UNIVERSITY AVEIRO

Neural Net Tracker

Neural Network Behavior Analysis Tool During Training

Diogo Bento
João Soares
Hugo Ferreira
Pedro Casimiro
Pedro Silva

Department of Electronics, Telecommunications and Informatics

Advisor: Mário Antunes

Second Advisor: Rui L. Aguiar

2021

Neural Net Tracker

Neural Network Behavior Analysis Tool During Training

Project in Informatics Report from the Degree in Informatics Engineering at the University of Aveiro, conducted by Diogo Bento, João Soares, Hugo Ferreira, Pedro Casimiro and Pedro Silva under the guidance of Mário Antunes, Doctoral Researcher at Instituto de Telecomunicações Aveiro and Rui Luís Aguiar, Head of Networks at Instituto de Telecomunicações Aveiro.

Keywords

Neural Network
Deep Neural Network
Artificial Intelligence
Docker
Machine Learning
Data Logging
Back Propagation

Abstract

Deep neural networks have been shown to be able to learn the most complex models, sometimes exceeding human capability in prediction and classification tasks, being used in a variety of important tasks such as Image Recognition and Stock Forecasting.

Neural networks are trained through an algorithm called back propagation, where the error is propagated from the end of the network to the beginning to make corrections. This training process is rather complex and not yet fully understood. There are several parameters that condition the precision obtained by a network and their impact on the training phase is not understood completely.

These technical problems can cause many major incidents and problems in areas using artificial intelligence, like face recognition where many have reported problems with gender and racial bias, or in the medical area where artificial intelligence is improving diagnostics of cancer but a simple technical problem can lead to a life-or-death situation.

In this project, we developed a periodic parameter collection platform that deploys neural networks in a containerized environment, said platform can launch several training experiments and persistently store the chosen parameters for later analysis, and includes a visualization interface capable of selecting, organizing, comparing, and showing the evolution of the parameters collected in a variety of relevant graphics and capable of managing the different training experiments.

The developed platform integrates a user registration and management system, users can use a HTML-based visualization interface to configure and launch single or several training experiments in a containerized environment.

It is capable of managing the life cycle of the different training experiments, analyzing the parameters collected through a variety of relevant graphs, comparing them with different training experiments associated via a tagging system, and downloading the data collected, with the goal of better understanding the impact different parameters have in the training phase and improving management of several training experiments, in order to decrease major incidents and problems in areas using artificial intelligence.

Contents

List of Figures

List of Tables

Abbreviations

JSON	JavaScript Object Notation
NN	Neural Network
AI	Artificial Intelligence
ML	Machine Learning
HTML	Hypertext Markup Language
CSV	Comma-separated values
ARFF	Attribute-Relation File Format
NPZ	NumPy file
WWW	World Wide Web

Chapter 1

Introduction

Deep neural networks are a set of algorithms, modelled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through machine perception, labelling or clustering of raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text, or time series, must be translated. Neural networks are trained through an algorithm named backpropagation, where the error is propagated from the end of the network to the beginning to make corrections. This training process is rather complex and hard to fully grasp due to the several parameters that condition the precision obtained by a network, furthermore, their impact on the training phase is not understood.

These technical problems can produce many major incidents and problems in areas using artificial intelligence, like in medicine, where artificial intelligence is improving diagnostics of cancer but a simple technical problem, including an under-performing artificial intelligence can create life-or-death situations, or in the area of face recognition where many have reported problems with gender and racial biases in artificial intelligence.

In this project, we developed a periodic parameter collection platform that deploys neural networks in a containerized environment, said platform can launch several training experiments and persistently store the chosen parameters for later analysis, and includes a visualization interface capable of selecting, organizing, comparing, and showing the evolution of the parameters collected in a variety of relevant graphics and capable of managing the different training experiments.

Early works to mitigate these technical problems and consequently decrease the major incidents and problems cause by them, are for example *TensorBoard*, that provides visualization and tooling needed for machine learning experimentation with tracking and visualization of several parameters and *NeuroX: A Toolkit for Analysing Individual Neurons in Neural Networks*, had some degree of success in the mitigation of the problem as they, like our work, allow the visualization and tracking of several parameters of the training process although they lack some critical parameters such as the learning rate, the works don't make persistent data gathering and can't be hosted on the cloud.

The objective of the work is to develop a platform for collecting internal parameters during the training phase of a neural network (for example hidden layer values, strength and direction of gradients, error). This collection of parameters should be periodical (frequency is defined as a parameter) and persistently stored for later analysis.

These parameters should be stored in a database optimized for the volume of data and able to deal with their temporal evolution (since training is typically a iterative refinement process), to facilitate the visualization of the evolution of metrics throughout the training.

The platform should be able to launch diverse training experiences as a service, container or virtual machine and manage its cycle (interactive training, collection of metrics and storage of same), with as much efficiency as possible, making the most of available hardware (graphic processing unit whenever possible). It should be possible to define a set of experiments to be performed and parameters to store. The platform launches, collects, and terminates the services necessary to complete the experience.

Finally, the platform must also support a viewer capable of selecting, organizing, and displaying the evolution of these parameters in a range of relevant graphics.

In addition to the introduction, this document has five more Chapters.

In chapter 2,the state of the art on systems that provide visualization and tooling needed for deep neural network training with gathering and visualization of several parameters and technology involved in the work is described. Chapter 3 presents the gathering process of requirements, the system's requirements, scenarios, personas, use cases and the terminology used. In chapter 4, the implementation of the working platform is described in different parts for each component. Chapter 5 indicates the results of the work and respective discussion and analysis.

Finally, chapter 6 makes an overview of the work done on this project, its main results, conclusions, and future work.

Chapter 2

State of the Art

2.1 Related Projects

The following section presents work whose domain and functionalities intersects with the objectives of this work. Several research pieces and projects regarding systems that that provide visualization and tooling needed for deep neural network training with gathering and visualization of several parameters predate the system developed in this project. An overview of those works will be presented in the following subsections.

2.1.1 TensorBoard

TensorBoard [17] is a tool for providing the measurements and visualizations needed during the machine learning experimentation.

The main functionalities are tracking experiment metrics like loss and accuracy,visualization of the model graph, view histograms of weights, biases, or other tensors as they change over time, project embeddings to a lower dimensional space, profiling TensorFlow programs and display images, text, and audio data

2.1.2 NeuroX: A Toolkit for Analyzing

Toolkit to facilitate the interpretation and understanding of neural network models with a focus on natural language processing.

The toolkit [2] provides several methods to identify salient neurons with respect to the model itself or an external task. A user can visualize selected neurons, ablate them to measure their effect on the model accuracy, and manipulate them to control the behavior of the model at the test time.

Such an analysis has a potential to serve as a springboard in various research directions, such as understanding the model, better architectural choices, model distillation and controlling data biases.

2.1.3 DeepConcolic: Testing and Debugging Deep Neural Networks

DeepConcolic [15] is a a deep neural network testing and debugging tool. *DeepConcolic* is able to detect errors with sufficient rigour so as to be applicable to the testing of deep neural networks in safety-related applications.

DeepConcolic implements a concolic testing technique for deep neural networks, and provides users with the functionality of investigating particular parts of a deep neural network.

2.1.4 DiffChaser: Detecting Disagreements for Deep Neural Networks

DiffChaser [20] is a automated black-box testing framework to detect untargeted/targeted disagreements between version variants of a deep neural network with the objective of discovering ways to preserve accuracy when going from an "original" version of a deep neural network to a compressed version optimized for mobile, for example.

To do so, it takes into account the structure and weights of the network.

2.1.5 NEUROSPF: A tool for the Symbolic Analysis of Neural Networks

NEUROSPF [18] is a tool for the symbolic analysis of neural networks.

Given a trained neural network model, the tool extracts the architecture and model parameters and translates them into a Java representation that is amenable for analysis using the Symbolic PathFinder symbolic execution tool.

Notably, *NEUROSPF* encodes specialized peer classes for parsing the model's parameters, thereby enabling efficient analysis. With *NEUROSPF* the user has the flexibility to specify either the inputs or the network internal parameters as symbolic, promoting the application of program analysis and testing approaches from software engineering to the field of machine learning.

NEUROSPF can be used for coverage-based testing and test generation, finding adversarial examples and also constraint-based repair of neural networks, thus improving the reliability of neural networks and of the applications that use them.

Some downsides of *NEUROSPF* are that it has no front-end, does not provide aggregated data and requires a model to finish training before being analyzed.

2.1.6 Seq2seq-Vis: A Visual Debugging Tool for Sequence-to-Sequence Models

Neural [14] sequence-to-sequence models have proven to be accurate and robust for many sequence prediction tasks, and have become the standard approach for automatic translation of text.

The models work with a five-stage blackbox pipeline that begins with encoding a source sequence to a vector space and then decoding out to a new target sequence. This process is now standard, but like many deep learning methods remains quite difficult to understand or debug.

Seq2seq-Vis presents a visual analysis tool that allows interaction and “what if”-style exploration of trained sequence-to-sequence models through each stage of the translation process. The aim is to identify which patterns have been learned, to detect model errors, and to probe the model with counterfactual scenario.

2.1.7 DeepLocalize: Fault Localization for Deep Neural Networks

DeepLocalize [19] is a debug tool that automatically determines whether the model is buggy or not, and identifies the root causes. It visualizes parameters such as weights and bias in order to identify problems in a certain layer.

To identify problems in a certain layer, it first enables dynamic analysis of deep learning applications: by converting it into an imperative representation and alternatively using a callback mechanism. Both mechanisms allows to insert probes that enable dynamic analysis over the traces produced by the deep neural network while it is being trained on the training data and then conduct dynamic analysis over the traces to identify the faulty layer that causes the error.

2.1.8 TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing

TensorFuzz [10] introduces testing techniques for neural networks that can discover errors occurring only for rare inputs. Specifically, coverage-guided fuzzing methods for neural networks.

In coverage-guided fuzzing, random mutations of inputs are guided by a coverage metric toward the goal of satisfying user-specified constraints. It describes how approximate nearest neighbor algorithms can provide this coverage metric for neural networks and then combine these methods with techniques for property-based testing. In property-based testing, one asserts properties that a function should satisfy and the system automatically generates tests exercising those properties.

2.2 Technology

This section presents the technologies applied in the development of the platform, by making a brief description of their purpose in the work, their main advantages, and disadvantages, as well as the main reasons leading to their choice.

2.2.1 TensorFlow and Keras

For the choice of a neural network training technology, we analysed *TensorFlow*, *Keras*, and *PyTorch* as valid machine learning frameworks for use in the system, taking in to account some aspects.

We chose to use *Keras* with *TensorFlow* because out of the frameworks analysed, *Keras* was the only one to have a native and simple JSON configuration of models.

Keras [12] is a deep learning API developed by Google and written in Python for implementing neural network.

Keras is embedded in *TensorFlow* and can be used to perform deep learning fast as it provides inbuilt modules for all neural network computations.

	# Modes	Documentation Quality	Team Experience	Configuration Support	Advisor Suggested
TensorFlow	103	Good	Few	Code Generation	Yes
Keras	103	Good	Few	Code Generation/JSON	Yes
PyTorch	103	Good	None	Code Generation	Yes

Table 2.1: NN Training Technologies Comparison

2.2.2 Grafana

For the visualization tool we discussed various technologies including *Kibana*, *Prometheus* and *Grafana*. *Kibana* was discarded due to a dependence on *Elasticsearch* and further research on *Prometheus* indicated that its purpose leans towards data monitoring and not visualization using graphics like intended. After that we considered graphic frameworks using *JavaScript*, but *Grafana* end up being the chosen technology because it has easy integration with the type of Database for the system, allow embedded interactive graphics and is less performance intensive compared with the alternative.

	Graph Types	Documentation Quality	Team Experience	Computational Point	Advisor Suggested
Grafana	9+ (Plugins)	Good	Few	Grafana Host + DB	Yes
JavaScript Graphs	9+ (Mix frameworks)	Varies	Few	Client + DB	No

Table 2.2: Visualization Technologies Comparison

Grafana [6] is an open source visualization and analytics software. It allows users to query, visualize, alert on, and explore metrics no matter where they are stored. It allows us to automatically query the database and generate interactive dashboards that support data filtering via query variables.

In addition to vanilla *Grafana*, We used the natel-plotly-panel plugin to enable some extra graph types.

2.2.3 TimescaleDB

The choice of a database took into account numerous factors and characteristics such as compression rate, scalability, efficiency, compatibility with *Grafana* and as such we considered several different database technologies.

Discarding relational databases for not adapting well to the type of data that will be stored, it was decided to use *TimescaleDB*, as *Cassandra* does not have a good compatibility with *Grafana* and although *InfluxDB* has similar characteristics to *TimescaleDB*, the last scales much better for high numbers of users, which we considered preferable accounting for future work.

TimescaleDB [11] is an open-source relational database for time-series data. It was created to make SQL scalable against time-series data.

	Type	Compression	Speed	Scalability	Grafana Compatibility	Documentation Quality	Advisor Suggested
MySQL	Relational	Yes (Mediocre)	Medium	Yes	Yes	Good	Against
Cassandra	Column	Yes	High	Yes	No	Good	Yes
PostgreSQL	Relational	Yes (Mediocre)	Medium	3rd Party	Yes	Good	Against
Apache Kylin	OLAP Cube	Yes (Mediocre)	High	Yes	No	Good	Unknown
TimescaleDB	Time Series	Yes	High	Yes	Yes	Satisfactory	Against
InfluxDB	Time Series	Yes (Unknown)	High	Yes	Yes	Satisfactory	Unknown

Table 2.3: Database Technologies Comparison

It provides an automatic partitioning across key-values and dates, provides native SQL support and is correspondingly easy to use like a traditional relational database, yet scales in ways previously reserved for NoSQL databases.

2.2.4 Django

For the Web Technology we took in to consideration *ReactJS*, *VueJS* and *Django*.

The main advantage of using a frontend framework, like *React* or *Vue*, is the separation between the frontend and the backend, not needing any changes to the backend for frontend modifications but because using a separate frontend would complicate other parts of the system we find that this separation would add unnecessary work.

With a built-in user management system, easy to use object-relational mapping, good scalability, the ability to expand its functionalities using new plugins and fast development for server and front end, *Django* was our choice.

	Language	Documentation Quality	Team Experience	Advisor Suggested
ReactJS	JavaScript	Good	Few	No
VueJS	JavaScript	Good	None	No
Django Templates	Python	Good	Few	Yes

Table 2.4: Web Technologies Comparison

Django [9] is a free, open source, high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development.

The main goal of the *Django* framework is to allow developers to focus on components of the application that are new instead of spending time on already developed components.

2.2.5 Flask

To reduce the waiting time while parsing Neural Network data or deploying a Neural Network we need a way to perform certain tasks asynchronously.

	Language	Background tasks (Async processing)	Documentation Quality	Speed	REST Support	Team Experience	Advisor Suggested	Problems
Django	Python	With Plugin	Good	Good	Included	Yes	Yes	None foreseen
Flask	Python	With Celery	Good	Very good	Included	Yes	Yes	None foreseen
NodeJS	JavaScript	Yes	Good	Good	Included	None	No	Hard to debug
Sanic	Python	Via Async/Tasks (Queues have no documentation)	Very incomplete	Good	Good	None	No	Young, incomplete documentation

Table 2.5: NN Data Parser Technologies Comparison

After an analysis of several solutions we decided that the best solution was to delegate tasks to additional components considered light and simple, using *Flask* + *Celery*, which helps maximize system responsiveness while doing background tasks based on request content asynchronously after responding to requests.

Flask [5] is a web framework/Python module that allows for easy development of web applications. It [7] has a small and easy-to-extend core: it's a micro framework that does not include an object relational manager or such features.

2.2.6 Docker

For the Virtualization/Containerization Technology to deploy our main components and neural networks we discussed *Qemu*, *Docker*, and *Kubernetes*.

	Type	Lightweight	Scalability	Documentation Quality	Team Experience	Advisor Suggested
Docker	Virtualization (Containers)	Yes	No	Good	Good	Yes
Qemu	Machine emulator	No	No	Good	None	No
Kubernetes	Virtualization (Containers)	Yes	Yes	Good	Few	No

Table 2.6: Virtualization/Containerization Technologies Comparison

As a virtual machine service *Qemu* was deemed too complex for the task at hand.

The remaining 2 technologies would work well together with *Kubernetes* being an "orchestration" software that would allow us to deploy containers on multiple devices, similarly to Docker's swarm mode, which has fallen out of favor. Considering the scalability of our system, *Kubernetes* was considered an overly extravagant technology to make use of.

Docker was the most convenient way technology to deploy our system, including dynamically instancing our machine learning environments, thanks to the compose features and the docker UNIX socket daemon.

Docker [13] is an open platform for developing, shipping, and running applications by using containers.

Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and deploy it as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

By taking advantage of *Docker*'s methodologies for shipping, testing, and deploying code quickly, we can significantly reduce the delay between writing code and running it in production.

Chapter 3

Conceptual Modelling

3.1 Design Procedure

To have a better understanding of the problem faced, there was a need of previous research and analysis to understand the functionality, an application should aim for in order to be useful and indeed accomplish the work objectives.

For that, we gathered requirements by initially discussing what the aim of the project should be, what users are targeted and what should constitute the system, creating personas and scenarios. Following this came researching the state of the art in debugging/analysis tools for Neural Net models by analysing various research papers, projects and technologies which led to a better understanding of where the project would be framed in the field and what functionality to aim for.

Afterwards, we had various meetings with our advisors to validate the technologies/architectural choices and discuss details related to the functioning and design of components of the work.

As of this point containerized Neural Networks will be referred to as “simulation”.

3.2 Requirements

3.2.1 Functional Requirements

- The user must be able to launch simulations
- The user must be able to analyze simulation data
- The user must be able to customize their simulation based on a file input
- The user should be able to download simulation data
- The system must recognize a simulation life cycle
- The system must be efficient
- The system must be consistent

3.2.2 Non-Functional Requirements

- System must work in the most recent versions of most web browsers
- The system must be efficient
- The system must be easy to use
- The system must guarantee data security
- The system must guarantee data integrity and consistency

3.3 Actors

The target user for the system is a person, whose knowledge of neural networks and their training process is fairly good and has good computer literacy, as such the interface was designed taking into consideration that knowledge and literacy ,while keeping project objectives in mind.

The main actors are described in the following list:

- **Client:** Represents the client, having access to all the features available in the system platform, in other words, this allows him to create a simulation in the system and analyse the obtained results of the simulation.
- **Admin:** Represents the administrators of the platform with the responsibility to manage the clients using the platform by altering the account privileges, check client resource use, among other.

3.3.1 Use Cases

Figure 3.1 presents the main use case models of the system.

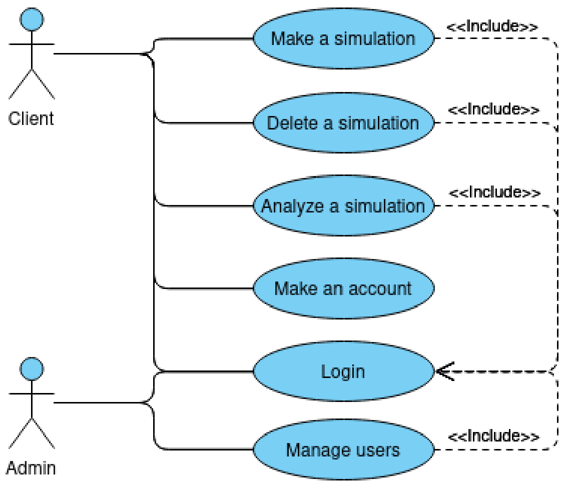


Figure 3.1: Use Case Model

As it can be seen, the Client is able to create a a new simulation, delete and analyse them, and the Admin can Manage the Client Accounts.

3.3.1.1 Client wants to make a new Simulation

In Figure 3.2 is presented the steps taken by a client to generate a new simulation.

First the client accesses the system and makes login with the credentials. The client then clicks in the new button and configures the simulation. After that client clicks the create button and the simulation is created or a message of error uppers indicating an error.

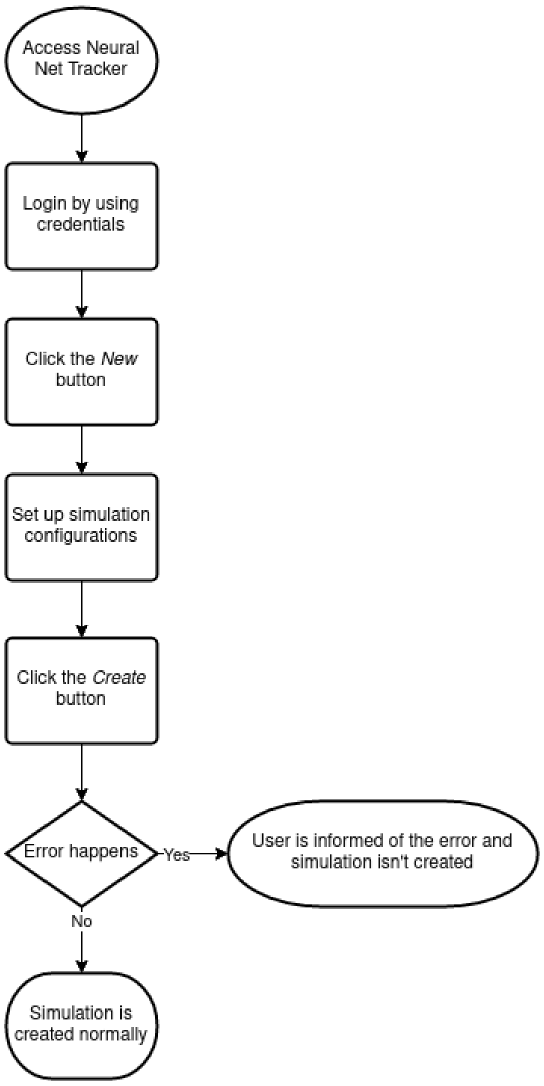

Access Neural Net Tracker
Login by using credentials
Click the *New* button
Set up simulation configurations
Click the *Create* button
Error happens
Yes
User is informed of the error and simulation isn't created
No
Simulation is created normally

Figure 3.2: Workflow - Client Wants to Make new Simulation

3.3.1.2 Client wants to analyse a Simulation

In Figure 3.3 is presented the steps taken by a client to analyse a simulation.

First the client accesses the system and makes login with the credentials. The client then chooses a simulation to analyze in the simulation list. Then depending on the status of the simulation, the client can make a complete or incomplete analyse of the simulation information and data.

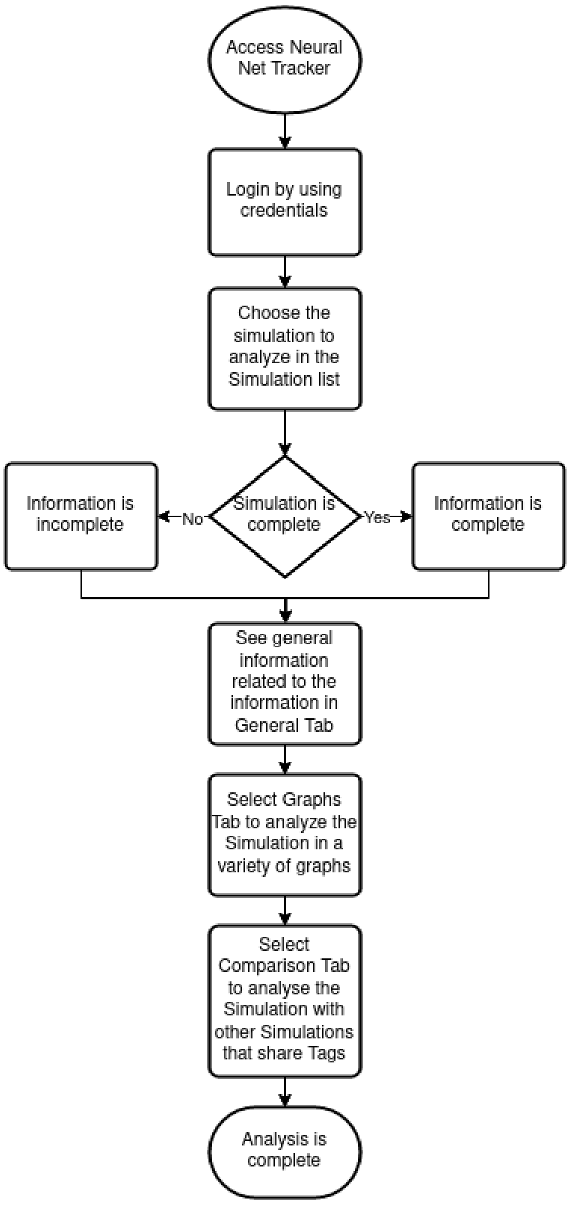

Access Neural Net Tracker
Login by using credentials
Choose the simulation to analyze in the Simulation list
Simulation is complete
No
Yes
Information is incomplete
Information is complete
See general information related to the information in General Tab
Select Graphs Tab to analyze the Simulation in a variety of graphs
Select Comparison Tab to analyse the Simulation with other Simulations that share Tags
Analysis is complete

Figure 3.3: Workflow - Client Wants to Analyse a Simulation

3.3.1.3 Client wants to Resume/Stop a simulation

In Figure 3.4 is presented the steps taken by a client to Resume/Stop a simulation.

First the client accesses the system and makes login with the credentials. The client then chooses a simulation to resume/stop in the simulation list. After that, depending on the status of the simulation, the client can resume or pause the simulation, then receiving a notification that in case of error receives indicates a error but in case of success confirms the operation.

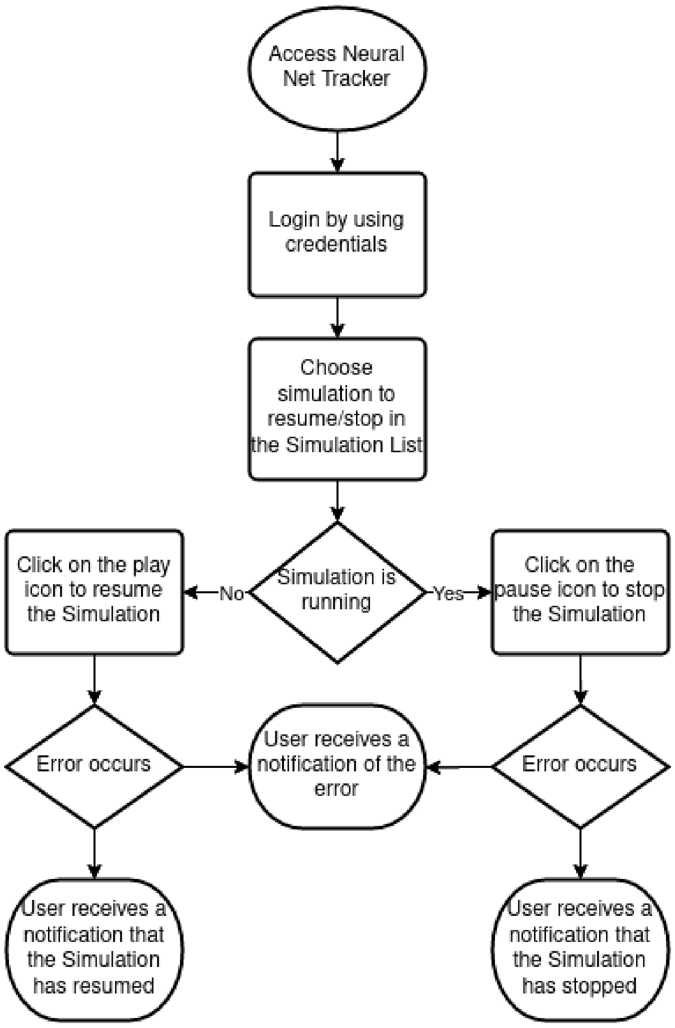

Access Neural Net Tracker
Login by using credentials
Choose simulation to resume/stop in the Simulation List
Simulation is running
No
Yes
Click on the play icon to resume the Simulation
Click on the pause icon to stop the Simulation
Error occurs
User receives a notification of the error
Error occurs
User receives a notification that the Simulation has resumed
User receives a notification that the Simulation has stopped

Figure 3.4: Workflow: Client wants to Resume/Stop a simulation

3.4 Personas

To help us understand the different client types needs, experiences, behaviours and goals, we created the following personas.

3.4.1 Persona 1 - Rui

Figure 3.5: Rui

Characteristics:

- 36 years old.
- Director of an organization dedicated to the identification of species of animals, that recently started using *Machine Learning*.
- Curious and dedicated person who is very fond of nature and his work.
- One of the main actors in the field responsible for the use of *Machine Learning* in the identification of species of animals.

Problem:

- Although having some correct results, most of them are still inacurate which makes the use of *Machine Learning* inefficient and possibly even misleading.
- Despite some attempts to improve the *Neural Network*, it continues to have poor results, if not worse.

Motivation:

- Wants a way to analyse the data from the training of the *Neural Network* in order to conclude if progress has been made in achieving the ultimate goal of a more accurate species identification.

3.4.2 Persona 2 - Ana

Figure 3.6: Ana

Characteristics:

- 32 years old.
- Manager responsible for administering and monitoring the training of deep neural network models of an artificial intelligence company located in Aveiro.
- Practices swimming and likes to run daily.
- Despite her professional life requiring organization and schedules, in her personal life she is a relaxed person who like to live in the moment.

Problem:

- Has had some difficulties managing and analyzing the training process of *Neural Networks* as these are normally time consuming processes with difficult interpretation of data which are also complicated to interrupt or resume.

Motivation:

- Wants an efficient and simple way to manage the different training processes of deep neural networks with the possibility of stopping or resuming them.

3.5 Scenarios

In the following subsections is presented the Scenarios prepared for each persona presented in the previous section.

3.5.1 Rui intends to create a simulation and analyse the obtained parameters

3.5.1.1 Create an account

Figure 3.7 presents the steps taken by Rui to create an account.

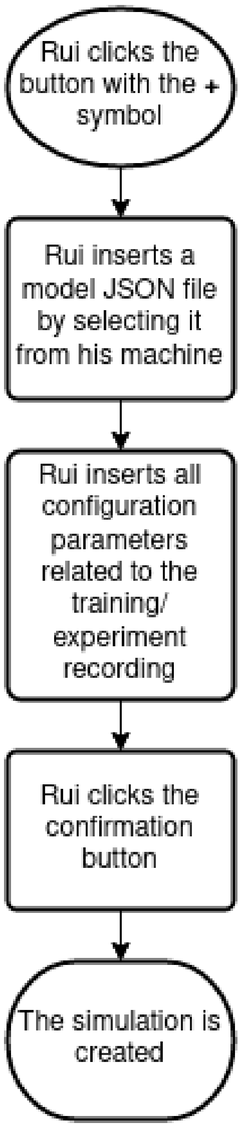

Rui clicks the button with the + symbol
Rui inserts a model JSON file by selecting it from his machine
Rui inserts all configuration parameters related to the training/ experiment recording
Rui clicks the confirmation button
The simulation is created

Figure 3.7: Workflow - Rui Creates an Account

3.5.1.2 Customize and deploy a simulation

Figure 3.8 presents the steps taken by Rui to sustomize and deploy a simulation.

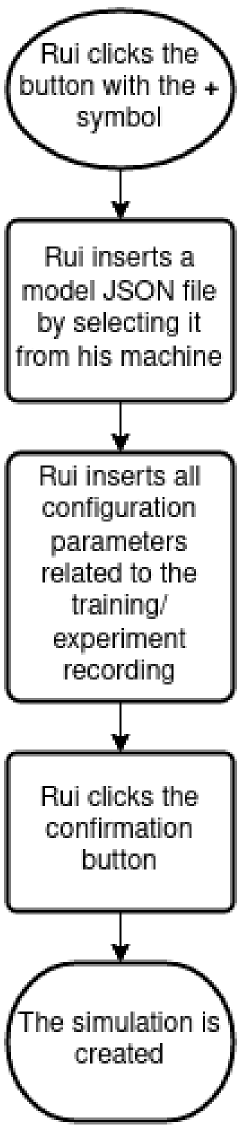

Rui clicks the button with the + symbol
Rui inserts a model JSON file by selecting it from his machine
Rui inserts all configuration parameters related to the training/ experiment recording
Rui clicks the confirmation button
The simulation is created

Figure 3.8: Workflow - Rui Customize and deploy a simulation

3.5.1.3 Analyse the obtained parameters

Figure 3.9 presents the steps taken by Rui to analyse the obtained parameters of the simulation.

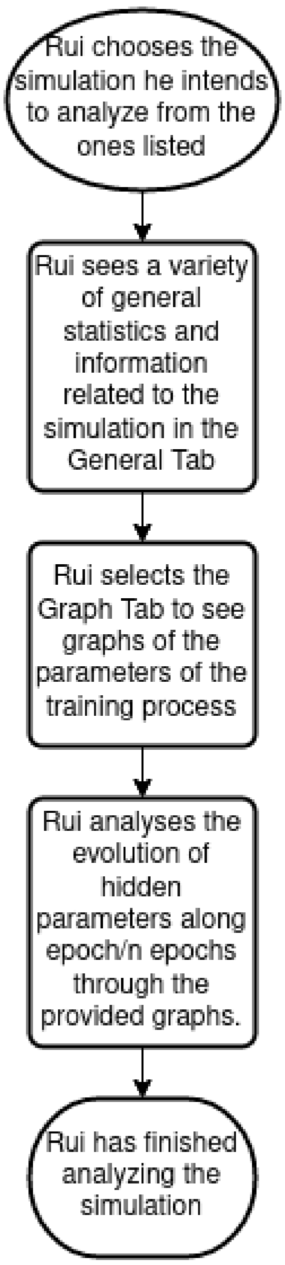

Rui chooses the simulation he intends to analyze from the ones listed
Rui sees a variety of general statistics and information related to the simulation in the General Tab
Rui selects the Graph Tab to see graphs of the parameters of the training process
Rui analyses the evolution of hidden parameters along epoch/n epochs through the provided graphs.
Rui has finished analyzing the simulation

Figure 3.9: Workflow - Rui Analyses the obtained parameters

3.5.2 Ana checks her simulation list and controls execution

3.5.2.1 Log In in the System

Figure 3.10 presents the steps taken by Ana to log In in the system.

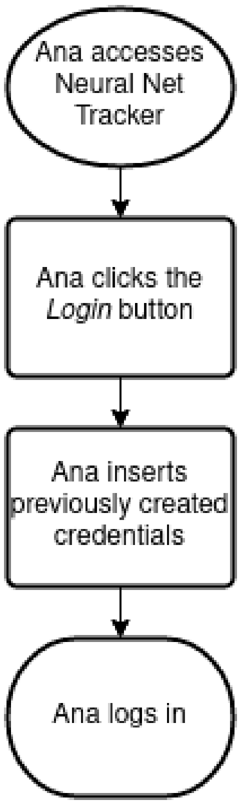

Ana accesses Neural Net Tracker
Ana clicks the *Login* button
Ana inserts previously created credentials
Ana logs in

Figure 3.10: Workflow - Ana Log In in the System

3.5.2.2 Stops and resumes simulations

Figure 3.11 presents the steps taken by Ana to stop and resume simulations.

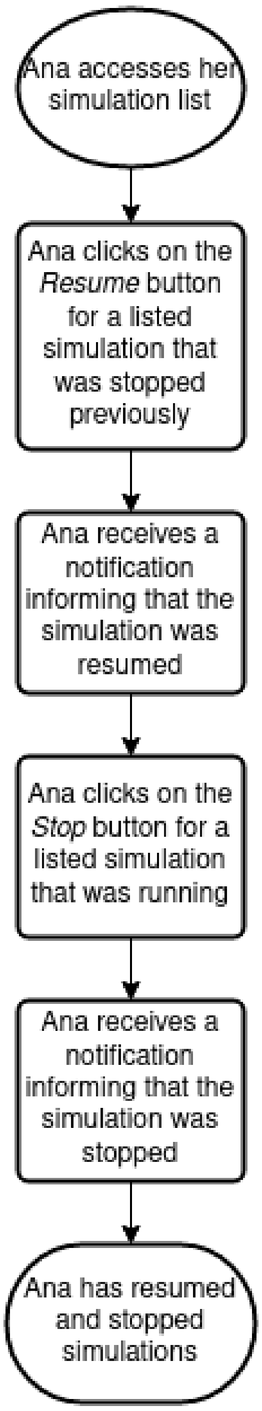

Ana accesses her simulation list
Ana clicks on the *Resume* button for a listed simulation that was stopped previously
Ana receives a notification informing that the simulation was resumed
Ana clicks on the *Stop* button for a listed simulation that was running
Ana receives a notification informing that the simulation was stopped
Ana has resumed and stopped simulations

Figure 3.11: Workflow - Ana stops and resumes simulations

3.6 Domain Model

The domain model presented on Figure 3.12 describes the various entities, roles and relationships of the system.

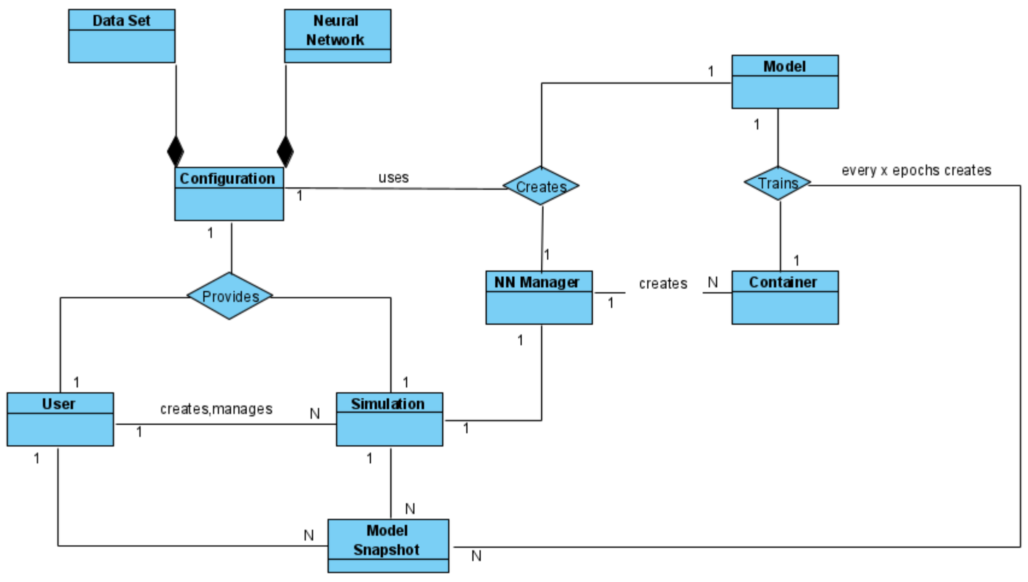


Figure 3.12: Model Domain Diagram

Chapter 4

Implementation

This section provides an overview on the implementation of the system. It describes the development of the different components and their interconnection.

Unfortunately, the machine used in the development of the system did not have a Graphics Processing Unit and as such we did not make use of said hardware component.

All source code and configuration files can be found at our project repository.[1]

4.1 Component Choice

This section provides a discussion and decision on the Technologies to use in each component.

4.1.1 Neural Network Training Technology

For the neural network training technology we use *Keras with TensorFlow*.

As mentioned in subsection 2.2.1 from chapter 2, we analysed *TensorFlow*, *Keras* and *PyTorch* as valid machine learning frameworks for use in the system but we chose *Keras* with *TensorFlow* because Keras was the only one to have a native and simple JSON configuration of models.

4.1.2 Visualization Technologies

For the visualization technology we use *Grafana*

As mentioned in subsection 2.2.2 from chapter 2, we discussed various technologies including *Kibana*, *Prometheus* and *Grafana*.

Grafana end up being the chosen technology because it has easy integration with the type of Database for the system, allow embedded interactive graphics and is less performance intensive compared with the alternatives.

[1]https://github.com/buckaroo69/PI18

4.1.3 Database Technology

For the database technology we use *TimescaleDB*

As mentioned in subsection 2.2.3 from chapter 2,the choice of a database took into account numerous factors and characteristics such as compression rate, scalability, efficiency, compatibility with *Grafana*.

The dedision went to *TimescaleDB*, as *Cassandra* does not have a good compatibility with *Grafana* and although *InfluxDB* has similar characteristics to *TimescaleDB*, the last scales much better for high numbers of users, which is considered preferable accounting for future work.

4.1.4 Web Technology

For the web technology we use *Django*

As mentioned in subsection 2.2.4 from chapter 2, we took in to consideration *ReactJS*, *VueJS* and *Django*.

Because using a separate frontend would complicate other parts of the system we find that this separation would add unnecessary work, so with a built-in user management system, easy to use object-relational mapping, good scalability, the ability to expand its functionalities using new plugins and fast development for server and front end, we chose *Django*.

4.1.5 Neural Network Data Parser Technology

For the neural network data parser we use *Flask*

As mentioned in subsection 2.2.5 from chapter 2, to reduce the waiting time while parsing neural network data or deploying a neural network the system needs a way to perform certain tasks asynchronously.

After an analysis, the best solution is to delegate tasks to additional components considered light and simple, using *Flask* + *Celery*, which helps maximize system responsiveness while doing background tasks based on request content asynchronously after responding to requests.

4.1.6 Virtualization/Containerization Technology

For the virtualization/containerization technology we use *Docker*

As mentioned in subsection 2.2.6 from chapter 2, we discussed *Qemu*, *Docker*, and *Kubernetes*, for the virtualization/containerization technology to deploy our main components and neural networks.

Docker was the most convenient way technology to deploy our system, including dynamically instancing our machine learning environments, thanks to the compose features and the docker UNIX socket daemon.

4.2 Architecture

The vast majority of the system is implemented in Python to maximize code readability and ease of implementation.

The back-end server was separated into the following components:

- An asynchronous deploying component
- An asynchronous parser that receives simulation updates

Components communicate via REST interfaces which will be detailed in each component's documentation.

For persistence we use *TimescaleDB*, a time-series database that features powerful compression and very fast query times.*TimescaleDB* is a *Postgres* extension that still supports relational features such as foreign keys, so said relational features were taken into account when constructing our data schema.

For the back-end server, responsible for serving our front-end, we used the *Django Web Framework* as it provides ready-to-use components for essential features like templating and security.

And finally, for both the deploying components and the parser, we used the *Flask* micro-framework for handling REST with the *Celery* library in order to distribute data processing to worker processes enabling us to have non-blocking asynchronous data processing in Python.

The final architecture is presented in Figure 4.1

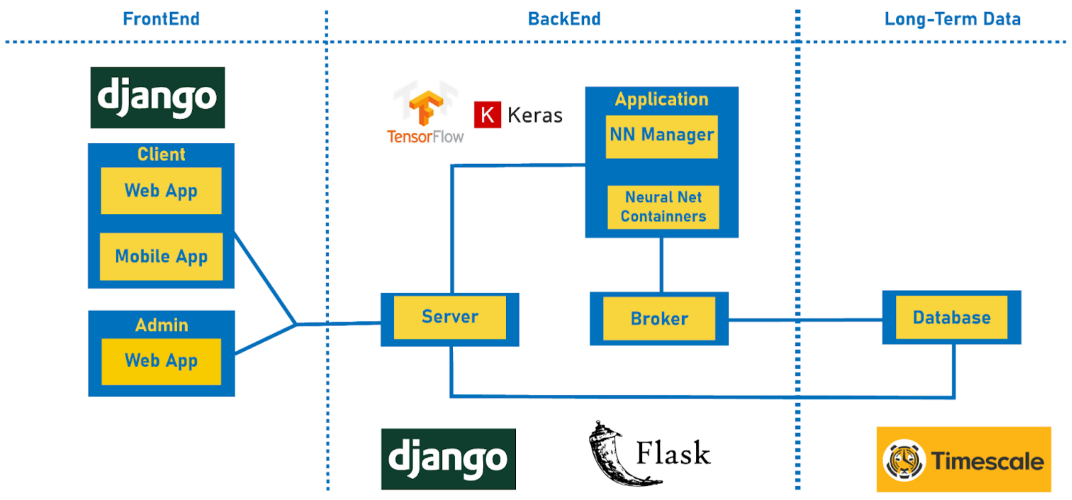


Figure 4.1: Architecture Diagram

4.3 Database

The system has several different tables to store neural network related information:

- **simulations**: A table for overall Simulation Information

- **epoch_values**: A table for error and accuracy on every logging interval
- **weights**: A table that stores Weights, per logging interval and layer
- **extra_metrics**: A table that stores optional metric information
- **Tags**: Allows for the association of simulations
- Django related tables to support user accounts, sessions and other features

We automatically create the database by using Docker volumes to provide SQL files to our database container at */docker-entrypoint-initdb.d/*

The final Data Schema is presented in Figure 4.2

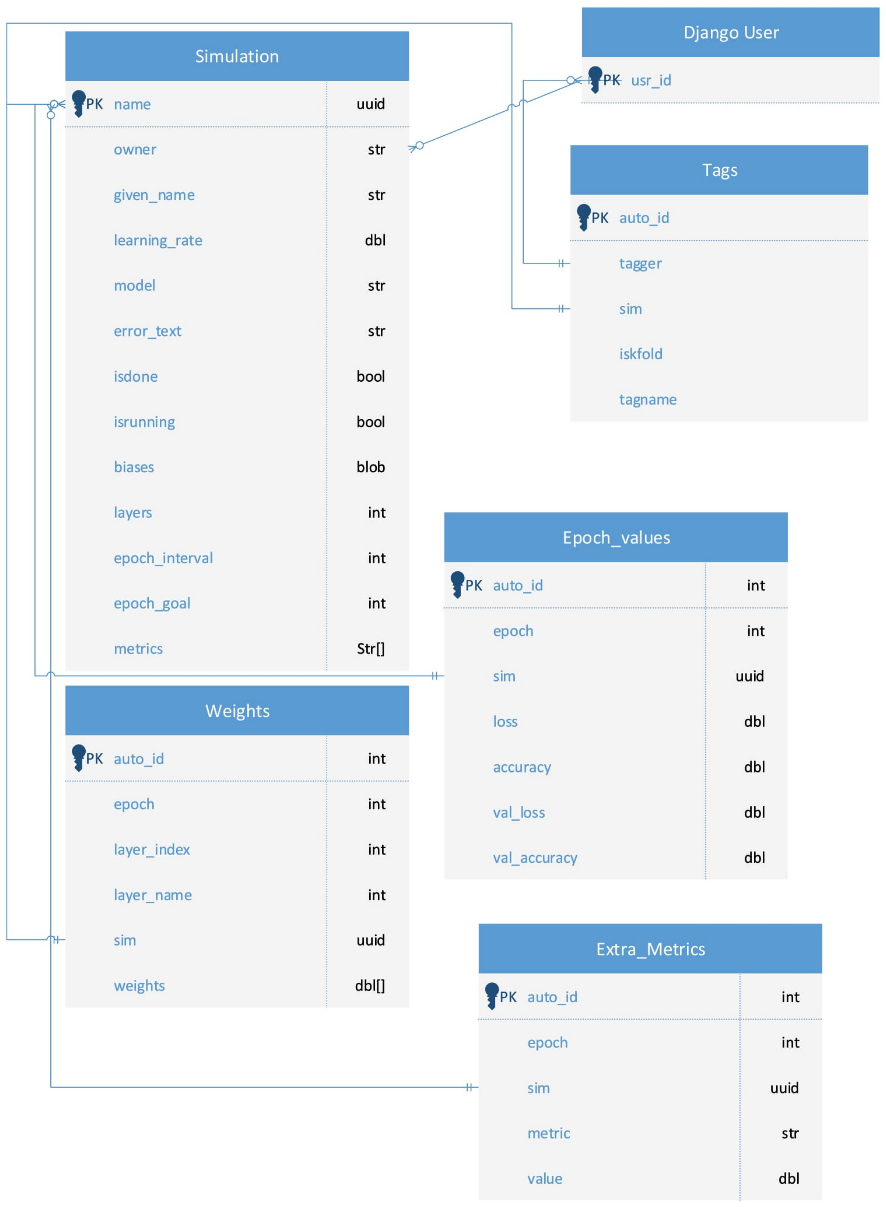


Figure 4.2: Data Schema

4.4 Django Front-End Server

Our system's users are provided with a Django-based user interface, available at port 8000 by default.

The Django server provides authentication features, provides a visual interface for the deployment of simulations, facilitates the management of the user's simulations, and allows analysis of the simulation data, including comparison and aggregation, by using the embedded graphs or by downloading the data and using a third-party system.

In the administration side, the Django server allows administrators to manage users, check their information and stats, and manage any user's simulations.

4.4.1 Web Pages

The home page of our system (Figure 4.3) is the only part of the system available for both users and non-users. It has everything related to the development of the system: a small explanation of its features, documentation, the repository with the source code, the demo and promotional videos, and who was involved in its making.

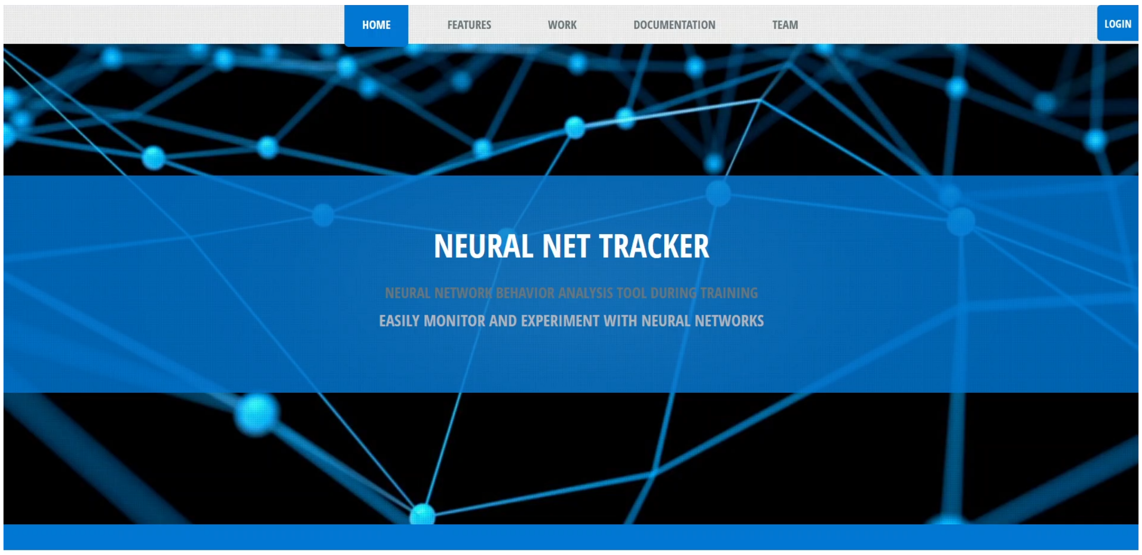

HOME
FEATURES
WORK
DOCUMENTATION
TEAM
LOGIN
NEURAL NET TRACKER
NEURAL NETWORK BEHAVIOR ANALYSIS TOOL DURING TRAINING
EASILY MONITOR AND EXPERIMENT WITH NEURAL NETWORKS

Figure 4.3: Home Page

The login and registration used in the system are the default *Django* ones, which only use a username and password.

The first page seen by an authenticated user is the *Simulation list* page (Figure 4.4), where the users can see all their simulations and use the inputs above the list to search, filter, and sort them.

In this page, each simulation shown contains their name, tags, status, and buttons to pause, start or delete them.

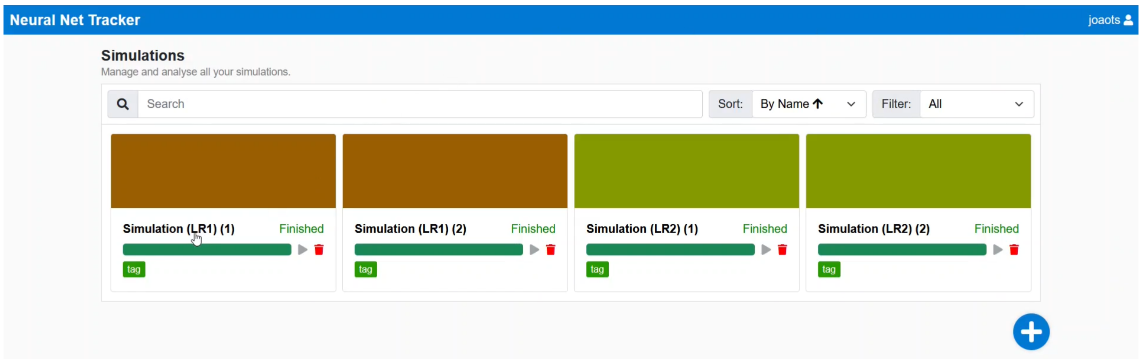


Figure 4.4: Simulation List Page

The most complex page of the system is the *Simulation creation* page (Figure 4.5), which can be accessed by clicking the floating action button in the *Simulation list* page. This page is used to insert all the data necessary to create and start a simulation.

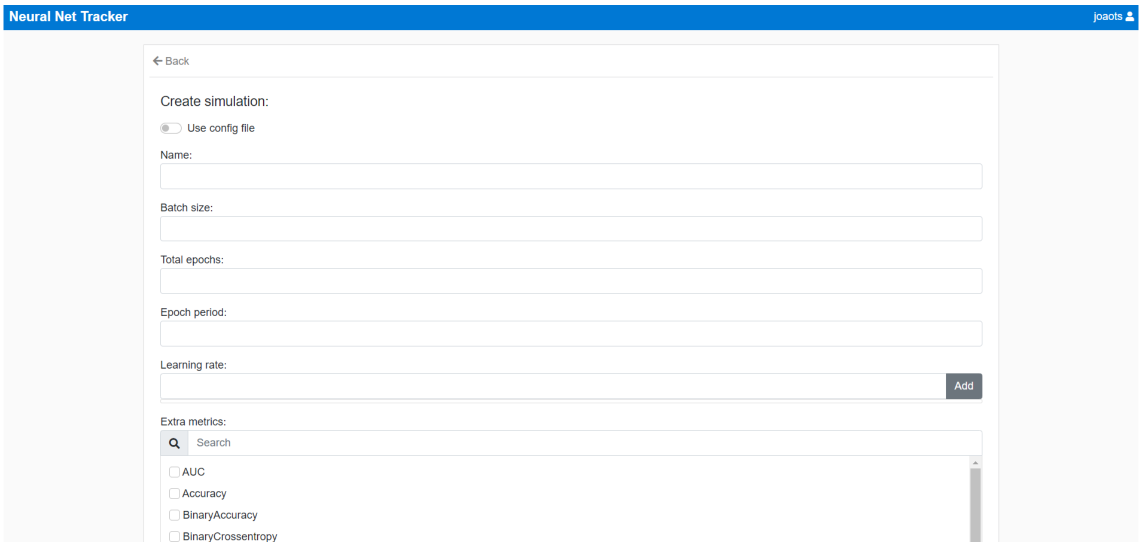
To make a simulation a model, training dataset, test dataset, validation dataset and configurations are needed. The datasets can either be files or URLs from where they can be downloaded and the configuration can be made in two ways, by manual input of form fields or by using a configuration file.

The configuration file uses the JSON format, as described in appendix A. The datasets files need to be in certain formats in order to work in our system; the explanation and examples of the content of the allowed formats can be found in appendix C.

Note: For all datasets, non-numeric values are not accepted, any non-numeric parameters must be converted to numeric values.

When more than one *optimizer*, *loss function* and/or *learning rate* is selected, our system will create enough simulations so that all possible combinations of the three are instantiated.

All of the properties used in the configuration file have their corresponding input field in the manual input.


Neural Net Tracker joaots

Back

Create simulation:

Use config file

Name:

Batch size:

Total epochs:

Epoch period:

Learning rate: Add

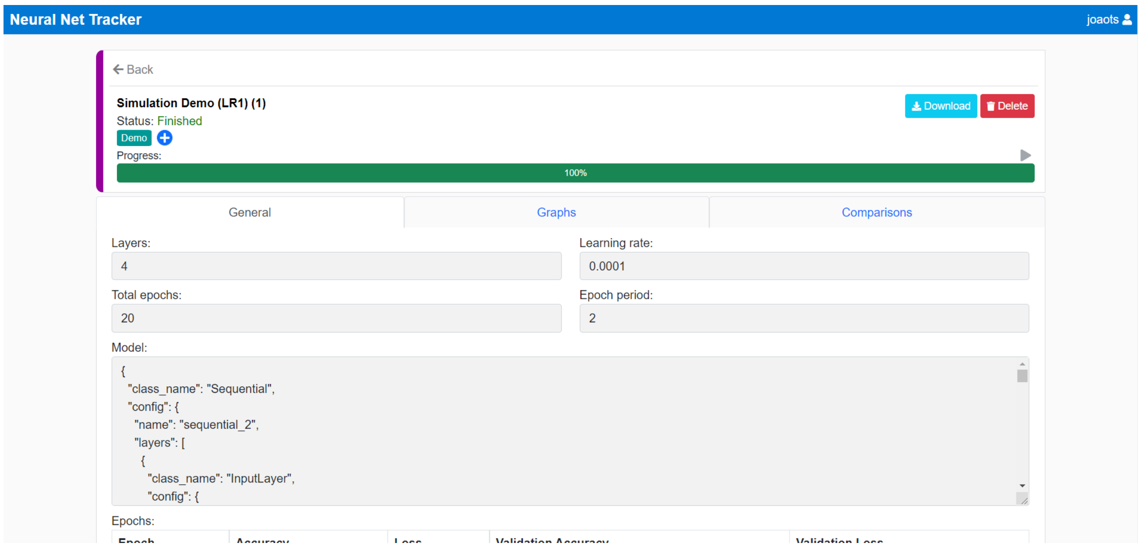
Extra metrics:

Search

- AUC
- Accuracy
- BinaryAccuracy
- BinaryCrossentropy

Figure 4.5: Simulation Creation Page

The last page the regular user can access is the *Simulation info* (Figure 4.6), where they can see everything about a simulation. The top section of this page contains much of the same information as the *Simulation list* page, with the addition of a download button to download the simulation's data to be used outside of our service and the buttons to add and remove tags.


Neural Net Tracker joaots

Back

Simulation Demo (LR1) (1) Download Delete

Status: Finished

Demo

Progress:

100%

General | Graphs | Comparisons

Layers: 4

Learning rate: 0.0001

Total epochs: 20

Epoch period: 2

Model:

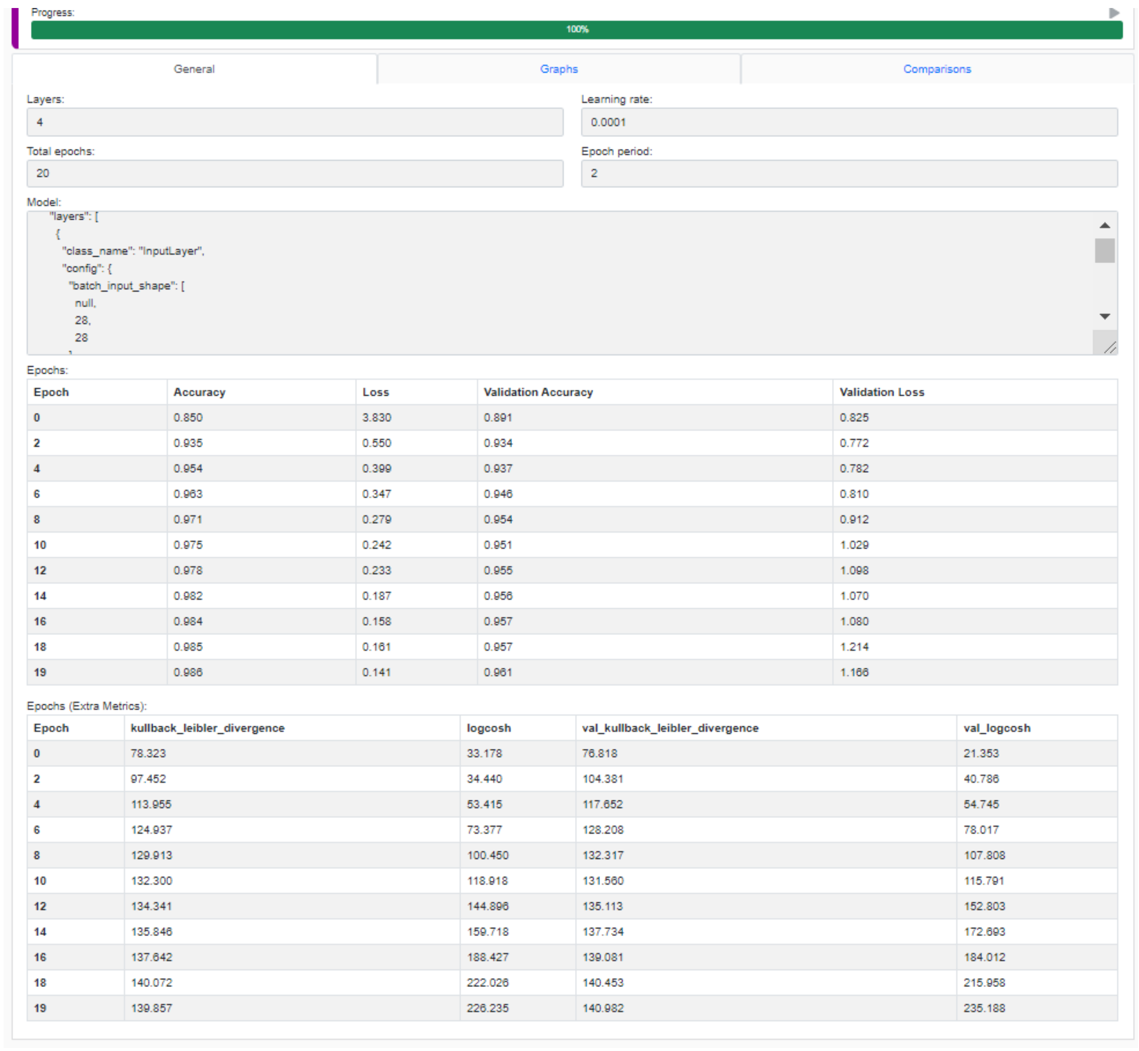
```
{
  "class_name": "Sequential",
  "config": {
    "name": "sequential_2",
    "layers": [
      {
        "class_name": "InputLayer",
        "config": {
```

Epochs:

Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss

Figure 4.6: Simulation Information Page

The *General* tab of the *Simulation info* page (Figure 4.7), contains some of the values inserted in the creation of this simulation, such as *Total Epochs* and the *Model*, as well as two tables containing the values gotten on each epoch, the first table containing *Accuracy* and *Loss* which are common to all simulations and the second table containing the extra metrics selected in the creation of the simulation.



Layers: 4

Learning rate: 0.0001

Total epochs: 20

Epoch period: 2

Model:

```
    "layers": [
     {
      "class_name": "InputLayer",
      "config": {
       "batch_input_shape": [
        null,
        28,
        28
```

Epochs:

Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
0	0.850	3.830	0.891	0.825
2	0.935	0.550	0.934	0.772
4	0.954	0.399	0.937	0.782
6	0.963	0.347	0.946	0.810
8	0.971	0.279	0.954	0.912
10	0.975	0.242	0.951	1.029
12	0.978	0.233	0.955	1.098
14	0.982	0.187	0.956	1.070
16	0.984	0.158	0.957	1.080
18	0.985	0.161	0.957	1.214
19	0.986	0.141	0.961	1.166

Epochs (Extra Metrics):

Epoch	kullback_leibler_divergence	logcosh	val_kullback_leibler_divergence	val_logcosh
0	78.323	33.178	76.818	21.353
2	97.452	34.440	104.381	40.786
4	113.955	53.415	117.652	54.745
6	124.937	73.377	128.208	78.017
8	129.913	100.450	132.317	107.808
10	132.300	118.918	131.560	115.791
12	134.341	144.896	135.113	152.803
14	135.846	159.718	137.734	172.693
16	137.642	188.427	139.081	184.012
18	140.072	222.026	140.453	215.958
19	139.857	226.235	140.982	235.188

Figure 4.7: Simulation Information - General Tab

The *Graphs* tab of the *Simulation info* page(Figure 4.8), presents all the graphs which show the simulation's data in a visual way, with Line graphs for the *Accuracy*, *Loss*, and extra metrics, and a 3D Scatter Plot for the simulation's weights. Each of these graphs has the necessary inputs to manipulate the data shown.

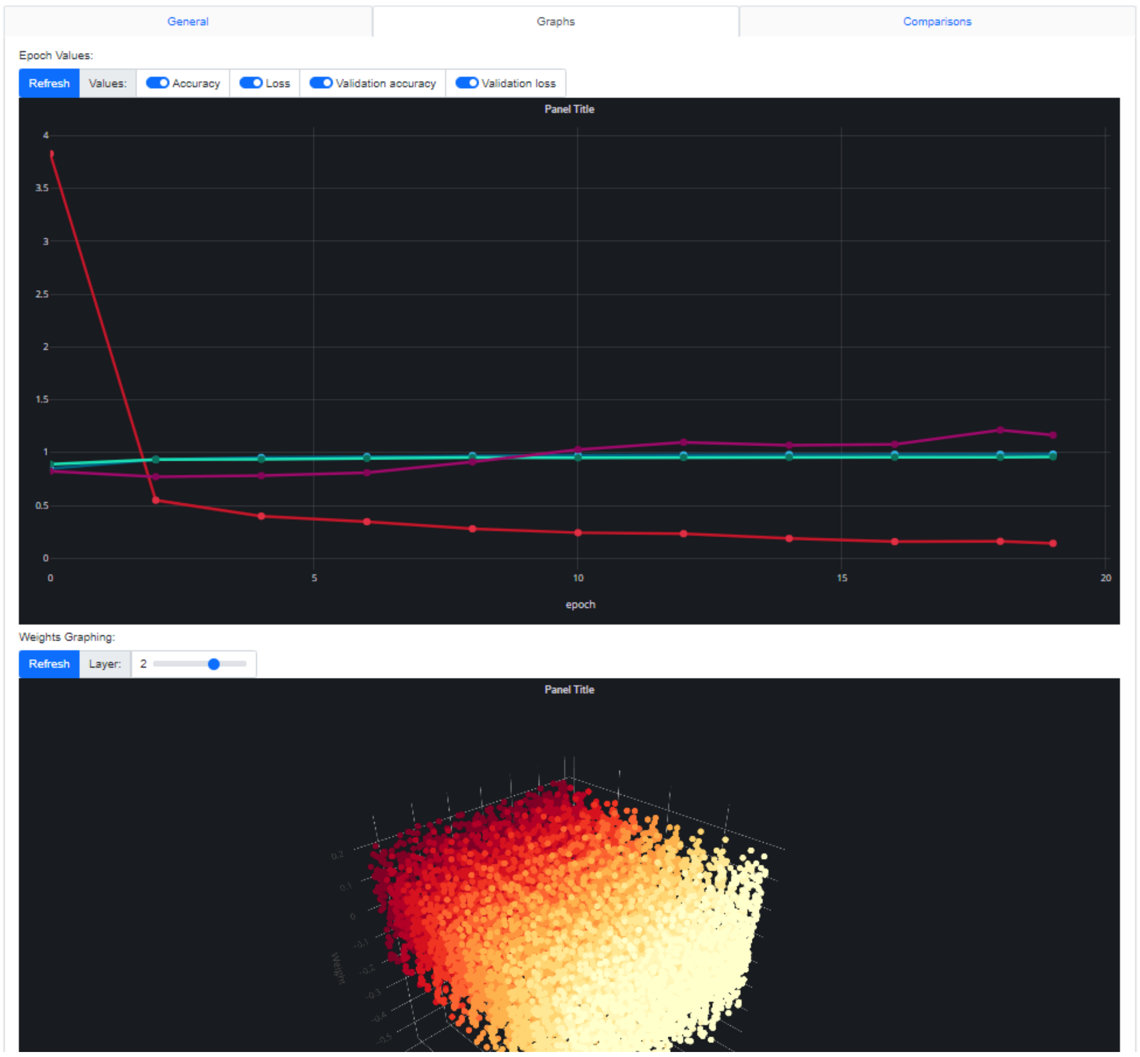
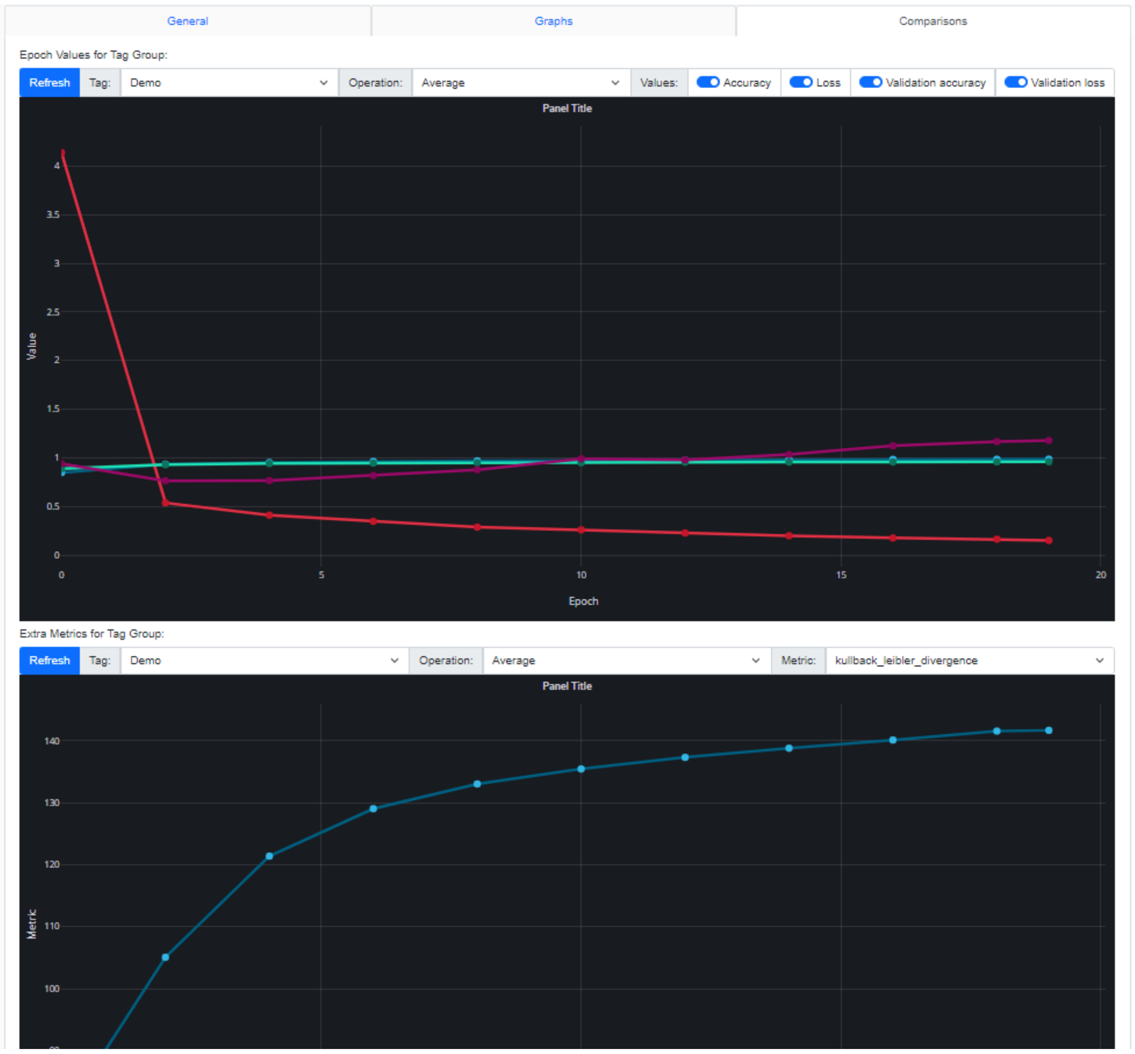


Figure 4.8: Simulation Information - Graphs Tab

The last tab of the *Simulation info* page, the *Comparisons* tab (Figure 4.9), has all the graphs to compare this simulation's data with another. Two of the graphs are used to aggregate and show the maximum, minimum, or average of the *Accuracy*, *Loss*, and extra metrics in all simulations with a specific tag, which the main simulation also has to have. The other two graphs subtract the *Accuracy*, *Loss*, and extra metrics with any of the user's other simulations. All the graphs in this tab are Line graphs and have the necessary inputs to manipulate the data shown.



Epoch Values for Tag Group:

Tag: Demo
Operation: Average
Values: Accuracy, Loss, Validation accuracy, Validation loss

Extra Metrics for Tag Group:

Tag: Demo
Operation: Average
Metric: kullback_leibler_divergence

Figure 4.9: Simulation Information - Comparison Tab

4.4.2 Admin Web Pages

Our system has two administration web pages which are not accessible by regular users, the *User list* page (Figure 4.10) and the *User info* page (Figure 4.11).

The *User list* page contains a list of all the accounts that ever logged in to our system, except ones that were deleted or the account of the administrator viewing this page. Each item of this list has the account's name, a button to grant or revoke admin privileges, a button to disable or enable the account (which makes the user not able or able to login into it), and a button to delete the account. Above the list there are inputs to search, filter, and sort accounts.

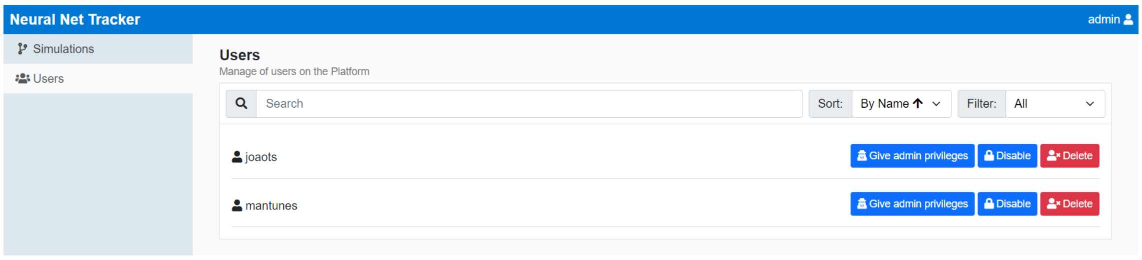
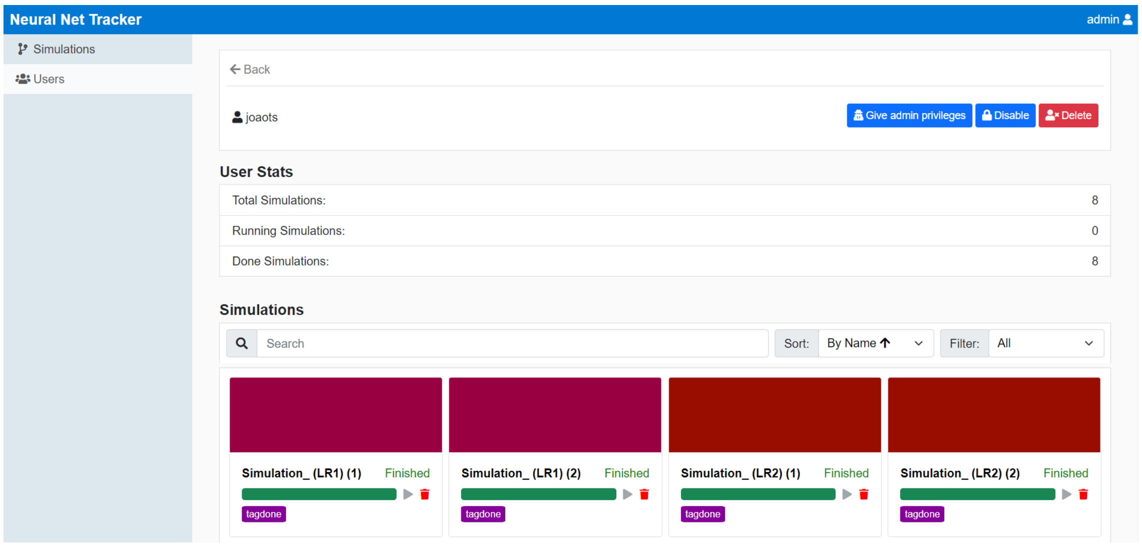


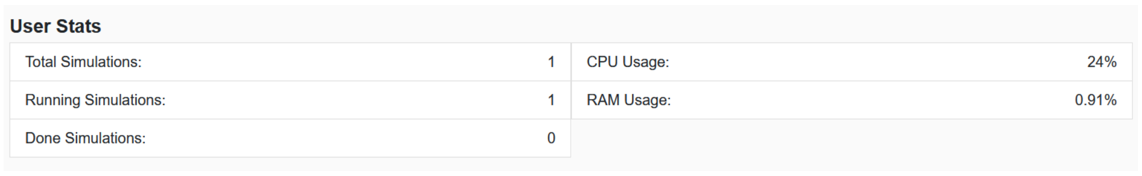
Figure 4.10: User List

The *User info* page is divided in three sections, the first one has the same information and buttons as each item in the previous page's list, the second one contains extra infomation about the account, such as how many simulations it has, how many simulations are running, and how many resources this account's simulations are using (Figure 4.12) , the third and last section contains what this account's user would see in its *Simulation list* page.



Total Simulations:	8
Running Simulations:	0
Done Simulations:	8

Figure 4.11: User Information



User Stats

Total Simulations:	1	CPU Usage:	24%
Running Simulations:	1	RAM Usage:	0.91%
Done Simulations:	0		

Figure 4.12: User Information - Resources

Administrators are also able to access the *Simulation info* page of any simulation, even if it doesn't belong to their account.

4.4.3 Configuration Parameters

The following parameters of the server can be configured by setting environment variables:

- **SELF_PORT**: Defaults to 8000
- **DATABASE_HOST**: Defaults to timescaledb
- **DATABASE_PORT**: Defaults to 5432
- **DATABASE_NAME**: Defaults to nntracker
- **DATABASE_USER**: Defaults to root
- **DATABASE_PASSWORD**: Defaults to postgres
- **GRAFANA_BASE_URL**: Used to embed grafana views,defaults to http://localhost:3000
- **DEPLOYER_BASE_URL**: Used to build deployer requests, defaults to http://tracker-deployer:7000

The final build of the server can be downloaded from Docker Hub at dioben/nntrackerua-server [2]

4.5 Neural Network Deployer

The deployer uses *Flask* with *Celery* workers in order to respond to requests as quickly as possible and then asynchronously perform tasks based on received requests.

This component exposes an API that allows the creation of new simulations, pause or remove said simulations, stop them prematurely, and get system performance information.

After receiving a request to any of the available endpoints, the deployer immediately makes a task corresponding to what was requested, which is put into a queue and picked up by any available *Celery* worker process.

In order to make a simulation, the responsible worker process first determines if the datasets used are stored locally or if it was provided a list of URLs, in which case it downloads the datasets

[2]https://hub.docker.com/repository/docker/dioben/nntrackerua-server

as needed. Following that, the deployer determines whether it is handling a K-fold simulation or not, if it is, it will launch multiple simulations at once, one for each fold pair made.

Finally, when launching a simulation, a container is made and each dataset is compressed into a zip and sent to the deployer where it's uncompressed, each container is then connected to the same global network for further communication with the parser. After the simulation is launched, the datasets downloaded/given are removed from the device's storage.

Simulation containers are instantiated via the Docker Daemon Socket, which is available at /var/run/docker.sock, this instantiation as well as other docker related operations are done using the official Docker SDK for Python.

4.5.1 API

The deployer provides the following API:

- **simulations [POST]**: Takes as input a JSON file with a *model* parameter and a *conf* parameter and orders the creation of a new simulation.
- **simulations/<simulation_id> [DELETE]**: Takes the container id of the simulation to delete as a request parameter.
- **simulations/<simulation_id>/<command> [POST]**: Takes the container id of the simulation as a request parameter, as well as a command to do onto the container: "START", "STOP" or "PAUSE".
- **simulations_statistics [GET]**: Returns a JSON file containing two parameters, *system_information* containing data related to the resource expenditure of the host system and *docker_containers_info* containing data related to the resources expenditure of each simulation container running.

4.5.2 Required files used by the deployer

In order to make a simulation, we need a JSON file with a *model* parameter and a *conf* parameter, this JSON file is automatically made by the server, taking into account the model JSON file provided by the user which is a representation of a Neural Network model outputted by the *Keras* Neural Net library, and the various configurations given by the user related to the making of the simulation including which loss function and optimizer to use. Examples of both files can be found in the project repository. Futhermore the deployer uses datasets files the same format as the ones referenced in apendix C.

4.5.3 Configuration Parameters

The following environment variables can be used to configure the deployer:

- **BROKER_URL**: link to a redis instance that celery depends on defaults to
redis://celery_deployer:6380

- **RESULT_BACKEND**: Link to where data is transferred from host to worker for it to be used, defaults to redis://celery_deployer:6380
- **DEPLOYABLE_NAME**: Defaults to dioben/nntrackerua-simulation
- **PARSER_URL**: Defaults to http://parser:6000
- **COMPONENTS**: Comma-separated list of docker containers to be monitored for performance statistics

Keep in mind that simulations deployed will inherit *PARSER_URL*
The Docker image for this component can be pulled at dioben/nntrackerua-deployer [3]

4.6 Neural Network Runtime Container

After being initialized, the simulation/*Neural Network* runtime container starts reading a plethora of necessary configuration parameters from the provided JSON configuration file related to the running simulation. After, it searches for the necessary dataset files copied by the deployer before container had started, loads each file, and parses them in the form of a *Tensorflow Dataset* object instance.

It's during this process that the file extension is taken account of in order to parse data into a single file format.

Afterwards, data processing operations like shuffling and batching are done.

Furthermore, configuration parameters are used to determine which optimizer to use, which loss function to use, which metrics should be sent to the parser, and their configuration.

Finally, training starts shortly after the model is obtained after parsing and compiling of the JSON model file provided.

During training, there's a callback configured to run every N epochs, determined from the configuration, in which a JSON containing the model, information related to the metrics, the running simulation id, the epoch in question, and all current weights are sent to the parser component to be processed and stored in a centralised database.

4.6.1 K-Fold Cross Validation

After all datasets are loaded and parsed to *Tensorflow Dataset* instances, when it is detected that the current simulation originated out of a simulation request in which the configuration file set a *K-Fold* simulation, the datasets will then be split into various training and validation folds, and the fold pair corresponding to the current simulation will be determined by other parameters in the configuration file.

[3] https://hub.docker.com/repository/docker/dioben/nntrackerua-deployer

4.6.2 Error Handling

During its runtime, if an error occurs, the simulation container will try to send an error message to the parser before shutting down in order for said message to be persistently stored and be shown to the user.

4.6.3 Libraries Used

The simulation container uses various Machine Learning and Data Manipulation libraries in order to both process the dataset files provided and offer support for functions like *K-Fold Cross Validation*, but also as the basis on which to parse the various file formats and to do model training. For all of this, the following libraries are used:

- **Numpy**: Used for data manipulation of matrices arrays in conjecture with *Tensorflow/Keras*.
- **Scikit-Learn**: Used in order to divide a dataset into various folds to be used during *K-Fold Cross Validation*.
- **Pandas**: Used during the parsing of CSV, JSON, ARFF and Pandas Dataframe dataset files.
- **Liac-Arff**: Used in order to parse ARFF files.
- **Tensorflow/Keras**: The combination of both is used as the Machine Learning framework from which the neural network model is defined, data is parsed to account for, and training occurs.

4.6.4 Configuration Parameters

The simulation supports the configuration of the following variables:

- **PARSER_URL**, which normally defaults to http://parser:6000.
- **DEPLOYER_DELETE_URL** which defaults to http://deployer:7000/simulations/

This component's Docker image can be found at dioben/nntrackerua-simulation [4]

4.7 Runtime Container Output Parser

The parser uses *Flask* with *Celery* workers in order to respond to requests as quickly as possible and then asynchronously perform tasks based on the received requests.

This component exposes an API that allows for simulations to send their information to the database. Simulations can either send an update, while they're running and when they finish, or send an error message, when a fatal error occurs.

[4]https://hub.docker.com/repository/docker/dioben/nntrackerua-simulation

After receiving a request to any of the available endpoints, the parser immediately makes a task corresponding to what was requested, which is put into a queue and picked up by any available *Celery* worker process.

The *Celery* worker assigned each task will then parse the JSON sent in the request and insert that information, such as weights, accuracy, loss, and other extra metrics, in case of an update or finish call, or an error message, in case of an error call, into the database.

4.7.1 API

The parser provides the following API:

- **update [POST]**: Takes as input a JSON file with a *model* parameter, a *weights* parameter, which contains all the weights in the simulation, and a *logs* parameter, which contains loss, accuracy and other extra metrics, and inserts this information into the *Weights*, *Epoch_values*, *Extra_metrics* tables of the database. It should be used for updates while the simulation is still running.

- **finish [POST]**: Takes as input a JSON file with a *model* parameter, a *weights* parameter, which contains all the weights in the simulation, and a *logs* parameter, which contains loss, accuracy and other extra metrics, and inserts this information into the *Weights*, *Epoch_values*, *Extra_metrics* tables of the database and updates the *simulations* table. It should be used for updates when the simulation finishes running.

- **send_error [POST]**: Takes as input a JSON file with an *error* parameter, which contains an error message and updates the *simulations* table of the database. It should be used for updates when the simulation has a fatal error.

4.7.2 Error Handling

The parser component has to maintain a connection with the database, so it can insert/update the database tables. For this effect, before every *Celery* worker starts executing its task it first checks if the connection with the database is still alive, if it isn't, it will try to reconnect and then continues executing its task.

4.7.3 Configuration Parameters

The following environment variables can be used to configure the parser:

- **DATABASE_HOST**: host address of the database, defaults to timescaledb

- **DATABASE_PORT**: port of the database, defaults to 5432

- **DATABASE_NAME**: name of the database parameter, defaults to nntracker

- **DATABASE_USER**: user parameter of the database, defaults to root

- **DATABASE_PASSWORD**: password parameter of the database, defaults to postgres
- **BROKER_URL**: link to a redis instance that celery depends on, defaults to redis://redis:6379
- **RESULT_BACKEND_URL**: Link to where data is transferred from host to worker for it to be used, defaults to redis://redis:6379

The parser is available at dioben/nntrackerua-parser [5]

4.8 Grafana Setup

We utilized Grafana with the *natel-plotly-plugin* extension in order to draw graphs oriented around a non-time x axis.

The provided repository contains configuration files provide the following features:

- Connect to our database
- Scan Dashboards from a given folder
- Instantiate Several Dashboards in separate folders

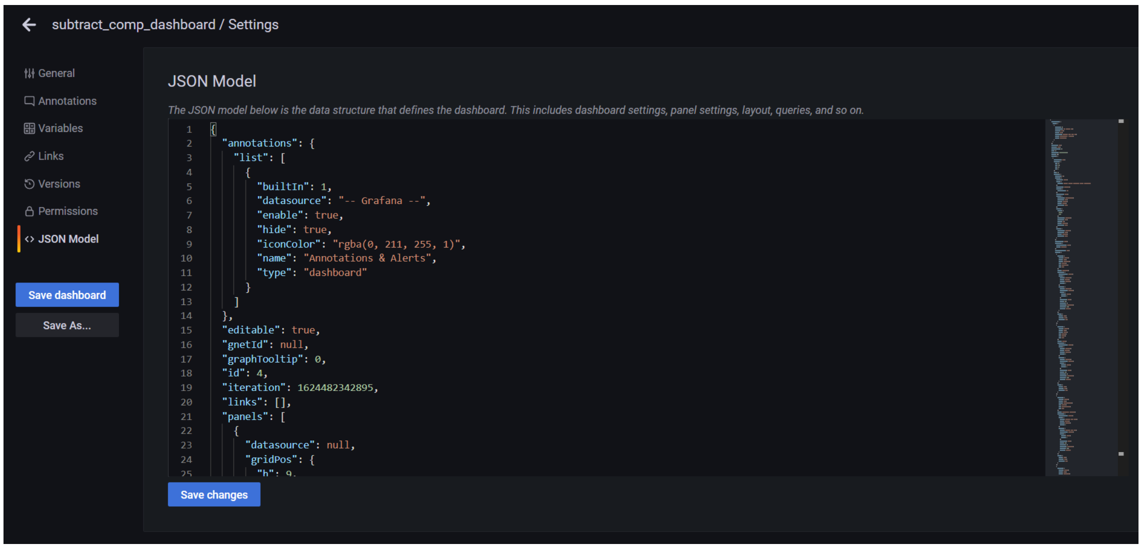
If our configuration files are used the following dashboards will be included:

- Display accuracy and loss for a given simulation over epoch
- Display the value of an optional metric for a given simulation over epoch
- Display a 3d scatter of the weights for a given layer, using epoch, weight array index and weight value.
- Display subtraction of accuracy and loss for 2 given simulations over epoch
- Display subtraction of an optional metric for 2 given simulations over epoch
- Display aggregate function of accuracy and loss for a given simulation tag over epoch
- Display aggregate function of an optional metric for a given simulation tag over epoch

Using our default configurations one can expand the existing list of dashboards by logging into Grafana at port 3000 using the credentials admin:grafana, creating their own dashboards via the GUI, and then copying the JSON configuration available under that dashboard's settings into a file in the appropriate folder.

An example configuration file can be found at appendix B and the web page where the configuration file can be copied can found on the figure 4.13.

[5] https://hub.docker.com/repository/docker/dioben/nntrackerua-parser



JSON Model

The JSON model below is the data structure that defines the dashboard. This includes dashboard settings, panel settings, layout, queries, and so on.

```
{
  "annotations": {
    "list": [
      {
        "builtIn": 1,
        "datasource": "-- Grafana --",
        "enable": true,
        "hide": true,
        "iconColor": "rgba(0, 211, 255, 1)",
        "name": "Annotations & Alerts",
        "type": "dashboard"
      }
    ]
  },
  "editable": true,
  "gnetId": null,
  "graphTooltip": 0,
  "id": 4,
  "iteration": 1624482342895,
  "links": [],
  "panels": [
    {
      "datasource": null,
      "gridPos": {
        "h": 9,
```

Figure 4.13: Extracting dashboard JSON from grafana

Some useful links:

- How to configure a Grafana Instance[6]
- Settings-related environment variables[7]

[6]https://grafana.com/docs/grafana/latest/administration/provisioning/
[7]https://grafana.com/docs/grafana/latest/administration/configuration/#configure-with-environment-variables

4.9 Deployment

The system's components are coordinated via docker-compose and is it intended that every part of the system to be replaceable.

The project relies heavily on docker, both for the launching of the system itself and for the deployment of simulations via the Docker UNIX socket.

The final deployment is presented in Figure 4.14

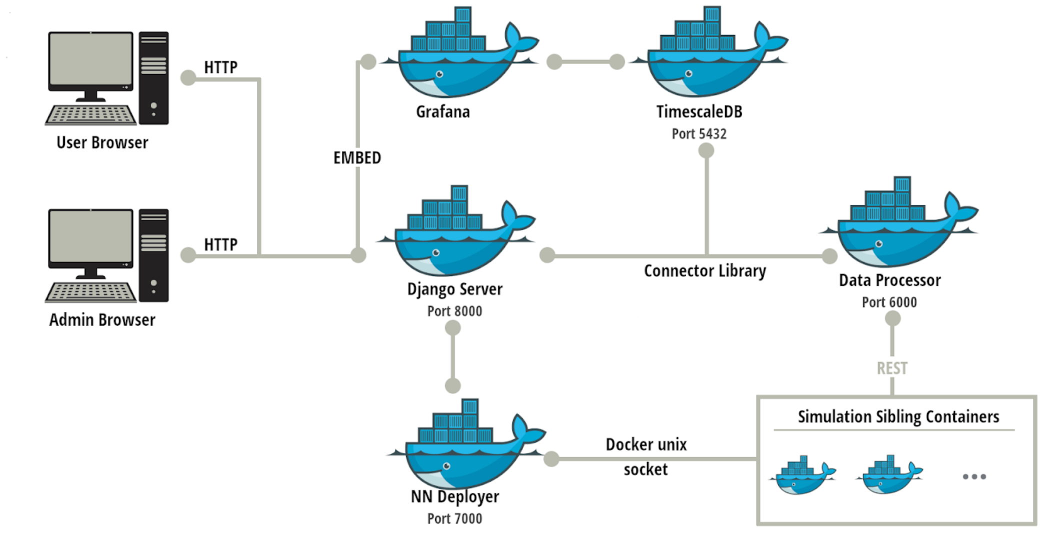


Figure 4.14: Architecture Diagram

Chapter 5

Results and Discussion

This chapter reviews the processes and results of the tests phases in this project and ends by analysing the main results obtained.

5.1 Sample

For the testing the system we used the files present in Appendix D to generate the Simulations explained in following section. These test's goal was to verify the capabilities of our system along with other requirements and objectives proposed for this system.

5.2 Method

We generated 8 simulation instances for our final database measurements using the following configuration:

- Batch Size: 32
- Epochs: 20
- Epoch Period: 2
- Learning Rate: 0.01 & 0.001
- Extra Metrics: *KLDivergence* & *LogCoshError*
- Optimizer: *RMSProp*
- Loss Function: *SparseCategoricalCrossentropy* - Configuration: from_logits with value True
- K-Fold Cross-Validation splits: 2

The obtained data was a total of 88 *Epoch Value* rows, 264 *Weights* rows, and 352 *Extra Metrics* rows, which adds up to 60 MB worth of data. If a different model was used there might have been more *Weights* rows, as with the one used only 1/4 of the layers contained values. We took a measurement of how storage use was distributed across tables and used multiple query command to obtain concrete data to measure performance of the expected most common queries in the system:

- Selecting all values from epoch_values for a simulation
- Selecting all values from extra_metrics for a simulation
- Selecting and unnesting all weights for a given layer of a simulation with

This query commands executed are present in Appendix E.

Storage use was obtained via *Postgre*'s *pg_database_size* and *pg_total_relation_size* functions as well as *TimescaleDB*'s *hypertable_size* function.

5.3 Results

The planning time overhead still makes up a large portion of query times for queries returning few data points, the weight related queries involve over 10000 data points and still meet speed expectations. Queries related to weights may reach hundreds of milliseconds for dense models with many epochs, but other query type delays are likely to stay under 10 milliseconds, which meets our expectations.

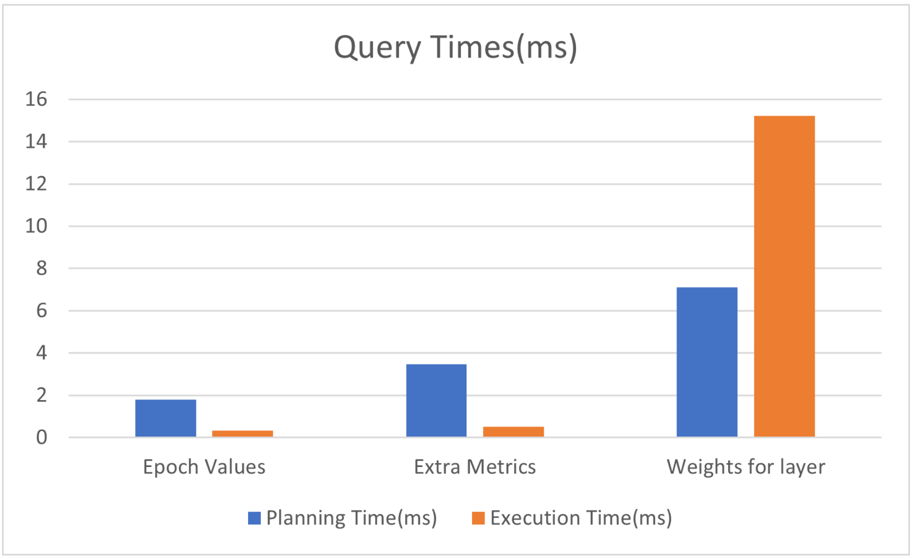


Figure 5.1: Query Times

The majority of the system's storage is dedicated to Weights. Weights made up over 75% of our system storage usage and we believe that this percentage will go up as more simulations are added, while the percentage occupied by *Others* (data unrelated to neural networks) will tend towards 0%.

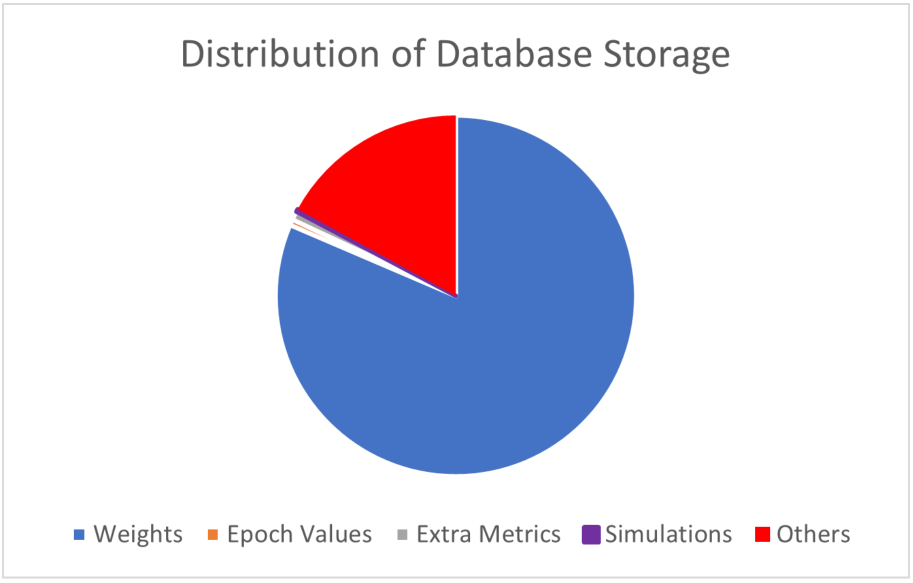


Figure 5.2: Distribution of Database Storage

Considering a storage expenditure rate of 60 MB per 88 epochs of data one can theoretically store over 1500000 epochs on a 1TB disk. Taking heavier models into account one can reasonably expect 250000 epochs worth of storage on the same disk.

Chapter 6

Conclusions and Future Work

6.1 Project Summary

This project was initiated by researching on all areas related to the subject, from similar tools for analysing neural network behaviour during training to the technologies being used. It was concluded that, although related projects had already been developed, none of them completely gathered the characteristics being implemented in this one. At the moment of writing this report, no projects or articles were found that performed periodic parameter collection, deployed neural networks in a containerized environment, launched several training experiments, persistently stored chosen parameters for later analysis, and included a visualization interface capable of selecting, organizing, comparing, and showing the evolution of the parameters collected in a variety of relevant graphics, capable of managing different training experiments.

Then we analysed the different technologies needed for the development of the project and conducted the gathering of system requirements, created Scenarios and Personas for a better specification of the project. After some meetings with our advisors, we perfected the project conceptual model and planned out the development phase of the project.

Project development was divided along each component needed, with teams set up to handle each component's functionality, with the added notice that some members were responsible for more than one additional role like the deployment of the whole system. Along the development there were small tests to verify the correct functioning of each component individually.

When the developing phase of the system was concluded, it was time to test the system to verify many aspects of it, along with making sure the system matched the objectives proposed.

6.2 Main Results

Taking into account the goals defined for this project it's clear that the main goals were accomplished, has it had been planned the final project performs periodic parameter collection for various neural network parameters, neural networks are deployed in a containerized environment, and the parameters gathered are stored persistently. The web client provides the ability to launch and manage multiple experiments and it also provides a visualization interface to analyse the results of this experiments.

6.3 Future Work

Future work on this project should start by improving the resources manager by allowing it to automatically stop a simulation if it is using too many resources, in addition to the current manual system.

As mentioned previously the neural network training technology used was *Keras + TensorFlow* but it could be enhanced by supporting the neural network training technology *Pytorch*.

At the moment the personal user functionalities and information are limited, so improving them by enabling the change and addition of personal information and enabling password and/or username updating would be a good enhancement in the user experience.

The system should also scale to run on several systems. This could be accomplished by implementing Kubernetes functionalities or by modifying the current deployment so that it runs a deployer instance per device and the service forwards requests to a new load-balancing component that distributes tasks across deployer instances.

Also, a feature to facilitate the management of simulations, by selecting multiple simulations at once and managing the life cycle of the selected simulations was discontinued in the conception of the project but would be a great improvement for the system.

Appendix A

Simulation Creation Configuration File

A.1 Properties

- name : The simulation's name
- dataset_train : A url to the training dataset
- dataset_test : A url to the test dataset
- dataset_val : A url to the validation dataset
- batch_size : The simulation's batch size
- total_epochs : How many epochs the simulation will run
- epoch_period : The period in which our service will collect the simulation's data
- optimizers : Optimizers to be used in each simulation and their configurations
- loss_function : Loss functions to be used in each simulation and their configurations
- learning_rates : Learning rates to be used in each simulation
- k-fold_validation : How many times the simulation will split
- k-fold_tag: Identification for all simulations caused by the split
- extra-metrics : Additional metrics to be gathered from each simulation epoch
- tags : Identifications for all simulations
- train_feature_name : Key name given in the map stored in the .npz training dataset file corresponding to the dataset features
- train_label_name : Key name given in the map stored in the .npz training dataset file corresponding to the dataset label

- test_feature_name : Key name given in the map stored in the .npz test dataset file corresponding to the dataset features
- test_label_name : Key name given in the map stored in the .npz test dataset file corresponding to the dataset label
- val_feature_name : Key name given in the map stored in the .npz validation dataset file corresponding to the dataset features
- val_label_name : Key name given in the map stored in the .npz validation dataset file corresponding to the dataset label
- label_column : Name of the column to be used as a feature vector

Appendix B

Grafana Dashboard Configuration

B.1 JSON Model

```
{
  "annotations": {
    "list": [
      {
        "builtIn": 1,
        "datasource": "-- Grafana --",
        "enable": true,
        "hide": true,
        "iconColor": "rgba(0, 211, 255, 1)",
        "name": "Annotations & Alerts",
        "type": "dashboard"
      }
    ]
  },
  "editable": true,
  "gnetId": null,
  "graphTooltip": 0,
  "id": 4,
  "iteration": 1624546541372,
  "links": [],
  "panels": [
    {
      "datasource": null,
      "gridPos": {
        "h": 9,
        "w": 12,
        "x": 0,
        "y": 0
      },
      "id": 2,
      "pconfig": {
        "fixScale": "",
        "layout": {
          "dragmode": "zoom",
          "font": {
            "family": "\"Open Sans\", Helvetica, Arial, sans-serif"
          },
          "hovermode": "closest",
          "legend": {
            "orientation": "h"
          },
          "showlegend": false,
          "xaxis": {
            "rangemode": "nonnegative",
            "showgrid": true,
            "title": "Epoch",
            "type": "auto",
            "zeroline": false
          },
          "yaxis": {
            "range": [
              -1000,
              1000
            ],
            "rangemode": "between",
            "showgrid": true,
            "title": "",
            "type": "linear",
            "zeroline": false
          },
          "zaxis": {
            "rangemode": "normal",
            "showgrid": true,
            "type": "linear",
            "zeroline": false
          }
        },
        "loadFromCDN": false,
        "settings": {
          "displayModeBar": false,
          "type": "scatter"
        },
        "showAnnotations": true,
        "traces": [
          {
            "mapping": {
              "color": "epoch",
              "size": null,
              "text": "accuracy",
              "x": "epoch",
              "y": "accuracy",
              "z": null
            },
            "name": "Accuracy",
            "settings": {
              "color_option": "solid",
              "line": {
                "color": "#005f81",
                "dash": "solid",
                "shape": "linear",
                "width": 4
              },
              "marker": {
                "color": "#33B5E5",
                "colorscale": "YlOrRd",
                "line": {
                  "color": "#DDD",
                  "width": 0
                },
                "showscale": false,
                "size": 10,
                "sizemin": 3,
                "sizemode": "diameter",
                "sizeref": 0.2,
                "symbol": "circle"
              }
            },
            "show": {
              "line": true,
              "lines": true,
              "markers": true
            }
          },
          {
            "mapping": {
              "color": "epoch",
              "size": null,
              "text": "loss",
              "x": "epoch",
              "y": "loss",
              "z": null
            },
            "name": "Loss",
            "settings": {
              "color_option": "solid",
              "line": {
                "color": "#C4162A",
                "dash": "solid",
                "shape": "linear",
                "width": 4
              },
              "marker": {
                "color": "#E02F44",
                "colorscale": "YlOrRd",
                "line": {
                  "color": "#DDD",
                  "width": 0
                },
                "showscale": false,
                "size": 10,
                "sizemin": 3,
                "sizemode": "diameter",
                "sizeref": 0.2,
                "symbol": "circle"
              }
            },
            "show": {
              "line": true,
              "lines": true,
              "markers": true
            }
          },
          {
            "mapping": {
              "color": "epoch",
              "size": null,
              "text": "val_accuracy",
              "x": "epoch",
              "y": "val_accuracy",
              "z": null
            },
            "name": "Validation Accuracy",
            "settings": {
              "color_option": "solid",
              "line": {
                "color": "rgb(29, 224, 180)",
                "dash": "solid",
                "shape": "linear",
                "width": 4
              },
              "marker": {
                "color": "rgb(12, 115, 94)",
                "colorscale": "YlOrRd",
                "line": {
                  "color": "#DDD",
                  "width": 0
                },
                "showscale": false,
                "size": 10,
                "sizemin": 3,
                "sizemode": "diameter",
                "sizeref": 0.2,
                "symbol": "circle"
              }
            },
            "show": {
              "line": true,
              "lines": true,
              "markers": true
            }
          },
          {
            "mapping": {
              "color": "epoch",
              "size": null,
              "text": "val_loss",
              "x": "epoch",
              "y": "val_loss",
              "z": null
            },
            "name": "Validation Loss",
            "settings": {
              "color_option": "solid",
              "line": {
                "color": "rgb(156, 10, 106)",
                "dash": "solid",
                "shape": "linear",
                "width": 4
              },
              "marker": {
                "color": "rgb(133, 5, 88)",
                "colorscale": "YlOrRd",
                "line": {
                  "color": "#DDD",
                  "width": 0
                },
                "showscale": false,
                "size": 10,
                "sizemin": 3,
                "sizemode": "diameter",
                "sizeref": 0.2,
                "symbol": "circle"
              }
            },
            "show": {
              "line": true,
              "lines": true,
              "markers": true
            }
          }
        ]
      },
      "pluginVersion": "7.5.7",
      "targets": [
        {
          "format": "time_series",
          "group": [],
          "metricColumn": "none",
          "rawQuery": true,
          "rawSql": "SELECT\n  1 AS \"time\",\n  a.epoch as epoch,\n  case when E'$
              {values:csv}E' like '%1%' then a.accuracy-b.accuracy else -10000000
              end as accuracy,\n  case when E'${values:csv}E' like '%2%' then a.
              loss - b.loss else -10000000 end as loss,\n  case when E'${values:csv
              }E' like '%3%' then a.val_accuracy - b.val_accuracy else -10000000
              end as val_accuracy,\n  case when E'${values:csv}E' like '%4%' then a
              .val_loss - b.val_loss else -10000000 end as val_loss\nFROM (select *
               from epoch_values where $__timeFilter(\"time\") AND sim_id = '
              $sim_id1') as a\ninner join (select * from epoch_values where
              $__timeFilter(\"time\") AND sim_id = '$sim_id2') as b\non a.epoch=b.
              epoch\nORDER BY 1",
          "refId": "A",
          "select": [
            [
              {
                "params": [
                  "loss"
                ],
                "type": "column"
              }
            ]
          ],
          "table": "epoch_values",
          "timeColumn": "\"time\"",
          "timeColumnType": "timestamp",
          "where": [
            {
              "name": "$__timeFilter",
              "params": [],
              "type": "macro"
            }
          ]
        }
      ],
      "title": "Panel Title",
      "type": "natel-plotly-panel",
      "version": 1
```

```
    }
  ],
  "refresh": "1m",
  "schemaVersion": 30,
  "style": "dark",
  "tags": [],
  "templating": {
    "list": [
      {
        "allValue": null,
        "current": {
          "selected": true,
          "tags": [],
          "text": [
            "validation accuracy"
          ],
          "value": [
            "3"
          ]
        },
        "description": "selectable values",
        "error": null,
        "hide": 0,
        "includeAll": false,
        "label": "Values",
        "multi": true,
        "name": "values",
        "options": [
          {
            "selected": false,
            "text": "accuracy",
            "value": "1"
          },
          {
            "selected": false,
            "text": "loss",
            "value": "2"
          },
          {
            "selected": true,
            "text": "validation accuracy",
            "value": "3"
          },
          {
            "selected": false,
            "text": "validation loss",
            "value": "4"
          }
        ],
        "query": "accuracy : 1, loss : 2,validation accuracy : 3, validation loss : 4",
        "queryValue": "",
        "skipUrlSync": false,
        "type": "custom"
      },
      {
        "current": {
          "selected": false,
          "text": "112b38ec-56c9-49ac-b4df-a499ababb798",
          "value": "112b38ec-56c9-49ac-b4df-a499ababb798"
        },
        "description": null,
        "error": null,
        "hide": 0,
        "label": null,
        "name": "sim_id1",
        "options": [
          {
            "selected": true,
            "text": "112b38ec-56c9-49ac-b4df-a499ababb798",
            "value": "112b38ec-56c9-49ac-b4df-a499ababb798"
          }
        ],
        "query": "112b38ec-56c9-49ac-b4df-a499ababb798",
        "skipUrlSync": false,
        "type": "textbox"
      },
      {
        "current": {
          "selected": false,
          "text": "112b38ec-56c9-49ac-b4df-a499ababb798",
          "value": "112b38ec-56c9-49ac-b4df-a499ababb798"
        },
        "description": null,
        "error": null,
        "hide": 0,
        "label": null,
        "name": "sim_id2",
        "options": [
          {
            "selected": true,
            "text": "112b38ec-56c9-49ac-b4df-a499ababb798",
            "value": "112b38ec-56c9-49ac-b4df-a499ababb798"
          }
        ],
        "query": "112b38ec-56c9-49ac-b4df-a499ababb798",
        "skipUrlSync": false,
        "type": "textbox"
      }
    ]
  },
  "time": {
    "from": "now-7d",
    "to": "now"
  },
  "timepicker": {},
  "timezone": "",
  "title": "subtract_comp_dashboard",
  "uid": "wTZDdD6Gz",
  "version": 1
}
```

Appendix C

Dataset File Formats

C.1 NPZ - NumPy file

Each dataset file should be a map stored by using numpy.save(), where there should be two key-value pairs, the first one should be a list of features of the dataset and second should be a list of corresponding labels, it's required that the user remembers which names were used as keys but there isn't any requirement for what these names should be.

Example: *train_dt.npz*

```
{
'train_feature' : [feature list]
'train_label' : [label list]
}
```

C.2 CSV - Comma-separated values

Each dataset file should be a csv table where each column corresponds to one of the input parameters of the neural network, with the exception of one which should correspond to the label. Taking all this into account, each row of non-label columns should form a input feature vector to be fed into the algorithm. The name of the label column should be taken into account, but there aren't any requirements regarding what that name should be.

Example: *train_dt.csv*

```
f1 | f2 | label
58   20     1
12   60     0
```

C.3 ARFF - Attribute-Relation File Format

In a similar manner to the .csv file, it's composed of a table where each columns corresponds to one of the input parameters of the neural network, with the exception of one which should correspond to the label, corresponding to what should be the @DATA section of the files, besides that it should also include a header section describing each column attribute by its name and type. The name of the label column should be taken into account, but there aren't any requirements regarding what that name should be.

Example: *train_dt.arff*

```
@RELATION EXAMPLE
@ATTRIBUTE label NUMERIC
@ATTRIBUTE f1 NUMERIC
@ATTRIBUTE f2 NUMERIC
@ATTRIBUTE f3 NUMERIC
@DATA
1,500,100,200
2,100,700,150
0,200,400,250
```

C.4 JSON - JavaScript Object Notation

Each dataset file should contain a map, which should contain a number of key-value pairs corresponding to each one of the inputs that would constitute a matrix of feature vectors and a key-value pair corresponding to the label vector, the key name of the label vector should be known by the user but there aren't any requirements on what that name should be.

Example: *train_dt.json*

```
{
"x1" : [134,214,3123],
"x2" : [1586,245,3123],
"label" : [0,4,5],
}
```

C.5 Pandas Dataframe (.zip and .pickle)

Each dataset file should be obtained by using the pandas.to_pickle() function when converting a pandas dataframe, by using the compression='zip' argument value or by outputting to a .zip file instead of a pickle one, it is possible to obtain a zip-compressed version of the pickled dataframe file. The user should take into account the names of the column in the dataframe object that corresponds to the label vector, there are not any requirements regarding what that name should be.

Example: *Pandas Dataframe Instance - train_df*

```
f1 | f2 | label
58   20     1
12   60     0
```

Example of conversion for both both compressed and uncompressed:

```
pandas.to_pickle(train_df, "./dataset_train.pickle")
pandas.to_pickle(train_df, "./dataset_train.zip")
```

Appendix D

Test Files

D.1 Model File

```json
{"class_name": "Sequential", "config": {"name": "sequential_2", "layers": [{"class_name": "InputLayer", "config": {"batch_input_shape": [null, 28, 28], "dtype": "float32", "sparse": false, "ragged": false, "name": "flatten_2_input"}}, {"class_name": "Flatten", "config": {"name": "flatten_2", "trainable": true, "batch_input_shape": [null, 28, 28], "dtype": "float32", "data_format": "channels_last"}}, {"class_name": "Dense", "config": {"name": "dense_4", "trainable": true, "dtype": "float32", "units": 128, "activation": "relu", "use_bias": true, "kernel_initializer": {"class_name": "GlorotUniform", "config": {"seed": null}}, "bias_initializer": {"class_name": "Zeros", "config": {}}, "kernel_regularizer": null, "bias_regularizer": null, "activity_regularizer": null, "kernel_constraint": null, "bias_constraint": null}}, {"class_name": "Dense", "config": {"name": "dense_5", "trainable": true, "dtype": "float32", "units": 10, "activation": "linear", "use_bias": true, "kernel_initializer": {"class_name": "GlorotUniform", "config": {"seed": null}}, "bias_initializer": {"class_name": "Zeros", "config": {}}, "kernel_regularizer": null, "bias_regularizer": null, "activity_regularizer": null, "kernel_constraint": null, "bias_constraint": null}}]}, "keras_version": "2.4.0", "backend": "tensorflow"}
```

D.2 Data Set Validation File

Link: https://drive.google.com/file/d/1n8nbIICz4PP-EZDd0f9LeRcxXmA9ZjS3/view?usp=sharing

D.3 Data Set Training File

Link: https://drive.google.com/file/d/1AsnQrKX80VuL4lCmKXnv9wbt9IwYmmsz/view?usp=sharing

D.4 Data Set Test File

Link: https://drive.google.com/file/d/1itJWjThJ6cIucDM0Bh1_AuZjGereW3Zx/view?usp=sharing

Appendix E

Query Commands

E.1 Get Weights for layer over time, unrolled into individual rows

```
EXPLAIN ANALYZE
SELECT
  "time" AS "time",
  epoch,
  w,
  array_index

FROM weights,unnest(weight) with ordinality as a(w,array_index)
WHERE
  "time" BETWEEN '2021-05-25T10:45:50.479Z' AND '2021-06-24T10:45:50.479Z'
  and layer_index= 2
  and sim_id= '34a30c0d-c59e-4f0a-9918-3460d3b9e5e4'
ORDER BY 1;
```

E.2 Epoch Values

```
EXPLAIN ANALYZE
SELECT * FROM epoch_values
where sim_id='34a30c0d-c59e-4f0a-9918-3460d3b9e5e4';
```

E.3 Extra Metrics

```
EXPLAIN ANALYZE
SELECT * from extra_metrics
sim_id='34a30c0d-c59e-4f0a-9918-3460d3b9e5e4';
```

References

[1] Joy Buolamwini. Artificial Intelligence Has a Problem With Gender and Racial Bias. Here's How to Solve It. https://time.com/5520558/artificial-intelligence-racial-gender-bias/, 2019. Last accessed 2021-06-20.

[2] Fahim Dalvi, Avery Nortonsmith, Anthony Bau, Yonatan Belinkov, Hassan Sajjad, Nadir Durrani, and James Glass. Neurox: A toolkit for analyzing individual neurons in neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):9851–9852, Jul. 2019.

[3] Docker. Docker Overview. https://docs.docker.com/get-started/overview/, 2021. Last accessed 2021-06-20.

[4] Aleksa Filipović. Ethical issues in the fields of artificial intelligence, self-driving vehicles, and autonomous weapon systems. 03 2019.

[5] Flask. What is Flask Python. https://pythonbasics.org/what-is-flask-python/, 2021. Last accessed 2021-06-20.

[6] Grafana. Grafana: Getting Started. https://grafana.com/docs/grafana/latest/getting-started/, 2021. Last accessed 2021-06-20.

[7] Nicholas Hunt-Walker. An introduction to the Flask Python web app framework. https://opensource.com/article/18/4/flask, 2018. Last accessed 2021-06-20.

[8] Jonathan Johnson. What's a Deep Neural Network? Deep Nets Explained. https://www.bmc.com/blogs/deep-neural-network/, 2020. Last accessed 2021-06-20.

[9] Sagara Technology Idea Lab. What is Django and Why is it Used? https://sagaratechnology.medium.com/what-is-django-and-why-is-it-used-2dafdc75ce67, 2020. Last accessed 2021-06-20.

[10] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. TensorFuzz: Debugging neural networks with coverage-guided fuzzing. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4901–4911. PMLR, 09–15 Jun 2019.

[11] Data Pilot. Introduction to TimescaleDB. https://kb.objectrocket.com/timescaledb/introduction-to-timescaledb-1560, 2020. Last accessed 2021-06-20.

[12] Simplilearn. What Is Keras: The Best Introductory Guide To Keras. https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-keras, 2021. Last accessed 2021-06-20.

[13] Open Source. What is Docker? https://opensource.com/resources/what-docker, 2021. Last accessed 2021-06-20.

[14] Hendrik Strobelt, Sebastian Gehrmann, Michael Behrisch, Adam Perer, Hanspeter Pfister, and Alexander M. Rush. Seq2seq-vis: A visual debugging tool for sequence-to-sequence models. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):353–363, 2019.

[15] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. Deepconcolic: Testing and debugging deep neural networks. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 111–114, 2019.

[16] Elizabeth Svoboda. Artificial intelligence is improving the detection of lung cancer. https://www.nature.com/articles/d41586-020-03157-9, 2020. Last accessed 2021-06-20.

[17] TensorFlow. TensorBoard: TensorFlow's visualization toolkit. https://www.tensorflow.org/tensorboard, 2021. Last accessed 2021-06-20.

[18] Muhammad Usman, Yannic Noller, Corina Pasareanu, Youcheng Sun, and Divya Gopinath. Neurospf: A tool for the symbolic analysis of neural networks, 2021.

[19] Mohammad Wardat, Wei Le, and Hridesh Rajan. Deeplocalize: Fault localization for deep neural networks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 251–262, 2021.

[20] Xiaofei Xie, Lei Ma, Haijun Wang, Yuekang Li, Yang Liu, and Xiaohong Li. Diffchaser: Detecting disagreements for deep neural networks. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 5772–5778. International Joint Conferences on Artificial Intelligence Organization, 7 2019.