Moreover, the kernel can be used for extraction of specific features present in the original series.

Convolution is not limited to one dimension, however, and can be generalized to higher dimensions. For image analysis both the data and kernel is often given in two dimensions. Color images often contain 3 channels however, in order to include all three RGB values for a given pixel. How this is treated in the convolutional neural network is discussed in a later chapter. The two dimensional discrete convolution is defined as

$$(f * *g)[n_1, n_2] = \sum_{m_2=-\infty}^{m_2=\infty} \sum_{m_1=-\infty}^{m_1=\infty} f[m_1, m_2]g[n_1 - m_1, n_2 - m_2] \tag{3.51}$$



(a) Image of a coffee mug.

(b) Convolution of 3.5a using the Sobel kernel.

Figure 3.5: Example of a 2D convolution applied to an image in order to extract some feature. In this case the Canny edge detection algorithm has been performed, with the Sobel kernel. Note the lack of coffee present in the mug.

Again depending on a suitable kernel, features from the input data can be extracted. For images, this is illustrated in fig. 3.5.

The feature extraction property is widely used in image processing, which inspires the use of trainable kernels in machine learning.

3.3.2 Trainable kernel

In contrast with more classical image analysis, the kernels used in convolutional neural networks are not predefined. In fact, they are not even static. The values in a given kernel is subjected to the backward propagation algorithm in a similar manner to every other tuneable parameter in the network. The derivation for this back-propagation is considerably longer than that of the feed-forward neural network, and interested reader is referred to Zhang [44] for a full derivation.

The dynamic nature of the convolutional neural network kernels offers some advantages. Most importantly, the user will not need any prior information about which parts of the data is interesting. Given a sufficiently labeled dataset, the network should iteratively update the kernels such that the relevant information for the purpose of the network is extracted. Not only does this save time, as the user need

not hand craft each kernel, but the network could discover some previously ignored feature in the data which was relevant.

Powerful as it may seem, the tuneable kernel has at least one flaw in the form of new parameters, which in turn slows down the back-propagation process, due to the new calculations which need to be performed. The kernel is however only one part of the convolution performed in a convolutional neural network.

3.3.3 Data stream

The convolution part of the neural network does not only consist of the kernel applications. The output of a convolution is subject to an activation function, exactly as the output from any other node in a neural network. In addition, a process called pooling is performed. At this point it is appropriate to introduce the term tensor, at least how a computer scientist would view a tensor. From an informatics perspective, a tensor is simply a nested array. The number of indices one must specify in order to receive a single value from this tensor would determine the rank of the tensor. A single floating point number would be a rank 0 tensor, as no index is needed in order to get the value. For an array of floats, one index is needed in order to get an element, thus the array would be a rank 1 tensor, etc. The tensor transformations which occur in the convolution part of the network is as follows.

Each kernel in a convolutional layer will provide one output. Hence, for a convolutional layer with n kernels, the layer will produce $n \cdot c$ outputs, where c is the number of channels in the provided tensor. This output tensor is then subject to an activation function, which works piece-wise on all the elements in the tensor. In order to reduce the dimensionality of the tensor, pooling is performed. Pooling is simply replacing values within a given region with some statistic from the values present in said region. This new singular value will take the place of all the old values, and thus reduce the overall elements in the tensor. The most common pooling operation is the maxpool, which returns the largest of the evaluated values. How many elements the pooling operator should take into consideration is a hyperparamter for the model, a common value is pooling over a 2 by 2 area, hence reducing the dimensionality by a factor of 4.

The effects of 2D discrete convolution on a rank-2 tensor is illustrated in fig. 3.6. Similarly the effect of the activation function and pooling is illustrated in fig. 3.7. Note that if the stride of the maxpool kernel was set equal to 1, no values would be omitted, and the resulting tensor would be

$$\begin{bmatrix} 0.14 & 0.10 & 0.14 & 0.44 \\ 0.14 & 0.25 & 0.25 & 0.44 \\ 0.04 & 0.25 & 0.25 & 0.17 \\ 0.10 & 0.10 & 0.49 & 0.50 \end{bmatrix}.$$

This process; use of convolution, use of an activation function, and pooling is repeated multiple times, and in the context of convolutional neural networks is simply referred to as a convolution layer. The reduction along some dimensions due to the convolution and pooling, and the expansion along others due to the number of kernels is illustrated in fig. 3.8.

After the desired number of convolution layers, the output tensor, which should now contain all the accumulated features, is flattened. This flat rank-1 tensor serves as the input into a feed-forward neural network, which then performs the final classification or regression. No rule dictates that this feed-forward neural network has to be a deep neural network with multiple hidden layers. The network should simply perform the final calculation, but if the convolution part of the network has extracted all the relevant information from the original input tensor, excessive nodes and connections is most likely not necessary. It is worth noting that for networks performing multiple classifications and/or regressions, having a specialized convolution part might not be favorable. In this case, it is tempting to speculate that it might be more favorable to have the convolution part of the network just summarize the features into a tensor of more generalized information, and to then have the feed-forward neural network carry out the final calculations, with custom connections for their individual purpose.

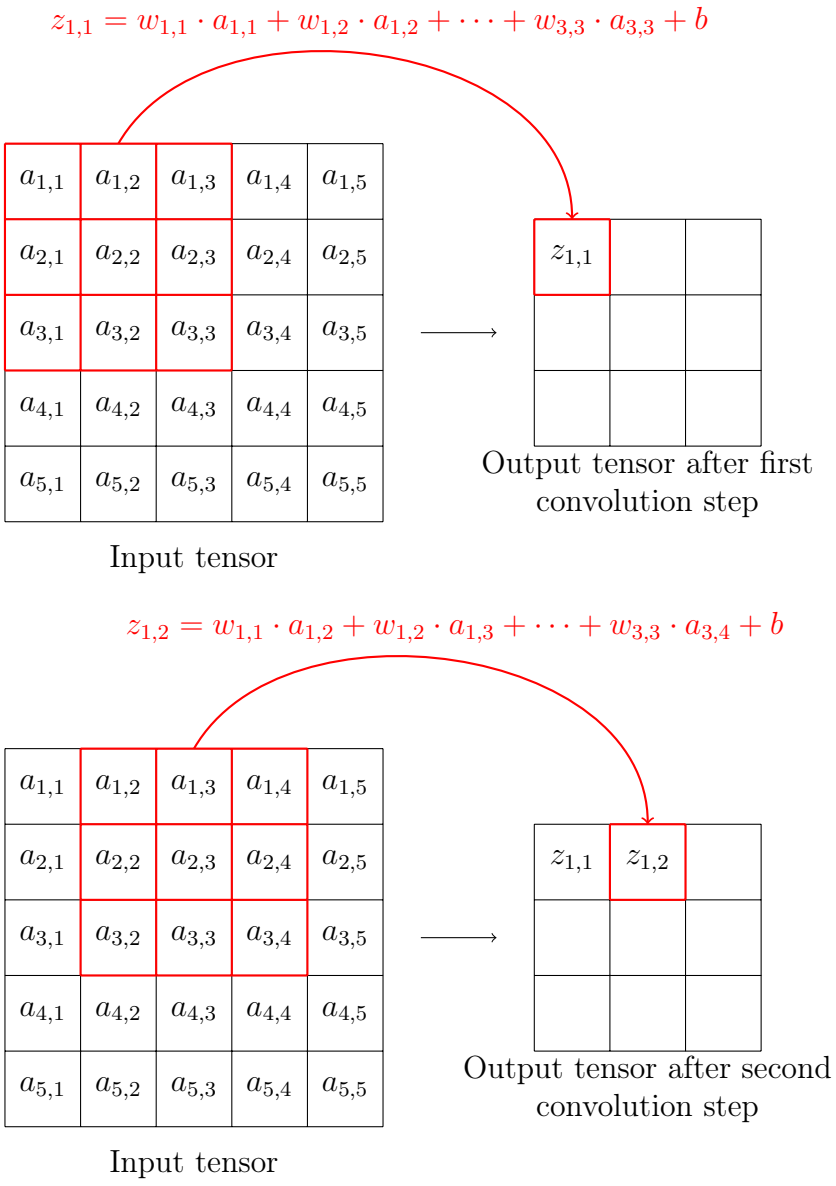


Figure 3.6: Illustration of 2D convolution on a rank 2 tensor. The kernel is a 3x3 rank-2 tensor, such that the resulting tensor is a 3x3 rank-2 tensor. In this example, the stride of the convolution kernel is equal to 1.

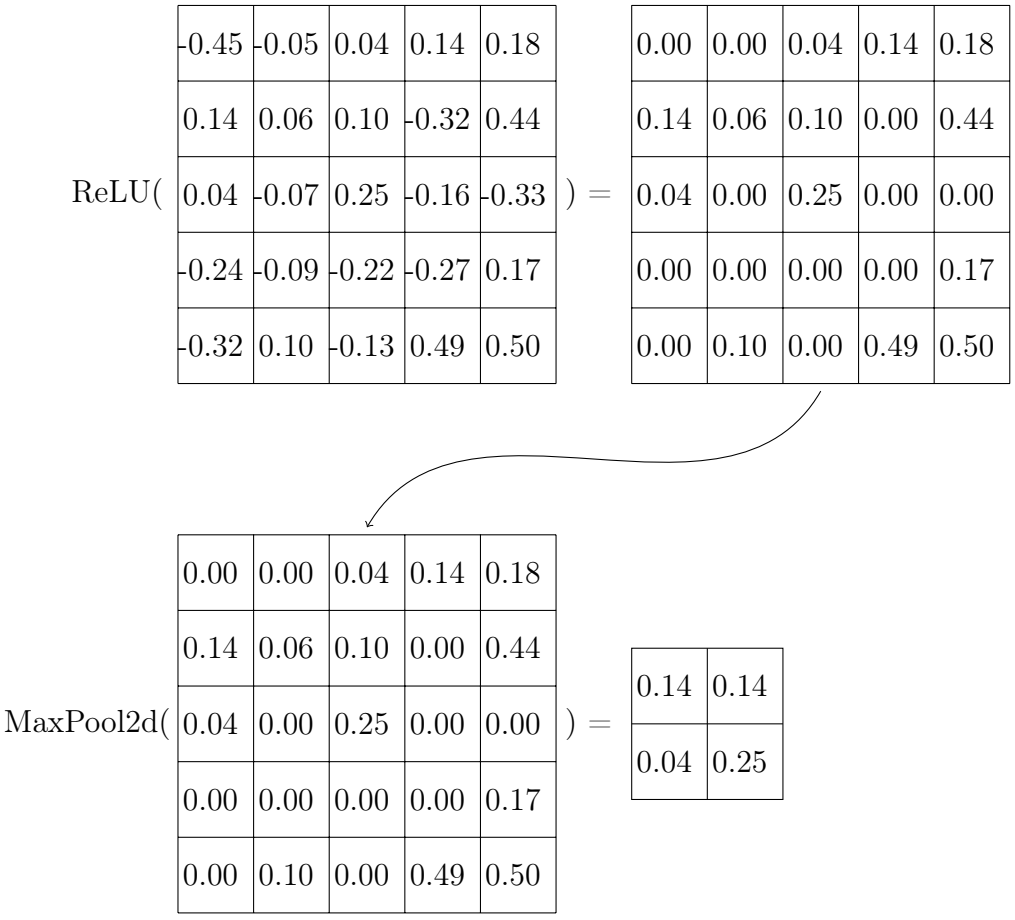


Figure 3.7: Illustration of how the ReLU activation function first works on a 5x5 rank-2 tensor, and then how the MaxPool2d function extracts the max value using a 2x2 kernel. In this example the stride of the MaxPool kernel is equal to its size, that is stride equal to 2. If the kernel would be out of bounds wrt. the tensor, the column or row in question is omitted. This is chosen deliberately as to accurately mimic how these funtions work by default in the PyTorch framework.

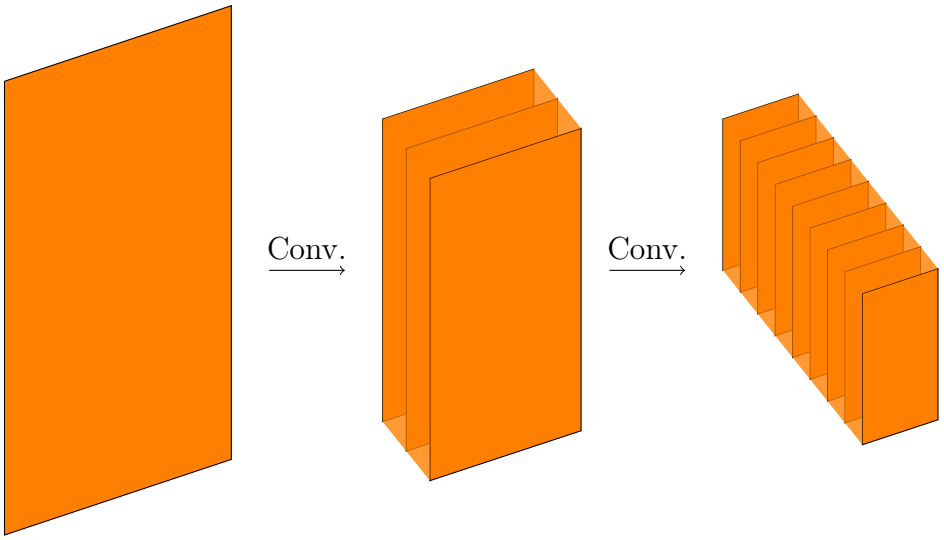


Figure 3.8: Illustration of how the dimensionality of a tensor changes when its passed through multiple convolution layers. In this case, each convolution has three kernels, such that the number of output channels is equal to the number of input channels times three.

Part III

Methodology & Implementation

Chapter 4

Laboratory setup

The data which composes the constructed dataset is gathered at the Hydrodynamics laboratory at the Institute of Mathematics, University of Oslo.

4.1 Layout

Experiments were conducted in a clear, 31m PVC pipe with an internal diameter, D=10cm. Schematics for the experimental setup is illustrated in fig. 4.1. For additional details on the pipe loop equipment, the reader is referred to Sanchis et. al [45].

The ruler which is attached underneath the pipe is visible in the captured video, albeit somewhat faintly and/or blurry. Using the ruler in combination with markers present on the pipe, it was found that one pixel in the video footage corresponds to 0.000457 meters along the pipe.

4.2 Hardware

Microphone

Three similar, but not identical, microphones (accelerometers) were used. They are from the same manufacturer, Brüel & Kjær. The type for microphones 1 and 2 is 4507-B-002, and the type for microphone 3 is 4507. One order of magnitude separates the microphones when it comes to calibration, this is accounted for by increasing the gain in the amplifiers accordingly.

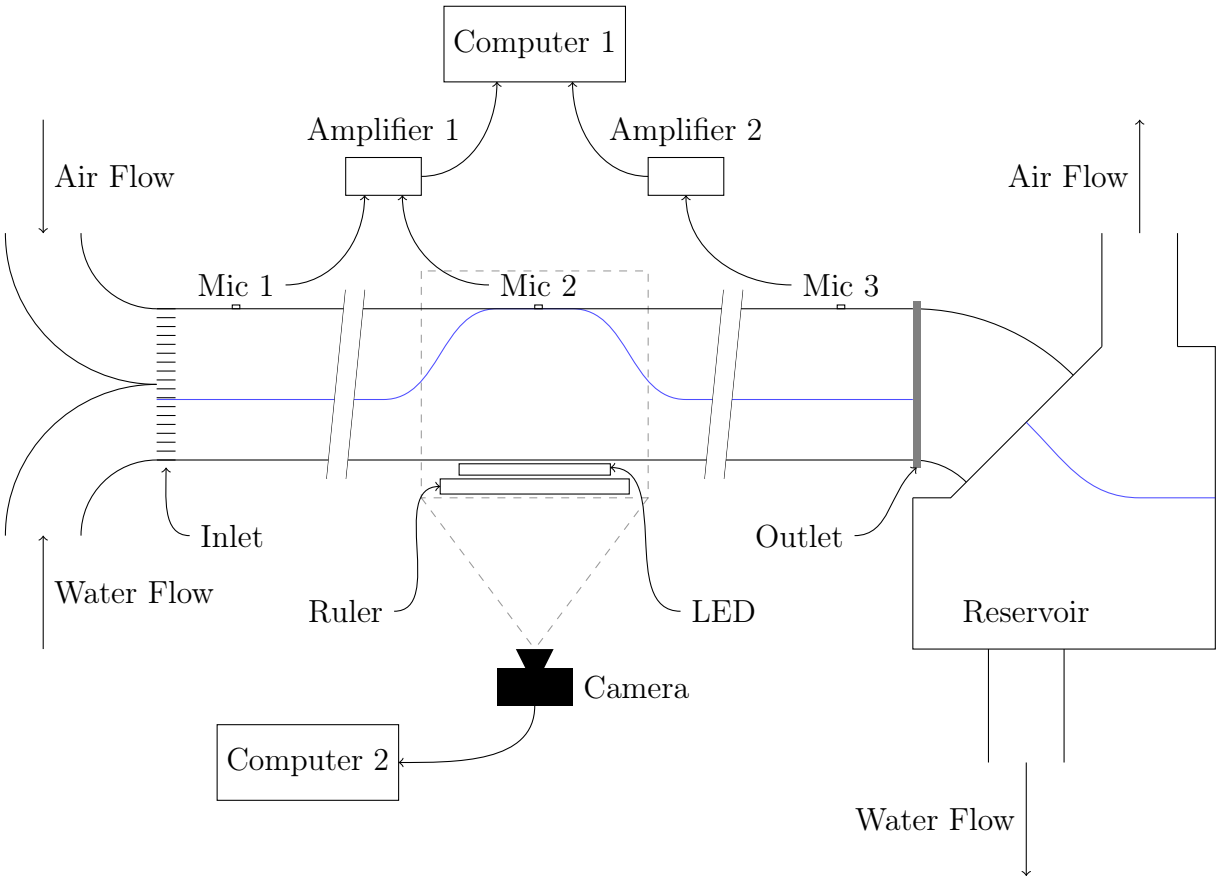


Figure 4.1: Schematics describing the experimental setup used for audio and video sampling. The horizontally striped area by the inlet indicates the honeycomb mechanism which reduces turbulence where the gas-liquid boundary is formed.

Amplifier

The linear amplifiers used are also supplied from Brüel & Kjær, and are similar, but not identical. The amplifier used for microphones 1 and 2, amplifier 1, is of the type 1704-A-002, and the type for the amplifier used for microphone 3, amplifier 2, is 1704-A-001. The gain on amplifier 2 is set to 10x the gain on amplifier 1 in order to account for the aforementioned calibration missmatch. The two amplifiers are connected to a computer, which used LabVIEW [46] for storing the audio data stream in realtime at 8000 Hz. This sampling frequency is kept constant throughout all experiments performed. The resulting measurements were written locally as `.lvm` files. The `.lvm` files consists of a header containing meta information about the experiment, along with the time for a measurement, and the three measurements corresponding to the

three distinct microphones.

Camera

The high speed camera used throughout the experiments was a Promon U1000 Mono [1] with a Carl Zeiss Planar 1,4/50 ZF.2 [2] lense. The camera was set to record an area of 1920x334 pixels at 30 frames per second. The live video was streamed to a computer, where the raw video files were stored locally. Conversion from `.raw4` to `.mp4` took place on the same computer after the experiment was conducted.

[1] https://www.aostechnologies.com/fileadmin/user_upload/PDFs/Process_Monitoring/AOS_Promon_U1000_en_2017_web.PDF

[2] https://www.zeiss.com/consumer-products/int/photography/classic/planar-1450.html

Chapter 5

Data

As in most other machine learning projects, the quality of the data is most likely what will make the project a success or a failure. After the collection of data through the setup described in chapter 4, we are left with both sound in the form of arrays of floating point numbers, and video in the `.raw4` format. The videos has since been encoded to `.mp4` files for easier handling. Treatments of both of these measurements are needed in order to extract any meaningful information from the data, as well as making it more manageable for the final neural network model at runtime.

Information about the measurements is presented in table 5.1, as the run number and/or filenames will be referenced throughout this chapter.

Filename	**Run number**	**Type**	**Length (time)**
`micData_49.lvm`	49	Aerated Slug and Breaking Wave	32 min 05 sec
`micData_51.lvm`	51	Aerated Slug and Breaking Wave	30 min 15 sec
`micData_53.lvm`	53	Aerated Slug and Breaking Wave	30 min 09 sec
`micData_54.lvm`	54	Noise	30 min 05 sec
`micData_55.lvm`	55	Noise	30 min 06 sec
`micData_58.lvm`	58	Noise	30 min 15 sec
`micData_59.lvm`	59	Plug Slug	22 min 18 sec
`micData_60.lvm`	60	Plug Slug	21 min 19 sec

Table 5.1: Overview of measurements used in the construction of the final data set. Note that "Noise" is equivalent to stratified or stratified-wavy flow, but no occurrences of slugs or breaking waves.

5.1 Data Treatment

5.1.1 Audio

The data gathered from the accelerometers is saved continuously through LabView during a run, and is saved as a `.lvm` file. Timestamps for the measurements are saved alongside the measurements for each of the accelerometers, As mentioned in section 4.2, the stored audio data consists of three channels which contain the measurements from the accelerometers. A sample of this data is shown in fig. 5.1.

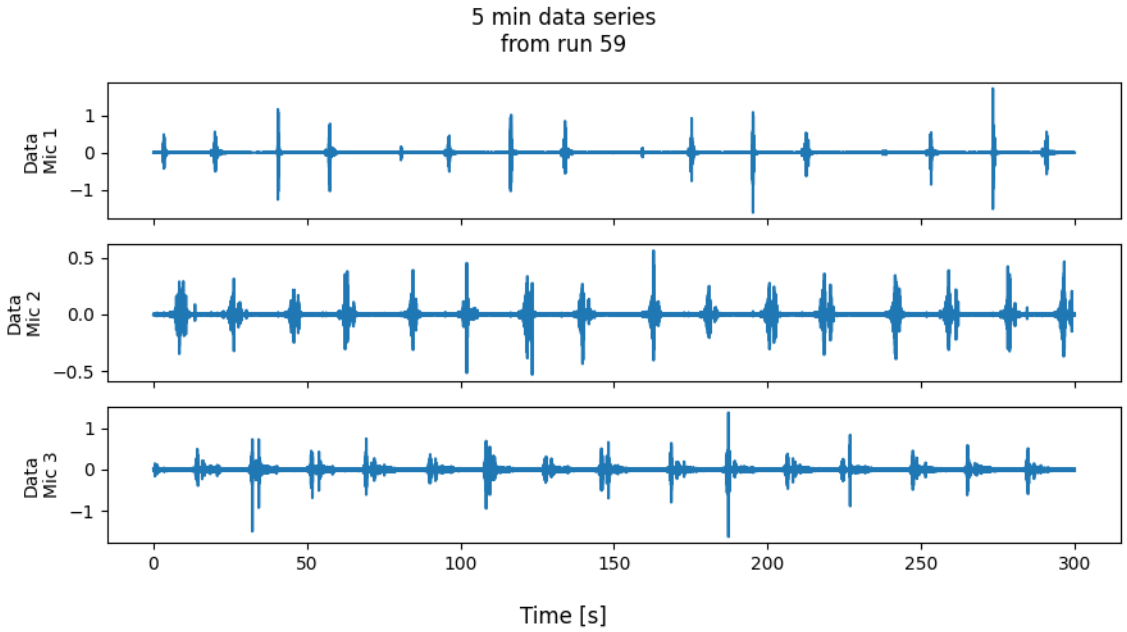


Figure 5.1: 5 minute data sample from run number 59.

Considering the length of the data files, as presented in table 5.1, the file is separated into shorter segments. The intention is for one segment to contain exactly one event, be it a slug or breaking wave. An example of an audio segment is illustrated in fig. 5.2.

A consequence of this way of splitting the data is that each segment has a different length. This is not a problem in of itself, but this should be kept in mind when splitting the noise data into segments. The intention is for the network to classify based on the content of the data, not the shape of the data. In order to ensure that the noise has a similar shape as the data, the same lengths are used when splitting the noise data. The following definition aims to clarify;

Let A be a data series containing n events, and each segment a_i contain the event

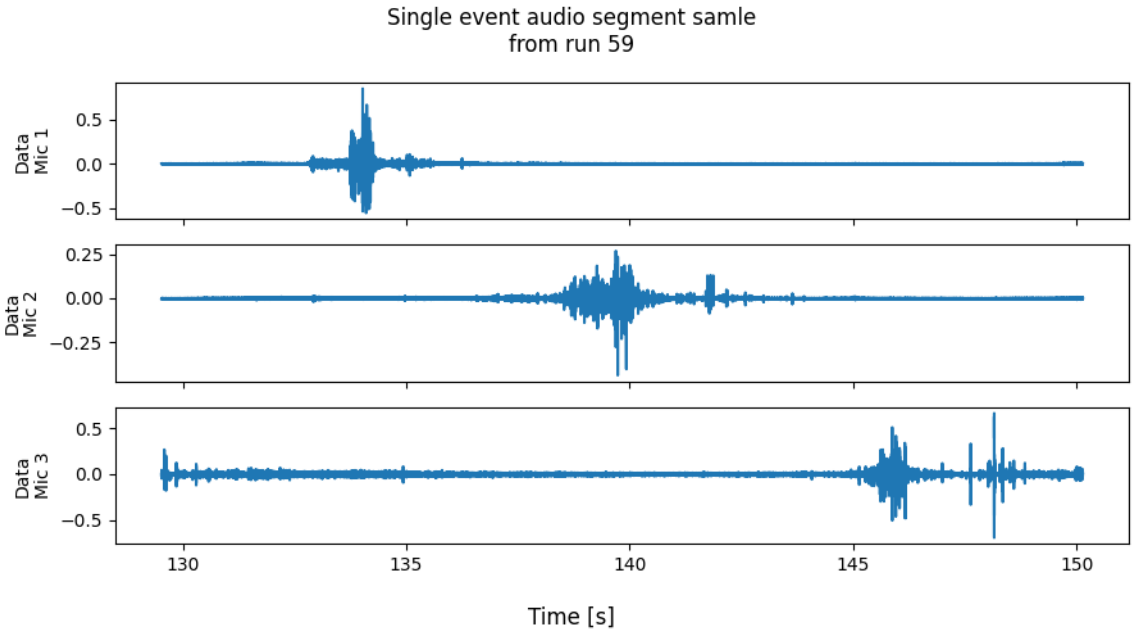


Figure 5.2: Sample of a data segment which contains one event. This event in particular corresponds to the eight peak in fig. 5.1.

i, and i only. Let t_i be the length of a_i. Let N be a data series containing noise. N is then split into segments n_i, such that n_i has length t_i.

This ensures that the shapes of the data is similar, and should not affect the training of the model.

The complete collection of all these segments constitute the constructed dataset. Stored data has no value without proper labels, at least when performing supervised learning.

5.1.2 Video and Automated Labeling

A necessity for supervised learning is the availability of proper data. That is; proper quality, and proper labeling. Ideally this labeling should be performed by humans to ensure that it is as precise as possible. This is, however, very costly when it comes to working hours. Teams can be hired for labeling, but if datasets are very large an automatic solution might be more appropriate. Unfortunately, this method is not without its consequences. If there was a simple way to classify data, there would be no need to train a neural network to perform the job, a deterministic algorithm could take its place. In addition, edge cases which can prove difficult to label even for humans will most certainly prove difficult for a labeling algorithm. We need to be

aware of these pitfalls when implementing an automatic labeler. Thankfully, for this dataset we have the video recordings which accompany the audio. We are then able to analyse the video in order to estimate the velocity and length for a given slug.

The automatic labeling process consists of multiple steps of data treatment.

1. Time slice for pixel column

Start by selecting a column of pixels in the video. For each frame, store the values in the column such that we obtain a new image with width equal to the number of frames. This process is shown through figures 5.3, 5.4 and 5.5.

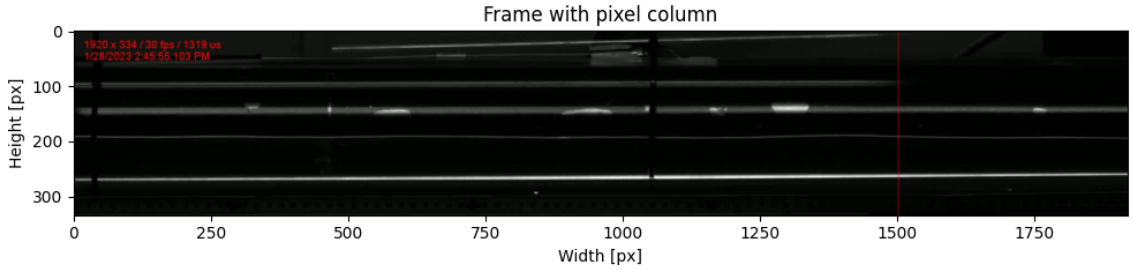


Figure 5.3: First frame of the mp4 file for run number 59, with the marked pixel column.

For this example, the column of pixel 1500 in fig. 5.3 was chosen. The time evolution for the chosen column is shown in fig. 5.4.

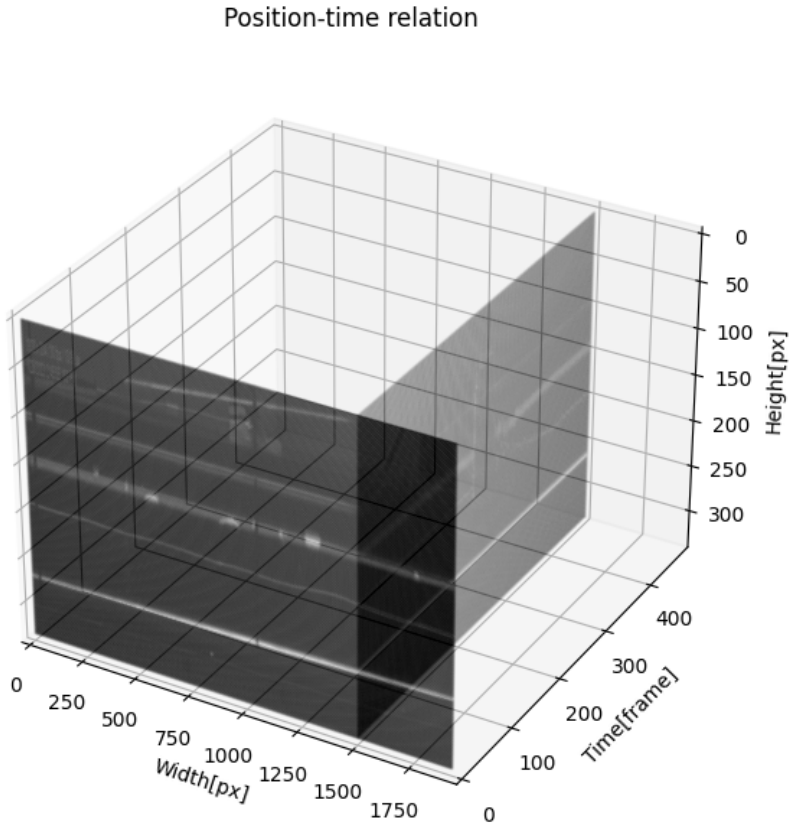


Figure 5.4: Illustration of how the values for a given column can be shown as an image with time along the x-axis and the height in pixel along the y-axis. In this example the column of pixel 1500 was chosen.

This operation generates the time slice for a single column, which can be seen in fig 5.5.

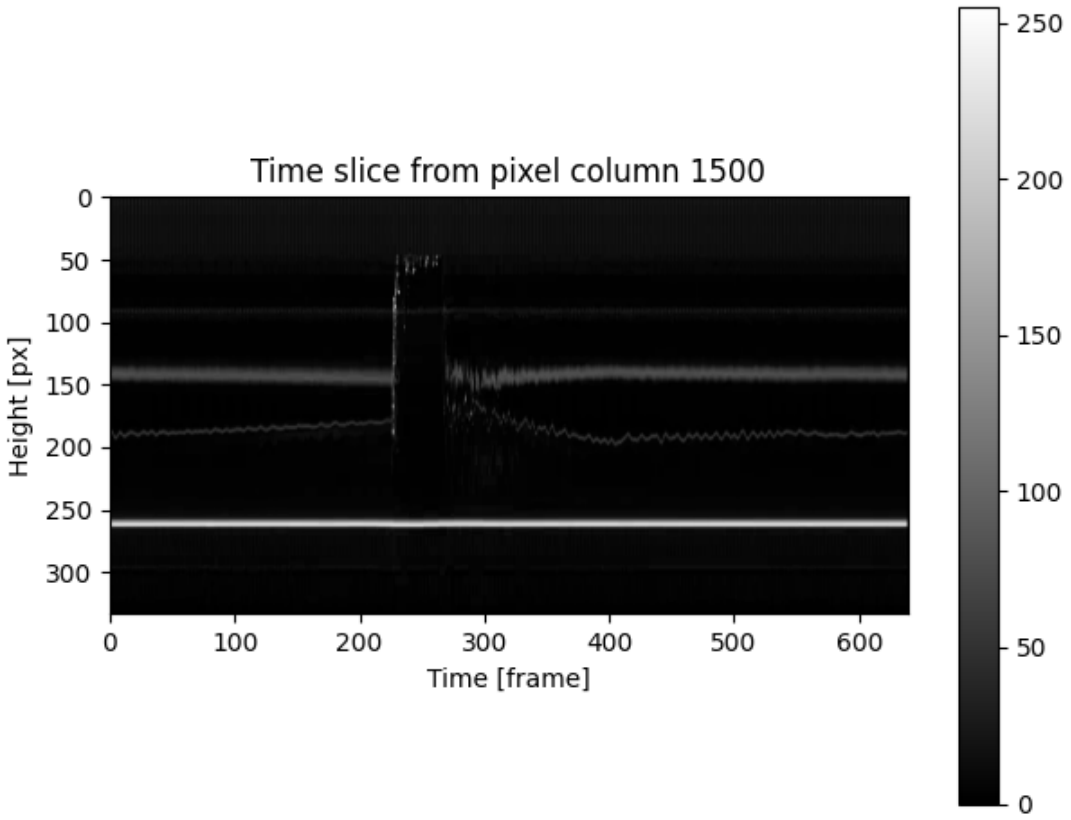


Figure 5.5: Time evolution at pixel column 1500 for the first event in run number 59.

We now have an understanding of how a single slice of the pipe evolves through time. In order to reduce the constant glare which is present in the image, each row has its mean subtracted from itself. The result is shown in fig. 5.6, and the image has a visible air-water boundary. The width of the slug corresponds to the time in which it occupied the measured slice as measured in frames, as the x-axis in the image is the frame count of the segment.

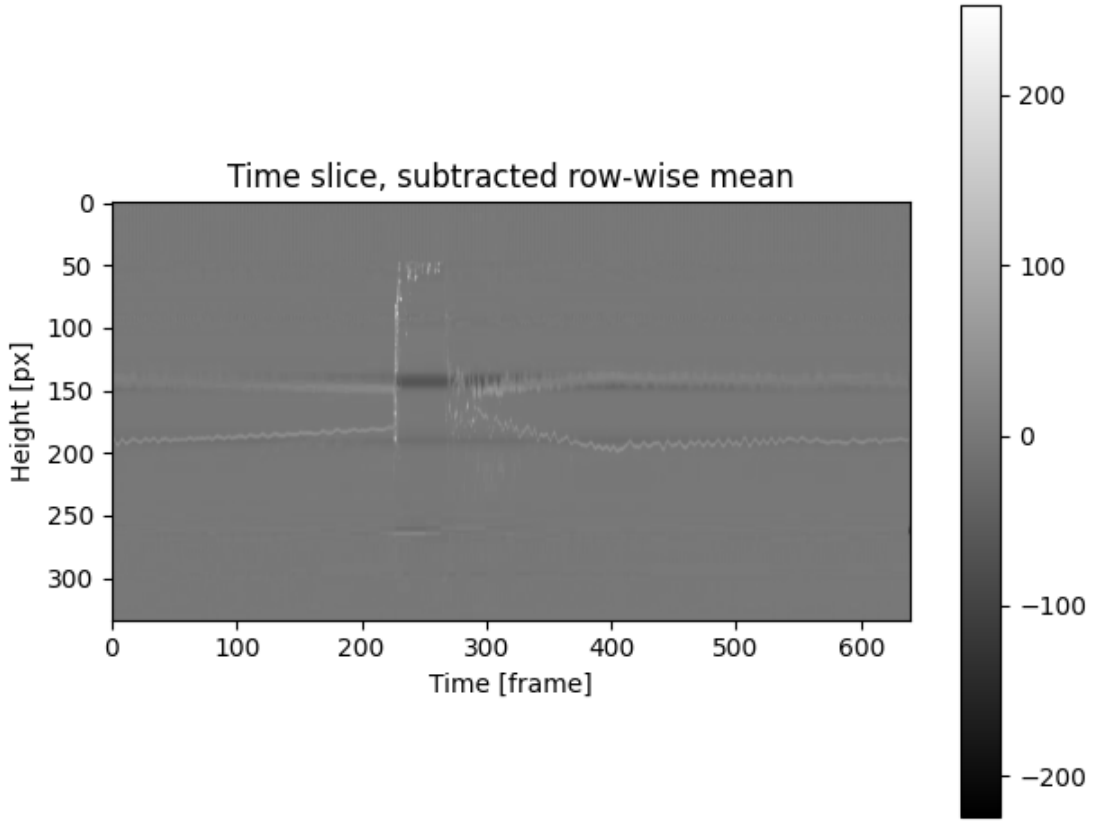


Figure 5.6: Time evolution at pixel column 1500 for the first event in run number 59, with the row-wise mean subtracted from each row.

Since all the information we need from the image is dictated by the slug edges, the image can be transformed into a one dimensional array by taking the mean for all the values column wise, which makes the data easier to analyse. This operation preserves the border locations as the columns containing the edges are brighter than the rest of the image, and the result will be referred to as intensity. In addition, the intensity is smoothed using a 1D convolution with the kernel equal to an array of length 11, with each element equal to 1/11, such that the convolved intensity is equal to the local average. An example of intensity is shown in fig. 5.7

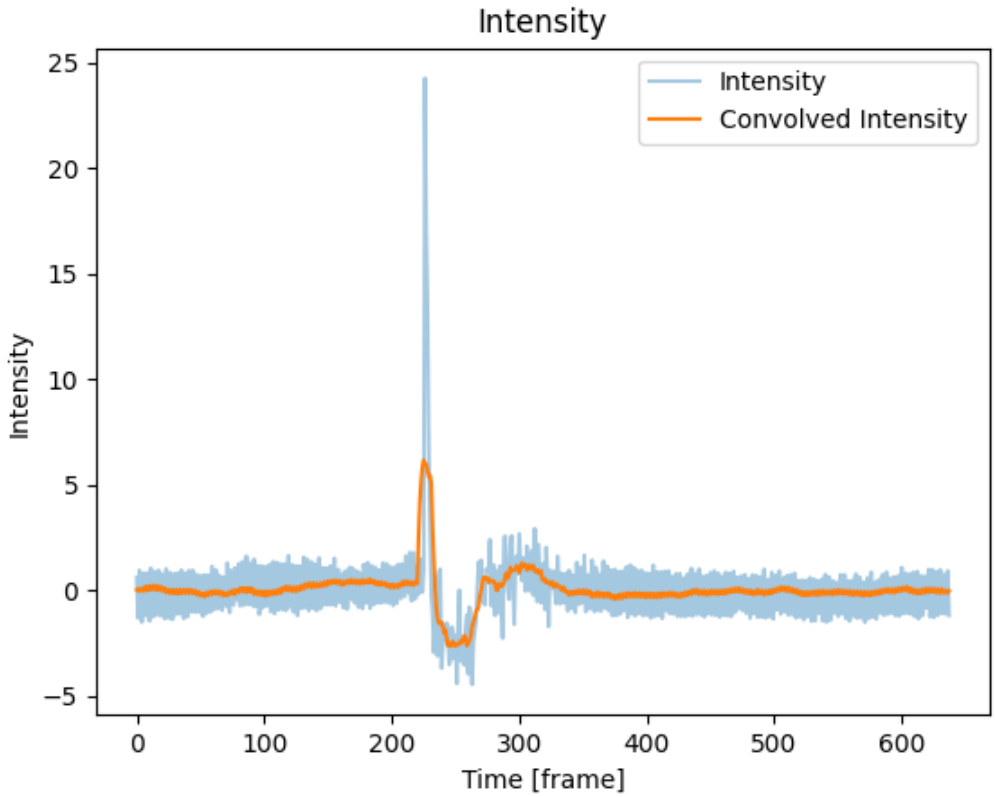


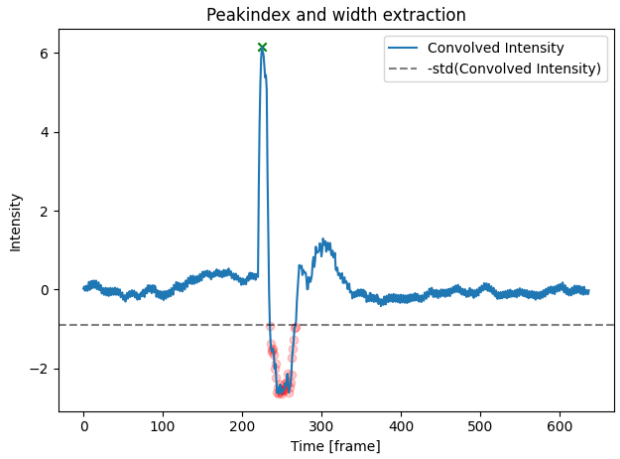
Figure 5.7: Intensity of the slug, used for further information extraction. Note that the intensity is dimensionless.

2. Peak extraction and Width calculation

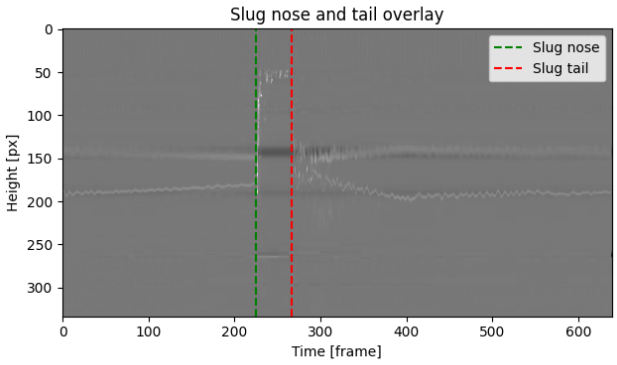
We now need to extract the peak and calculate the time width of the slug segments.

In the plug slug case, the intensity is generally lower between the nose and tail of the slug. This is exploited in order to find the first peak, which corresponds to the nose. By finding the first convolved intensity value less than negative one standard deviation of the convolved intensity, the peak can then be found by the taking largest value before this point. The width of the slug, nose to tail, is then from the peak until the values are no longer less than -std(convolved intensity). The process is illustrated in fig. 5.8.

For the aerated slugs, there is only one distinct peak in the intensity plot, as the slug diffuses a lot of the light from below, due to the dispersed air bubbles. In this case, the peak is found by taking the largest value of the intensity values. The width of the slug is estimated as the distance between the last point with value less



(a) Peak and troth for the convolved intensity, along with the troth cutoff at negative one standard deviation.



(b) Image from fig. 5.6, with an overlay of the estimated nose and tail of slug indices.

Figure 5.8: Slug nose and tail estimation method for plug slugs.

than std(intensity) before the peak itself, and the first point with value less than std(intensity) after the peak. The process is illustrated in fig. 5.9.

3. Velocity and Length estimation

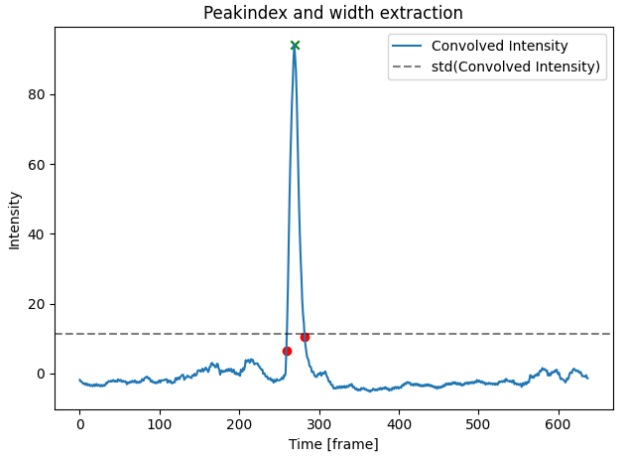
In order to find the velocity of the slug nose, we use the measurements taken for a range of pixel columns. By plotting the frame number for the peak as a function of pixel columns, we are able to regress a line through the data points. Due to inconsistencies in the peak and width extraction process, the RANSAC regression (RANdom SAmple Consensus) [ref] was used in order to discard any outliers. Note that this plot has position on the x-axis and time given in frames (1/30 s) on the y-axis. The slope of the computed regression line is then the estimate for the inverse of the velocity.

For the estimated time lengths it was found that the median value was a good estimation.

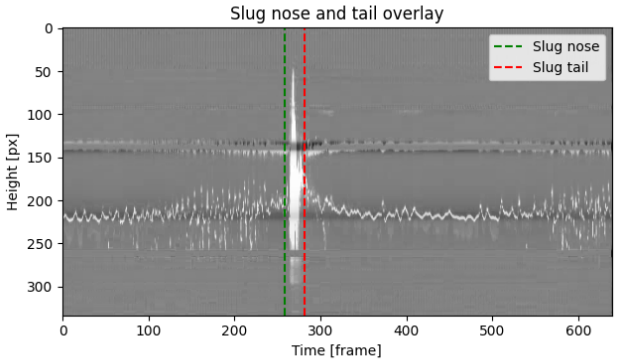
Examples of the process behind velocity and length estimation are shown in fig. 5.10.

4. Unit conversion and final calculation

Finally a unit conversion is performed. As mentioned, the prior step provides the inverse of the velocity as Δ frame / Δ pixel. Using that one pixel equals 0.000457



(a) Peak extraction and estimations of the slug width.



(b) Row-wise mean subtracted intensity, with overlay of the estimated nose and tail of slug indices.

Figure 5.9: Slug nose and tail estimation method for aerated slugs.

meters, the final velocity estimate is given as

$$\hat{v} = \frac{1}{\text{slope}} \cdot 0.000457 \frac{\text{m}}{\text{pixel}} \cdot 30 \frac{\text{frame}}{\text{s}} = \frac{0.000457 \cdot 30}{\text{slope}} \frac{\text{m}}{\text{s}}.$$

In order to get the final estimated length, the estimated length in time (frames), $\hat{t}$, is multiplied with the estimated velocity. That is

$$\hat{l} = \hat{t} \cdot 30 \frac{\text{frame}}{\text{s}} \cdot \hat{v}.$$

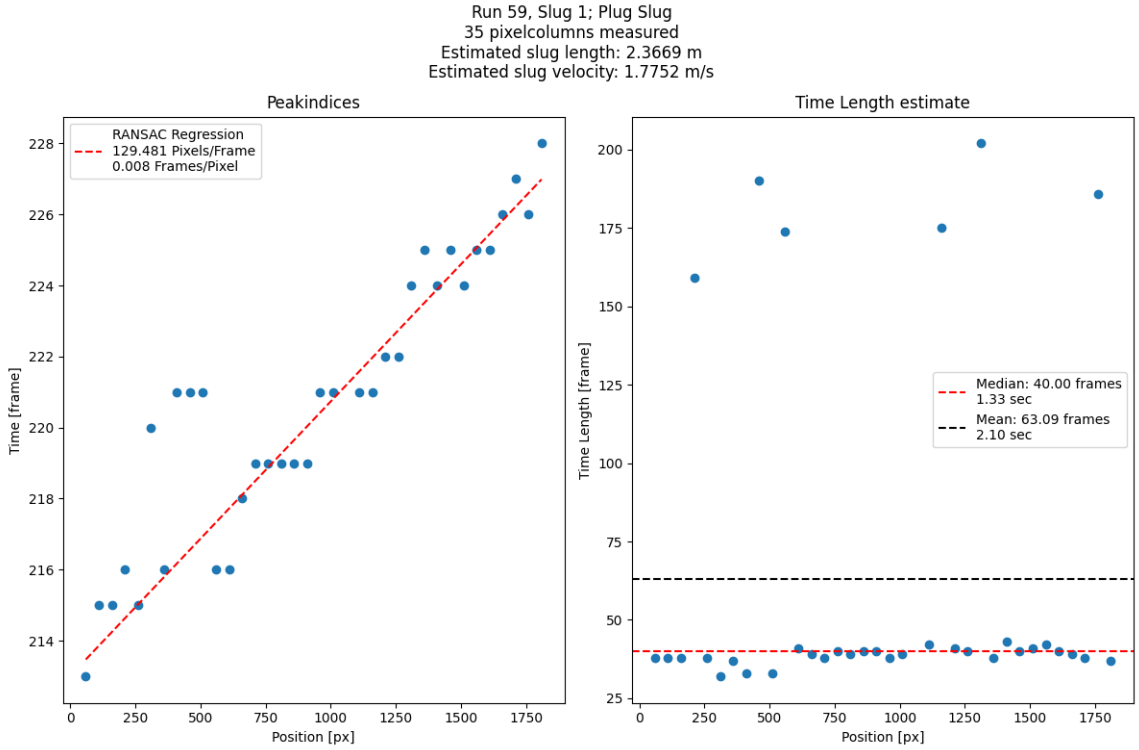
These values are written to a `.csv` file alongside additional information about the event which the estimations correlate to.

Class label estimation

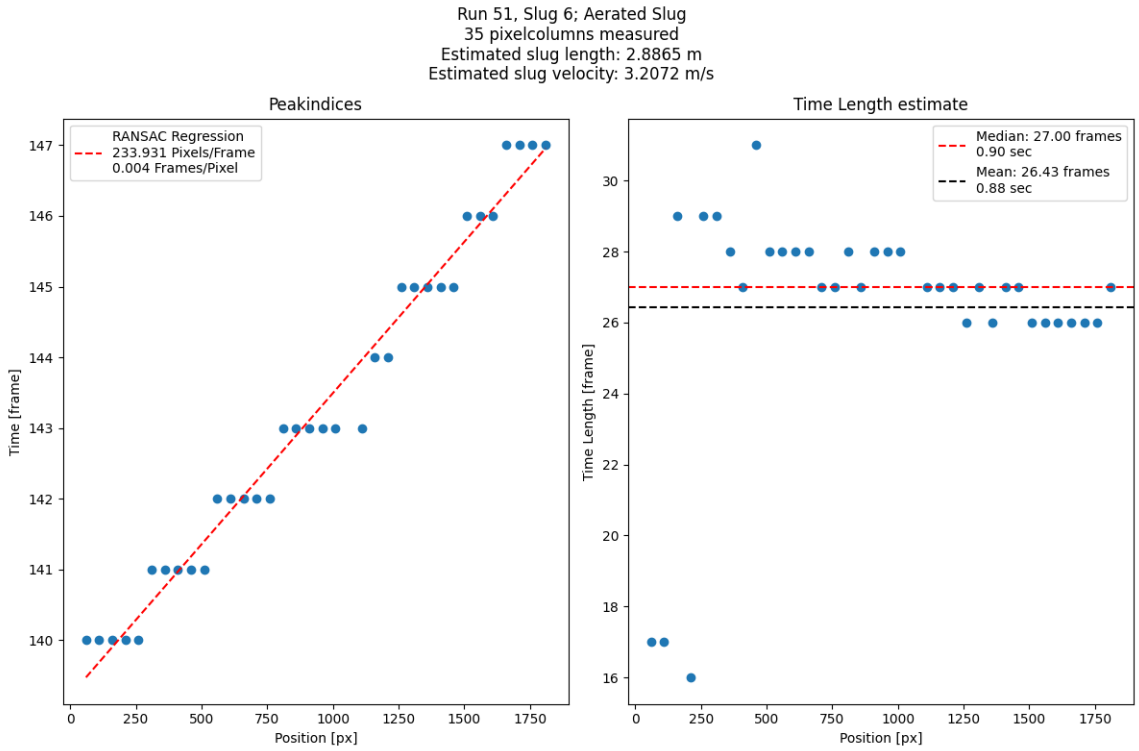
Ultimately, developing one single algorithm for sorting all possible events was not possible given the scope of this thesis work. A more custom solution was implemented, based on previous knowledge about the runs. Because every event in the runs 59 and 60 were slug plugs, these were automatically labeled as such.

In a similar fashion, seeing as the runs 54, 55 and 58 only contains noise, these segments were automatically labeled as well.

For runs 49, 51 and 53, however, both aerated slugs and breaking waves were present. The separation of these classes was again done by way of video analysis. By



(a) Plug slug example.



(b) Aerated slug example.

Figure 5.10: Examples of velocity and length estimations, both for the plug slug and the aerated slug case.

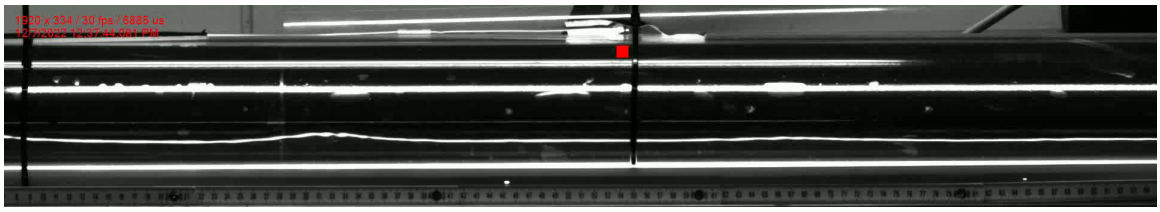


Figure 5.11: Region of which the intensity was averaged over for aerated slug/breaking wave classification.

definition, as described in chapter 2, for an event to be classified as a slug, it needs to occupy the entire cross-sectional area of the pipe. To check whether this occurred for a given event, a selected region close to the pipes ceiling had its average value computed for each frame. The region was chosen to be close to the 2nd accelerometer, with the intention being that a slug should fill the pipe whilst traveling past a microphone. The region is marked in red, as shown in fig. 5.11. If the maximum of these average values was above a certain threshold, the event was classified as an aerated slug. If not, the event was classified as a breaking wave. It was found that 220 was a satisfactory value for the threshold. The maximum value the averaged intensity can have is 255, as can be seen in fig. 5.5. In order for an even to be classified as aerated, the averaged intensity within the measured region must be close to white.

5.2 The Constructed Dataset

The constructed dataset is available in the repository danaars/Pipesound.

The final dataset consists of 995 text files, as well as a `.csv` file containing the different labels, as well as additional meta information. Each text file contains the audio data for exactly one event, unless the file contains noise, in which case there should be no significant events. The filenames are on the format "[event number]_r[run number].txt", and the counting start at 1. For instance, "8_r59.txt" contains the audio data for the eight event in run number 59.

Run number	Event			
	Count	**avg. length [s]**	**max length [s]**	**min length [s]**
49	115	16.74	25.70	9.06
51	180	10.08	17.41	6.95
53	138	13.11	29.13	6.63
54	108			
55	180			
58	138			
59	69	19.39	22.59	15.77
60	67	19.09	20.06	13.77

Table 5.2: Chosen statistics about the events. Note that the runs 54, 55 and 58 contained noise data, thus no event is present. The count column for these runs reflect the number of segments each run was split into.

The distribution for class labels is shown in fig. 5.12.

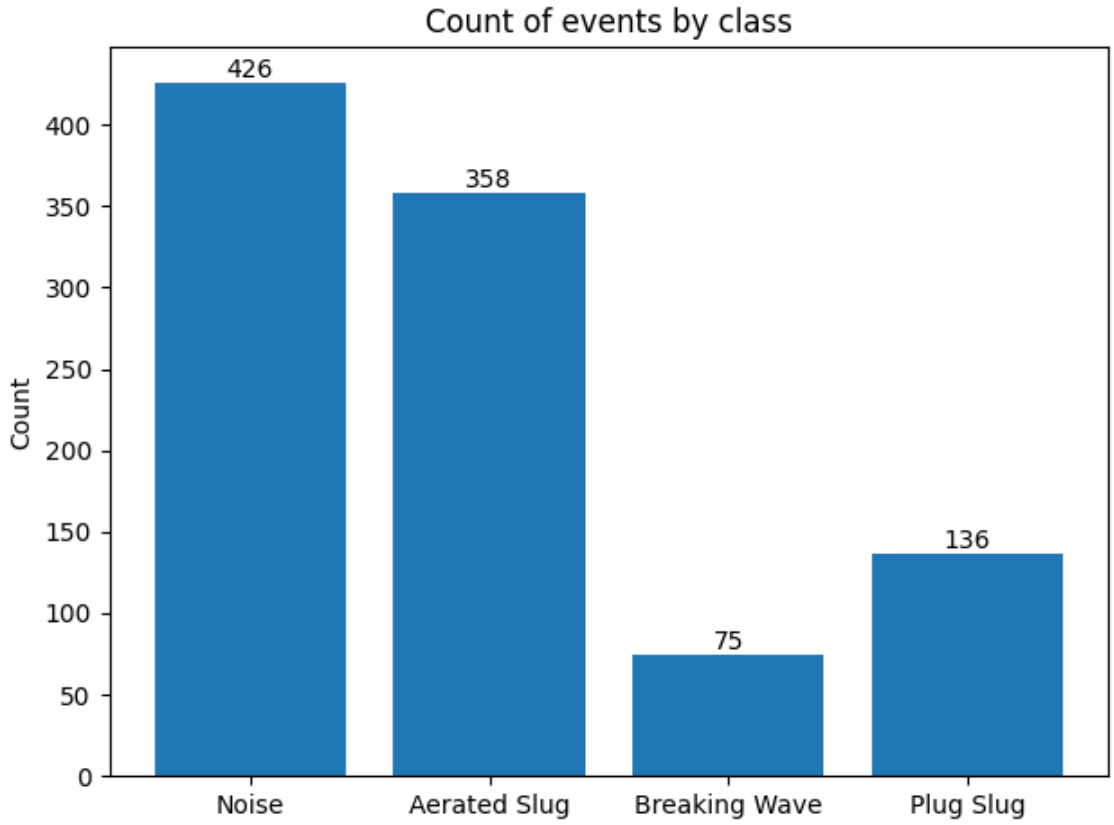


Figure 5.12: Count of the different classes in the final data set. Note that the number of noise events is equal to the number of aerated slug events and number of breaking wave events combined.

Chapter 6

Specific Application of the CNN

All machine learning models used in this thesis work is created with the PyTorch framework, which is one of the main frameworks used for artificial intelligence implementations in python.

As the test case for the CNN method, given in appendix A, is considered a success, a similar implementation is used. There are some differences, however, which must be accounted for. The aforementioned MNIST case study only concerns itself with classifying the audio data, whereas this specific application has to both classify and perform multiple regressions. As a consequence, the network will at some point branch into multiple sub-networks which each will provide one prediction.

6.1 Architecture

The convolutional neural network architecture utilized in this work consists of three convolution layers, followed by three distinct feed-forward neural networks, each with two hidden layers. The overarching architecture was kept constant across implementations, and is illustrated in fig. 6.1. In both the 1D and 2D case the number of nodes in each hidden layer was set to 64.

1D

The kernel sizes for the 1D convolution case are provided in table 6.1.

2D

The kernel sizes for the 2D convolution case are given in table 6.2.

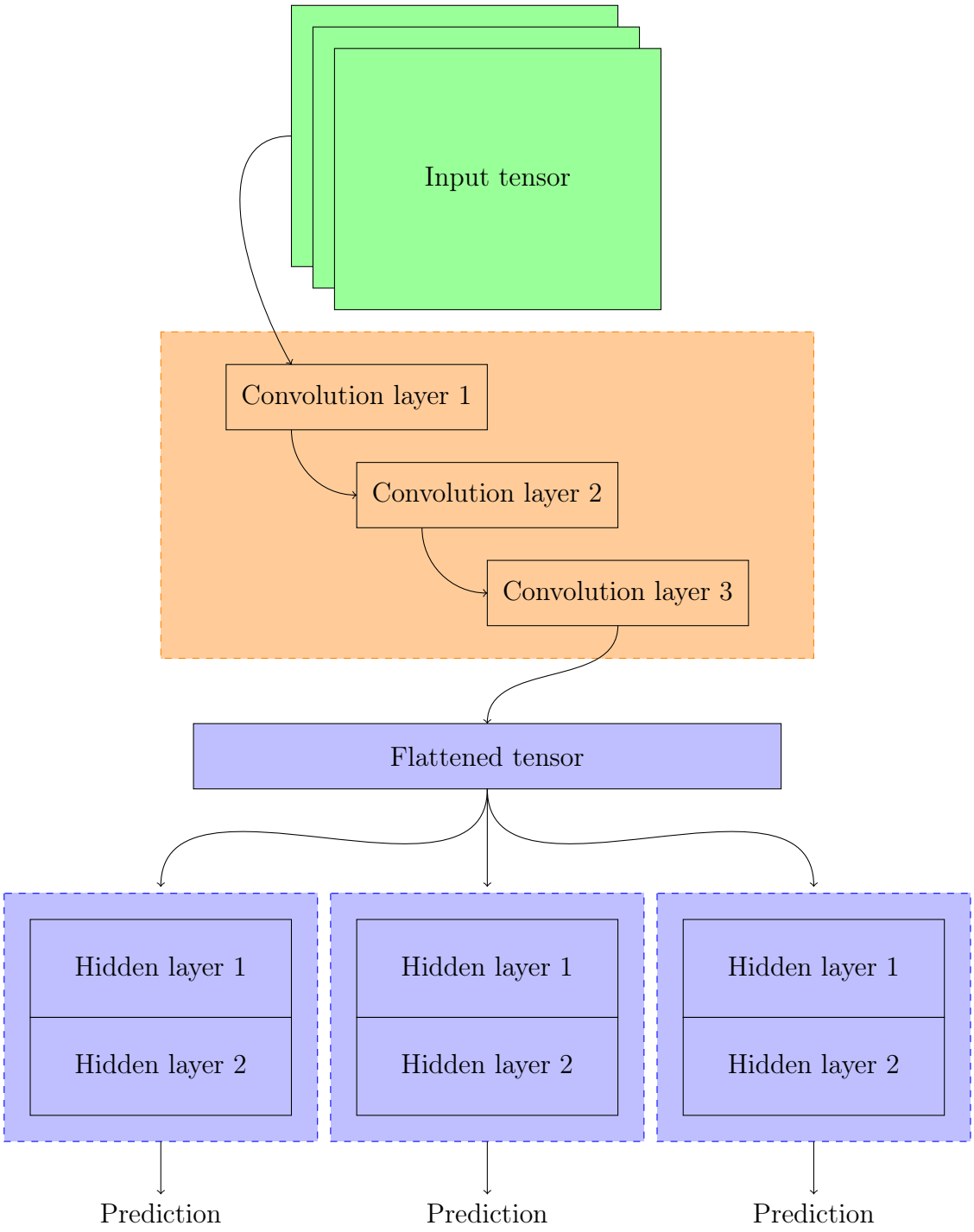


Figure 6.1: Sketch of the final convolutional neural network layout used in both the 1D and 2D cases. The three final feed-forward neural networks are independent from each other, such that the loss from one will not affect the others. This is indicated by the disconnected blue regions. In contrast, all convolution layers are connected, and are thus grouped in the same region.

Convolution layer	Kernel size	Channels out
1	16	8
2	8	16
3	5	32

Table 6.1: Kernel sizes and number of kernels (channels out) for each convolution layer.

Convolution layer	Kernel size	Channels out
1	16x16	8
2	8x8	16
3	5x5	32

Table 6.2: Kernel sizes and number of kernels (channels out) for each convolution layer.

6.1.1 Custom loss function

The loss function which was utilized for both the 1D and 2D case is a linear combination of the Cross Entropy Loss function, and the Mean Squared Error function as described in subsection 3.2.3. As the output of the model consists of three distinct predictions, they are collectively denoted by the vector $\hat{\mathbf{y}}$, and the corresponding label is denoted by $\mathbf{y}$. The loss function used for training can then be written as

$$L(\hat{\mathbf{y}}, \mathbf{y}) = \alpha L_{CE}(\hat{y}_{clf}, y_{clf}) + \beta \text{MSE}(\hat{y}_{vel}, y_{vel}) + \gamma \text{MSE}(\hat{y}_{len}, y_{len}), \tag{6.1}$$

where $\hat{y}_{clf}$ is the classification prediction, $\hat{y}_{vel}$ the velocity prediction and $\hat{y}_{len}$ the slug length prediction. Note that the variables $\alpha, \beta, \gamma \in \mathbb{R}$ can be tuned in order to emphasise one of the loss functions metric more strongly than the others. In this application, the values were simply set to $\alpha = \beta = \gamma = 1$.

The proposed loss functions contain the properties of the Cross Entropy loss function and the Mean Squared Error function, as this is simply a linear combination. Moreover, the location in parameter space which minimizes the proposed loss function is not necessarily the same location that would minimize the functions individually.

6.2 Data Pre-Processing

When data is treated in some manner which does not generate new data points, it is generally not considered augmentation, but rather pre-processing of the data. The

intention of which is to make the convergence of the training faster, or in some cases, even feasible.

6.2.1 Scaling

Scaling is an important factor when it comes to training and final performance of the network [ref]. In this work two scaling methods are compared, normalization and standardization. For an input $\mathbf{x}$ the normalized output χ is given by $\chi = \mathbf{x}/\max(\mathbf{x})$ such that $\chi \in [0, 1]$. The standardized output is given by $\chi = \frac{\mathbf{x}-\text{mean}(\mathbf{x})}{\text{std}(\mathbf{x})}$. This does not put an upper or lower bound in the data, but rather scales the distribution and ensures that $\text{std}(\chi) = 1$. For symmetric probability distributions the expected value of χ would be zero, but since the underlying distribution of the data samples is unknown, the expected value is not known *a priori*.

6.2.2 Transformations

For the 1D case, a transformation is strictly not necessary, as the model accepts the audio data without issue. If we wish to transform the data, a natural choice is the Fourier Transform, calculated by the ever-relevant Fast Fourier Transform [47]. The Fast Fourier Transform algorithm is arguably the most important tool within modern signal analysis, and is the foundation of the transformations used in the 2D case.

In order to even perform a 2D convolution on the audio data, a transformation is necessary, as the audio data itself is simply multiple 1D arrays. For the initial testing a spectrogram transformation (wavelet transformation) [1] was used, as it seemed promising from the test case presented in Appendix A. The specific implementation is done by the `spectrogram` function found within scipy's `scipy.signal` module. There are, however, other candidate functions. Most noteably the Short-Time Fourier Transform, which is a localized Fourier Transform.

6.3 Implementations

The following combinations of data pre-processing and data augmentations were tested.

6.3.1 1D

- **No Scale**; Incoming data is fed directly into the network.

[1] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.spectrogram.html

- **Channel wise normalization**; Incoming data has each channel normalized by their channel-wise max.
- **Full normalization**; Incoming data has channels normalized by the max across all channels.
- **Channel standardization**; Incoming data is standardized with respect to the mean and standard deviation for each channel.
- **Full standardization**; Incoming data is standardized with respect to the mean and standard deviation across all channels.
- **Discrete Fourier Transform**; Data is transformed with the Fast Fourier Transform before being fed into the network.
- **Discrete Fourier Transformation and normalization**; Data is transformed with the Fast Fourier Transform and then normalized.
- **Discrete Fourier Transformation and standardization**; Data is transformed with the Fast Fourier Transform and then standardized.

6.3.2 2D

- **No scale**; Incoming data is transformed directly into a spectrogram.
- **Channel wise normalization before spectrogram transformation**; Incoming data is normalized channel wise and then transformed into a spectrogram.
- **Normalization before spectrogram transformation**; Incoming data is normalized, then transformed into a spectrogram.
- **Spectrogram transformation then channel-wise normalization**; Data is transformed into a spectrogram, then normalized channel-wise.
- **Spectrogram transformation then normalization**; Data is transformed into a spectrogram, then normalized.
- **Spectrogram transformation then channel-wise standardization**; Data is transformed into a spectrogram, then normalized.
- **Spectrogram transformation then standardization**; Data is transformed into a spectrogram, then normalized.

- **Short-Time Fourier Transformation then channel-wise normalization**; Data is transformed then normalized with respect to the channel-wise max.
- **Short-Time Fourier Transfromation then normalization**; Data is transformed then normalized.

6.4 Metrics

The collection of performance metrics was performed after each epoch, through the implementation of the python package `torchmetrics`.

For the classification, the accuracy is used for model performance evaluation. Due to the distinct labels— The accuracy is simply defined as the number of correct classifications/predictions divided by the number of total classification/predicitons.

When it comes to regression, it does not make sense to discuss whether a prediction was correct or not, as the output values are continuous. A better metric is to specify how close the prediction was to the label. A common choice for regression metric is the R2 score. Whilst for instance the mean squared error can be used as a metric, the value it provides is rather arbitrary, and is usually used only for comparing the model against itself. The R2 score on the other hand compares the error made by the model with the error from simply guessing the sample mean, and is defined as

$$R2 = 1 - \frac{SS_{res}}{SS_{tot}}, \tag{6.2}$$

where the $SS_{res} = \sum_i (y_i - \hat{y}_i)^2$ is the sum of squared residuals, and $SS_{tot} = \sum_i (y_i - \bar{y})^2$ is the total sum of squares. This comparison between prediction and mean makes the R2 score more suitable for comparisons between different models.

Note that for both the accuracy and R2 score the highest possible score is 1. This corresponds to an accuracy of 100%, and for regression it would mean that $y_i - \hat{y}_i = 0 \; \forall \; i$, i.e. a perfect prediction every time.

Part IV

Results & Discussions

Chapter 7

Results & Discussion

The proposed model has been trained with a constant architecture, as explained in section 6.1. The training itself took place on the high performance computing machine learning nodes at the institute of informatics at the university of Oslo [1]. Training was either performed on a RTX 2080 Ti or a RTX 3090 graphics processing unit.

For studying the performance of the proposed models the number of epoch was set to 50. The ADAM algorithm for stochastic gradient descent was used, and the initial learning rate set to 0.001, which is the default value for PyTorch. The batch size used throughout the testing was set to 1, as anything higher could run into memory problems during training. A study of favorable hyperparameters was not conducted.

Note that tables in this section show the best performance metrics achieved, during training. The best values did not necessarily occur for the same epoch, that is, for the same model.

7.1 Multi-channel

7.1.1 2D

Scaling

Initially the general effects of scaling was tested with the spectrogram transformation. From the results presented table 7.1, multiple things are evident. Firstly, scaling has a significant effect on the performance of the models.

Secondly, and most crucially, scaling has the largest effect when performed after the transformation. Due to the black box nature of the convolutional neural networks used, an exact explanation as to why cannot be given, but one might speculate that

[1] Machine learning infrastructure (ML Nodes), University Centre for Information Technology, University Of Oslo, Norway.

this is related to the potentially large values of the data. Scaling the input values does not change the underlying frequencies, and thus the numerical values in the tensor containing spectrogram data may still be large. Activation functions tend to be the most interesting for small values, preferably in the region about 0, and if the values are kept low throughout the network, this feature of the activation functions can be exploited to its greatest capacity.

Finally we observe that the difference in normalizing channel wise and normalizing with regard to the full tensor is rather small. An exception to this however is the length predictions, which seem to perform better when the full normalization is performed.

Scaling	**Best**					
	Accuracy		**Velocity R2**		**Length R2**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
No scaling	0.421	0.457	-1.919e-5	-2.503e-6	-1.466e-5	3.040e-6
Channel-wise normalization*	0.413	0.487	-7.033e-6	-7.057e-5	-5.519e-5	-5.484e-6
Full normalization*	0.437	0.392	-9.667e-5	-7.093e-5	-0.001	-8.130e-5
Channel-wise normalization	**1.0**	0.939	**0.988**	**0.667**	**0.975**	0.252
Full normalization	**1.0**	**0.959**	0.986	0.635	0.969	**0.426**

Table 7.1: Performance metrics for the proposed 2D model with different data scaling, on multiple channels. The * indicates that the scaling takes place before a transformation, that is, on the raw audio data itself. The boldface highlights the best metric for the given column.

When examining scaling with standardization, the results from normalization is taken into account. Hence, the standardization is only done after the wavelet transformation. The results are presented in table 7.2.

Scaling	**Best**					
	Accuracy		**Velocity R2**		**Length R2**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Channel-wise standardization	**1.0**	0.929	0.981	0.481	0.956	0.362
Full standardization	**1.0**	**0.935**	**0.983**	**0.639**	**0.965**	**0.391**

Table 7.2: Performance metrics with data subjected to channel-wise and full standardization after the spectrogram transformation. The boldface highlights the best metric for the given column.

From table 7.2 it is apparent that the standardization of the data performs somewhat worse than normalizing. Comparing the full normalization with the full standardization we see that normalization performs better on every metric. Still, the standardization produces generally good result, again testifying to the effectiveness of appropriate scaling.

The accuracy of the classification for the model subjected to the full normalization is presented in fig. 7.1.

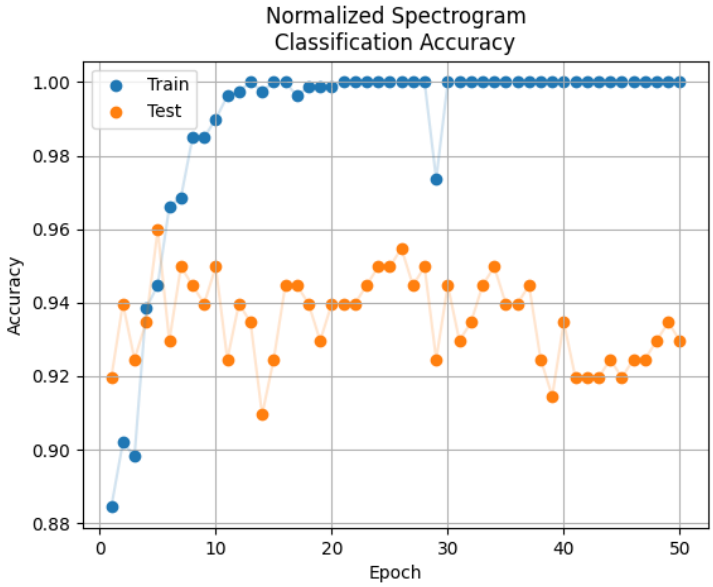
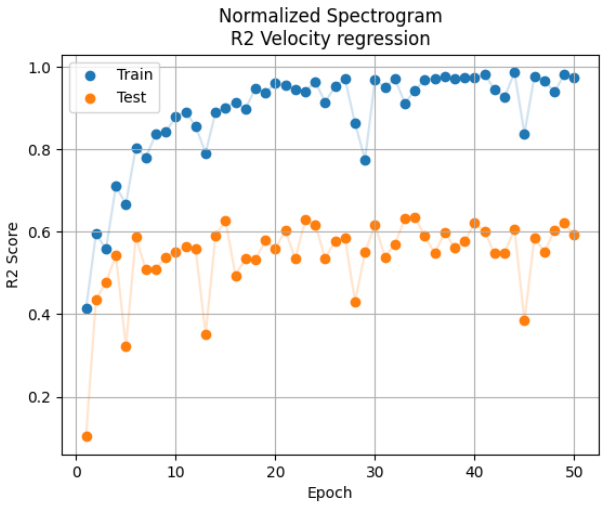


Figure 7.1: The accuracy score for the proposed model when the data has undergone a spectrogram transformation followed by normalization.

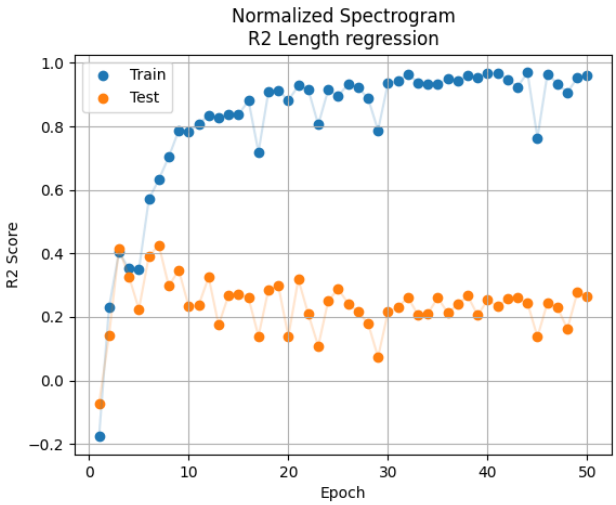
The R2 scores for the velocity and length regressions are shown in fig. 7.2.

From fig. 7.1 we see that the highest accuracy occurred early in the training, at epoch 5. As the model keeps training the accuracy for the previously unseen test data has a slight downward trend. This could indicate that the model is somewhat overfitted to the training data.

When it comes to fig. 7.2 we see that the length regression is generally better early on, as was also the case for the accuracy score. Whilst this could further indicate overfitting, we also see that the model is stable after epoch 30, if not slightly improving. For the velocity regression the performance of the model seems to improve with increased number of epochs.



(a) R2 score for the velocity regression.



(b) R2 score for the length regression.

Figure 7.2: The R2 scores for the proposed model when the data has undergone a spectrogram transformation followed by normalization.

Transformations

We now aim to study whether using a different transformation alters the results. Note that since a transformation must be used in order to obtain 2D data, the spectrogram transformation is already performed and the results can be found in table 7.1.

Based on the insights gathered from the scaling, only normalized and channel-wise normalized Short Time Fourier Transforms were tested. The results are presented in table 7.3.

Scaling	**Best**					
	Accuracy		**Velocity R2**		**Length R2**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Channel-wise normalization	**1.0**	0.945	0.989	0.576	0.979	0.356
Full normalization	**1.0**	**0.950**	**0.990**	**0.661**	**0.983**	**0.428**

Table 7.3: Performance metrics with data subjected to channel-wise and full normalization after the Short Time Fourier Transformation. The boldface highlights the best metric for the given column.

We see from table 7.3 that, again, full normalization performs better than the channel-wise implementation. By comparing table 7.1 and table 7.3 we see that

the models perform extremely similarly. The short time fourier transform is however marginally better. This is expected as the Short Time Fourier Transform does not only have a higher temporal resolution, but also the real and imaginary values provided.

The accuracy of the classification for the model with a normalized Short Time Fourier Transform is presented in fig. 7.3. Moreover, the R2 scores for the velocity and length regressions are shown in fig. 7.4.

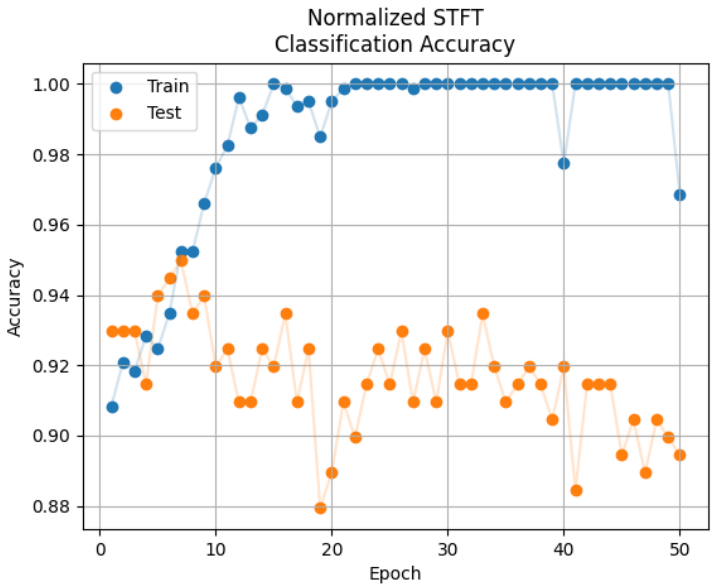


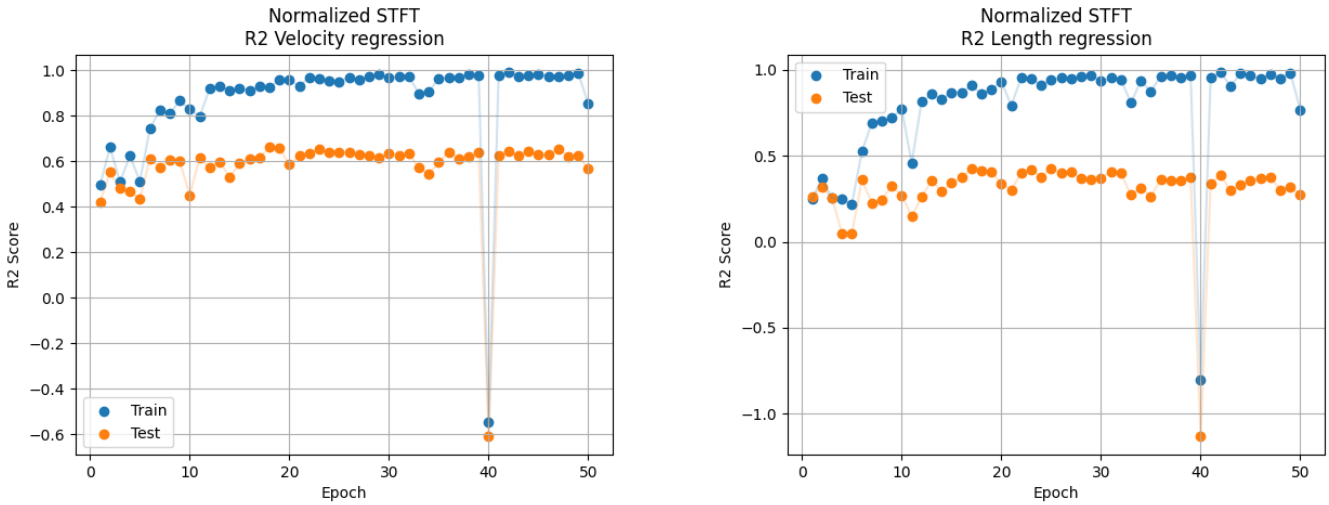
Figure 7.3: The accuracy score for the proposed model when the data has undergone a spectrogram transformation followed by normalization.

The same general patterns emerge in fig. 7.4 as in fig. 7.2, with the velocity regression R2 score for the test data stabilizing at around 0.6, whilst the length regression R2 score for the test data has a slight downward trend and ends up at about 0.3 - 0.25. By comparing fig. 7.3 and fig. 7.1 we see a similar trend of variance within the test data.

7.1.2 1D

Scaling

In contrast to the 2D case, the 1D case can be completed without the use of transformations. Five different alternatives for scaling are tested, and the results are presented in table 7.4.



(a) R2 score for the velocity regression.

(b) R2 score for the length regression.

Figure 7.4: The R2 scores for the proposed model when the data has undergone a spectrogram transformation followed by normalization.

Scaling	Best					
	Accuracy		Velocity R2		Length R2	
	Train	Test	Train	Test	Train	Test
No scaling	**1.0**	**0.975**	**0.993**	0.635	0.985	0.299
Channel wise normalization	**1.0**	0.915	0.988	0.499	0.984	0.104
Full normalization	**1.0**	0.919	0.988	0.616	0.984	0.341
Channel-wise standardization	**1.0**	0.920	0.991	0.544	**0.988**	0.200
Full standardization	**1.0**	0.960	0.989	**0.702**	0.983	**0.346**

Table 7.4: Performance metrics for the proposed 1D model with different scaling. The boldface highlights the best metric for the given column.

It is evident from table 7.4 that these performances are more similar to each other than what is seen in table 7.1. A possible explanation for this is that these values are generally small, as can be seen from 5.1. Hence normalizing and standardizing has a diminished effect compared to after transformations.

Transformations

When it comes to transformations for the 1D multichannel data, the discrete Fourier transform is tested, through the implementation of the fast Fourier transform.

Scaling	Best					
	Accuracy		Velocity R2		Length R2	
	Train	Test	Train	Test	Train	Test
No scaling	**1.0**	0.884	**0.992**	0.518	**0.985**	0.249
Full normalization	**1.0**	**0.985**	0.991	**0.787**	0.976	**0.412**
Full standardization	**1.0**	0.940	0.987	0.641	**0.985**	0.243

Table 7.5: Performance metrics for the proposed 1D model with a Discrete Fourier Transform and different scaling. The boldface highlights the best metric for the given column.

For the 1D transformation and scaling, table 7.5 shows that the data which is normalized again provide the best test scores. This is similar to the 2D case. By comparing table 7.4 and table 7.5 it would seem like subjecting the data to discrete fourier transform before normalizing results in a small increase in performance.

The accuracy of the classification for the model with a normalized Fast Fourier Transform is presented in fig. 7.5. Moreover, the R2 scores for the velocity and length regressions are shown in fig. 7.6.

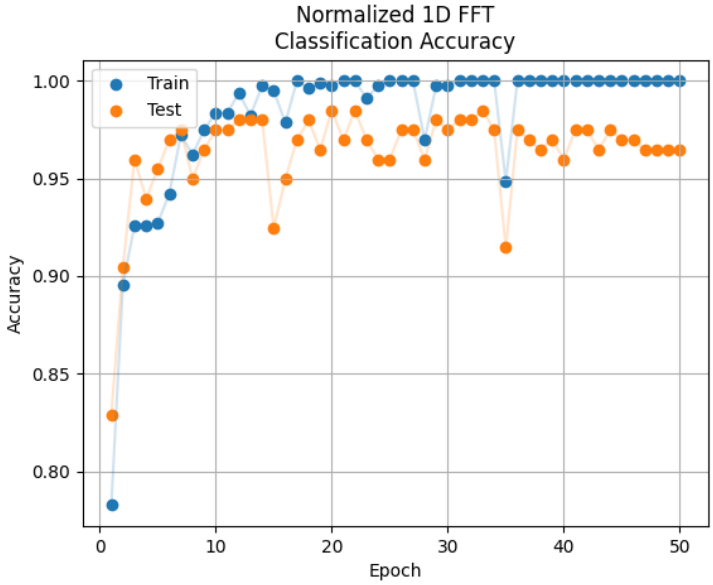
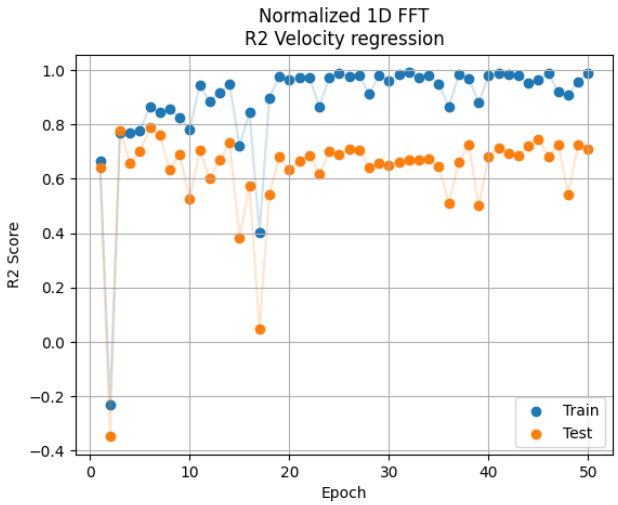
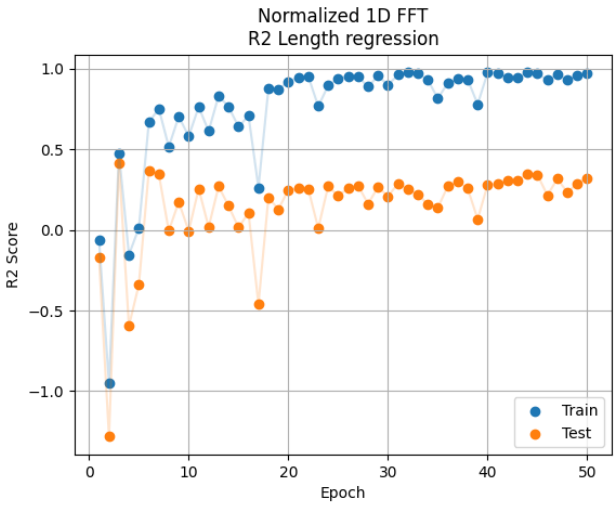


Figure 7.5: The accuracy score for the proposed model when the data has undergone a Fast Fourier Transformation followed by normalization.



(a) R2 score for the velocity regression.



(b) R2 score for the length regression.

Figure 7.6: The R2 scores for the proposed model when the data has undergone a Fast Fourier Transformation followed by normalization.

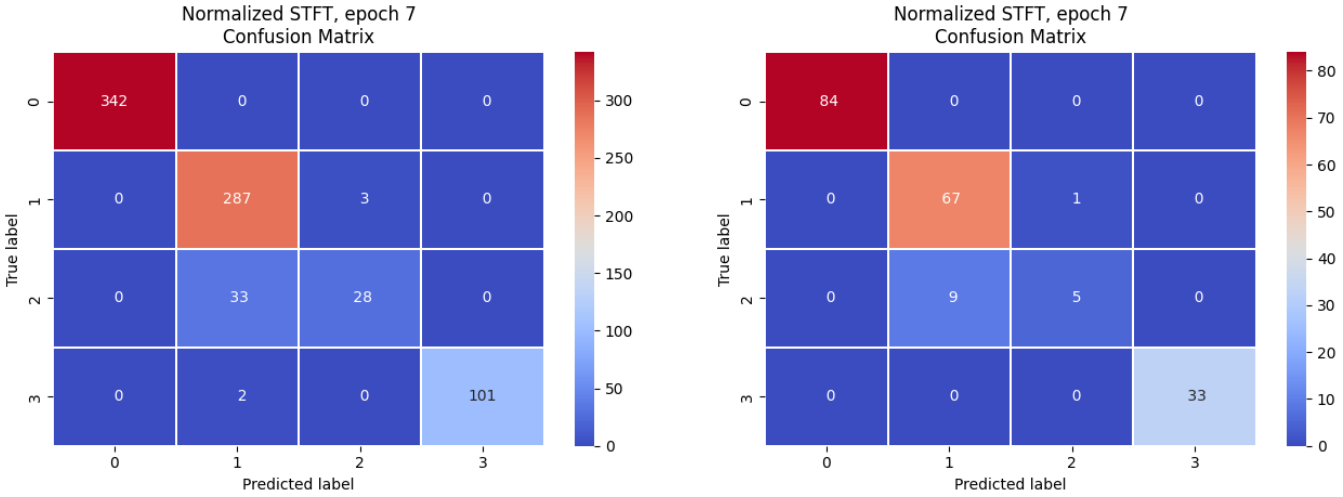
Judging from the best metrics alone, it would seem like the Fast Fourier Transform with normalization performs the best overall, whilst the Short Time Fourier Transformation with normalization performs the best in the 2D case.

For further analysis, the model was chosen at the epochs which performed the best with respect to accuracy of test data. That is, epoch 7 for the Short Time Fourier Transform, and epoch 20 for the Fast Fourier Transform. The confusion matrices for the normalized Short-Time Fourier Transform is presented in fig. 7.7. The confusion matrices for the normalized Fast Fourier Transform is presented in fig. 7.8.

7.2 Single Channel

The performance of the proposed model is tested when only trained on data from one of the three microphones. Data from microphone 2 was chosen, as it should correlate strongest with the label obtained from video analysis. The position of microphone 2 in relation to the camera frame is as shown in fig. 4.1.

The results of training on data from a single microphone are presented in table 7.6.



(a) Confusion matrix for training data.

(b) Confusion matrix for testing data.

Figure 7.7: The confusion matrices for the normalized Short-Time Fourier Transform at epoch 7.

Scaling	**Best**					
	Accuracy		**Velocity R2**		**Length R2**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
STFT normalized	**1.0**	0.935	0.983	0.723	0.969	0.40
FFT normalized	1.0	0.955	0.988	0.722	0.979	0.337

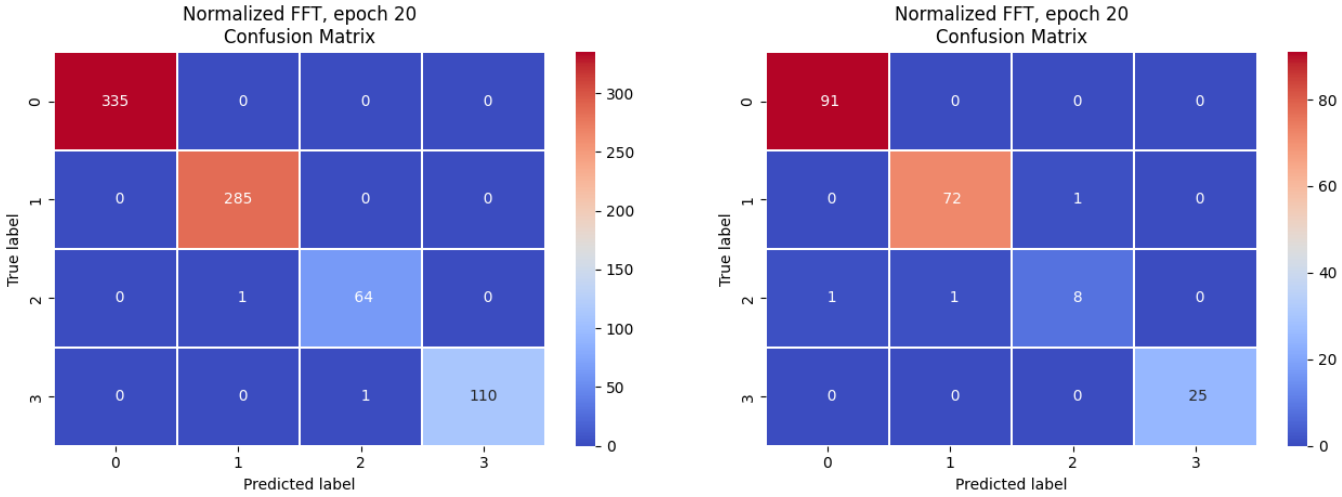
Table 7.6: Performance metrics for the proposed model when only subjected to data from one microphone. The boldface highlights the best metric for the given column.

From tables 7.6 and table 7.3 we see that the 2D implementation performs better on all metrics except from the R2 score for the length regression on the training data. Seeing as this is training data, this result is not significant. Still, only using one microphone results in a surprisingly good score.

The accuracy of the classification for the model with a normalized Fast Fourier Transform is presented in fig. 7.9. Moreover, the R2 scores for the velocity and length regressions are shown in fig. 7.10.

From tables 7.6 and table 7.5 we see that the 2D implementation again performs better on all metrics that matter.

The accuracy of the classification for the model with a normalized Fast Fourier Transform is presented in fig. 7.11. Moreover, the R2 scores for the velocity and



(a) Confusion matrix for training data.

(b) Confusion matrix for testing data.

Figure 7.8: The confusion matrices for the normalized Fast Fourier Transform at epoch 20.

length regressions are shown in fig. 7.12.

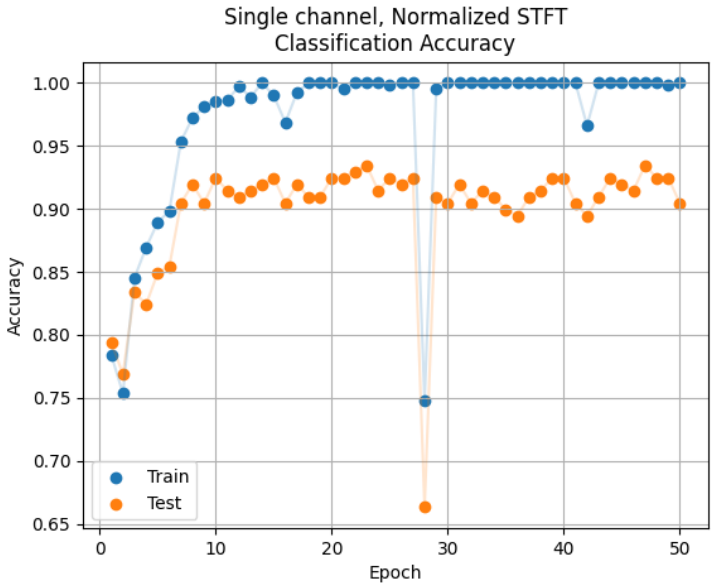
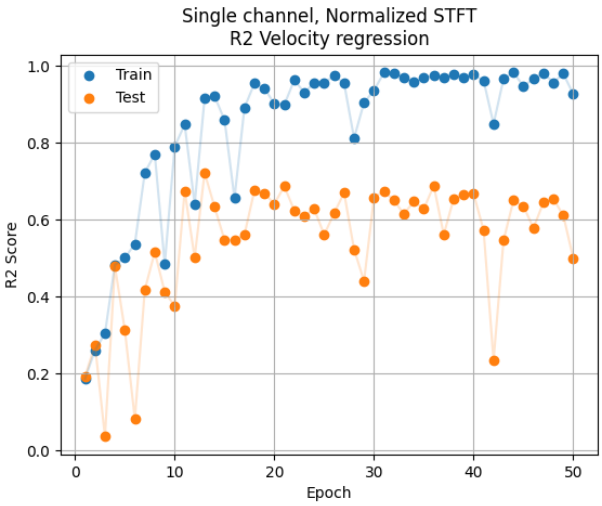
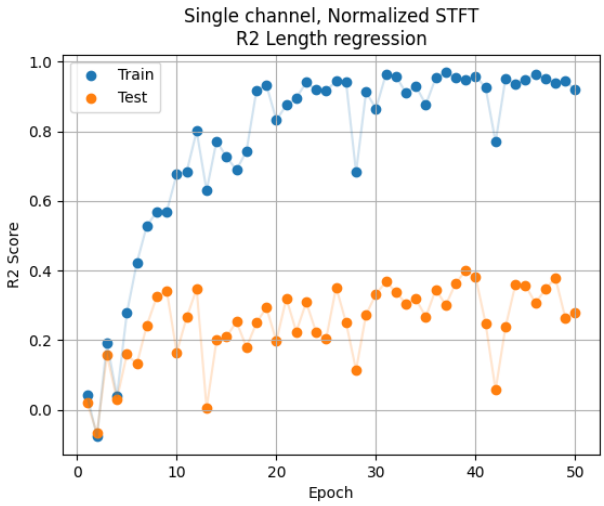


Figure 7.9: The accuracy score for the proposed model when the data from microsphone 2 has undergone a Short Time Fourier Transform followed by normalization.



(a) R2 score for the velocity regression.



(b) R2 score for the length regression.

Figure 7.10: The R2 scores for the proposed model when the data from microsphone 2 has undergone a Short Time Fourier Transform followed by normalization.

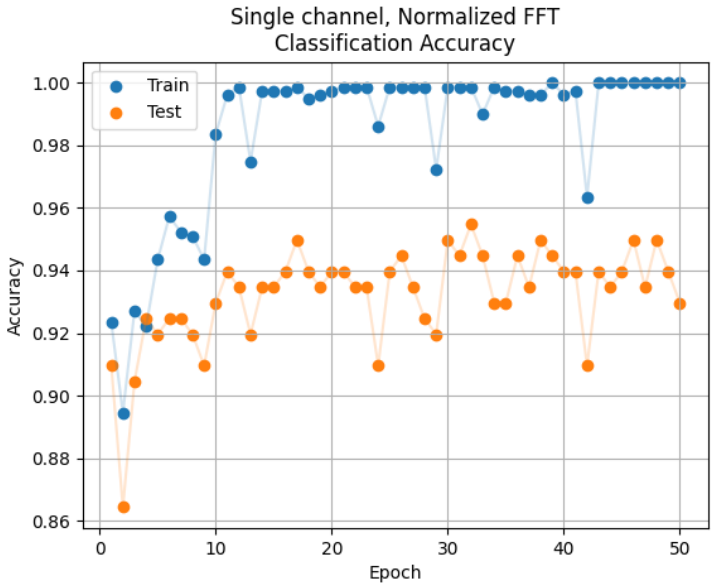
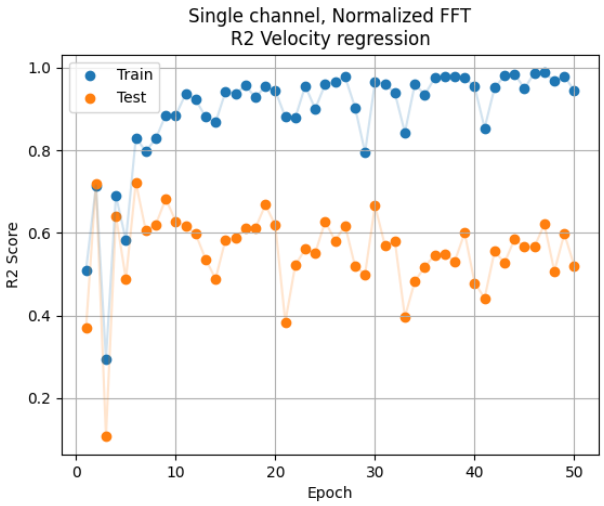
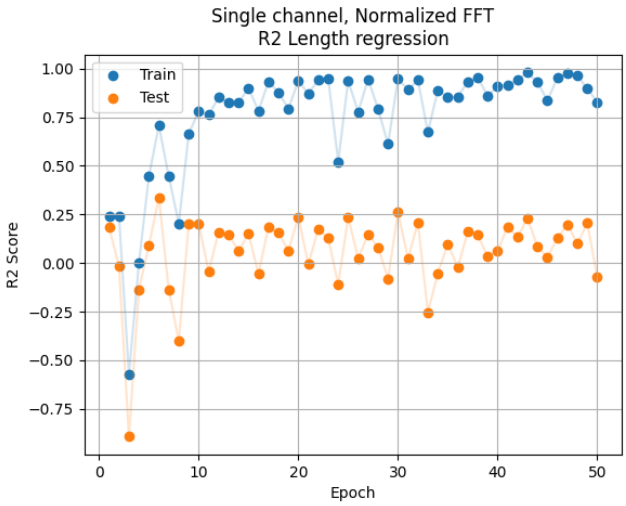


Figure 7.11: The accuracy score for the proposed model when the data from microsphone 2 has undergone a Short Time Fourier Transform followed by normalization.



(a) R2 score for the velocity regression.



(b) R2 score for the length regression.

Figure 7.12: The R2 scores for the proposed model when the data from microsphone 2 has undergone a Short Time Fourier Transform followed by normalization.

Chapter 8

Conclusion

This thesis has studied the novel use of convolutional neural networks applied to acoustic emissions. A dataset has been created by use of experimental data from accelerometers, and automatic labeling algorithms applied to video recordings of the same experiments.

The proposed model used to classify flow-regimes for multi-phase flow through pipes shows and shows high accuracy. The highest accuracy for previously unseen data was 98.5%. This is in agreement with other applications of machine learning applied to multi-phase flow-regime identification [5–15, 17, 19].

The proposed model seems to be somewhat prone to overfitting. This is not surprising as the model itself has not been optimized for best possible performance. Rather a comparison between the treatment of data was performed in order to gauge which transformations highlighted the necessary features for flow-regime identification. No definitive conclusion can be drawn on which transformation performs best in the 2D case, as both the spectrogram transformation and the Short Time Fourier Transform perform similarly. Taking small differences into account, it was found that Short Time Fourier Transform gave the highest accuracy in combination with Normalization when using a 2D convolutional neural network.

When it comes to regression it is evident that the multi-channel setup using three microphones is able to provide good predictions on the velocity of slugs. From the sample data shown in fig. 5.1, it is evident that the passing of slugs generate a distinct sound signature. This property could be exploited in order to make a deterministic algorithm for slug velocity.

For the 1D case it was found that the discrete Fourier transform containing real imaginary and absolute values gave the highest accuracy when combined with normalization. These metrics were even better then that of the 2D model, suggesting that the frequencies which compose the signal are more important than when the acoustic emission is measures, at least for these short time samples.

Based on the trends of overfitting it is evident that more data is needed. This would in addition lead to better metrics for both classification and regression. The most difficult cases were the breaking wave. This is expected as the breaking wave had the fewest data samples as can be seen from fig. 5.12. The combination of a small dataset and few breaking waves made the models which randomly were assigned the breaking wave samples in the test dataset appear worse, even though the accuracy on the other three categories generally were equal with an overall high accuracy.

8.1 Future Work

Multiple path forward has revealed themselves thuring this thesis work. The software used for this study could be the foundation for a device which performs realtime multi-phase flow-regime identification. For this to be the case, further development is necessary as well as implementation of micro controllers and/or sensor technology.

As has already been discussed, the deep learning model proposed could have benefited from more data. Further collection of data and application of new better labeling algorithms could be used in order to improve upon the results. In addition more labels could be introduced.

An alternative to collecting data is artificially generating more, by way of resampling. This could be implemented, but should be done with caution as the data quality would be compromised.

In order to bypass the data labeling part in its entirety, it would be possible to implement unsupervised deep learning methods which sort data without the need for labels. Keep in mind that this form for deep learning required extreme amounts of data.

Methods similar to those presented here could be used in the study of single phase flow, particularly in the detection of the transition between turbulent and laminar flow.

Finally, by implementing more precise data, similar methods could be implemented in order to predict the entire velocity field of the multi-phase flow.

Bibliography

[1] E. S. Kordyban and T. Ranov, 'Mechanism of slug formation in horizontal two-phase flow,' *Journal of Basic Engineering*, vol. 92, no. 4, pp. 857–864, 1st Dec. 1970, ISSN: 0021-9223. DOI: 10.1115/1.3425157. [Online]. Available: https://doi.org/10.1115/1.3425157 (visited on 05/05/2023).

[2] J.-M. Godhavn, M. P. Fard and P. H. Fuchs, 'New slug control strategies, tuning rules and experimental results,' *Journal of Process Control*, vol. 15, no. 5, pp. 547–557, 1st Aug. 2005, ISSN: 0959-1524. DOI: 10.1016/j.jprocont.2004.10.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959152404001131 (visited on 10/05/2023).

[3] E. Storkaas, S. Skogestad and V. Alstad, 'Stabilization of desired flow regimes in pipelines,' 1st Jan. 2001.

[4] A. Sausen, P. Sausen and M. de, 'The slug flow problem in oil industry and pi level control,' in *New Technologies in the Oil and Gas Industry*, J. S. Gomes, Ed., InTech, 31st Oct. 2012, ISBN: 978-953-51-0825-2. DOI: 10.5772/50711. [Online]. Available: http://www.intechopen.com/books/new-technologies-in-the-oil-and-gas-industry/the-slug-flow-problem-in-oil-industry-and-pi-level-control (visited on 09/05/2023).

[5] H. Xu, T. Tang, B. Zhang and Y. Liu, 'Identification of two-phase flow regime in the energy industry based on modified convolutional neural network,' *Progress in Nuclear Energy*, vol. 147, p. 104 191, 1st May 2022, ISSN: 0149-1970. DOI: 10.1016/j.pnucene.2022.104191. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0149197022000695 (visited on 10/05/2023).

[6] N. Baba, Y. Yamashita and Y. Shiraishi, 'CLASSIFICATION OF FLOW PATTERNS IN TWO PHASE FLOW BY NEURAL NETWORK,' in *Artificial Neural Networks*, T. Kohonen, K. Mäkisara, O. Simula and J. Kangas, Eds., Amsterdam: North-Holland, 1st Jan. 1991, pp. 1617–1620, ISBN: 978-0-444-89178-5. DOI: 10.1016/B978-0-444-89178-5.50149-4. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780444891785501494 (visited on 10/05/2023).

[7] S. Cai, H. Toral and J. Qiu, 'Flow regime identification by a self-organising neural network,' in *ICANN '93*, S. Gielen and B. Kappen, Eds., London: Springer, 1993, pp. 868–868, ISBN: 978-1-4471-2063-6. DOI: 10.1007/978-1-4471-2063-6_251.

[8] S. Cai, H. Toral, J. Qiu and J. S. Archer, 'Neural network based objective flow regime identification in air-water two phase flow,' *The Canadian Journal of Chemical Engineering*, vol. 72, no. 3, pp. 440–445, 1994, _eprint: https://onlinelibrary.wiley.c ISSN: 1939-019X. DOI: 10.1002/cjce.5450720308. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cjce.5450720308 (visited on 13/05/2023).

[9] H. Wu, F. Zhou and Y. Wu, 'Intelligent identification system of flow regime of oil–gas–water multiphase flow,' *International Journal of Multiphase Flow*, vol. 27, no. 3, pp. 459–475, 1st Mar. 2001, ISSN: 0301-9322. DOI: 10.1016/S0301-9322(00)00022-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0301932200000227 (visited on 13/05/2023).

[10] T. Xie, S. M. Ghiaasiaan and S. Karrila, 'Artificial neural network approach for flow regime classification in gas–liquid–fiber flows based on frequency domain analysis of pressure signals,' *Chemical Engineering Science*, vol. 59, no. 11, pp. 2241–2251, 1st Jun. 2004, ISSN: 0009-2509. DOI: 10.1016/j.ces.2004.02.017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0009250904001381 (visited on 13/05/2023).

[11] B. M. Abbagoni and H. Yeung, 'Non-invasive classification of gas–liquid two-phase horizontal flow regimes using an ultrasonic doppler sensor and a neural network,' *Measurement Science and Technology*, vol. 27, no. 8, p. 084 002, Jun. 2016, Publisher: IOP Publishing, ISSN: 0957-0233. DOI: 10.1088/0957-0233/27/8/084002. [Online]. Available: https://dx.doi.org/10.1088/0957-0233/27/8/084002 (visited on 13/05/2023).

[12] Y. Zhang, A. N. Azman, K.-W. Xu, C. Kang and H.-B. Kim, 'Two-phase flow regime identification based on the liquid-phase velocity information and machine learning,' *Experiments in Fluids*, vol. 61, no. 10, p. 212, 14th Sep. 2020, ISSN: 1432-1114. DOI: 10.1007/s00348-020-03046-x. [Online]. Available: https://doi.org/10.1007/s00348-020-03046-x (visited on 13/05/2023).

[13] E. Åbro, V. A. Khoryakov, G. A. Johansen and L. Kocbach, 'Determination of void fraction and flow regime using a neural network trained on simulated data based on gamma-ray densitometry,' *Measurement Science and Technology*, vol. 10, no. 7, p. 619, Jul. 1999, ISSN: 0957-0233. DOI: 10.1088/0957-0233/10/7/308. [Online]. Available: https://dx.doi.org/10.1088/0957-0233/10/7/308 (visited on 13/05/2023).

[14] C. M. Salgado, C. M. N. A. Pereira, R. Schirru and L. E. B. Brandão, 'Flow regime identification and volume fraction prediction in multiphase flows by means of gamma-ray attenuation and artificial neural networks,' *Progress in Nuclear Energy*, vol. 52, no. 6, pp. 555–562, 1st Aug. 2010, ISSN: 0149-1970. DOI: 10.1016/j.pnucene.2010.02.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0149197010000375 (visited on 13/05/2023).

[15] E. Nazemi, S. A. H. Feghhi, G. H. Roshani, R. Gholipour Peyvandi and S. Setayeshi, 'Precise void fraction measurement in two-phase flows independent of the flow regime using gamma-ray attenuation,' *Nuclear Engineering and Technology*, vol. 48, no. 1, pp. 64–71, 1st Feb. 2016, ISSN: 1738-5733. DOI: 10.1016/j.net.2015.09.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1738573315002144 (visited on 13/05/2023).

[16] Z. Yang, H. Ji, Z. Huang, B. Wang and H. Li, 'Application of convolution neural network to flow pattern identification of gas-liquid two-phase flow in small-size pipe,' in *2017 Chinese Automation Congress (CAC)*, Oct. 2017, pp. 1389–1393. DOI: 10.1109/CAC.2017.8242984.

[17] G. M. Hobold and A. K. da Silva, 'Machine learning classification of boiling regimes with low speed, direct and indirect visualization,' *International Journal of Heat and Mass Transfer*, vol. 125, pp. 1296–1309, 1st Oct. 2018, ISSN: 0017-9310. DOI: 10.1016/j.ijheatmasstransfer.2018.04.156. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0017931017346100 (visited on 13/05/2023).

[18] M. Du, H. Yin, X. Chen and X. Wang, 'Oil-in-water two-phase flow pattern identification from experimental snapshots using convolutional neural network,' *IEEE Access*, vol. 7, pp. 6219–6225, 2019, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2888733.

[19] C. Shen, Q. Zheng, M. Shang, L. Zha and Y. Su, 'Using deep learning to recognize liquid–liquid flow patterns in microchannels,' *AIChE Journal*, vol. 66, no. 8, e16260, 2020, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/aic.16260, ISSN: 1547-5905. DOI: 10.1002/aic.16260. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/aic.16260 (visited on 13/05/2023).

[20] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, 'Gradient-based learning applied to document recognition,' *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, Conference Name: Proceedings of the IEEE, ISSN: 1558-2256. DOI: 10.1109/5.726791.

[21] A. Krizhevsky, I. Sutskever and G. E. Hinton, 'ImageNet classification with deep convolutional neural networks,' *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 24th May 2017, ISSN: 0001-0782. DOI: 10.1145/3065386. [Online]. Available: https://dl.acm.org/doi/10.1145/3065386 (visited on 13/05/2023).

[22] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 10th Apr. 2015. DOI: 10.48550/arXiv.1409.1556. arXiv: 1409.1556[cs]. [Online]. Available: http://arxiv.org/abs/1409.1556 (visited on 13/05/2023).

[23] G. G. Yen and H. Lu, 'Acoustic emission data assisted process monitoring,' *ISA Transactions*, vol. 41, no. 3, pp. 273–282, 1st Jul. 2002, ISSN: 0019-0578. DOI: 10.1016/S0019-0578(07)60087-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0019057807600871 (visited on 10/05/2023).

[24] S. Al-lababidi, A. Addali, H. Yeung, D. Mba and F. Khan, 'Gas void fraction measurement in two-phase gas/liquid slug flow using acoustic emission technology,' *Journal of Vibration and Acoustics*, vol. 131, no. 64501, 18th Nov. 2009, ISSN: 1048-9002. DOI: 10.1115/1.4000463. [Online]. Available: https://doi.org/10.1115/1.4000463 (visited on 13/05/2023).

[25] F. A. Holland and R. Bragg, '7 - gas–liquid two-phase flow,' in *Fluid Flow for Chemical Engineers (Second Edition)*, F. A. Holland and R. Bragg, Eds., Oxford: Butterworth-Heinemann, 1st Jan. 1995, pp. 219–267, ISBN: 978-0-340-61058-9. DOI: 10.1016/B978-034061058-9.50009-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780340610589500098 (visited on 09/05/2023).

[26] K. E. Crandell, R. O. Howe and P. L. Falkingham, 'Repeated evolution of drag reduction at the air–water interface in diving kingfishers,' *Journal of The Royal Society Interface*, vol. 16, no. 154, p. 20190125, 15th May 2019, Publisher: Royal Society. DOI: 10.1098/rsif.2019.0125. [Online]. Available: https://royalsocietypublishing.org/doi/full/10.1098/rsif.2019.0125 (visited on 10/03/2023).

[27] 'JFS biomimicry interview series: No.6,' JFS Japan for Sustainability. (), [Online]. Available: https://www.japanfs.org/en/news/archives/news_id027795.html (visited on 10/03/2023).

[28] W. S. McCulloch and W. Pitts, 'A logical calculus of the ideas immanent in nervous activity,' *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1st Dec. 1943, ISSN: 1522-9602. DOI: 10.1007/BF02478259. [Online]. Available: https://doi.org/10.1007/BF02478259 (visited on 10/03/2023).

[29] F. Rosenblatt, 'The perceptron: A probabilistic model for information storage and organization in the brain,' *Psychological Review*, vol. 65, pp. 386–408, 1958, Place: US Publisher: American Psychological Association, ISSN: 1939-1471. DOI: 10.1037/h0042519.

[30] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016.

[31] A. Vaswani *et al.*, 'Attention is all you need,' in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html (visited on 10/05/2023).

[32] J. Berkson, 'Application of the logistic function to bio-assay,' *Journal of the American Statistical Association*, vol. 39, no. 227, pp. 357–365, 1944, Publisher: [American Statistical Association, Taylor & Francis, Ltd.], ISSN: 0162-1459. DOI: 10.2307/2280041. [Online]. Available: https://www.jstor.org/stable/2280041 (visited on 11/05/2023).

[33] A. N. Kolmogorov, *Foundations of the theory of probability*, in collab. with G. A. S. L. University of Florida. New York: Chelsea Pub. Co., 1950, 90 pp. [Online]. Available: http://archive.org/details/foundationsofthe00kolm (visited on 15/05/2023).

[34] G. Cybenko, 'Approximation by superpositions of a sigmoidal function,' *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1st Dec. 1989, ISSN: 1435-568X. DOI: 10.1007/BF02551274. [Online]. Available: https://doi.org/10.1007/BF02551274 (visited on 10/03/2023).

[35] K. Hornik, M. Stinchcombe and H. White, 'Multilayer feedforward networks are universal approximators,' *Neural Networks*, vol. 2, no. 5, pp. 359–366, Jan. 1989, ISSN: 08936080. DOI: 10.1016/0893-6080(89)90020-8. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0893608089900208 (visited on 10/03/2023).

[36] G. Gripenberg, 'Approximation by neural networks with a bounded number of nodes at each level,' *Journal of Approximation Theory*, vol. 122, no. 2, pp. 260–266, 1st Jun. 2003, ISSN: 0021-9045. DOI: 10.1016/S0021-9045(03)00078-9. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021904503000789 (visited on 14/05/2023).

[37] B. Hanin and M. Sellke, *Approximating continuous functions by ReLU nets of minimal width*, 10th Mar. 2018. DOI: 10.48550/arXiv.1710.11278. arXiv: 1710.11278[cs,math,stat]. [Online]. Available: http://arxiv.org/abs/1710.11278 (visited on 14/05/2023).

[38] P. Kidger and T. Lyons, *Universal approximation with deep narrow networks*, 8th Jun. 2020. DOI: 10.48550/arXiv.1905.08539. arXiv: 1905.08539[cs, math, stat]. [Online]. Available: http://arxiv.org/abs/1905.08539 (visited on 14/05/2023).

[39] S. Park, C. Yun, J. Lee and J. Shin, *Minimum width for universal approximation*, 15th Jun. 2020. DOI: 10.48550/arXiv.2006.08859. arXiv: 2006.08859[cs, stat]. [Online]. Available: http://arxiv.org/abs/2006.08859 (visited on 14/05/2023).

[40] D. E. Rumelhart, G. E. Hinton and R. J. Williams, 'Learning representations by back-propagating errors,' *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986, Number: 6088 Publisher: Nature Publishing Group, ISSN: 1476-4687. DOI: 10.1038/323533a0. [Online]. Available: https://www.nature.com/articles/323533a0 (visited on 28/04/2023).

[41] M. A. Nielsen, *Neural networks and deep learning*, Type: misc, 2018. [Online]. Available: http://neuralnetworksanddeeplearning.com/.

[42] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 29th Jan. 2017. DOI: 10.48550/arXiv.1412.6980. arXiv: 1412.6980[cs]. [Online]. Available: http://arxiv.org/abs/1412.6980 (visited on 04/05/2023).

[43] L. Alzubaidi *et al.*, 'Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions,' *Journal of Big Data*, vol. 8, no. 1, p. 53, 31st Mar. 2021, ISSN: 2196-1115. DOI: 10.1186/s40537-021-00444-8. [Online]. Available: https://doi.org/10.1186/s40537-021-00444-8 (visited on 15/05/2023).

[44] Z. Zhang, 'Derivation of backpropagation in convolutional neural network (CNN),'

[45] A. Sanchis, G. W. Johnson and A. Jensen, 'The formation of hydrodynamic slugs by the interaction of waves in gas–liquid two-phase pipe flow,' *International Journal of Multiphase Flow*, vol. 37, no. 4, pp. 358–368, 1st May 2011, ISSN: 0301-9322. DOI: 10.1016/j.ijmultiphaseflow.2010.11.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0301932210001941 (visited on 10/05/2023).

[46] R. Bitter, T. Mohiuddin and M. Nawrocki, *LabVIEW: Advanced programming techniques*. Crc Press, 2006.

[47] K. Rao, D. Kim and J.-J. Hwang, *Fast Fourier Transform - Algorithms and Applications* (Signals and Communication Technology). Dordrecht: Springer Netherlands, 2010, ISBN: 978-1-4020-6628-3 978-1-4020-6629-0. DOI: 10.1007/978-1-4020-6629-0. [Online]. Available: http://link.springer.com/10.1007/978-1-4020-6629-0 (visited on 07/05/2023).

Appendix A

MNIST Audio Case Study

In order to ensure the validity of convolutional neural network and implementation utilized in this thesis work, a test was conducted on an audio version of the MNIST dataset [1]. The audio data is subjected to a spectrogram transformation, before being fed into the network. The proposed model has an accuracy above 95% for previously unseen data.

AudioMNIST dataset

The Modified NIST database of handwritten digits contains in total 70000 samples and is a subset of a larger set available from NIST (the USA's National Institute for Standards and Technology). Each sample is originally given as a 28x28 array where elements value is given by an unsigned 8-bit integer, but the dimensionality changes across different sources [2]. Due to its size, relative simplicity, and availability (included in the main machine learning frameworks by default [3]) the MNIST dataset for handwritten digits has become a natural starting point for multi label classification.

The AudioMNIST dataset consists of 30000 samples of English spoken digits and its total length is roughly 9.5 hours. Data was accumulated from a total of 60 different individuals, both male and female, ranging in age from 22 to 61 years old. Each speaker provided 500 samples, 50 samples per digit. Although additional information (dialect, age, sex, etc.) was not utilized in this study, knowing that variation between

[1] https://github.com/soerenab/AudioMNIST

[2] http://yann.lecun.com/exdb/mnist/

[3] SciKit Learn: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html#sklearn.datasets.load_digits
PyTorch: https://pytorch.org/vision/stable/generated/torchvision.datasets.MNIST.html#torchvision.datasets.MNIST
Tensorflow: https://www.tensorflow.org/api_docs/python/tf/keras/datasets/mnist/load_data

speakers is present in the dataset might have a soothing effect on the sceptic reader. However, it is worth noting that the majority of speakers were male (48/60), non-native speakers (57/60). The distribution of origin for the speakers is given in figure A.1, which will give an indication of the dialects the model is exposed to.

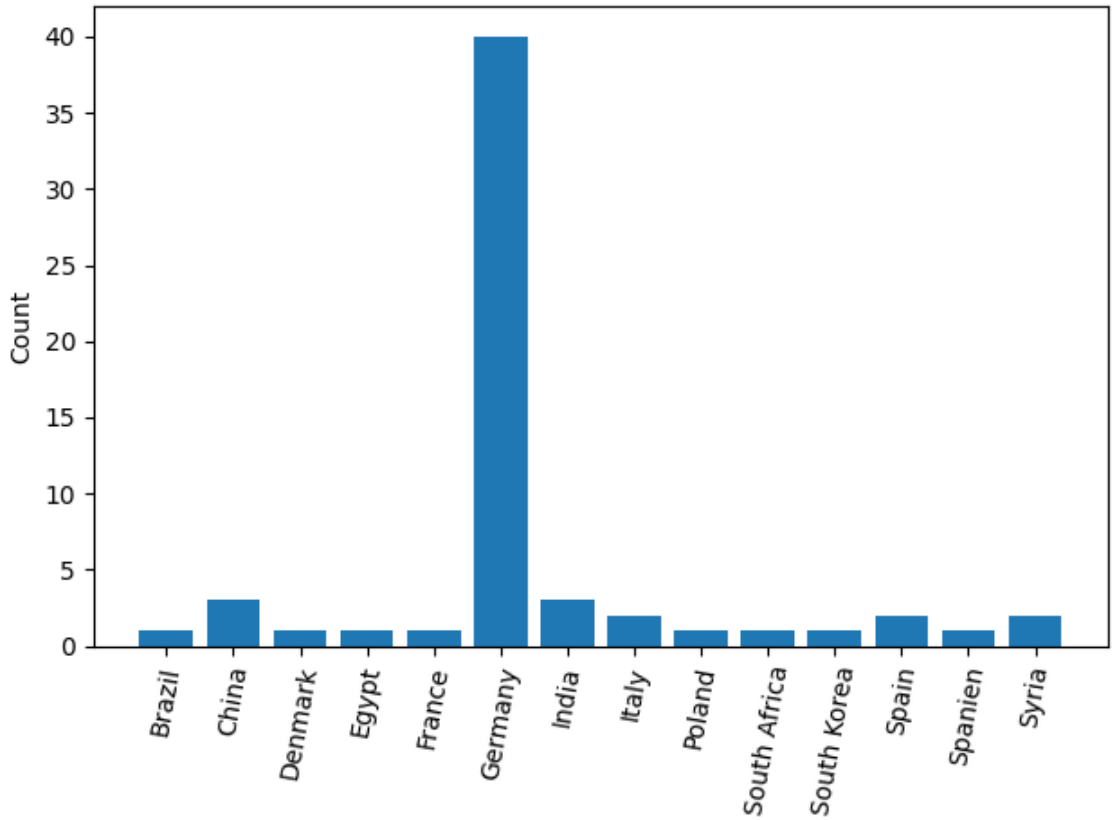
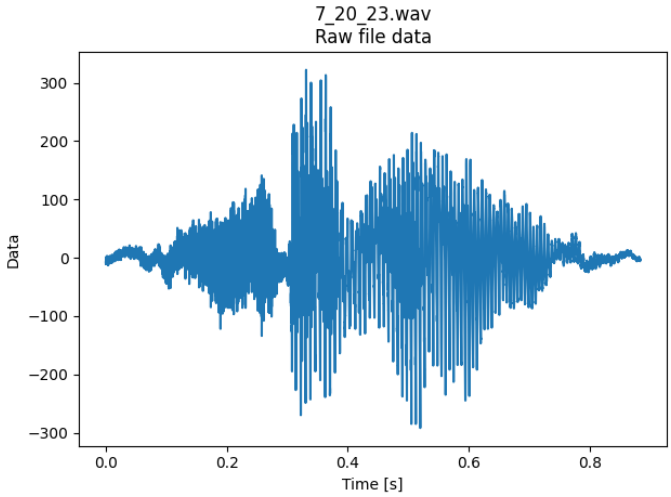
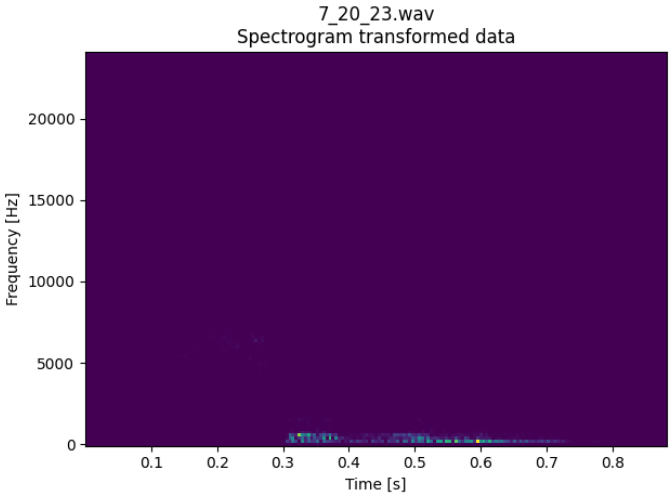


Figure A.1: Number of speakers for the AudioMNIST dataset, sorted by their country of origin. Note: The double category count (Spain/Spanien) is a consequence of the meta data file provided by the Audio-source.

As a consequence, the model might show a higher proficiency in classifying spoken digits when the voice comes from a male with a German dialect. Confirming this is outside the scope of this case study.



(a) Raw data. Note that the y-axis is dimensionless as sound is measured in dB, which is dimensionless. In this case, the data value is a signed 16-bit integer.



(b) Spectrogram transformation of the data shown in A.2a. Note that the lower frequencies dominate because this is a recording of a human voice.

Figure A.2: Example of data and its spectrogram transformation, file `7_20_23.wav` was used.

Data treatment

Each data point from the dataset is a short (∼0.5-2 sec) .wav file with sampling rate of 48 kHz. Utilizing the method `scipy.io.wavfile.read` from the scientific python package, the .wav file is converted into a 1D array of 16-bit signed integers. The array is then transformed into a spectrogram by `scipy.signal.spectrogram`. Our sample has now gone from a sound clip to a matrix, which can be represented as a picture, as can be seen in figure A.2.

In order to ensure identical dimensionality for all data points, the matrices were zero-padded. Since the sound clips vary in length, the matrices will vary in width, but the network which the model is build upon expects data in a specific format. Thus, it is necessary that all input data has the same shape. These spectrogram matrices of data are then saved, to later be loaded during the training of the model itself. Treating the data this way offers a few advantages. Firstly, the amount of matrices loaded in memory is now a parameter which can be changed before each training cycle. Considering how the matrices occupy a total of $\sim$ 20 GB, loading them all to memory is not necessarily a possibility for older hardware.

Furthermore, the burden of computing the spectrogram for a given sound clip, as well as padding, can now be avoided at runtime, ensuring more computational

resources towards the training of the model.

Architecture and Model Hyperparameters

In this case study a Convolutional Neural Network was utilized. The model has a sequential structure, consisting of three convolutions, a flattening of the data into a 1D array, and then a feed forward section which in the end produces the final prediction. The layers, and additional information, are presented in Table A.1

Sequential Order	Layer	Activation Function	Additional	Number of Kernels	Kernel Size
1	Conv2D	Relu	Max Pooling 2x2	8	16×16
2	Conv2D	Relu	Max Pooling 2x2	16	8×8
3	Conv2D	Relu	Max Pooling 2x2	32	5×5
4	Flatten				
5	Dense	Relu			
6	Dense	Relu			
7	Dense	Relu			

Table A.1: Structure of the tested convolutional neural network, with layer-wise explanations.

Seeing as this is a classification problem, the Cross Entropy Loss function was used as a loss function.

The Adaptive momentum Stochastic Gradient Descent algorithm was used, thus no learning rate hyperparameter was explicitly used. By default, however, PyTorch uses an initial learning rate of 0.001. The upper limit on number of epochs was set to 30.

Results & Discussion

The accuracy of the model during training is illustrated in fig A.3.

The highest accuracy for the test dataset was 0.9795 which was achieved at epoch 22. For the train dataset the highest accuracy was 0.9914 and achieved at epoch 12.

The confusion matrices for epoch 12 and epoch 22 are given in fig A.4 and fig A.5.

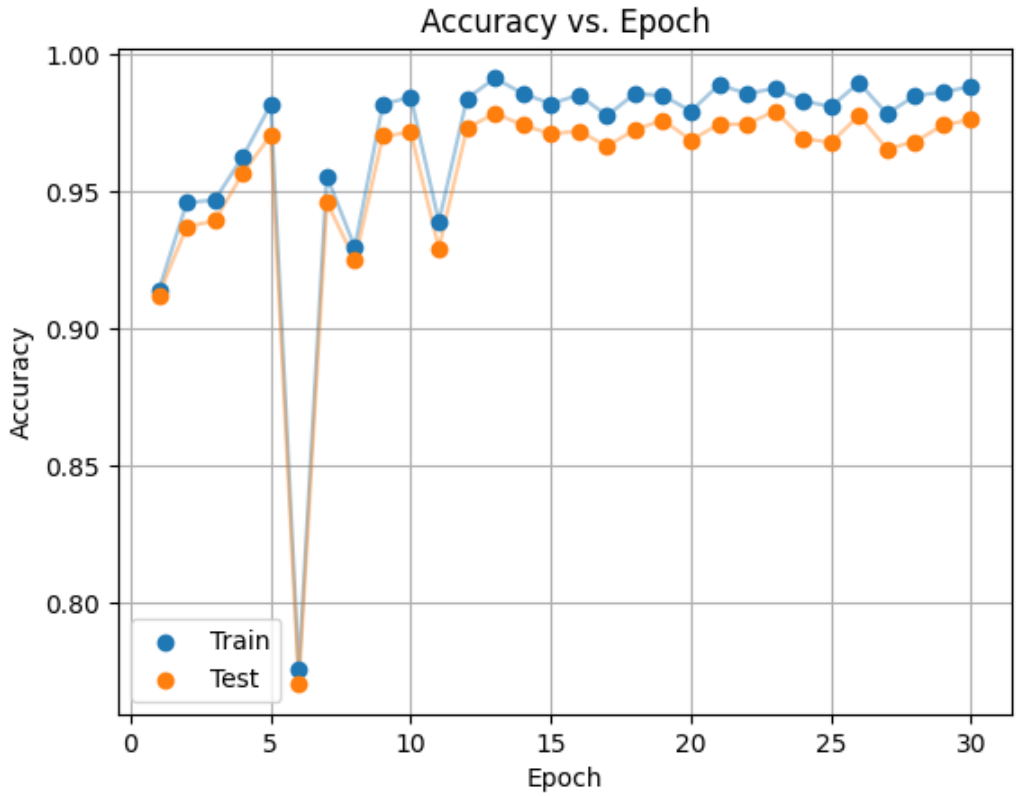
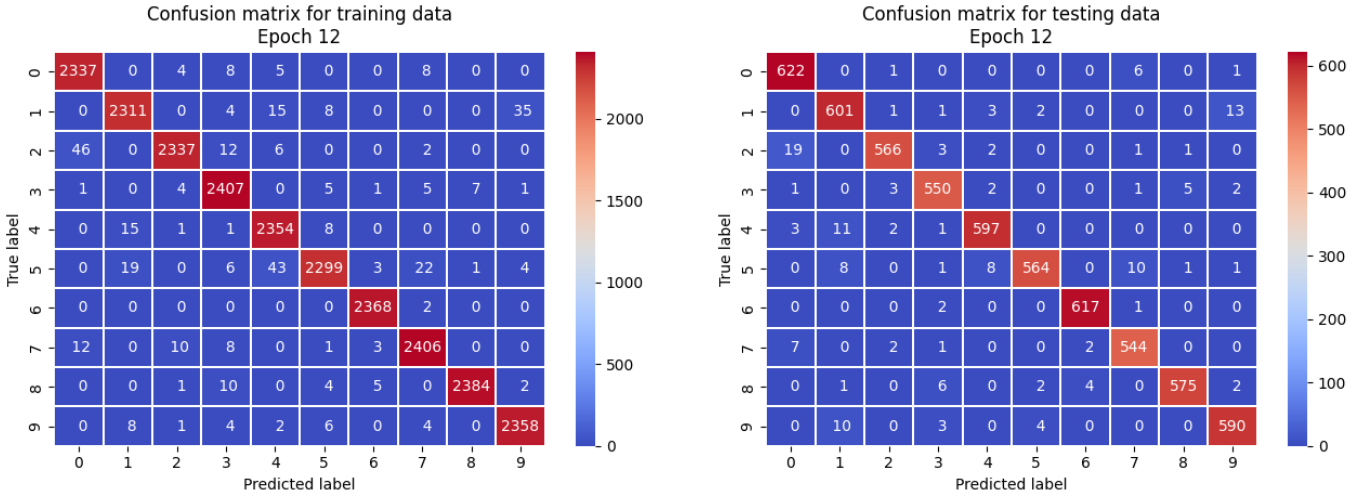


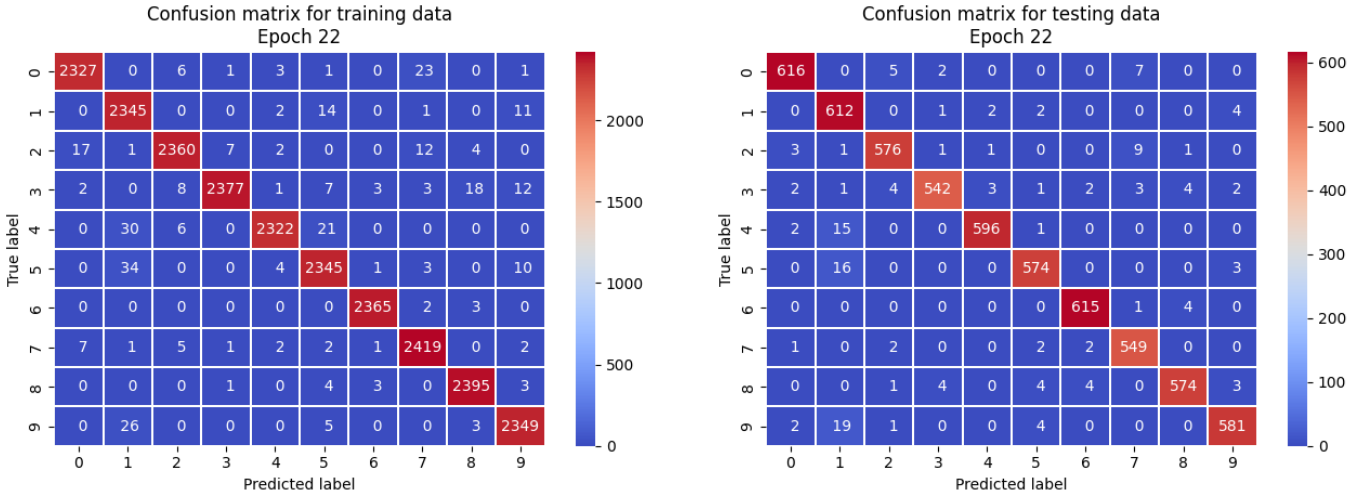
Figure A.3: The accuracy of the model as a functions of epochs trained, given as a fraction. Note that testing was done after each epoch, meaning that even for epoch = 1 the model had seen every data point and updated its weights and biases accordingly.



(a) Training data.

(b) Testing data.

Figure A.4: Confusion Matrix for True labels vs. Predicted labels for train and test data in epoch 12.



(a) Training data.

(b) Testing data.

Figure A.5: Confusion Matrix for True labels vs. Predicted labels for train and test data in epoch 22.

From figures A.3, A.4 and A.5 it is clear that the model is able to classify the audio signals with high precision. When studying the confusion matrices we see that for the training set at epoch 12 the lowest accuracy was for the audio samples with the true label of 5. In this case the accuracy is 0.9591 (2299/2397). For the best test set epoch, the lowest accuracy is 0.9571 (581/607) and occurred for the true label 9. Even though some of the other values on the diagonal are smaller, the accuracy's are better. This is simply due to the differences between the sizes in the data set. The fact that the proposed model still has some errors in the training data indicates that the model is resilient against overfitting.

Conclusion

In this study a convolutional neural network is applied in order to study classification of spoken digits which has undergone a spectrogram transformation. The aim is to explore whether this kind of implementation is suitable for further audio classifications. The results show that the model has a high accuracy (>95%) when tested on previously unseen data. To conclude, the use of a spectrogram transformation in combination with convolutional neural networks in order to classify audio data is a promising way to correctly classify previously unseen data. Given the reason for this case study; Testing the validity of the proposed method, we can safely conclude that this implementation meets this authors expectations.