Introducing MLOps Practices in the SPIRA Continuos Training Pipeline

Daniel Angelo Esteves Lawand

Capstone Project
presented to the discipline
MAC0499

Supervisors:
Prof. Dr. Alfredo Goldman vel Lejbman
MSc. Renato Cordeiro Ferreira

São Paulo, December of 2023

Contents

Chapter 1

Introduction

precisa de uma introducao anterior contextualizando o spira trabalho do edresson eh experimental, e o nosso eh produtizar. (Uma explicação válida para o início do TCC) Until now all the others works were about creating new functionalities from scratch for SPIRA. This work is about "refactor" and to make viable the edresson's work. So the challenge is from understanding what other author did, and redo it in a more viable-production way.

Chapter 2

MLOps

Chapter 3

Neural Networks

3.1 Definition

Neural networks are a computational technique with the goal of learning patterns from data. Artificial neural networks came from the first works that attempted to model networks of neurons in the brain (McCulloch and Pitts, 1943). As any other machine learning model, a neural network receives a set of input data, processes it, and generates a set of outputs – which can be numbers or categories. Figure 3.1 shows a schematic representation of an artificial neural network.

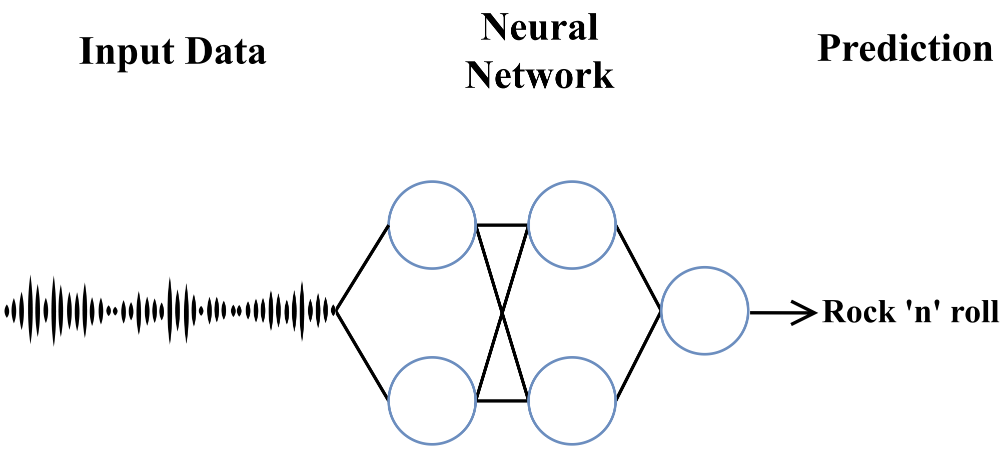


Figure 3.1: ***Schematic representation of an artificial neural network.*** *In this example, it receives a sound wave sampled from a song and classifies its genre in a predefined category.*

3.1.1 Unit

Neural networks are constituted by a set of nodes. Each node is called a **unit** or a **neuron**. Each unit has at least one connection to another unit. A neuron is a function that takes a set of real valued numbers as input, performs some computation on them, and produces an output.

This computation is a weighted sum of its inputs with one additional term known as the **bias term**. It allows the result to be nonzero even when the inputs are zero. Afterwards, the unit applies an **activation function** (often non-linear), resulting finally in its outputs. The data flows through the network from the initial inputs to the final outputs.

To illustrate how the computation in an unit is calculated, take the example of the **perceptron** shown in Figure 3.2, which is a single-unit neural network for linear classification that results in a binary output.

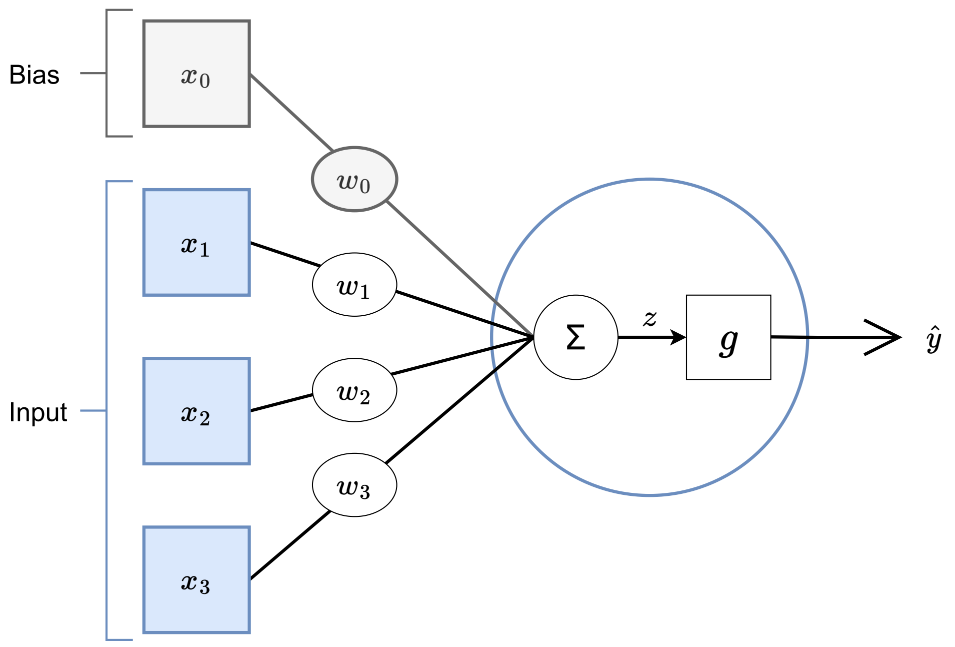


Figure 3.2: ***Perceptron:*** *The neuron unit takes three inputs x_1, x_2 and x_3, and the bias term x_0. For each input, it has corresponding weights w_1, w_2 and w_3, as well a weight w_0 for the bias term. The output of the weighted sum is z, which is fed into the activation function g then producing the output $\hat{y}$.*

The weighted sum can be represented as:

$$z = \sum_{i=0}^{n} x_i \cdot w_i. \tag{3.1}$$

After the calculation, the neuron applies its activation function g to the result z. For the perceptron, the output of the neuron is also the final output of the neural network, denoted by $\hat{y}$. Therefore,

$$\hat{y} = g(z), \tag{3.2}$$

and the perceptron can be summarized as:

$$\hat{y} = g\left(\sum_{i=0}^{n} x_i \cdot w_i\right). \tag{3.3}$$

3.1.2 Layers

Neural networks can have more than one unit, which can be organized into **layers**. A **feed-forward neural network** (FNN) is a multi-layer network. It has connections only in one direction, i.e., the outputs from one layer become inputs to the successor layer. Therefore, there is no feedback to predecessor layers and the network's architecture forms a directed acyclic graph.

A feed-forward neural network is also called a **multi-layer perceptrons**. However, units in a FNN are not perceptrons, since the units in a FNN have non-linear activation functions whereas perceptrons have linear activation functions.

Commonly, a FNN is also a **fully connected network**, meaning that a inputs of a unit in the layer k come from the outputs of all the units in the predecessor layer $k-1$. Therefore, every pair of units from two adjacent layers is connected.

There are three kinds of layers:

- **Input layer**: It is the set of **input units**, denoted by $\boldsymbol{x}$. Each unit j in this layer has a scalar value x_j that represents a **feature** of the input data. All the input units pass their data to the units in the next layer.
- **Hidden layer**: It is a set of **hidden units**, denoted by $\boldsymbol{h_i}$, where $1 \leq i \leq n$, and n is the number of hidden layers. Each unit j in a hidden layer i is represented as $h_{i,j}$. Together, the hidden units compute the intermediate outputs of an artificial neural network.
- **Output layer**: It is the set of **output units**, denoted by $\boldsymbol{\hat{y}}$. Each unit j in this layer computes the final output $\hat{y}_j$, also known as a **prediction**. It can be a real valued number, or some sort of classification.

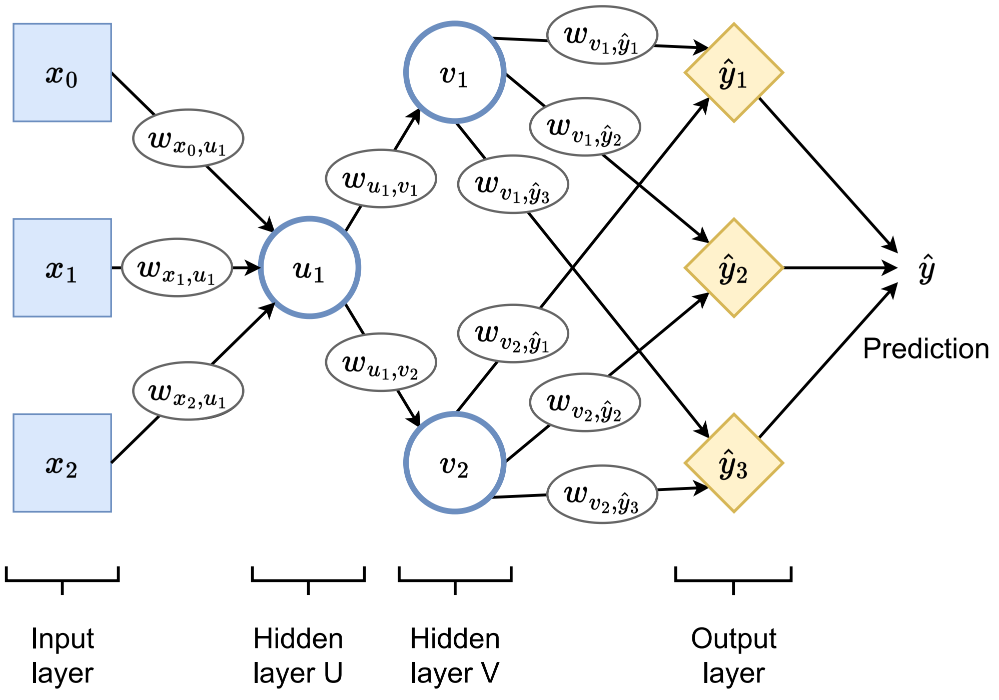


Figure 3.3: ***A three layer FNN*** *with one input layer, two hidden layers, and one output layer. The input layer is usually not counted when enumerating layers. The hidden layers were represented as* $\boldsymbol{u}$ *and* $\boldsymbol{v}$ *instead of* $\boldsymbol{h_1}$ *and* $\boldsymbol{h_2}$ *to simplify the image.*

Units in the input layer are represented in a different format due to their different behavior from other units in the feed-forward neural network. Input units don't do any computation. They only store scalar values derived from the raw data (features).

Both hidden and output units make a weighted sum of their inputs and apply an activation function, resulting in the output. The outputs of a hidden unit are the input for other units. These associations are shown at Figure 3.3.

Consider now a neural network similar to Figure 3.4. Let's generalize the computation made for hidden and output units:

- **First hidden layer computation**: Let $a_{1,j}$ denote the output of the hidden unit $h_{1,j}$ in the first hidden layer. Let $w_{x_k,h_{1,j}}$ be the weight between units x_k and $h_{1,j}$. Let $g_{1,j}$ denote the

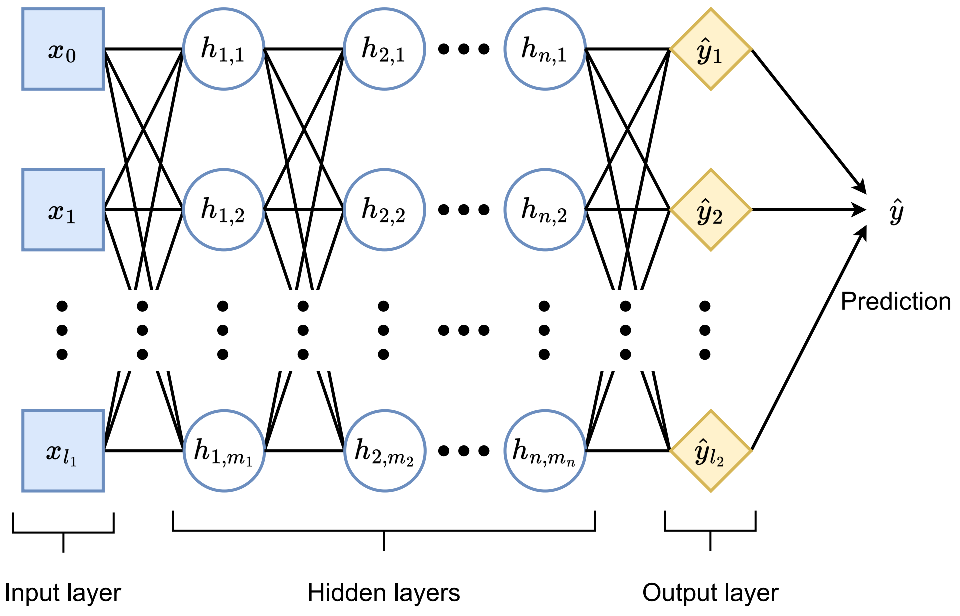


Figure 3.4: ***A generic FNN*** *with one input layer, n hidden layers, and one output layer. The weights were hidden in the image to simplify it.*

activation function associated with the unit $h_{1,j}$. Then we have:

$$a_{1,j} = g_{1,j}\left(\sum_k x_k \cdot w_{x_k,h_{1,j}}\right) \equiv g_{1,j} \cdot z_{1,j}, \tag{3.4}$$

where $0 \leq k \leq l_1$ indicates each input unit, and $1 \leq j \leq m_i$ indicates each unit in the hidden layer $i = 1$.

- **Other hidden layers computation**: Let $a_{i,j}$ denote the output of the hidden unit $h_{i,j}$. Let $w_{h_{i-1,d},h_{i,j}}$ be the weight between units $h_{i-1,d}$ and $h_{i,j}$. Let $g_{i,j}$ be the activation function associated with the unit $h_{i,j}$. Then we have:

$$a_{i,j} = g_{i,j}\left(\sum_i a_{i-1,d} \cdot w_{h_{i-1,d},h_{i,j}}\right) \equiv g_{i,j} \cdot z_{i,j}, \tag{3.5}$$

where $2 \leq i \leq n$ indicates which hidden layer that unit belongs, $1 \leq j \leq m_i$ indicates each unit in the hidden layer i. And, $1 \leq d \leq m_{i-1}$ indicates each unit in the hidden layer $i - 1$.

- **Output layer computation**: The outputs of the last hidden layer $\boldsymbol{h_n}$ are inputs for the units in the output layer.

Let $\hat{y}_k$ denote the output of the $k-$th unit in the output layer. Let $h_{n,j}$ denote a hidden unit in the last hidden layer $\boldsymbol{h_n}$. Let $a_{n,j}$ denote the output of unit $h_{n,j}$. Let $w_{h_{n,j},k}$ denote the weight between the $h_{n,j}$ and the $k-$th unit in the output layer. Let g_k denote the activation function associated with the kth unit in the output layer. Then we have:

$$\hat{y}_k = g_k\left(\sum_j a_{n,j} \cdot w_{h_{n,j},k}\right) \equiv g_k \cdot z_k, \tag{3.6}$$

where $1 \leq k \leq l_2$ indicates each output unit, $h_{n,j}$ is a unit in the last hidden layer $\boldsymbol{h_n}$, and $1 \leq j \leq m_n$ indicates each unit in the hidden layer n.

The arrangement of neurons into layers allows a neural network to derive features from its input, thus enabling it to identify patterns in its training data. Neural networks have the ability to approximate complex functions and classify data across an array of domains, such as image classification, natural language processing, speech recognition, and others.

When a neural network has many hidden layers, it is known as **deep neural networks**. The field of **deep learning** is dedicated to study and apply these neural networks.

3.1.3 Training and optimization process

In **supervised learning**, a neural network is trained using the **back-propagation** algorithm. This algorithm adjusts the values of the neural network's weights. These weights are **parameters** used to make predictions. For that, it relies on **labeled** data. The algorithm has the following steps:

1. **Weights initialization**: The weights in a neural network are initialized randomly.
2. **Forward**: Input data is fed forward through the network, from the input layer to the output layer.
3. **Loss computation**: Compare the network's predictions to the ground truth labels, calculating a value known as **loss**. The choice of loss function depends on the problem, as detailed below.
4. **Backward**: For each layer, starting from the output layer to the input layer (in the reverse direction of the forward step), compute the gradient of the loss with respect to the weights of that layer. Update the weights using an optimization algorithm.
5. **Repeat**: Repeat steps 1 to 4 for a specified number of iterations (**epochs**), or until the loss converges.

Figure 3.5 shows the training steps of a neural network following the back-propagation algorithm.

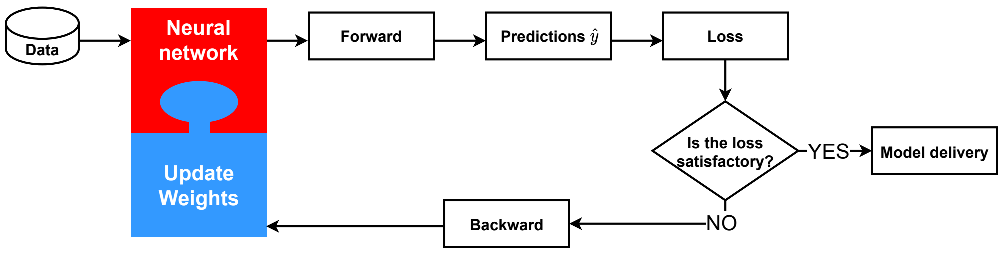


Figure 3.5: ***Training steps of a neural network.***

During training, the goal is to minimize the loss function, which improves the neural network's overall predictive **accuracy**. The **loss function** is a mathematical function that indicates the magnitude of the error that the neural network has made in its prediction (Raff, 2022; Russell and Norivg, 2021; Trochim, 2017). There are different techniques for reducing the loss function. One of them is the gradient descent. Furthermore, there are different types of loss functions.

The one used in the SPIRA project is know as **cross-entropy** (Casanova et al., 2021). It goes beyond the scope of this research to cover the details of minimizing the loss function. For more information, refer to Russell and Norivg.

3.2 [WIP]Convolutional Neural Networks

Consider a feed-forward neural network that receives an image as input. if the image with n pixels was represented as a vector, and the neural network have n units in the first hidden layer. Supposing the neural network is fully connected between the input and the first hidden layer, it would have n^2 weights between them. Which means for a typical megapixel RGB image, it would have nine trillion weights. The parameter space of such magnitude would need correspondingly a large quantities of training images and a massive computational budget to run the training algorithm.

there are two issues about this approach 1 - pesos 2 - adjacencia

These considerations suggest that we should construct the first hidden layer so that each hidden unit receives input from only a small, local region of the image.

This kills two birds with one stone.

First, it respects adjacency, at least locally. (And we will see later that if subsequent layers have the same locality property, then the network will respect adjacency in a global sense.)

Second, it cuts down the number of weights: if each local region has l n pixels, then there will be ln n 2 weights in all.

3.3 [WIP] PyTorch

PyTorch (Referência de PyTorch?) is a Python (referencia de python?) library for Deep Learning.

Building a Convolutional Neural Network in

Chapter 4

Objectives

Chapter 5

Architecture

To a clearer communication, the SPIRA architecture made by the research (Casanova et al., 2021) will be called *SPIRA v1*. The architecture proposed by this research will be called *SPIRA v2*.

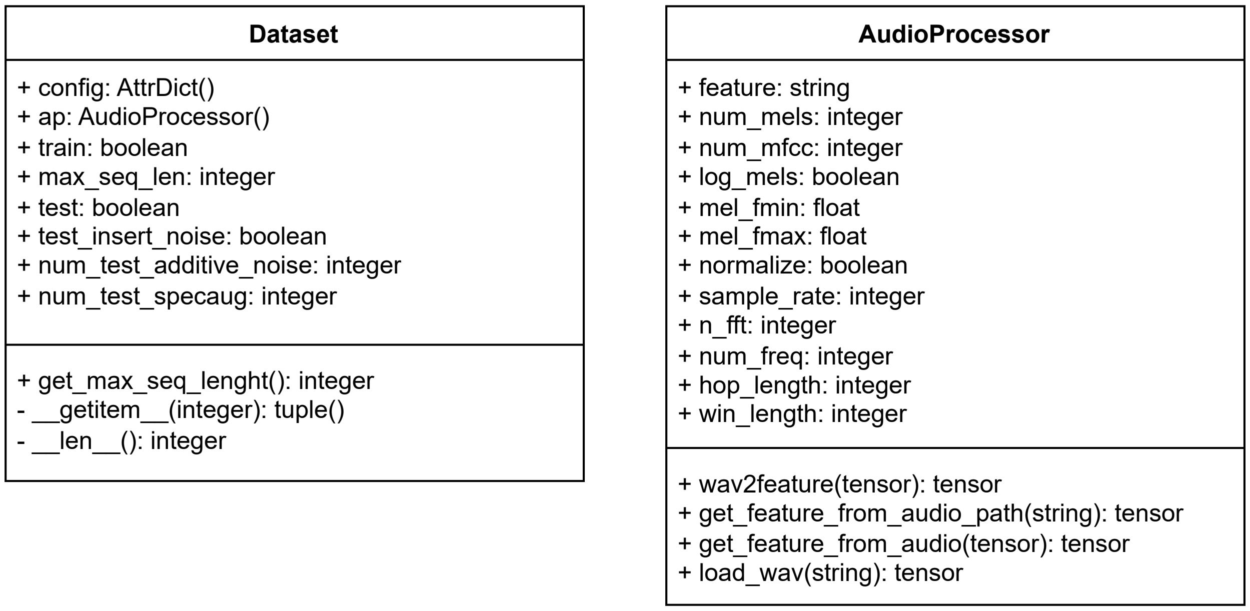
This chapter presents the architecture proposed in this research. Chapter 3 will discuss what are the issues with *SPIRA v1* and how the proposed architecture deals with them. Afterward, Chapter 7 will discuss about the consequences of the suggested changes.

5.1 Old Architecture

It was thought to provide a visualization of *SPIRA v1* system, however the system structure is not clear. So it will be provide the class Diagram 5.1 that shows its key components.

It is possible to see classes with a lot of attributes.

Diagram 5.1 *SPIRA v1* Class Diagram.



5.2 New Architecture

The next sections describe the *SPIRA v2* system using the **C4 model** (Brown, 2018). Four groups of abstractions describe the structure of the system, from lower to higher level of detail: **context**, **containers**, **components** and **code**.

5.2.1 Context

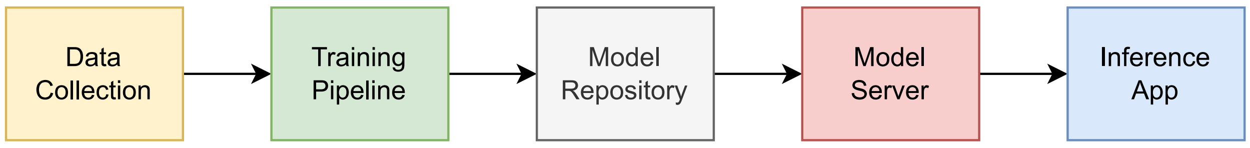
The system context diagram is the highest level of abstraction. It shows the system in the context of other systems or tools. It gives a view of the big picture of the system landscape. The SPIRA system aims to be a tool for the insufficiency respiratory triage in hospitals used by health professionals. The system is not ready to be used, but professionals in different areas are working to put it into production. Since this is a consumer centered product that does not rely on external system, the context diagram is trivial.

5.2.2 Containers

According to Brown, the system containers give us a "high-level shape of the software architecture and how responsibilities are distributed across it" (Brown, 2018). Diagram 5.2 shows the system containers. SPIRA has five major services:

- **Data Collection**: is the service used to collect data from patients and store it in SPIRA's data lake (Housley and Reis, 2022; Kleppmann, 2017).
- **Training Pipeline**: is the service that creates a neural network model artifact from the collected data. This research focus on this part.
- **Model Repository**: is the service that stores model artifacts and manages their versions (Lakshmanan, 2021).
- **Model Server**: is the service that uses a model artifact stored in model repository and makes the inference (Lakshmanan, 2021).
- **Inference App**: is the service that receives data, communicates with the model server, asks the server to make an inference using a message-driven request response communication protocol (Boner, 2016).

Diagram 5.2 SPIRA Container.

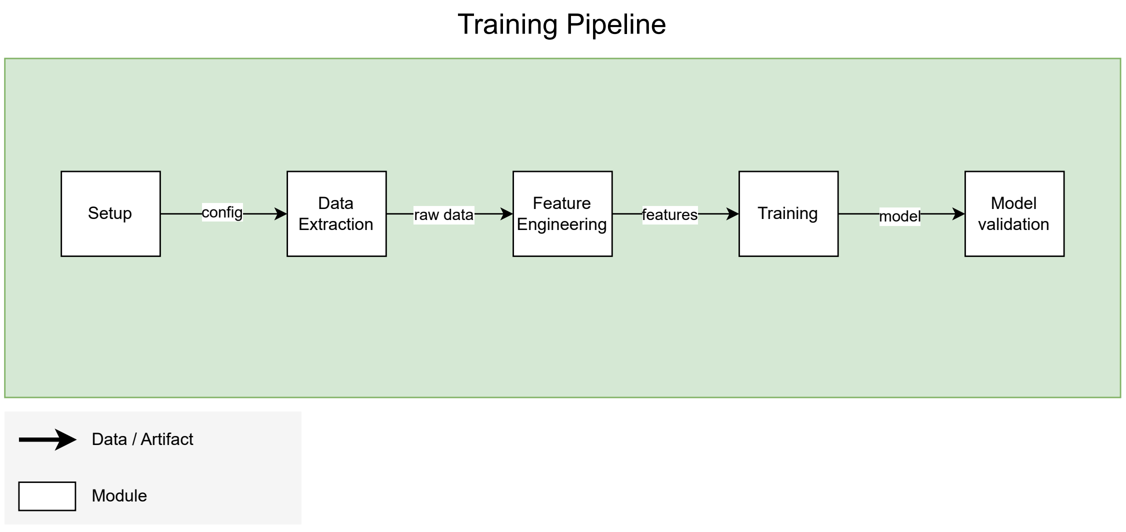


5.2.3 Components

The system components give more detailed information of a container, showing the different components in it. This research focus on the training pipeline. Diagram 5.3 shows the training pipeline components. It's components are:

- `setup`: it loads user defined configurations for the training pipeline.
- `data extraction`: it extracts the raw data obtained by the data collection app.
- `feature engineering`: it applies transformations (such as data processing and data augmentation) in the raw data to produce features.
- `training`: it uses the features to learn the parameters of the convolutional neural network.
- `model validation`: it receives a trained model and evaluates metrics regarding its accuracy.

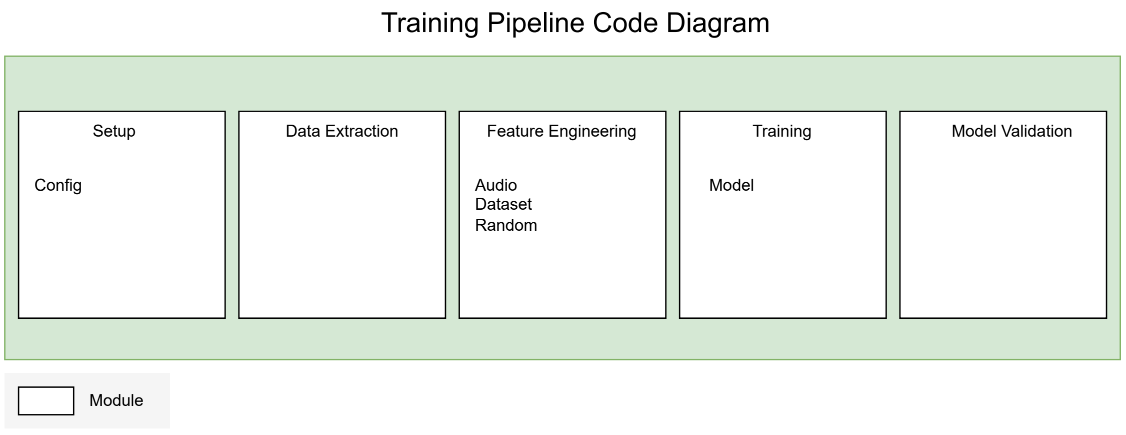
Diagram 5.3 Training Pipeline Component.



5.2.4 Code

For simplicity, this research will not cover all details of each component above. In turn, it will showcase the key classes that compose the architecture, as shown in Diagram 5.4.

Diagram 5.4 Training Pipeline Code.



- `Config`: It encapsulates the configuration in the configuration setup.

- `Audio`: It encapsulates the raw data, which is an audio file format that represents the digital sampling of an audio wave.
- `Dataset`: It encapsulates the set of features and labels.
- `Random`: It encapsulates the randomness.
- `Model`: It encapsulates the process of creating, training and predicting with a CNN model.

Chapter 6

Bad Smells

Chapter 7

Results

Chapter 8

Analysis

Chapter 9

Conclusões e considerações futuras

Bibliography

Jonas Boner. **Reactive Microservices Architecture Design Principles for Distributed Systems**. 2016. 11

Simon Brown. The C4 model for visualising software architecture. **Infoq.Com**, 2018. 11

Edresson Casanova, Lucas Gris, Augusto Camargo, Daniel da Silva, Murilo Gazzola, Ester Sabino, Anna S. Levin, Arnaldo Candido, Sandra Aluisio, and Marcelo Finger. Deep Learning against COVID-19: Respiratory Insufficiency Detection in Brazilian Portuguese Speech. In **Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021**, 2021. doi: 10.18653/v1/2021.findings-acl.55. 8, 10

Matt Housley and Joe Reis. **Fundamentals of Data Engineering**. O'Reilly Media, 6 2022. 11

Martin Kleppmann. **Designing Data-Intensive Applications: The Big Ideas behind Reliable, Scalable, and Maintainable Systems**. 2017. 11

Valliappa Lakshmanan. **Machine learning design patterns : solutions to common challenges in data preparation, model building, and MLOps / Valliappa Lakshmanan, Sara Robinson, and Michael Munn.** 2021. 11

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. **The Bulletin of Mathematical Biophysics**, 5(4), 1943. ISSN 00074985. doi: 10.1007/BF02478259. 3

Edward Raff. **Inside Deep Learning**. 2022. 7

Stuart Russell and Peter Norivg. **Artificial Intelligence: A Modern Approach (Global Edition)**. 2021. 7, 8

Piotr Trochim. What is a loss function in simple words?, 2017. URL https://stackoverflow.com/questions/42877989/what-is-a-loss-function-in-simple-words?rq=4. 7