COPPE
Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia
UFRJ

IDENTIFICATION OF DEFECT-PROPAGATION STAGE IN RIGID PIPES BY MEANS OF ACOUSTIC-EMISSION DATA IN STREAMING FORMAT AND NEURAL NETWORKS

Luiz Rennó Costa

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Luiz Pereira Calôba

Rio de Janeiro
Março de 2019

IDENTIFICATION OF DEFECT-PROPAGATION STAGE IN RIGID PIPES BY MEANS OF ACOUSTIC-EMISSION DATA IN STREAMING FORMAT AND NEURAL NETWORKS

Luiz Rennó Costa

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2019

Costa, Luiz Rennó

Identification Of Defect-Propagation Stage In Rigid Pipes By Means Of Acoustic-Emission Data In Streaming Format And Neural Networks/Luiz Rennó Costa. – Rio de Janeiro: UFRJ/COPPE, 2019.

XV, 82 p.: il.; $29,7$cm.

Orientador: Luiz Pereira Calôba

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2019.

Referência Bibliográfica: p. 63 – 66.

1. Neural Networks. 2. Acoustic Emission. 3. Industry 4.0. I. Calôba, Luiz Pereira. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Lakad Matataaag! Normalin,
Normalin.

Acknowledgements

Gostaria de agradecer a todos.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

IDENTIFICAÇÃO DE ESTÁGIO DE PROPAGAÇÃO DE DEFEITOS EM DUTOS RÍGIDOS POR MEIO DE DADOS DE EMISSÃO ACÚSTICA EM FORMATO DE STREAMING E REDES NEURAIS

Luiz Rennó Costa

Março/2019

Orientador: Luiz Pereira Calôba

Programa: Engenharia Elétrica

Testes destrutivos e não destrutivos são a base para entender as propriedades físicas dos materiais. Por exemplo, defeitos em um tubo sob pressão podem se propagar até produzir uma falha; portanto, uma correta identificação e análise de defeitos é da extrema importância prática. Uma ampla classe de testes não destrutivos explora o fato que materiais sob pressão emitem ondas acústicas. Estas ondas geram dados que são analisados por um especialista para a avaliação do material. Este trabalho propõe o uso de redes neurais *feedforward* para automatizar o processo de análise de dados. Para alcançar este objetivo, as emissões acústicas são divididas em três classes de acordo com seu "padrão": Sem Propagação (NP), Propagação Estável (SP) e Propagação Instável (UP). A capacidade de classificar corretamente as emissões acústicas geradas por um defeito permite classificar por qual o nível de risco o sistema está passando. Foi possível alcançar uma taxa de classificação de mais de 85% para dois conjuntos distintos de dados, mostrando que tais métodos têm potencial para aplicações práticas.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

IDENTIFICATION OF DEFECT-PROPAGATION STAGE IN RIGID PIPES BY MEANS OF ACOUSTIC-EMISSION DATA IN STREAMING FORMAT AND NEURAL NETWORKS

Luiz Rennó Costa

March/2019

Advisor: Luiz Pereira Calôba

Department: Electrical Engineering

Destructive and non-destructive tests are basic for understanding physical properties of materials. For instance, defects in a pipe under pressure can propagate until failure; therefore, a proper identification and analysis of defects is of the greatest practical importance. A wide class of non-destructive tests exploits the fact that materials under pressure emit acoustic waves, and data from acoustic emissions are analysed by a specialist. This thesis proposes the use of feedforward neural networks to automate the process of data analysis. To achieve this goal, acoustic emissions are divided into three classes according to their "pattern": no propagation (NP), stable propagation (SP) and unstable propagation (UP). The ability of correctly classifying the acoustic emissions generated by a defect permits to classify which level of risk the system is undergoing. A classification rate higher than 85% was achieved using two distinct datasets, showing that such methods have a potential for practical applications.

Contents

List of Figures

List of Tables

List of Symbols

S_u	Ultimate Strength, p. 18
$\Phi(\cdot)$	Activation Function, p. 7
$\bar{x}$	Mean of x, p. 15
$\circ$	Hadamard (element-wise) product., p. 9
η	Learning stepsize, p. 8
$\mathbf{W}_{(l)}$	Weight Matrix for layer l, p. 8
$\mathbf{u}_l$	Input of layer l., p. 8
$\mathbf{z}_l$	Output of layer l., p. 8
$\oslash$	Hadamard (element-wise) division, p. 10
σ_x	Standard Deviation of x, p. 15
$\mathbf{x}$	Model Input Data, p. 3
$\mathbf{y}$	Model Target Data, p. 3

List of Acronyms

ADC	Analogue-to-Digital Converter, p. 24
AE	Acoustic Emission, p. 1
AF	*Average Frequency*, p. 37
AI	Aritificial Intelligence, p. 5
ANN	Artificial Neural Network, p. 6
ASL	*Average Signal Level*, p. 37
CM	Confusion Matrix, p. 57
CTP	*Count-To-Peak*, p. 37
CV	Cross-Validation, p. 16
FME	First Moment Estimator, p. 12
GD	Gradient Descent, p. 8
HDT	Hit Definition Time, p. 34
HLT	Hit Lockout Time, p. 34
KE	Kaiser Effect, p. 20
LAMEF	*Laboratório de Metalurgia Física*, p. 22
LS	Linearly-separable, p. 41
MA	*Mean Amplitude*, p. 37
MLP	Multi-Layer Perceptron, p. 7
ML	Machine Learning, p. 3
NDT	Non-Destructive Testing , p. 19

NI	National Instruments, p. 24
PAC	Physical Acoustics, p. 22
PCA	Principal Component Analysis, p. 21
PDT	Peak Definition Time, p. 34
PLA	Perceptron Learning Algorithm, p. 6
RES	*Resolution Level*, p. 37
RF	*Reverberation Frequency*, p. 37
RMS	Root-Mean-Square, p. 37
SME	Second Moment Estimator, p. 12
SNR	Signal-to-Noise Ratio, p. 33
SOM	Self-Organizing Map, p. 20
TDMS	Technical Data Management Streaming, p. 24
TOFD	Time-Of-Flight Diffraction, p. 23
TTE	Transition Time Estimation, p. 41
UFRGS	*Universidade Federal do Rio Grande do Sul*, p. 22
VTHR	*Variable Threshold*, p. 37

Chapter 1

Introduction

1.1 Introduction

The increasing demand for energy over the past decades created a surge of renewable sources, as opposed to the more traditional petroleum fuel. However, that still is an essential fuel for our society, so that several companies like Shell, Exxon, and Petrobras have extensive and complex networks for extracting and distributing different kinds of liquid fuel through pipelines. Brazil for instance has roughly 8.000 kilometres (km) [7] of oil pipelines (both refined and crude) scattered throughout the country.

In order to maintain and prevent failure of those extensive structures (which could be catastrophic) several tests, both destructive and non-destructive were developed, and amongst the latter, one that stands out is the Acoustic-Emission (AE) test. It relies on the radiation of acoustic (physical) waves that occurs when a material undergoes irreversible changes both at a micro and macroscopic scale (Section 1.8).

Acoustic-Emission testing is used to determine if a structure has a defect or not. Which is intuitive since a irreversible change most likely means damage to the structure. This thesis however tries to take it one step further.

Since each AE comes from a permanent damage, subsequent AEs should be different. As the material is progressively damaged, the AEs should become all the more frequent as well.

Those changes can come from the propagation of a crack, corrosion, or in other words, irreversible damage to the material structure. Therefore it should be possible to monitor and tell in real time how dangerous it is.

Unfortunately, operating pipelines produce a lot of noise, which can easily drown the AE signal. Therefore, the capture and processing of the data is essential. Industrial equipment are somewhat robust and have been used successfully as a tool for this NDT. However, industrial AE equipment requires a specialized analyst to interpret its data.

1.2 Objective

In order to bring this section of the industry to the trends of Industry 4.0, this work proposes an adaptive model, capable of learning what the specialized analyst does. Furthermore, this work takes it one step further proposing a fully fledged data mining and processing software as an alternative to the existing industrial equipment.

The main objective of this thesis is to develop an automated system to extract AE waves from a streaming data source and an artificial neural network model to determine which propagation stage the crack is at, based on the aforementioned treated data.

1.3 Theory

This section gives a brief revision of the theoretical topics this thesis extends upon.

1.3.1 Learning

Learning is something all beings have experienced one way or another, it is intrinsic to our nature and an essential process to our evolution as a society. It can be defined as a acquisition of knowledge through interaction, be it with the outside world like books, people, or with one's own self.

Knowledge however, is a more abstract concept and can be interpreted in several different ways, for instance, both knowing how to sew a scarf and differentiating square from circle can be considered knowledge. The difference between those is that in the latter, the knowledge is static, interacting with it means only identifying which is which. Knowing how to sew a scarf however, means attaining domain over the cloth's process of transformation, given a simple piece of cloth (input), one would always be able to transform it into a scarf.

The ability to learn a transformation process is something rather powerful because one can change properties of the output merely changing the input, instead of relearning the whole process again, for example, being taught how to sew using only blue cloth does not impede one to sew a red scarf, needed only to change the fabric one applies the process.

Trying to create a machine that is able to act like we (thinking entities) do has been an open challenge since the 1950s [8] and motivated countless studies amongst several decades on the field of Machine Learning (ML).

It is important to note however that mathematical complexity is a central factor in ML. Separating squares from circles can be done using rather simple mathematical formulae, sewing a scarf not. One can even map all the hand movements and develop formulae that reign it, but the complexity would make computation unrewarding. The goal of ML is to create a simpler model capable of acquiring that complex knowledge through learning.

However, another problem is still unresolved, how to measure the learning process. One can find it learnt how to sew a scarf when it does not deforms when pulled, or when it satisfies someone else. Expanding on that latter concept, one can for instance, attribute a beauty scale for the scarf (suppose for simplicity that the scale goes from 1 to 10), and only consider that it learnt how to sew a scarf when it can reliably sew a scarf that receives a 10 from the beauty scale.

The creation of a measure capable of telling if learning is in fact happening is of utmost importance for the learning process. We can then extend the definition of learning to the acquisition of knowledge through interaction that, as a by-product, increases the performance measure associated with that knowledge. The contrary is also possible, instead of increasing a performance measure, one can decrease an error measure.

Learning from the ML point of view can be separated in 3 main classes, Supervised Learning (Section 1.3.2), Unsupervised Learning (Section 1.3.3) and Reinforced Learning. All those have the same principles, they all have a model, performance measure and a task to perform. The main difference is how they receive and interpret the input data.

1.3.2 Supervised Learning

As an undergrad is taking a calculus course, he is given several examples and exercises. Examples are composed of simpler questions and the necessary procedures, after reading them and taking the more challenging exercises, he gives them to the teacher, which in turn proceeds to tell him what he did right or wrong. He is then given a score based on how many questions he had right (how many hit the "target"), this procedure is repeated until the undergrad is satisfied with his score.

That is a prime example of Supervised Learning, as it can be defined as a method to train the model by directly teaching it. This is achieved through the addition of "labels" or "targets". The procedure from a mathematical point of view (Figure 1.1) is fairly straightforward, your input data $\mathbf{x}$ is fed to a model that yields the output $\tilde{\mathbf{y}}$. The output is compared to the target $\mathbf{y}$ and a performance score or cost $J(\cdot)$ is calculated.

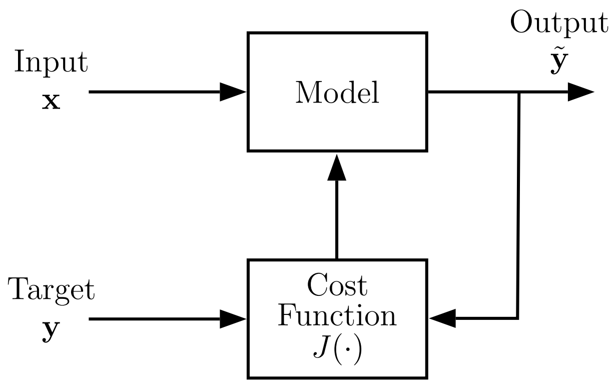


Figure 1.1: Simplified Supervised Learning Schematic

1.3.3 Unsupervised Learning

Contrary to the aforementioned Supervised Learning method, Unsupervised Learning does not have a well specified "target" and therefore the model now has a different purpose, to create some sort of structure based solely on the inputs and the relationships between them. Unsupervised Learning typically translates to clustering.

Clustering is the action of grouping together inputs that have some degree of similarity, for instance, trying to separate different animals by the number of legs they possess, or which environment they reside is a type of unsupervised clustering since it depends only on the characteristics of the input data. A good example of this approach is recommendation systems.

1.4 Artificial Neural Networks

What mainly highlights our species is the seemingly unending capacity to learn and adapt, both individually and collectively as a society. One can not deny that one of the main reasons for it is the development of our most complex and important organs, the brain. The brain hosts 10 to 20 billion neurons, its main component. Neurons are specialized cells physically interconnected through its dendrites and axons (Figure 1.2). These connections are susceptible to electrical impulses called synapses, sent through the axons and received from the dendrites.

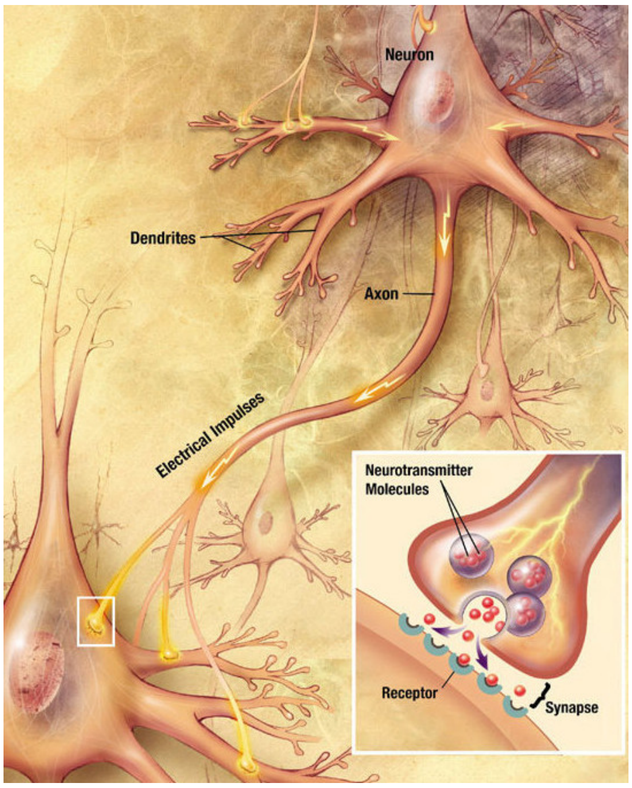


Figure 1.2: Structure of a Neuron and its Connections. Taken from [1]

It has always been an interest of mankind to create an artificial brain. Several cinematographic pieces depict (in a romanticized way) an Artificial Intelligence (AI) like *2001, a Space Odyssey*; *I, robot*; etc. This topic also inspired countless researchers to study and develop mathematical models that try to mirror the behaviour of the brain.

It began in the end of 1943 when Warren S. McCulloch [9] and Walter Pitts started to "translate" the neurons to a mathematical model using logical observations. They stated that the neuron behaved much like a "all-or-none" model, that is, either it activated and sent forth a signal, or nothing happened. However, that only allowed for binary inputs and outputs, which significantly hindered the model.

It was only after the definition of what is known as Hebb's rule [10] that the model would be revamped:

> "Let us assume that the persistence or repetition of a reverberatory activity (or 'trace') tends to induce lasting cellular changes that add to its stability. ... When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

This rule basically states that interconnecting neurons have a change in synaptic efficiency. It was then that Frank Rosenblatt [11] developed what is now known as a *Perceptron*, creating weights that change according to the activation of the neuron.

The Artificial Neural Network (ANN) is a supervised learning model (as the name states) inspired by that complex structure of the interconnected neurons, mainly composed of *Perceptrons*.

1.5 Perceptron

The *Perceptron* can be interpreted as mathematical model for the neuron (Figure 1.3), it receives an input vector ($\mathbf{x}$) through the dendrites, modifies it and releases an output ($\mathbf{y}$) down the axons.

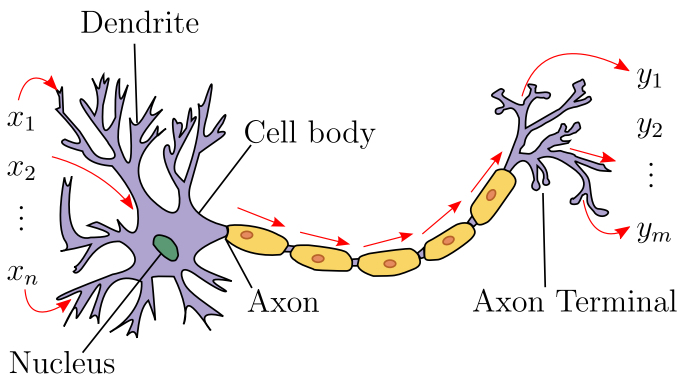


Figure 1.3: Structure of a Neuron with Abstract Inputs and Outputs. Adapted from [2].

Where x_n and y_m are elements from the n and m-dimensional input and output vectors ($\mathbf{x}$ and $\mathbf{y}$) respectively. Classically, this model was used as a binary classifier using the synapses as weights, $m = 1$ and the addition of a bias (b) element, which gives:

$$y = f(\mathbf{wx} + b) = sign(\mathbf{wx} + b) = \begin{cases} 1, & \text{if } \mathbf{wx} + b > 0 \\ -1, & \text{otherwise} \end{cases} \tag{1.1}$$

Where $\mathbf{w}$ is a $[1 \times n]$ vector of weights. This model can be interpreted as a line in the input space that can both be translated (relative to the origin) and rotated. That line acts as a boundary separating two regions, or in other words, classifying two different groups of data.

The *Perceptron* Learning Algorithm (PLA) is rather simple, for

However, the boundary is still a straight line, meaning that if the data is not linearly separable (vast majority of real cases) it is impossible to achieve zero classification error. It would be interesting then to create different sorts of non-linear boundaries, to achieve that, one can "smooth" the underlying function $f(\mathbf{wx})$ using for example $tanh(\cdot)$, $arctan(\cdot)$, etc.

Also, this thesis is going to adopt a slightly different notation, $\mathbf{wx} + b \rightarrow \mathbf{wx}$ where $w_0 = 0$ and $x_0 = 1$ thus giving:

$$\mathbf{y} = f(\mathbf{wx} + b) = f(\mathbf{wx}) \rightarrow \Phi(\mathbf{wx}) \tag{1.2}$$

Where $\mathbf{w}$ are the synapses (weights) and $\Phi(\cdot)$ is an arbitrary mapping (activation function). It is important to note that there is a certain "flow" of information from $\mathbf{x}$ to $\mathbf{y}$, there is no feedback in the *Perceptron* model (Figure 1.4).

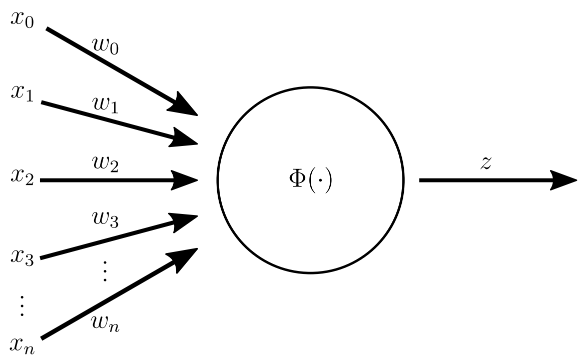


Figure 1.4: Complete Perceptron Mathematical Model.

1.5.1 Multi Layer Perceptron

The *perceptron* is a simple model, and thus rather limited. However, like the brain, real adaptive prowess comes when interconnecting several layers of *perceptrons* (Figure 1.5). Therefore creating a Multi Layer *Perceptron* (MLP) network. These kind of ANN are referred to as *shallow* or *vanilla* networks.

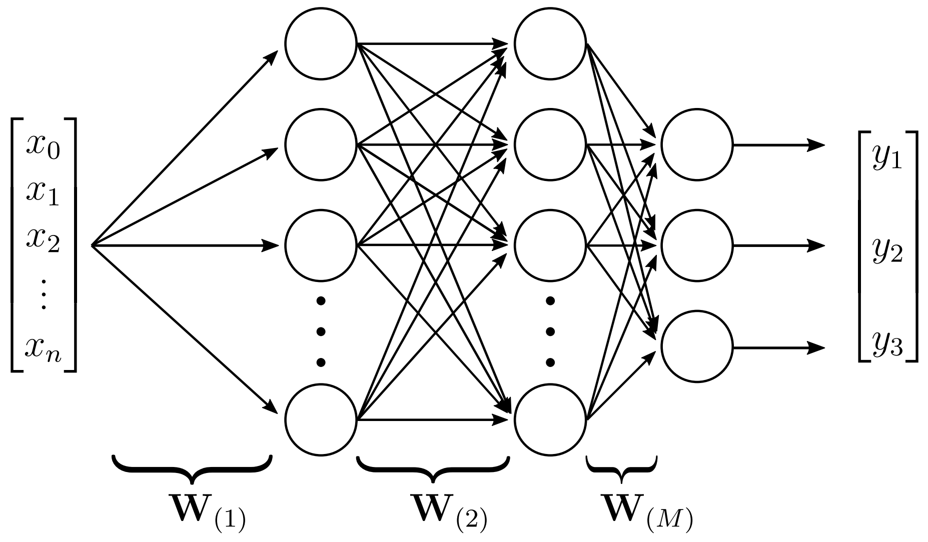


Figure 1.5: Multi Layer *Perceptron* Schematic.

Although one may think this to be a rather straightforward modification, it was not until 1986 when Hinton et al. developed an algorithm that was capable of efficiently training such a complex network, *Backpropagation* [12].

1.5.2 Backpropagation

Backpropagation as a learning algorithm for ANNs was first defined in 1986 [12], this work gave fuel to researchers since ANNs where always hindered by the computational power available at the time.

First, defining $\mathbf{W}_{(l)}$ the weight matrix for layer l, and using the notation $\mathbf{W} = (\mathbf{W}_{(1)}, ..., \mathbf{W}_{(M)})$, the neural network defines a function $\mathbf{y} = f(\mathbf{x}; \mathbf{W})$.

Where $\mathbf{x}$ is the vector of the independent variables, and $\mathbf{W}$ can be interpreted as a parameter. While $\mathbf{y}$ is a notation for the output of the model, $\tilde{\mathbf{y}}$ is the value that is actually observed, in correspondence of some $\mathbf{x}$. Given an observation pair $(\mathbf{x}, \tilde{\mathbf{y}})$, there is an associated loss $J(\mathbf{y}, \tilde{\mathbf{y}})$ that measures how far the model output $\mathbf{y}$ is from the actual output $\tilde{\mathbf{y}}$. For instance, one common choice is the Euclidean square norm $J(\mathbf{y}, \tilde{\mathbf{y}}) = \|\mathbf{y} - \tilde{\mathbf{y}}\|^2$.

$$J(\boldsymbol{\xi}) = \boldsymbol{\xi}\boldsymbol{\xi}^T \quad \Bigg| \quad \boldsymbol{\xi}(\mathbf{y}, \tilde{\mathbf{y}}) = \mathbf{y} - \tilde{\mathbf{y}} \tag{1.3}$$

Given a sample $(\mathbf{x}_{(i)}, \tilde{\mathbf{y}}_{(i)})$ of N observations, $i = 1, \ldots, N$, the loss function $J(\boldsymbol{\xi})$ is defined as the sum (or average) of the costs for each observation in the sample. Training a network means to find the value of $\mathbf{W}$ that minimizes J. Seeing it as an optimization problem, several solutions can be applied, for instance, one can apply Gradient Descent (GD), an optimization algorithm that uses the gradient:

$$\mathbf{W}_{(l)}[k+1] = \mathbf{W}_{(l)}[k] - \eta \frac{\partial J}{\partial \mathbf{W}_{(l)}}[k] \tag{1.4}$$

Where k is the kth iteration. However, calculating this gradient for the inner layers is a bit tricky since it depends on the outer layers. This can be done by iterating each layer backwards (from output to input). Before detailing the algorithm *per se*, a few variables and notation need defining. The input of the l-th layer is denominated $\mathbf{u}_{(l)} = \mathbf{W}_{(l)}\mathbf{z}_{(l-1)}$, where $\mathbf{z}_{(l-1)}$ is the output of the previous layer, $l-1$. Calculating the gradient on the last layer gives:

$$\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{W}_{(M)}} = \frac{\partial J(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}} \frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{z}_{(M)}} \frac{\partial \mathbf{z}_{(M)}}{\partial \mathbf{u}_{(M)}} \frac{\partial \mathbf{u}_{(M)}}{\partial \mathbf{W}_{(M)}} \tag{1.5}$$

Since $\boldsymbol{y} = \mathbf{z}_{(M)}$, $\mathbf{z}_{(M)} = \boldsymbol{\Phi}(\mathbf{u}_{(M)})$ and $\mathbf{u}_{(M)} = \mathbf{W}_{(M)}\mathbf{z}_{(M-1)}$, Equation 1.5 yields:

$$\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{W}_{(M)}} = \frac{\partial \boldsymbol{\Phi}(\mathbf{u}_{(M)})}{\partial \mathbf{u}_{(M)}} \circ \frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{y}} \, \mathbf{z}_{(M-1)}^T \tag{1.6}$$

Where $\circ$ is the Hadamard (element-wise) product. Although the transition from Eq. 1.5 to Eq. 1.6 may not be trivial, dimensional analysis can be done to check if at least the dimensions match. $\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{y}}$ is a $[N_y \times P]$ matrix, $\mathbf{z}_{(M-1)}$ has $[N_y \times P]$ dimension and $\frac{\partial \boldsymbol{\Phi}(\mathbf{u}_{(M)})}{\partial \mathbf{u}_{(M)}}$ is a $[N_y \times P]$ matrix. With P being the number of observation points used and N_y the output dimensionality (number of parameters).

Calculating the gradient for the second-to-last layer using Eq. 1.5:

$$\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{W}_{(M-1)}} = \frac{\partial \boldsymbol{\Phi}(\mathbf{u}_{(M-1)})}{\partial \mathbf{u}_{(M-1)}} \circ \mathbf{W}^T_{(M)} \left(\frac{\partial \boldsymbol{\Phi}(\mathbf{u}_{(M)})}{\partial \mathbf{u}_{(M)}} \circ \frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{y}} \right) \mathbf{z}^T_{(M-1)} \tag{1.7}$$

Defining:

$$\boldsymbol{\delta}_{(M)} = \frac{\partial \boldsymbol{\Phi}(\mathbf{u}_{(M)})}{\partial \mathbf{u}_{(M)}} \circ \frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{y}} \tag{1.8}$$

It is possible to rewrite Equations 1.6 and 1.7 as:

$$\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{W}_{(M)}} = \boldsymbol{\delta}_{(M)} \mathbf{z}^T_{(M-1)} \tag{1.9}$$

$$\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{W}_{(M-1)}} = \boldsymbol{\delta}_{(M-1)} \mathbf{z}^T_{(M-2)} \tag{1.10}$$

Where:

$$\boldsymbol{\delta}_{(M-1)} = \frac{\partial \boldsymbol{\Phi}(\mathbf{u}_{(M-1)})}{\partial \mathbf{u}_{(M-1)}} \circ \left(\mathbf{W}^T_{(M)} \boldsymbol{\delta}_{(M)} \right) \tag{1.11}$$

Thus one can arrive at the two main formulae for applying any optimization method that requires the first order gradient:

$$\boldsymbol{\delta}_{(l)} = \frac{\partial \boldsymbol{\Phi}(\mathbf{u}_{(l)})}{\partial \mathbf{u}_{(l)}} \circ \left(\mathbf{W}^T_{(l+1)} \boldsymbol{\delta}_{(l+1)} \right) \tag{1.12}$$

$$\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{W}_{(l)}} = \boldsymbol{\delta}_{(l)} \mathbf{z}^T_{(l-1)} \tag{1.13}$$

Needed only to define:

$$\boldsymbol{\delta}_{(M+1)} = \frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{y}} \tag{1.14}$$

$$\mathbf{W}_{(M+1)} = \mathbf{I} \tag{1.15}$$

The fact that $\boldsymbol{\delta}_{(l)}$ depends on $\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{y}}$ and it is calculated from the exterior layers to the interior, one can interpret $\boldsymbol{\delta}_{(l)}$ as a portion of the error that is "propagated" from layer $l+1$ "back" to layer l, thus the name of the algorithm, *Backpropagation.*

With Equations 1.12 - 1.15 one can implement an ANN with generic neurons and any number of hidden layers. Although both $\Phi(\cdot)$ and $\Phi'(\cdot)$ need to be defined, alongside $\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{y}}$. Note that the algorithm is independent of a specific $J(\boldsymbol{\xi})$. Any cost function can be used as long as its derivative with regard to $\mathbf{y}$ exists. The simplest implementation (Algorithm 1) can be applied to both single input samples or batch learning, being careful to take out the mean gradient between all the used samples.

Algorithm 1: Backpropagation Pseudocode.

```
Input: $\mathbf{x}$: $(N \times S)$ Matrix of input data.
Input: $\mathbf{y}$: $(L \times S)$ Matrix of labels or measurements.
Input: $\eta$: Gradient Descent stepsize.
Input: $\mathbf{W}$: Initial Weights.
Output: $\mathbf{W}$: Trained Weights.
1  $\mathbf{u}_0 \leftarrow \mathbf{x}$                    /* Initial network input assigment */
2  while $\mathbf{W}$ not converged do
3      foreach layer do                              /* Feedforward (0 to $M$) */
4          $\mathbf{z}_{(l)} \leftarrow \Phi(\mathbf{W}\mathbf{u}_{(l)})$
5          $\mathbf{u}_{(l+1)} \leftarrow \mathbf{z}_{(l)}$          /* Layer $(l)$ ouput to $(l+1)$ input */
6      end
7      $\boldsymbol{\delta}_{(M+1)} \leftarrow \frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{y}}$
8      $\mathbf{W}_{(M+1)} \leftarrow \mathbf{I}$
9      foreach backwards_layer do              /* Backpropagation ($M$ to 0) */
10         $\boldsymbol{\delta}_{(l)} \leftarrow \Phi'(\mathbf{u}_{(l)}) \circ \left(\mathbf{W}^T_{(l+1)}\boldsymbol{\delta}_{(l+1)}\right)$
11         $\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{W}_{(l)}} \leftarrow \boldsymbol{\delta}_{(l)}\mathbf{z}^T_{(l-1)}$
12         $\mathbf{W}_{(l)} \leftarrow \mathbf{W}_{(l)} - \eta\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{W}_{(l)}}$                 /* Update Weights */
13     end
14 end
```

Bare bone backpropagation is based on a first degree Gradient Descent, which makes it a flawed algorithm when dealing with highly non-linear complex surfaces (which are more likely the case when dealing with real data). A few useful modifications can be implemented, specially *rmsprop*. It accumulates a moving average of the squared gradient for each weight and uses it to change $\mathbf{W}$ accordingly:

$$\mathbf{R}[k] = \rho\mathbf{R}[k-1] + (1-\rho)\left(\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{W}_{(l)}}[k]\right)^{\circ 2} \tag{1.16}$$

$$\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{W}_{(l)}}[k] = \frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{W}_{(l)}} \oslash \sqrt{\mathbf{R}[k]}, \tag{1.17}$$

where $\oslash$ represents the Hadamard (element-wise) division. The rmsprop algorithm can be achieved including a simple line to track the averaged gradient and modifying how the weights are updated (highlighted in Algorithm 2).

Algorithm 2: Rmsprop Pseudocode.

```
Input: $\mathbf{x}$: $(N \times S)$ Matrix of input data.
Input: $\mathbf{y}$: $(L \times S)$ Matrix of labels or measurements.
Input: $\eta$: Gradient Descent stepsize.
Input: $\rho \in [0,1)$: Exponential decay rate.
Input: $\mathbf{W}$: Initial Weights.
Output: $\mathbf{W}$: Trained Weights.
1  $\mathbf{u}_{(0)} \leftarrow \mathbf{x}$                      /* Initial network input assigment */
2  $\mathbf{R} \leftarrow 0$
3  while $\mathbf{W}$ not converged do
4      foreach layer do                                /* Feedforward (0 to $M$) */
5          $\mathbf{z}_{(l)} \leftarrow \Phi(\mathbf{W}\mathbf{u}_{(l)})$
6          $\mathbf{u}_{(l+1)} \leftarrow \mathbf{z}_{(l)}$          /* Layer $(l)$ ouput to $(l+1)$ input */
7      end
8      $\boldsymbol{\delta}_{(M+1)} \leftarrow \frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{y}}$
9      $\mathbf{W}_{(M+1)} \leftarrow \mathbf{I}$
10     foreach backwards_layer do                /* Backpropagation ($M$ to 0) */
11         $\boldsymbol{\delta}_{(l)} \leftarrow \Phi'(\mathbf{u}_{(l)}) \circ \left(\mathbf{W}^{T}_{(l+1)}\boldsymbol{\delta}_{(l+1)}\right)$
12         $\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{W}_{(l)}} \leftarrow \boldsymbol{\delta}_{(l)}\mathbf{z}^{T}_{(l-1)}$
13         $\mathbf{R}_{(l)} \leftarrow \rho\mathbf{R}_{(l)} + (1-\rho)\left(\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{W}_{(l)}}\right)^{\circ 2}$      /* Moving Average */
14         $\mathbf{W}_{(l)} \leftarrow \mathbf{W}_{(l)} - \eta\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{W}_{(l)}} \oslash \sqrt{\mathbf{R}_{(l)}}$      /* Update Weights */
15     end
16 end
```

Other modifications to the vanilla backpropagation can be useful, like *Nesterov's momentum* [13] and learning rate (η) decay. Also, several other useful optimization algorithms can be used, like Adam [14], Levenberg-Marquardt [15], ADADELTA [16], etc. Adam in particular is a very interesting algorithm, it is a slightly modified version of the rmsprop. Adam estimates both first and second order momentum, while rmsprop estimates only the latter (Equation 1.16), however, instead of using both quantities directly, the algorithm bias-correct both estimates:

$$M[k] = \rho_1 M[k-1] + (1-\rho_1)\left(\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{W}_{(l)}}[k]\right) \tag{1.18}$$

$$R[k] = \rho_2 R[k-1] + (1-\rho_2)\left(\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{W}_{(l)}}[k]\right)^{\circ 2} \tag{1.19}$$

$$\hat{M}[k] = M[k]/(1-\rho_1^k) \tag{1.20}$$

$$\hat{R}[k] = R[k]/(1-\rho_1^k) \tag{1.21}$$

Equations 1.18 and 1.19 represent the moving average for the first and second order moments respectively. That is, both equations represent the First Moment Estimator (FME) and Second Momentum Estimator (SME). Equations 1.20 and 1.21 are bias-corrected counterparts to the FME and SME. Note that Equations 1.19 and 1.16 are the same. The changes with respect to the rmsprop algorithm are highlighted in red (Algorithm 3).

Algorithm 3: Adam Pseudocode. Adapted from [14]

```
Input: $\mathbf{x}$: $(N \times S)$ Matrix of input data.
Input: $\mathbf{y}$: $(L \times S)$ Matrix of labels or measurements.
Input: $\eta$: Gradient Descent stepsize.
Input: $\rho_1,\rho_2 \in [0,1)$: Exponential decay rates.
Input: $\mathbf{W}$: Initial Weights.
Output: $\mathbf{W}$: Trained Weights.
$\mathbf{u}_{(0)} \leftarrow \mathbf{x}$                          /* Initial network input assigment */
$\mathbf{R} \leftarrow 0$
$n \leftarrow 0$
while $\mathbf{W}$ not converged do
    $n \leftarrow n+1$
    foreach layer do                                     /* Feedforward (0 to $M$) */
        $\mathbf{z}_{(l)} \leftarrow \Phi(\mathbf{W}\mathbf{u}_{(l)})$
        $\mathbf{u}_{(l+1)} \leftarrow \mathbf{z}_{(l)}$                  /* Layer $(l)$ ouput to $(l+1)$ input */
    end
    $\boldsymbol{\delta}_{(M+1)} \leftarrow \frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{y}}$
    $\mathbf{W}_{(M+1)} \leftarrow \mathbf{I}$
    foreach backwards_layer do                       /* Backpropagation ($M$ to 0) */
        $\boldsymbol{\delta}_{(l)} \leftarrow \Phi'(\mathbf{u}_{(l)}) \circ \left(\mathbf{W}^T_{(l+1)}\boldsymbol{\delta}_{(l+1)}\right)$
        $\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{W}_{(l)}} \leftarrow \boldsymbol{\delta}_{(l)}\mathbf{z}^T_{(l-1)}$
        $\mathbf{M}_{(l)} \leftarrow \rho_1\mathbf{M}_{(l)} + (1-\rho_1)\left(\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{W}_{(l)}}\right)$                /* FME */
        $\mathbf{R}_{(l)} \leftarrow \rho_2\mathbf{R}_{(l)} + (1-\rho_2)\left(\frac{\partial J(\boldsymbol{\xi})}{\partial \mathbf{W}_{(l)}}\right)^{\circ 2}$              /* SME */
        $\hat{\mathbf{M}}_{(l)} \leftarrow \mathbf{M}_{(l)}/(1-\rho_1^n)$                     /* Bias-Corrected FME */
        $\hat{\mathbf{R}}_{(l)} \leftarrow \mathbf{R}_{(l)}/(1-\rho_2^n)$                     /* Bias-Corrected SME */
        $\mathbf{W}_{(l)} \leftarrow \mathbf{W}_{(l)} - \eta\hat{M}_{(l)} \oslash \left(\sqrt{\hat{R}_{(l)}} + \epsilon\right)$          /* Update Weights */
    end
end
```

1.6 Training a Machine Learning Model

Outlying the model is but the beginning, a ML model without useful and relevant data is worthless. Data can come in varying forms, not only numbers. Take for instance the Adult dataset (Table 1.1) from UCI Machine Learning Repository [6]. Taking only 6 variables from the 15 originally extracted already demonstrates the type variability. This dataset is used to predict whether a person (single record) has an Yearly Income of over 50.000 dollars.

Name	Type
Age	Numerical (integer)
Education	Categorical (string)
Sex	Binary (string)
Marital-Status	Categorical (string)
Hours-Per-Week	Numerical (integer)
Yearly Income	Binary (boolean)

Table 1.1: Partial Data Description and Respective Type From the Adult dataset [6].

As the reader probably noticed, the algorithms previously defined (Section 1.5.2) can only be applied to numerical matrices. The step of taking raw data and preparing it to be "fed" to the model is known as *Preprocessing*.

Preprocessing is a topic on its own and has no right way to be done. Each dataset needs to be treated individually, although a few techniques are somewhat universal. Its important to note that for an ANN, usually $\Phi(\cdot)$ is a sigmoid function (Figure 1.6). These functions have a derivative close to 0 when near the borders ($\|\mathbf{Wx}\| >> 0$). This fading derivative stops the learning process since the weight adaptation directly depends on the gradient.

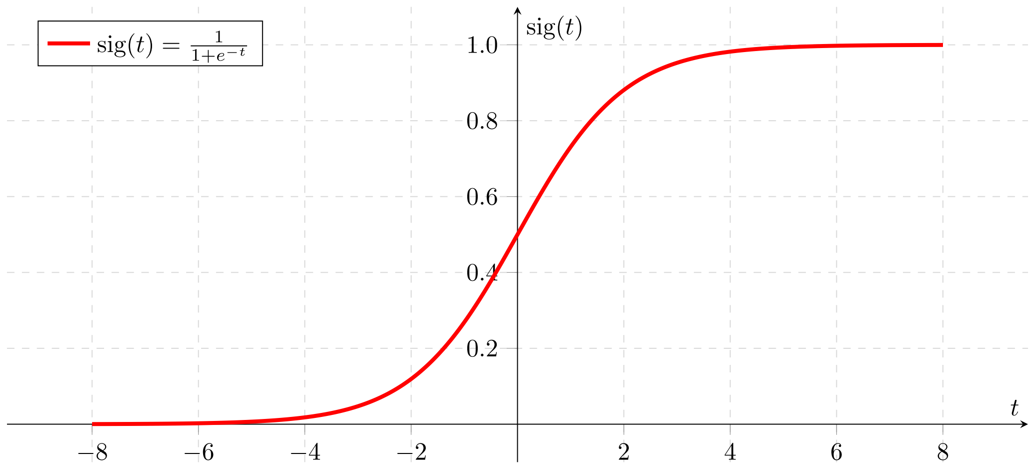


Figure 1.6: Example of a Sigmoid Function. Adapted From [3].

One way to avoid the fading gradient (at least in the beginning of the training process) is to normalize the data to a small range, say $[-1, 1]$. This normalization also makes different variables equally relevant since they can have vastly dissonant dynamic ranges. A good way to normalize data is by the Z-Score normalization (Equation 1.22). It changes the mean ($\bar{x}$) and variance (σ_x) to 0 and 1 respectively:

$$x' = \frac{x - \bar{x}}{\sigma_x} \tag{1.22}$$

Where $\bar{x}$ and σ_x are the mean and standard deviation of the parameter. Note that this is just an exemplary normalization, there are several others that can be equally beneficial (or even more so) when applying to a specific training dataset. This is done on numerical variables, now if one has a categorical input (i.e. Marital-Status from Table 1.1) one must convert it to numbers. The simplest way would be to increasingly enumerate them:

$$\begin{aligned} \text{“Married”} &= 1 \\ \text{“Single”} &= 2 \\ \text{“Divorced”} &= 3 \\ &\vdots \end{aligned} \tag{1.23}$$

The problem with that codification is that the distance between the labels also varies. For instance, the difference between “Divorced“ and “Married” is $3 - 1 = 2$ while “Single“ and “Married” are $2 - 1 = 1$ unit apart. This causes imbalances in the training and must be avoided. In order to make sure all categories have the same importance, a transformation is needed. A good one is the maximally-sparse codification:

$$v_j = \begin{cases} 1, & \text{if } j = C_i \\ -1, & \text{otherwise} \end{cases} \tag{1.24}$$

Where i is the previous categorical notation (Eq. 1.23) and j is the j-th element of the vector. Applying Eq. 1.24 yields:

$$\begin{aligned} \text{“Married}'' &= [1,\ -1,\ -1,\ ...]^T \\ \text{“Single}'' &= [-1,\ 1,\ -1,\ ...]^T \\ \text{“Divorced}'' &= [-1,\ -1,\ 1,\ ...]^T \\ &\vdots \end{aligned} \tag{1.25}$$

This ensures equally spaced categories since their distance is constant. Taking the Euclidean norm on any 2 different labels gives:

$$\|\text{“Married}'' - \text{“Divorced}''\| = \|[2,\ 0,\ -2,\ 0,\ ...]^T\| = 2\sqrt{2} \tag{1.26}$$

Before applying the relevant transformations, one had a matrix $P \times N$, being P the number of chosen parameters plus any and N the amount of observation samples. This categorical transformation adds P_c rows, 1 for each different label.

One needs to be careful, a simple parameter with 10 category adds 10 rows to the input matrix, raising complexity.

However, the model can only learn from collected samples. One can not state that having a small error in training will always imply in a small error deploying the model with new data. In order to guarantee it one must create a new subset from the collected samples, normally called *Test Set*. The test works as “unknown” samples to the model, therefore trying to simulate real environments. Evaluating how the error in the test set progresses as the training goes is of utmost importance. In sum, if the model applied to the test set yields acceptably low error, one has a satisfactory model.

Additionally, one can further split the training set into a *Validation* set. It serves as a monitoring measurement, a Test set that is continuously monitored throughout training. This set is important for stopping the algorithm as soon as a minimum is found, saving computational energy and time. The test set is only used after training has completed. A good way to achieve a balanced split between the sets is using a k-fold Cross-Validation

1.6.1 k-fold Cross-Validation

Cross-Validation (CV) is a technique used to train and validate a ML model. It divides the entire dataset in k complementary divided parts, using 1 as testing and the other $k-1$ as training samples. The samples that are used for testing are cycled throughout the iterations of the CV (Figure 1.7). This cycling ensures that all data is used for evaluating the model. It gives insight on how the model should behave in practice, with unknown data.

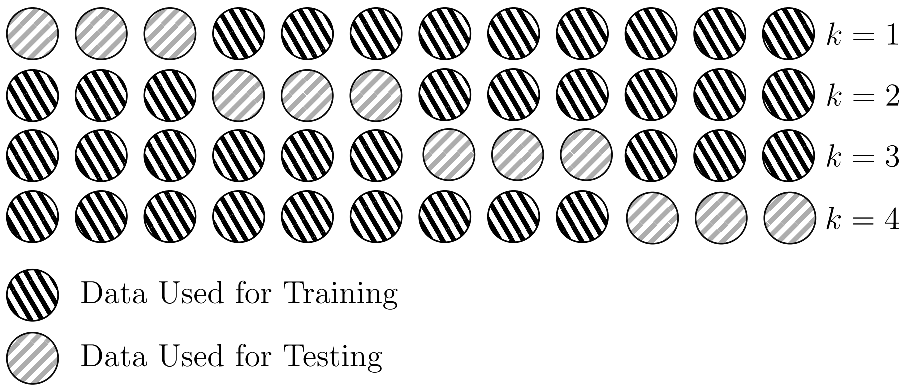


Figure 1.7: Example of Cross Validation with 4 Folds.

The results from the test set for each iteration are combined using both statistical mean and standard deviation. These results are performance measures previously set for the model. Some good metrics for classification include *Cross-Entropy*, *Accuracy* and *Confusion Matrix*.

These metrics help determine if the model is both accurate (error close to zero) and precise (low variance). Cross-Validation is also used to determine the best value for a specific hyperparameter. A good example for that is the evaluation of the best ANN architecture by changing the number of layers in the hidden layer.

1.7 Mechanical Properties

An important fundamental quantity for materials is stress. It is defined as the force across a boundary divided by its area (Equation 1.27), it is essentially pressure and even has the same units, pascals (Pa). Given a simple geometric shaped material and uniform stress distribution (Figure 1.8) one can easily define the stress vector:

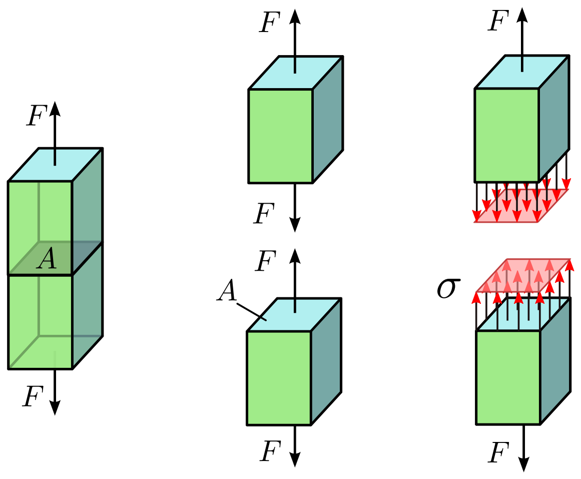


Figure 1.8: Stress Vector Illustration. Taken from [4].

Where F is the load applied to the object, A is the cross-section area and thus σ can be defined as:

$$\sigma = \frac{F}{A} \tag{1.27}$$

However, how stress applies to a more general case, supposing several different loads (with varying orientations) is highly non-linear. Also, the material can have stress intensifiers, like holes or cracks. Such imperfections cause a localized rise of stress due to the concentration of force lines (Figure 1.9) around the crack.

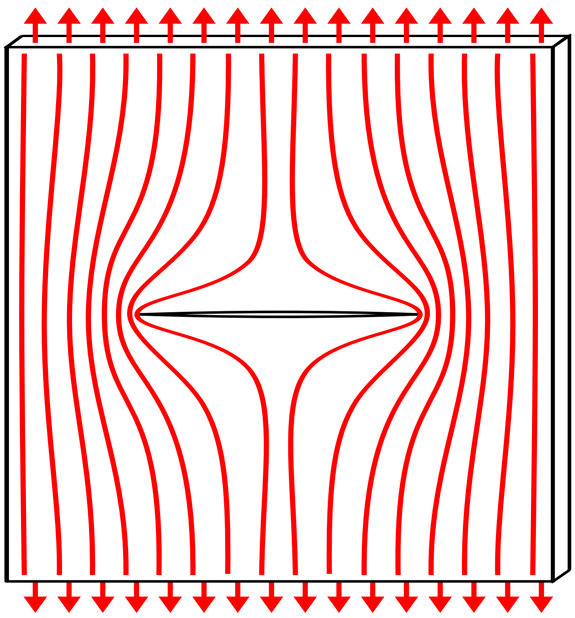


Figure 1.9: Force Lines Within a Cracked Material. Taken from [5].

These stresses (concentrated or not) generate strains, a deformation measurement. Most mechanical properties can be obtained from the stress-strain curve (Figure 1.10) which describes the stress applied to the material (usually with a simple geometric shape) while measuring its displacement. The curve starts with a linear relation between strain and stress, this implicates elastic deformation. When applying increasingly larger loads, the curve starts showing non-linear relationships, this implies plastic (permanent) deformation. The stress reached when the material has 0.2% plastic displacement is called *Yield Strength* (S_y). The maximum stress value is known as *Ultimate Strength* (S_u) and the maximum strain that the material can tolerate before breaking is known as *Fracture Strain* (σ_f). This is the behaviour for a metallic material.

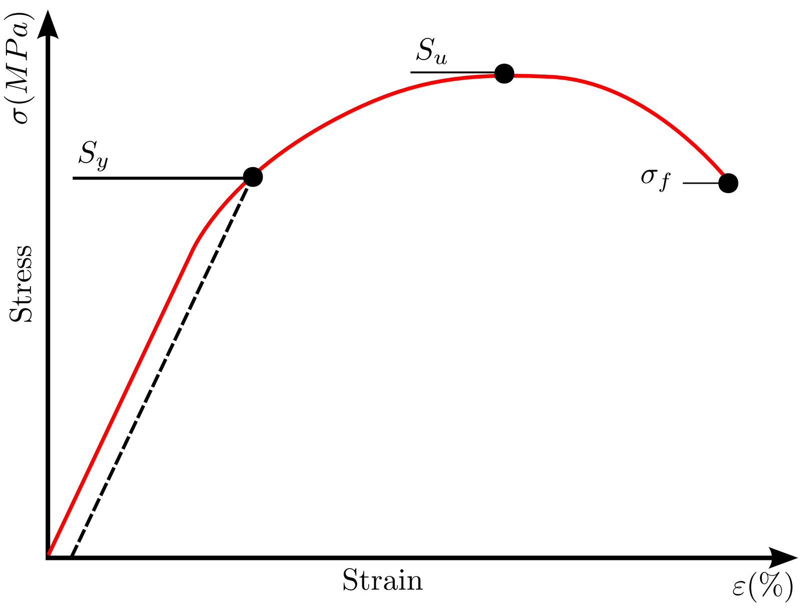


Figure 1.10: A Stress-Strain Curve

It is important to notice that as the material begins to present plastic deformation, less stress is needed to strain the material. Since cracks and other imperfections concentrate stress, it is easy to see how dangerous they are. Applying load to a cracked material can easily make the crack propagate since stress concentrates around it, easily reaching σ_f near the crack.

Therefore inspection and maintenance are of the utmost importance. Identifying the defect without jeopardizing the structure can be done through Non-Destructive Testing (NDT). It is a class of methods that do not result in damage or destruction to the material's structure. Such NDT techniques include for instance liquid penetration (using a coloured liquid to identify cracks), visual testing and acoustic emission (AE).

1.8 Acoustic Emission

Materials with discontinuity have stress risers by definition (Figure 1.9). These discontinuities accumulate significant amount of stress. Once the crack propagates due to external effects (such as increased loading), the accumulated stress energy gets rapidly dissipated, generating acoustic (elastic) waves (Figure 1.11). This is a phenomenon first studied by Joseph Kaiser dating back to post World War II Germany in the 1950s [17].

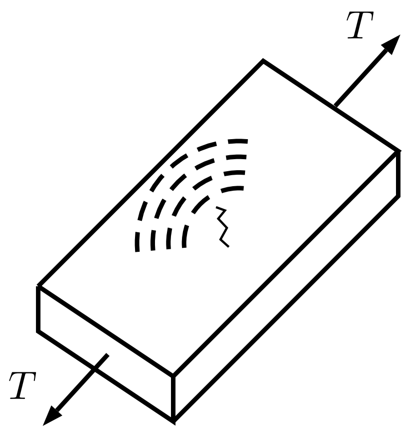


Figure 1.11: A Schematic Tensioned Block With a Crack Radiating Acoustic Emission.

His work shed light on the AE phenomenon and showed its potential as a tool for inspecting and monitoring structural defects [18] [19]. One of the most important findings is the so called *Kaiser Effect* (KE). It describes a pattern of AE from a body under cyclic mechanical stress. If an AE is radiated from a material under certain load, there will be no further emissions until the stress is exceeded.

Since AE comes from irreversible changes, it is reasonable to expect that this phenomenon can help identify different problems, such as crack propagation. Another strong point of AE analysis is that it propagates throughout the material, making it theoretically possible to evaluate the danger from afar. Unfortunately, this thesis will not discuss how distance between the AE source and sensors affects the AE because there is no reliable information regarding the sensors physical disposition. It is known however that it heavily affects the AE signal [20].

1.9 Bibliography Review

Studies on AE and its uses dates all the way back to the 1950s' with Kaiser [17] which highlighted the potential of AE as a tool for inspection. As technology progressed, the classical way of manually analysing and interpreting the AE data became obsolete, and new techniques surfaced.

Several different techniques have been studied on this subject, Sun et al. [21] proposed a classifier based on Self-Organizing Maps (SOM) to help distinguish the crack-related AE from interference and noise. The SOM is first used to extract features by looking at the weight vectors. Once the SOM finishes training, it can be seen which neurons were activated by each class (in this case, real AE or interference). Then a test set is applied to the SOM in order to classify it based on these "defining" neurons.

Huguet et al. [22] also employs a SOM in order to cluster the AE data. However, what differs this from the previous work is that the objective of this paper is to identify damage modes. These damage modes are very similar to the NP, SP and UP classes defined for this thesis. The article defines them as matrix cracking, debonding, pull-out and fibre fracture. The SOM receives the AE parameter data and creates clusters for the classes. It also discusses the important of the threshold parameter for the AE tests.

Da Silva et al. [23] started a chain of researches that culminated in this thesis. The objective of that work was to apply an ANN model capable of analysing weld bead defects. It used the same relevance criterion applied to this work (Section 2.6.4) in order to treat the data and determine the important features. It also implemented a non-linear Principal Component Analysis (PCA) to further clean the data.

Two years later, a subsequent work from Da Silva et al. [24] tried classifying the AE data from a crack inserted on a 3meter closed cylindric steel pipe. A progressive hydrostatic loading was done to steadily increase the internal pressure. After collecting the AE data (in the form of parameters) they calculated a correlation matrix (similar to Figure 2.15) and made a relevance analysis (Section 2.6.4) in order to determine which AE parameters were the most important. The classification was done using 2 classes, No-Propagation and Propagation. The classifier also bears resemblance to this thesis, it was also an ANN, albeit with 7 neurons on its single hidden layer.

The first big change in the landscape came from Pinto et al. [25], [26] which used a similar test object and hydrostatic loading. The main change was with respect to the amount of classes. Instead of separating into two [24], the authors found that it would be wiser to differentiate 3 classes, No Propagation, Stable Propagation and Unstable Propagation, the same classes used in this work. Also used an ANN as the classifier.

Orlando Géa [27] tried to tackle exactly the same aforementioned problem, however, with a different dataset. While Pinto [26] used only the parameters from the AE, Géa [27] focused on the time series waveform from the AE. In both works, it was shown that frequency-domain parameters are highly valuable to the classifier. These works mix both unsupervised and supervised learning methods including SOMs.

The newest work, from Luiza Marnet [28], uses data from the same test as this thesis, however, the author only used AE parameter data. After applying the same method described further ahead (Section 2.6.2), it was shown that there are 4 classes instead of 3. A semi-supervised (both supervised and unsupervised) method using an ANN and SOM was implemented with good results.

Chapter 2

Materials and Methods

This chapter focus on describing the used materials and methods utilized in this thesis. The acoustic emission tests (Section 2.1) had 3 separate data acquisition systems, one using an industrial device (DISP-16C) made by Physical Acoustics (PAC), one with another industrial device (AMSY-5) made by Vallen Systems and a custom one denominated Streaming.

Both Streaming and Vallen data were analysed throughout the project duration, however, this thesis concentrates solely on the Streaming data. Both are similar in the way that they acquire temporal data from the AE (not its parameters), however, the AE waveforms captured by the Vallen system had several disadvantages when compared to the Streaming counterpart.

The Vallen data is a collection of fixed length AEs concatenated to form a $L \times N$ matrix where L is the AE length and N the number of captured AEs. This matrix was provided as a MATLAB formatted data file (.MAT) containing the waveform using 64-bit double-precision floating-point format. Unfortunately, this system is not guaranteed to capture all waveforms, this severely hinders some essential preprocessing stages (Sections 2.3.2 and 2.3.3), making it a rather unreliable (the captured AE may not be from the crack propagation) and with no means of improving its reliability, therefore all of the Vallen data was discarded.

Thus, this chapter begins detailing the destructive test done, extends to the raw data format used, describes all the preprocessing done and the reasoning behind it, then details the waveform capture procedure all the way to creating the final dataset used to train a neural network model (Section 1.4) with parameters from both the AE temporal data and its frequency spectrum.

2.1 Acoustic Emission Test

The AE tests were performed by engineers of the Physical Metallurgy Laboratory (LAMEF) at the Federal University of Rio Grande do Sul (UFRGS). It used a close-ended steel (API XL 60 series) pipe with 20 inch diameter, 40 metre length and 1.45 centimetres of thickness. The pipe has a protective rubber cape (normally used when in operation). A semi-elliptical pre-crack extending until half of its thickness (approximately $0.7cm$) was inserted at half its length.

An array of sensors was then disposed on top of the naked steel and the rubber coating (Figures 2.1 and 2.2) throughout the tube's length. The crack was surrounded by measuring devices, either strain gauges or Time-Of-Flight Diffraction (TOFD) [29], [30] sensors. These devices are used to determine the depth of the crack.

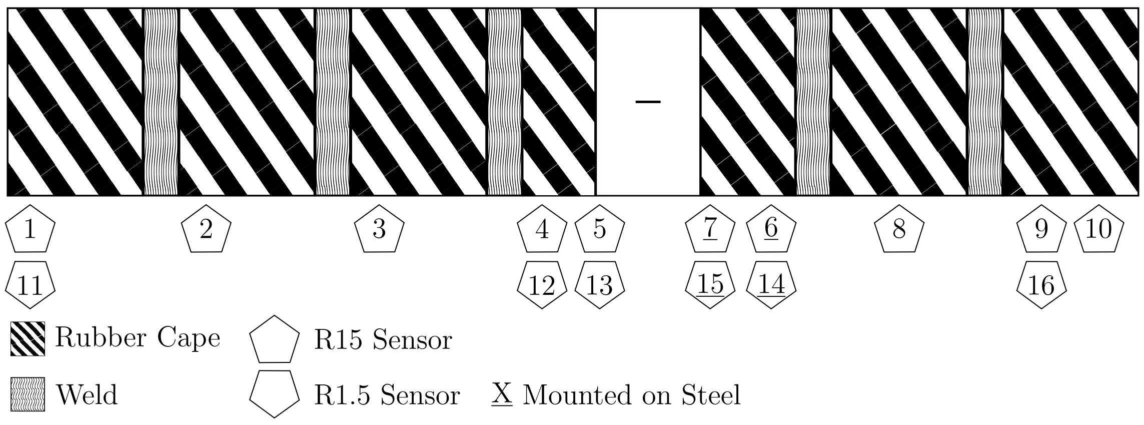


Figure 2.1: Sensor Array Disposition for Test CP2.

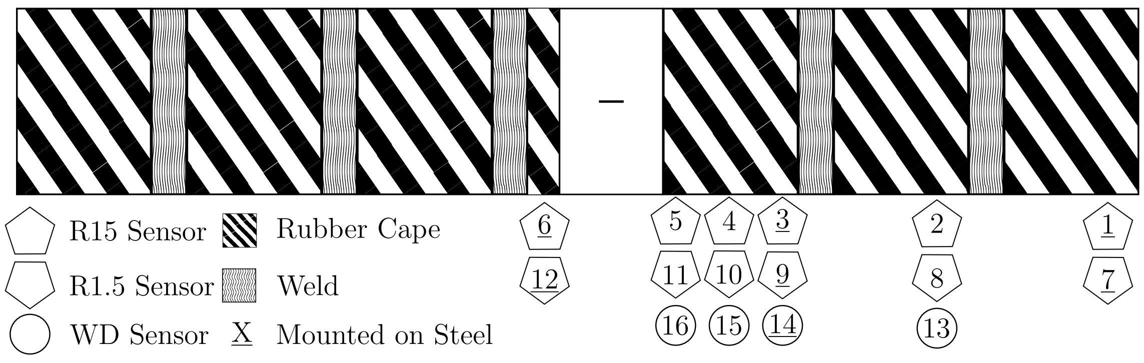


Figure 2.2: Sensor Array Disposition for Test CP3.

Three different streaming sensors (Figure 2.1) were used, the R1.5 [31], R15 [32] and WD [33] sensors. Their inner workings and configuration (including conditioning circuits) will not be discussed in this work. It is important to note however, that they work on different frequency ranges (Table 2.1) and naturally have diverse responses.

Table 2.1: Streaming Sensors Frequency Range.

Sensor	Frequency Range (kHz)
R1.5	5 - 20
R15	50 - 400
WD	100 - 900

These sensors were then sampled using a (not specified) 16 bit Analogue-to-Digital Converter (ADC) at $2.5GHz$ during the whole test. The contiguous data was then put into special formatted files created by National Instruments (NI), the Technical Data Management Streaming (TDMS) file [34].

After fixing the sensor array, the pipe is filled with water, thus starting the hydrostatic test. This consists of two periodical sections, a gradual increase followed by a static plateau. Both of equal duration, approximately 10 minutes. These two sections are repeated until the maximum test pressure P_t is reached and kept for (at most) the same amount of time, 10 minutes (Figures 2.3, 2.4 and 2.5). This regular increase in pressure until burst is denominated *cycle*. In total, 4 (four) tests were done, with the first one discarded since it did not burst. These were denominated CP1, CP2, CP3 and CP4 respectively. Unfortunately, there is no information about the sensor placement for test CP4.

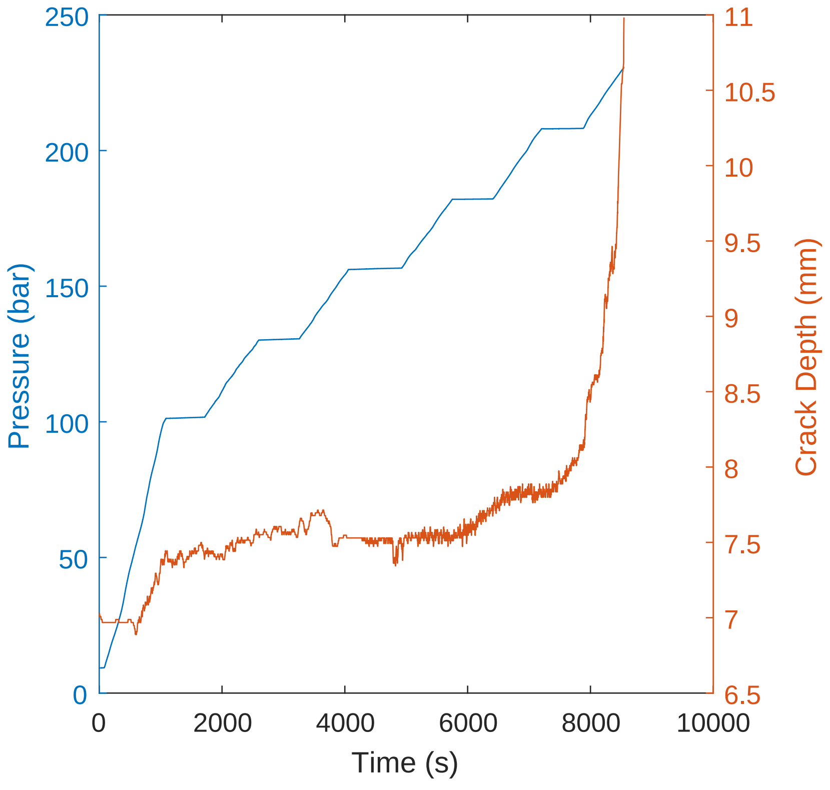


Figure 2.3: Pressure and Crack Dimension for Test CP2.

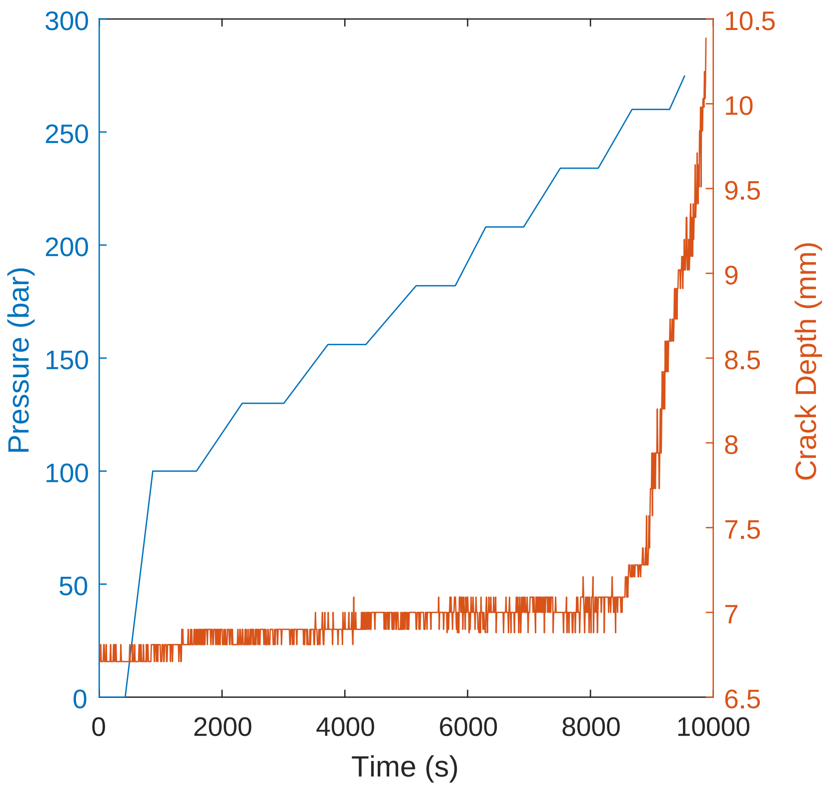


Figure 2.4: Pressure and Crack Dimension for Test CP3.

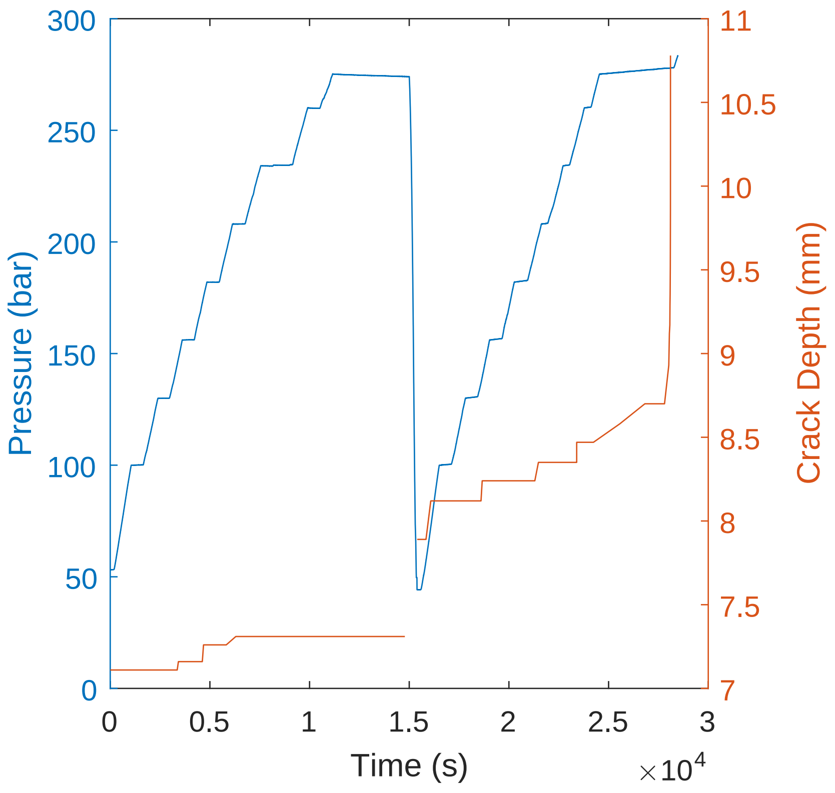


Figure 2.5: Pressure and Crack Dimension for Test CP4.

Strange readings such as noise, (Figure 2.4), plateaus and linear behaviour (Figure 2.5) for the crack development are to be expected since its width is measured by an optical feedback from the TOFD sensors. An image is relayed to the engineers operating the test and is used to determine depth of the discontinuity.

The most important aspect of the graphs is the way the crack propagates according to the rise (or lack thereof). If the pressure rises and the crack is still at the same depth, one classifies this behaviour as No Propagation. As the crack follows the pressure, rising alongside it, it is known as Stable Propagation and if the crack propagates without a rise in pressure, it behaves explosively and is denominated Unstable Propagation.

This is the way the classes were previously separated and its done so by specialists and in previous works [27], [26].

2.2 Streaming Raw Data

Normally, these files can be opened using some specific NI software like *DIAdem* but the LAMEF engineers made slight modifications (Figure 2.6) when saving the ADC data, using a 32*bit* word (normally holding one sample) made of two concatenated 16*bit* integers (that came from the ADC), thus effectively doubling the amount of data stored without requiring additional space.

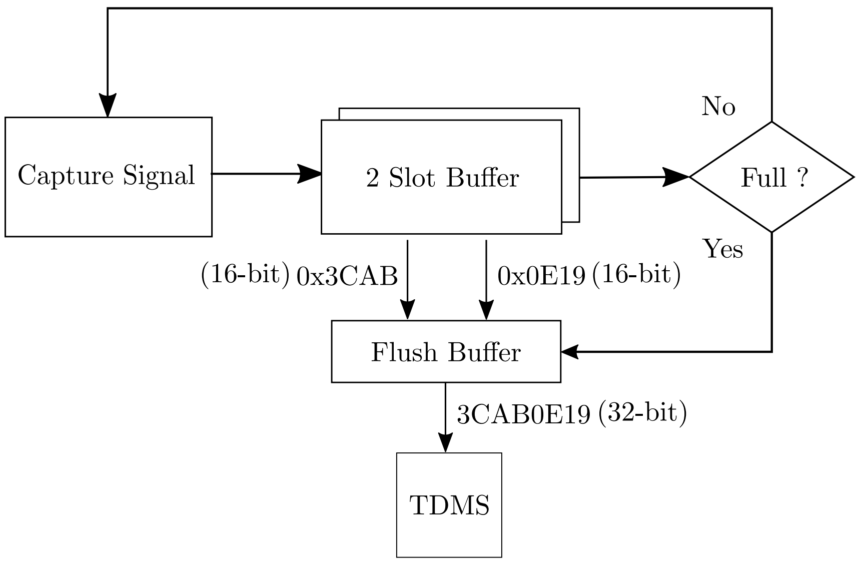


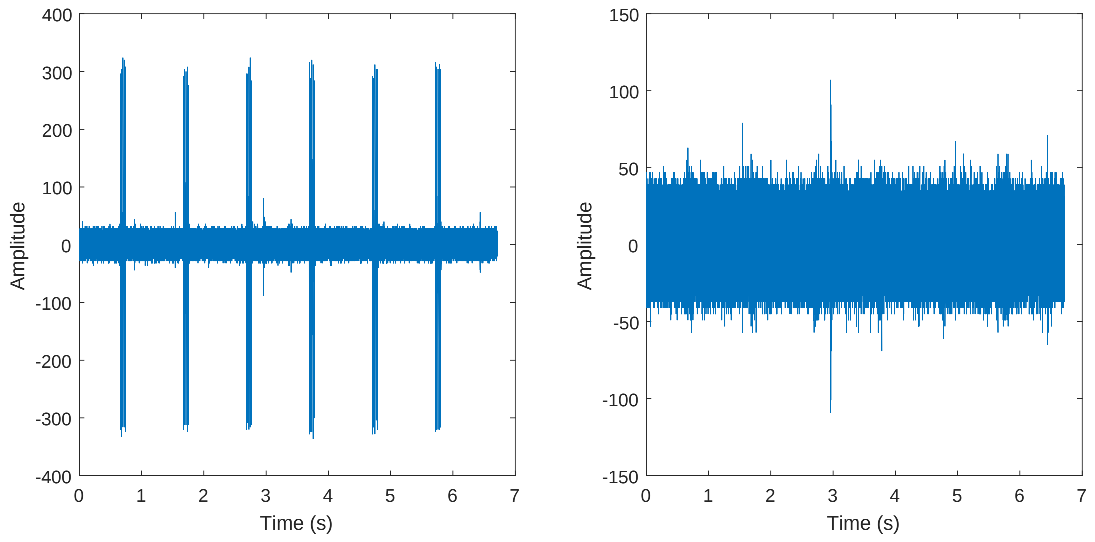
Figure 2.6: Schematic of The Low-Level Modifications Done to Signal Acquisition. With Two Example 16-bit Words (0x3CAB and 0x0E19).

In order to read the TDMS files, LAMEF provided a special compiled LABView routine that transforms each TDMS file to a binary one and a MATLAB script that loads the file to memory. Out of all the tests, only the first one (CP1) did not burst, even after 5 filling cycles, since identifying the burst is essential (Section 1.2) all CP1 data was discarded and it will not be discussed any further.

A single TDMS file translates to roughly 6.7 seconds (Figure 2.7), containing 2^{24} samples taken from its 16 different sensors (channels) leading to a $2^{24} \times 16$ integer (16-bit) matrix. In sum (Table 2.2), all data came from (eventually) ruptured ducts and theoretically contain useful data.

Table 2.2: Hydrostatic Test Summary

Test	Date	Pre-Crack Depth (millimetres)	Rupture Cycle	Pressure at Rupture (bar)	Data Volume (Gigabytes)	Files
CP1	18/03/2015	5.5	X	-	3900	X
CP2	28/07/2015	7.9	1	230	900	1513
CP3	06/11/2015	6.6	1	264	900	1500
CP4	23/05/2016	7.0	2	283	1500	4261



(a) TDMS File 900, Channel 12. (b) TDMS File 900, Channel 7.

Figure 2.7: Data from TDMS File 900, Both Channel 12 (a) and 7 (b) From Test CP3 with Zero (0) Mean.

2.3 Preprocessing

The preprocessing was likely the lengthiest part of this thesis, it is mainly divided in 3 big blocks, an initial resolution analysis (Section 2.3.1) to determine which files contain useful information, a TOFD signal removal (Section 2.3.2) and a final identification and cleaning of the pressure pump AE signal (Section 2.3.3).

These three stages are invaluable for treating the enormous amount of data ($\approx$ 1TB per test) and coming up with a reliable and efficient way to extract AE from the propagating crack (Section 2.4).

2.3.1 Resolution Analysis

Upon first inspection, not all files seemed to contain relevant data (Figure 2.7a), most of them contained only a noise bar (Figure 2.7b) thus creating the first problem, determining which files (and channels) contained relevant information so that time would not be wasted trying to find AE waves immersed on noise.

One simple way to determine digital signal quality is through its resolution, signals that are encoded using 10 bits (therefore 1024 digital levels) should have more valuable information than signals encoded with 2 bits (therefore 4 digital levels). Using that as a base, it was heuristically stipulated that 8 bits would be the bare minimum (since it gives approximately 1% of the ADC's dynamic range).

The count of digital levels and effective bits used to encode each file and channel was determined (Figure 2.8) and only those that had a minimum of 8 bits were considered good enough to be further inspected.

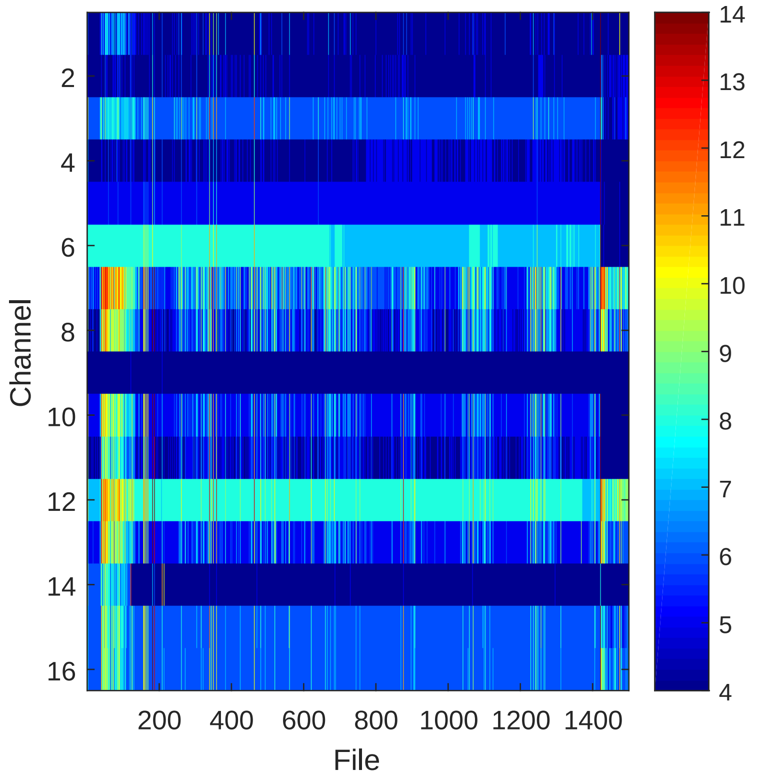
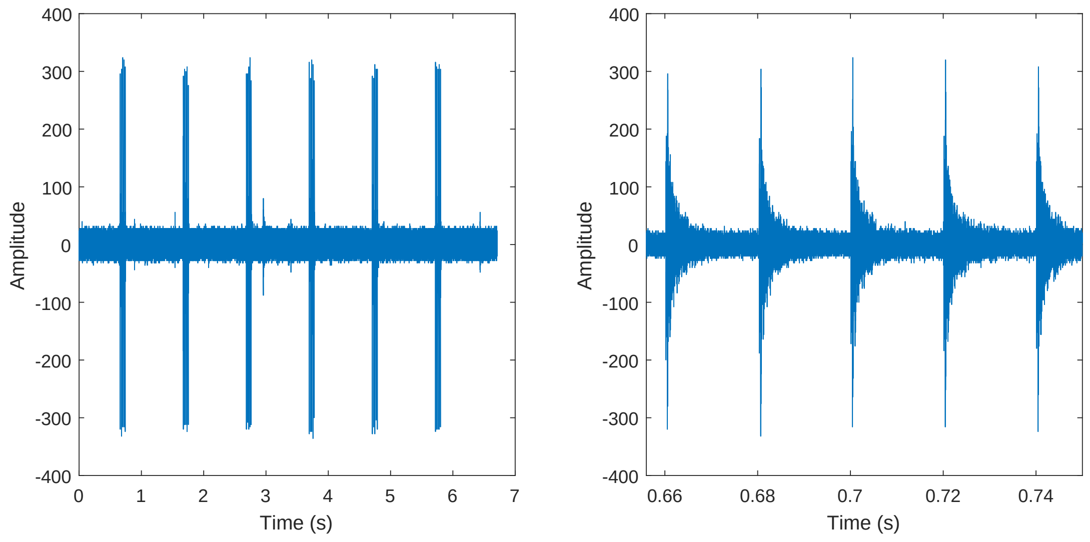


Figure 2.8: Bit Resolution Colormap for Test CP3.

2.3.2 TOFD Removal

TOFD was not a concern in previous works (Section 1.9) since it was not evident (even though it can be seen analysing the parameter data). This however, is not true when working with Streaming data, the TOFD signal is clear and has a well defined period and also comes in blocks (Figure 2.9).

Each block consist of 5 AEs separated by $20ms$ (Figure 2.9b) and each block is 1 second away from each other (Figure 2.9a). These well defined periods can then be used to completely remove all TOFD signals from the entire test.



(a) TDMS File 900, Channel 12.

(b) TDMS File 900, Channel 12 Focused on a Single TOFD Block

Figure 2.9: Data from TDMS File 900, Highlighting the TOFD Characteristics.

To remove those, it is necessary to first identify it, without going into much detail (those are in Section 2.4) a threshold level is defined for each file/channel combination and each point is compared to that threshold, if its amplitude surpasses it, it receives a 1 (TRUE) and 0 (FALSE) if not (Figure 2.10).

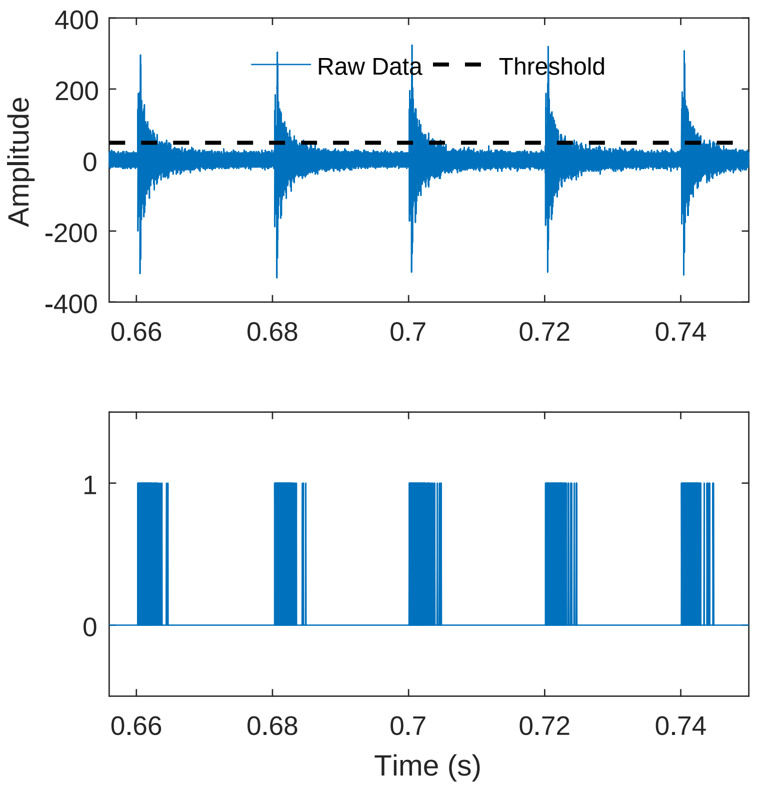


Figure 2.10: Data from TDMS File 900, Channel 12 Containing the Threshold and Indexes that Respectively Surpass It.

Once the beginning of each TOFD wave is found, a simple check is done to verify if they are separated by $20ms + -10\%$. If so, their indexes are saved and the waves are thus identified. In order to allow a finer control over the TOFD block removal, an additional number of samples is taken from both before and after each wave (Figure 2.11). A total of 50000 samples (25000 for each side) were chosen to guarantee the entire block removal, the downside is that any AE that happens to fall inside a TOFD block is also lost, however, considering that each block takes $100ms$ each second, that equates to 10% of data being discarded, which is quite acceptable.

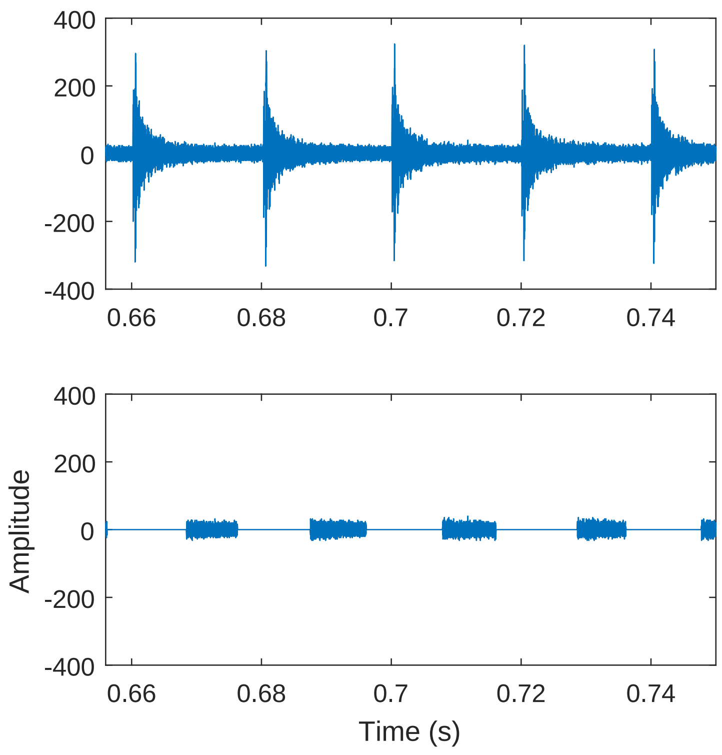


Figure 2.11: Removal of Time-Of-Flight-Diffraction Sensor Signal.

2.3.3 Pressure Pump Removal

Noise from the pressure pump was also not considered in previous works (Section 1.9) and it is an indispensable part of the test, used to fill and pressure the duct. This was discovered by accident when, by analysing multiple files at once, a huge unknown (and somewhat periodic) signal appeared (Figure 2.12).

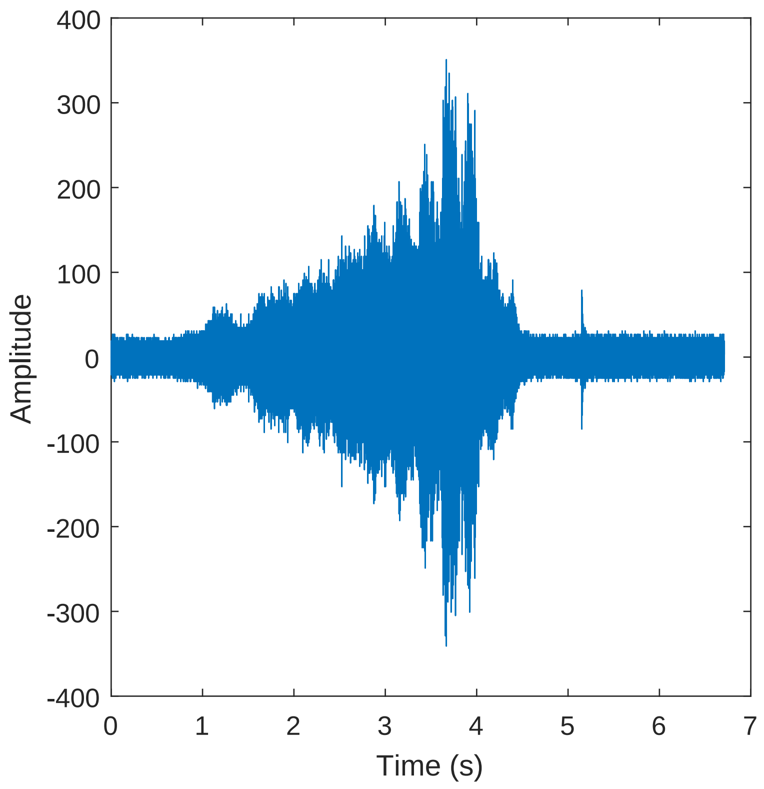


Figure 2.12: Data From TDMS File 751, Channel 7.

Once crossing the files in which that signal appeared with the logs provided by the LAMEF engineers, it was possible to infer that these signals only appeared at the beginning and end of pressure rising, thus concluding that it is indeed a AE signal generated by the bomb and not the crack propagating.

Its removal was done manually, all 7000 more files were checked 10 at a time (a total of more than 700 different plots) and all files that had even a portion of the bomb signal were discarded.

2.4 Wave Capture

Both Section 2.3 and the current one contain all that was done in order to transform over 3 Terabytes of noise filled data into a usable, clean and expandable database containing useful acoustic emission data.

It begins by defining a *noise level* for each file/channel combination and using this information to create a floating threshold level that works similar to that described in Section 2.3.2, once that threshold is exceed (both positive or negatively) the beginning of a "hit" is defined, and specific timing parameters (Section 2.4.2) are used to determine its end, thus extracting individual AE waves.

For each of those waves an array of time-domain parameters (Section 2.4.3) is calculated alongside its power spectrum (Section 2.4.4). Both are then processed (Section 2.6) again to be used as input for the neural network.

2.4.1 Estimating Noise Level

Normally, AE tests use industrial equipment with a fixed Threshold (a level that when exceeded triggers the wave capture) which may or may not be changed throughout the test by its technician. However, instead of trying to define a fixed level for each channel and file, a fluctuating limit was created based on the level of noise for each file/channel. This is interesting since the Signal-to-Noise Ratio (SNR) can (and does) change a lot because of the varying amplitude of the AE signals. Using an adaptive limit based on the noise level itself is an attempt to stabilize the SNR for each captured wave.

It is based on a simple metric of standard deviation (Equation 2.1) applied to each file and channel. In other words, suppose the data from one channel, a $2^{24} \times 1$ array, knowing that AE is a relatively rare event and that there exists a bar of white noise (Figure 2.7), it is safe to say that this array is mostly composed of Gaussian white noise.

$$\sigma_X = \sqrt{E[(X - \mu_X)^2]} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_X)^2} \tag{2.1}$$

Where X is the array of size $N = 2^{24}$, σ_X is its standard deviation, μ_X its mean and x_i the i-th array element. Considering that this is composed mostly of Gaussian white noise, it is possible to state:

$$Noise_X \approx 3\sigma_X \tag{2.2}$$

This provides a noise level estimate (Figure 2.13) that is then (heuristically) amplified by a factor of 3 to create a fluctuating threshold able to isolate acoustic emissions.

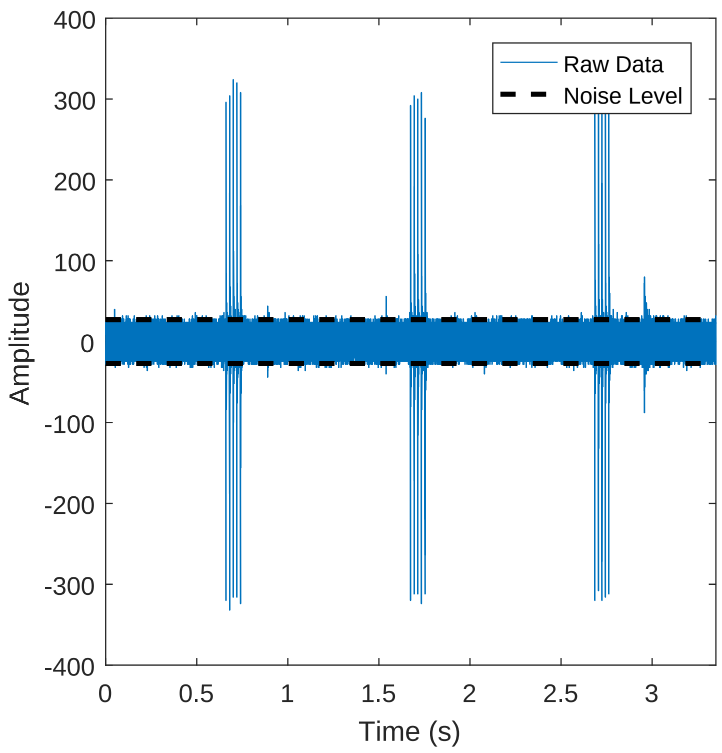


Figure 2.13: Partial Data From TDMS File 900, Channel 7 Highlighting the Noise Level

2.4.2 Timing Parameters

Defining a threshold is the first step in completely extracting AE information from the data. The DISP16C (used on professional AE tests) user's manual [35] specifies 3 timing parameters used to capture AE, these are the Hit Definition Time (HDT), Hit Lockout Time (HLT) and Peak Definition Time (PDT). However, PDT is used to determine which peak is the highest (important for calculating AE parameters) while capturing the wave at real time. Since this work uses static data (recorded from the streaming), determining the highest peak makes no sense because there is only one highest peak for each hit (the maximum amplitude), therefore its definition will not be covered in this thesis, if needed, the reader can find it at [35].

HDT is the most important parameter and is used to define and fix the end of the "hit". Suppose a signal that exceeds the threshold at a certain time t_0, once HDT seconds has elapsed while the signal had no threshold crossings (stayed within the threshold limits) a hit and its end are defined. If one sets this parameter too high, adjacent events may appear as a single hit and if set too low, a single AE may be separated into multiple hits.

HLT can be interpreted as a "dead time" after the definition of a hit, it is a slot of time where no hit can begin even if the signal exceeds the threshold. It is used to eliminate lower amplitude echoes that can occur from the AE's reflection throughout the system. It used to have more importance in the 1980's when computational power was rather limited [35].

The tuning of these hyper-parameters is not studied in this thesis since the script made to capture the waves takes around 8 hours for each CP, so it would be extremely time-consuming and deviate from the main objective of this work, although the author would highly recommend this analysis. Those parameters are then defined as suggested by the LAMEF engineers:

- HDT: $1000\mu s$
- HLT: $800\mu s$
- PDT: N/A

This concludes all necessary steps to extract clean acoustic emission waves and without external interference (at least of what was able to be seen from these tests).

In sum (Figure 2.14) it begins by selecting the important files, ones that have a higher change of containing useful AE by checking the overall resolution per channel. For each file, TOFD signals are identified and properly removed (Figure 2.11) by exploiting their periodicity. Files containing noise from the pressure pump (Figure 2.12) are manually inspected and discarded. From the remaining valid ones, a white noise level estimate is calculated and used to create a "Variable Threshold". This Variable Threshold is then used to identify AE according to the instructions from [35], using HDT and HLT as previously stated.

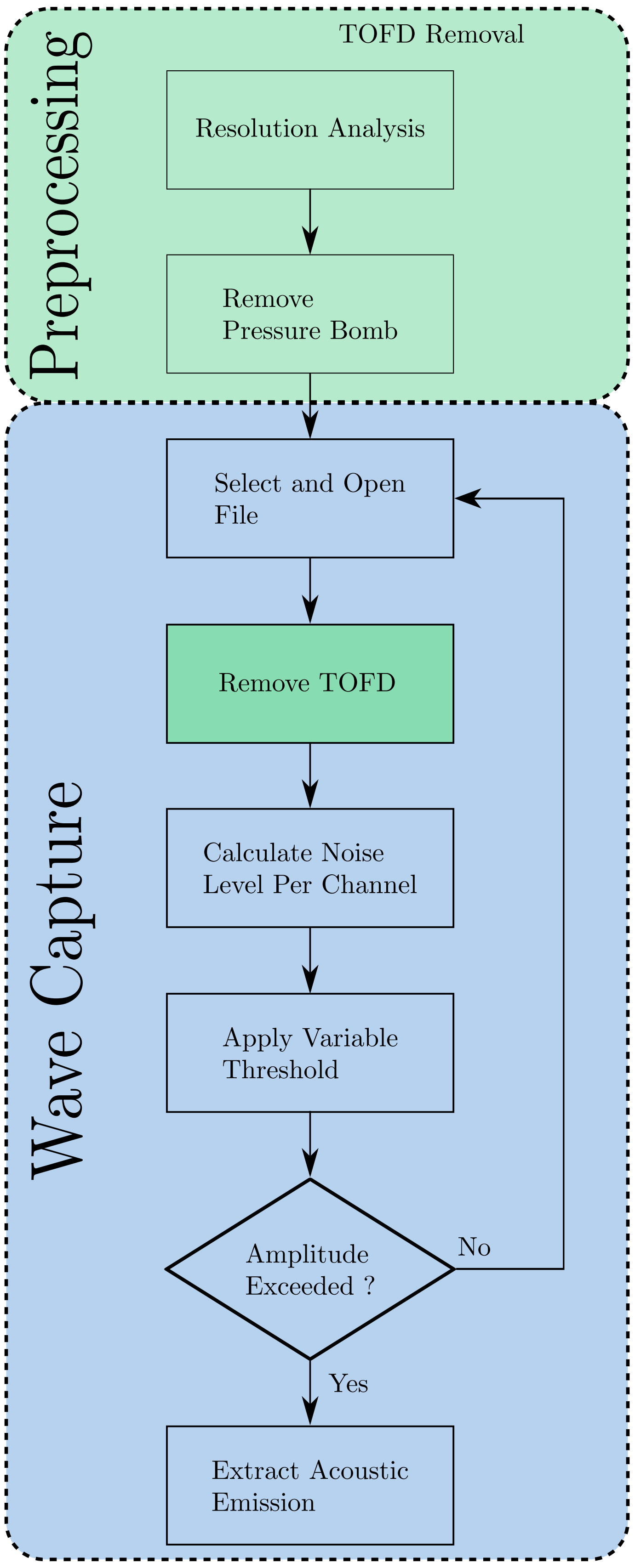


Figure 2.14: Summary Streaming Data Treatment Workflow.

2.4.3 Acoustic Emission Parameters

For each captured wave (Section 2.4) an array of parameters is extracted (Table 2.3) based on [35]. In total, 10 time domain parameters were extracted, 5 less than the standard 14 used in [28]. This is because not all parameters had their exact formulae written on the manual, some only had a brief description, other had certain unknown related variables (for example, software pre-amplifier configured on a test-by-test basis). There are 9 "classical" parameters, and one developed for this thesis, the *Variable Threshold* (VTHR).

Table 2.3: Extracted Acoustic Emission Parameters.

Parameter (P) [Unit]	Description	Definition
Rise Time (RT) [s]	Amount of time between the Hit start and the peak amplitude	-
Count (C)	Number of times the acoustic emission exceeded the Variable Threshold.	-
Energy (E)	Classical energy of the acoustic emission.	$\int_{t_0}^{t_f} Hit(t)^2 dt$
Duration (D) [s]	Duration of the acoustic emission.	$t_f - t_0$
RMS (RMS)	Acoustic emission Root Mean Squared value.	$\sqrt{\frac{1}{(t_f - t_0)} \int_{t_0}^{t_f} Hit(t)^2 dt}$
Max Amplitude (A)	Peak amplitude of the acoustic emission.	max(Hit(t))
Max Amplitude dB (A_{dB}) [dB]	Peak amplitude of the acoustic emission (in dB).	$10log(A)$
Mean Amplitude (MA)	Mean amplitude of the rectified signal.	$\frac{1}{(t_f - t_0)} \int_{t_0}^{t_f} \lvert Hit(t) \rvert dt$
Resolution Level (RES)	Count of digital levels (resolution) used to digitalize the Hit.	-
Average Signal Level (ASL) [dB]	Amplitude (in dB) moving average.	-
Count-To-Peak (CTP)	Number of times the acoustic emission exceeded the Variable Threshold before the Maximum Amplitude is reached.	-
Average Frequency (AF)	A rough frequency estimation Count and Duration. using	$\frac{C}{D}$
Reverberation Frequency (RF)	-	$\frac{(C-CTP)}{(D-RT)}$
Variable Threshold (VTHR)	Variable used to determine the beginning of the Hit.	$3 \times NoiseLevel$ (Section 2.4.1).

2.4.4 Frequency Data

An important set of frequency domain parameters were listed as relevant by other related works [28] [27] [26], called FREQ-PP1, FREQ-PP2, FREQ-PP3 and FREQ-PP4. They are the mean relative power for each respective frequency range (Equation 2.3), according to [35], these ranges are manually configured before testing. In this thesis, a technique based on linear correlation was developed to estimate which frequency slots would be the most important at classifying the AE derived from the crack.

$$FREQ-PPX = \frac{\int_{F_X} |H(f)|^2 df}{\int_0^{Fs/2} |H(f)|^2 df} \tag{2.3}$$

Where F_X is the frequency range, Fs is the sampling frequency, $H(f)$ is the Fourier Transform of the wave, and $|\cdot|$ its magnitude.

This technique consists of a visual analysis of a correlation plot (Figure 2.15) between each calculated frequency for each wave and their respective target class output (binary function). In other words, it is a correlation between input and output where a single frequency point (for all captured waves) is used as input, repeating that for each frequency we can create a correlation graph for all output classes, NP, SP and UP. Important to note that the highest possible frequency is $1.5MHz$, more than 3 times the maximum frequency present in the graph. This is due to highly uncorrelated noise on the higher frequencies thus giving no relevant information (and therefore discarded).

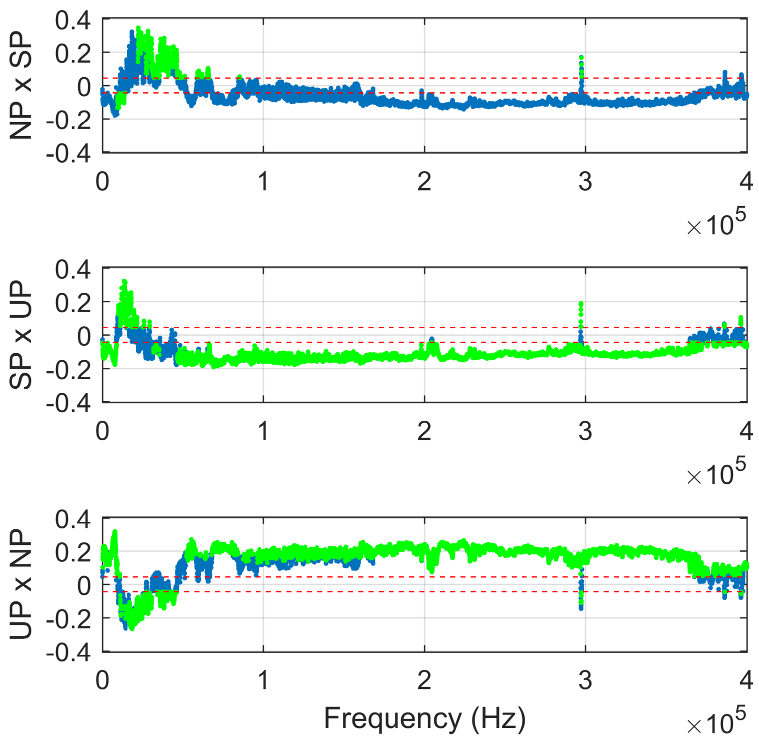


Figure 2.15: Correlation Plot Between Each Frequency and The Output Classes, No-Propagation, Stable Propagation and Unstable Propagation for Test CP3.

This plot highlights which frequency points have distinct numerical signals (positive or negative) for each pair of output classes, meaning that the highlighted (green) points indicate a positive correlation to one class and negative correlation to another (both classes are indicated by the y-axis label).

To further clarify this explanation, take for instance the second graph (Figure 2.15), all points (blue and green) are from the correlation between each frequency and the output class SP, the green points then represent which frequencies that have a different numerical sign when compared to the UP class. The frequency ranges (F_X) are then defined by visually clustering these points.

2.5 Database Structure

After extracting the available AE waveforms, calculating a lengthy array of parameters, a simple database was consolidated. It is composed of 3 MATLAB files (one for each test) and contain a plethora of variables, their design was object-oriented and consists mainly of 3 classes (Appendix A), *StreamingClass*, *Wave* and *Streaming Model*.

Each *StreamingClass* object has an array of *Waves* that is dynamically allocated when identifying the AE waves (Section 2.4), however, in order not to add another considerable amount of time to the already lengthy process of extracting AE from the data, the captured waves parameters are only calculated after all files are processed.

Each *Wave* holds all calculated parameters (Section 2.4.3) and also some important meta-data like absolute time regarding the whole test, the relative trigger index from the file, etc. This allows for different sized waves to be stored alongside their parameters, creating a more generic approach from both PAC and VALLEN alternatives, since it can not only store parameters, but also variable-length waveforms (VALLEN only stores waves with the same length).

2.6 Model Definition

The proposed model for this thesis is an ANN (Section 1.4), since this is a extremely non-linear phenomenon and most likely requires a robust model with a good generalization capability [36]. The parameters chosen for the ANN were:

- Batch Size: 20
- Learning Rate (η): 0.1
- Patience: 50
- Maximum number of epochs: 1000

- Training Algorithm: Adam (Algorithm 3)
- Loss Function: Categorical Cross Entropy

Batch size is the amount of samples to be used per iteration (epoch) of the algorithm. Patience is the maximum amount of iterations that the algorithm will run while the validation-set error does not improve.

The subsequent sections detail how the number of neurons and their disposition was chosen (Section 2.6.1), identification of the separation indexes that best divide the three classes (Section 2.6.2) and the final definition of the important inputs to feed the ANN (Sections 2.6.3 and 2.6.4).

All analyses were done for each test (Section 3.1.3) to determine if the relevant inputs are immutable, as they should (supposing the tests were done with the same sensors).

2.6.1 Network Size

In order to define the network size, 20 different networks were trained and compared using cross validation (Section 1.6.1) and the classification rate for each class (Figure 2.16). The best network is chosen weighting both computational complexity and classification performance, in this case, accuracy.

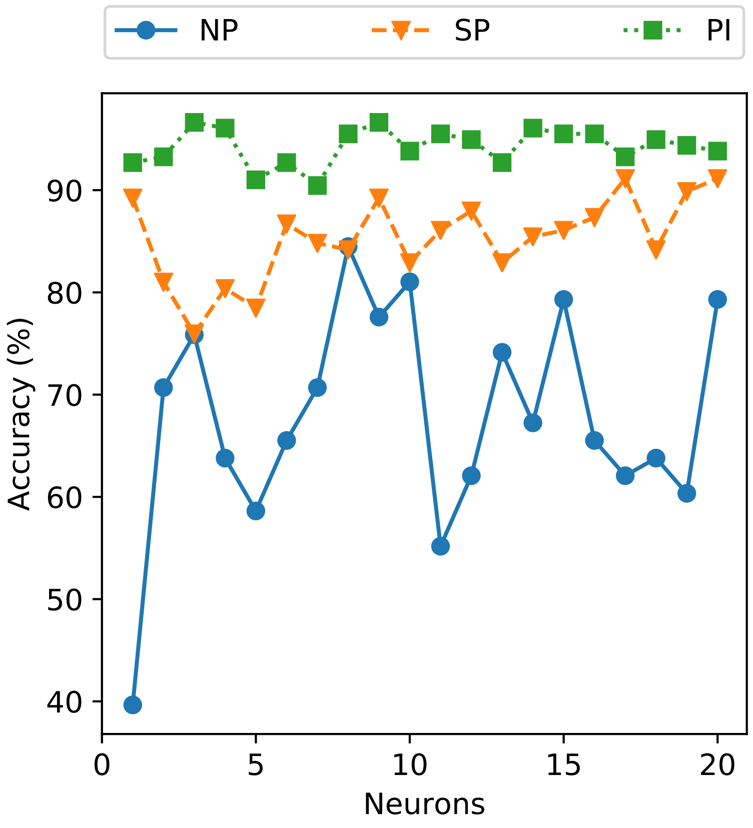


Figure 2.16: Network Size Investigation for Test CP3

2.6.2 Transition Time Estimation

Transition Time Estimation (TTE) is a technique idealized by Francesco Noseda that helps define the point that best separates two sequential different patterns (distributions) which is exactly the case for this work since the target classes NP, SP and UP (in principle) happen one after another.

It was previously implemented on a related past work [28] with relatively good success. As studies about it are still in its first steps, there is no theoretical proof for it yet, although an attempt was made by the author (Appendix B).

The idea behind it is simple, suppose two linearly-separable (LS) distributions, f and g, and a transition time n_t. The unknown phenomenon $\Gamma[n]$ behaves like f before t_T and g after:

$$\Gamma[n] = \gamma = \begin{cases} f, & \text{if } n < n_t \\ g, & \text{if } n \geq n_t \end{cases} \tag{2.4}$$

Now, one takes $P/2$ points randomly from each distribution. In order to determine n_t, one slides a window of length L across the P -point dataset (Figure 2.17). This window labels (to the model) both halves differently. In order words, $L/2$ points are labelled "Class 1"(red) and "Class 2" (blue). Eventually, as the window reaches the end ($n = n_f$) almost all samples have already been labelled as both classes at different times (thus the blue-to-red gradient).

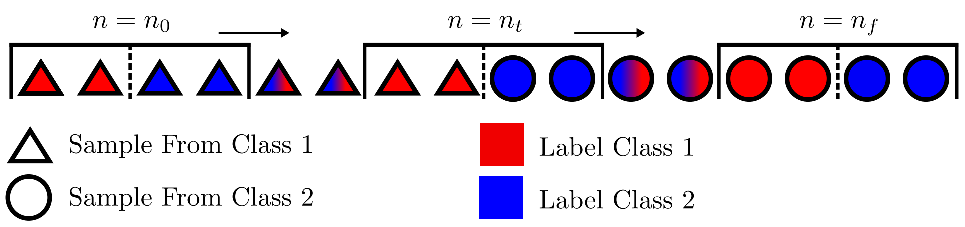


Figure 2.17: Example of a Sliding Window With Length $L = 4$ for the Transition Time Estimation Method.

The important variable to observe is the classification error ($E_C[n]$). E_C is a stochastic process (Appendix B). For each n, there is an expected error and an associated variance σ^2. However, not much can be inferred since they depend on the data distribution and the model used. However, one thing can be assured, while the window contains only the samples from one class (distribution), the probability of the error being zero (LS samples) is bound by a function of the window length L (Appendix B):

$$P_{\max}(E[n] = 0) = 1 - \frac{1}{2^{L-1}} \; \forall n \mid C([n - L/2, n + L/2]) = C_i \tag{2.5}$$

Where C_i is the i-th class, which can be either 1 or 2. As the window slides across the samples, one should reach a classification error shaped much like a funnel (Figure 2.18). This "error-funnel" has a few interesting properties (proof on Appendix B), given that the classifier is guaranteed to separate two LS sets, the minimum error $E_{\min}$ is exactly 0.

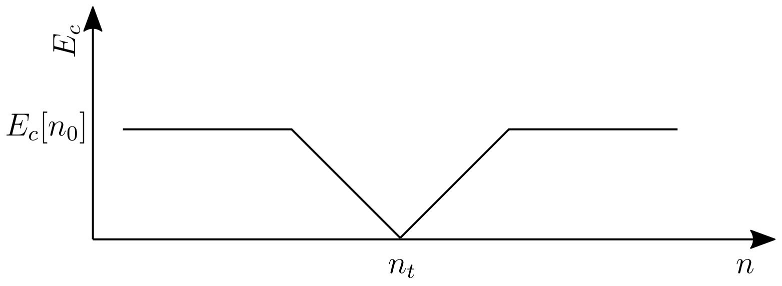


Figure 2.18: Classification Error (Cost) Example for The Transition Time Estimation Method Applied to a Linearly-Separable Dataset.

2.6.3 Input Correlation Analysis

After defining the best output targets, an analysis was done to discover which inputs are important for the classification (this spans until the end of Section 2.6.4) starting with the correlation analysis.

This consists of inspecting the correlation matrix (Figure 2.19) and spotting inputs with significant correlation coefficient. Coefficients near $+1$ indicate that those inputs contain very similar information, if one rises the other does so as well, and the opposite can be said if the coefficient is near -1, once one variable rises the other one lowers.

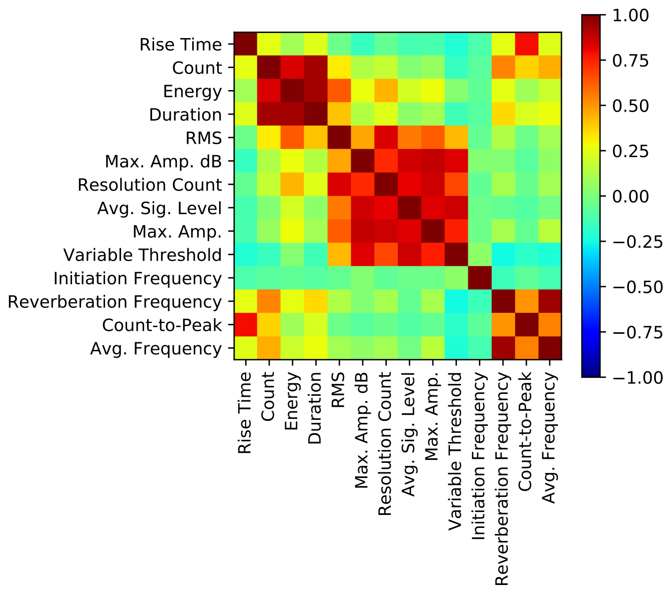


Figure 2.19: Linear Correlation Matrix for Test CP3

This is a good indicator of overlapping information from the inputs and combined with the relevance analysis (Section 2.6.4), one can determine which inputs are going to be discarded.

2.6.4 Relevance Analysis

Although linear correlation can show "overlapping information" between inputs, it is (as the name implies) a linear indicator. If those inputs have different non-linear relationships, the correlation coefficient may not indicate such. Considering the output is a binary function (classifier), correlation between input and output may not be the best metric to measure its importance to the model.

In order to complement the correlation analysis method, a relevance [37] one was implemented. It has a simple implementation, after training and testing a model with all inputs, each input is replaced by their mean and the model is tested again. A general (Equation 2.6) and by class (Equation 2.7) accuracy metric is then used to measure relevance.

$$R_i = \frac{1}{N} \sum_{n=1}^{N} (m_n - M_{in}) \Bigg| \, \mathbf{x}_i = \mu_{\mathbf{x}_i} \tag{2.6}$$

$$R_{ik} = \frac{1}{N} \sum_{n=1}^{N} (m_{nk} - M_{ink}) \Bigg| \, \mathbf{x}_i = \mu_{\mathbf{x}_i} \tag{2.7}$$

Where i is the i-th parameter, $\mathbf{x}_i$, N is the amount of different models to be compared, m_n is a model trained with all parameters and M_{in} the model trained replacing $\mathbf{x}_i$ with its mean $\mu_{\mathbf{x}_i}$. Also, for Equation 2.7, k is class k (in this case, NP, SP or UP).

Parameters that are important for classifying have high positive value, the ones that possess negative relevance show that by removing those inputs the result gets better, indicating that those are most likely noise regarding the classification.

Chapter 3

Results

This chapter focus on all the results of this work, including those from the preprocessing stages. Each technique was applied individually to each CP and compared in this chapter.

3.1 Preprocessing

3.1.1 Resolution Analysis

The resolution colormap for the tests (Figures 3.1, 3.2 and 3.3) show that only a few files were encoded with over 10 bits. Moreover, the vast majority of the data was encoded with less than 8 bits. Several channels were as good as dead, such as channels $8-10$ from test CP2. Others, albeit possessing an acceptable resolution (channel 8 from test CP4) have nothing but noise.

This strongly suggests that something was incorrectly calibrated or the system is so noisy that most AE were immersed. This severely hinders the effectiveness of the model since noise tends to appear at the output (if shown at the input). Moreover, the files and channels that met the minimum bits requirement (8 bits) represent a small fraction of the entire data (Table 3.1).

This step was fundamental to this work since it allowed only a portion of the data to be inspected. And it also showed that (at least for these datasets) the data is intrinsically precarious.

Table 3.1: Amount of Relevant Data From the Resolution Analysis.

Test	Total Data (Files × Channels)	Possibly Relevant Data	Ratio (%)
CP2	24208	1310	5.41
CP3	23984	3781	15.76
CP4	68176	6296	9.23

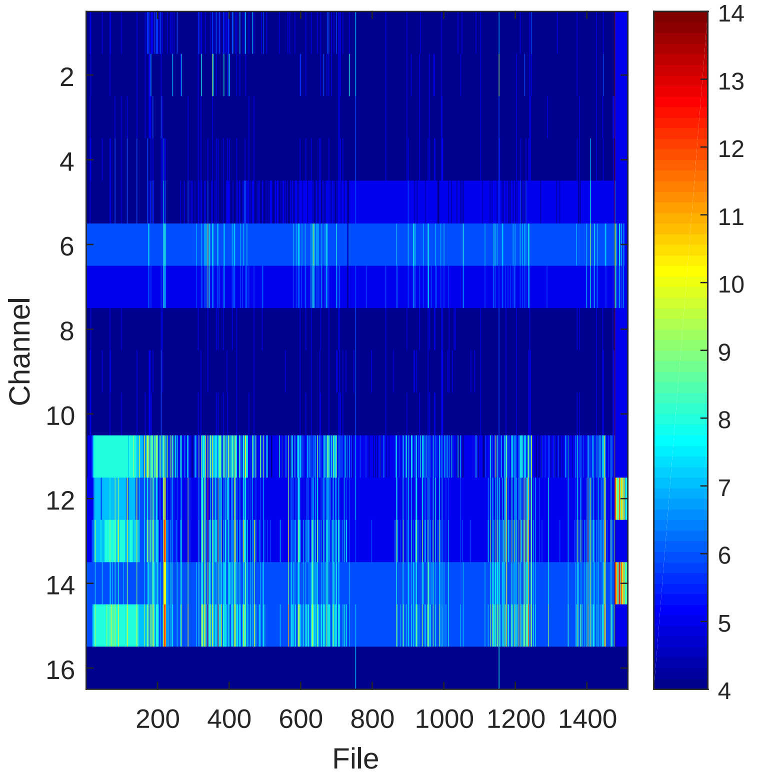


Figure 3.1: Bit Resolution Colormap for Test CP2.

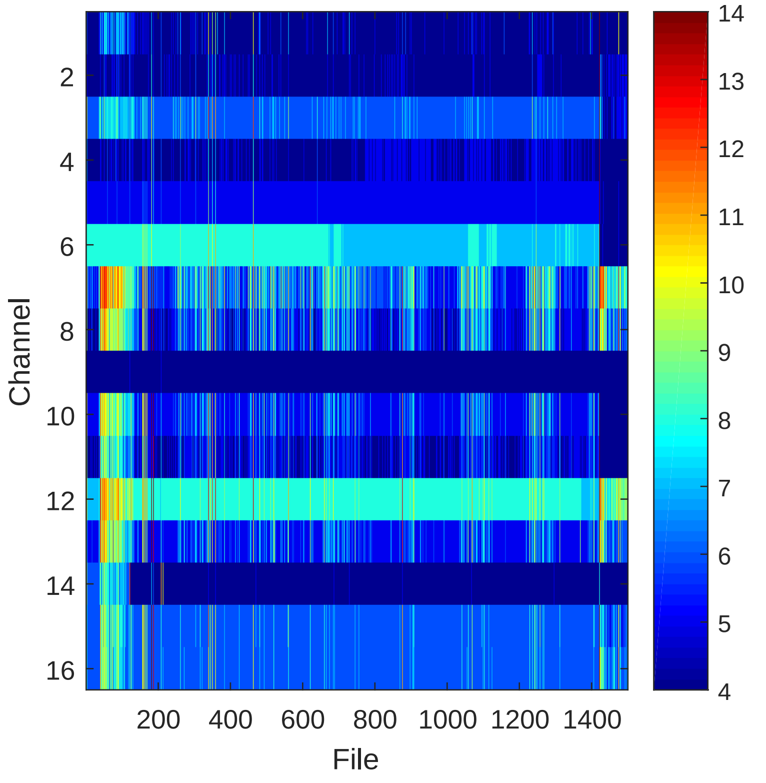


Figure 3.2: Bit Resolution Colormap for Test CP3.

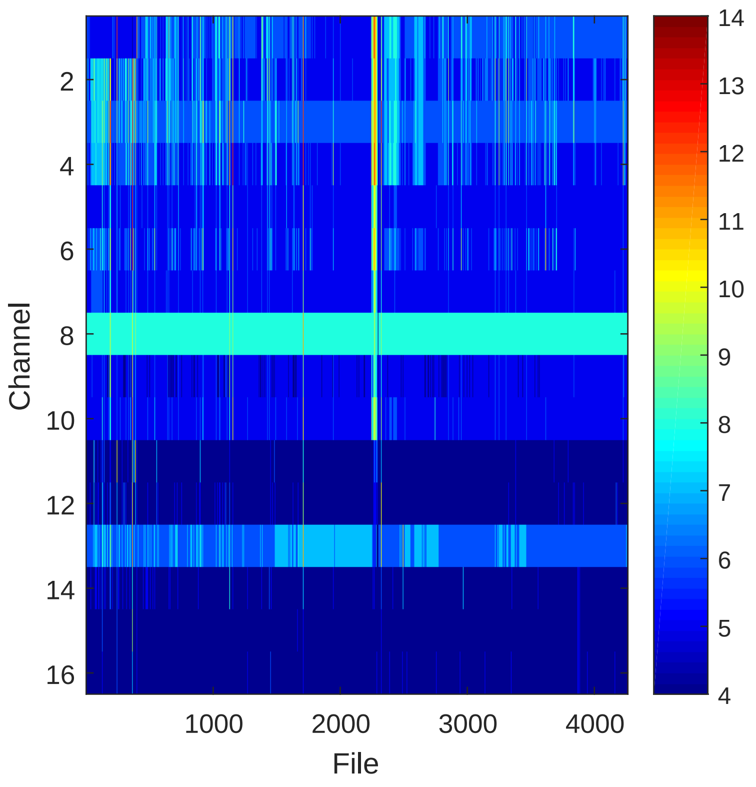


Figure 3.3: Bit Resolution Colormap for Test CP4.

3.1.2 Frequency Data

The relevant frequency ranges were determined after visually analysing the correlation plots for each CP (Figures 3.4, 3.5 and 3.6). In total, 8 frequency ranges were chosen:

- $\Omega_1 = 0 - 2.250\text{kHz}$
- $\Omega_2 = 3.09 - 4.96\text{kHz}$
- $\Omega_3 = 5.50 - 9.54\text{kHz}$
- $\Omega_4 = 10.34 - 27.81\text{kHz}$
- $\Omega_5 = 31.01 - 50.12\text{kHz}$
- $\Omega_6 = 50.66 - 58.71\text{kHz}$
- $\Omega_7 = 62.21 - 71.07\text{kHz}$
- $\Omega_8 = 82.09 - 113.03\text{kHz}$

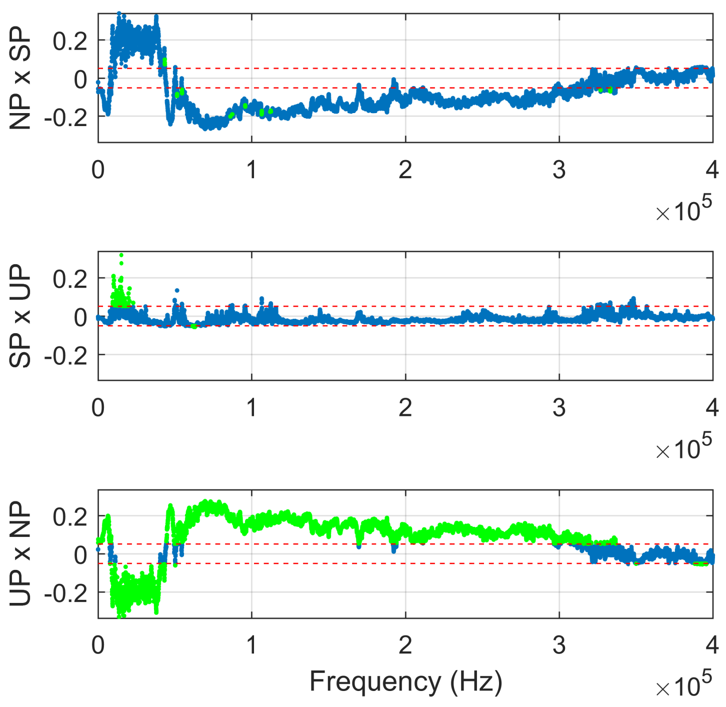


Figure 3.4: Correlation Plot Between Each Frequency and The Output Classes: No-Propagation, Stable Propagation and Unstable Propagation for Test CP2.

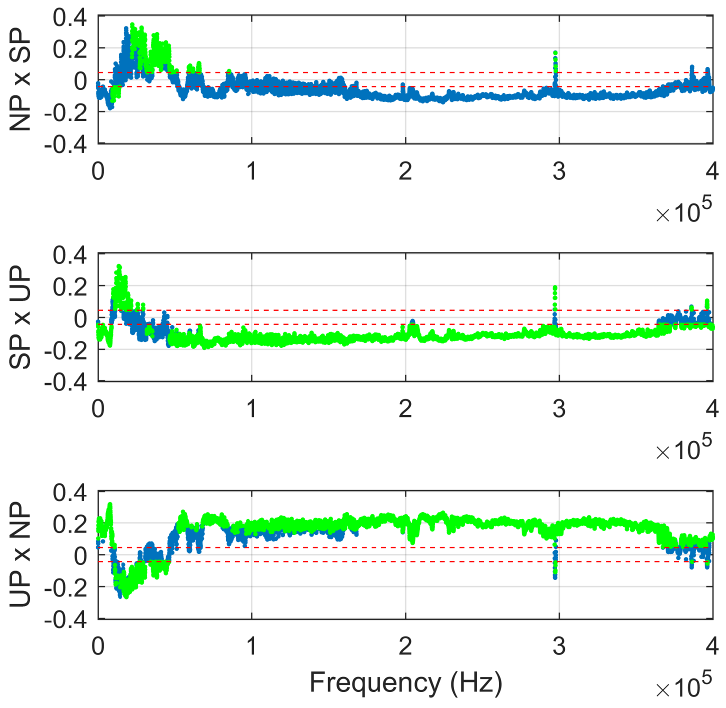


Figure 3.5: Correlation Plot Between Each Frequency and The Output Classes: No-Propagation, Stable Propagation and Unstable Propagation for Test CP3.

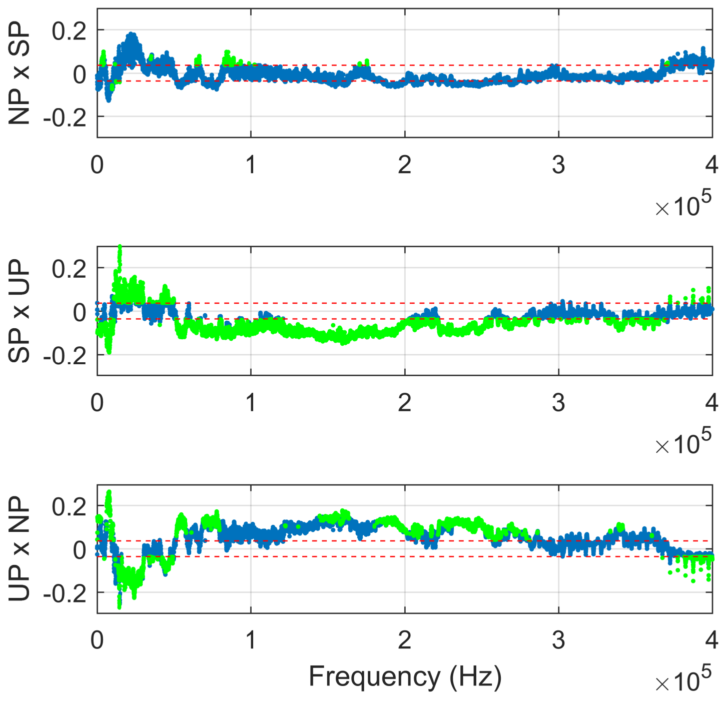


Figure 3.6: Correlation Plot Between Each Frequency and The Output Classes: No-Propagation, Stable Propagation and Unstable Propagation for Test CP4.

3.1.3 Input Correlation & Relevance Analysis

All three correlation matrices (Figures 3.7, 3.8 and 3.9) have mainly three areas of high correlation. Most are expected, for instance, *Max. Amp. dB* and *Max. Amp* represent the same quantity, but with different scales. There are no significant negative correlation coefficients. In all, these results indicate that a good portion of these parameters can be eliminated. In order to rule out which parameters are important to the classifier, one needs to complement the aforementioned results with a Relevance Analysis.

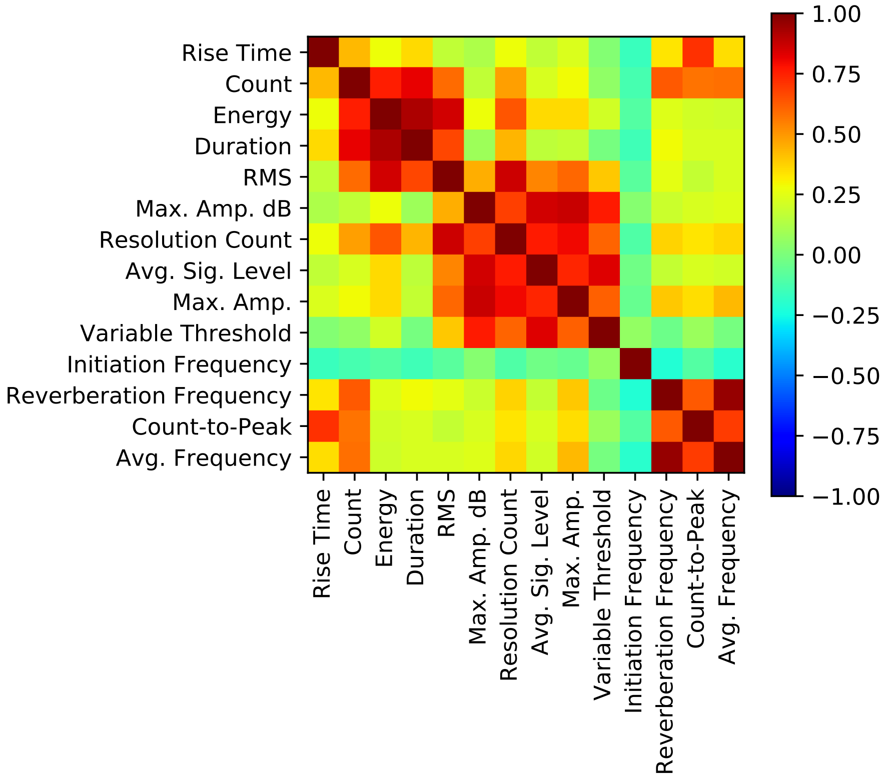


Figure 3.7: Linear Correlation Matrix for Test CP2

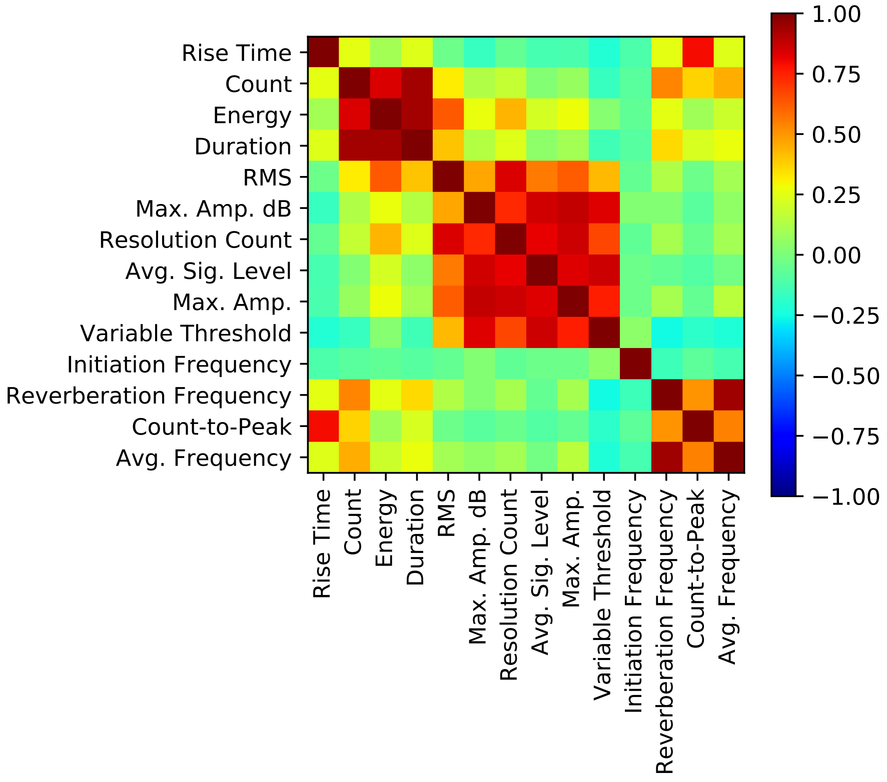


Figure 3.8: Linear Correlation Matrix for Test CP3

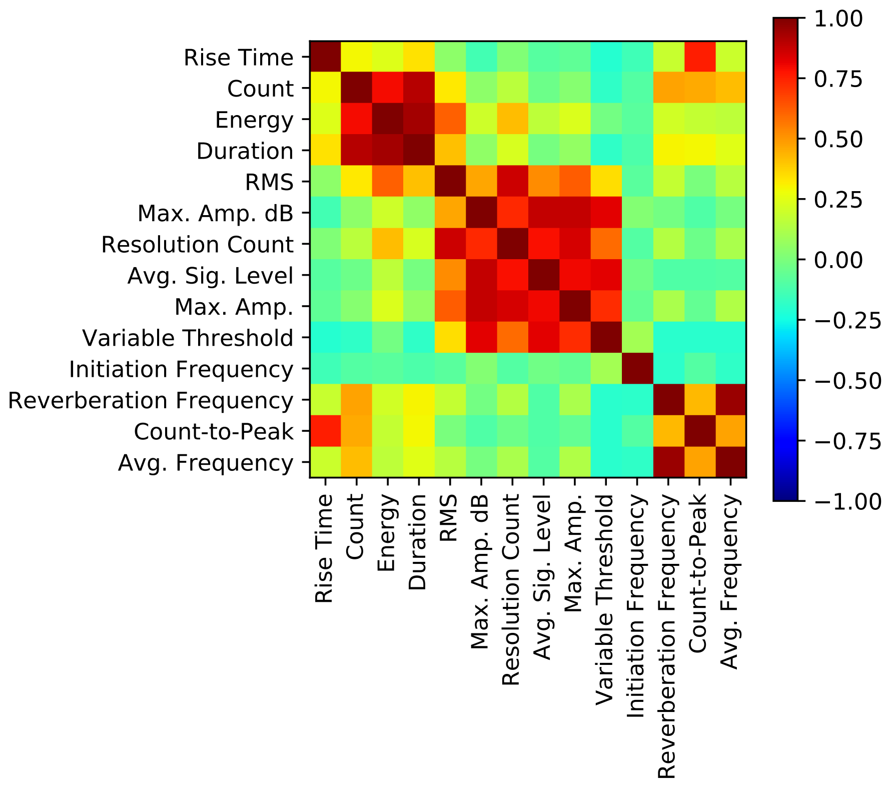


Figure 3.9: Linear Correlation Matrix for Test CP4

The relevance was calculated for each class (Figures 3.10, 3.11 and 3.12). For all tests, the last 5 parameters were highly relevant for both NP and SP classes. The first 5 parameters however were shown to have a relatively low relevance for all classes.

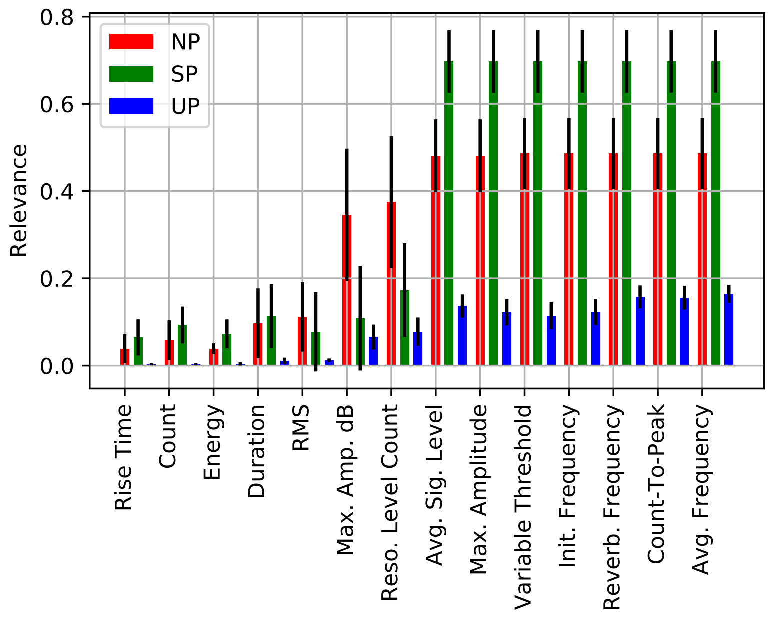


Figure 3.10: Relevance Analysis by Class for Test CP2.

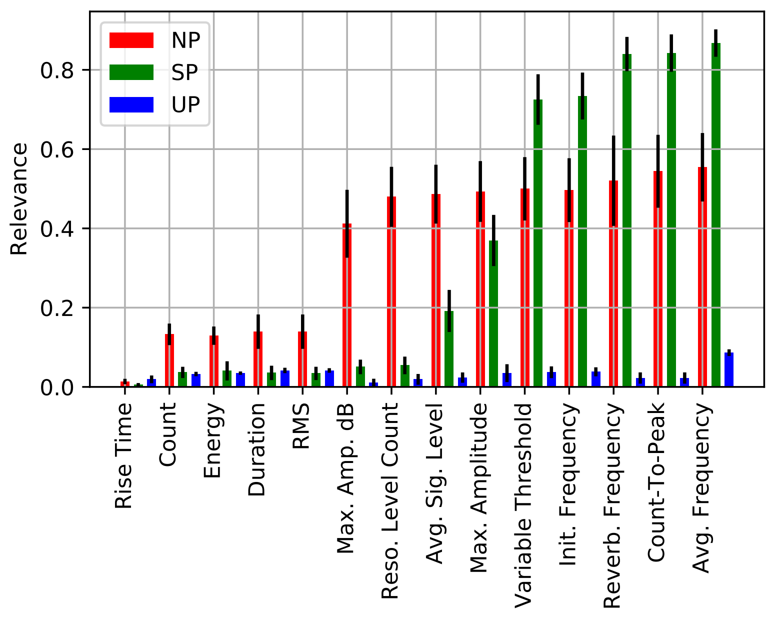


Figure 3.11: Relevance Analysis by Class for Test CP3

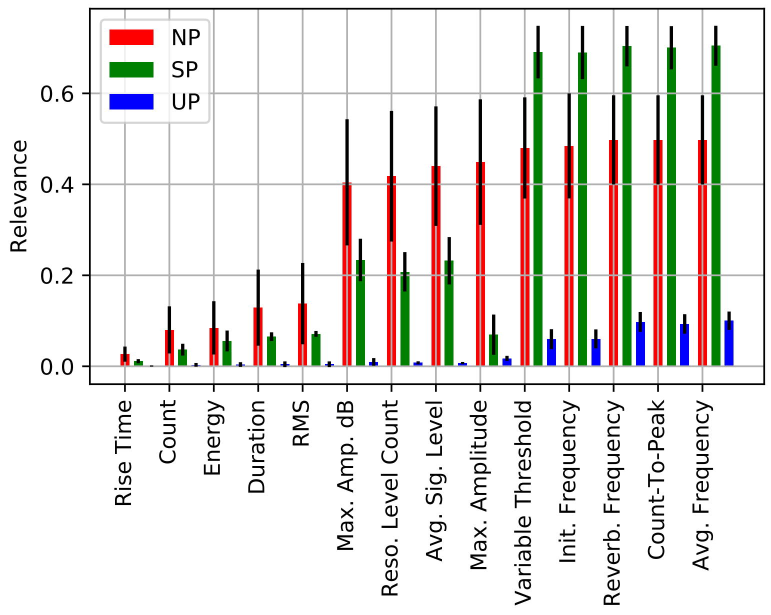


Figure 3.12: Relevance Analysis by Class for Test CP4

The analysis for test CP2 (Figure 3.10) displays a strong relevance for class SP of parameters *Average Signal Level* and *Max. Amplitude* that is not present at the later tests. Excluding the first five parameters, all others have a significant (higher than 30%) relevance for class NP (arguably the hardest one to differentiate). Class SP is mainly separated by the last 5 parameters, considering the high relevance (higher than 70%).

Taking both analyses into consideration, in an attempt to select only the relevant parameters and minimize overlapping information, the AE parameters chosen to "feed" the ANN were:

- Average Frequency
- Count-To-Peak
- Initiation Frequency
- Max. Amplitude
- Variable Threshold

According to the results from test CP2, the last 7 parameters are the most relevant. Out of those 7, the last 3 (*Count-To-Peak*, *Reverberation Frequency* and *Avg. Frequency*) are correlated (Figure 3.7), specially the last two. Therefore, *Count-To-Peak* and *Avg. Frequency* were chosen. *Initiation Frequency* is not correlated to any other variable, a strong relevance across all tests demonstrated that this parameter is essential and therefore was picked. *Max. Amplitude* and *Variable Threshold*, together with the aforementioned parameters, amount to the 5 most relevant inputs. These are also highly correlated to the other relevant parameters like *Average Signal Level*.

After defining the inputs for the ANN, one still needs to determine the best architecture. For this work, a shallow neural network with a single hidden layer was used. The amount of neurons was varied from 1 to 20. The samples were separated using 5-fold cross-validation (Section 1.6.1). According to this analysis, a satisfying size for the ANN would be 15 neurons at the hidden layer (Figures 3.13, 3.14 and 3.15).

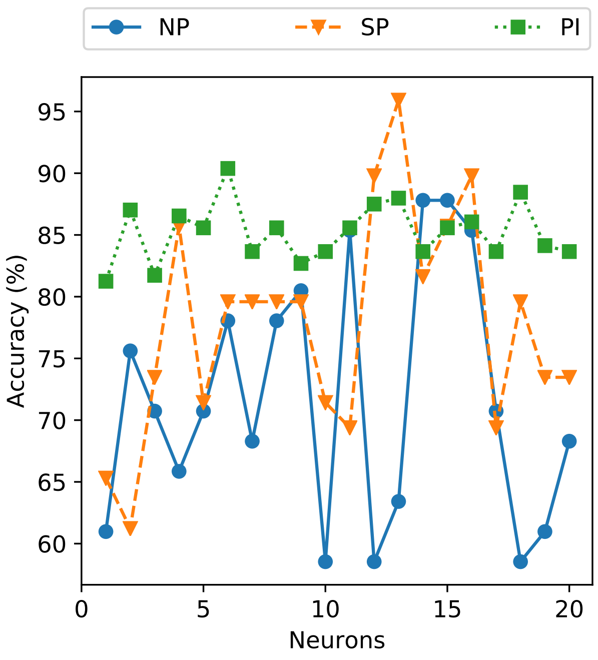


Figure 3.13: Network Size Investigation for Test CP2

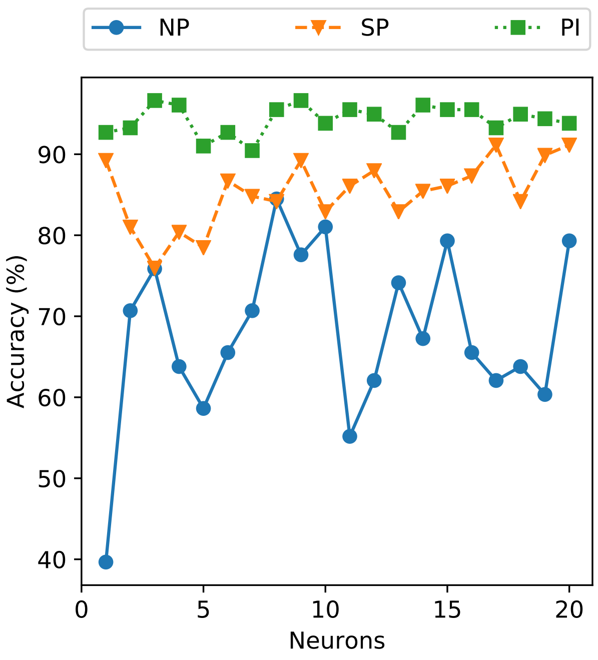


Figure 3.14: Network Size Investigation for Test CP3

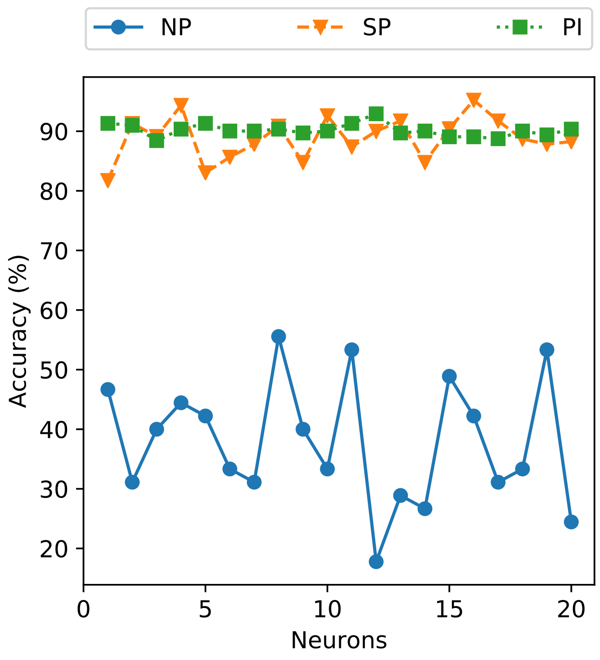


Figure 3.15: Network Size Investigation for Test CP4

In sum, the model consists of a shallow artificial neural network (Section 1.4) with 15 neurons on its single hidden layer, "fed" with 13 inputs. Five extracted directly from the time-domain AE parameters (Section 2.4.3) and 8 frequency parameters (Section 2.4.4). An additional pre-processing stage was done to ensure that the mean and standard deviation of each parameter were set to 0 and 1 respectively (Z-Score normalization).

3.2 Transition-Time-Estimation

The TTE method was applied at a later stage and presented some interesting results. For all tests, the cost graph displayed 3 valleys per cycle (test CP4 has 2 cycles). This is an indication that there is a fourth class to be considered. The same results were found on a related work that applied this technique [28] which supports the idea that there are 4 classes instead of 3.

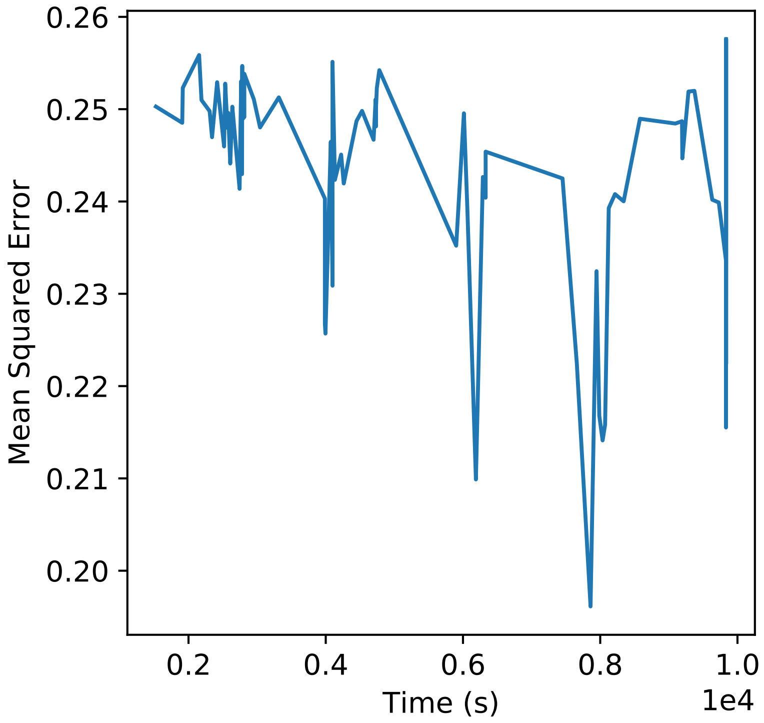


Figure 3.16: Transition-Time-Estimation Method Applied to Test CP2

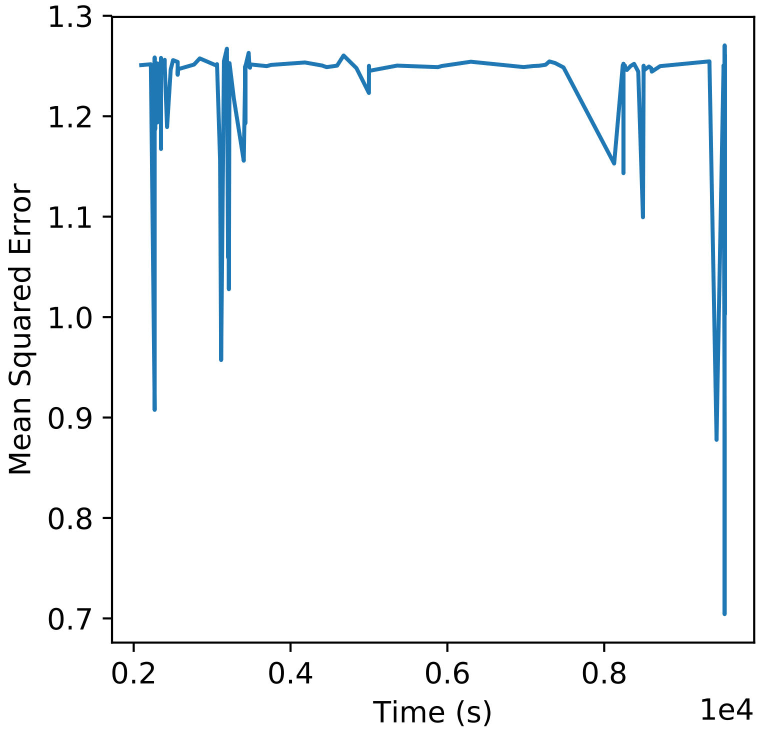


Figure 3.17: Transition-Time-Estimation Method Applied to Test CP3

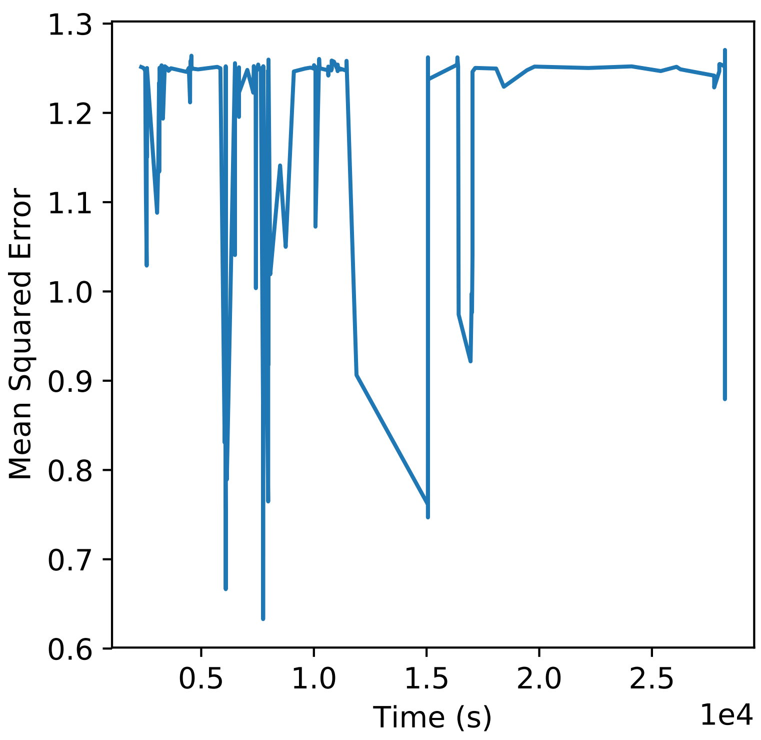


Figure 3.18: Transition-Time-Estimation Method Applied to Test CP4

The results for test CP4 (Figure 3.18) show when the cycle is restarted (around $1.5e4s$) and have an abnormal amount of valleys due to it. However, it is possible to notice that both cycles have 3 valleys, which is in accordance with the previous results.

3.3 Classification Results

For each test, 25 different networks and training/validation/test shuffled sets were instantiated. The train-validation-test split was done following a 0.7/0.15/0.15 ratio. The only test that had a different division (0.7/0.3) was test CP2 due to the highly unbalanced classes. For each one of the 25 simulations, 4 different neural networks were created and only the best was selected. This gives a total of 100 different neural networks.

In total, the tests had 1494, 1978 and 2934 samples for tests CP2, CP3 and CP4 respectively. The classes were balanced in the training by means of altering the cost based on what class that sample belongs. In other words, if we have 10 samples, divided as 2/5/3 for classes NP, SP and UP, their respective weights would be 0.5/0.2/0.333 when calculating the error from each sample.

Although the results from TTE (Section 3.2) indicate 4 classes for tests CP2 and CP3, there was a higher classification error rate using a 4-class labelling. The best class division came from ignoring the first valley. In other words, the division between NP and SP was given by the second valley (Figures 3.16 and 3.17) instead of the first. The additional classes were interpreted as “transition” classes, and therefore gained the label “NP/SP”.

The metric used is the Confusion Matrix (CM). The rows represent the real class, while columns predicted class. For each test, the approximate amount of samples for each model can be seen below (Tables 3.2 and 3.3):

Class	CP2	CP3
NP	56	18
NP/SP	22	37
SP	8	127
UP	136	116

Table 3.2: Aproximate Amount of Samples of The Test Set for Each Run Using 4 Classes for Tests CP2 and CP3.

Class	CP4
NP_1	65
SP_1	40
UP_1	50
NP_2	24
SP_2	84
UP_2	170

Table 3.3: Aproximate Amount of Samples of The Test Set for Each Run Using 6 Classes for Test CP4.

3.3.1 CP2

According to the TTE method (Section 3.2), there should be 4 classes, however, the results were unsatisfactory (Table 3.4a). With over 50% and 32% confusion between the first two classes, and an overall bad classification rate. The standard deviation (Table 3.4b) indicates that the model was not capable of separating the classes (apart from the UP class).

Merging the first two classes did not seem to improve the results much further, showing over 71% incorrect predictions for class SP (Table 3.5a). This indicates that the separation between these two classes is highly non-trivial (at least for this dataset). This can be due to the extremely unpredictable and complex phenomenon that is AE. It is possible that the division between classes is more of a fuzzy than a hard division.

The classes are also really unbalanced, the least populated one (SP) has around 70 samples for the test set even with 30% of the data. This also hinders the classification task. Due to the low amount of samples, the standard deviation also becomes high (Table 3.5b), indicating that the model was not stable.

(%)	NP	NP/SP	SP	UP
NP	**57.71**	32.21	10.07	0.00
NP/SP	52.18	**36.36**	11.45	0.00
SP	48.80	24.00	**7.20**	20.00
UP	0.67	0.18	0.35	**98.80**

(a)

(%)	NP	NP/SP	SP	UP
NP	**40.64**	39.46	27.27	0.00
NP/SP	45.96	**44.54**	31.02	0.00
SP	29.98	29.39	**19.50**	0.00
UP	0.92	0.47	1.31	**1.40**

(b)

Table 3.4: Mean (a) and Standard Deviation (b) of the Confusion Matrix Using 4 Classes for Test CP2 in Percentage.

(%)	NP	SP	UP
NP	**92.83**	7.17	0.00
SP	71.50	**20.56**	7.94
UP	1.25	0.02	**98.73**

(a)

(%)	NP	SP	UP
NP	**6.33**	6.33	0.00
SP	11.80	**12.19**	0.62
UP	0.46	0.08	**0.46**

(b)

Table 3.5: Mean (a) and Standard Deviation (b) of the Confusion Matrix for Test CP2 in Percentage.

3.3.2 CP3

The results from test CP3 are very analogous to the previously presented CP2 results. Again the TTE method indicated an existence of 4 different patterns, but after training the ANNs, there is still a high rate of confusion between classes NP and NP/SP (21%). It is important to note that there is a significant rate of confusion between classes NP and SP which should not happen (for this specific class division). This is because the "distance" between the two classes is (supposedly) too far. Even still, it was decided that the amount of confusion between the first two classes was too significant, and the classes were fused, much like what was done previously.

This skyrocketed the classification rate, test CP3 presented a satisfactory result (Table 3.7a), yielding 89% and (approximately) 97% correct predictions for classes SP and UP. Unfortunately, the performance for class NP was lower, achieving around 79%. This is a slight step-down from Test CP2 but with high rates of correct predictions still. The values of the standard deviation (lower than 2%) indicate that the model is robust and stable. Important to note that the standard deviation got lower, meaning that the 4 classes division was a more unstable one.

(%)	NP	NP/SP	SP	UP
NP	**63.56**	8.67	27.78	0.00
NP/SP	21.56	**54.44**	24.00	0.00
SP	29.07	2.20	**67.91**	0.82
UP	0.86	0.86	1.72	**96.55**

(a)

(Samples)	NP	NP/SP	SP	UP
NP	**4.99**	4.99	0.00	0.00
NP/SP	2.64	**2.22**	1.91	0.00
SP	2.23	0.63	**2.24**	0.27
UP	0.00	0.00	0.00	**0.00**

(b)

Table 3.6: Mean (a) and Standard Deviation (b) of the Confusion Matrix Using 4 Classes for Test CP3 in Percentage.

(%)	NP	SP	UP
NP	**78.79**	21.21	0.00
SP	10.85	**89.15**	0.00
UP	2.54	0.50	**96.96**

(a)

(%)	NP	SP	UP
NP	**0.81**	0.81	0.00
SP	1.50	**1.50**	0.00
UP	0.52	0.53	**0.30**

(b)

Table 3.7: Mean (a) and Standard Deviation (b) of the Confusion Matrix for Test CP3 in Percentage.

3.3.3 CP4

Test CP4 is different since it had 2 cycles. The TTE method applied to this dataset created not 4, but 7 different patterns (Figure 3.18). Considering what was done to the previous tests, the first division was also discarded (between NP and NP/SP) which brings a total of 6 classes. These 6 classes were then divided into 2 sets of NP, SP and UP, each set per cycle.

This division did not yield good results, as the correct predictions (excepting the UP$_2$ class) did not reach 62%. The CM (Table 3.8a) shows a big confusion for the first cycle as a whole. The model could not identify class NP$_1$ well, since it is heavily spread across all other classes.

An attempt to join all these classes together as done simply fusing the cycles, that is, NP$_1$ was joined with NP$_2$ and so forth. This still did not convey good results (Table 3.9a) with a low 53% correct prediction rate for class SP. The high values of standard deviation (Table 3.9b) also show that the model is not stable. However, classes NP and UP are reasonably well defined, with correct prediction rates of 84% and 76% approximately.

(%)	NP$_1$	SP$_1$	UP$_1$	NP$_2$	SP$_2$	UP$_2$
NP$_1$	**56.61**	10.73	23.45	8.06	1.15	0.00
SP$_1$	25.10	**49.80**	19.60	4.60	0.10	0.80
UP$_1$	37.70	7.56	**41.41**	11.93	1.11	0.30
NP$_2$	14.24	15.20	9.12	**61.12**	0.32	0.00
SP$_2$	23.75	3.85	5.65	5.30	**40.65**	20.80
UP$_2$	0.00	0.34	0.16	0.16	3.57	**95.77**

(a)

(%)	NP$_1$	SP$_1$	UP$_1$	NP$_2$	SP$_2$	UP$_2$
NP$_1$	**5.32**	3.93	5.11	2.17	1.07	0.00
SP$_1$	4.55	**4.41**	4.57	2.80	0.49	1.17
UP$_1$	3.95	2.51	**4.74**	2.46	1.05	0.68
NP$_2$	3.93	3.20	3.49	**2.66**	1.09	0.00
SP$_2$	3.12	1.41	2.94	1.67	**5.42**	5.27
UP$_2$	0.00	0.45	0.26	0.30	1.25	**1.51**

(b)

Table 3.8: Mean (a) and Standard Deviation (b) of the Confusion Matrix With All Classes for Test CP4 in Percentage.

(%)	NP	SP	UP
NP	**84.93**	11.07	4.00
SP	33.72	**52.71**	13.57
UP	20.09	6.23	**73.69**

(a)

(%)	NP	SP	UP
NP	**3.79**	3.93	0.88
SP	2.08	**7.74**	7.09
UP	0.87	2.07	**1.71**

(b)

Table 3.9: Mean (a) and Standard Deviation (b) of the Confusion Matrix After Joining Cycles for Test CP4 in Percentage.

3.4 Conclusion

In the end the objectives were met, the tools that eventually can build a fully automated system to extract and process big amounts of data from an AE test were implemented. However, it is unclear if an ANN-based classifier can differentiate the 3 pre-determined stages of the crack.

The results for test CP3 are acceptable and usable, which is not the case for the other two tests, CP2 and CP4. For test CP2, the lack of enough samples from classes SP and NP took a heavy toll on the classifier. Test CP4 did not burst, which also hindered significantly the quality of the data. A lot of different techniques to preprocess it were implemented but unfortunately it was not enough to overcome its unpredictability. Furthermore, the author highly stresses the importance of a more controlled hydrostatic test.

However, the fact that these techniques resulted in a good classification for test CP3 indicates that it should be possible to apply a shallow neural network classifier for this problem. Unfortunately, considering the overall state of this dataset, it was not possible to achieve a good result.

The TTE method works to an extent. Although its results match that from [28], the subsequent application of these results to the data did not work out. A possible reason would be that the window size was not big enough, resulting in fluctuations of possible pattern combinations within the same class. Also, it is possible that a single class possesses more than one pattern, or is even a multi-modal distribution.

Chapter 4

Future Works

After working for over an year with the data, the author started to develop a feeling that trying to classify each AE instance separately is but a hindrance. The parameter extracted in this work (VTHR) was an attempt to feed information about context to the neural network. A related past work [26] developed a way to create information about the context by changing the classification based on past samples.

Since context seems to be key, a possibly good alternative to a shallow MLP would be some architecture that learns the context. Recurrent Neural Networks have been used for this purpose (albeit in the field of text interpretation) and could pose a solid model for this problem. Different models could also be used for distinguishing the first two classes (the biggest challenge) like the AdaBoost classifier.

Other points would be to perfect the TTE method or further investigate other transformations for the AE, like the Wavelet Transform. A prototype could also be built with a fully automated data collection, filtering and interpreting.

Bibliography

[1] "English: Drawing Illustrating the Process of Synaptic Transmission in Neurons, Cropped from Original in an NIA Brochure." 2009-12-30, first publication of original unknown.

[2] WIKIMEDIA COMMONS. "Neuron". `https://upload.wikimedia.org/wikipedia/commons/b/b5/Neuron.svg`, dez. 2006.

[3] MARTINTHOMA. "English: Sigmoid Function". maio 2014.

[4] JORGE STOLFI FROM WIKIMEDIA COMMONS. "File:Axial Stress Noavg.Svg", *Wikipedia*, fev. 2013.

[5] IGOR KOKCHAROV FROM WIKIMEDIA COMMONS. "File:CrackForceLines.Png", *Wikipedia*, dez. 2018.

[6] RONNY KOHAVI AND BARRY BECKER. "UCI Machine Learning Repository: Adult Data Set". https://archive.ics.uci.edu/ml/datasets/Adult, 1994.

[7] "Energy and Air Pollution - World Energy Outlook 2016 Special Report", p. 266, 2016.

[8] TURING, A. M. "Computing Machinery And Intelligence", *Mind*, v. LIX, n. 236, pp. 433–460, 1950. ISSN: 0026-4423, 1460-2113. doi: 10.1093/mind/LIX.236.433.

[9] MCCULLOCH, W. S., PITTS, W. "A Logical Calculus of the Ideas Immanent in Nervous Activity", *Bulletin of Mathematical Biophysics*, v. 5, n. 4, pp. 115–133, dez. 1943. ISSN: 1522-9602. doi: 10.1007/BF02478259.

[10] MORRIS, R. G. *D.O. Hebb: The Organization of Behavior, Wiley: New York; 1949*, v. 50. 1999 Nov-Dec.

[11] FRANK ROSENBLATT. *The Perceptron, A Perceiving and Recognizing Automaton.* Relatório Técnico 85-460-1, Cornell Aeronautical Laboratory, INC, Buffalo, N. Y., jan. 1957.

[12] RUMELHART, D. E., HINTON, G. E., WILLIAMS, R. J. "Learning Representations by Back-Propagating Errors", *Nature*, v. 323, n. 6088, pp. 533–536, out. 1986. ISSN: 1476-4687. doi: 10.1038/323533a0.

[13] RUDER, S. "An Overview of Gradient Descent Optimization Algorithms", *arXiv:1609.04747 [cs]*, set. 2016.

[14] KINGMA, D. P., BA, J. "Adam: A Method for Stochastic Optimization", *arXiv:1412.6980 [cs]*, dez. 2014.

[15] MORÉ, J. J. "The Levenberg-Marquardt Algorithm: Implementation and Theory". In: *Numerical Analysis*, Lecture Notes in Mathematics, Springer, Berlin, Heidelberg, pp. 105–116, 1978. ISBN: 978-3-540-08538-6 978-3-540-35972-2. doi: 10.1007/BFb0067700.

[16] ZEILER, M. D. "ADADELTA: An Adaptive Learning Rate Method", dez. 2012.

[17] JOSEPH KAISER. *Untersuchungen über das Auftreten von Geräuschen beim Zugversuch*. Tese de Doutorado, Technische Hochschule München, Germany, 1950.

[18] DUNEGAN, H. L., HARRIS, D. O., TATRO, C. A. "Fracture Analysis by Use of Acoustic Emission", *Engineering Fracture Mechanics*, v. 1, n. 1, pp. 105–122, jun. 1968. ISSN: 0013-7944. doi: 10.1016/0013-7944(68) 90018-0.

[19] LINDLEY, T. C., PALMER, I. G., RICHARDS, C. E. "Acoustic Emission Monitoring of Fatigue Crack Growth", *Materials Science and Engineering*, v. 32, n. 1, pp. 1–15, jan. 1978. ISSN: 0025-5416. doi: 10.1016/0025-5416(78)90206-9.

[20] NI, Q.-Q., IWAMOTO, M. "Wavelet Transform of Acoustic Emission Signals in Failure of Model Composites", *Engineering Fracture Mechanics*, v. 69, n. 6, pp. 717–728, abr. 2002. ISSN: 0013-7944. doi: 10.1016/S0013-7944(01)00105-9.

[21] SUN, H., KAVEH, M., TEWFIK, A. "Self-Organizing Map Neural Network for Transient Signal Classification in Mechanical Diagnostics", pp. 539–543, jan. 1999.

[22] HUGUET, S., GODIN, N., GAERTNER, R., et al. "Use of Acoustic Emission to Identify Damage Modes in Glass Fibre Reinforced Polyester", *Com-*

posites Science and Technology, v. 62, n. 10-11, pp. 1433–1444, ago. 2002. doi: 10.1016/s0266-3538(02)00087-8.

[23] DA SILVA, R. R., CALÔBA, L. P., SIQUEIRA, M. H., et al. "Pattern Recognition of Weld Defects Detected by Radiographic Test", *NDT & E International*, v. 37, n. 6, pp. 461–470, set. 2004. ISSN: 09638695. doi: 10.1016/j.ndteint.2003.12.004.

[24] DA SILVA, R. R., SOARES, S. D., CALÔBA, L. P., et al. "Detection of the Propagation of Defects in Pressurised Pipes by Means of the Acoustic Emission Technique Using Artificial Neural Networks", *Insight - Non-Destructive Testing and Condition Monitoring*, v. 48, n. 1, pp. 45–51, jan. 2006. ISSN: 13542575. doi: 10.1784/insi.2006.48.1.45.

[25] PINTO, C., SILVA, R., CALÔBA, L., et al. "Uso de redes neurais artificiais na detecção de propagação de defeitos em dutos rígidos", *Matéria (Rio de Janeiro)*, v. 17, n. 3, pp. 1084–1097, 2012. ISSN: 1517-7076. doi: 10.1590/S1517-70762012000300006.

[26] CARLOS FERNANDO CARLIM PINTO. *Monitoração De Defeitos Em Dutos Rígidos Por Análise Dos Parâmetros De Emissão Acústica Utilizando Redes Neurais*. Tese de Doutorado, Federal University of Rio de Janeiro, Rio de Janeiro, RJ, dez. 2014.

[27] ORLANDO GÉA. *Monitoramento Da Propagação De Defeitos Em Dutos Rígidos Por Redes Neurais: Estudo Da Emissão Acústica Baseado Na Forma De Onda*. Tese de Doutorado, Federal University of Rio de Janeiro, Rio de Janeiro, RJ, dez. 2015.

[28] LUIZA RIBEIRO MARNET. *Monitoramento de Defeitos em Dutos Rígidos Longos por Parâmetros de Emissão Acústica e Redes Neurais*. Tese de Doutorado, Federal University of Rio de Janeiro, Rio de Janeiro, RJ, 2018.

[29] CHARLESWORTH, J. P., TEMPLE, J. A. G. *Engineering Applications of Ultrasonic Time-of-Flight Diffraction*. United States, Research Studies Press, 2001. ISBN: 978-0-86380-239-3.

[30] SILK, M. G., LIDINGTON, B. H. "The Potential of Scattered or Diffracted Ultrasound in the Determination of Crack Depth", *Non-Destructive Testing*, v. 8, n. 3, pp. 146–151, jun. 1975. ISSN: 0029-1021. doi: 10.1016/0029-1021(75)90024-9.

[31] PHYSICAL ACOUSTICS. "R1.5 - 5-20 kHz Very Low Frequency AE Sensor". https://www.physicalacoustics.com/by-product/sensors/R1.5-5-20-kHz-Very-Low-Frequency-AE-Sensor, .

[32] PHYSICAL ACOUSTICS. "R15I-AST - 150 kHz Integral Preamp AE Sensor". https://www.physicalacoustics.com/by-product/sensors/R15I-AST-150-kHz-Integral-Preamp-AE-Sensor, .

[33] PHYSICAL ACOUSTICS. "WD - 100-900 kHz Wideband Differential AE Sensor". https://www.physicalacoustics.com/by-product/sensors/WD-100-900-kHz-Wideband-Differential-AE-Sensor, .

[34] NATIONAL INSTRUMENTS. "The NI TDMS File Format". .

[35] MISTRAS GROUP. "DiSP with AEwin USER'S MANUAL". jul. 2011.

[36] TAMURA, S., TATEISHI, M. "Capabilities of a Four-Layered Feedforward Neural Network: Four Layers versus Three", *IEEE Transactions on Neural Networks*, v. 8, n. 2, pp. 251–255, mar. 1997. ISSN: 1045-9227. doi: 10.1109/72.557662.

[37] JOSÉ MANOEL DE SEIXAS, LUIZ PEREIRA CALÔBA, IGOR DELPINO. "Relevance Criteria for Variance Selection in Classifier Designs". In: *International Conference on Engineering Applications of Neural Networks*, pp. 451–454, 1996.

Petro Acoustic Emission Documentation

Release alpha

Luiz Renno Costa

Feb 25, 2019

CONTENTS:

This documentation is incomplete and its meant to be a guideline for anyone interested in learning how the project's code is structured. Not all methods are documented, only the ones that the author judged relevant. Eventually all methods and parameters will be documented.

CHAPTER

ONE

STREAMING

This is the Streaming module.

1.1 Streaming Class

1.1.1 Main Parameters

class `Matlab.Streaming.classDesign.@StreamingClass.`**`StreamingClass`**(*CPString, varargin*)

The main class, it holds all information regarding an AE streaming test.

Parameters

- **`Waves`** (*`Wave Array`*) – A holder for all waves captured throughout the test.
- **`StreamingModel`** (*`StreamingModel Object`*) – Used for all modelling purposes, holds inputs, targets, etc.
- **`hdt`** (*`double`*) – Timing parameter, in seconds (Check PAC Manual).
- **`hlt`** (*`double`*) – Timing parameter, in seconds (Check PAC Manual).
- **`pdt`** (*`double`*) – Timing parameter, in seconds (Check PAC Manual).
- **`countWaveform`** (*`double`*) – Amount of captured Waveforms
- **`spIndexes`** (*`double Array`*) – Contains all indexes of the NP class. (sp -> NP)
- **`peIndexes`** (*`double Array`*) – Contains all indexes of the SP class. (pe -> SP)
- **`piIndexes`** (*`double Array`*) – Contains all indexes of the UP class. (pi -> UP)

- **timePE** (*double*) – Starting time for SP class in seconds. (pe -> SP)
- **timePI** (*double*) – Starting time for UP class in seconds. (pi -> UP)
- **cycleDividers** (*double Array*) – Holds which files are the cycle dividers, concerning the streaming files.
- **adjusted** (*Bool*) – Flag that indicates if the time parameters were already adjusted.
- **noiseLevelMatrix** (*Cell Array*) – Variable that holds the noise level per file per channel. Each component of the array corresponds to a diffent cycle (demarked by cycleDividers).
- **fields** (*Cell Array*) – An array of fixed strings for each Wave object parameter.
- **description** (*string*) – A description for the test.
- **folderTDMS** (*Cell Array*) – Holds which folders the TDMS files are. MUST BE ABSOLUTE PATH.
- **folderMatlabCopy** (*string*) – Path for the local .mat files copied from the streaming files.
- **fileTemplate** (*Cell Array*) – Strings that specify the filename format. Used for reading the TDMS or .mat local files.
- **TOFDReferenceChannel** (*double*) – Holds which streaming channel is the reference when calculating time delays between channels.
- **sortedFolder** (*double Array*) – Variable used ONLY for the CP4 test. It tells which file is in which folder. This is needed since the files for CP4 are split between two folders and in a somewhat-random order.
- **totalFiles** (*double Array*) – Total files for each nominated cycle (demarked by cycleDividers).
- **numBitsFileChannel** (*Cell Array*) – An array which each element is a matrix of the bit resolution per file and channel of each cycle (demarked by cycleDividers).
- **numUniqueElementsChannel** (*Cell Array*) – The same as numBitsFileChannel, but with the total amount of digital levels used instead of bits.
- **filesToSkip** (*Cell Array*) – An array of lists that holds which files to skip when reading the test. Filled manually.
- **tofdDifferences** (*struct*) – Variable that holds which channels to remove TOFD from, and how they are related, that is, how they are delayed in relation to each other.

- **`minBits`** (*`double`*) – The minimum acceptable bits level for inspecting a TDMS file for waves.
- **`power`** (*`double array`*) – Matrix with the single-sided power spectrum for each wave.
- **`normalizedPower`** (*`double array`*) – Matrix with the normalized single-sided power spectrum for each wave.
- **`phase`** (*`double array`*) – Matrix with the phase for each wave.
- **`frequencyArray`** (*`double array`*) – An array with each frequency.
- **`freqSlots`** (*`double`*) – Array with the frequency slots used to generate the frequency inputs for the model.

1.1.2 Constructor

`StreamingClass.`**`StreamingClass`**(*CPString*, *varargin*)

The main constructor for the whole project. Instantiates a StreamingClass object that holds everything necessary to read and analyse all TDSM files related to an acoustic emission test. All parameters were defined to ensure the best usage. Some can be changed, like timePE. Others however, can not, as changing them will completely break the constructor, or heavily endanger the quality of the collected data.

Parameters

- **`CPString`** (*`string`*) – String to control which test to start analysing. Can be either ‘CP2’, ‘CP3’ or ‘CP4’.
- **`varargin`** – Can be used to specify which cycle/file combination to start. Must be (initialCycle, initialFile), otherwhise initialFile = 1.

1.1.3 Methods

`Matlab.Streaming.classDesign.@StreamingClass.`**`adjustCycles`**(*this*)

Adjusts the triggerTime for the waves from different cycles. After adjusting the cycles, the adjusted parameter is set to TRUE (1).

`Matlab.Streaming.classDesign.@StreamingClass.`**`divideClasses`**(*this*)

Divides the 3 classes based on this.timePE and this.timePI. Also creates the arrays that hold the wave indexes for each class and the target to be used on the model.

`Matlab.Streaming.classDesign.@StreamingClass.`**`identifyWaves`**(*obj*, *rawData*, *channels*, *fs*, *noiseLevel*, *fileNumber*, *lastIndex*, *backupPath*, *cycle*)

Identifies all acoustic emission waves from within a streaming file. This function follows the guidelines contained on the PAC Manual describing how the AE capture is done. It uses only two (HDT and HLT) of the three timing parameters. The threshold is a multiplied noiseLevel (by a euristically defined number).

Parameters

- **`channels`** (*`double array`*) – Array containing all channels to investigate.
- **`rawData`** (*`int16 Array`*) – The raw collected wave data.
- **`fs`** (*`double`*) – Sampling Frequency
- **`noiseLevel`** (*`double Array`*) – An array with the estimated noise level per channel.
- **`fileNumber`** (*`double`*) – Which file to investigate.
- **`backupPath`** (*`string`*) – Absolute path to the local .mat files.
- **`cycle`** (*`double`*) – Tells which cycle the file is located at.

Returns A StreamingClass object with the captured waves stored.

Returns The indexes where the last captured waves ended (per channel).

`Matlab.Streaming.classDesign.@StreamingClass.`**`propertyVector`**(*this*, *propertyString*)

Returns a array of a chosen property for ALL captured Waves.

Parameters **`propertyString`** (*`string`*) – Which wave parameter to export.

Returns Array of an Wave property.

`Matlab.Streaming.classDesign.@StreamingClass.`**`createFrequencyData`**(*this*, *fs*)

Calculates FFT of N points for all waves. Used to calculate normalizedPower, power, phase and frequencyArray.

Parameters **`fs`** (*`double`*) – Sampling Frequency, normally 2.5Ghz.

Returns StreamingClass object with normalizedPower, power, phase, and frequencyArray calculated.

`Matlab.Streaming.classDesign.@StreamingClass.`**`defineInputs`**(*this*)

Creates the input matrix to be used for training any model. It changes the input parameter inside the StreamingModel object.

Returns A StreamingClass object with defined inputs stores at the StreamingModel object.

Returns The matrix used to feed the model.

`Matlab.Streaming.classDesign.@StreamingClass.`**`reportStreaming`**(*this*, *language*)

Creates several figures to report a StreamingClass object

Parameters **`language`** (*`String`*) – Allows the user to select the language (not working).

Returns A struct holding all figure handles.

1.2 Model Class

1.2.1 Main Parameters

`class` `Matlab.Streaming.classDesign.@StreamingModel.`**`StreamingModel`**(*input*, *target*)

StreamingModel class, used to hold any important variables to train a neural network model. With the frequency data from the correlation analysis.

Parameters

- **`target`** (*`double Array`*) – Target variable for training.
- **`input`** (*`double Array`*) – Input variable for training.
- **`corrStruct`** (*`struct`*) – Structure containing the results from the correlation analysis.
- **`frequencyDivisions`** (*`double Array`*) – Relevant frequencies (from the correlation analysis).
- **`indexesChosenFrequencies`** (*`bool Array`*) – Logical array for the chosen frequencies.

- **`figHandles`** (*`Handle`*) – Figure handles from the correlation analysis.
- **`frequencyArray`** (*`double Array`*) – Array of frequencies.
- **`trainedModel`** (*`struct`*) – Contains all information regarding the trained model (like confusion matrices).
- **`variables`** (*`Cell Array`*) – Strings that specify which acoustic emission parameters were used.

1.2.2 Constructor

`StreamingModel.`**`StreamingModel`**(*input*, *target*)

1.2.3 Methods

`Matlab.Streaming.classDesign.@StreamingModel.`**`trainModel`**(*this*, *includeIndexes*, *neuralNetStructure*)

`Matlab.Streaming.classDesign.@StreamingModel.`**`corrAnalysis`**(*this*, *inputVariable*, *variableString*)

`Matlab.Streaming.classDesign.@StreamingModel.`**`corrAnalysisChannel`**(*this*, *inputVariable*, *variableString*, *channel*, *chArray*)

1.3 Wave Class

The wave class contains all acoustic emission parameters and a method that calculates those parameters using the acoustic emission waveform data.

class `Matlab.Streaming.classDesign.@Wave.`**`Wave`**(*varargin*)

Class that contains an acoustic emission wave and its parameters

Parameters

- **`rawData`** (*`int16 array`*) – Array that stores the entire wave, the values have no phisical meaning.
- **`channel`** (*`uint8`*) – Channel where the wave was captured from.
- **`riseTime`** (*`double`*) – Time between triggerTime and the Maxmimum Amplitude in seconds.
- **`count`** (*`double`*) – Amount of times the wave has positively crossed over the Threshold.
- **`energy`** (*`double`*) – Energy of wave,
- **`duration`** (*`double`*) – Duration of the wave, in seconds.
- **`rms`** (*`double`*) – RMS energy of the wave.
- **`maxAmplitudeDB`** (*`double`*) – Maximum amplitude of the wave, in dB.
- **`resolutionLevelCount`** (*`uint16`*) – Amount of levels used to digitilize the wave.
- **`averageSignalLevel`** (*`double`*) –
- **`countToPeak`** (*`double`*) – Amount of times the wave has positively crossed the Threshold until the Max. Amplitude time.
- **`averageFrequency`** (*`double`*) – Average Frequency of the wave.
- **`reverberationFrequency`** (*`double`*) – Reverberation Frequency of the wave.
- **`initiationFrequency`** (*`double`*) – Initiation Frequency of the wave.
- **`maxAmplitude`** (*`int16`*) – Maximum amplitude of the wave.
- **`threshold`** (*`double`*) – The threshold used to capture the wave (3*noiseLevel of the file).
- **`meanAmplitude`** (*`double`*) – Mean of the Amplitude of the wave.
- **`absoluteTriggerIndex`** (*`double`*) – The index (with respect to the file) where the trigger first ocurred.

- **`triggerTime`** (*`uint32`*) – Absolute time of the entire test when the wave was capture, in seconds.
- **`relativeTriggerIndex`** (*`double`*) – Always 5000 (irrelevant parameter)
- **`splitFile`** (*`Bool`*) – A Boolean that informs if the wave was capture between files.
- **`splitIndex`** (*`double`*) – The index on the latter file (from the split) where the wave ended.
- **`file`** (*`double`*) – Which file the wave was captured from.

1.3.1 Constructor

`Wave.`**`Wave`**(*varargin*)

Wave class constructor, receives a variable-size input, however only instantiates 6 inputs.

Parameters **`varargin`** (*`Cell Array`*) – All necessary parameters to instantiate a Wave Object. In order: rawData, channel, threshold, triggerTime, absoluteTriggerIndex, relativeTriggerIndex.

1.3.2 Methods

`Matlab.Streaming.classDesign.@Wave.`**`calculateParameters`**(*wave*, *fs*, *streamingClass*)

Method that calculates the AE wave parameters according to PAC Manual

Parameters

- **`fs`** (*`double`*) – Sampling frequency, usually $fs=2.5Ghz$.
- **`streamingClass`** (*`streamingClass`*) – An instance of streamingClass.

Returns An Wave object with calculated parameters.

1.4 User Guide

This user guide contains only an example script that loads the tests data and trains a simple neural network on top. It is meant to give an extremely simple way to start using the developed code.

```matlab
% A basic example for the Petro Acoustic Emission work
%
% Loads the complete data from the 3 tests and
% trains it with a simple neural network with 10 neurons.
%
% This script is meant to be a starting point for
% anyone wanting to further indulge in the projects
% usage.

load('streamingOBJCP2.mat') % Loads CP2 object.
load('streamingOBJCP3.mat') % Loads CP3 object.
load('streamingOBJCP4.mat') % Loads CP4 object.
load('frequencyDivisions.mat') % Loads frequencyDivisions variable.

% Script that does pre-calculations like FFTs,
% a few wave removals, etc.
prepareCPS;

% Cell array with each CP.
cps_cell = {CP2, CP3, CP4}

% Iterates each CP
for cp_index = cps_cell:

    % Uses the frequency ranges from 'frequencyDivisions'
    % to create the frequency variables from
    % the normalized power spectrum.
    cp.StreamingModel.frequencyDivisions.normalizedPower = ␣
↪frequencyDivisions;
    [cp, ~] = cp.defineInputs();

    % Trains the model using a simple one-layered
    % neural network with 10 neurons.
    cp.StreamingModel = cp.StreamingModel.trainModel([],[10]);

    % Calculates the mean of the confusion matrix
    % in percentage form.
    100*mean(cp.StreamingModel.trainedModel.confusionMatrix.
↪percentValidation,3)

    % Calculates the standard deviation of the confusion matrix
    % in percentage form.
    100*std(cp.StreamingModel.trainedModel.confusionMatrix.
↪percentValidation, [], 3)

end
```

MATLAB MODULE INDEX

Appendix B

Transition Time Estimation "Proof"

First and foremost, this is not a complete proof of the method, this only proves two things:

- $E[n_0] > 0$, $E[n_f] > 0$ asymptotically.
- $E[n_t] = 0$.

Where $E[n]$ is any metric of classification error (MSE, categorical cross entropy, etc) at iteration n. n_t, n_0 and n_f are the transition, initial and final iteration of the algorithm. For all intended purposes, the two distributions have no overlap in all dimensions, that is, they are LS.

By definition, a PLA has 0 classification error if the data presented to it is LS. With this, we have by definition:

$$E[n_t] = 0 \tag{B.1}$$

Because when $n = n_t$, the window has $L/2$ samples from each class, and by feeding it to a PLA, the error is always 0. However, when the window has only samples from one class, there is still a chance that the error is 0, given that the points happened to spread themselves around a line. Therefore the first objective is to prove:

Lemma 1. *Given a limited distribution f, the probability that two subsets of $N/2$ points taken from the f are linearly separable is bounded by $P = 1/2^{N-1}$.*

Proof. The only two binary sets capable of being separated by a line would be $N/2$ zeros followed by $N/2$ ones, or vice-versa. This gives a probability of:

$$P = 2P_0^{N/2}P_1^{N/2}, \tag{B.2}$$

where P_0 is the probability of being classified as Class 0, P_1 is analogous. Knowing that $P_0 + P_1 = 1$ one can reach:

$$P = 2P_0^{N/2}(1 - P_0)^{N/2} \tag{B.3}$$

In order to find the maximum value for this equation, one derives in respect with P_0, making $N/2 = k$ one has:

$$\frac{\partial P}{\partial P_0} = 2kP_0^{k-1}(1 - P_0)^k - 2kP_0^k(1 - P_0)^{k-1} \tag{B.4}$$

Setting Equation B.4 equal to zero yields:

$$\begin{aligned} 2kP_0^{k-1}(1 - P_0)^k &= 2kP_0^k(1 - P_0)^{k-1} \\ P_0^{-1} &= (1 - P_0)^{-1} \\ P_0 &= 1 - P_0 \\ P_0 &= \frac{1}{2} \end{aligned} \tag{B.5}$$

Therefore, the maximum probability, $P_{\max}$ is given when $P_0 = P_1 = 1/2$, which applying to Equation yields:

$$\begin{aligned} P_{\max} &= 2\left(\frac{1}{2}\right)^{N/2}\left(\frac{1}{2}\right)^{N/2} \\ P_{\max} &= 2\left(\frac{1}{2}\right)^{N} = \frac{1}{2^{N-1}} \end{aligned} \tag{B.6}$$

And, by definition, $P_{\max} > P$. $\square$

Therefore, as long as the window is big enough, the probability that $E[n] > 0$ for all n given that:

$$C[n + k] = i \quad | \quad \frac{-L}{2} \leq k \leq \frac{L}{2}, i = 0 OR i = 1 \tag{B.7}$$

which means that, as long all samples within the window's boundary are from the same binary class i.

This means that, given a sufficiently large L, as long as $P/2 > L$ the probability of having no initial error is asymptotically close to 0. If one has $L = 50$, $P(E[n_0] = 0) \approx 8.9e - 16$. This implies that the graph should look like a funnel (Figure 2.17). The problem is that, by the writing of this work, there is no proof that the error monotonically descends to 0. One can however, try to explain it by induction and considering how the gradient changes.

Given a model with parameters θ, one can define an update rule like:

$$\theta[k] = \theta[k-1] - \eta\frac{\partial\theta}{\partial\xi_i}, \tag{B.8}$$

Where ξ_i is the error prevenient from sample i. This error can be defined as the distance from the model's output and the target's label. If we suppose that the θ space is convex, this update rule will always reach the best θ possible, θ^*. However, if we suddenly start changing correct labels to wrong labels, the gradient will change, and this update rule will make θ move further away from θ^*.

So this means that, as long as the window has a gradual increase of correct labels inside it, the error should decrease accordingly. This means that having a set of samples like 000000—001111 is better than 000000—000011. Simply because the former has more samples with the correct labelling, and the gradient will be "more correct" than it would when comparing with the latter case.