João Paulo Ramos Carrasco Ferreira

Serviços Web para Aprendizagem Automática

Web Machine Learning Services

João Paulo Ramos Carrasco Ferreira

Serviços Web para Aprendizagem Automática

Web Machine Learning Services

"*Cloud is the digital wonderland of Internet of Things, powered by Artificial Intelligence and Big Data.*"

— Enamul Haque

Universidade de Aveiro
2021

João Paulo Ramos Carrasco Ferreira

Serviços Web para Aprendizagem Automática

Web Machine Learning Services

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica da Professor Carlos Costa, Professor Associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Professor Doutor Augusto Marques Ferreira da Silva
Professor Associado

vogais / examiners committee

Professor Doutor Carlos Manuel Azevedo Costa
Professor Associado

Professor Doutor Rui Pedro Sanches de Castro Lopes
Professor Coordenador

agradecimentos /
acknowledgements

Em primeiro lugar, quero agradecer à minha família, em especial aos meus pais, avós e à minha irmã, não só por me ajudarem a tomar decisões a nível profissional, mas também por me apoiarem sempre ao longo da minha vida.
Um grande agradecimento a todos os professores que viram potencial em mim ao longo dos anos na universidade, em especial ao professor Carlos Costa por acreditar e apostar nas minhas capacidades, por me guiar no processo da construção desta dissertação e me ajudar a chegar tão longe.
Agradecimentos a todos os meus amigos que conheci durante este percurso, especialmente à Mariana Rodrigo e ao Gonçalo Vítor.
Muito obrigado a todos os que diretamente e indiretamente contibuíram para a conclusão do meu mestrado.

Palavras Chave aprendizagem profunda, redes neuronais, sistemas distribuídos, monitorização, aprendizagem automática, classificação de imagem

Resumo

A utilização de deep learning tem vindo a aumentar em diferentes áreas aplicacionais devido à sua eficiência em processos de previsão e à capacidade de se adaptar a vários tipos de dados. No entanto, a sua complexidade matemática e as otimizações dos processos de treino são desafiantes. Mais ainda, o acesso a hardware dedicado e a configuração correcta de um ambiente de trabalho são muitas vezes barreiras à sua utilização. Esta dissertação propõe uma arquitetura e descreve a implementação de uma plataforma web de serviços de deep learning para utilização de estudantes e investigadores. Esta plataforma permite abstrair os processos complexos que suportam esta tecnologia, simplificando e democratizando o seu uso. Trata-se de uma solução distribuída e multiutilizador que oferece serviços, desde hardware a software, através de um navegador web comum. A solução permite a adição fácil de unidades computacionais de processamento, a inserção de dados de estudo, o desenho dos modelos com ferramentas visuais, a monitorização de sessões de treino, teste e validação destes modelos.

Keywords deep learning, neural networks, distributed systems, monitorization, machine learning, image classification

Abstract

Deep learning has become increasingly popular over the years, having proved their efficiency in input-output functions for distinct types of data. However, their mathematical complexity and the training optimizations can be challenging. Moreover, access to dedicated hardware and the setup of a working environment is most of the times a barrier to its usage. This dissertation describes the design and implementation of a web platform to be used by students and advanced researchers. This platform aims to abstract the processes behind this technology and simplify its usage. It is a multi-user distributed platform that offers services, software and hardware, through a common web browser interface. The solution allows easy addition of new computational nodes, the upload of datasets, the visual design of models and datasets, the monitoring of hardware and training sessions, and also supports validation and test procedures.

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
CNN	Convolution Neural Network
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CSV	Comma Separated Values
CUDA	Compute Unified Device Architecture
DBMS	Database Management System
DL	Deep Learning
DNN	Deep Neural Network
DOM	Document Object Model
DoS	Denial of Service
GAN	Generative adversarial network
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IO	Input-Output
JSON	JavaScript Object Notation
JWT	Json Web Token
ML	Machine Learning
NLU	Natural Language Understanding
px	pixel
RGB	Red-Green-Blue color model
rem	root em
RNN	Recurrent Neural Network
SQL	Structured Query Language
SSD	Solid State Drive
SSR	Server Side Render
SVG	Scalable Vector Graphics
UI	User Interface
URL	Uniform Resource Locator
WWW	World Wide Web
WebML	Web Machine Learning

1

Introduction

This chapter describes the context and the motivation that supported this work and the writing of the current text. The main goals of this work are covered, as well as the document structure.

1.1 Context and Motivation

Nowadays, massive amounts of data are being generated daily by all areas of science and industry. This creates a need for novel machine learning and artificial intelligence methods capable of using this amount of data for analysis. Deep Learning (DL) is one such method that can go beyond the coverage of linear programming [1]. Having grown to be one of the main components of contemporary research in artificial intelligence, DL has had its impact felt in various scientific fields [2], [3]. Some of these fields where DL architectures have been applied include speech recognition, Natural Language Understanding (NLU), acoustic modeling, image recognition, game development, computational biology and cancer screening, having, in some cases, produced results that surpass the human level capability and performance [1], [3]–[5].

Deep learning has the world's leading economies and technology companies racing to advance this technology (Fig. 1.1). The required skills in advanced data science, however, are relatively hard to come by and even for highly trained professionals it can be discouraging to approach the rapidly increasing body of knowledge in the field [2], [3].

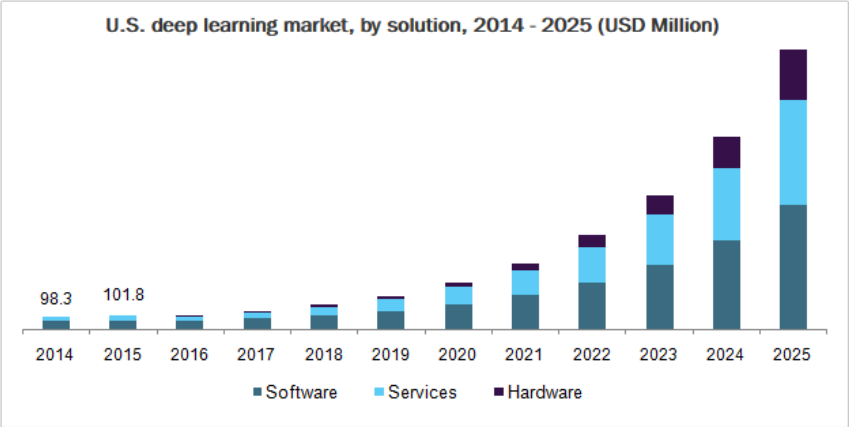

Figure 1.1: Illustration of the AI market growth [6]

There is a need to start to both simplify and accelerate AI. In the past, there was a time where many programmers were needed to build and develop machine learning algorithms. In the future, there will be a need to use these algorithms, instead of designing them.

With the advent of technology, many processes have been modified in human life. Nowadays, we live in a society that produces and stores data massively. As a consequence, machine learning is finding more and more applications in almost every sector. This creates a need to visualize and easily operate these new machine learning mechanisms. This need can be fulfilled by exploring and taking advantage of complexity abstraction. Abstraction is very important for the evolution and development of neural networks to cope with the complexity underneath them. This complexity is assembled from non-elementary math, hardware drivers, updated software and also a compatible working environment.

Over the recent years, big corporations such as Microsoft and Google invested greatly in the development of graphical and non-graphical interfaces related to deep learning techniques. Their focus is to ease the difficulty of Artificial Neural Network (ANN) models deployment to the users that want to use it or that simply want to learn more about the subject. There has been an increasing demand for user-friendly graphical interface services that entail DL. Developing an easy-to-use Graphical User Interface (GUI) masking DL complexity under the hood is rewarding. It offers the opportunity for the users to take less time in engineering their models for their specific use cases and be able to visually analyze their performance in real datasets.

1.2 Objectives and Limitations

The platform developed alongside this dissertation intends to deliver a multi-user, multi-privilege and user-friendly interface to increase user productivity of the training, prediction and evaluation of neural network models. The developed platform is named Web Machine Learning (WebML). The WebML interface is a web solution that should be accessed through the browser for the highest compatibility between devices and also to avoid version incompatibilities found in regular downloadable applications. WebML is also responsible for efficient use of the remote hardware and its real-time monitoring. The hardware details and service implementation should be completely abstract for the common user. The user should only focus on his own DL tasks using the provided User Interface (UI).

Graphics Processing Units (GPUs) are usually the most expensive hardware and it is valuable to have the highest quality when compared to the other computational components. Since there is a maximum number of GPUs and Central Processing Units (CPUs) a computation machine can handle, WebML should also be able to function with multiple hardware provider terminals and be easily expandable to new equipment. This behavior allows the platform to serve multiple users in parallel without interfering with each other until the hardware or network limit is blocked by a bottleneck. WebML was built with the intention to behave as flexible as possible to unpredictable problems.

The focus of this project is on the user interface interactivity and innovation while coping with the weakness of the current state of art projects (Section 2.9). The scope of the platform was chosen regarding the requirements and possibilities to deal and solve common and non-common use-cases in better terms than the competition.

In summary, the goals of the dissertation are the following:

- Real-time monitored use of dedicated hardware for neural network training;
- Optimized user-interaction and smart design to provide an intuitive and easy to use web platform for machine learning experts and for users without prior knowledge on DL;
- Robust distributed system to enable a great redundancy level to provide fail-safe mechanisms and security;
- Detailed logging system regarding all the user interactions, errors and performance indicators for future analysis using machine learning.

These listed goals were mandatory, however, there is a set of sub-objectives that can improve WebML greatly. These sub-objectives include, for example, the increase of compatibility to the many problems neural networks can solve.

Due to the fixed time frame, some limitations had to be set on this project to ensure the final product was robust. A project like WebML is never actually finished, as it can be adapted and extended for all current types of deep learning analysis and for future discoveries on DL. The architecture and organization of the code must allow it to easily be extensible in the future. When the project architecture is already extendable, the next step is to actually start developing these extensions in order to properly add usefulness to the platform. These extensions consist mainly in the development of different dataset formats and the corresponding neural network category to train it on.

1.3 Methodology

The development of this project followed the work methodology described in the following steps:

- Define the set of functional and non-functional requirements for WebML's development;
- From the inspection of functional and non-functional requirements, collect and study all the background knowledge required for the task and master it;
- Compare and select the best technologies to be used in the implementation, analyzing each of the advantages and disadvantages in every step;
- Apply the best programming strategies and gather all resources to develop the platform;
- Document the developed work.

1.4 Document Structure

This document is structured as follows:

- **Chapter 1 - Introduction** - it introduces key concepts and contextualizes the current work. The main objectives are also identified in this section;
- **Chapter 2 - Background** - it presents the background and work in all related and relevant topics in order to better understand the remaining content of this document;
- **Chapter 3 - Proposal** - it explains in detail all functional and non-functional requirements of the platform, introduces the use-cases and the overall system architecture;
- **Chapter 4 - WebML Service Implementation** - it clarifies how datasets are handled and how different models interact with each respective dataset format. This chapter includes the server-side logic;
- **Chapter 5 - WebML Results** - it explains all the workflow of the graphical interface, both for a regular user and for administrators. This chapter can be used as an advanced tutorial for different use cases;
- **Chapter 6 - Conclusion** - it presents the main contributions of this work and all the possible extensions to the future.

2

Background

This chapter presents the background and work in all related and relevant topics in order to better understand the remaining content of this document. Important key concepts such as machine learning interface, as well as front-end technologies are explained.

2.1 Artificial Intelligence

Artificial Intelligence (AI) is a form of intelligence demonstrated by machines created by humans. *"The word 'intelligence' is defined as the ability to understand, learn, and use knowledge and skills in new situations. The roles and tasks of AI are to process and recognize the data acquired and then to perform specific tasks [7]."*

Artificial intelligence has seen its popularity rise over the years and is already part of our daily lives, being available in almost all mobile devices. Today's AI potential and the implementation of tasks is possible thanks to the development of other 2 recent technologies: Machine Learning (ML) and DL. Machine learning is a subset of artificial intelligence and deep learning is a subset of machine learning (Fig. 2.1).

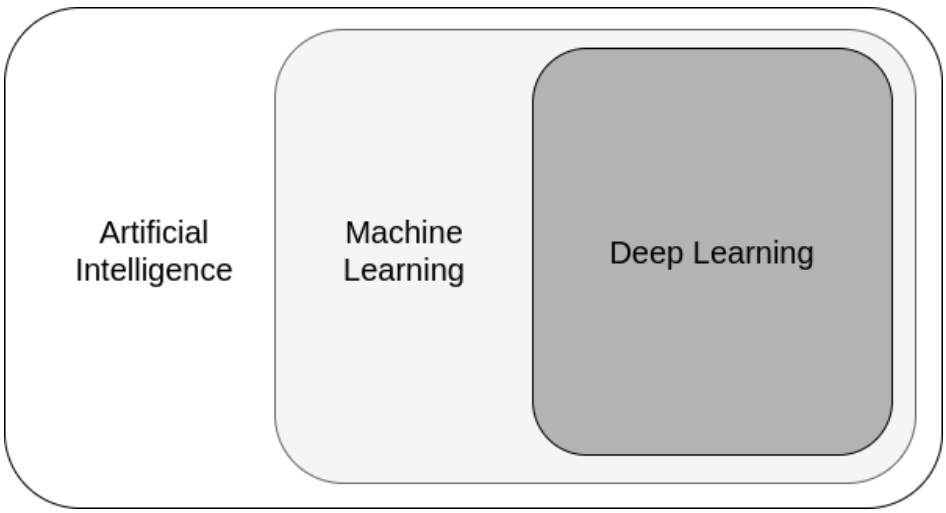

Figure 2.1: AI scope

ML and DL methods do not use standard logic to make predictions. Instead these methods use algorithms to learn from data. The backbone for any machine learning application is powered by big data [8].

2.2 Machine Learning

The name machine learning was invented by Arthur Samuel in 1959 [9], as *"the field of study that gives computers the ability to learn without being explicitly programmed"*. Machine learning is the subset of the artificial intelligence field that tries to achieve human-like intelligence with the use of statistical techniques that enable machines to improve at tasks with experience. The experience is the data that ML algorithms use to create a mathematical model. A core objective of a learner is to generalize from its experience [10]. Generalization in this context is the ability of a learning machine to perform accurately on new, unseen examples/tasks after having experienced a learning dataset. ML can also be considered the capacity of a computer application to learn from significant amounts of data. Different metrics to evaluate the performance are defined according to the type of learning and the desired objective. Metrics help to extend and optimize an algorithm. The most known metric in a ML model is the accuracy, which is the ratio of number of correct predictions to the total number of input samples [11].

There are different learning techniques whose differences lie on the correspondent input data format and type of ML algorithm. In general, there are 3 types of ML algorithms: supervised learning, unsupervised learning, and reinforcement learning [12], [13].

- **Supervised learning** - Supervised learning relates input data to output using labeled datasets. Labeled datasets are datasets that contain a respective value (output) for each entry (Fig. 2.2). This learning technique is employed to learn to identify patterns or behaviors in a training dataset where the output is known. It is fundamentally the learning that relates input data to its output through error correction techniques. Typically, this approach is used to solve classification and regression problems [14]. Regression refers to the process of finding a model or function for distinguishing the data into continuous real values instead of using classes or discrete values. An example of regression would be the prediction of the price of houses from the house's features. Classification is the process of finding or discovering a model or function in order to categorize the data into classes. An example of classification would be the prediction of a species of an animal (like a cat or dog) from an inputed image.
- **Unsupervised learning** - Unsupervised learning is used to identify underlying associations and patterns in unlabeled data input, which is data that have no associated output. It is commonly used for clustering. Clustering is the process of organizing similar data or objects into groups. Unlabelled data is usually easier to generate or gather than labelled data [13]–[15].
- **Reinforcement learning** - In Reinforcement learning the system is not trained with the sample dataset. It is a behavioural learning model that learns through trial and error. For example, an agent learning to play games though experience. The agent is provided with a set of actions to interact with its environment that can result in either reward or punishment. The agent should follow actions that generally result in bigger rewards. As such, a sequence of successful decisions will result from the process being reinforced [14], [16]–[18].

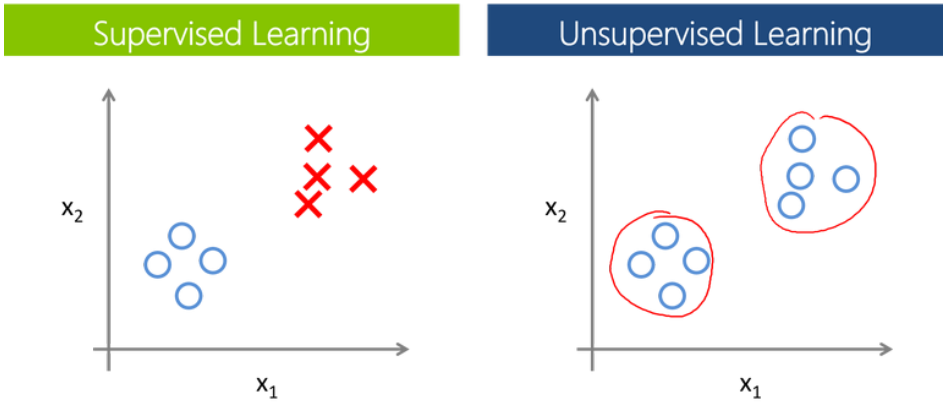

Figure 2.2: Supervised vs Unsupervised Learning [19]

Regarding these 3 types of learning, supervised learning is the most common [20]. Reinforcement learning is, so far, the least popular, having being studied less extensively than the others [15], [21].

In order to properly train and evaluate performance of learning on data, data needs to be splited (sliced) in 3 different parts (Fig. 2.3).

- **Training Data** - The slice of data we use to train our model. This data is exposed directly (both inputs and outputs) to the model so the model can learn by adjusting its weights as the patterns of data get identified. Training data is the most important and for such is always assigned with the biggest portion compared with the other slices of data.
- **Validation Data** - The slice of data which is used to do a frequent evaluation of model. It is used along with the training data in the training sessions in order to improve the involved hyper-parameters (initially set parameters before the model begins learning). This data plays its part when the model is actually training.
- **Testing Data** - Once the model has finished its training process, the testing data provides the unbiased evaluation. In other words, testing data is data dedicated to human validation or post-train data for analysis.

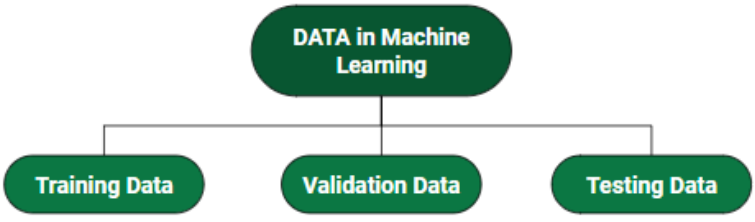

Figure 2.3: Data split in machine learning

Non-deep-learning machine learning is usually designed as classical machine learning (classical ML), whereas machine learning is a summary term that includes both classical ML and deep learning (Fig. 2.4). While deep learning has been growing in popularity in the past few years, classical ML is still very extensive across different research fields and industries [22].

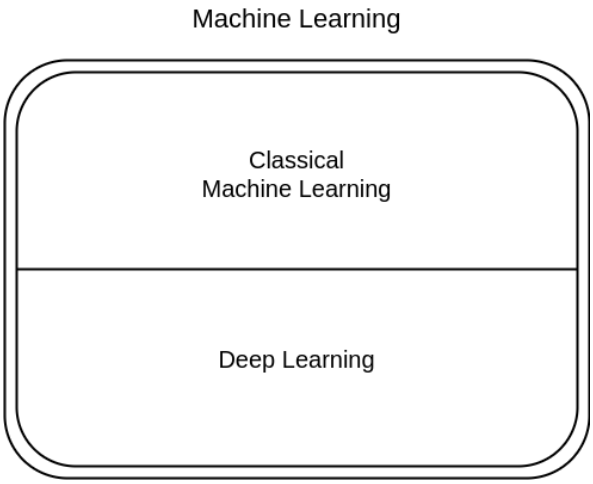

Figure 2.4: Machine learning: classical machine learning + deep learning

2.3 Deep Learning

Deep learning is a higher level of machine learning, which uses ANNs to analyze different factors with a structure that is similar to the human neural system. With the increase of the amount of data available, DL is becoming an important and an essential field in machine learning. Deep Learning is the subset of ML that uses ANNs with more than one hidden layer to model complex patterns between inputs and outputs. ANNs are widely used for modeling and predicting multi-dimensional and time-series data due to their capacity for learning complex patterns. An ANN is inspired by the biological neural networks that constitute animal brains. The neural networks are theoretically able to estimate functions where there is a relationship between the input and the output. The parameters of the neural network are determined only by the dataset features and are not limited to any analytical model. This approach allows it to be able to predict non-linear relationships in the data and is capable of achieving state of the art performance on high amounts of data when compared with classical machine learning algorithms. Deep neural networks architecture have been stabilising quite recently although the first general, working learning algorithm for deep learning was published by Ivakhnenko and Lapa [23] in 1967. Since the 2010s, advances in both machine learning algorithms and hardware have led to more efficient methods for training Deep Neural Networks (DNNs) that contain many layers of non-linear hidden layers and a very large output layer [24]. Due to recent widespread use of deep learning api for general proposes it is not an easy task to define the current effectiveness of these models. The word models is often used to refer to neural networks.

Feed-forward neural networks are the type of ANNs most used and also the most simple. In feed-forward neural networks, the neural network architecture is composed by layers. Layers can be seen as mutable processing steps in the neural network. These are usually mistaken for the values of the vectors in a certain step of the calculation process. Conventionally, many argue that input layer exists and have no processing applied to it and some articles also agree that the model input can be considered a layer, which is not actually true since there is no attributed processing or learning. Dialog speeches regarding this are often ambiguous and there is actually 2 different conventions. A single convention will be followed over the course of this document where the input is considered to be a vector and not a layer and layers are considered to be the applied processing in which a given

input is transformed into an output and learning can be applied. These layers are connected between themselves and each one of the layers computes an output through its input and feeds this output as the input of the next layer [25]. Each of this input-output transformations are usually performed by 2 steps. The first step is to apply a matrix multiplication between the input and the updatable weights (decimal values) of each layer and the second step is to apply an activation function to each resulting value (Fig. 2.5). Layer processing can differ slightly from this definition since there are multiple types of layers when referring to a neural networks structure.

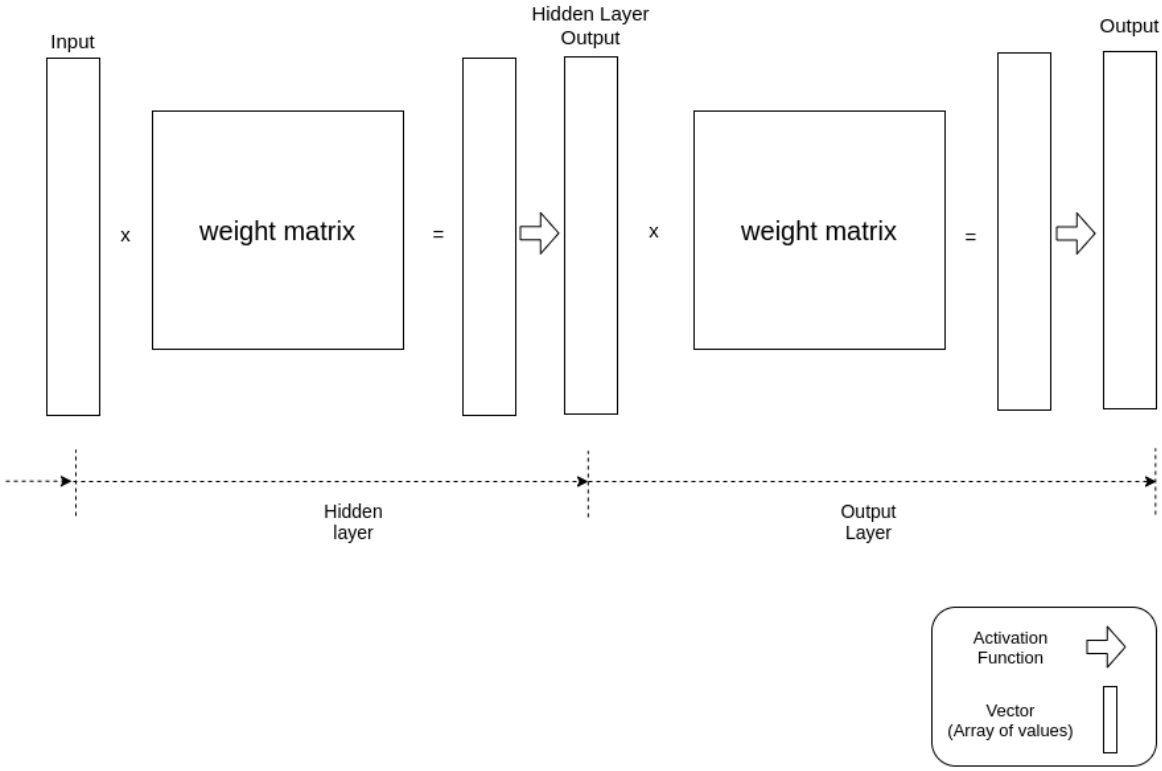

Figure 2.5: Neural network layers

Neural network layers can be more complex and specific for different learning requirements. Image classification and image localization benefits from Convolution Neural Networks (CNNs) that are built using multiple convolution layers. A convolution layer is a layer that applies multiple filters on neighbor pixels across the whole three dimensional input (width, height, RGB channels) in order to transfer new pixel patterns to the next layers as output. Text processing benefits from Recurrent Neural Networks (RNNs) that internally have recurrent layers. Recurrent layers are layers where the data is processed by using its output as an part of its input in smaller recurrent units, this feedback feature allows data to be learnt as a sequence easier than using a simple feed-forward approach.

In the training phase of an ANN, the training set is presented to the network and the weights of the layers are adjusted to predict the correct value for a given input set. The backpropagation algorithm, short for "backward propagation of error", is used to update the weights according to the output error of a loss function, using an optimizer function [26]. The main objective behind the learning phase is to

[0] https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

reduce the difference (error) between the model prediction and the actual value in order to make these 2 values the closest as possible.

Today its almost unthinkable to build a neural network from scratch using pure math on a standard programming language. There are several frameworks that help the modulation and computational Application Programming Interfaces (APIs) to increase work speed and reduce development time of solutions. Nowadays, DL is often applied using high-level frameworks usually using Python programming language. Python is the programming language of choice for ML and AI developers for a long time because of its small and compact instructions required to create and modify data compared with the majority of other languages. Python also benefits from the extensive libraries helping ease the workload [27]. Some of these libraries are fully-operational DL frameworks. In the following list are presented some of the most known:

- Theano;
- PyTorch;
- CNTK;
- Caffe and Caffe2;
- TensorFlow;
- Keras;
- FastAI.

Theano is one of the oldest DL frameworks that enable easy defining, optimizing and evaluation of powerful mathematical expressions. Theano also provides the ability to create custom C (compiled programming language) code for mathematical operations [27]. CNTK is developed by Microsoft and is optimized for voice, handwriting, and image recognition, the big disadvantage is the limited community support. TensorFlow framework is owned by Google and is the most used DL in the last years. PyTorch is the framework of choice for a large number of researchers and competitor of TensorFlow. PyThorch has a big community support, it stands out from other deep learning libraries for its declarative data parallelism, Model micro-services and servers, support for easy integrated graphical display of the results. Caffe framework was introduced by Facebook, the DL enjoyed the framework and made it push through the next level, the Caffe2 framework. Caffe2 framework is built for mobile and large-scale deployments, at Facebook, it's known as "the production-ready platform". Caffe2 framework is implemented in C++ alowwing it to have lightweight deep learning models. Also Caffe2 was merged into PyTorch1.0 to increase the advantages and disavantages [28].

2.3.1 TensorFlow Framework

TensorFlow is an end-to-end open source machine learning platform developed in multiple programming languages, Python being the most popular. TensorFlow relies on data-flow graphs with mutable state working as a functional API, it generates these computational graphs that pipeline predictions and learning from backpropagation (Code 1). The semantics of this library is operational and similar to the the literature found on programming languages. Some authors even argue that TensorFlow is actually a programming language on top of other programming languages [29]. It is the programmer responsibility to setup a compatible environment and design a valid computational graph that matches the specific requirements of the programmer's propose. TensorFlow is mostly used for ML and DL

since its release back in 2015. Although TensorFlow provides a great level of math abstraction, it is one of the lowest-level deep learning APIs for Python at the same level of Theano library.

```
import tensorflow as ft
#declare tensors
A = tf.constant([[3, 7], [1, 9]])
B = tf.constant([[5, 6], [2, -3]])

#matrix multiplication
result = tf.matmul(A, B)
```

Code 1: Matrix multiplication using tensorflow (v2)

TensorFlow had a major version of the framework (TensorFlow 2.0). It is a big step compared with the version 1.0, its library's API was significantly simplifyed.

2.3.2 Keras Framework

Keras is also a library and a functional API that is built on top of TensorFlow. Keras also can be executed on top of Theano or CNTK. This higher abstraction level makes Keras even simpler to use than TensorFlow. Keras is the deep learning solution of choice for many university courses and it is widely recommended as one of the best ways to learn deep learning. An example of a Python script slice is shown in Figure 2.6.

```
def createModel():
  inputs = Input(shape=[96,96,3])
  x = Conv2D(filters=128, kernel_size=3, padding='valid', activation='relu')(inputs)
  x = MaxPooling2D(pool_size=2, padding='valid')(x)
  x = Conv2D(filters=32, kernel_size=2, padding='valid', activation='relu')(x)
  x = MaxPooling2D(pool_size=2, padding='valid')(x)
  x = Conv2D(filters=64, kernel_size=3, padding='valid', activation='relu')(x)
  x = Flatten()(x)
  x = Dense(units=256, activation='relu')(x)
  x = Dense(units=128, activation='relu')(x)
  x = Dense(units=128, activation='relu')(x)
  outputs = Dense(activation='softmax', units=43)(x)
  model = Model(inputs=inputs, outputs=outputs)
  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
  return model
```

Figure 2.6: Keras Convolution Code Example - Python

There are several hyper-parameters in the training phase using Keras API. The **batch size** is the number of training examples in one forward and backward pass in the network. After the number of samples specified in a batch size is analyzed, the internal parameters of the neural network are updated using the backpropagation algorithm. The number of **epochs** defines the number of times that the learning algorithm will work through the whole dataset. In the prediction phase, the neural network works as a simple input-output function.

```
classifier.fit(X_train, y_train, epochs = 10, batch_size = 128) #training

prediction = classifier.predict(X_test) #testing
```

Code 2: Keras code for train models and predict results

Keras and TensorFlow libraries have been extended to the possibility of using GPUs to significantly accelerate the training process. If the user has multiple GPUs available, he is allowed to configure which ones to use on training sessions of even to use CPU only.

2.4 Graphics Processing Unit (GPU)

A GPU is a processor that is specially-designed to handle intensive parallel computations. Over the last decades there has been a growing interest in the use of GPU for non-graphical applications. GPU's technology is advancing far more faster than that of conventional CPUs (Fig. 2.7).

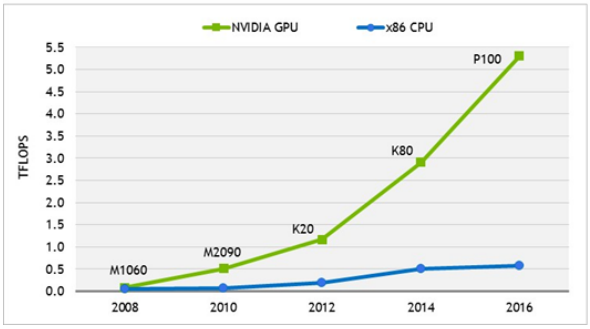

Figure 2.7: GPU vs CPU growth from [30]

Modern GPUs are very efficient at manipulating computer graphics, image processing and for DL. Their highly parallel structure makes them more efficient than general-purpose CPUs for algorithms that process large blocks of data in parallel. Getting started with GPU programming can be simple, however being able to fully utilize GPU hardware is an art that can take months or years to master. It is always a good idea to use GPUs on top of known libraries and have the recommended drivers installed in the hosting operative system [31], [32]. Although GPUs are much faster then CPUs for DL proposes, DL cannot work without a CPU and it can work properly without a GPU at the cost of performance (Table 2.1).

CPU	**GPU**
Low amount of cores	Thousands of cores
Complex control logic	High computations per memory access
Optimize for serial operations	Build for parallel operations
Low latency tolerance	High latency tolerance
High clock frequency	Relatively low clock frequency

Table 2.1: CPU vs GPU [33]

2.5 Web Application Interfaces

Web applications are derived from web based systems, which have the additional functionality to deal with the business logic of an organization, the back-end service [34]. They often supply a public client interface for the World Wide Web (WWW) (Fig. 2.8). In a web based system, the web application service is usually coupled along with a GUI and a database service. Currently, back-end services are developed using frameworks that accelerate the development.

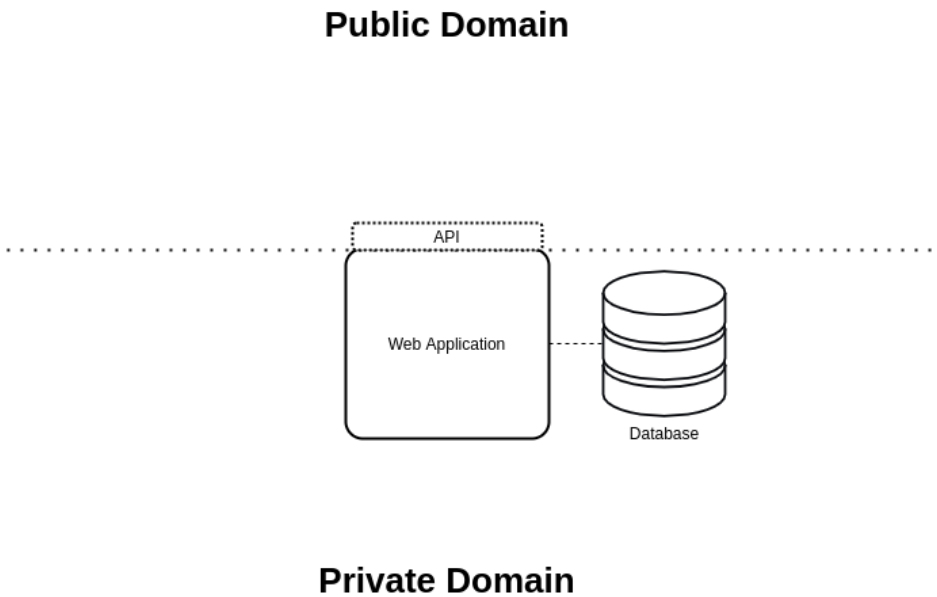

Figure 2.8: Web Application Environment

2.5.1 Back-end Frameworks

Frameworks have become a necessary part of the back-end development in the modern age, as the trend of developments is always growing by the year. It is not smart to develop a fully operational back-end service without using open-source libraries or dedicated software. There are multiple frameworks available supported by millions of developers around the world and currently popular frameworks are always up to date [35]. These frameworks offer different coding tools, programming languages, interfaces and other features. According to the GitHub ranking status of 2020, some popular frameworks for back-end development are presented in Table 2.2.

Framework	**Programming Language**
Flask	Python
Django	Python
String Boot	Java
Laravel	PHP
ExpressJS	JavaScript (NodeJS)
Ruby on Rails	Ruby
Meteor	JavaScript (NodeJS)
Koa	JavaScript (NodeJS)
Nest	JavaScript (NodeJS)
Asp .NET	C#

Table 2.2: Popular back-end frameworks

The list presented above is a small fraction of the available back-end frameworks. With so many, it can become difficult to select the framework that provides the best starting point for a new web application [36]. There are some factors that should be considered along with the task in hand, that is, along with the web application that is going to be developed. Some of those factors, that may affect the developers' decision, are:

- Effort to learn;
- Productivity;
- Performance of the framework;
- Caching support;
- Scalability;
- Web security.

Regarding the most popular back-end frameworks, Asp .NET has a strong influence in front-end development but it is ideal for both cases since C# is a consistent and efficient compiled programming language and Asp. NET provides a good interface for databases. Django and Flask are both Python frameworks, while Django is usually a front-end framework, Flask works by configuring web endpoints (Uniform Resource Locator (URL) path). These Python frameworks take advantage of the advantages of data management that Python provides, but also its performance disadvantages and lack of consistency compared to other programming languages. Laravel is a back-end PHP framework, it was one of the most used frameworks but nowadays PHP is a relatively less productive programming language compared with its competitors. NodeJS is the most used environment to deploy a back-end service. NodeJS is a JavaScript environment that runs on the operative system instead of a browser container. Although JavaScript is an interpreted language, its optimizations over the years are remarkable and its performance can even be compared with non-interpreted languages (machine code). JavaScript running on a NodeJS environment provides non-blocking single-threaded asynchronous services that are recommended for multiple client (non-computationally intensive) requests. Although single-threaded responses seem to be slower, it is not the case. Because executing a new thread (worker) for every request (ex. Asp. NET and Spring Boot) require significant additional time, while queuing client requests (ex. NodeJS environment) in a queue is better for Input-Output (IO) tasks as long as the computational logic is faster than the time for assigning a new thread. Regarding the most popular NodeJS back-end frameworks, ExpressJS stands out in ranking and community support resulting in better version updates regarding bug fixes and functionalities.

2.6 Web Graphical Interfaces

Using the browser for web navigation has proven to be a very efficient tool due to the vast compatibility over the many devices that are available in the global market. The advantages of web browser navigation compared to a downloadable software are well known, one of them is that the web application is always updated by default without the need to installation procedures. The browser engine works by fetching the data of the requested URL and graphically building a web application for the user. The contents of small and compact application written in Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript (Code 3) are downloaded every time a user refreshes or requests a page from the browser. These 'web applications' are executed in a safe-environment that has limited access to the disk unless explicitly requested by the user.

```
<!-- This is the comment section -->
<html>
  <head>
    <title>This is the title of the webpage!</title>
  </head>
  <body>
    <p>Hello World!!</p>
  </body>
</html>

<script>
// JavaScript comments
document.body.onload = function() {
    console.log("Page is now loaded");
}
</script>

<style>
/* CSS comments */
p {
    color: red;
}
</style>
```

Code 3: HTML file example

CSS code is placed between 'style' tags and JavaScript is placed between 'script' tags.

2.6.1 HTML

The HTML language was first created by Tim Barners Lee in 1990. HTML is the standard language for creating web pages and applications, and it is thus one of the fundamental technologies of WWW [37].

The advantages of HTML in the construction of a web page are remarkable: it is an open source language with a simple structure, easy to use and to learn, supported by all types of browsers, among many others.

To structure and create content, HTML uses series of standard tags. The 'head' and 'body' are examples of these tags [38].

2.6.2 CSS

CSS is the standard language for styling structured documents in web pages. CSS allows the control of the display of HTML items in a page. From selecting the color of the text, to the spacing between components, it offers a wide range of possibilities to better design the page [39]. Furthermore, it is supported by all browsers and reduces the weight of web applications that otherwise would require hard coded styling derived from HTML or JavaScript.

However, CSS lacks variables and functions, which enable code reuse and structured programming. Alternatively, CSS Preprocessors like SCSS (Fig. 2.9) and LESS have been introduced as superset languages to extend CSS by supporting those missing constructs [38], [40].

file1.css > ...

```css
/* css comments */
.content {
    width: 100%;
}
.content_title {
    font-size: 20px;
    color: #833c3c;
}
.content_subtitle {
    font-size: 15px;
    color: #833c3c;
}
```

file2.scss > ...

```scss
// scss comments
$primary-color: #833c3c;
.content {
    width: 100%;
    &_title {
        font-size: 20px;
        color: $primary-color;
    }
    &_subtitle {
        font-size: 15px;
        color: $primary-color;
    }
}
```

Figure 2.9: CSS vs SCSS

2.6.3 JavaScript on Browser Environment

Alongside HTML and CSS, JavaScript is a core technology in the web world [41]. JavaScript is a interpreted programming language that enables interactive web pages and is an essential part of web applications. The vast majority of websites use it for client-side page behavior, and all major web browsers have a dedicated JavaScript engine to execute it [42].

As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative programming styles. In the browser environment, JavaScript can be applied to the Document Object Model (DOM) (Code 4), which is the main representation model of all the components are that are displayed in the screen for the user (Fig. 2.10). JavaScript language excels at IO behavior, such as networking, storage, or graphics facilities. It is generally considered a good practice to include as much JavaScript as possible using external files reducing the overall size of the web application [43].

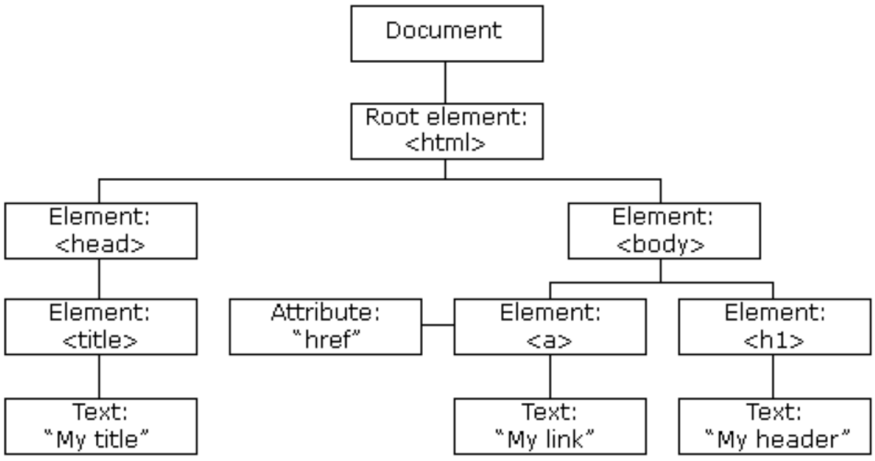

Figure 2.10: DOM tree example [44]

```
//get element with id "demo"
var demo = document.getElementById("demo");
//change its display text to "Hello World"
demo.innerHTML = "Hello World";
//change background color"
document.body.style.background = 'red';
```

Code 4: Browser JavaScript applied to the DOM

2.6.4 Front-end Frameworks

Web development is all about providing the client the correct HTML, CSS, JavaScript from a public reachable server. It is possible to write the code of a whole web page using plain client code but it is not a smart move. Frameworks really boost the speed of development, most of them provide the serving service itself while a few only compile static client code.

Nowadays, JavaScript frameworks are the most used compared to another programming languages front-end frameworks. This trend is to be expected because the same programming language can be used for client and server code, allowing easy replicated code. One example is the case of the users' input validations. These validations are mandatory on the server-side API but are also recommended in submission forms on the web page since there is no point in submitting a request that is going to be invalidated. This would allow to implement the same validation, replicating code on the client and on the server application, sparing time instead of implementing the same logic in multiple programming languages.

When choosing the JavaScript front-end framework that best fits a given project, it is important to evaluate the efficiency for the specific task. Another important aspect is to evaluate the developers' background knowledge of the frameworks and to balance it to the level of difficulty when it comes to learn them. The 3 most known frameworks are React, Angular and Vue (Fig. 2.11).

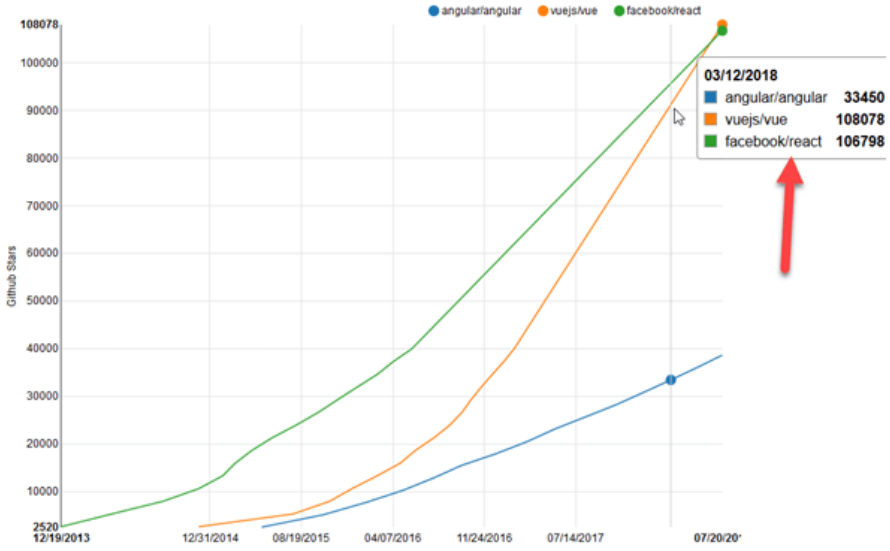

Figure 2.11: JavaScript Frameworks - Angular vs React vs Vue [45]

Angular is a open-source web framework introduced by Google in 2016. Angular is a TypeScript framework that has been relatevly losing its popularity over the years compared with its competitors, React and Vue.

2.6.5 Vue and Nuxt

Vue (pronounced as 'view' in English) is a progressive JavaScript framework for building user interfaces. Unlike other frameworks, Vue was designed from its conception to be incrementally adoptable. Vue files try to simplify the code that React and Angular did not manage to simplify, making it easy to adopt and integrate with other existing libraries or projects (Code. 5). Vue is also perfectly capable of powering sophisticated web pages with modern tools and supporting libraries [46].

```
<template>
  <section>
    <!-- children components here -->
  </section>
</template>

<script>
export default {
  data() {
    return {
      //component properties here (work as local variables)
    }
  },
  methods: {
    //methods here
  },
  computed: {
    //computed properties here
  }
}
</script>

<style>/* css here */</style>
```

Code 5: Vue file template

Figure 2.12: Nuxt + Vue [47]

NuxtJS is a framework designed to provide a strong architecture following official Vue guidelines. NuxtJS offers Server Side Render (SSR). SSR is the process where server sends all prepossessed data directly to the browser with the completed final HTML/CSS layout. The client browser doesn't need to call additional APIs to fetch data from the source to manipulate the DOM. This features allows the possibility, without extra API requests from the client side (Fig. 2.13), to have all post-processed data populated in browser (Fig. 2.14) [48], [49].

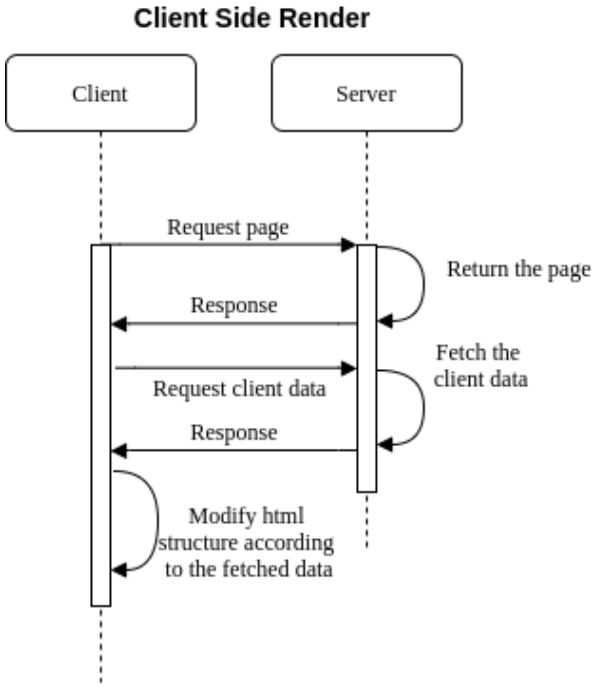

Figure 2.13: Sequence diagram of client side rendering

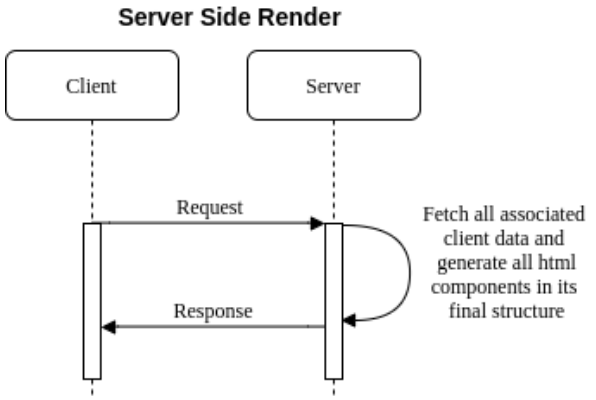

Figure 2.14: Sequence diagram of server side rendering

Without SSR, websites that have authentication procedures and user sessions linked to a user account won't load the user requested data on page refresh. Instead, in the best case scenario, they would have to wait for the authentication procedure to happen on the client side, what in turn, would result in an unpleasant visual transition.

2.7 Docker

Docker is an open platform that provides the ability to package and run an application in a loosely isolated environment called a container. Docker containers are lightweight application processes that encapsulate applications, abstracting them from the host machine, wrapping up a piece of software in a complete filesystem that contains everything it needs to run. They share the host's kernel [50]. A Docker container is a package with the application code and all its dependencies that is usually

independent from the host environment, allowing the deployment and execution of applications in different environments with significantly less effort. A Docker container is similar in functionality to a virtual machine where the virtual environment is isolated and monitored. Docker has proved to be superior to the other competing solutions. The containers are created from a Docker image that specifies the contents of the container. Images can be set up from other base images of previously built operating systems or environments. Docker images can be pushed or pulled remotely. The big advantage is that it is possible to load a fully functional environment with all dependencies installed without requiring them to be installed manually. Docker can build images automatically by reading and executing the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build, users can create an automated build pipeline that executes several command-line instructions one after another. The resulting docker image obtained by building it from the Dockerfile in example Code 6 is a complete working environment that can leverage from GPU parallel processing power to run Python scripts that requires TensorFlow and Keras libraries.

```
FROM TensorFlow/TensorFlow:latest-gpu
WORKDIR /home
RUN pip install Keras pillow matplotlib scikit-learn pandas
EXPOSE 8000/tcp //port 8000 is reachable from outside the environment
```

Code 6: Dockerfile

2.8 Databases

A database is an organized collection of data or information. They are structured to easily storage, retrieve, modify and delete data, with the help of various data-processing operations [51].

A Database Management System (DBMS) is a set of computer programs assigned to execute operations on the information stored within a database.[52] DBMSs usually respond to database queries and procedures that are requested through an API. A query is a language expression that describes data to be retrieved or modified from a database. Queries can be categorized as data creation and data destruction [53]. With millions of data transactions taking place every second, database optimization is essential to guarantee that services do not harm the clients.

Modern DBMS technologies are supplied with enough tools to greatly increase efficiency on data retrieval using mechanisms like indexing. A database index is a physical access structure for a database table that indicates where the records are physically stored on the disk. One way to better understand the concept of database index is to look at it the same way as we look at a regular index of a book. The first few pages of a book (indexes) guides the reader to the right place [54].

A database is organized with a file or a set of files. Each information is stored into records. Records are organized into tables that include information about relationships between the stored content. Using specific query languages, users can rapidly search, rearrange, group, and select data from many records [55].

There are two types of database:

- Relational Databases
- Non-Relational Databases

Both databases types have their advantages over the other and today's DBMS are optimized and stable for production environments in both types of databases.

2.8.1 Relational Databases

A relational database organizes data into tables and forms relations between them [56]. The most used structured language for relational databases is Structured Query Language (SQL). SQL is standard language for interacting with management systems [57]. It allows the joining of tables and other tasks through queries (Code. 7).

```
SELECT *
FROM customers
JOIN orders ON customers.customer_id = orders.customer_id
```

Code 7: SQL query using joins

2.8.2 Non-relational Databases

A non-relational database stores data in a non-tabular form, and tends to be more flexible than the traditional, SQL-based, relational database structures. It does not follow the relational model provided by traditional relational database management systems. Non-relational databases are often used when large quantities of complex and diverse data need to be organized. Non-relational databases often perform faster because a query doesn't have to inspect several tables in order to deliver an answer, as relational datasets often do. Non-relational databases are therefore ideal for storing data that may be changed frequently or for applications that handle many different kinds of data. They can support rapidly developing applications requiring a dynamic database able to change quickly and to accommodate large amounts of complex, unstructured data. Non-relational databases are ideal for storing data in json format that contain nested objects.

Programmers and analysts may take benefit of non-relational databases as it has simpler modeling constraints than the relational databases [58].

There are several advantages to using non-relational databases, including:

- **Massive dataset organization** - In the age of Big Data, non-relational databases can not only store massive quantities of information, but they can also query these datasets with ease. Scale and speed are crucial advantages of non-relational databases.
- **Flexible database expansion** - Data is not static. As more information is collected, a non-relational database can absorb these new data points, enriching the existing database with new levels of granular value even if they do not fit the data types of previously existing information.
- **Multiple data structures** - Nowadays data collected from users takes on a myriad of forms, from numbers and strings, to photo and video content, to message histories. A database needs the ability to store these various information formats, understand relationships between them, and perform detailed queries. No matter what format your information is in, non-relational databases can collate different information types together in the same document.
- **Built for the cloud** - A non-relational database can be massive. As databases can grow exponentially, in some cases, they need a hosting environment that can grow and expand with. The cloud's inherent scalability makes it ideal for non-relational databases.

Applications must be able to query data efficiently and deliver results almost instantly. Non-relational databases are a natural choice for this kind of environment. They offer both security and

agility, allowing for rapid development of applications in an agile environment. Easier and less complex to manage than relational databases, they can also yield lower data management costs while providing superior performance and speed [59].

2.9 Related Work

This section describes the state of the art related to advanced user interfaces for deep learning.

User interfaces for deep learning are not a recent development, big companies have invested on this type of approach for some years now. All current online projects have their problems and limitations. It is important to look at their weaknesses in order to learn from them and avoid making the same mistakes when building our own competitor application. Some of them are full web integrated while others need a downloadable software to work properly. This section there will cover 4 technologies that have been known for its usefulness in the following list:

- **Azure Machine Learning Studio** - Web application developed by Microsoft;
- **Neural Designer** - Non-web application developed for multiple operating systems;
- **Deep Cognition** - Semi-web application with extension to be launched in the cloud;
- **FastAI** - Advanced deep learning library for Python.

2.9.1 Azure Machine Learning Studio

Azure Machine Learning (Azure ML) belongs to Microsoft. Is a cloud-based service for creating and managing machine learning solutions. It's designed to help data scientists and machine learning engineers to leverage their existing data processing and model development skills and frameworks. Also, help them to scale, distribute and deploy their workloads to the cloud. Azure Machine Learning Studio is the web portal for data scientist developers in Azure Machine Learning. The studio combines no-code and code-first experiences for an inclusive data science platform [60]. Azure ML provides management directly on the browser of a wide list of machine learning features:

- Models;
- Datasets;
- Datastores;
- Compute resources;
- Jupyter Notebooks;
- Experiments;
- Run logs;
- Pipelines;
- Pipeline endpoints - remote access to model prediction utilities.

One of the biggest advantages of Azure ML studio is the visualization power of this tool for experienced users, (Fig. 2.15). This tool requires a significant adaptation from the user as it can be difficult to get started. The best course of action for new users is to start by following along the guided tours that the platform offers. Azure ML studio is built for all skill levels. However, for more complex machine learning experiments it may require the user some solid background knowledge on machine learning in order to not get confused by the displayed terms. There is an "Advanced settings" section where users can define their desired settings for the training job, such as early exit conditions, cross validation methods, algorithms selection and others [61].

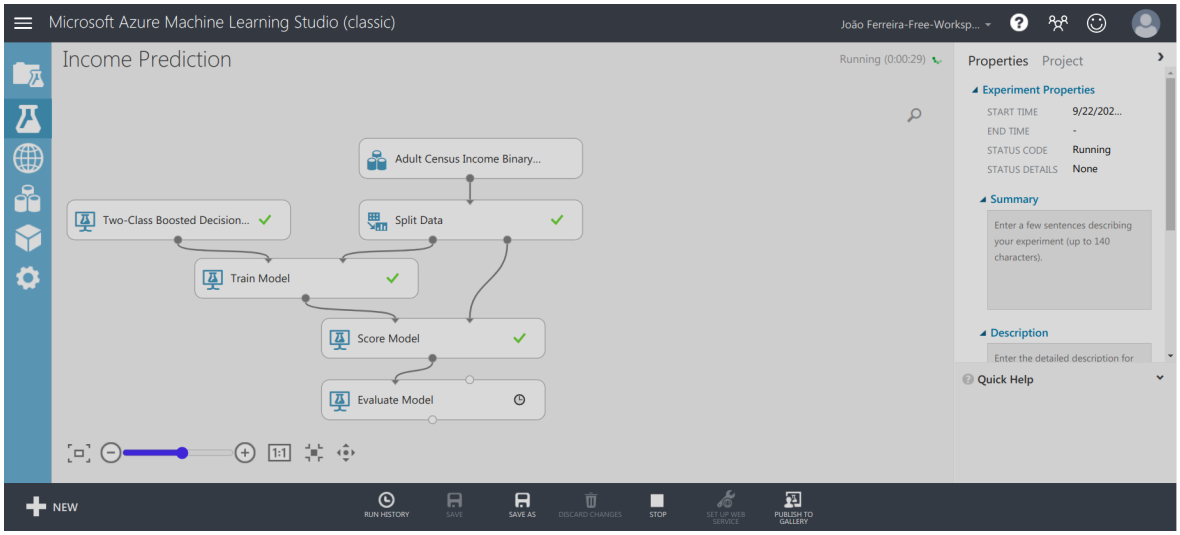

Figure 2.15: Azure ML Data Modeling

Another great advantage is the ability to deploy models, this deploying feature corresponds to a supplied external API for the user to call along with the data and receive a prediction response. This functionality uses machine learning models, including neural networks, in the cloud without the need for the user to have the necessary hardware and to code the implementation. Although Azure ML Studio offers many advantages for machine learning experiments, it is not so intuitive for deep learning experiments, giving the users that want to build a specific layered structure for a neural network model, a hard time.

2.9.2 Neural Designer

Neural Designer is a data science and machine learning platform that helps you build, train, and deploy neural network models. This platform is closely related to the one discussed on Azure Machine Learning Studio.

Neural designer is a non-browser application, it requires an installation in order to use it. Neural designer software presents an old-fashion UI design that aims to provide common deep learning configurations and settings while also displaying input features of models that can be selected from target datasets (Fig. 2.16). The input of Neural Designer experiments is a dataset, and the result is a predictive model. That result takes the form of an explicit mathematical expression, which can be exported to any computer language or system [62]. Visiting 'Neural Designer' website, the user gets very well clarified on neural networks and its potentialities as its tutorials are very responsive and friendly. Its dynamic design guides the user into downloading and trying its software. Downloading this software can lead to complications depending on the operating system at hand due to some required libraries. When the downloaded application is launched, the user is faced with a very different design and intuition from the previous encounter with their website. It can prove very complex to get started with the first experiment as the data presented from a target dataset is strangely displayed in some scenarios. It also behaves differently on different operating systems.

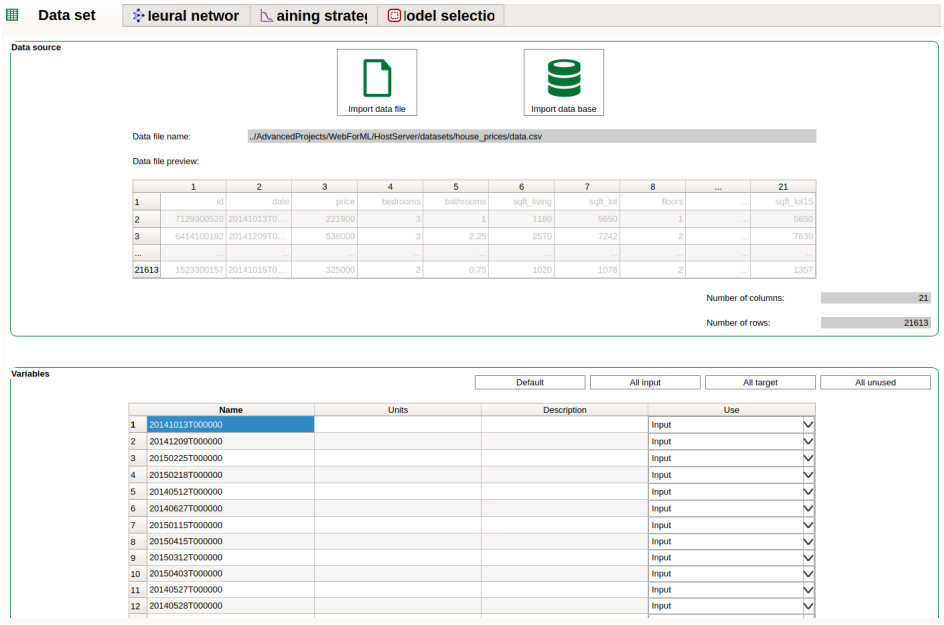

Figure 2.16: Neural Designer Software Print

2.9.3 Deep Cognition

Deep Cognition is not a full-web service as the users cannot simply browse through their website to be able to use it like Azure ML Studio does. Instead, Deep Cognition requires the user to download a software installer for their services. This installer possesses an intelligent installation procedure that adapts to the operative system limitations and hardware. After the installation procedure is finished, the user is able to execute an executable file that provides multiple options to start or stop a web service (2.17). This service launches a web graphical interface server called 'Deep Learning Studio' running on a local port of the host computer. The user is allowed to experiment this tool in the browser after its registration in the official website and when the service is up and running. While Azure ML studio performed neural network training sessions remotely using remote hardware, in Deep Cognition training sessions are performed locally using the user's hardware which poses problems and limitations.

```
Usage: ./dlsctl <cmd> [options]

================================================================================
|  cmd      | option|          description                                      |
================================================================================
   config             Configure options.
                      Note that you might need to restart the software for settings
                      to take effect. It will be saved in internal configuration
                -p    Port number to listen on (Default: 22000)
                -i    Specify public ip of computer (if software is accessed remotely)
                -s    Show current settings

   install            Installs deep learning studio server software.

   start              Starts deep learning studio services.

   stop               Stops deep learning studio services.

   restart            Restarts deep learning studio services.

   status             Shows status of deep learning studio services.

   help               Shows this usage/help message.
```

Figure 2.17: Deep Cognition Service Controller

Deep Cognition goal is to provide its users an easy to use platform to develop and deploy AI. This platform can be used on their infrastructure or in the cloud. Its mission is to help developers realize the potential of AI by providing them with Deep Learning Studio [63]. A new user using this platform is guided to a list of project samples that significantly help the user understand the concept of projects, models and datasets that Deep Cognition adopts (Fig. 2.18). Providing examples to new users is a smart approach to improve user learning curve, without this feature, users would be forced to resort to trial and error or learning from the provided video tutorials available in Deep Cognition website. These project examples include most important types of neural networks ranging from CNNs to RNNs.

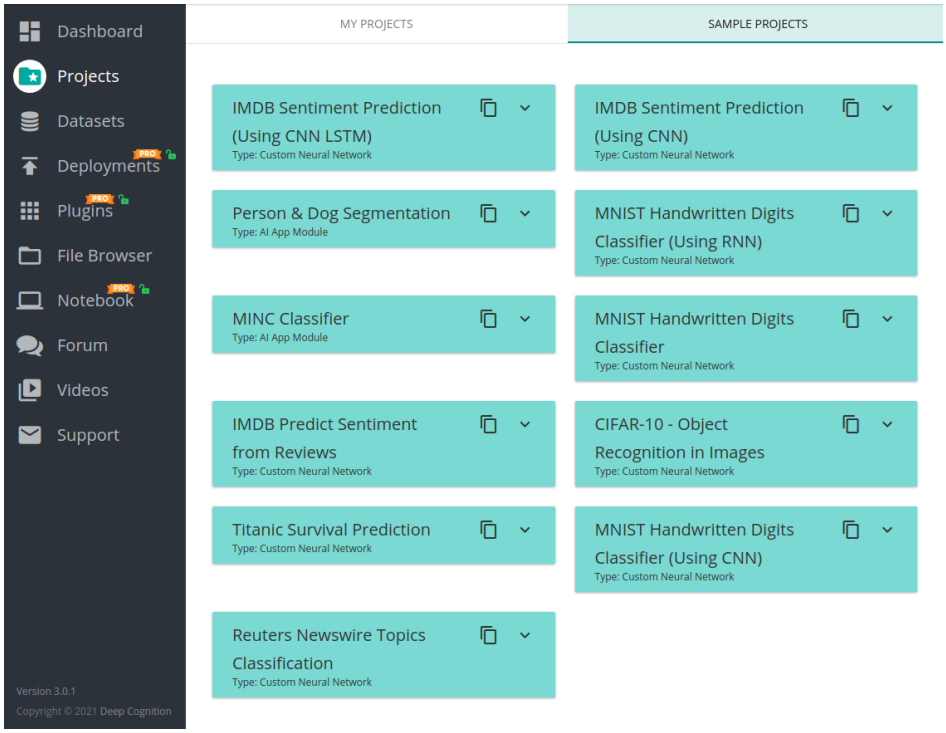

Figure 2.18: Deep Cognition Sample Projects

In order to demonstrate the scope of this studio features, the project 'MNIST Handwritten Digits (Using CNN)' is demonstrated in the following Figures (2.19 and 2.20). The MNIST dataset is one of the most used image datasets for benchmark models and to illustrate ML and DL experiments. The model in this example is a relatively small convolution model that contains 10 layers, 2 of them are convolution layers. The Deep Learning Studio provides multiple options for model configuration including advanced layer configurations. Deep Learning Studio functions by using Keras library behind the scenes for building and training models, it is essentially a graphical interface for generating code written in Keras library, The code generated for the model can be inspected by clicking on the 'view code' icon (Fig. 2.21).

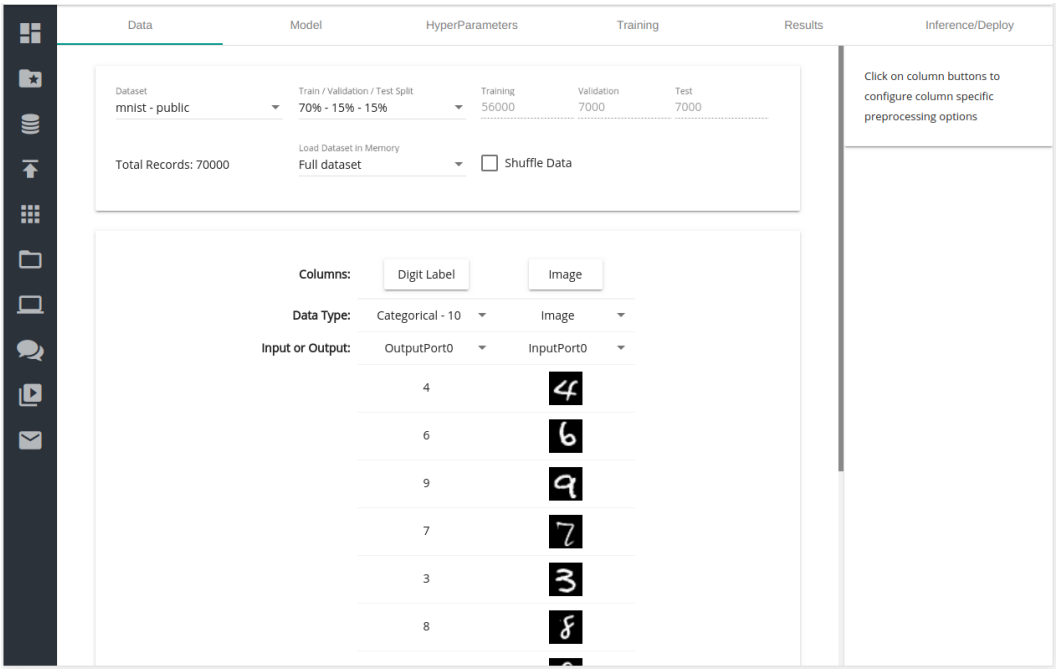

Figure 2.19: Deep Cognitive MNIST Handwritten Digits (Using CNN) - Data

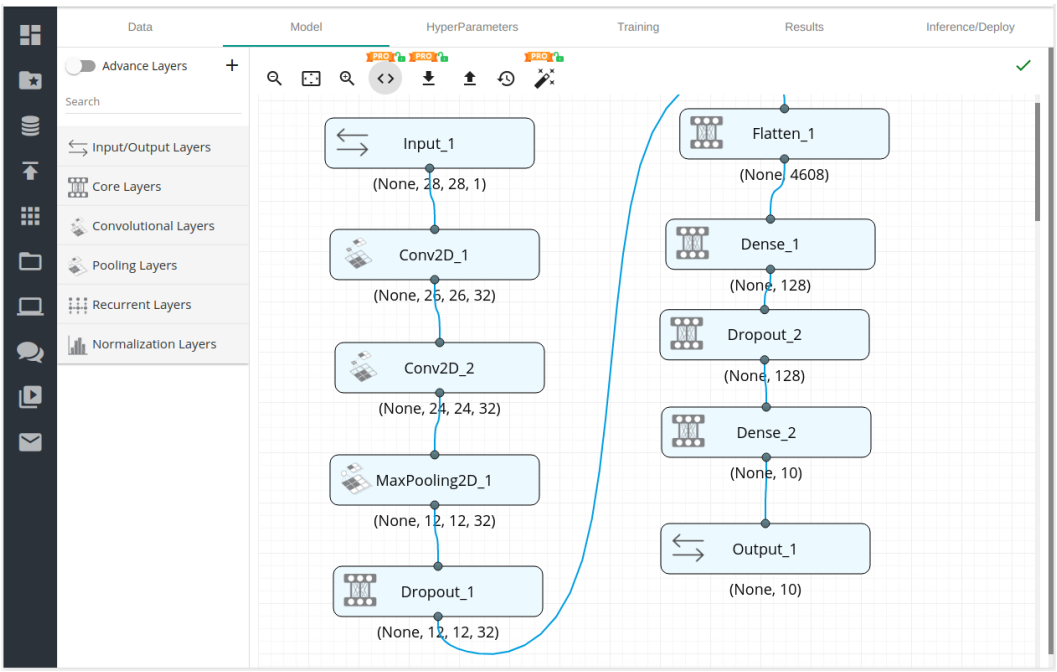

Figure 2.20: Deep Cognitive MNIST Handwritten Digits (Using CNN) - Model

Figure 2.21: Deep Cognitive code generation for Keras library

2.9.4 FastAI

FastAI is not a graphical interface like the Azure ML or Neural Designer. Instead, FastAI is a high-level deep learning library for Python like Keras (section 2.3.2) which provides state-of-the-art results in standard deep learning domains, and provides researchers with low-level components that can be mixed and matched to build new approaches. Deep learning libraries for Python range from low to high-level APIs, the lower the level of the library, the higher the knowledge and code size required (Fig. 2.22).

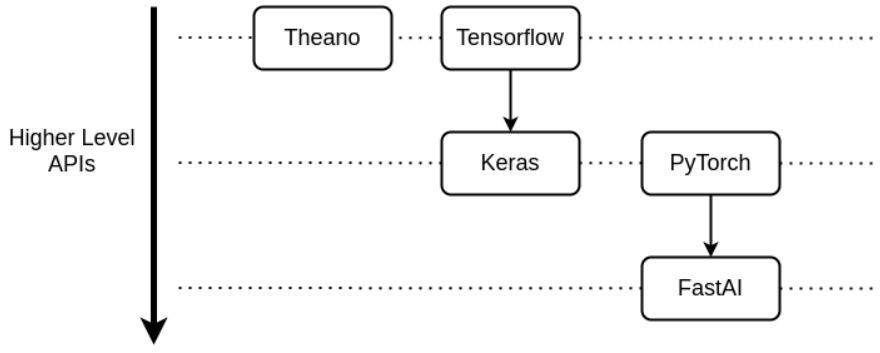

Figure 2.22: Deep learning libraries for Python

FastAI abstraction level is possible thanks to a carefully layered architecture, which expresses common underlying patterns of many deep learning and data processing techniques in terms of decoupled abstractions. These abstractions can be expressed concisely and clearly by leveraging the dynamism of the underlying Python language and the flexibility of the PyTorch library, a competitor of Keras library [64]. Although FastAI is not a graphical interface, it is still a respected tool in DL world and as the name suggests (FastAI), it provides multiple DL tools in a reduced amount lines of

code. A small example of the use of this library can be seen in (Code. 8).

```
from fastai.vision import *
path = untar_data(MNIST_PATH)
data = image_data_from_folder(path)
learn = cnn_learner(data, models.resnet18, metrics=accuracy)
learn.fit(1)
```

Code 8: fast.ai code example for images

It is possible to see on this example how compact the code to execute a training session from a pre-trained neural network, which in this case is a 'resnet18' structure. FastAI allows short and efficient code blocks for different types of data.

- **vision** - for image datasets or CSV datasets that represent images;
- **text** - for text datasets on CSV or clear text;
- **tabular** - for CSV tabular data or tables from databases;
- **medical** - for DICOM files/datasets.

All of this data types interactions have been developed to the point where modern and professional AI can leverage from this light and easy to use API. FastAI framework also provides ready to use datasets without a need to have previously download them.

3

Proposal

This chapter is intended to present the WebML proposal. It explains in detail all functional and non-functional requirements of the platform, use cases and the overall system architecture. WebML platform tries to merge the positive features of the previously discussed platforms in section 2.9 and discard the negative features. Essentially, WebML incorporates the remote training features of Azure ML Studio and the advanced GUI of Deep Cognitive. Moreover, it proposes an extensible distributed architecture that allows the easy addition of new computational nodes and the transparent management of available resources and datasets.

3.1 Functional Requirements

Functional requirements are the primary tool for a customer to communicate its needs. These requirements generally define the main scope of a given project.

In order to better identify the functional requirements of WebML, actors and their use cases must be studied. There are 2 main actors that use WebML: the platform administrators and regular users. The administrators are responsible for managing users and equipment. They are responsible for monitoring hardware and the overall performance of WebML. It is also their responsibility to assign user privileges and limitations. Managing users includes register and unregister them to the platform. Users cannot register themselves in WebML. This policy is applied to prevent disturbance of service in case of higher GPU session requests.

To identify the use cases, the firsts to be accounted for are the neural network specific use cases. Neural networks are non-linear decision-making tools, they serve as complex input-output functions. They can be used to identify and model complex relationships between inputs and their given outputs and to find patterns in data [65]. Neural networks are great at identifying hidden patterns that cannot be easily programmed using logic.

The following list represents different possible use cases for neural networks:

- Predict when equipment failure might occur and prevent it;
- Evaluate the behavior of a system;
- Recognize the actions that a user performs in specific scenarios;
- Classify images efficiently and accurately;

- Analyze clinical data to diagnose diseases quickly and efficiently;
- Predict future results based on past data, using machine learning techniques;
- Identify customers likely to leave, allowing companies to take action in advance;
- Identify who is interested in a certain company's products and services.

This previous list refers to the use of neural networks only. Any deep learning application should offer the possibility to use all of the above use cases since these are specific to neural networks. WebML's users must be able to train models on a target dataset and test their performance. To allow this behavior, users must be able to create, edit and remove the neural network structure using the UI. Since users may want to train a model on a customized dataset, they also must be able to upload and delete their own datasets. Creating a neural network structure from scratch every time they want to test the same experiment with small adjustments can be time-consuming and lead to mistakes. In order to solve this issue, WebML's users must be able to clone the model structures and also to create 'placeholder/dynamic' models that can adapt to any target dataset, by 'shaping' it along with that dataset. An additional important feature, outside of the scope of the UI, is the possibility of 'download experiment'. This feature must allow users to download a pre-built project regarding a target model where users can train models offline and use their own hardware in Linux operating systems, the provided value mainly depends on the user's purposes. Multiple reasons can lead a user to take advantage of this tool, such as the study the code base, incorporate DL code into the user's projects or test local hardware.

The complete list of WebML use cases for **non-administrators** that goes beyond the neural networks specific use cases are described below in table 3.1.

Use cases	Description
login/logout	User authentication process
upload datasets	Insert datasets into the platform
download datasets	Download datasets from the platform
remove datasets	Remove datasets from the platform
create models	Create neural network models
edit models	Edit neural network models
clone models	Copy the structure of a neural network model
shape models	Transform a model to fit dimensions to a dataset
remove model	Remove a neural network structure
remove weights	Remove a model's weights
train models	Train or re-train neural network models using a dataset
use model prediction	Insert test data as input to evaluate model performance
download model weights	Download training results of a neural network model
download model experiments	Download training scripts for offline tasks

Table 3.1: Regular user use cases

The platform must also provide extra use cases for the administrator such as management of hardware, users and models. For the purpose of simplifying and helping the reader understand these use cases, hardware provider nodes/machines will be referred to as **slaves**. Slaves require at least one GPUs board as part of their components and are the computational machines that are assigned to perform the hardware intense tasks like training sessions and prediction requests.

The administrators specific use cases are displayed in table 3.2.

Use cases	Description
register users	Register users to allow them to use the platform
unregister users	Remove the user from the platform
inspect user sessions	Inspect training sessions and hardware usage of a specific user
register nodes	Register hardware nodes (slaves) to allow them to serve the platform
unregister nodes	Remove hardware nodes from the platform
monitor nodes	Monitor all hardware nodes
inspect node sessions	Inspect training sessions and hardware usage of a hardware node

Table 3.2: Administrator use cases

The complete picture of use cases is summarized in Figure 3.1.

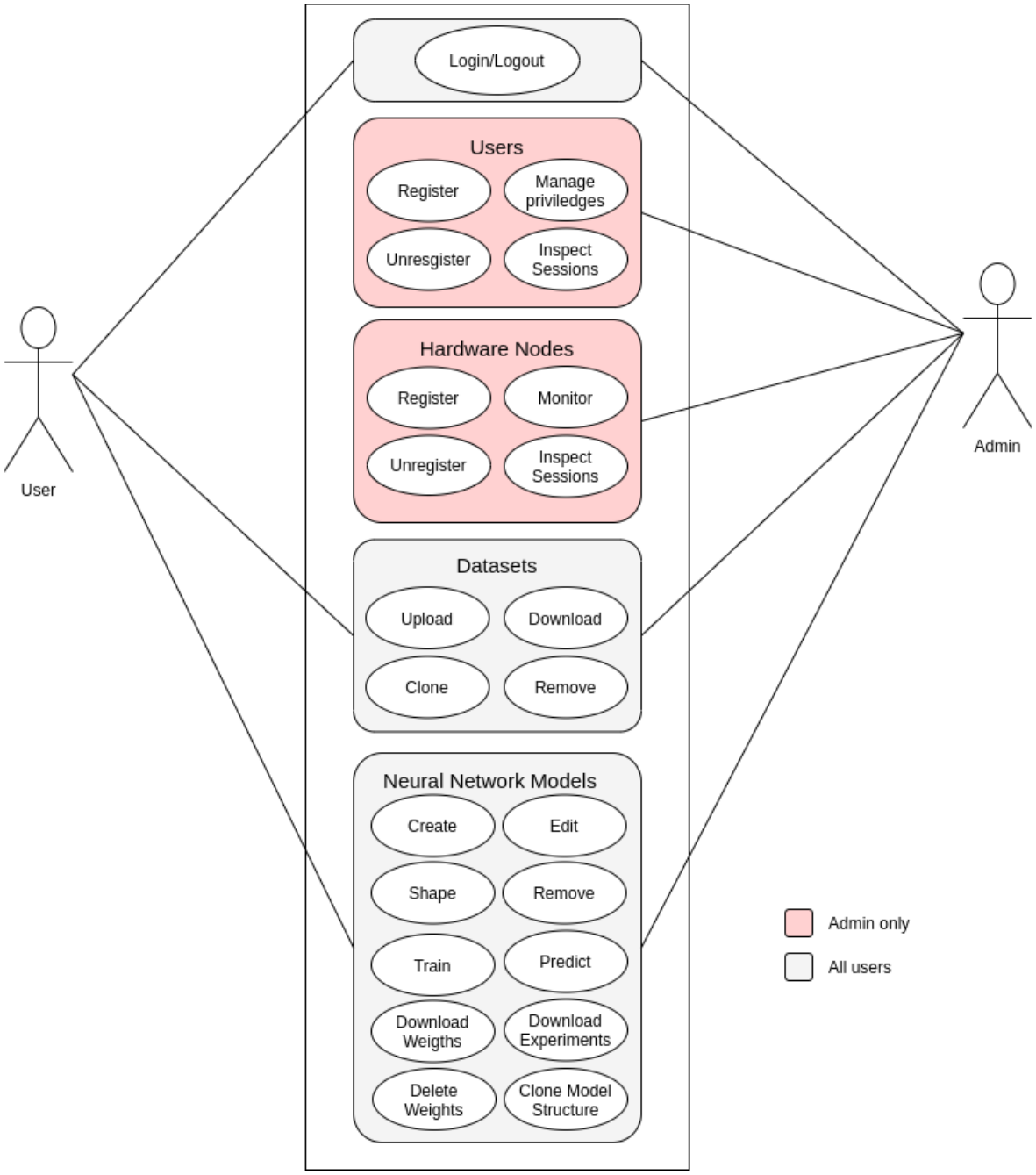

Figure 3.1: Use cases

3.2 Non-Functional Requirements

To provide contrast over the existing platforms, WebML needs to be equipped with the following non-functional requirements:

1. Performance;
2. Fault tolerance;
3. Data integrity and consistency;
4. Extensibility;
5. Usability.

3.2.1 Performance

High performance is a requirement for complex and highly computational platforms like WebML. The whole system must be fast enough otherwise it will start degrading its utility and value. The main points where speed was invested are the following:

- Internal network communication;
- Training Computations;
- Server to client communications;
- Internal data storage;
- Internal server caches;
- Database queries.

Performance also refers to the quality of the results, in this case, the quality and the final accuracy of the neural network after the necessary training sessions. For each of the available neural networks' formats, the best models and processing scripts must be analyzed and developed to improve the final model accuracy. Although the training scripts are optimized for DL, the user is still responsible for building the proper model structure.

3.2.2 Fault Tolerance

The possibility of a service failure should not be discarded, this should be taken into account by relying on backups or on fail-safe services. In WebML, the user should not perceive the internal failures of the system. When a hardware node from the internal network fails, the system architecture must be redundant and redirect requests to the available and functional remaining nodes without disrupting the service. This, however, is not possible for the neural network training sessions taking place at the moment of the failure. There must be a unique running service that is responsible for ensuring this replication and transparent behavior, assigning to this service the function of distributed system management.

3.2.3 Data Integrity and Consistency

The WebML platform must also be able to protect the data in the case of system failure, that is, the integrity and consistency of the data must be assured as long as all application servers have compatible versions with each other. Corruption of data may imply service failure and/or system restrictions derived from errors. The security mechanism cannot compromise the platform's performance and usability. WebML platform benefits from multiple layers of security dedicated to user requests, helping in prevention and identification of inconsistencies in the content of the data. Input validations and database constraints are an example of these layers of security, known to be an efficient security mechanism. Monitoring tools are also part of the WebML security system as they also help identify problems and prevent further harm to the platform.

3.2.4 Extensibility

WebML is not limited to just one hardware provider for predicting requests and training sessions. It allows multiple nodes (computational machines), with a variable number of GPUs of different series, to join the WebML network in order to work together serving multiple clients in parallel efficiently.

3.2.5 Usability

Finally, the application needs to be intuitive and easy to use. A new user should not feel confused when using the platform, instead, they should rapidly adapt to the provided tools in the UI. It is important that this UI abstracts the inside architecture and implementation that are not relevant, and only delivers the crucial information and tools regarding the workflow to the user. Although it should not be the main focus, neglecting the visual component of the platform could lead to less promising results regarding user interaction and could also result in less willingness when it comes to use the service.

3.3 Architecture and Framework Proposal

In order to build WebML, it was required to follow a distributed architecture. In this architecture, several computational service providers work simultaneously to achieve a specific objective.

Most of the back-end services complexity should be hidden to the regular user and monitored by the administrators. WebML requires three different services (unique server applications) to work properly: master, slave and the graphical interface web server (UI web server). For a full working system, both master node and the UI web server are unique in the network while multiple slave servers can coexist as long as they are settled in different computational machines. The summary of the architecture is displayed in Figure 3.2.

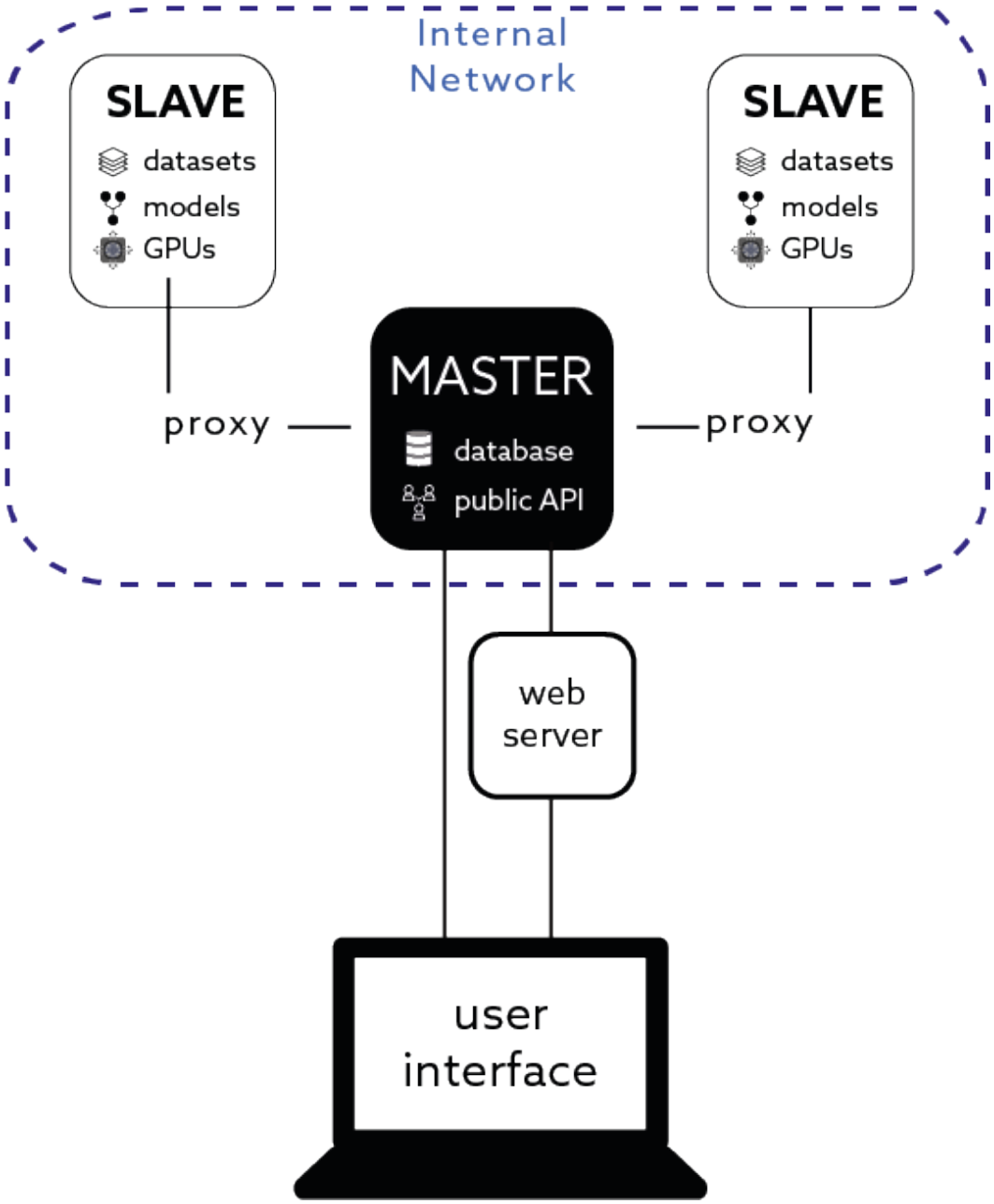

Figure 3.2: WebML Service Architecture

List of interacting components of Figure 3.2

- **Master server** - provides the logic of the distributed system, data management and provides the public API for the client;
- **Slave server** - provides the hardware required for neural networks training sessions and stores big datasets and models weights on its SDD storage;
- **Graphical Interface Web Server (Web server)** - provides the GUI to the user;
- **Client** - the client uses the platform by accessing it trough a web URL that refers to the graphical interface web server API.

3.3.1 Master

The main purpose of the master server is to proxy all communications between clients and hardware nodes (slaves) and also store metadata about users, models and datasets on its non-relational database. The master is responsible for all authentication and authorization procedures from the clients and the hardware nodes.

Several server application frameworks for the master node were examined. High performance and easy code maintenance are two requirements for this application server, as such, a few frameworks and technologies discussed in Section 2.5 were considered to respond to these needs.

Since the master node does not perform any heavy computations and its main responsibilities are real-time communications and proxy, the framework chosen to this task was ExpressJS. ExpressJS. It runs on a NodeJS environment that was designed with real-time, push-based architectures in mind providing a very maintainable code for web services.

ExpressJS library/framework is a very popular option for NodeJS server applications. It is minimalist, easy to read, and maintain web framework. It is possible to write a server code in 3 lines of code that return 'hello world' to the user with just 3 complete code instructions (Code 9).

```
const app = require('express')();
app.get('/', (req,res) => res.send('hello world') );
app.listen(3000, () => console.log('Listening on port 3000') );
```

Code 9: Express library - hello world

For real-time asynchronous communication between slave nodes and clients, the master node provides a WebSocket service, both clients and slave nodes connect to this service using an authentication token that identifies their role in the platform. Note that the same client user can be connected from multiple terminals while unique identified slave nodes can have a single connection only. WebSocket services are required in order to allow bidirectional communications, instead of regular restful API services that only enable requests from the client-side only.

To keep an efficient data management, master stores all the users and model related information in a non-relational database (MongoDB). At first sight a relational database like SQL seems more appropriated because of the relation of the content, but as the models have a very complex graph structure, they would make the retrieval of data very slow requiring multiple 'joins' and WebML does not require any N-to-M relations. Neural network models are stored as JavaScript Object Notation (JSON) format to be easily manipulated. The highest number of relations is related to the user collection, and aside from administrators, each user only fetches its own data. All user relative data must be indexed in their proper attributes (attributes that are used for database queries) in order to increase database data retrieval performance.

In MongoDB, each table is represented as a Schema Model (not to be confused with a neural network model). In order to keep everything robust and consistent, master node needs to define a Schema Model for each of the following items:

- datasets;
- logs;
- models;
- users.

A more detailed overview of each of schema models is introduced latter in Schema Modeling.

Regarding dataset content information, master node does not store the complete dataset content, only its metadata. This metadata refers to important properties like format, name, and Slave nodes that have the datasets on the local disk. This procedure is required in order to maintain consistency and avoid redundancy between different slaves as these can have the same datasets.

The same logic applies to the models (neural network graphs). The master server stores all the information regarding the use of the neural networks but not the weights after training (this requires extra storage in a separated file).

3.3.2 Slave

Slave nodes are responsible for the big data processing in training sessions and prediction requests.

Slaves have a reduced number of functions and their efficiency highly depends on the quality of the hardware. Each slave must contain a modern processor that can be efficient computing parallel processes, high RAM memory space and a fast Solid State Drive (SSD) and contain GPUs that are fast and that have a good amount of available internal volatile memory. The core functions are the following:

- store uploaded datasets;
- train model on a specified dataset and store its result;
- use models for prediction;
- evaluate a model performance.

From an architecture point of view, slaves require the biggest and most complex setup in order to function properly. Multiple software packages are a prerequisite including GPU drivers, CUDA and multiple ML packages.

Slave modules' extensive dependencies includes TensorFlow and Keras packages compatible versions that allow pipeline training flows and also access to multiple GPUs. These dependencies are not easy to account for without a proper package management. NVIDIA docker containers help significantly to this task as it can be very difficult to install all dependencies in the raw Linux system. In order to ease this process, a bash script is helpful to install every dependency required in a single command 'bash install.sh' including nvidia drivers, docker and nvidia docker. This script involves a small amount of artificial intelligence to cope with possible errors and with slightly different Linux systems.

Regarding the slave identification, each slave can be considered as a user with privilege 'Node', that can be authenticated with a specified token similar to a user. Each slave terminal (machine) needs to be registered by an administrator. There can only be one instance of a unique slave connected to the WebSocket pool of the master server.

Slaves store datasets content and model weights, these files can take up a significant amount of disk space creating a need to establish constrains on datasets and model size.

Each node is a process where the main thread is dedicated for the communication with the master and to the rest of the threads dedicated to the hardware execution scripts (training and prediction) being able to use multiple GPUs at the same time without having blocking issues.

The server that contains the API and the WebSocket connection to the master is programmed in JavaScript running on NodeJS environment for the same reasons as the master node. Having the same

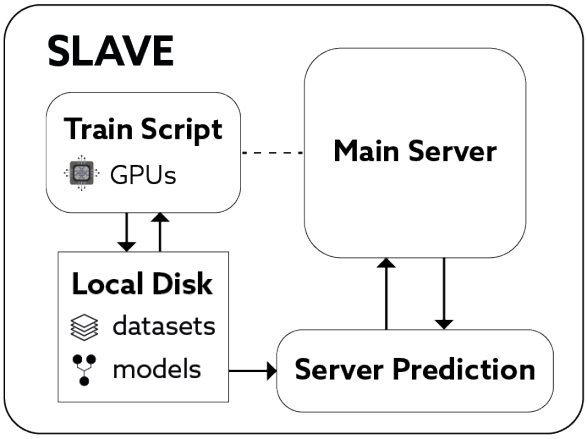

Figure 3.3: Slave architecture

programming language for multiple services is very helpful to increase development speed since several validation processes are shared by distinct application modules.

On launch, this NodeJS application server is executed along with an IO Python script thread responsible for model prediction using local saved models. This Python thread is launched in parallel with the main thread and are dependent on one another.

For training, the slave node contains a template Python script for each dataset format that obeys to different metrics in a local folder. When requested, a specific training script in a different programming thread is called and supervised in real-time to stream training progress to all users that have permission to use the master node as a data streaming proxy.

Datasets

Datasets' content and information are synchronized on the master node and slave nodes.

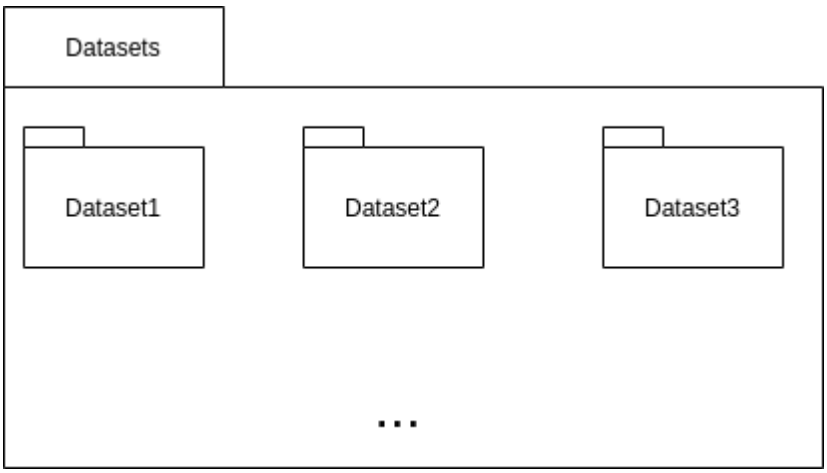

Figure 3.4: dataset storage

Datasets are stored as folders on the root folder of the slave server. Besides the data itself, each dataset should contain compact metadata locally along with the whole content summarizing all information necessary to evaluate the content inside (Fig. 3.5).

Figure 3.5: properties.json

Datasets can have different formats, and each format and its validations will be better explained in a later chapter: WebML Service Implementation.

Models

While neural network models metadata is stored on the master node, Slave nodes store the model weights (all the trained parameters of the model for each layer) in order to be able to use it locally on the datasets.

Each model's weights are saved when a training session is completed, these weights can later be deleted by the owner or be retrained to produce other even further improved trained weights.

Models weights are not the only content stored in order to make a model fully work, models can have different formats and computational graphs, each format may require an extra pipeline file to incorporate data normalization and one hot encoding (preprocessing technique). The details of these models will be explained later in Models.

3.3.3 Graphical Interface Web Server

This graphical web server is responsible to provide the UI containing the prepossessed HTML, CSS and JavaScript for the client in the browser. This is the less complex server in terms of communication and computation. This server is optimized for the user.

The web server works as a disposable component. Disposable in the sense that, even without this service, the rest of the system still works through the APIs and WebSocket service, it can be replaced by any other user interface web server that connects to the same service API.

From the three big JavaScript front-end frameworks (Angular, Vue and React), the framework of choice for front-end development is NuxtJS, which is an extension of Vue that uses NodeJS in the background.

Vue is the youngest member of the family of JavaScript frameworks. It has practically removed the drawbacks of the other frameworks to offer easy development tools to software developers. Although it is the least used framework of all time, it is most loved (highest ratings).

The main reason NuxtJS is most useful for WebML is due to its SSR properties. SSR is the process where the server sends all prepossessed data directly to the browser containing the final HTML/CSS layout. Server-side render data flow in WebML is displayed in Figure 3.6.

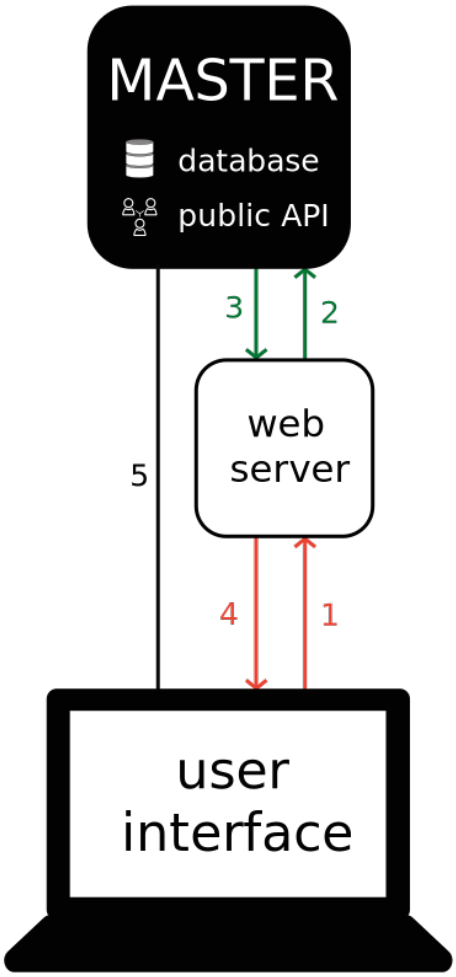

Figure 3.6: Server-side render flow

1. The client sends a request to the web server.
2. The web server sends an API request to the master node to fetch the user data.
3. After the web server processes all authentication procedures and retrieves data from the database it returns all the relative user data back to the Web server.
4. The Web server pre-renders all the Vue components and user content back to the Client
5. All the other requests aside from the first client's request are directly directed to the master's API.

It is important to notice that the graphical web server and the master node could be merged into a single server module to slightly accelerate this first request. However, in this case, this merged server would have all the responsibility and the UI would not be disposable anymore. If the speed is a big concern, the web server can be launched in the same machine as the master node and proxy server-side rendering requests via the localhost network.

3.3.4 Client

The client is the device that fetches data via web URL from the web server mentioned above. This client keeps an open WebSocket connection to the master node so that it receives real-time content and training progress updates from the slave nodes. In order to have access to any of the content inside the page, each client needs to login first. After the login is completed, a cookie with an authentication token is stored under the GUI server's domain. This procedure enables page refresh without the need to re-login each time and will have a faster user-related data fetch due to the SSR described above. Every other model API related communication is performed directly to the master node resulting in a faster response. Each client possesses an associated privilege, it causes action limitations and displayed information restrictions. For instance, admins have detailed information about all the slaves connected and all the users that are using it or have used it before. A more detailed explanation about this user interface will be shown at Chapter WebML Results.

3.3.5 Architecture Summary

The architecture itself is relatively complex due to the possibility of computation power upgrades and availability (number of slaves connected to the network). All slave node terminals and clients are connected to the master through its WebSocket server service so that all the communications work in real-time. This distributed system works with high data streaming communications that are all processed and proxied via the master node for data consistency. The three application modules are built in a way to improve communication speed for the clients in non-blocking requests and executions.

This architecture keeps a good redundancy level in case a slave stops working since it can redirect the training and prediction tasks to the other Slave nodes in the network. The graphical interface web server is disposable and may be replaced using the API documentation from the master server alone. It may be tricky to be able to improve this system to have a master node redundancy because having more than one master node would require a much more complex robust abstract coordination WebSocket communications between clients, slaves and themselves. The current architecture makes it impossible for WebML to be used in case the master gets shutdown and may only be reused when it restarts.

Each application module must be developed using the right frameworks for the task. On the server-side logic, the chosen framework for the main process of the master and the slave modules is ExpressJS that runs on top of NodeJS environment coded in JavaScript. Slave modules also require proper frameworks for their DL logic. Keras is the recommended DL tool for the task since is a high-level layer-oriented framework for deep neural networks and one of the most used worldwide. For the front-end development, Vue is the selected framework over the other two competitor frameworks, React and Angular. Vue alone does not allow SSR and its file organization is not as organized as NuxtJS, its extension.

A visual representation of the frameworks of WebML architecture can be seen in Figure 3.7.

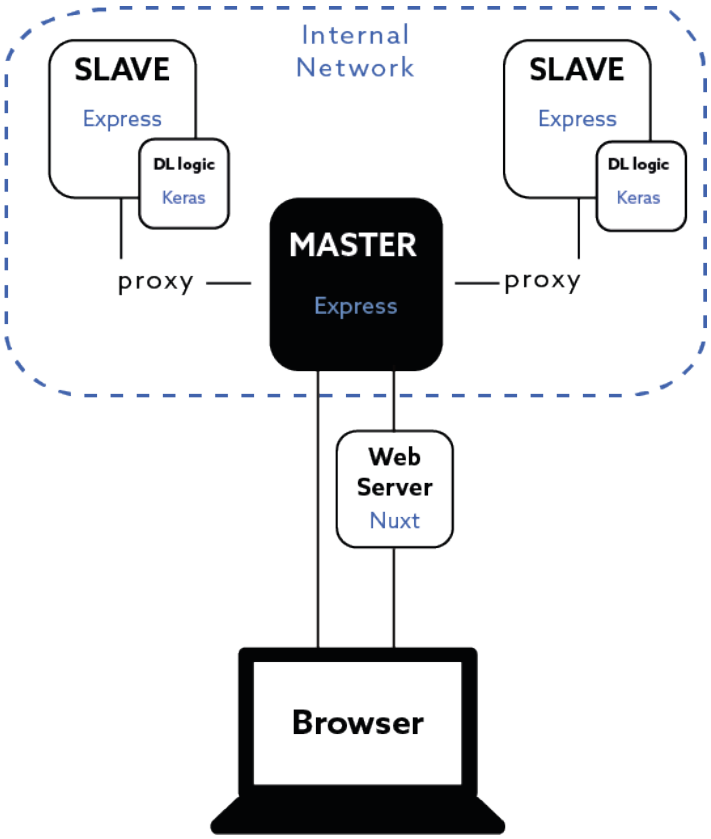

Figure 3.7: properties.json

3.4 Schema Modeling

The full schema model of WebML is quite simple. Depending on the requirements, non-relational databases can significantly help to reduce the complexity and number of tables. The complete schema is displayed in Figure 3.8.

The model schema (neural network schema) is the most complex in terms of validation. The model property of the Model schema is a complex JSON object that describes the structure of a neural network.

As mentioned previously, slaves need a special authentication to be allowed to connect to the network. For the purpose of simplification, they are handled in a similar fashion as the users. Although a new table collection 'Nodes' or 'Slaves' could be created for this purpose, it is more advantageous to be considered as a user in the database. One of the reasons is that computational machines (slaves) require human interaction for the installation and deployment procedures since they cannot set themselves up. Another reason is due to the fact that the authentication process (login) is the same as the user or the administrator, which imposes responsibility upon someone on the service provided when active.

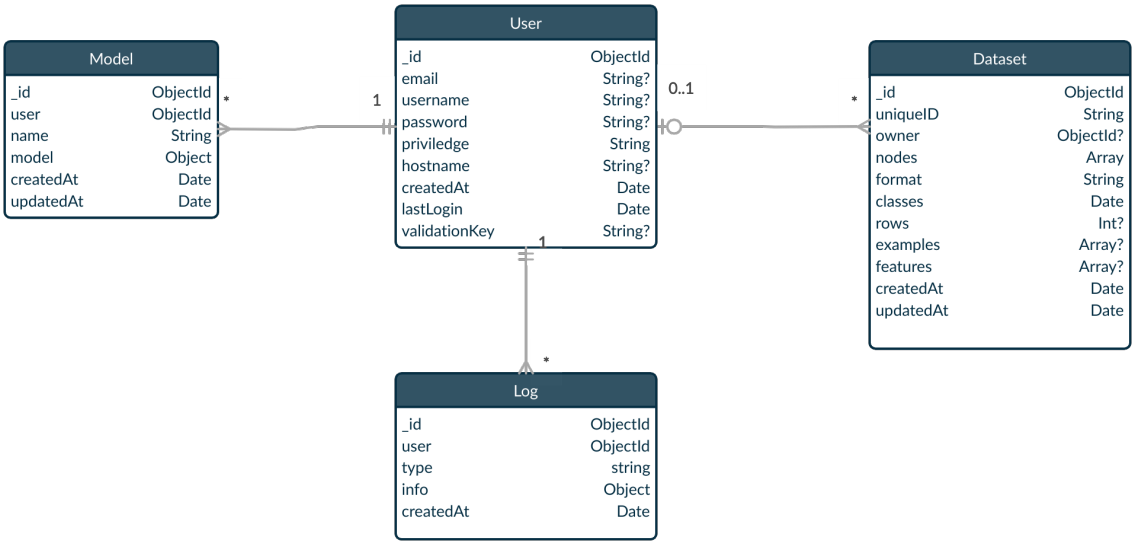

Figure 3.8: MongoDB class model

4

WebML Service Implementation

This section describes in detail the implementation of major components and features of WebML's back-end services.

4.1 WebSocket Service

WebSocket services are simple and efficient when running on a NodeJS environment using the 'socket-io' library. The code displayed in Code. 10 shows that even a server running from a compact script like this can use the same port as the restful API service. This makes it possible to deploy both services under https (secure http) protocol using the same URL.

```
const http = require('http')
const io = require('socket.io')
const app = require("express")()

const server = http.createServer(app)
const socket = io(server)
socket.on('connection', usersocket => {
    console.log('A client connected');
})
server.listen(3000,() => {console.log('Listening in port',3000)})
```

Code 10: Socket-IO code

The WebSocket service is implemented on the master node via the NodeJS environment. WebSocket's clients are served through the browser or NodeJS environment (case of slave server application). While a slave can only be connected once, an individual user can be connected multiple times. Opening another tab in the browser is enough to have more than a single connection. WebML needs a fast message delivery to the connected parties. In order to guarantee fast delivery to all the users in the pool, efficient data structures are required. For performance metrics, algorithms and data-structures are often classified using the 'big O notation'. Big O notation is used to classify algorithms according to how their execution time grows according to the input size [66]. In JavaScript, hash-maps can be represented efficiently as a single object (Code 11). A hash-map that uses users' unique identifier as keys and an array of each user connections as correspondent values allows communications to this user in O(k) speed, where 'k' is the number of the user connections.

```
{ //socketmap
    userid1: [ ...user1_socket_connections ],
    userid2: [ ...user2_socket_connections ],
    userid3: [ ...user3_socket_connections ],
}
```

Code 11: WebSocket user hash-map

Although this data structure is appropriated for regular users, it is not enough to ensure fast message deliveries to the administrators. Administrators receive the progress status of all training sessions from every user in the pool. Otherwise, they could not monitor hardware and models' progress. With this hash-map alone, redirecting a message to all admins would imply searching all the keys and verify users' privileges which would delay message distribution in O(n+k) time where 'n' is the number of all connected users and 'k' is the number of all administrator connections. Since this approach can be harmful to the server, administrators need to be mapped in a dedicated data structure. A simple array of administrators' identifiers for this matter is good enough, an array of administrators' connections is slightly better. Every piece of information that administrators should be notified of via WebSocket is streamed directly by iterating this connection array. Regarding this JavaScript's array structure, adding an administrator connection is performed in O(1), removing a connection is performed in O(n) where 'n' is the number of administrator connections. Slave node connections should also have a dedicated array following the same logic as the administrators for the cases when all the slave nodes need to receive update messages (Code 12).

```
[...admin_socket_connections] //Admin connections (array)
[...slave_socket_connections] //Slave connections (array)
{ //socketmap (javascript object/hash-map)
    userid1: [ ...user1_socket_connections ],
    userid2: [ ...user2_socket_connections ],
    userid3: [ ...user3_socket_connections ],
}
```

Code 12: WebSocket data structures

4.2 Service Security and Data consistency

Any web service nowadays greatly benefits from protection against Denial of Service (DoS) and any other cyber-attacks. Also, a robust data consistency is recommended for any distributed system project. WebML requires efficient and secure authentication and authorization due to its multi-user and multi-privilege nature.

4.2.1 Authentication and Authorization

WebML imposes three different privileges a user can have. One user cannot have more than one privilege.

- **Node** - slave node;
- **User** - regular user;
- **Admin** - administrator.

WebML stores user data in its non-relational database storing the password hashed. Hashing a password allows one-way authentication and prevents password cracking in case the database is

compromised. WebML validates authentication and authorization using the Json Web Token (JWT) method. JWT represents the entity claims, securely between two parties [67]. A JWT is generated in successfully login requests. These generated tokens are signed with the master node private key. An example of an JWT is demonstrated in Figure 4.1.

- the header, describes the type and signature algorithm applied to the token;
- the payload, or the body, describes the claims and privileges of the session owner in JSON format;
- the signature of the resulting encoded header, payload and the private key.

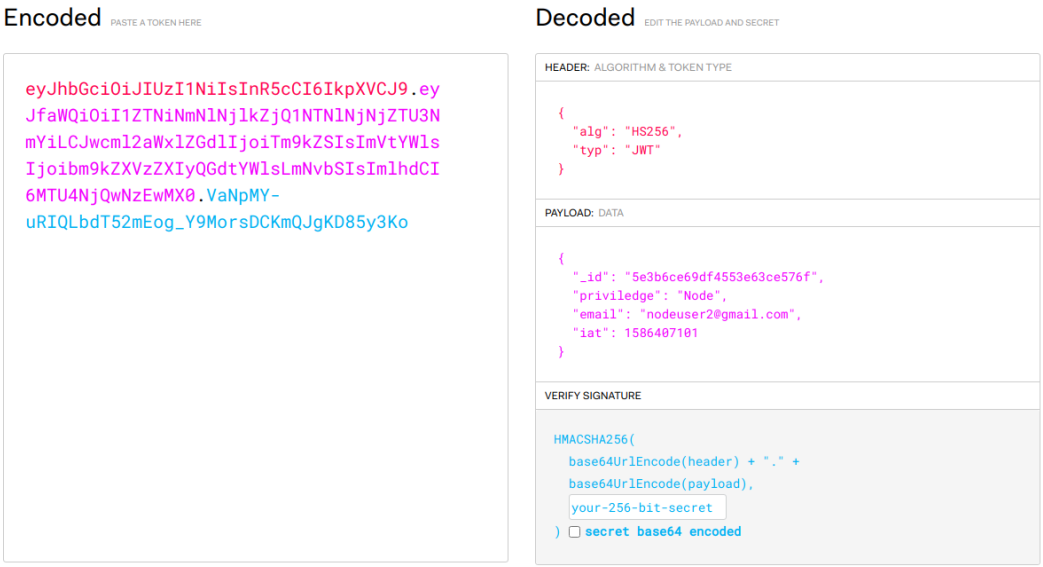

Figure 4.1: JSON Web Token

The login request procedure requires the user to send its credentials over a secure communication channel. A sequence diagram representing the workflow is demonstrated in Figure 4.2.

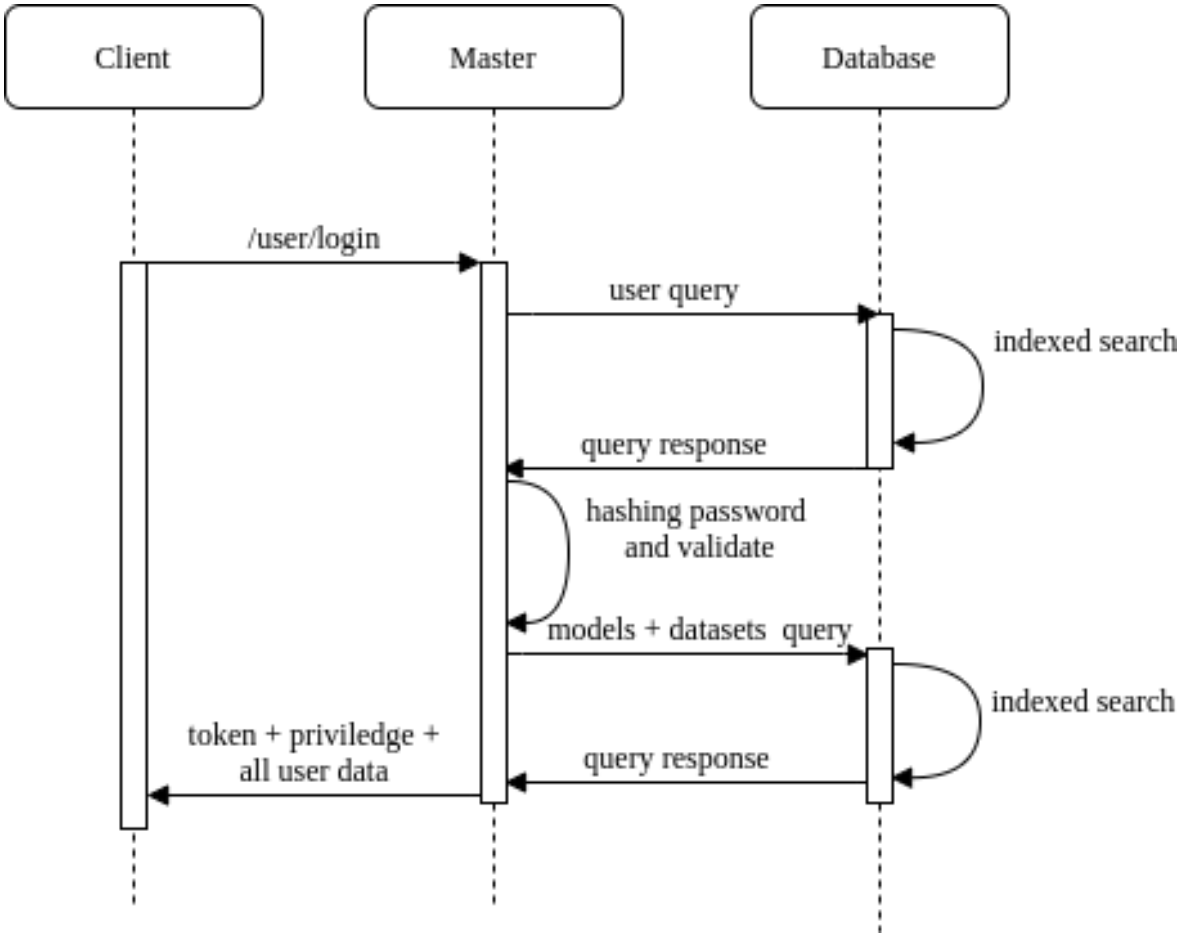

Figure 4.2: Authentication sequence diagram

Sending a JWT token over an insecure channel **allows** the attacker to steal the sender's identity while the token is valid. The attacker can use the the victim's active session, taking advantage of all of the victim's privileges with this token. Although this identity theft does **not allow** the attacker to compute the victim's password or the server's private key, it is still a major security breach. It is important to never perform any communication in an insecure channel in order to prevent severe damage involving theft of admin or node authentications. Login requests generate the token while all other API requests require this authentication token attached to their headers. WebSocket connections need to be authenticated as well. If performed in a secure channel (wss), these have the bidirectional communication advantage of requiring the authentication token procedure only once, right after the connection has been established. Since it is infeasible to intercept a ciphered connection, the connections stay authenticated until their closure. For each bidirectional WebSocket connection there is a respective user identification and privilege, or no authentication at all when the user is not logged in.

4.2.2 Creating and Registering Users and Slaves

The user is free to login after they are already registered. Since a visiting user is not allowed to register, it needs to request an administrator to deal with the registration procedure. How does the administrator generate a new user to the platform without actually fill in the full registration form (username + email + password)? WebML's registration process is built efficiently and securely, in such a way, that the administrator does not need to fill any of the user's fields at all. The only requirement

is to generate a new empty user instance to the database using the registration endpoint. When a non-validated user is generated, it is generated along with a validation key. WebML supplies the administrator with a registration link for that user containing this validation key in the URL query string instance. This link allows the user to properly register itself after filling the form with the three requested fields and send the registration request (username, email and password). Slaves need to be registered by the administrator as well. These follow a very similar procedure to the user creation. The only difference is the fact that the administrator needs to set the 'hostname' of the newly created slave as the only required field.

4.2.3 Service Endpoints and Validations

Validation is one of the most important perks any restful API must have, as it secures servers from running malicious or defective code on the back-end services. It also adds one extra data consistency layer to help ensure robust data in stored into databases.

The master server application must validate inputs from the user and from the slaves from all endpoints. The full list of master node endpoints is displayed in table 4.1.

Endpoint	Description
/user/login	login request
/user/renewdata	retrieve all user data
/user/create	create a non-validated user instance in the database
/user/register	register and validate a user
/user/delete	unregister a user
/models/save	save a model
/models/new	create a model
/models/download/:modelid	download model weights
/models/deleteTrainedModel	delete trained model
/models/downloadExperiment	download a model project/experiment
/models/train	request a training session
/models/reportTrainingSuccess	report a successfully training session (slave only)
/models/setName	set a model name
/models/setModelShapeCSV	shape a model to a dataset on csvdata format
/models/setModelShapeImage	shape a model to a dataset on image format
/models/delete	delete a model structure and all its trained models
/models/deleteTrainedModel	delete a trained model from the model's structure
/models/predictImage	request a image prediction
/models/predictCSVrow	request a csvdata image prediction
/datasets/sample/:dataset/:idx	get a random sample image request from a image dataset
/datasets/sample/:dataset/:idx/:file	get a specific sample image request from a image dataset
/datasets/csvUpload	upload a csv dataset
/datasets/imageUpload	upload a image dataset
/datasets/delete	remove a dataset

Table 4.1: Master service endpoint list

One of the most known and effective libraries for validation requests is 'Joi'. Joi describes data schemas using a simple, intuitive, and readable code made for JavaScript language.

In order to show some of the processes behind request validations, only a simple example and a more complex example will be presented to bring an abstract overview of the validation process. The simple example would be the validation performed in login requests on endpoint '/user/login' (Code 13).

```
const schema = {
    email: Joi.string().max(255).email(),
    password: Joi.string().min(5).max(255)
}
const { error } = Joi.validate(obj, schema);
```

Code 13: Validation on login requests - endpoint **/user/login**

After defining a schema, validating an object calling 'Joi.validate(obj, schema)' returns another object that contains a property error. This property is set to null if the input object is valid, containing the error message otherwise. Schema displayed on Code 13 forces an email and a password with a maximum of 255 characters each.

Training requests are complex to validate. It requires 3 steps to complete the whole validation process including the database requests (Code 14, Code 15 and Code 16).

```
const schema = Joi.object().keys({
    modelid: Joi.string(),
    trainedModel: Joi.string().allow(null).optional(),
    trainingParams: Joi.object().keys({
        num_epochs: Joi.number().integer().min(1).max(15).optional(),
        batch_size: Joi.number().integer().min(1).max(1024).optional(),
        shear_range: Joi.number().min(0).max(0.5).optional(),
        zoom_range: Joi.number().min(0).max(0.5).optional(),
        rotation_range: Joi.number().integer().min(0).max(45).optional(),
        horizontal_flip: Joi.boolean().optional()
    })
});
var { error } = Joi.validate(obj, schema);
```

Code 14: Training request validation step 1 - nested object validation

```
//req.user is the information retrieved from the user token
const modelObj = await Model.findOne({ user: req.user._id, _id: obj.modelid })

if(!modelObj) {
    return 'No model found'
}

const datasetObj = await Dataset.findOne({name: modelObj.model.dataset});

if(!datasetObj) {
    return 'No dataset found';
}
```

Code 15: Training request validation step 2 - Database validation

The second validation step forces a database query validation using mongoose library (mongoDB driver). First, it retrieves the model object from the database and validates the user ownership, then it queries its correspondent target dataset for further information to proceed with the training session.

```
const paramsSchema = Joi.alternatives(
    Joi.object().keys({
        format: Joi.string().valid('images'),
        trainingParams: Joi.object().keys({
            num_epochs: Joi.number().optional(),
            batch_size: Joi.number().optional(),
            shear_range: Joi.number().optional(),
            zoom_range: Joi.number().optional(),
            rotation_range: Joi.number().optional(),
            horizontal_flip: Joi.boolean().optional()
        })
    }),
    Joi.object().keys({
        format: Joi.string().valid('csvdata'),
        trainingParams: Joi.object().keys({
            num_epochs: Joi.number().optional(),
            batch_size: Joi.number().optional(),
        })
    }),
)
var { error } = Joi.validate(obj, paramsSchema);
```

Code 16: Training request validation step 3 - alternative validation

In the last training request validation step (Code 16), there are different schema validations for image format and csvdata format, since images training request may have been supplied with more training parameters. Joi allows multiple alternatives for all the different request possibilities. This allows for a robust and easy to read code when compared to using multiple conditional validations.

4.3 Hardware Monitoring and Management

There are several hardware requirements that need to be fulfilled in order to ensure proper functioning. Hardware monitoring is one of the main features that WebML offers to the administrators and to self-organization. The strategy chosen was to take advantage of the resulting output of specific dedicated Linux commands and turning them into useful information as an outcome of properly applied processing. This post-processed information about the current hardware status is streamed on each update, to all administrators connected via the WebSocket pool. The hardware components that are monitored are the secondary memory, the CPU and the GPUs. In WebML, CPU information is static and the only retrieved information is the CPU model name and its correspondent number of threads. On the other end, both secondary memory and GPU status are non-static and are streamed in real-time through the WebSocket service.

In WebML, every slave node contains at least one GPU. There no is limit to how many slaves can join the network and the same applies to the GPUs. Increasing the number of slave nodes will result in an increased number of GPUs available to provide their processing power for training sessions.

For each GPU, there is a correspondent maximum amount of memory that usually depends on the GPU model (Table 4.2). Each training session occupies a portion of this memory. A training session will be refused via hardware when there is not enough available GPU memory to allocate. For WebML services, both the performance and maximum memory of the GPU is very important. The speed (performance) of neural network training sessions is influenced by the clock frequency and mainly by the available CUDA cores on the GPU. The amount of possible training sessions running in parallel is relatevly proportional to the sum of the available memory of all GPUs available to be used on WebML.

GPU	Memory (Frame Buffer)	CUDA cores
GTX 1050	2GB/4 GB	640/768
GTX 1060	3GB/6GB	1152/1280
RTX 2060	6GB	1920
RTX 2070	8GB	2304
GTX 1080	8GB	2560
RTX 2080	8GB	2944
RTX 3060	8GB	3840
RTX 3070	8GB	5888
RTX 3080	10GB	8704

Table 4.2: List of Nvidia GPUs (GTX and RTX series)

In order to gather information about all GPUs current status, Nvidia provides a useful bash command in Linux operative systems: *nvidia-smi*. Executing this command will result in a multi-line output with detailed information about the current GPUs' status (Fig. 4.3).

```
$ nvidia-smi
Sun Dec  6 02:55:08 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 455.45.01    Driver Version: 455.45.01    CUDA Version: 11.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  GeForce GTX 1050    On   | 00000000:01:00.0 Off |                  N/A |
| N/A   49C    P0    N/A /  N/A |    593MiB /  2000MiB |      1%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A      1794      G   /usr/lib/xorg/Xorg                 28MiB |
|    0   N/A  N/A      2131      G   /usr/bin/gnome-shell               48MiB |
|    0   N/A  N/A      3553      G   /usr/lib/xorg/Xorg                137MiB |
|    0   N/A  N/A      3740      G   /usr/bin/gnome-shell               90MiB |
|    0   N/A  N/A      4130      G   ...gAAAAAAAAA --shared-files       22MiB |
|    0   N/A  N/A      9356      G   ...AAAAAAAA== --shared-files      259MiB |
+-----------------------------------------------------------------------------+
```

Figure 4.3: nvidia-smi output

On Figure 4.3, only the red highlighted information is useful to WebML. It is important to mention that every GPU on the device can be referenced by a fixed integer index starting from 0 that will never change unless the hardware is manually modified. The master node demands the slave to deliver information regarding this index and the name of all available GPUs before it can join the hardware provider pool using the WebSocket service. This information can be extracted from the left side of "nvidia-smi" output (Fig. 4.3). Slaves also monitor hardware regularly for the currently used GPU memory by executing and extracting information from this output in a short interval. In order for the retrieved information to be efficient and scalable, there is a need to execute more complex bash instructions. One of the more efficient ways to extract specific information from a big output result is to use a regex expression to filter the target slice. Bash instructions along with regex expressions can sometimes become illegible depending on the complexity of the task (Code. 17).

```
#small slice of the monitoring bash code
step1=$(nvidia-smi -g $idx | grep ^'|'.*MiB.*/.*MiB.*'|'$ | tr ' | ' '\n')
step2=$(echo $step1 | grep [0-9]MiB | tr '\n' '//' | sed 's/MiB//g')
```

Code 17: Regex expression to retrieve GPU memory information.

Slave nodes have an internal state regarding all the correspondent GPUs. This internal state is updated by continuously retrieving the information from this monitoring tool every 0.75 seconds. The master node should be notified of updated internal states from the slaves. These notifications are not continuous since they would add significant overhead considering that WebML has no limit to the number of slaves that can be connected to the pool of hardware providers. Instead, the master node is only notified on GPUs that suffered significant changes on their current 'used memory' value relative to the last notification. Each of these updates to the master server are distributed to all connected administrators. Administrators are supplied with the information regarding all the slave nodes, their hardware usage and all the running training sessions. Training sessions are the only process that requires GPU. The amount of memory required for a session depends on the complexity of its correspondent model and the size of the target dataset.

4.4 Model-Dataset Dependency

Models are built to learn and auto-evaluate from datasets. This implies that models are dependent on datasets in order to be trained. For a specific format of a dataset, it is attributed to a specific format of neural network. Different datasets on the same model structures force different objectives, which means the model structure itself is not enough to assume or deduce its true objective and coverage. The real objective of a trained model also depends on the datasets it has been trained on. For instance, the same model structure may serve the purpose of distinguishing between cats and dogs (dataset A) but can also be used to distinguish between cars and boats (dataset B). The selected dataset (A or B) is what defines the objective (Fig. 4.4).

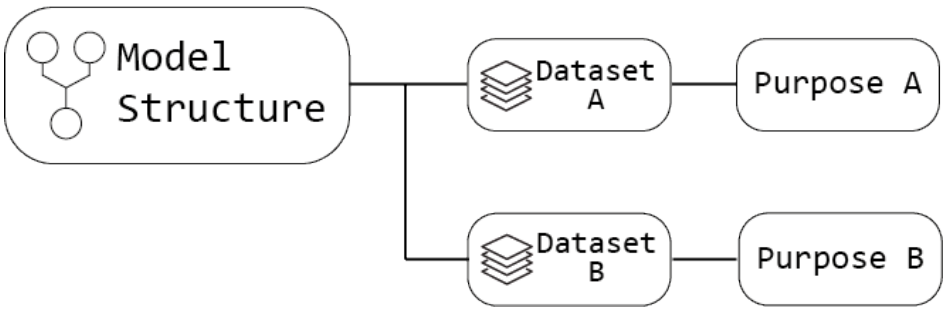

Figure 4.4: Models and dataset dependency

To cope with this fact and to help avoid training errors from the users, a neural network structure can only be applied to a single dataset. Although this restriction prevents training on multiple datasets, it sure helps to prevent multiple deceiving training sessions. In an extreme case where the user wants multiple data from multiple data sources to be trained on, a better approach would be to merge all these multiple data sources into a single dataset. This merge procedure results in better performance, homogenizing the training.

There are two formats available for both datasets and models, they are required to be the same in order to be compatible with each other, **images** format and **csvdata** format.

4.5 Datasets

A dataset is a collection of data. In the case of tabular data (Fig. 4.5), a dataset represents database tables, where each row corresponds to a given record of the data set in question and every column of a table represents a particular feature. Datasets can also consist of a collection of documents or files [68].

row0	feature1	feature2	feature3	feature4	...	featureN
row1	v11	v12	v13	v14	...	v1N
row2	v21	v22	v23	v24	...	v2N
row3	v31	v32	v33	v34	...	v3N
row4	v41	v42	v43	v44	...	v4N
...	...	...	...	...	...	...
rowN	vN1	vN2	vN3	vN4	...	vNN

Figure 4.5: Tabular data format

WebML supports the two dataset types mentioned (images and Comma Separated Values (CSV)). Slave serves store the dataset's content, they have the responsibility to synchronize all data between themselves with the help of the master node. This synchronization is reinforced by an intelligent streaming distribution algorithm. The bases of this algorithm consist on the assurance that each dataset is converted to a compressed format (.zip), so it can be broadcast to all candidate slaves faster via HTTP/HTTPS protocols. One big disadvantage that comes with this approach is the high amount of duplicated data, for example, a Slave with 30GB of datasets forces all other Slaves to have the same 30GB of the same data to allow training sessions on any Slave. A workaround for this issue is to limit the number of slaves that have a copy of the same dataset, selecting only the top free disk space available.

The master server also plays a role in caching recently uploaded datasets (compressed) until the scheduled datasets sharing is completed to all slave nodes.

4.5.1 CSV Datasets

Datasets stored in CSV format are considered to be tabular data. For the sake of data processing consistency, each CSV dataset reserves its first row for the header (column names) as displayed in Figure 4.6.

Data Preview

RowNum...	Custom...	Surname	Credit...	Geogra...	Gender	Age	Tenure	Balance	NumOfP...	HasCrC...	IsActi...	Estima...	Exited
1	15634602	Hargrave	619	France	Female	42	2	0	1	1	1	101348.88	1
2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
3	15619304	Onio	502	France	Female	42	8	159660.8	3	1	0	113931.57	1
4	15701354	Boni	699	France	Female	39	1	0	2	0	0	93826.63	0
5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.1	0

Figure 4.6: CSV examples

When a csvdata dataset is inserted, a Slave node inspects the whole file, attributing types and restrictions to each column. Each column can have one of the following data types:

- **Int** - Integer values only;

```
{
 "format": "csvdata",
 "rows": 10000,
 "examples": [...
 ],
 "features": [
  {
   "feature": "RowNumber",
   "isLabel": false,
   "isNumber": true,
   "isFloat": false,
   "dataType": "Int",
   "blocked": false,
   "oneHotEncoding": 10000,
   "uniqueValues": 10000
  },
  {
   "feature": "CustomerId",
   "isLabel": false,
   "isNumber": true,
   "isFloat": false,
   "dataType": "Int",
   "blocked": false,
   "oneHotEncoding": 10000,
   "uniqueValues": 10000
  },
  {
   "feature": "Surname",
   "isLabel": true,
   "isNumber": false,
   "isFloat": false,
   "dataType": "Label",
   "blocked": true,
   "oneHotEncoding": 0,
   "uniqueValues": 2932
  },
  ...
}
```

Code 18: properties.json file from Figure 4.6

```
pipeline = Pipeline(steps=[
    ('one_hot_encoder', OneHotEncoder(categories='auto')
    ('scaler', StandardScaler()),
    ('classifier', classifier)
])
```

Code 19: Prepossessing + Training pipeline structure

- **Float** - Decimal values;
- **Label** - String values, these need to be one hot encoded in order to be valid for the neural network;
- **Binary** - 0 or 1 values.

After the referred dataset inspection, the result is then stored on a properties.json file (Code. 18).

CSV data requires a significant amount of preprocessing in order to be prepared to be inputted into the neural network. Data of the type 'label' is not a valid candidate for a neural network input, it needs to go through a 'one hot encode' process.

To accomplish these requirements, it is possible to create a pipeline structure that can store the whole training flow using a Python library API named by Sklearn (Code 19)

The purpose of a pipeline is to assemble several steps that can be cross-validated together while

setting different parameters. The result of the training sessions of this model can be stored in a single file following a binary format with some tricky memory-settings.

4.5.2 Image Datasets

Images correspond to a 3-dimensional data input, dimensions being width, height and channels, where channels represent the color pigments Red-Green-Blue color model (RGB) of each pixel coordinate (width, height). In order for the images to be properly labeled and interpreted, they require a consistent non redundant folder schema (Fig. 4.7, and example in Figure 4.8), where 'training_data' and 'validation_data' folders are required along each of the classes containing the correspondent images inside. This format is defined by the Keras API library.

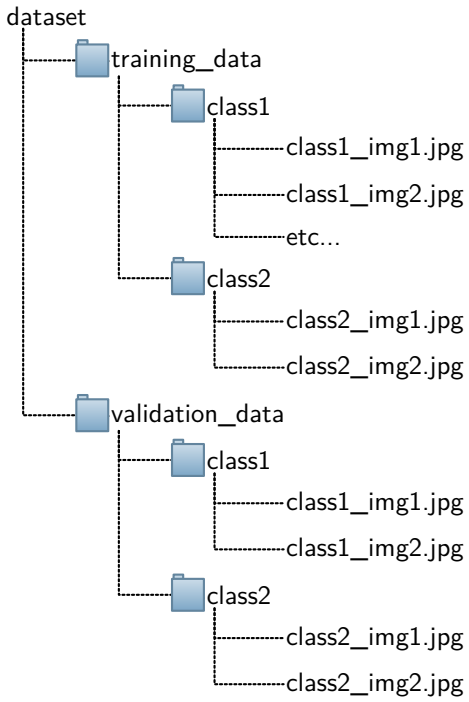

Figure 4.7: Images format directory structure

Figure 4.8: Images format dataset example (cats and dogs)

4.5.3 Dataset Ownership and Storage

Datasets can either be public or belong to a specific user. There are several issues to take care off in order to address this behavior. The same user cannot have two datasets with same name in their library. This means that they cannot upload a new dataset with an existing public dataset name set neither a currently owned dataset name. Selecting a new dataset name triggers 3 validations on its string value:

- Restrict allowed characters to prevent forbidden name that can have conflicts on file's locations;
- Verify public dataset name match - the name cannot be equal to a public dataset's name;
- verify whether the dataset's name matches any of the dataset's name owned by the current user - a user cannot have 2 datasets with the same name.

In order for this verification to be efficient, there should be a database (MongoDB) indexed search algorithm to speed up the process. Querying the inputted dataset name (indexed) in the database is not fast enough since multiple different users can have a dataset with the same name resulting in O(n) query speed where 'n' is the number of datasets from different users with the same name. A better approach would be to create a compound index using the name and owner fields which results in O(1) speed since these two keys together are a unique compound key of a dataset. Although this greatly accelerates the querying speed for name verification, it is more advantageous to create a single uniqueID key. This key is formed by concatenating the owner and dataset name with special characters that are forbidden in the name itself (Code. 21).

```
if(owner==null) { //public
    uniqueID = datasetname;
} else {    //private
    uniqueID = '$'+owner+'/'+datasetname;
}
```

Code 20: Defining unique key of a dataset

This uniqueID brings other advantages aside from the index query key in the database. Slave servers also need to store datasets properly and efficiently. In a Linux directory structure, it is forbidden to store 2 folders with the same name within the same folder. To take advantage of the uniqueness of the dataset's uniqueID, the relative path of the dataset directory can be the same as its own identifier. Since for each user, all private dataset names are required to be different, slaves can store all datasets of the user in the same folder which is named with the character '$' concatenated with the id of the user. A demonstration of this storage can be seen in Figure 4.9. Note that in this example '$5eb16aafe51bd563a4ec3b68/dataset1' is the uniqueID in mongoDB database and it is also the relative path of the dataset. This is possible because of the name restrictions imposed on the user when a dataset is uploaded.

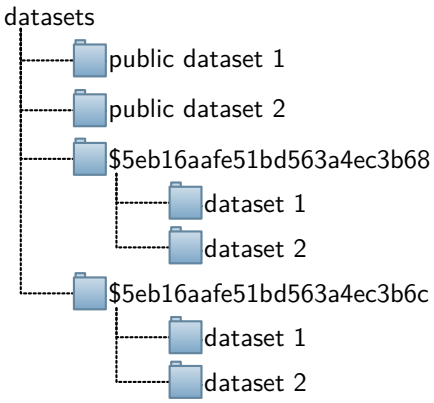

Figure 4.9: Dataset slave directory structure

Slave nodes perform multiple disk queries to build efficient data structures regarding the local directories in its dataset folder. After this information is gathered and processed properly it is sent to the master node in the WebSocket login request. The master server is then responsible to update the MongoDB database consistently, updating all of the dataset information regarding the content sent through a slave authentication request (Code. 21). These database operations are required to be atomic (all or none) using database transactions, if any of them fails, all updates are rolled back.

```
await Promise.all([
    Dataset.updateMany({
            uniqueID: { $in: [...slaveDatasets] }
        },
        {
            $addToSet: { nodes: nodeid.toString() }
        }
    ), //case 1: add this slave to all its stored local datasets
    Dataset.updateMany({
            nodes: nodeid.toString(), uniqueID: { $not: { $in: [...slaveDatasets] } }
        },
        {
            $pull: { nodes: nodeid.toString() }
        }
    ), //case 2: remove all the datasets that this slave no longer owns
    Dataset.insertMany(
        slaveDatasets.filter(d => !masterDatasets.has(d)).map(d => datasetObj[d])
    ) //case 3: insert new datasets
])
```

Code 21: Using mongoose to update dataset relative data

4.5.4 Dataset Upload

Uploading a dataset is one of the user's main use cases. Without the possibility of uploading a dataset, it would be impossible to train a model on different datasets aside from the public ones.

The implementation of dataset upload requests requires several procedures. Datasets are uploaded from the client browser to the master node API. The first step of completing this task is the validation of the dataset content. The uploaded dataset will be discarded if the dataset is corrupted or it is not in a valid format. It is important to note that an owner is assigned to the uploaded dataset and that this owner corresponds the user that requested the upload.

The overall list of a dataset upload request is the following:

1. Validate the uploaded dataset;

2. Generate properties.json file containing the dataset metadata;
3. Create a zip file containing the content of the dataset and the generated metadata file;
4. The master server then uploads this zip file (dataset) to all candidate slaves in the pool, one by one;
5. Each slave node unzips the uploaded zipped dataset file into the local dataset directory structure presented previously in Figure 4.9;
6. After the unzipping process, the slave server updates its local variables regarding the local content and notifies the master server that this dataset is ready to use;
7. The first slave notification of the complete dataset integration will trigger the storage of the dataset metadata into the MongoDB database;
8. The master then notifies that the dataset is ready to be used to the owner and the administrators using its WebSocket connections.

A sequence diagram of this procedure is represented in Figure 4.10. Items 1-4, 7-8 are executed by the master server while 5-6 are executed by the slave nodes. The dataset can be used by the platform as soon as the first slave to integrate the dataset finishes its procedure. The uploading task from the master node to the remaining candidate slaves is processed asynchronously in order to prevent that other services are blocked. Zipping and unzipping processes are the most computer intensives tasks on this uploading procedure as they can occupy a substantial amount of CPU and file system processing percentage.

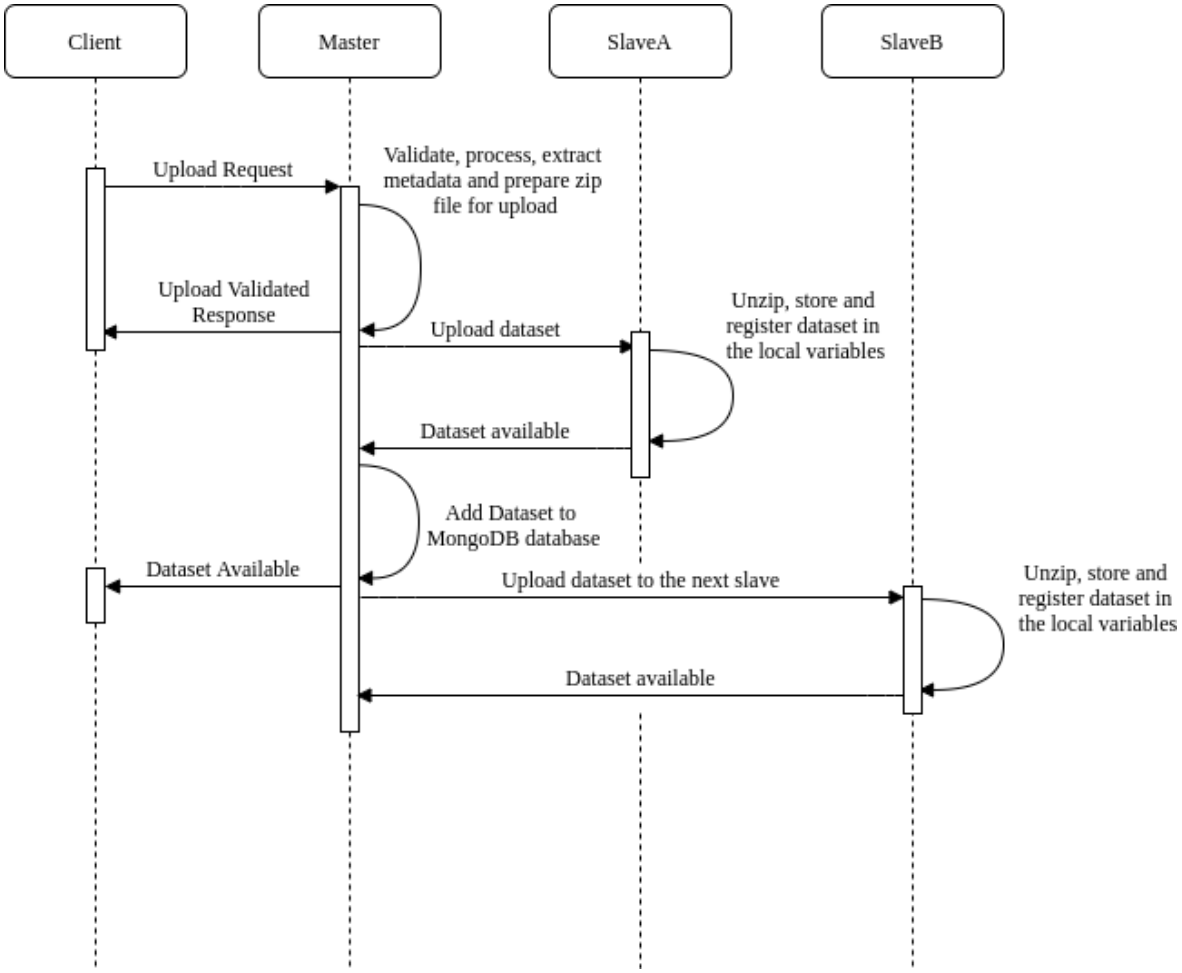

Figure 4.10: Sequence diagram of dataset upload

Different dataset formats follow different procedures in items 1-3 described in the list above.

Since csvdata datasets are uploaded in 'utf-8' format of a CVS file (.csv), they can be validated, inspected and evaluated directly from the uploaded file. A local temporary directory needs to be created in order to zip the file along with the generated metadata file (properties.json).

On the other end, image datasets are already zipped when they are uploaded. Unzipping and then zipping again would have significant unnecessary costs in the processing time and resources. Instead, the uploaded zipped file is analyzed and evaluated directly allowing the creation of the metadata file. This metadata file can then be directly inserted into the zip file using the bash command line.

4.6 Models

Models or neural networks are the main feature of WebML's platform. Each user can have a collection of model structures and their correspondent trained models. Models' structure are stored in the master server database while the trained weights that result from successfully training sessions of these structures are stored on slave's local filesystem. The main goals of WebML's models are the remote training and predictions.

4.6.1 Model Structure

Neural network structures are the specification of their input and output size, all the number of hidden layers in the neural network, its activation functions and learning loss functions. Neural network model structures can have multiple ways to be represented, they can be represented as code (Fig. 2.6), json or in binary format. Model structures in WebML are stored in a database collection called 'Model', every structure has an owner, creation date, last updated date, and the whole model's layers structure. In WebML, the master node stores the structure of a neural network in a MongoDB database as shown in (Code. 22).

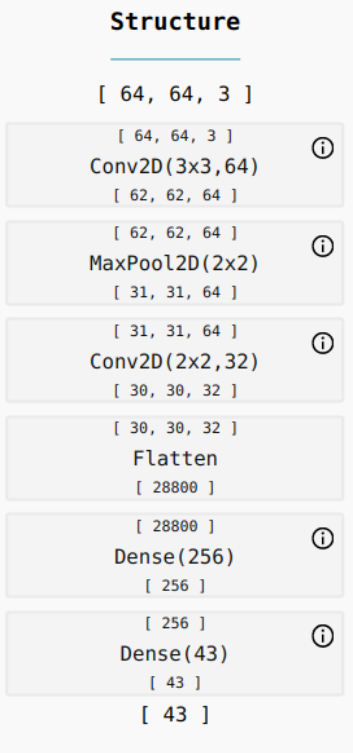

Figure 4.11: Convolution example model representation referred in (Code. 22)

```
{
    "_id" : ObjectId("5f32b646055b2a53ecf8c42c"),
    "trainedModels" : [    ],
    "user" : "5eb16aafe51bd563a4ec3b66",
    "model" : {
        "name" : "miniVGG-traffic",
        "size" : [
            64,
            64,
            3
        ],
        "optimizer" : "adam",
        "dynamic" : false,
        "loss" : "categorical_crossentropy",
        "layers" : [
            {
                "type" : "Conv2D",
                "layer" : {
                    "filters" : 64,
                    "kernel_size" : "3",
                    "padding" : "valid",
                    "activation" : "relu"
                },
            },
            {
                "type" : "MaxPooling2D",
                "layer" : {
                    "pool_size" : 2,
                    "padding" : "valid"
                },
            },
            {
                "type" : "Conv2D",
                "layer" : {
                    "filters" : 32,
                    "kernel_size" : 2,
                    "padding" : "valid",
                    "activation" : "relu"
                },
            },
            {
                "type" : "Flatten",
                "layer" : { },
            },
            {
                "type" : "Dense",
                "layer" : {
                    "activation" : "relu",
                    "units" : "256",
                },
            },
            {
                "type" : "Dense",
                "layer" : {
                    "activation" : "softmax",
                    "units" : 43
                },
            }
        ]
    },
    "createdAt" : ISODate("2020-08-11T15:12:59.204Z"),
    "updatedAt" : ISODate("2020-08-11T15:17:17.391Z")
}
```

Code 22: Convolution model structure json representation, graphical representation in (Fig. 4.11)

4.6.2 Model Object and Layer Validations

It is important that models are validated first in every request regarding the models like training sessions and prediction requests. Errors may accumulate, persist and be hard to track if the validation procedure is not performed properly.

In order to validate a model structure, every layer needs to be validated depending on the layer type. There is a list of restrictions in the model objects that need to be reinforced. The first step to validate the model is to use the previously discussed library Joi. The model structure object is the most complex object to validate since there are 3 different states regarding its representation.

- dynamic models;
- images format models;
- csvdata format models.

Some properties of the model object are mandatory, others depend on the model state. For example, while the model is in dynamic state it will not have the dataset property attached to it. After a dynamic model is shaped into a non-dynamic state, the list of its valid properties will depend on its format. The list of the model object keys that are required independently of the model's state are the following:

- layers - array representing the actual learning structure;
- size - array representing input size dimensions;
- optimizer - name of the model optimizer;
- loss - name of the loss function.

The full overview of the different possibilities regarding the different model states can be studied if inspected carefully in Code. 23.

WebML supports the most used types of neural network layers for both image classification and csvdata inspection, the full list is represented in Table 4.3.

Layer name	Supports csvdata	Supports images
Conv2D		X
MaxPooling2D		X
Flatten		X
BatchNormalization	X	
Dropout	X	X
Dense	X	X

Table 4.3: Keras neural network support

In the validation procedure, the output size of each layer is calculated in order. The model must be invalidated if any of those output size fails to match the same dimension or if any of the dimensions size is smaller or equal to 0. A small overview of each layer's properties can be inferred by inspecting its correspondent layer validations in Code 24.

The full validation procedure is slightly more complex than what has been referred so far. There is also the need to prevent an excessive number of layers and number of matrices cells. If this issue is not taken into consideration, impossible model structures could be created by the users without

```
const schemaFraction = {
    layers: Joi.array().min(1).items(Joi.object().keys({
        type: Joi.string().valid(...possibleLayerTypes),
        layer: Joi.object(),
    })),
    size: Joi.array().items(Joi.number().strict().allow(null).integer()),
    optimizer: Joi.string().valid(...optimizers),
    loss: Joi.string().valid(...losses),
}
const schema = Joi.alternatives(
    { //1 - dynamic
        ...schemaFraction, //reusing keys
        dynamic: Joi.bool().valid(true),
        format: Joi.string().valid("images", "csvdata"),
    },
    { //2 - non-dynamic images
        ...schemaFraction, //reusing keys
        dynamic: Joi.bool().valid(false),
        dataset: Joi.string().min(2).max(128),
        format: Joi.string().valid("images"),
    },
    { //3 - non-dynamic csv
        ...schemaFraction, //reusing keys
        dynamic: Joi.bool().valid(false),
        dataset: Joi.string().min(2).max(128),
        format: Joi.string().valid("csvdata"),
        inputArray: Joi.array().items(Joi.number().integer().min(0)),
        outputArray: Joi.array().items(Joi.number().integer().min(0)),
        oneHotEncoding: Joi.array().items(Joi.number().integer().min(0)),
    }
)
const { error } = Joi.validate(model, schema);
```

Code 23: Model object validation

```
const layerValidations = {
    Conv2D: {
        filters: Joi.number().integer().min(1).max(1024),
        kernel_size: Joi.number().integer().min(1).max(64),
        padding: Joi.string().valid('valid', 'same'),
        activation: Joi.string().valid(...activations)
    },
    MaxPooling2D: {
        pool_size: Joi.number().integer().min(1).max(16),
        padding: Joi.string().valid('valid', 'same')
    },
    Flatten: {},
    Dropout: {
        rate: Joi.number().min(0).max(1)
    },
    BatchNormalization: {
        momentum: Joi.number().min(0).max(1)
    },
    Dense: {
        units: Joi.number().integer().allow(null),
        activation: Joi.string().valid(...activations)
    }
}
```

Code 24: Model layer validations

enough GPU memory to be trained on. In order to add extra levels of data consistency related to model structure, this validation steps are executed in the web page, in the master application server and on the database. This code replication is facilitated due to both the web page and the master application server being written in JavaScript.

4.7 Training Sessions

When a neural network is compiled (design the structure), weights are initialized randomly. In the process of training, the bad performing random neural network starts improving by reducing the prediction error. A loss function is selected, and the prediction errors need to be as small as possible. Improving the network is possible, because we can change its input-output behavior by adjusting weights. The objective is to compute the best weights for the model [69]. Each training script requires different configurations and different datasets. Training scripts are generating by editing code sections in cloned placeholder training scripts. Another possibility to avoid cloning code, would be to adapt training requests using environment variables on the same script. This approach would be too complex for the model object requiring the model structure to be sent as string then parsed and carefully transformed into valid Keras code. This training scripts provide multiple advanced tools such as different evaluation metrics for any specific task, saving only the best performing model, dedicated prepossessing, progress status and others. Training can take a long time until the output error stabilizes into its lowest level. Image classification, for example, require an initial prepossessing for resizing each image and for data augmentation (technique used to increase the size of data by cloning and applying noise to it). To greatly accelerate this computation, a training session performed on the GPU is significantly faster (including preprocessing) than a session on a CPU (Table 4.4). Keras library version also has a significant impact on the model training speed. The WebML architecture allows multiple slaves to provide multiple GPUs in parallel. There is a need to monitor and leverage from all the hardware that is provided to the platform, choosing what is the best task distribution for all the slaves and GPUs. A valid approach is to always choose the Slave with the highest free memory GPU. This benefits the balance between powerful GPUs and available space for training. As the highest free memory GPUs are usually the fastest, the overall speed is leveraged by the priority that candidate slaves are chosen. The best GPUs will, in average, be the most used and selected first specially if the platform has low execution amount of training sessions running in the moment.

Two candidate slaves are represented in Figure 4.12, 'slave 1' provides 2 GPUs and 'slave 2' provides 3 GPUs. The accumulated GPU memory is not taken into consideration, only the best amount of the highest free available GPU. In this case, 'slave 1' is the chosen candidate slave because 'slave 2' does not have any GPU with more than 2500 MB of available memory.

Hardware	Avg. Time (epochs: 10, batch size: 64)
i7-7700HQ (CPU)	6 min. 29s
i7-10875H (CPU)	4 min. 50s
GTX 1050 mobile (GPU)	2 min. 46s
GTX 1060 mobile (GPU)	2 min. 17s
RTX 2060 mobile (GPU)	1 min. 55s
RTX 2070 (GPU)	1 min. 47s

Table 4.4: Average training speed (5 attempts each) of the model structure represented in (Fig. 4.11)

Multiple processing entities are needed in order to accomplish a successfully training request flow and require coordination between themselves to function properly. Requests must be sent from client to the master, from the master to a valid candidate slave. This candidate slave generates the training script and monitors the training session (Fig. 4.13).

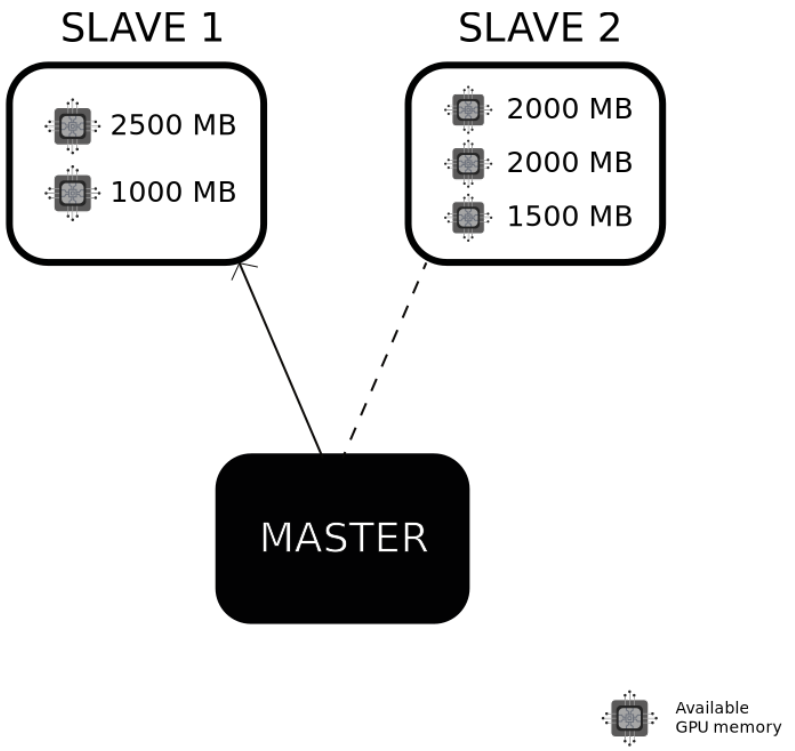

Figure 4.12: Candidate slave selection order

Training sessions can take a while to be completed. As explained later, in chapter WebML Results, the users have access to the progress status of their training sessions so they can estimate how long it will take to complete each session. This progress information is extracted from the training script thread to the slave node that streams the updated progress status to the master node. The master evaluates the candidate users that should receive this information. The correspondent target users that receive the data stream relative to the progress status are the model's owner and all the administrators connected (Fig. 4.14). It is important the efficiency of the distribution algorithm since this distribution can occur multiple times in a second. If an iteration of the distribution algorithm takes longer than the time interval between data streams, the program gets blocked since the master node is built in a single-threaded NodeJS environment. It is possible to deliver data streaming efficiently in O(n) where 'n' is the number of target communications including the administrators.

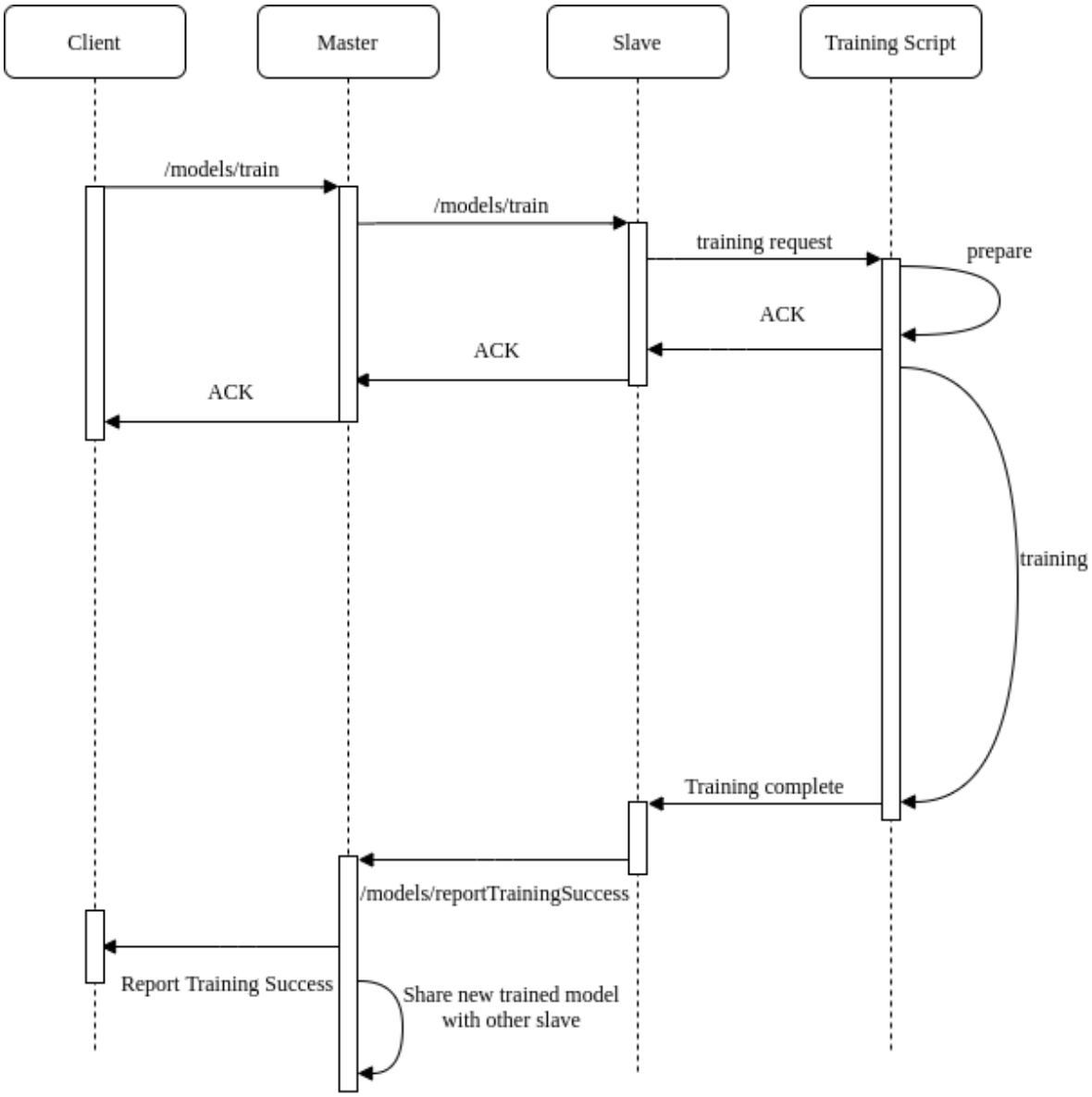

Figure 4.13: Training request sequence diagram

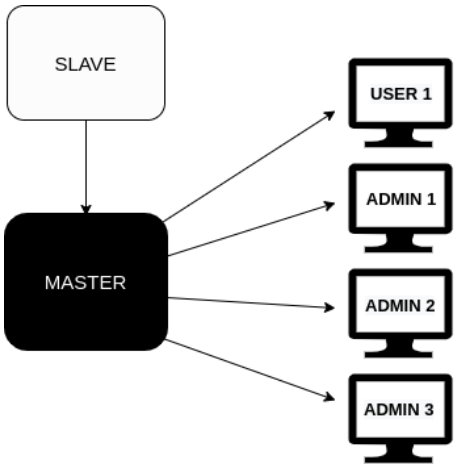

Figure 4.14: Training sessions data streaming

4.8 Prediction Requests

A neural network prediction is the output result of an input request. In WebML, predictions are only possible when a trained model is specified along with an input data. The request must go from the client to the master, from the master to a valid candidate slave, from this slave to its prediction thread where it computes the result and returns it all the way back to the client. This prediction request data and headers are different according to their formats. The sequence diagram is shown in 4.15.

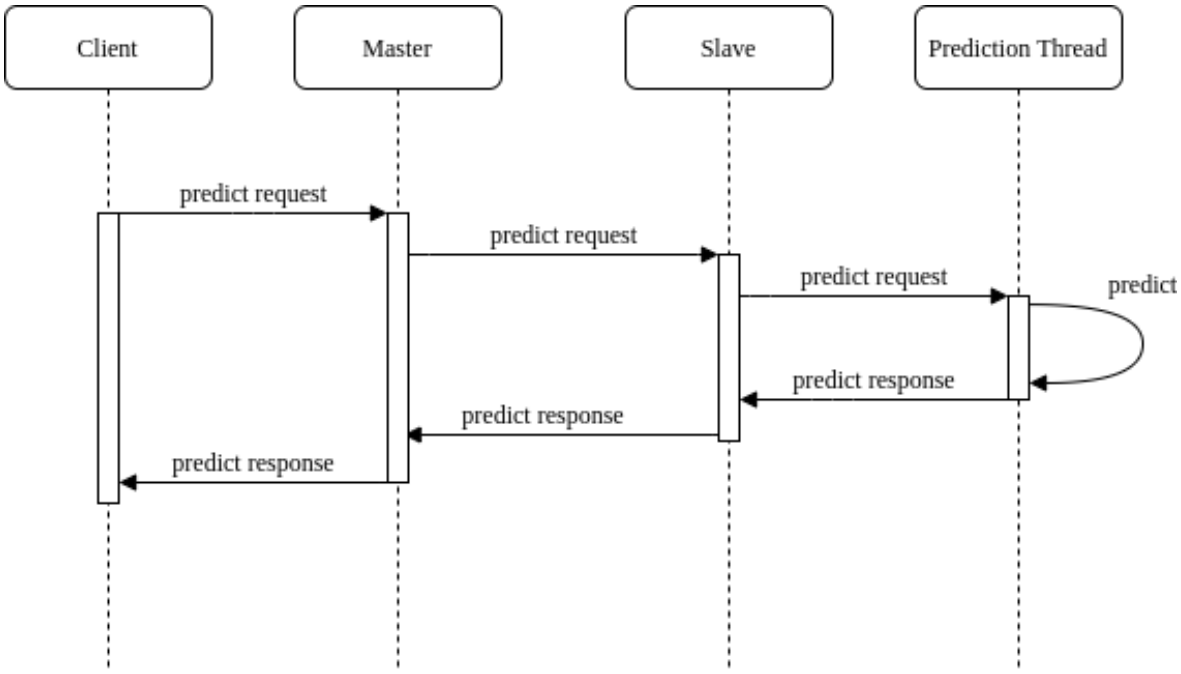

Figure 4.15: Predict request sequence diagram

Image Format Classification Request

For image classification prediction requests, an image input must be uploaded along with the prediction request. When the master node receives an image prediction request, it stores it in volatile memory to proxy it to a valid candidate slave. On the candidate slave, the uploaded image is resized to the input size of the model before exposing it to the neural network pipeline. The result of this classification request is the chosen class index, the correspondent name and 16 image examples of this class extracted from the 'training_data' folder of the model's target dataset.

CSV Format Prediction Request

For CSV format, input data of prediction requests consists on a single row (array) containing a valid input where each feature follows its correspondent valid type, the response is also a single row (array) containing the answer. Although it is not common, the neural network can have multiple output values. Each of these output values can even improve each other by redirecting the focus of the training to more subtle features, for example, adding the shape of that traffic sign as output when classifying traffic signs may help recognize it even if it is not an input feature and is not received in the prediction request!

4.9 Trained Weights

Model weights are the matrices of decimal values resulting from a successful training session. A non-regular user has limitations on the size and number of weights per model to avoid DoS and non-intentional services disturbance. Weights can be many and may require significant disk space. WebML allows all weights to be downloadable from the public API as long as the user has the permission to download it. For the CSV data, weights are stored in the same file as the whole pipeline structure shown in (Code. 19). In order to use a trained model for prediction or further training, it is required for a valid candidate slave node to be selected. Only slave nodes with the requested trained model and its dependency dataset stored on disk are considered valid nodes. This can be very restrictive as the whole training session is only executed on a single slave. The master node stores every model weights' identifiers that each connected slave has in Set data structure (faster queries). This implies that every time a slave node connects to the master, it notifies the master of the list of weights stored in the disk. In order to avoid training rejections due to this restrictions, the model weights are broadcast to the other registered slaves after a training session. Although this approach duplicates data, it is important to create this data redundancy for security and service availability.

4.9.1 Model Weight Sharing

The process of model weight sharing starts right after the last execution from a successful training session of the master server represented previously in (Fig. 4.13) - 'Share new trained model with other slave'. Model weight sharing is quite a simple procedure, all online slaves that contain the dataset where the model was trained on will be notified to download the weights from the source which is the chosen slave for the training task (Fig. 4.16).

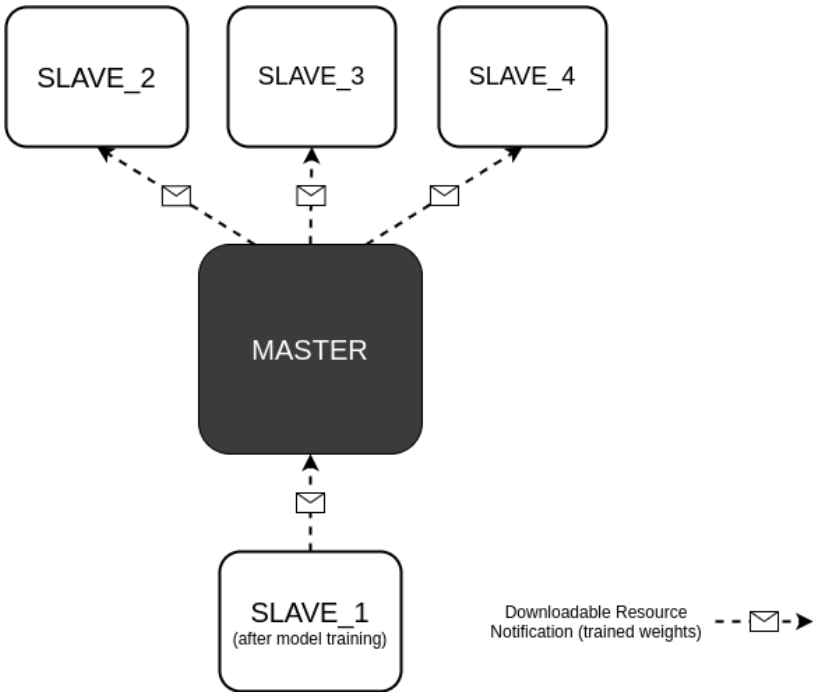

Figure 4.16: Training report notification

Since the slave architecture is built asynchronously using NodeJS framework, downloading model

weights will not stop or significantly impact any parallel training sections or model predictions. Once a slave completes the download process, the master node updates the set of its correspondent weights adding one entry.

4.9.2 Model Weights Removal

Users are imposed a maximum number of trained models related to a structure. After a model is trained, the user can remove it to allow for the saving of more trained weights. Model weight removal works similar to model weight sharing. The client requests the master node to remove the trained model, the master removes it from the database and requests all the candidate slaves (slaves containing the model stored) to remove the binary data from the disk. This process is represented in Figure 4.17.

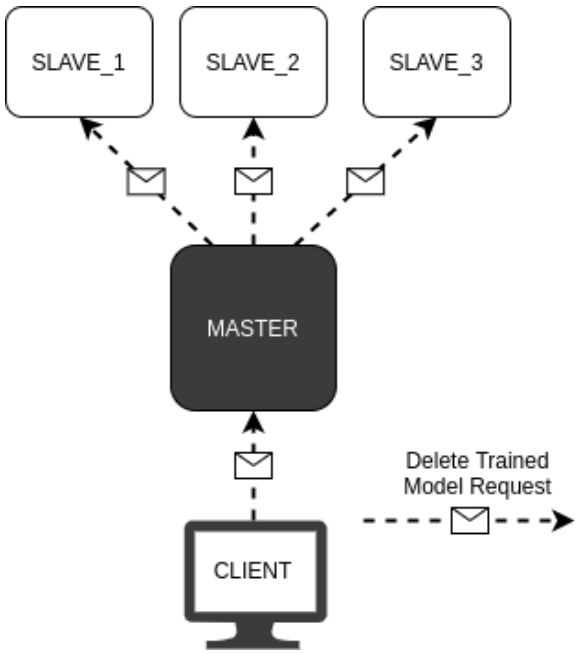

Figure 4.17: Delete trained model request

4.10 Download Experiment

WebML does not limit its users to train and test models remotely using the UI. The 'download experiment' tool aims to deliver the code base along with the trained models of a target model structure to the user in order to offer programming flexibility. Only model structures that are set to **non-dynamic** are valid candidates for this tool. Although this 'download experiment' mechanism is not required for neural network model training and evaluation purposes, it is an additional useful functionally for many users. This feature delivers a zipped file to the user's computer, containing the following content (Fig. 4.18):

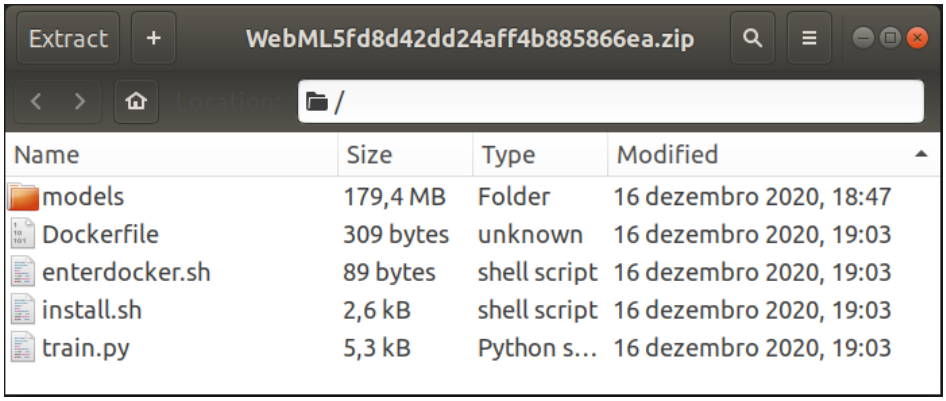

Figure 4.18: Download experiment zip file

- **models/** - folder containing all the trained weights of this model structure, it can be empty;
- **Dockerfile** - docker file containing all the dependencies needed to run 'train.py' script ;
- **activatevenv.sh** - single command bash script to activate docker environment;
- **install.sh** - bash script to install drivers, library dependencies and required docker image;
- **train.py** - Python training script of the model.

In order to benefit from the supplied files, the user must be using Linux operating system and also unzip the zipped file that has been downloaded from the platform. The most relevant file is the Python script 'train.py'. This script allows the user to train the correspondent model structure in any compatible datasets. This model's structure is directly incorporated on the code of this script. This file is similar to the template files provided for the internal training session described above in section 4.7. The difference between these two scripts lies mostly on the flexibility of training procedures regarding the configuration through arguments. Since the 'download experiment' feature aims to deliver the user multiple easy options to train models, it is possible to change training sessions' configurations such as epochs and batch size by simply pass the different settings as arguments (Code 25).

```
Python3 train.py --dataset ./datasets/datasetA --epochs 5 --modelname newmodel.h5
```

Code 25: Command line for training procedure

The main issue of this feature is dealing with recurrent dependency incompatibility issues regarding the required Python libraries, mostly TensorFlow and Keras. Docker virtual environments' is the chosen solution for dependency issues such as these. Although docker images are sufficient to clone almost perfect environments for most of the cases, it is not the case for environments that require GPU hardware. External driver dependencies are also needed in order to ensure GPU access.

Installing these dependencies manually can be challenging as it often results in package errors and requires extra steps to work properly. This is where 'install.sh' file proves itself useful. This file ensures and verifies that all requirements are met one by one. The sequence of commands relative to this installation script is the following:

1. install recommended nvidia-drivers;
2. install basic environment dependencies;
3. install docker;
4. resolve docker user permissions;

5. install nvidia-docker2;
6. resolve nvidia access to docker containers;
7. build an image based on the Dockerfile instructions;
8. ask the user to reboot the computer if needed.

A full installation resulting from a complete execution of this script will set up a compatible environment that will be mounted on a docker image once it is finished. This installation can take a significant amount of time (>20 min.) until it is completed. Some target computers already have some of these dependencies installed when the script is launched. In order to avoid re-installing these dependencies, the script auto-evaluates the current state of the host system and, based on this state, will only install the dependencies that are missing. After the environment has been built for a docker image (installation completed), the user must have access to its local disk in case the target dataset chosen for the training is not in the same folder. By default, docker containers function as a private virtual environment and will not share or provide access to the local disk's files unless it is explicitly authorized using docker volumes. Docker volumes are a mechanism for persisting data generated by and used by docker containers [70], working as a link between the hosting operating system and the virtual environment. Considering a docker image with 'gpudocker' as tag name, the following command must be executed in order to enter the virtual environment with access to the local file system (Code 26):

```
docker run -v /home:/home -it --runtime=nvidia --rm gpudocker bash -c "cd $(pwd) && bash"
```

Code 26: Command line activating docker environment

Being forced to write this command line whenever the user wants to activate the environment can be repetitive and counter-productive. Fortunately, it is possible to encapsulate this relatively long command line into a single bash file. In the case of WebML, this bash file is 'activatevenv.sh' and it is also provided in the downloaded zipped file. The service implementation regarding the process of gathering these files for the user requires multiple sequential steps starting from the user's request and ending in the delivery of the assembled zip file to the user. The application server responsible for this process is the master server. In order to accelerate this process, the master server stores locally a template folder for each model format (images/csvdata) already containing the 5 documents/files described above since the only differences will lie on the 'train.py' script and the models' folder. The models' folder is stored empty and the training script functions as a placeholder code (partial code made for portions of code to be injected). The sequence of actions executed by a 'download experiment' request can be analysed in the diagram of Fig. 4.19.

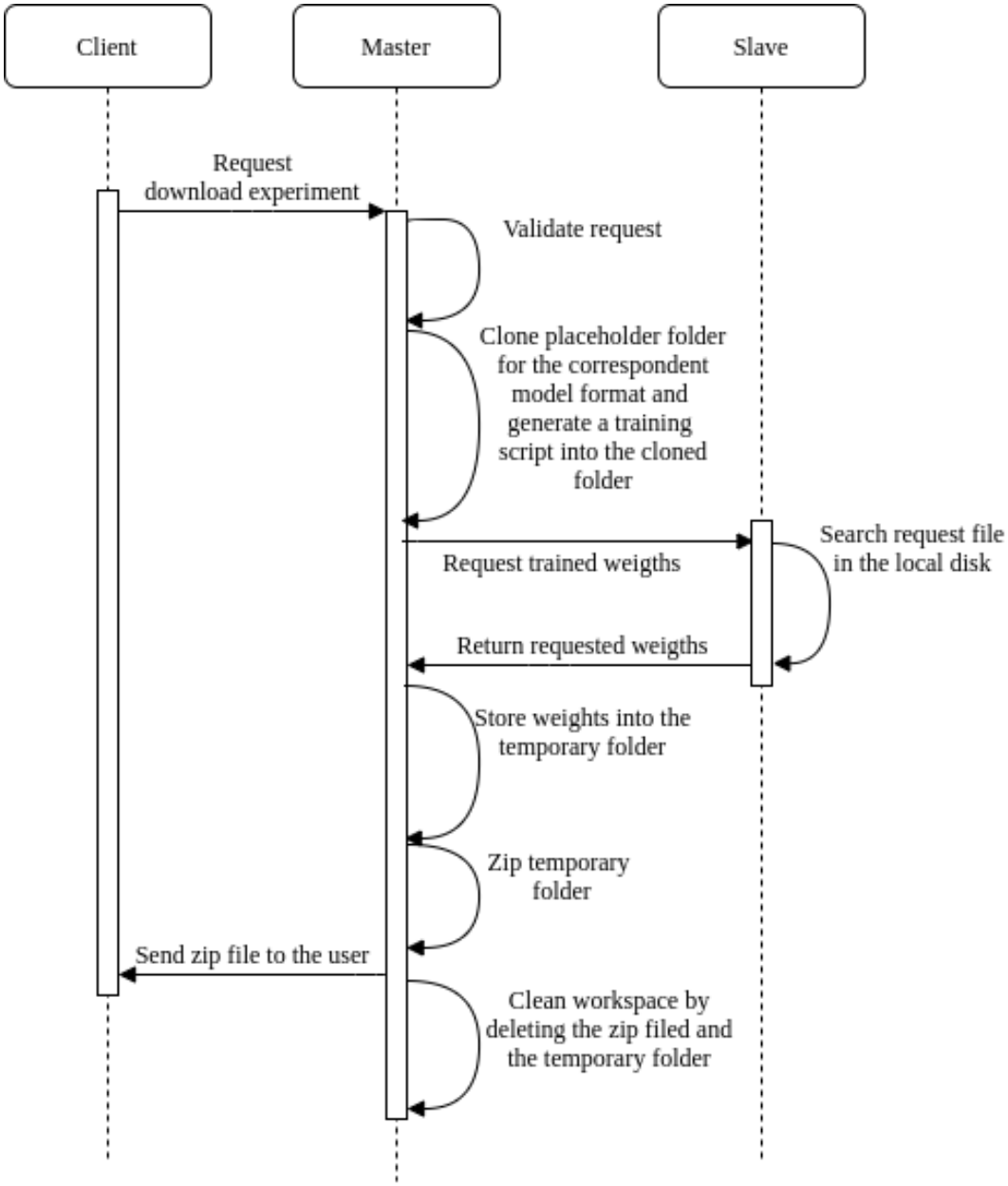

Figure 4.19: Sequence diagram of download experiment procedure

1. Request validation - The user must be owner of the request model structure and the model structure must be **non-dynamic**;
2. Clone template folder - The template folder of the correspondent model format is is copied to a temporary folder. This temporary folder is meant to assemble the necessary files to deliver to the user;
3. Generate training script - The training script 'train.py' is generated by injecting portions of code in the placeholder file such as the model's structure and performance metrics. This file is placed in the temporary folder;
4. Download trained weights - the master node will download all the correspondent trained weights from the slaves, one by one, into the models' folder inside the temporary folder;
5. Zipping the temporary folder - the master node zips the temporary folder;

6. Delivering to the user - The zipped file is delivered to the user;
7. Clean the workspace - The zipped file and temporary folder are deleted.

Regarding this list of steps, step 4 and 5 can take significantly more time to complete than the rest since weights can consume significant amount of disk space, delaying the downloading and zipping process. There is a possibility for the system to crash while on this procedure. To cope with this possibility, WebML also offers a robust automatic recovery implementation similarly to a database transaction where the tasks are either committed or rolled back, recovering the workspace into its expected state, in case of failure.

4.11 Deployment

It is important to write installation and deployment scripts in order to ease slave extensions and refactor the deployment process as fast as possible in case of failure or when version updates are required. Deploying the WebML platform securely requires a proxy server like NGINX or Apache, serving its proxy services through https protocol, preferentially using port 443. This proxy services should only redirect requests from the client when the target URL is for the master server or for the graphical interface web server (Fig. 4.20). A automatic installation script in bash was also developed to set up WebML's services locally and remotely.

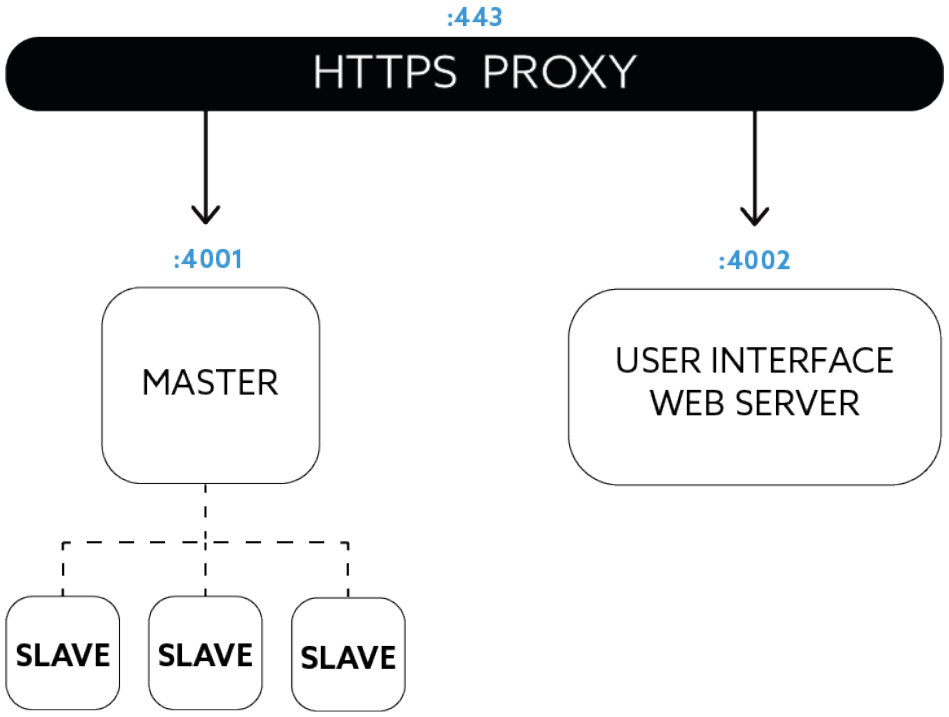

Figure 4.20: Deployment proxy services

Slaves are internally connected to the master server after they exchange authentication signatures between them. These connections are refused when the master application server version is not compatible with an outdated version of the slave application server. Slave connections are also refused if the master version is not compatible with the slave version.

4.11.1 Master

The master node requires MongoDB service to be running and accessible to the application server. In order to maximize performance these two services should be on the same Linux machine. This approach will reduce security if no automatic backups are available. Aside from the NodeJS environment installation, it is required to install dependencies that are stored in package file 'package.json'. Executing a simple command in the directory of this file will install all dependencies. Although the master deployment can be fully performed on a docker instance, the complexity of the installation is similar to launching with a production tool like 'pm2'. The master node requires 3 environment variables to be able to be launched.

The environment variable list is as follows/is presented bellow:

- PrivK - Master node private key, this key is used to sign JWTs;
- mongo - URL to the mongoDB database instance;
- port - Listen port;
- version - Master node application version;
- compatibility-version - Slave node minimum compatible version.

4.11.2 Slave

The slave deployment process is quite complex when compared to the other services. Slaves require special docker instances (Nvidia) to guarantee high compatibility for a high variety of Linux machines. The slave has network requirements in order to be deployed. Slaves need to be accessible from the master server and also be able to access the master server, able to send API requests and web-socket messages. WebML's directory contains a full installation script written in bash and a completed Dockerfile. Slaves have a longer list of environment variables.

Environment variable list:

- PrivK - Slave node private key, this key is used to sign the master's JWT;
- port - Listen port;
- mainserversocket - WebSocket URL of the master server;
- mainserverapi - URL of the master server;
- token - valid slave authentication JWT;
- public_api - This is a secure URL that is sent to the master node, using this URL the master knows how to contact this slave restful API;
- version - Slave node application version;
- compatibility-version - Master node minimum compatible version.

4.11.3 Web server

The web server is meant to be running on a NodeJS environment on production settings. The web server requires a restrict installation procedure because of some client-side libraries that have conflicts, it is recommended to use the 'yarn' package manager for this matter. 'yarn' is an efficient package manager for NodeJS environment. The list of environment variables is as follows:

- exAPI - URL of the master server accessed from the browser;
- sServer - WebSocket URL of the master server accessed from the browser;
- localServer - URL of the master server accessed from this web server, should be localhost if hosted on the same machine and be exactly like 'exAPI' otherwise.

The ‘localServer’ environment variable is very important for performance if the Web server and master server are being executed on the same machine. It reduces the loading time of the page to about half the time. It leverages from the previously explained SSR on Figure 3.6. When the user refreshes the page, being already logged in (authentication token delivered via cookies), the Web server fetches the data from the localhost instead of using the public IP internet routing.

5

WebML Results

This chapter presents the results and explains all the workflow of the graphical interface, both for the regular user and the administrators. This chapter can be followed as an advanced tutorial for all the use cases specified in Chapter 3.

WebML involved a significant planning time for the user interaction and information display when compared to the time spent in the overall development. In order to start the development of the UI, a key task was identifying all the use cases and the variety of potential users. Many improvements over the course of the development of the platform have been developed regarding punctual feedback from small meetings and presentations of WebML. Multiple feedback of different groups of users was gathered over time and notes were taken.

WebML's UI was projected to deliver a robust and intuitive workflow to the usage of neural networks. It was also intended to supply enough freedom to navigate between the pages and maintain control of the different processes. The WebML's web page follows a simple color layout, having all the indispensable information available at every step of the user's journey. The information displayed on the web page is adapted to each context with the intent to hide excessive information and guide the user to the next steps. Lastly, the user must feel the value that the platform can provide and also the time spared on coding training/testing deep learning scripts.

Admins have extra functionalities and monitorization extensions that non-admin users have no access to, these different groups of users are supplied with different views related to their different privileges.

The interface was conceived for those who want to develop their own neural network easily and quickly. It is beneficial for a wide range of users, even those without background knowledge in machine learning.

5.1 Landing Page

The landing page is usually a single web page that appears in response to a requested raw URL, especially for visitors. Landing pages are often linked to social media and marketing content. The WebML's landing page does not follow this principle since it is not a 'free to use' platform and registrations are required to be generated by an administrator. The overall display of the landing page

serves as a login page (Fig. 5.1). However, it is also possible to register by filling the registration form available on the landing page using a generated registration URL by an administrator (Fig. 5.2).

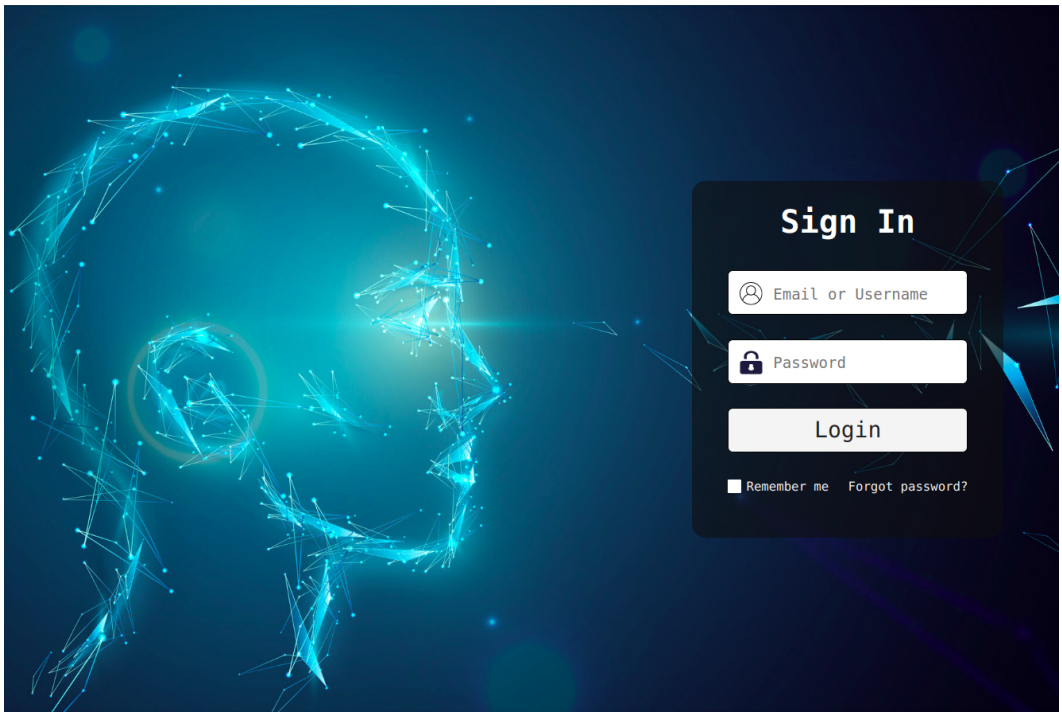

Figure 5.1: Landing page

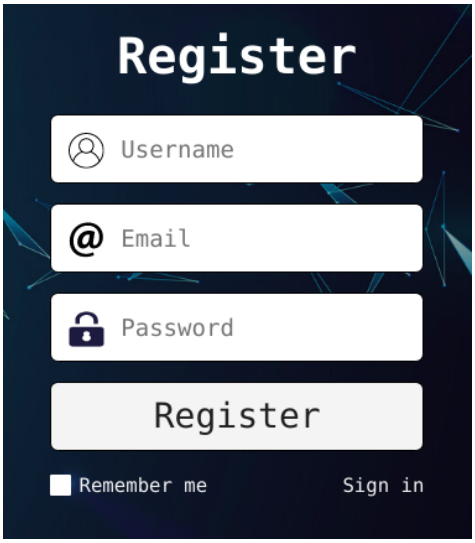

Figure 5.2: Landing page - Register

5.2 Graphical User Interface Layout

Although it is not the main focus of WebML, UI design was also important to deliver greater satisfaction and add intuitive guidance to the users. WebML focused on a clean interface, without irrelevant information and with the use of many icons in order to reduce the need for reading and to improve the understanding of the processes. Having a web page without standard and clean icons will

harm the user's understanding and will significantly occupy a higher amount of the screen's available space because of the larger requirement for the text to make up for the lack of visual information delivered by the standard, well-known icons. From all the available icon formats, Scalable Vector Graphics (SVG) is the best choice to integrate on a web page as they scale adaptively to multiple sizes. A significant amount of WebML icons were customized and designed directly using tools like 'draw.io' and 'Adobe Illustrator'. The main theme colors of WebML UI are black and white to increase contrast and enable the possibility to invert colors according to the user's choice (dark theme). The main colors and font sizes are globally set in a file displayed in Figure 5.3. This file is written in SCSS (CSS extension). This file's constants configurations contain hexadecimal values for colors that follow RGB format and sizes that use root em (rem) units instead of pixel (px) units. For web pages, rem units are the best units to build a perfect scalable layout because of their scalable nature. Editing this file will change the layout configuration in all the pages.

```
$xxsmall-font: 1.1rem;
$xsmall-font: 1.35rem;
$small-font: 1.6rem;
$basic-font: 1.7rem;
$big-font: 2.5rem;
$bigger-font: 2.8rem;

$header-size: 7.5rem;

$background-color: #f9f9f9;
$background-invert: #011;

$gcolor1: #0d0d0d;
$gcolor2: #262626;
$gcolor3: #595959;
$gcolor4: #a6a6a6;
$gcolor5: #eaeaea;
$gcolor6: #f4f4f4;
$gcolorcolor1: #0e1ac0;
$gcolorcolor2: #1853d1;
$gcolorcolor3: #66b5ff;
$gcolorcolor4: #57a0ff;
$font-color: black;
```

Figure 5.3: Global design constants

WebML is based on a non-standard UI approach, where the whole platform is the menu and where you can zoom in to enter a specific page or zoom out to go back to the menu overview. This means that when the user is visualizing the menu (Fig. 5.4) they are actually looking at the platform as a whole and can navigate to every corner of it. Aside from the login page, WebML's **UI!** (**UI!**) contains a total of 9 different pages, all of them accessible from the menu. The pages are displayed in a 3x3 configuration and it is possible to see them all at once in zoom out mode (Fig. 5.4), as it all works as

a complex and yet intuitive circuit/workflow to complete training and testing tasks. To make this document easier to read, this zoomed-out page is named **FloatingBoard**. To access individual pages, a simple click on the target page is all it takes. By selecting one of the 9 pages from the menu, the platform will be making a transition to that page directly by zooming in towards the page.

NuxtJS framework allows data to be saved in a store called using a Vue library called Vuex. A 'store' is basically a container that holds an application **state**. In the context of this interface, despite the current page the user is on, there is always a global selected model, a selected dataset, a format and a training model. These state properties are set to null when not selected. Administrators have a much higher complexity on their state since they receive extra information from the back-end to be able to monitor the whole application and hardware.

Figure 5.4: FloatingBoard - zoom out user interface

The list of the user pages are the following:

- WelcomePage;
- TutorialPage;
- AboutPage;
- AccountPage;
- OverviewPage;
- DatasetPage;
- ModelPage;
- TrainPage;
- TestPage.

For all these pages inherent to the user interface, models are presented in 3 different stages. When models are created, start as a simple format placeholder where they still can't be trained. In order

to be trained, the user needs to 'shape' the model by specifying the target dataset and some other parameters in case the model belongs to the CSV data format. The different states are the following:

- **dynamic** - In this state, the model's output size is not defined and the model currently has no target dataset. In order to transform a dynamic state into an editable state, the model needs to be **shaped** for a dataset;
- **editable** - In this state, the user can edit the layer structure, the optimizer and the loss function. A model is considered to be in this state after it has been already shaped to a dataset;
- **static** - In this state, the model's structure has been configured. This state is set after the first training request of the model. The model can only return to the editable state by removing all trained model weights.

A state diagram regarding WebML's model states is represented in Figure 5.5.

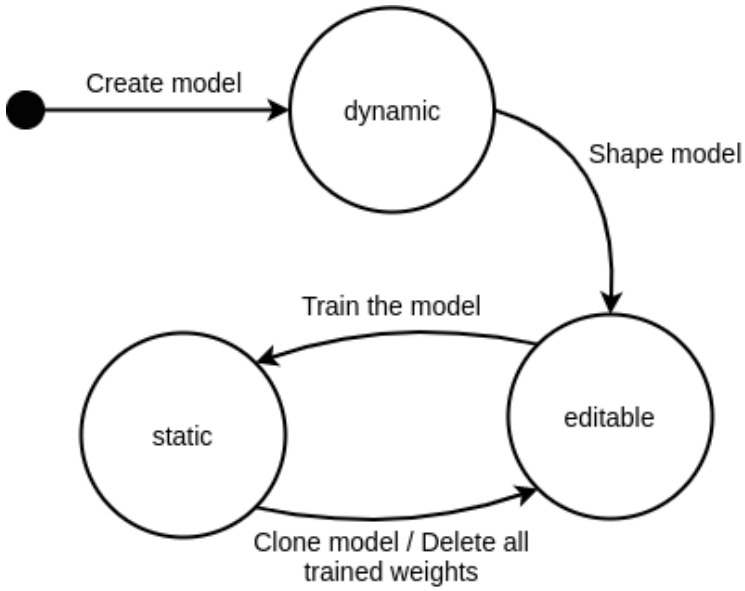

Figure 5.5: Model state diagram

As mentioned in previous chapters (3 and 4), models have 2 different formats. Datasets also share the same pool of formats:

- **CSV** (csvdata format) - tabular data, multiple types of learning are possible;
- **image** (images format) - format dedicated to image classification.

These two formats are displayed and managed differently for the user in the UI.

5.3 Navigation

Humans are excellent at geographical/spatial orientation. The WebML's navigation aims to satisfy the user by providing them an easy way to go back to a previously visited page just like a person guides itself on a road using a GPS. Moving between pages as well as zooming in and out are performed with smooth transitions. These visual transitions are important because otherwise, the user would not have a spatial idea of the current page in relation to the others. The web page navigation is built in an intuitive way and the relative positions of pages were conceived to be as correlated as possible. Aside

from re-directions of the pages themselves, different control keys can also be used to allow the user to navigate throughout the pages:

- ArrowRight - Move to the page on the right;
- ArrowLeft - Move to the page on the left;
- ArrowDown - Page down;
- ArrowUp - Page up;
- Escape or SpaceBar - Zoom-in/Zoom-out.

5.4 WelcomePage

The welcome page refers to the page shown after a successful login. This is the most suitable page for the new user to start its journey in WebML. On this page, the control keys of the global navigation are revealed to the user along with the option to change the color theme. Fig. 5.6 and Fig. 5.7 are the displayed representations of the Welcome page in light mode and dark mode respectively.

This page can redirect the user to other pages by clicking:

- **tutorial** - route to TutorialPage, the page designated to show a tutorial;
- **start** - route to OverviewPage, the page designated to start the workflow;
- **admin/account** - route to AccountPage, the page designated to manage the account.

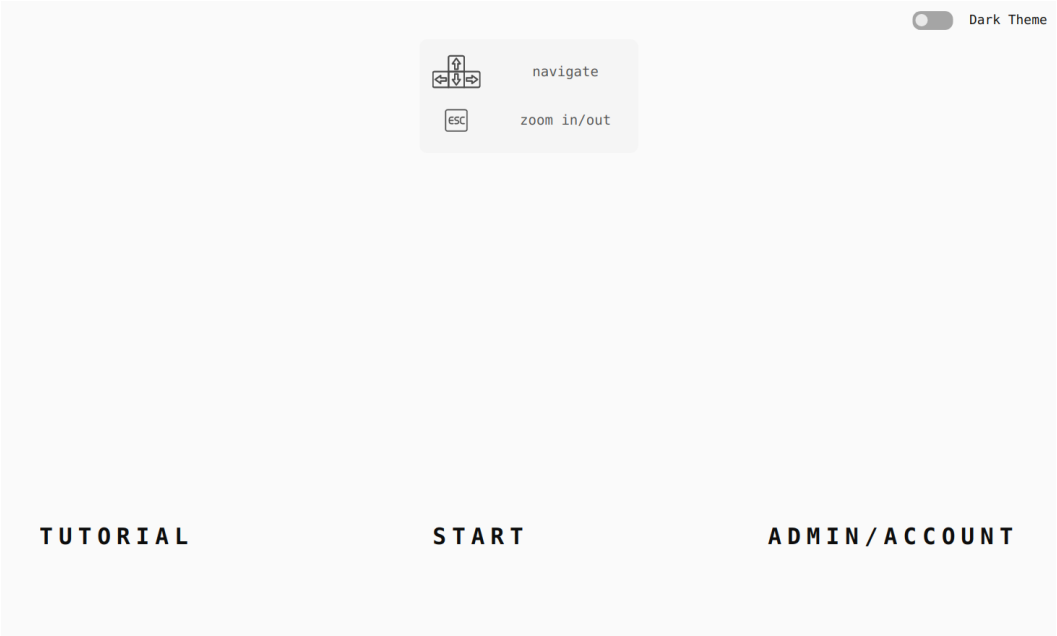

Figure 5.6: Welcome page

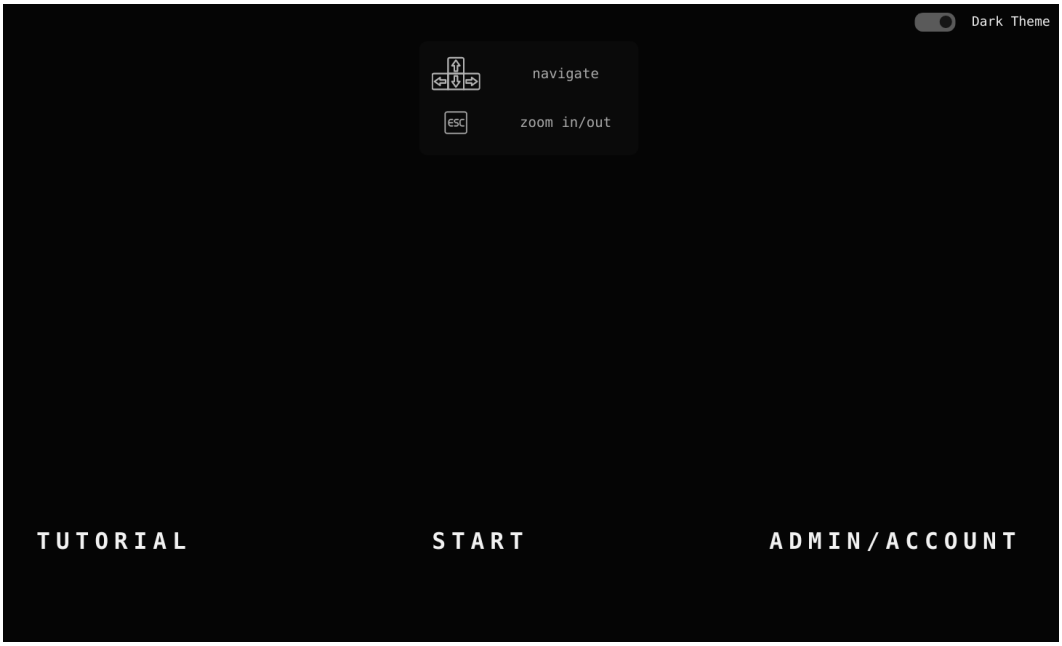

Figure 5.7: Welcome page - dark theme

5.5 TutorialPage

The TutorialPage is placed on the top right corner of the FloatingBoard. This page offers a set of videos in order to help the new user become familiar with WebML's interaction procedures to complete deep learning tasks (Fig. 5.8).

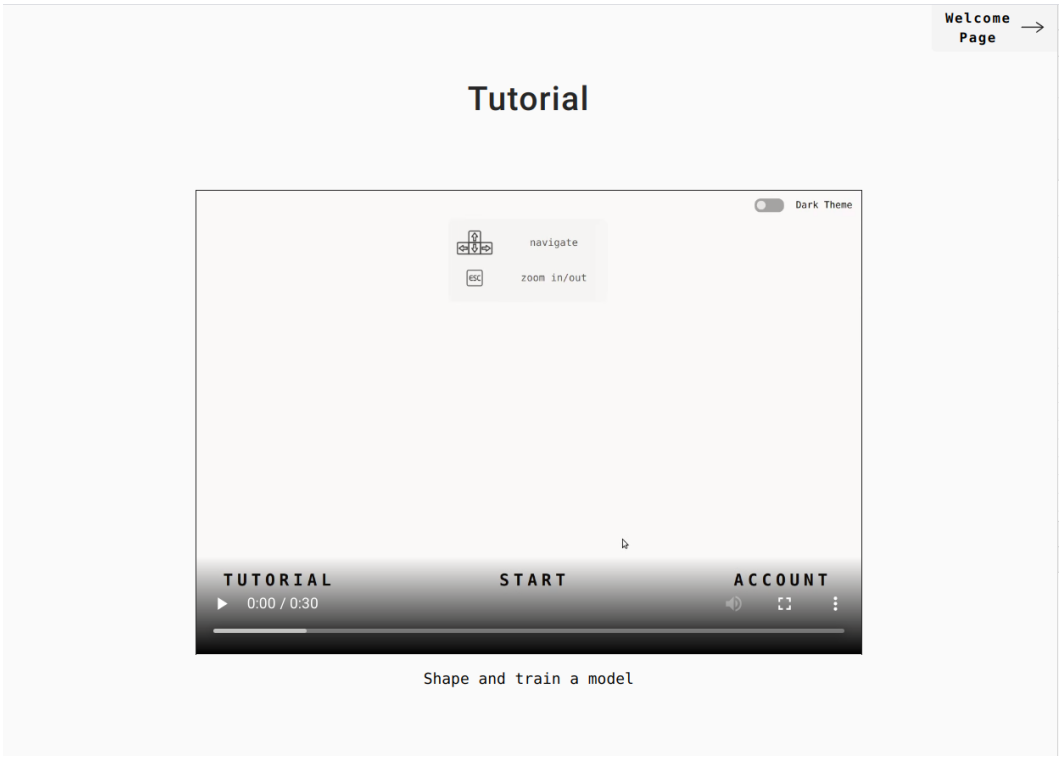

Figure 5.8: Tutorial Page

5.6 AboutPage

The About Page is positioned on the left side of the FloatingBoard. This page's only objective is to deliver general information about the WebML platform (Fig. 5.9).

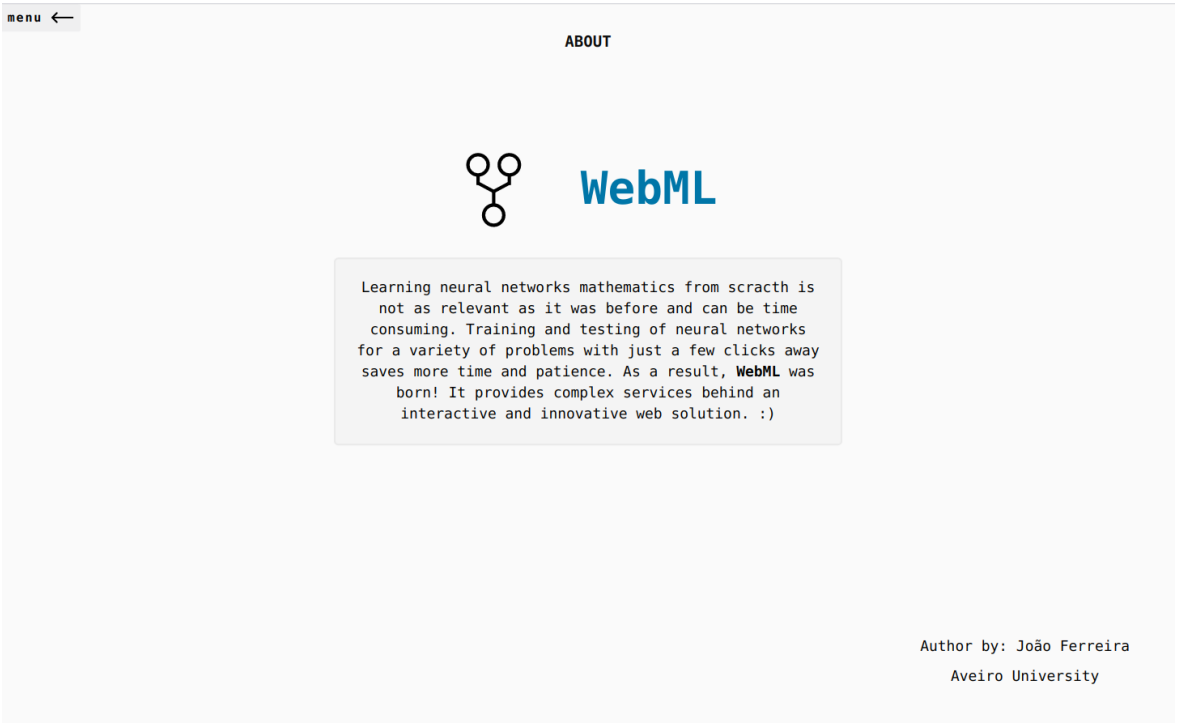

Figure 5.9: About page

5.7 AccountPage

This page is positioned on the top left side of the FloatingBoard. The AccountPage does not interfere with the workflow of models and datasets. For the regular user, it is just an informative page regarding the account information (Fig. 5.10). This page also provides a sidebar offering multiple options to navigate to other pages or even to logout.

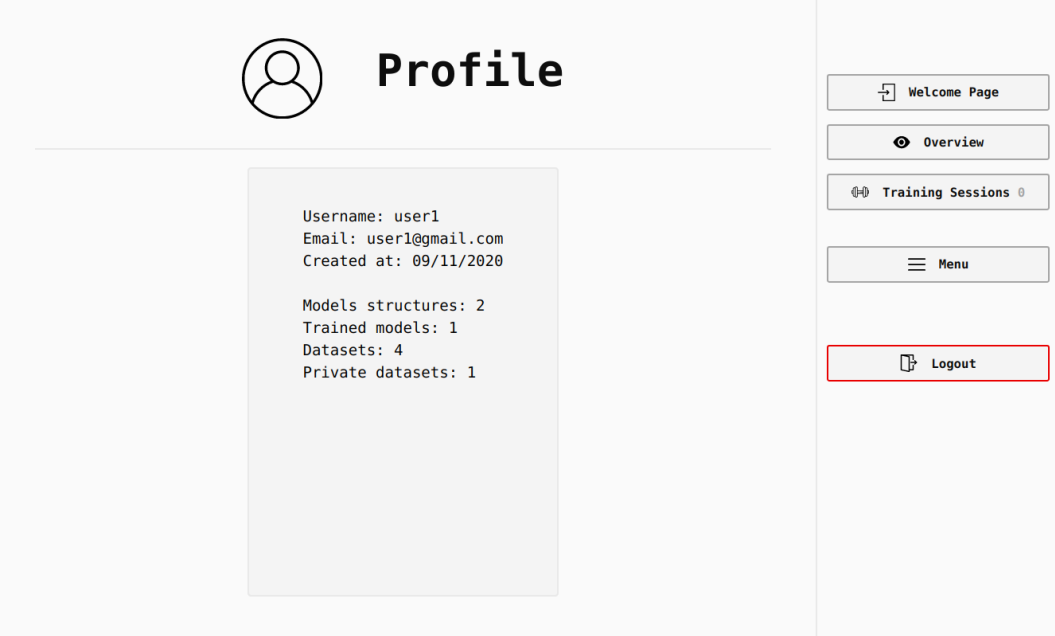

Figure 5.10: Account page - profile

On the other end, the administrator has a wide range of tools in this page. Upon entering the page, it is presented with the same profile page as the regular user. The main difference is the switch component on the top of the sidebar that allows the administrator to change to admin view (Fig. 5.11).

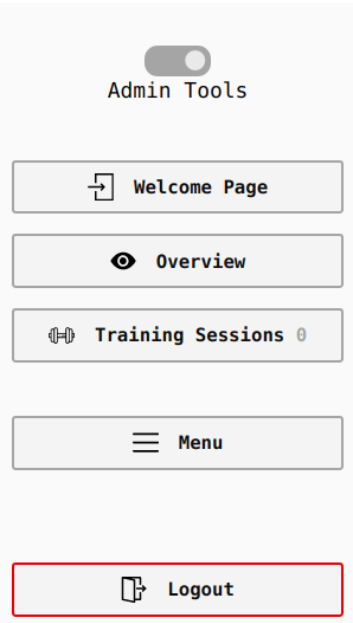

Figure 5.11: Account page - admin sidebar

The admin section is called Administration. This section provides monitorization and management tools for both the slave nodes and the users in the platform. Most of the administrator unique use cases are all covered in Administration. This section is also divided into 2 similar UI subsections, the slave subsection and the user subsection.

In the WebML's UI, slaves are called nodes and this slave subsection is an example of this terminology (Fig. 5.12). The right side of this view (excluding the sidebar) is reserved to register new nodes to the node pool. The admin user is requested to input a new hostname every time a node is registered. This hostname will then appear in the place where the title 'Terminal1_Aveiro' is presented in the example figure.

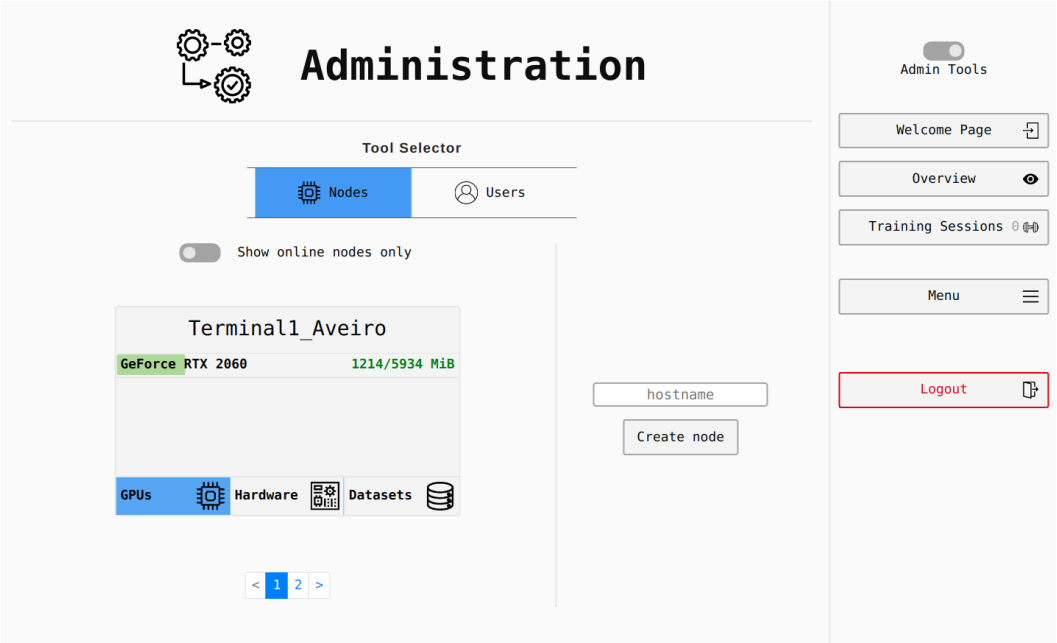

Figure 5.12: Account page - admin tools for slaves/nodes

Registering a slave (node), generates a unique validation key and sets its instance to the non-validated state. The slaves or users in non-valid state have no username, email, or password. These fields are allowed to be inputted using the registration link that is generated in the UI view (Fig. 5.13).

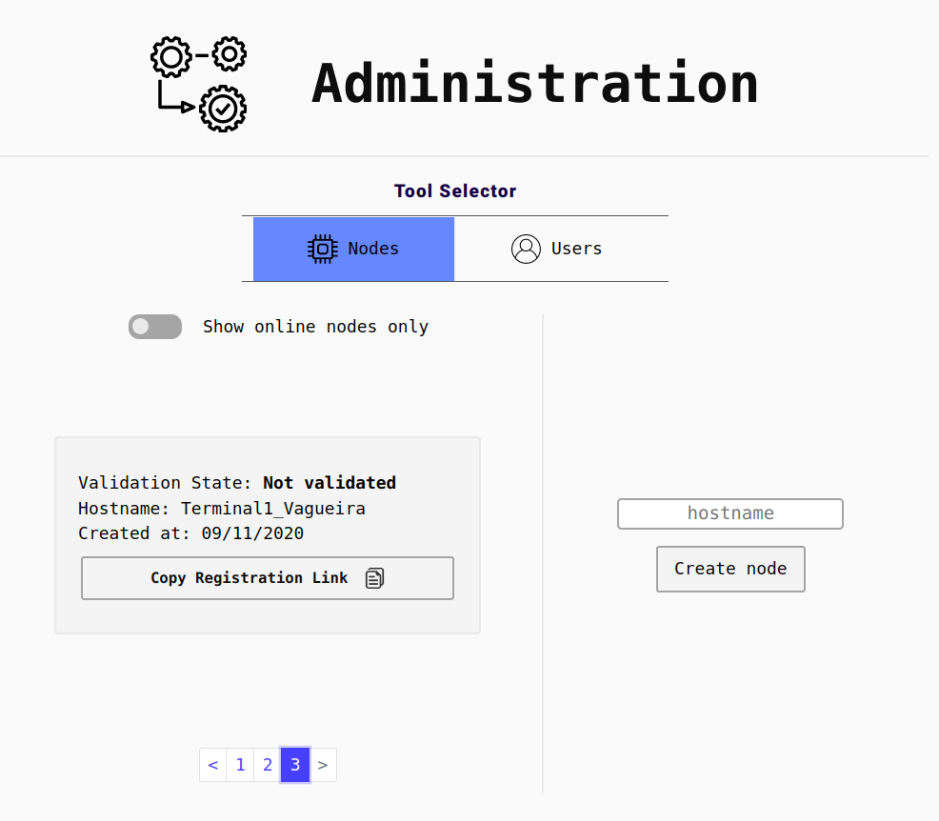

Figure 5.13: Account page - admin tools for non validated slaves

The registration link redirects the user to the login page. This URL generated by the administrator is the root of the WebML domain name with a query string attached in the URL containing the validation key that is required to properly validate the node/user.

- example: https://ua.webmlurl.pt?validationKey=registrationkeyhere

In this example, 'ua.webmlurl.pt' is the domain name redirected to the Web server API that is running on port 443 (HTTPS).

After register data is sent and validated, the state of the registered node is now valid and can now supply its hardware in the node pool (Fig. 5.14).

Figure 5.14: Account page - administrator tools for validated slave/node

When a validated node is online and available, the administrator can inspect its hardware and the list of its stored datasets in real-time. These visualization tools are available on the left bottom corner of the page (Fig. 5.12) and provide 3 different information tabs:

- **GPUs** - It is presented a list of the available GPUs. For each GPU item, its model name is displayed on the left and its correspondent memory information is displayed on the right;

- **Hardware** - It is presented information about 2 hardware components, the secondary memory storage and the CPU;
- **Datasets** - It is presented the list of stored datasets of the referent node.

The hardware information displayed in the second tab for each node (Fig. 5.15) presents attributes such as the CPU model name and its correspondent thread number and also information about the secondary memory (disk) regarding its type along with the space status of the disk. Administration tools for the users follow a very similar display as the nodes/slaves (Fig. 5.16). The registration procedure is also similar, the difference lies on the fact that the user creation does not require a hostname set by the administrator.

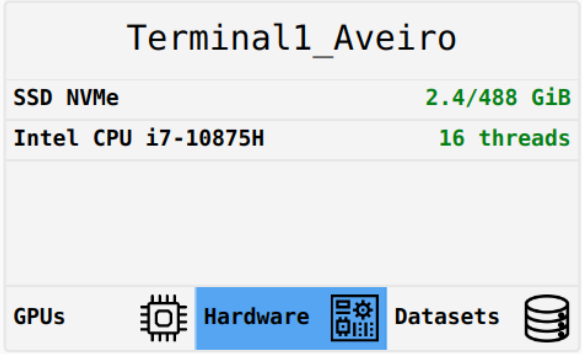

Figure 5.15: Account page - slave hardware info

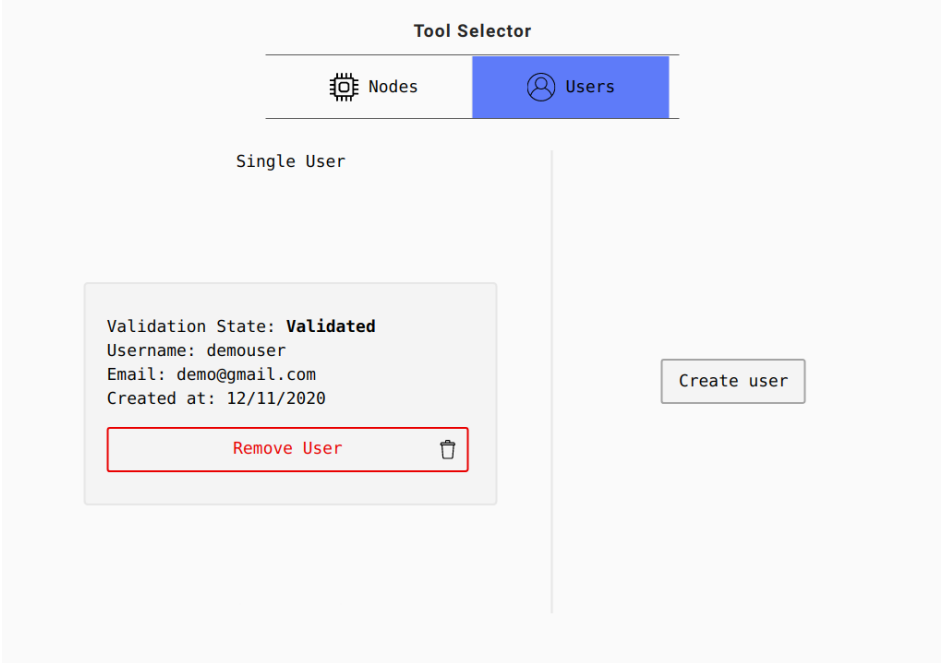

Figure 5.16: Account page - admin tools for users

5.8 OverviewPage

This is the main page used when testing neural networks and different dataset configurations. It is the second page a new user should enter and it is one of the WebML's most navigated pages.

The overview page is placed in the middle of the FloatingBoard. The centered position of this page demonstrates its high importance and its relation to most of the other pages.

The OverviewPage is responsible for guiding the user to successfully train a model on a dataset. This contains 2 modes/tabs: 'overview' and 'shaping', 'overview' being the default tab. All models are set to the **dynamic** state after their creation in the ModelPage. In order to change from dynamic state to editable state, models need to be shaped for a dataset. Fig. 5.17 shows the default display of a selected dynamic model and a selected dataset. All datasets are compatible choices with dynamic models that have the same format. In this case, a dynamic model of the CSV data format is selected alongside a valid dataset. The gray section displayed in the middle of the OverviewPage on the default tab is called **TrainingFlow** and leads the user to click on the 'Shape Model' button. Clicking on this button redirects the user to the 'shaping' tab on the same page.

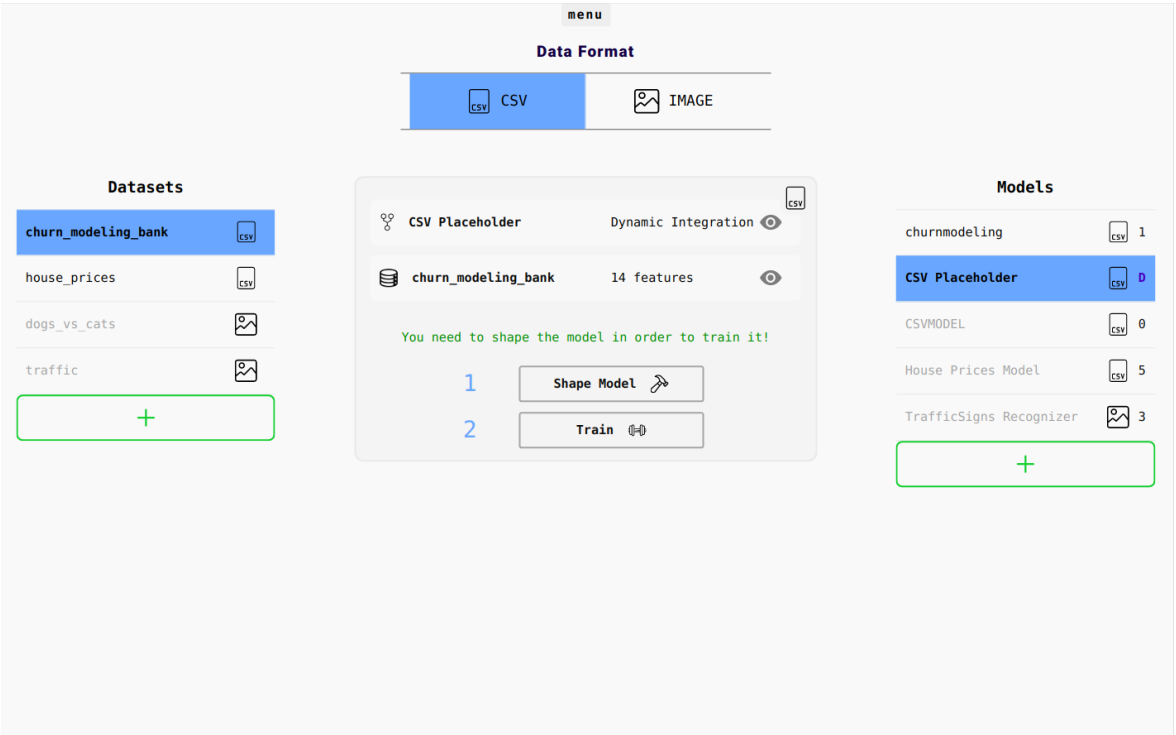

Figure 5.17: Overview page - placeholder model

The shaping tab information display and actions differ substantially with the different formats. The most simple shaping procedure is the one that applies to the image format (Fig. 5.18). The left side of the page tab presents a dataset preview by revealing an example image of each class. On the right side, it is possible to observe the model that will result from this shaping. Since this format is meant for classification, the user is left with 3 options in the action section (middle web component):

- Shape New Model - Create a new model by cloning the selected model with a new name retrieved from the user input. This new model will be set to the editable state and will contain the same structure as the cloned model except for the output size that will be resized to the number of classes of the dataset;
- Shape And Override - This procedure is similar to the 'Shape New Model' procedure, but it overrides the current model. The name of the model is not changed and the input field will be

ignored;

- Cancel - Returns to the 'overview' tab.

The cancel button follows good reverting practice in the UI. After each action of the user, it should be easy and intuitive to return to a previous state by reverting the user's last action.

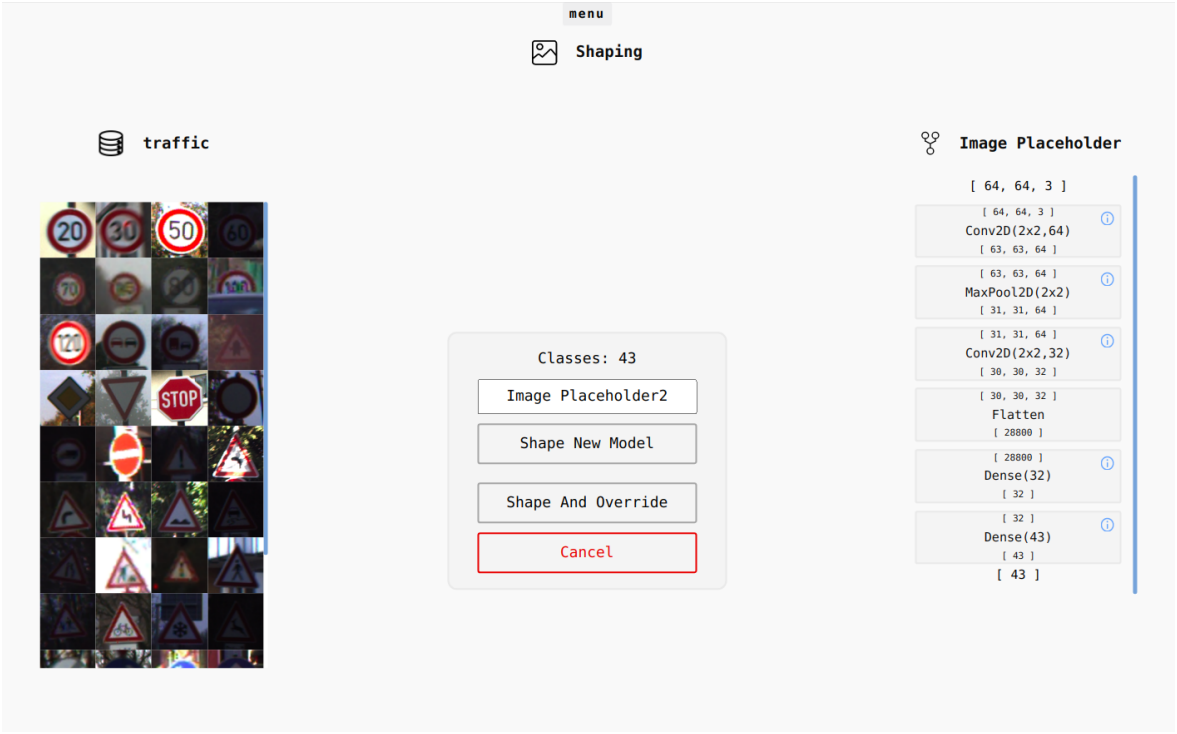

Figure 5.18: Overview page - shaping an image model

On the other end, CSV shaping procedure is quite more complex. A valid WebML's CSV file contains multiple features (columns). Users are free to use as many as they want as input and output. It is even possible to use the same feature as input and output, and this is not something unheard of. There is a significant amount of practical cases where we can find this setup, auto-encoders neural networks are an example of this practice. The OverviewPage's 'shaping' tab dedicated for CSV format shaping is represented in Figure 5.19. On the top right side of the page, there is the placeholder model that is what will result from the shaping procedure. On the 'shaping' tab, the user is prompt with 3 different sections:

- Dataset preview section;
- Feature engineering section;
- Action section.

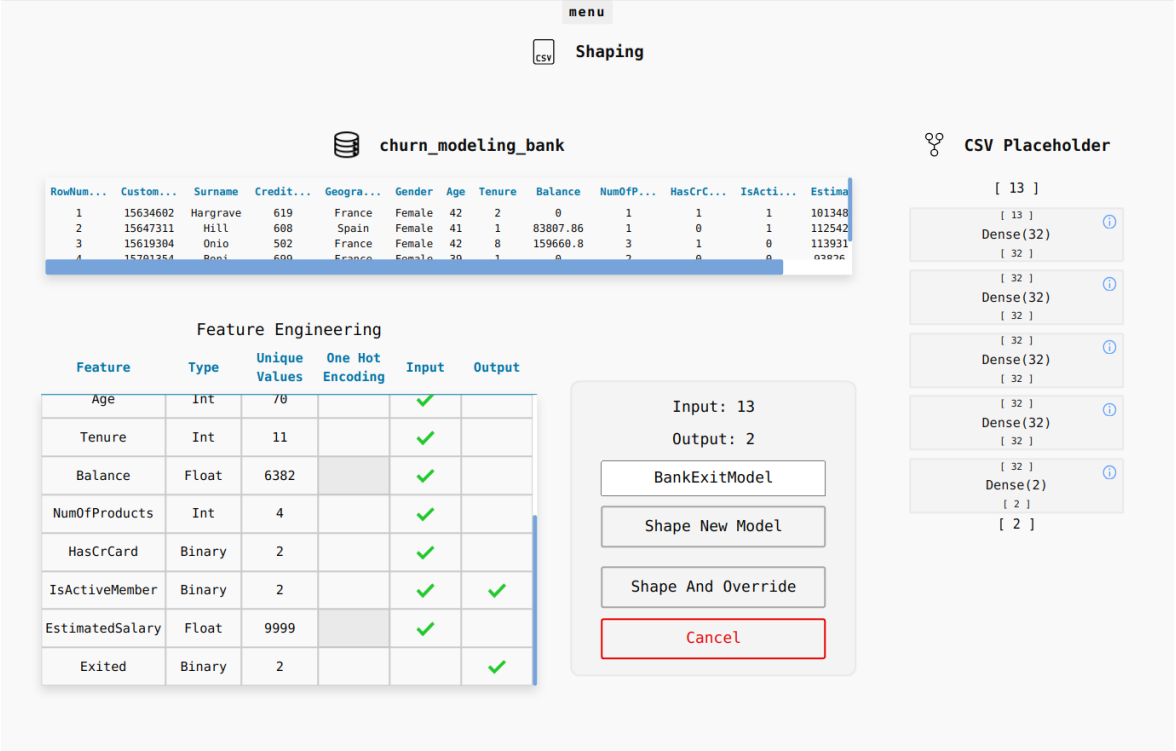

Figure 5.19: Overview page - shaping a CSV model

The data preview section is a small slice of the whole dataset. This preview is important to the user, otherwise, it would be extremely difficult to infer the true nature of every feature, even with its correspondent datatype. The feature engineering section provides a table to the user containing 2 more detailed information about each feature:

- **Type** - The data type of the correspondent feature;
- **Unique Values** - The number of unique values a feature contains on the entire dataset, for example, binary data type contains 2 unique values (0 and 1).

The input and output size are feature dependent, one hot encoding any selected features will increase the size of the input/output layer. The 3 right columns of this table are inputs to the user. For each feature, there is the possibility to 'one hot encode' it (the feature), select it as input, and select it as output. The buttons of the action section follow the same behavior as the image format action section. Having concluded the shaping phase, the user now has a model in the **editable** state and the page redirects the user to the previous 'overview' tab 5.20. In this state, the model is ready to be trained on the chosen dataset.

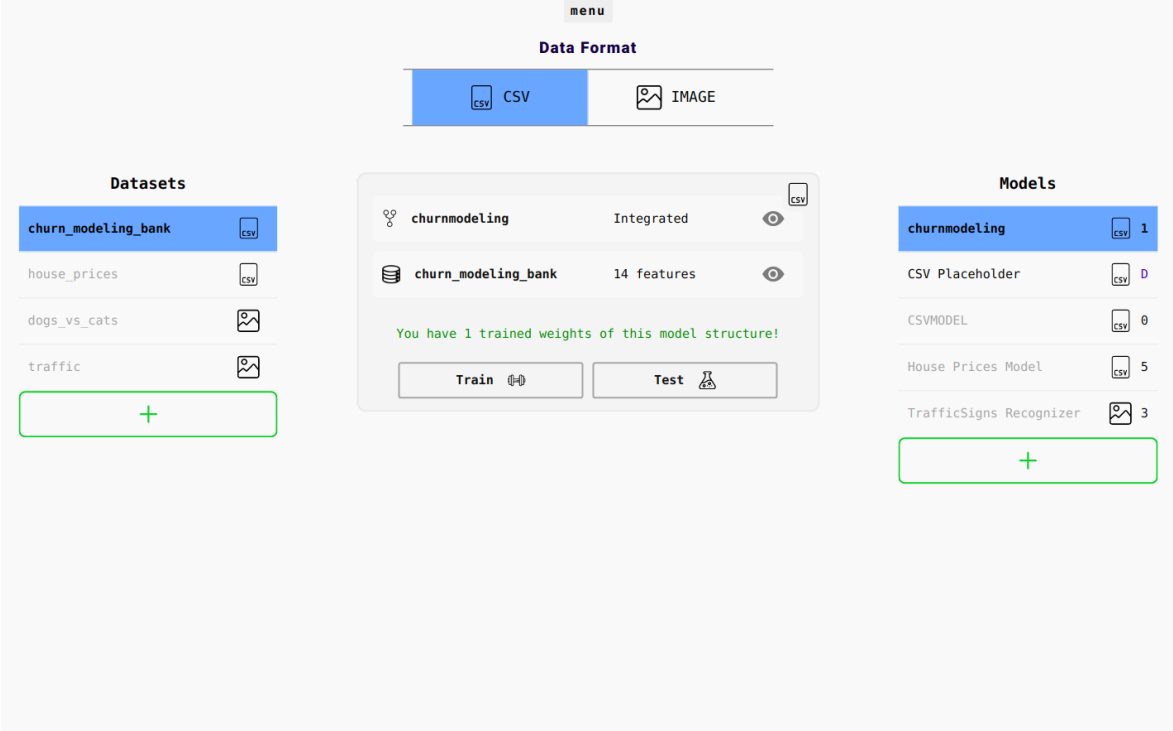

Figure 5.20: Overview page - model ready to train on dataset

The following 3 figures present the TrainingFlow section/component on 3 consecutive workflow steps to successfully train a model:

- The model is in editable state and the user is allowed to train it - Figure 5.21;
- By clicking the train button, the user is prompt with training options inputs (training parameters and data augmentation settings), having default values for most use cases - Figure 5.22;
- After the structure has at least one trained model, it is set to static state and, from that moment on, it is possible to test this trained models - Figure 5.23.

Figure 5.21: Training Flow - editable state view

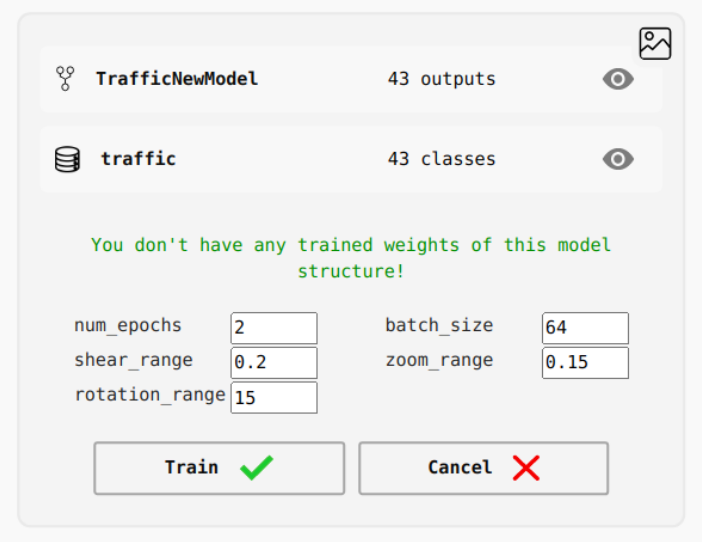

Figure 5.22: Training Flow - training view

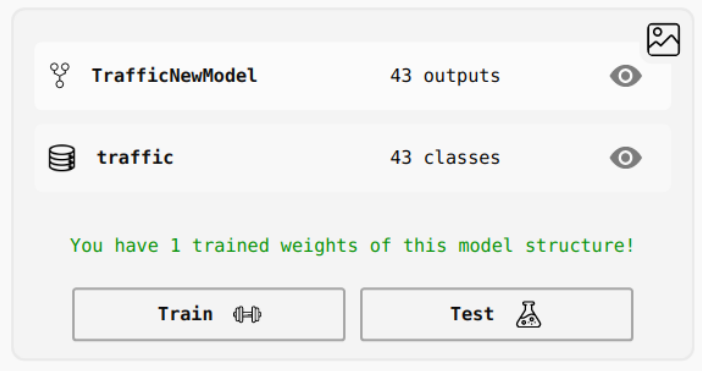

Figure 5.23: Training Flow - static state view

It is important to note that after the user requests a training session, WebML redirects the user to the ModelPage. If the user instead wants to test the model on the 'test' button, it will be redirected to the TestPage.

5.9 ModelPage

The ModelPage is dedicated to displaying detailed information regarding the models and possible actions. On this page, the user is allowed to create, edit, clone, download a pre-built project/experiment regarding the model, delete single trained weights and also the entire model structure. This page has 2 modes/tabs, the creation of a model structure and the display of a model structure.

5.9.1 Model creation

Creating a model involves a simple step by selecting the model's format (images or CSV). In the case where the user selected the image format, they will be prompt to specify the input size (Fig. 5.24).

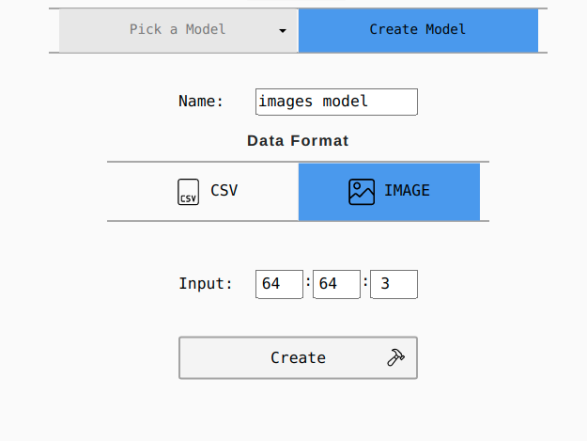

Figure 5.24: Model creation - image format

5.9.2 Model editing

Model edition is delivered to the user with a multiple action/configuration page regarding a model (Fig. 5.25).

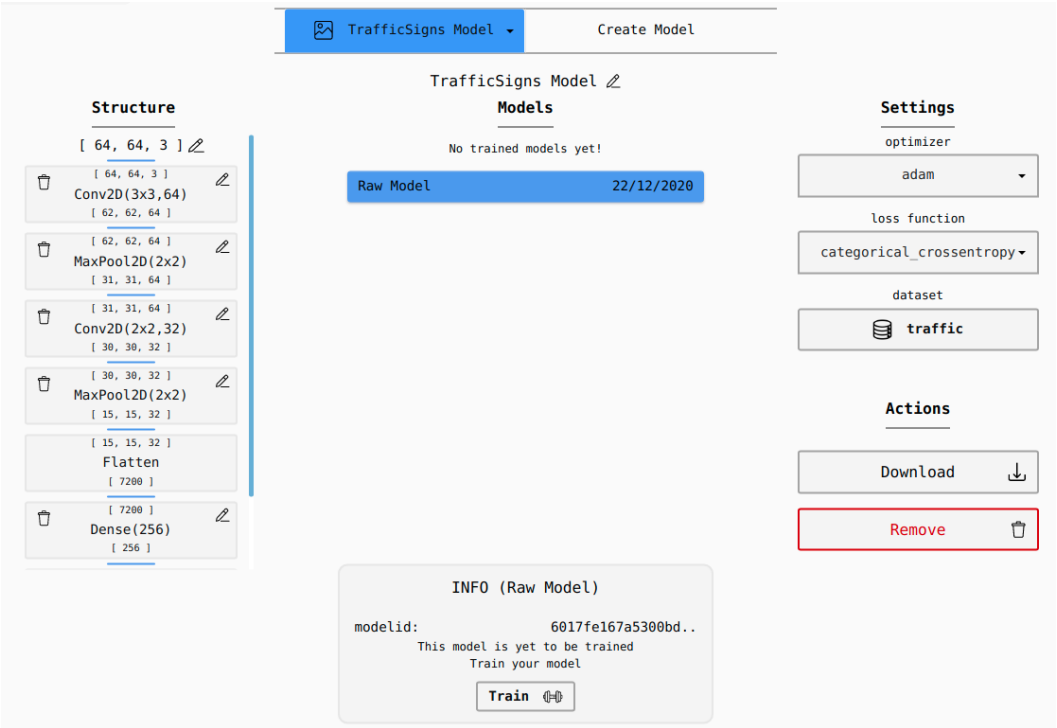

Figure 5.25: Model Page view - editable model

After a model is created, the user can edit and add layers to the neural network graph structure, each type of layer has its own properties. in Figure 5.26 are represented the 3 possible courses of action in a sequential layer structure.

- Edit properties of a layer;
- Add a layer;
- Remove a layer.

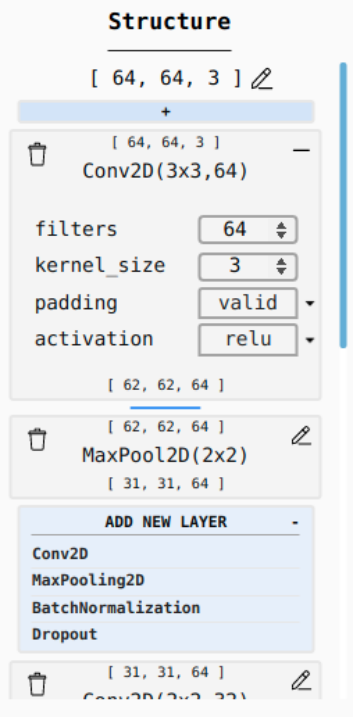

Figure 5.26: Model structure editor

Although transitions are not the core features of the interface, they are very helpful when editing a layer structure. The user can follow the size increase of the layer structure using their eyes. Without these transitions, HTML elements would just appear on the user's screen and it could lead to confusion when adding, editing or removing a layer. Models also have configurable loss functions and optimizers. By default, these model attributes are set to the most likely best choice for a specific format or dataset. The users are allowed to set these attributes to other values. Changing loss functions and optimizers can be useful especially for more experienced data scientist users (Fig. 5.27).

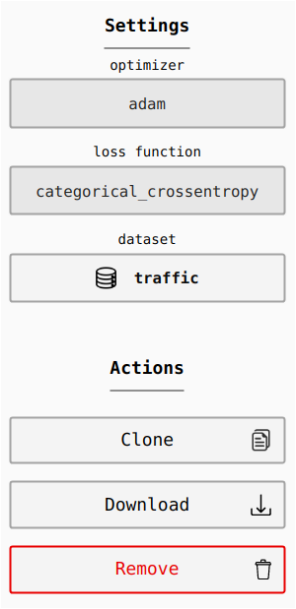

Figure 5.27: Model settings

5.9.3 Static model view

When the model's state is set to static, it is not possible to edit the model structure. The model view in this state is shown in Figure 5.28. Although these restrictions are applied to ensure data consistency, there is always the option to clone the model. Cloning the model allows the user to have a replication of the cloned model with the same model structure and the same target dataset but in the edit mode where there are no associated trained weights.

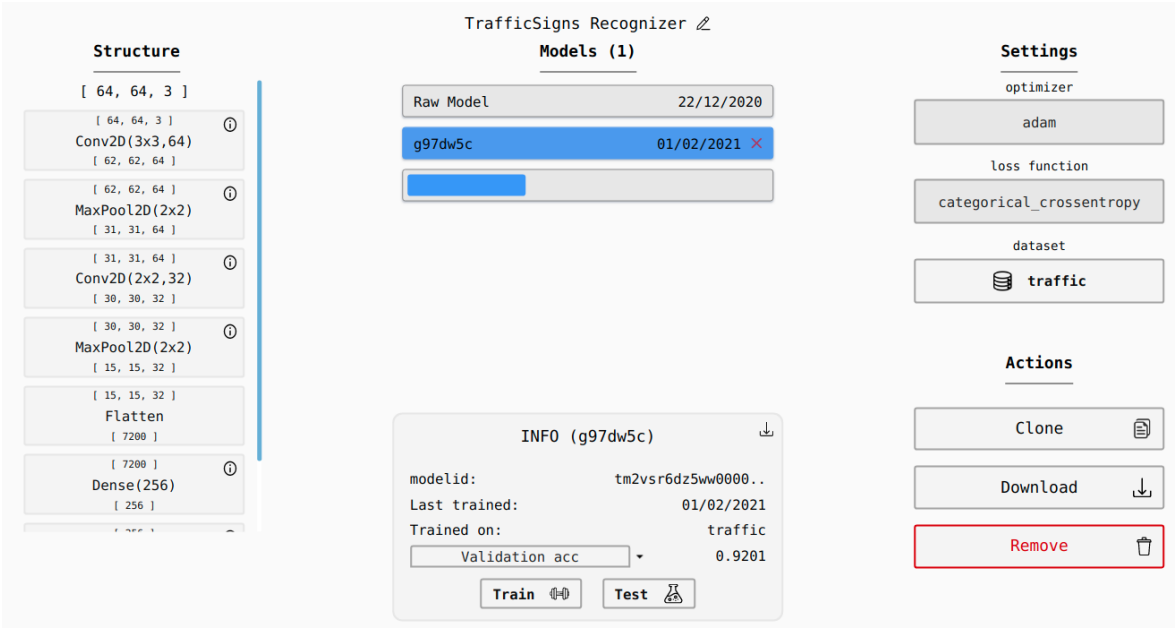

Figure 5.28: Model page view - static model

The model is no longer in editable mode when the first training session is requested. Requesting any training for a recently created model will set its layers as static. The same three vertical sections are displayed in figure 5.28, structure, models and settings. The structure section (left side) displays the structure of the neural network in layers, where parameters of each layer can be inspected. The middle section (Models) represents all the information related to the training weights and training sessions. The represented model in the example figure contains 1 trained model (trained weights) and is currently executing one training session that is about 30% complete. Trained weights can be obtained from previously trained weights in order to spare time for users. In the settings section, the model can no longer be saved but can be cloned. Making a cloning request of a model creates a new model with the same layer structure without any trained weights. This new cloned model on editable mode. In the static model's view, model structures have an additional feature in the UI that allows the user to download an experiment project regarding the target model and its trained weights.

5.9.4 Download Experiment

Download experiment feature allows the users to continue their work offline by downloading all the required files and all the trained weights that were originated by the UI into a zipped file. The compressed files provided are only compatible with the Linux operating system. The main purpose of these files is to provide significant help for copying or editing code for training sessions. It also provides the option to train already trained weights even further just like it is possible when using the

UI. To benefit from this feature, the user is required to install the correspondent dependencies and set up a virtual environment as was previously explained in section 4.10. The overall usefulness and value of this feature depend on the user's intents, current hardware and software, and background knowledge. The main Python script 'train.py' has multiple arguments that offer flexibility to the user. From all these arguments, only one is required. The list of arguments that both data formats contain is displayed in Table 5.1.

argument	description
dataset (-d)	location of the target dataset (required)
batch (-b)	training batch size
epochs (-e)	training epoch number
weights (-w)	location of the already trained weights
modelname	filename of trained weights result

Table 5.1: List of argument options on the 'download experiment' training script

The argument options of the training script 'train.py' are not the same for the different formats. Image format experiments have extra arguments regarding data augmentation. These arguments are the training parameters displayed in Figure 5.22.

5.10 DatasetPage

The DatasetPage is a page where individual datasets can be inspected or where new datasets can be uploaded. These 2 different options are represented in different tabs on the same page. Inspection of different formats displays different information to the user.

5.10.1 Inspect CSV datasets

In the case of CSV data, a preview of the dataset is displayed to allow the user to raise intuition about the dataset content (Fig. 5.29). The user is also provided with higher detailed information about all the features range of values and data types in a table presented below the preview.

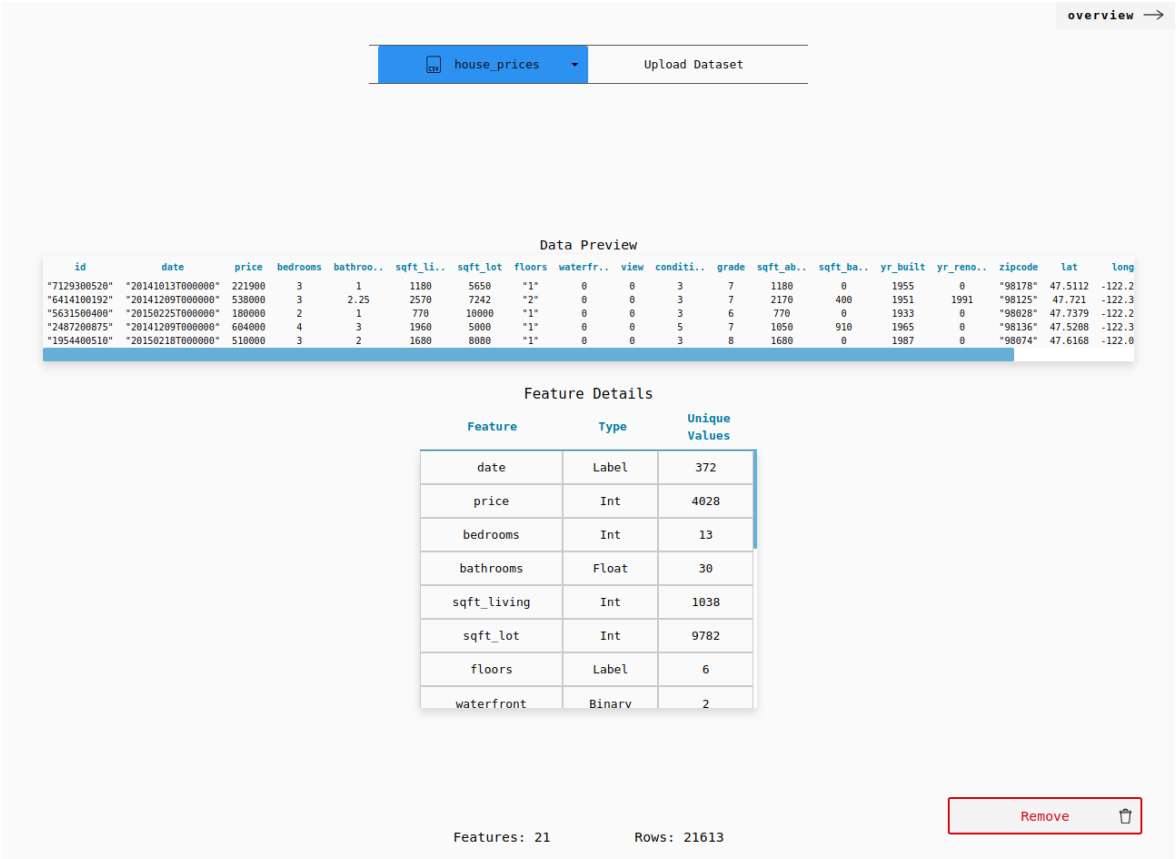

Figure 5.29: Dataset page - inspecting a CSV dataset

5.10.2 Inspect image datasets

Opposite to the CSV dataset, image datasets do not supply a high amount of information to the user. The number of classes along with an example of each class is enough for the user to understand what the dataset is about and how to use it. In the example Figure 5.30 the selected dataset is composed of 43 different classes of German traffic signs [71].

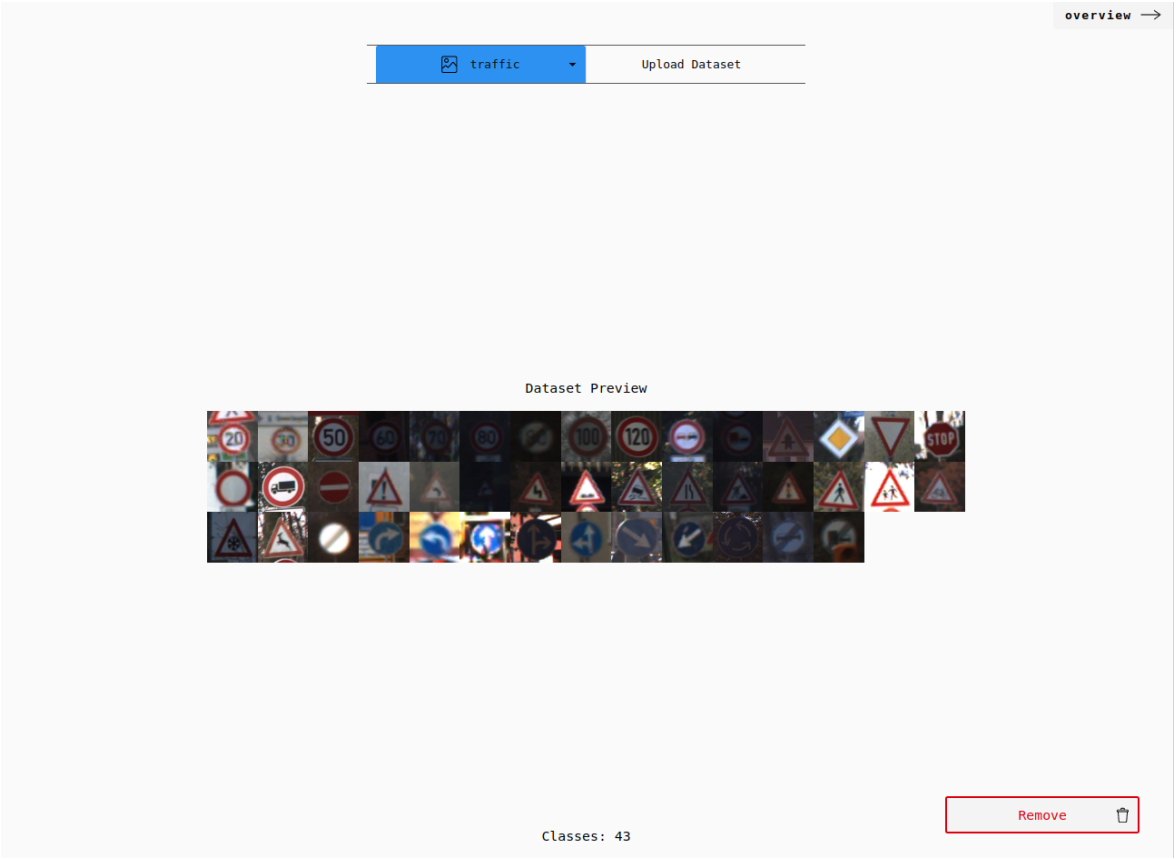

Figure 5.30: Dataset page - inspecting a image dataset

5.10.3 Uploading datasets

Uploading a dataset is as simple as selecting the respective files from the disk into the correct format container displayed in Figure 5.31. The hardest step in this procedure is to validate a dataset that fulfills all the requirements in order to be consumed. Image classification dataset requires a rigid directory structure inside a zip archive while CSV data only requires a CSV file with headers in the first row. Uploading a big dataset can take a while until it is completed and validated. The user can and should take advantage of this uploading time to create a model dedicated to targeting this dataset.

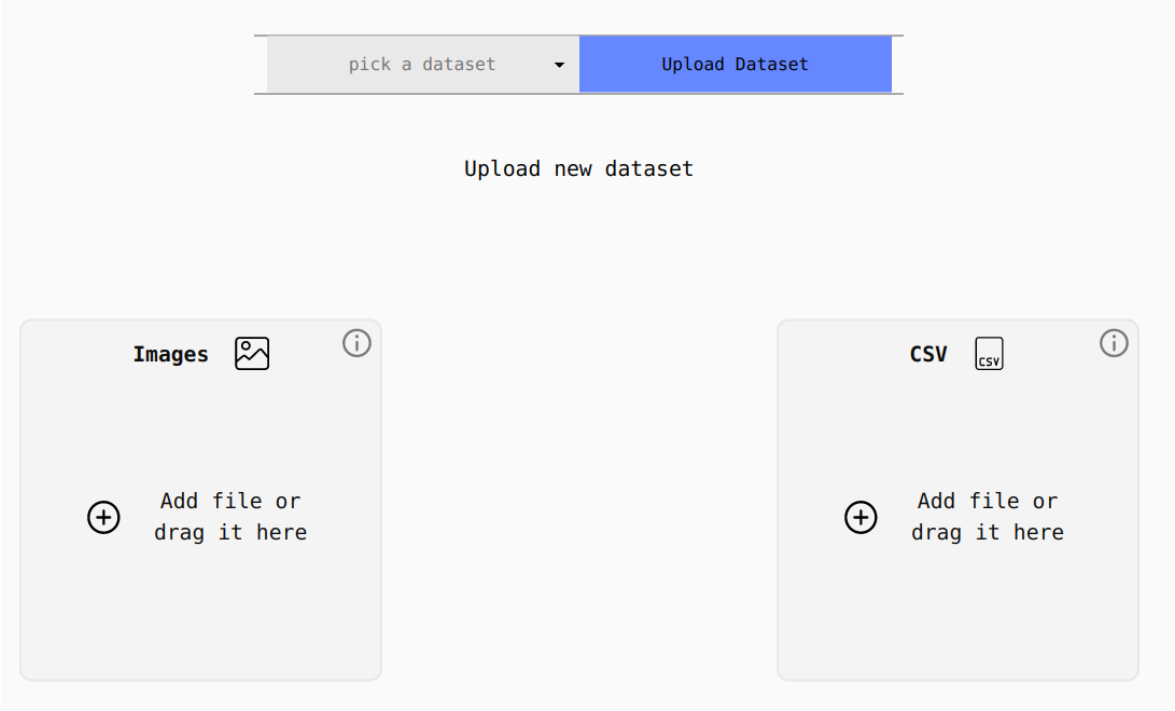

Figure 5.31: Dataset page - uploading section

The current UI, allows the user to either drag a file into the uploading container component or directly insert it through the 'upload file' pop up from the browser. The UI recommends a name to the dataset when a file is inserted (not uploaded) into the container based on the name of the original file (Fig. 5.32). This file's name can be edited as long as it is different from other datasets owned by the current user or from the public ones.

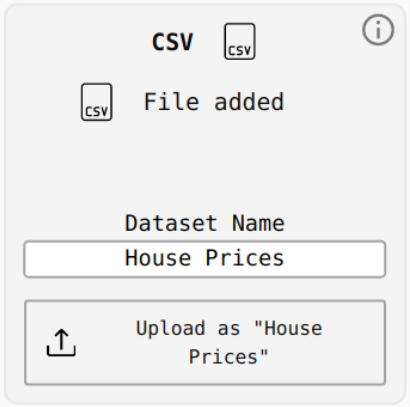

Figure 5.32: Dataset page - file upload

In these circumstances the user must press the upload button to continue. Uploading big files can take a long time to transfer through the internet. Users are supplied with a progress status to estimate how long the file will take until it finishes uploading (Fig. 5.33).

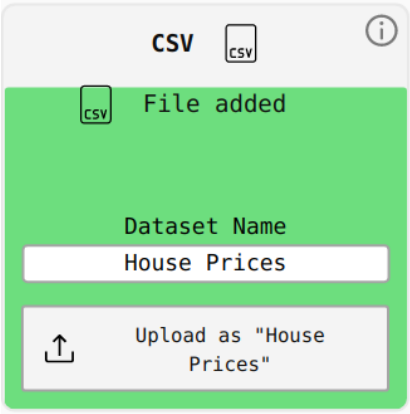

Figure 5.33: Dataset page - uploading progress (75%)

When the upload is completed, WebML's services carefully examine and validate all the uploaded dataset content. If for any reason the file is not in a valid format, WebML takes the responsibility of helping and indicating the issue that invalidated the dataset uploading (Fig. 5.34).

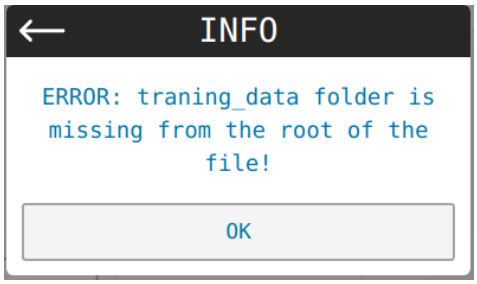

Figure 5.34: Dataset page - error uploading dataset of image format

5.11 TrainPage

The training page is where all training sessions and previously trained models are presented (Fig. 5.35) of all structures the user owns. This page is essential to keep track of all the training sessions related information. Although the ModelPage provides information on all the current training sessions and the trained models of a specific structure. The training page contains two sections, a smaller one on the left and a bigger one on the right. The left section is dedicated to the current training sessions where information such as epoch number, estimated time left and several training evaluation metrics are presented. The section on the right is meant to be a model training history, it shows brief information for the models sorted by the date of when the training was completed. The information related to each trained model is very similar to the one presented at the bottom of the ModelPage extending the structure name property. Both of these sections (training and history) are scrollable as each of them can grow significantly in the bottom direction when the respective content increases. The training page is considered an optional page. Its content is not too relevant for a regular user since the main pages are enough to keep the user satisfied and to fulfill their goals. However, administrators take higher advantage of this page since they are provided with all the training sessions of models in real-time for better monitoring.

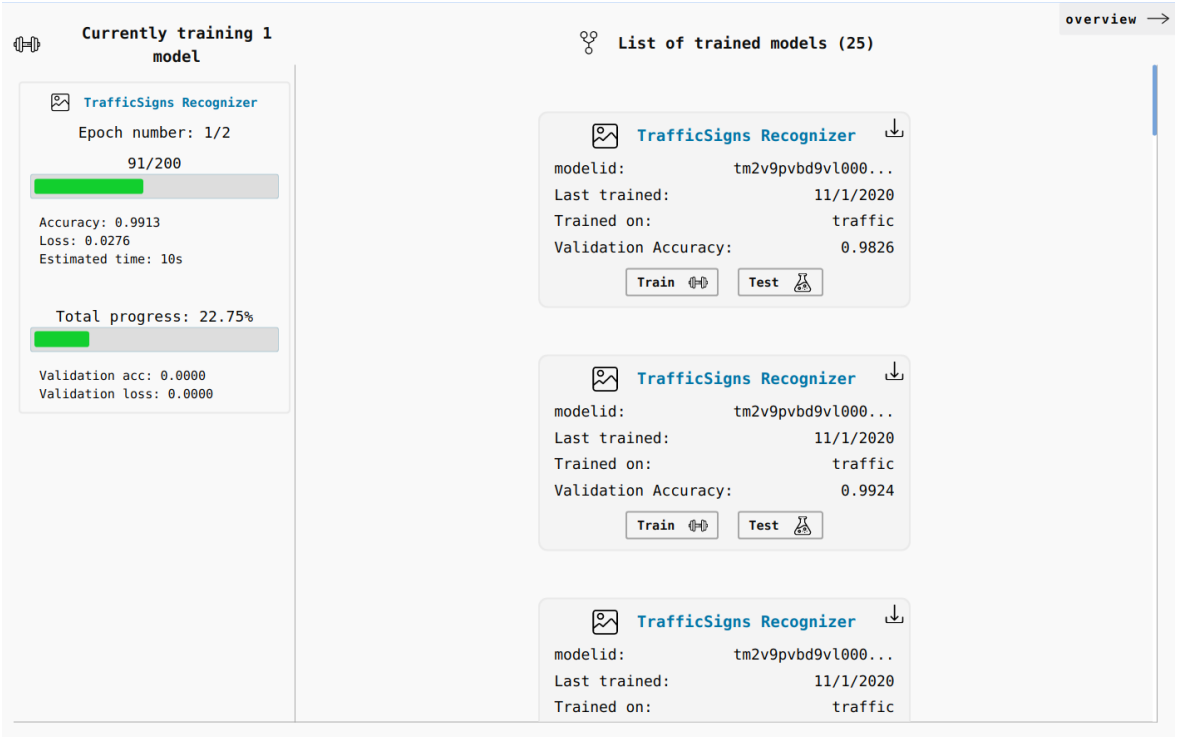

Figure 5.35: Training page

5.12 TestPage

This is the page where the user can test the trained models. Testing a model involves the user to input some data to a model and observe its prediction output. For a model to be considered trained, its correspondent model structure was already created in ModelPage, shaped in OverviewPage and requested to be trained. For each different model formats (images and csvdata) the user is prompt with a completely different interface to properly test the model.

5.12.1 Image classification

Image classification prediction requires only an image as input. The user can upload an image from its device or use the webcam. The test page interface for image classification shows in the panel the inputted image on the left and the output result on the right (Fig. 5.36). On the output side, the user is informed of the class name (from the dataset) with its correspondent examples of images from the dataset.

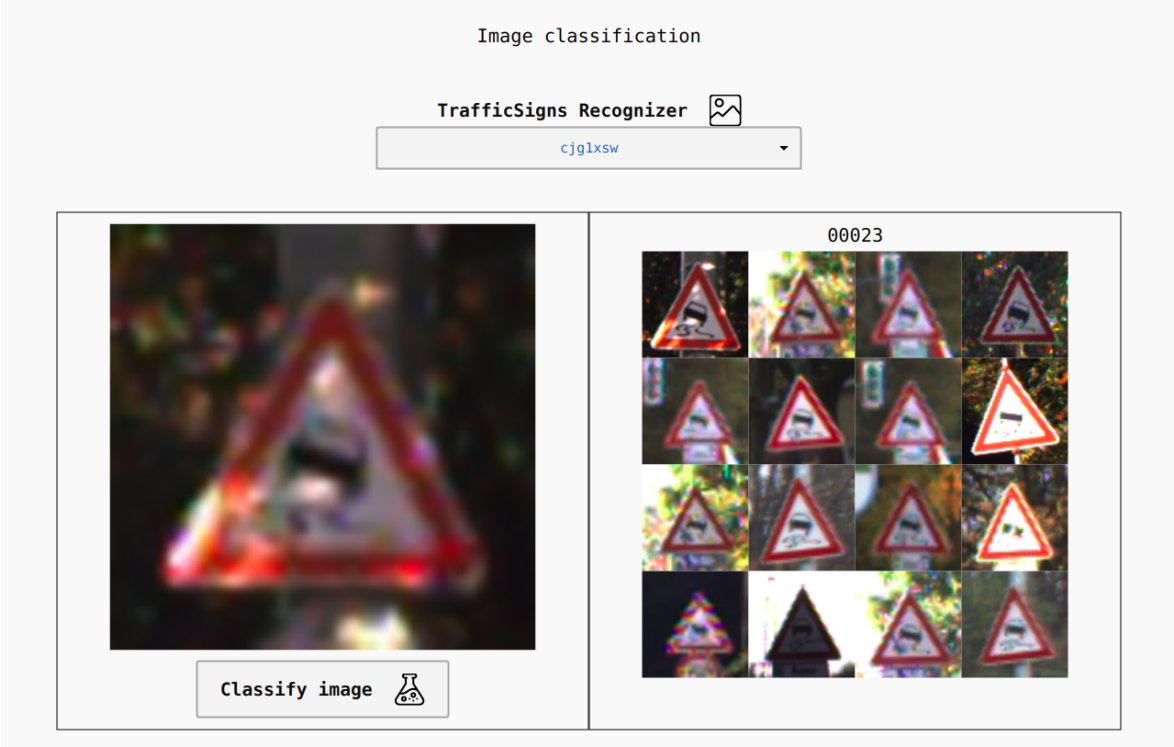

Figure 5.36: Image classification test section

5.12.2 CSV feature prediction

CSV feature prediction is quite more complex to the user when compared to the image classification where sending the image is enough. The user is required to input every feature that was set as input at the time the model was shaped. This interface table is similar to the model shaping back in OverviewPage (Fig. 5.19).

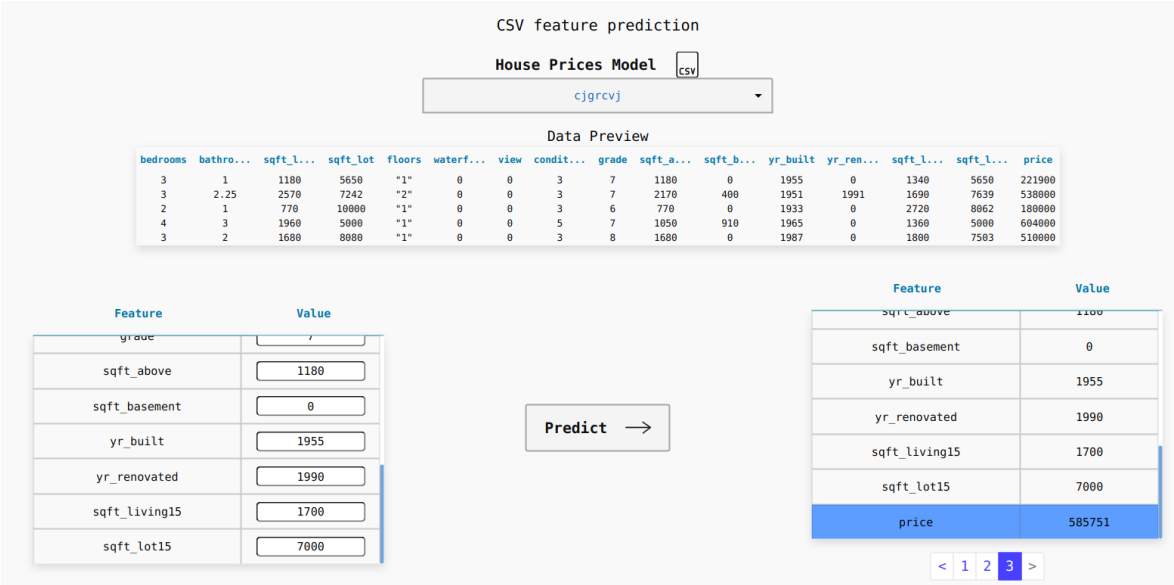

Figure 5.37: CSV feature prediction section

6

Conclusion

This section presents the main contributions of this work and all the possible extensions to the future. Although the resulting platform has exceeded the initial expectations, there is still room for further improvements and extensions.

6.1 Contributions

The main objective of this dissertation was to study, design and develop a distributed web platform for deep learning processes, capable of simplifying the training and testing of neural networks for a variety of users and a variety of problems, including teaching and research. As deep learning has become popular in the last few years but the access to a efficient GPU hardware and the setup of a working environment is still a barrier to a more generalized usage. As a solution, WebML was conceived. It is a platform that abstracts the processes behind this technology, integrating distributed network nodes with dedicated hardware, and providing complex services behind an interactive and innovative web portal.

The navigation on WebML stands out from the average web page where a regular navigation bar is used and is the setup that dominates the market. In WebML, every page has a relative position to each other. Smooth transitions and control keys over the navigation system satisfy the user by taking advantage of the innate orientation skills that humans are born with. The user interface was designed to be very user-friendly and intuitive in the workflow. Navigating back to a previously seen page is facilitated since all pages are visible in zoom out mode. The almost black and white design of the UI increase the contrast that is advantageous to the visualization of the web page elements.

WebML allows a set of complex operations with just a few mouse clicks, including the registration and of new computational nodes and users, the management of datasets, the design and training of models among others. The models can be trained with public and private datasets and multiple metrics are provided to evaluate the training performance. In the end, it is possible to test the models against new input samples and download the prediction scripts, among other utilities. All the processes can be monitored and managed by the system administrators, including the users, models, training sessions and the state of computational nodes and their GPUs.

Training sessions require intelligent management to cope with the multiple hardware providers in order to efficiently schedule tasks. A mechanism for smart distribution of the load through the

available hardware was implemented, seeking to target the terminal containing the GPU with the highest available memory. This approach is beneficial to the overall solution performance and to provide the best computational capacity for the training sessions.

Finally, WebML leverages from an in-depth study and analysis of tabular data (CSV files), mainly extensible models with integrated preprocessing steps, the so-called pipeline models. This method enables the storage of the preprocessing steps and the neural network model in a single file. All this process was possible thanks to memory management algorithms that solved recurring problems associated with saving Keras models.

6.2 Future Work

As future work, many extensions to WebML should be considered:

- Increase relevant content delivered to the user int tutorial and about page;
- Add redundancy for the Master server in case of system failure;
- Allow the creation of more flexible models. Keras functional API can handle models with non-linear topology, shared layers, and even multiple inputs or outputs with different loss functions and relevance;
- Allow visualization of the best models' structures for every public dataset;
- Add support for a wider range of model layer types;
- Add target individual class selection for models with image format as opposed to classifying the whole dataset where all classes all selected;
- Add compatibility to other types of data formats for Generative adversarial networks (GANs), reinforcement learning, natural language processing and other techniques;
- Add an automatic model builder for any proposed dataset. From the training set only, the platform should be able to define, build and train a model without the need for the user to define any layer;
- Display learning curves of training sessions;
- Add custom object-oriented deep learning layers. The platform should allow the users to create and edit parameterizable custom layers. These custom layers can be modeled and by assuming the task of a groups of sequential layers;
- Display multiple chart analysis for the administrator using the history data logged in the database;
- Improve the download experiment feature to be more easily integrated into more complex projects;
- Allow administrators to set any non-public dataset as public.

6.3 Final Considerations

Technology evolution is based on gathering existing technologies and creating something new from them. Unless the intention is to improve the current wheel, there is no point in investing time reinventing it. WebML was only conceivable due to the fact that modern times are blessed with electricity, computers, telecommunication systems, mathematical studies, fully developed frameworks and online search engines that accelerate all the process. Nowadays there is no longer a need to be aware of the complexity behind all the tools we use. The only requirement is knowing how to use these modern tools, it is the strength of complexity abstraction. A cellphone user is not required to know

about network routing to make a phone call across the world as well as someone that wants to use a neural network does not require to know the math underneath it or even how to program. WebML users only need to know their objectives and how to use the provided graphical interface. Abstraction is essential to humans and technology, otherwise, evolution would be impossible.

Bibliography

[1] Emmert-Streib; Yang; Feng; Tripathi; Dehmer, "An introductory review of deep learning for prediction models with big data", 2020. DOI: 10.3389/frai.2020.00004.

[2] S. S. et al., "A review of deep learning with special emphasis on architectures, applications and recent trends", 2020. DOI: https://doi.org/10.1016/j.knosys.2020.105596.

[3] S. A. M. A., "Review of deep learning algorithms and architectures", 2019. DOI: 10.1109/ACCESS.2019.2912200.

[4] D. Coombes. (2017). Deep learning for games, [Online]. Available: https://developer.nvidia.com/deep-learning-games.

[5] H. Z. et al., "Deep learning for image-based cancer detection and diagnosis a survey", 2018. DOI: 10.1016/j.patcog.2018.05.014.

[6] (). U.s. deep learning market, by solution, 2014 - 2025 (usd million), [Online]. Available: https://www.grandviewresearch.com/static/img/research/us-deep-learning-market.png.

[7] M. Maclik Mba, "What is artificial intelligence?", Nov. 2019. [Online]. Available: https://www.researchgate.net/publication/337089782_What_is_Artificial_Intelligence.

[8] K.-D. Gronwald, "Artificial intelligence", in. Jul. 2020, pp. 111–135, ISBN: 978-3-662-59810-8. DOI: 10.1007/978-3-662-59811-5_7.

[9] A. L. Samuel, "Some studies in machine learning using the game of checkers", *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, Jul. 1959, ISSN: 0018-8646. DOI: 10.1147/rd.33.0210.

[10] B. C. M, *Pattern Recognition and Machine Learning*. Springer, 2012, ISBN: 978-0-387-31073-2.

[11] (Feb. 2018). Metrics to evaluate your machine learning algorithm, [Online]. Available: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234.

[12] J. Schmidt, M. R. G. Marques, S. Botti, and M. A. L. Marques, "Recent advances and applications of machine learning in solid-state materials science", 2019. DOI: https://doi.org/10.1038/s41524-019-0221-0.

[13] F. Maleki, K. Ovens, K. Najafian, B. Forghani, C. Reinhold, and R. Forghani, "Overview of machine learning part 1: Fundamentals and classic approaches", 2020. DOI: https://doi.org/10.1016/j.nic.2020.08.007.

[14] M. Abdallh, K. Bilal, and A. Babiker, "Machine learning algorithms", *International Journal of Engineering, Applied and Management Sciences Paradigms*, vol. 36, pp. 17–27, Jun. 2016.

[15] A. Dogan and D. Birant, "Machine learning and data mining in manufacturing", 2020. DOI: https://doi.org/10.1016/j.eswa.2020.114060.

[16] (). Data science and machine learning, [Online]. Available: https://www.ibm.com/uk-en/analytics/machine-learning.

[17] R. Sharma, A. Sinha, and M. Prateek, "Use of reinforcement learning as a challenge: A review", 2013. DOI: 10.5120/12105-8332.

[18] A. Hammoudeh, "A concise introduction to reinforcement learning", 2018. DOI: 10.13140/RG.2.2.31027.53285.

[19] (2017), [Online]. Available: https://lakshaysuri.wordpress.com/2017/03/19/machine-learning-supervised-vs-unsupervised-learning/.

[20] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", 2015. DOI: doi:10.1038/nature14539.

[21] H. Suna, H. V. Burtonb, and H. Huangb, "Machine learning applications for building structural design and performance assessment: State-of-the-art review", 2020. DOI: https://doi.org/10.1016/j.jobe.2020.101816.

[22] Raschka, Sebastian and Patterson, Joshua and Nolet, Corey, "Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence", *Information*, vol. 11, no. 4, 2020, ISSN: 2078-2489. DOI: 10.3390/info11040193. [Online]. Available: https://www.mdpi.com/2078-2489/11/4/193.

[23] A. G. Ivakhnenko and V. G. Lapa, "Cybernetics and forecasting techniques", 1967. [Online]. Available: https://books.google.pt/books?id=rGFgAAAAMAAJ&redir_esc=y.

[24] A.-r. Mohamed, "Deep neural networks for acoustic modeling in speech recognition", Apr. 2015. [Online]. Available: http://airesearch.com/ai-research-papers/deep-neural-networks-for-acoustic-modeling-in-speech-recognition.

[25] S. Kulkarni and G. Harman, "Neural networks: Perceptrons", in. Apr. 2011, pp. 86–98, ISBN: 9780470641835. DOI: 10.1002/9781118023471.ch9.

[26] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Neurocomputing: Foundations of research", in, J. A. Anderson and E. Rosenfeld, Eds., Cambridge, MA, USA: MIT Press, 1988, ch. Learning Representations by Back-propagating Errors, pp. 696–699, ISBN: 0-262-01097-6. [Online]. Available: http://dl.acm.org/citation.cfm?id=65669.104451.

[27] C. D. Costa. (2020). Best python libraries for machine learning and deep learning, [Online]. Available: https://towardsdatascience.com/best-python-libraries-for-machine-learning-and-deep-learning-b0bd40c7e8c.

[28] M. Opala. (2019). Deep learning frameworks comparison – tensorflow, pytorch, keras, mxnet, the microsoft cognitive toolkit, caffe, deeplearning4j, chainer.

[29] M. Abadi, M. Isard, and D. G. Murray,

[30] T. Bianco, "Gpu acceleration advancing the evolution of fast and big data", 2017. [Online]. Available: https://www.datanami.com/2017/07/24/gpu-acceleration-advancing-evolution-fast-big-data/.

[31] D. Atkin, "The right gpu for you", 2007. [Online]. Available: https://web.archive.org/web/20070506033224/http://computershopper.com/feature/200704_the_right_gpu_for_you.

[32] A. Brodtkorb, T. Hagen, and M. Satra, "Graphics processing unit (gpu) programming strategies and trends in gpu computing", *Journal of Parallel and Distributed Computing*, vol. 73, pp. 4–13, Jan. 2013. DOI: 10.1016/j.jpdc.2012.04.003.

[33] J. Rowe, "The continuing importance of gpus for more than just pretty pictures", Mar. 2017. [Online]. Available: https://www10.mcadcafe.com/blogs/jeffrowe/2017/03/16/the-continuing-importance-of-gpus-for-more-than-just-pretty-pictures/.

[34] H. Shah and T. Soomro, "Node.js challenges in implementation", *Global Journal of Computer Science and Technology*, vol. 17, pp. 72–83, May 2017. [Online]. Available: https://www.researchgate.net/publication/318310544_Nodejs_Challenges_in_Implementation.

[35] R. Rickards. (2020). Which back-end frameworks better to learn in 2020, [Online]. Available: https://morioh.com/p/d024b656ccc2.

[36] (2021). Server-side web frameworks, [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks.

[37] B. Warf, *Hypertext markup language*, 2018. DOI: 10.4135/9781473960367.n127.

[38] M. Daubs, "Html and css", in. Dec. 2019, pp. 762–764, ISBN: 9781483375533.

[39] B. Mason, "Cascading style sheets", in. Jan. 2003, pp. 58–99. DOI: 10.1007/978-1-4302-5362-4_3.

[40] D. Mazinanian and N. Tsantalis, "An empirical study on the use of css preprocessors", 2016. DOI: 10.1109/saner.2016.18.

[41] D. Flanagan, "Javascript - the definitive guide",

[42] (), [Online]. Available: https://w3techs.com/technologies/details/cp-javascript/.

[43] M. Frisbie, "Javascript in html", in. Oct. 2019, pp. 13–24, ISBN: 9781119366560. DOI: 10.1002/9781119366560.ch2.

[44] (2019). Javascript html dom, [Online]. Available: https://www.w3schools.com/js/js_htmldom.asp.

[45] A. Kumar, "React vs. angular vs. vue.js: A complete comparison guide", [Online]. Available: https://dzone.com/articles/react-vs-angular-vs-vuejs-a-complete-comparison-gu.

[46] C. Rojas, "Working with vue.js", in. Dec. 2019, pp. 83–146. DOI: 10.1007/978-1-4842-5334-2_5.

[47] D. Sozo. (2018). 10 reasons to use nuxt.js for your next web application, [Online]. Available: https://medium.com/vue-mastery/10-reasons-to-use-nuxt-js-for-your-next-web-application-522397c9366b.

[48] (2020). What is ssr and why you need it, [Online]. Available: https://appradius.co/blog/what-is-ssr-react-ssr-next-js-zeit.

[49] T. F. Iskandar, M. Lubis, T. F. Kusumasari, and A. R. Lubis, "Comparison between client-side and server-side rendering in the web development", vol. 801, 2020. DOI: 10.1088/1757-899X/801/1/012136.

[50] J. Shetty, "A state-of-art review of docker container security issues and solutions", *American International Journal of Research in Science, Technology, Engineering Mathematics*, Jan. 2017.

[51] O. Grillmeyer, "Database management systems", in. Jan. 1998, pp. 285–318, ISBN: 978-1-4419-2855-9. DOI: 10.1007/978-1-4757-2937-5_12.

[52] Aparajitha.R.S.V, and Kavitha, M.K and Monisha, T.R.P and Pavithra, T.S.B and P, Vinoth, "Database management systems", *International Journal of Computer Applications*, vol. 1, Feb. 2010. DOI: 10.5120/179-310.

[53] A. Bhardwaj and K. Sharma, "Types of queries in database system", *International Journal of Computer Applications in Technology*, vol. 7, pp. 149–152, Oct. 2015. [Online]. Available: https://www.researchgate.net/publication/303313899_Types_of_Queries_in_Database_System.

[54] M. Gupta and D. Badal, *Comparative study of indexing techniques in dbms*, Mar. 2012.

[55] "Database", May 2020. [Online]. Available: https://www.britannica.com/technology/database.

[56] P. Mccaffrey, "Relational databases", in. Jan. 2020, pp. 17–29, ISBN: 9780128149157. DOI: 10.1016/B978-0-12-814915-7.00002-8.

[57] J. Eckstein and B. Schultz, "Basic structured query language (sql)", in. Nov. 2017, pp. 215–251, ISBN: 9781119329411. DOI: 10.1002/9781119430087.ch10.

[58] S. Tamane, "Non-relational databases in big data", Mar. 2016, pp. 1–4. DOI: 10.1145/2905055.2905194.

[59] (2020). What is a non-relational database?, [Online]. Available: https://www.mongodb.com/non-relational-database.

[60] (2020). What is azure machine learning studio?, [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/overview-what-is-machine-learning-studio.

[61] Tzvi Keisar. (2019). Simplifying ai with the new automated machine learning ui, [Online]. Available: https://azure.microsoft.com/pt-pt/blog/simplifying-ai-with-automated-ml-no-code-web-interface.

[62] Mar. 2015. [Online]. Available: http://edutechwiki.unige.ch/en/Neural_Designer.

[63] (2017). About us - deepcognition.ai, [Online]. Available: https://deepcognition.ai/about-us/.

[64] J. H. S. Gugger, "Fastai - a layered api for deep learning", Feb. 2020. DOI: 10.3390/info11020108. [Online]. Available: https://arxiv.org/abs/2002.04688.

[65] (2020). Neural network : Artificial intelligence, [Online]. Available: https://en.wikipedia.org/wiki/Neural_network#Artificial_intelligence.

[66] S. Bae and S. Bae, *Big-o notation*, 2019. DOI: 10.1007/978-1-4842-3988-9_1.

[67] K. Shingala, "Json web token (jwt) based client authentication in message queuing telemetry transport (mqtt)", Mar. 2019. [Online]. Available: https://www.researchgate.net/profile/Krishna_Shingala/publication/331587509_JSON_Web_Token_JWT_based_client_authentication_in_Message_Queuing_Telemetry_Transport_

MQTT/links/5ca1f32392851cf0aea5d2e4/JSON-Web-Token-JWT-based-client-authentication-in-Message-Queuing-Telemetry-Transport-MQTT.pdf.

[68] R. Snijders Matzat, "Big data: Big gaps of knowledge in the field of internet science", 2012. [Online]. Available: https://www.ijis.net/ijis7_1/ijis7_1_editorial.pdf.

[69] V. Bushaev. (2017). How do we train neural networks ?, [Online]. Available: https://towardsdatascience.com/how-do-we-train-neural-networks-edd985562b73.

[70] (). Use volumes, [Online]. Available: https://docs.docker.com/storage/volumes/.

[71] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing and Christian Igel, "Detection of traffic signs in real-world images: The german traffic sign detection benchmark", in *International Joint Conference on Neural Networks*, 2013.