Politecnico di Torino

Politecnico di Torino

Ingegneria Informatica (Computer Engineering)

A.Y. 2023/2024

October 2024

DeepBlocks: visually constructing, debugging and training Deep Neural Networks

Supervisors:

Luigi De Russis

Tommaso Calò

Candidate:

Filippo Castellarin

Abstract

Programming Deep Neural Networks (DNNs) can be a daunting task for individuals with limited programming expertise. Moreover, the potential of Deep Learning (DL) models could be vastly expanded if more people had access to this technology. Conversely, for those already coding Neural Networks (NNs), comprehending runtime errors, pinpointing issues, and resolving them remains a significant hurdle.

DeepBlocks is introduced as a Visual Programming (VP) tool that enables the construction, debugging, training, and evaluation of DNNs through the addition and connection of blocks, along with the configuration of training parameters.

The application was tested by actual users, whose feedback was instrumental in further development and also conveyed gratitude for democratizing the power of Deep Learning. This was achieved by providing an intuitive tool that allows a wider audience to experiment with Deep Neural Networks.

Acknowledgements

Un sentito grazie ai relatori Luigi e Tommaso per la loro infinita disponibilità e tempestività ad ogni mia richiesta. Grazie per avermi fornito ogni materiale utile alla stesura dell'elaborato.

Un grazie speciale se lo meritano mia mamma Paola, mio fratello Mattia e mia nonna Anna, che mi sono sempre stati vicini e mi hanno spronato in ogni momento a entrare nel mondo universitario e fare esperienze di vita indimenticabili, che non esiste solo lo studio e che con l'umiltà si arriva dappertutto.

Grazie ad Andrea, Bruno, Giulio, Hilary, Ilaria, Letizia, Luca, Luca, Martina, Pierpaolo, Sabrina per tutto ciò che abbiamo vissuto insieme. Senza di voi questi anni di università non sarebbero stati gli stessi, ma grazie a voi, e alle tantissime esperienze insieme, me li porterò per sempre nel cuore. Tra mare, montagna, macchina, treno, pullman, barca, lauree, nord e sud abbiamo fatto di tutto insieme, e non mi scorderò mai la bellezza di ogni istante speso con voi.

Grazie a tutti i miei colleghi di corso, a coloro con cui ho condiviso progetti di gruppo. Un grazie particolare ai miei compagni del progetto SwapMap.

Grazie a tutti i miei compagni di avventure con cui ho condiviso dei momenti unici e che hanno contribuito ad affrontare il mio percorso universitario in perfetta armonia con il mio percorso di crescita personale. Grazie ad Alter.POLIS, HKN, al corpo europeo di solidarietà e a tutti quei progetti che ho portato avanti durante il mio percorso accademico volti a formare la mia persona e far sbocciare la mia personalità senza la quale non avrei potuto vivere così tanti bei momenti insieme a così tante belle persone.

No pueden faltar las miles de gracias para Aurora, que siempre ha estado a mi lado en cualquier etapa de mi universidad, incluso antes, y que siempre me ha acogido en su casa en mis viajes post-exámenes de febrero. Gracias por todo lo que representas para mí.

Table of Contents

List of Figures

Glossary

VP
: Visual Programming

AI
: Artificial Intelligence

ML
: Machine Learning

DL
: Deep Learning

NN
: Neural Network

ANN
: Artificial Neural Network

DNN
: Deep Neural Network

ONNX
: Open Neural Network Exchange

EUD
: End User Development

UI
: User Interface

GUI

Graphical User Interface

CE

Cross Entropy

BCE

Binary Cross Entropy

MSE

Mean Squared Error

IDL

Interface Definition Language

IDE

Integrated Development Environment

Med-Fi

Medium Fidelity

SGD

Stochastic Gradient Descent

RPC

Remote Procedure Call

ViT

Vision Transformer

NLP

Natural Language Processing

Chapter 1

Introduction

This thesis addresses the challenge of implementing Deep Learning (DL) pipelines for individuals with limited or minimal programming expertise. It proposes the development of a web application designed to simplify the creation and debugging of models, tailored to the user's specific preferences. This chapter introduces the overview of the work (1.1), its aim (1.2), its contribution (1.3) and finally the document structure (1.4).

1.1 Overview

Machine Learning (ML) algorithms are becoming increasingly prevalent across various industries. Marketing and finance are two sectors where the use of ML algorithms is particularly prioritized. In marketing, for example, ML can analyze user preferences from online searches or shopping carts to provide personalized recommendations. Additionally, ML is utilized to design targeted marketing campaigns by identifying products and services that align with specific customer interests. ML algorithms have been employed as personal assistants such as Amazon's Alexa, Apple's Siri and Google Assistant. This communication involves speech recognition, speech-to-text conversion, Natural Language Processing (NLP) and text-to-speech. Moreover, ML is behind message bots which are used for customer service to ensure fast responses [1].

Machine Learning can also apply to healthcare where it relies on the collection of patient data. Using systems and tools designed to sort and categorize data, ML algorithms can discover patterns in datasets that allow medical professionals to identify new diseases, improve diagnosis and predict treatment outcomes [2] [1]. DeepDR is a Deep Learning system that can detect early-to-late stages of diabetic retinopathy contributing to timely treatment [3].

Building ML pipelines is challenging. The first challenge lies in designing and

setting up the pipeline itself due to its complexity, the required expertise, the associated costs, and the need for continuous monitoring and updating. It requires careful orchestration of the data processing sequence, model training, evaluation, and deployment steps [4]. Another challenge is maintaining the pipeline from the Concept Drift which is when the relationship between the input and target changes over time that usually occurs when real-world environments change in contrast to the training data the model learned from. Data Drift is another type of drift, but this is caused by unforeseen changes in the input data. Models trained from historic datasets may become less accurate and effective over time as the underlying relationships between the data shifts evolves [5] [4].

Constructing a Machine Learning pipeline is a substantial endeavor that necessitates meticulous planning and consideration of various elements. Initially, it is crucial to comprehend the problem and the data at hand to tailor the ML pipeline design. This comprehension impacts each phase of the pipeline. Subsequently, selecting the appropriate tools is essential, depending on the project's specific needs, such as the programming language and framework. Monitoring the model's performance is vital as it aids in promptly responding to environmental shifts and updating the model by retraining with new data or modifying the feature set, thereby averting Concept Drift [4].

Debugging Machine Learning pipelines is complex due to the multiple interconnected stages involved, from data pre-processing to model deployment. Each stage can introduce errors, and the abstract error messages often fail to pinpoint the exact source of the problem. Issues such as data inconsistencies, incorrect feature engineering, or flawed model configurations can propagate through the pipeline, leading to unexpected results. The integration of various tools and frameworks can also introduce compatibility issues. To address these challenges, developers must use a systematic approach, employing logging and visualization tools such as UMLAUT and ModelTracker to trace errors, debug model performances and ensure pipeline reliability [6] [7].

Given the significant challenges in designing and developing DL networks aforementioned, including the complexities of debugging interconnected stages and managing various tools and frameworks, the learning curve for mastering ML pipelines is exceptionally steep. As a result, becoming proficient in ML pipelines requires not only technical expertise but also extensive experience in managing the nuances of end-to-end Machine Learning workflows [8].

Visual Programming (VP) tools applied to the ML sector such as TeachableMachine [9], VisualBlocks [10] and DeepBlocks [11] have emerged to tackle the complexities faced by Machine Learning programmers, offering specialized solutions. These VP tools facilitate the creation and training of ML models by providing a straightforward drag-and-drop interface on a canvas. This allows users to easily configure parameters and define connections between different elements [12].

However, while VP tools offer increased simplicity, they may compromise on capabilities such as debugging or constructing intricate networks. NVIDIA DIGITS[1], for instance, was the first DL specific tool to provide an user interface to perform tasks like data loading and one-click training. However, it is only possible to train pre-built networks and it does not have the capability to design models from scratch. Another example is Google's Tensorboard[2] which is often used as a supplement to visualize Tensorflow models and finally Netron[3] is used to visualize existing models coded in various libraries [8].

1.2 Aim

The thesis aims to develop a new version of the DeepBlocks application developed by Tommaso Calò and Luigi De Russis [11] utilizing Visual Programming (VP) to construct, debug, and train Deep Learning (DL) pipelines. This will address the challenges in design and implementation, reducing the learning curve. The tool's design is intended for use by Machine Learning (ML) students to become acquainted with the DL pipeline environment and by ML novices and experts to facilitate straightforward network construction and debugging.

When designing Deep Learning networks, it is important to consider that as the network size grows, it can become difficult to manage and interpret. By adopting a block-based approach (see more in Section 3), users can sequentially arrange blocks on a canvas by adding and connecting them, which improves the network's scalability and readability as it expands.

When creating DL networks, it is difficult to individuate potential issues. The VP tool aims also to validate the DL network prior to initiating the training by sequentially verifying each block for incorrect parameters, highlighting issues on the User Interface (UI). This assists users in pinpointing issues and offers potential solutions for resolution. Once the DL network validation is verified as successful, users can proceed to train the pipeline and analyze the outcomes. Visualizations are helpful to record the model's history, facilitating user analysis of pipeline modifications against previous iterations as well as getting insights from the model's performances through metrics and plots.

The rise of Visual Programming paradigms and web-based platforms is making the DL network creation more accessible to a broader audience. The implementation of a web application that follows the VP paradigm offers an intuitive, user-friendly environment that can be accessed via the Internet. By providing a drag-and-drop

[1] `https://developer.nvidia.com/digits`

[2] `https://www.tensorflow.org/tensorboard`

[3] `https://netron.app/`

interface and simplifying the configuration of DL pipelines, the platform enables users to experiment with and deepen their understanding of DL concepts without the need for advanced programming skills. As a result, it opens up opportunities for a diverse range of individuals to engage with and contribute to the field of Deep Learning, fostering a more inclusive and educational environment.

1.3 Contribution

This work makes several contributions to the advancement of Machine Learning and Deep Learning technologies. Firstly, it offers a comprehensive literature review of Visual Programming (VP) tools tailored for creating and debugging ML pipelines. This review not only catalogs existing tools but also critically examines their functionalities, identifying gaps and challenges that current solutions face. This analysis provides a strong foundation for understanding how VP tools can be optimized for more efficient and user-friendly DL pipeline development, addressing the needs of both novice and experienced users.

Secondly, the project focuses on the design and development of a three-tier web application specifically crafted for building DL pipelines. This architecture was chosen for its ability to separate concerns across different layers: the frontend delivers an intuitive interface that simplifies the process of constructing DL networks through a drag-and-drop mechanism; the middleware ensures smooth communication and data handling between the user interface and the backend service and other external services; and the backend server is responsible for the heavy lifting of model creation, training and debugging. This tiered structure enhances scalability, maintainability, and security, allowing the application to handle increasingly complex DL models while remaining accessible and responsive.

Moreover, the implementation details, including the source code, are provided to facilitate replication, customization, and further innovation.

Finally, the project includes a user testing phase, which was designed to gather qualitative and quantitative feedback from a diverse set of users, ranging from beginners to more experienced users. The results of this testing provide valuable insights into the usability, functionality, and overall effectiveness of the web application as well as precious comments for further development. This user-centered approach is fundamental to the project's goal of democratizing access to DL technologies, making them more accessible and usable for a wider audience.

1.4 Thesis structure

This thesis is structured to guide the reader through the development and evaluation of a web-based Visual Programming (VP) platform for Deep Learning pipelines.

Chapter 2 begins by providing a comprehensive overview of the existing literature, situating this work within the broader context of Machine Learning and VP tools. This chapter identifies key aspects of VP tools for Machine Learning, challenges, and gaps in current solutions, setting the stage for the subsequent contributions.

Following this, Chapter 3 delves into the specifics of ML pipelines, detailing the methodologies used in this project, particularly the application of Visual Programming techniques. This chapter outlines how VP methodologies are employed to simplify the creation, debugging, and management of DL pipelines, making these complex processes more accessible as well as presenting the gRPC environment, chosen for the implementation of the VP web-based tool.

Chapter 4 then discusses the design phase of the web application, explaining the rationale behind the architectural choices and the User Interface design. This chapter explores how the application's design supports scalability, usability, and integration of DL models, ensuring a seamless user experience.

In Chapter 5, the focus shifts to the practical aspects of the work, presenting a detailed account of the implementation process. This chapter covers the technical challenges encountered, the solutions developed, and the step-by-step construction of the application's three-tier architecture, including the frontend, middleware, and backend components.

Chapter 6 then delineates the user testing phase, from the structure of the tests to the analysis of the results. This chapter provides insights into how the application was evaluated by a well defined group of users, the feedback received, and how this feedback could be useful for further refinements of the platform.

Finally, Chapter 7 draws together the findings from the previous chapters, offering conclusions about the efficacy and potential impact of the developed platform. It also suggests directions for future work, emphasizing the ongoing need for innovation in making DL technologies more accessible to a broader audience.

Chapter 2

Related Work

This chapter provides descriptions of the scientific papers researched during this thesis, offering insights into the research community's efforts to assist users in creating and training Machine Learning pipelines, potentially through the use of the Visual Programming paradigm.

This phase has been launched to pinpoint potential new features for development within DeepBlocks.

2.1 DeepBlocks

DeepBlocks by Tommaso Calò and Luigi De Russis [11], supervisors of this project, provides an overview of the main features that must be retained in the new version of the application.

A key feature of the tool is its ability to verify network correctness prior to initiating the training project. It accomplishes this by iterating through the model's modules and invoking the forward function. Should an error occur, the application assists in pinpointing the erroneous node by highlighting it and notifying the user of the specific issue.

Another significant aspect is the adoption of the Visual Programming paradigm, which allows users to place blocks on a canvas, link them as desired, and configure the block parameters prior to initiating the verification or training process.

The final notable feature of the DeepBlocks desktop application is the capability for users to create "superblocks". A superblock is a node that can encompass other nodes within it. While it appears as a singular node on the canvas, users can delve into it to reveal the individual nodes that constitute the superblock. In the desktop version of DeepBlocks, these superblocks are displayed on the left side, complete with all their details and parameters clearly specified.

Users have the option to select a loss function from a dropdown list of predefined

functions or to upload a file with a custom loss function for use during the training process.

2.2 Visual Blocks

Visual Blocks by R. Du et al. [12] is a Visual Programming tool to build and train Machine Learning pipelines.

During the formative studies the authors found out that creating a Visual Programming interface for multimedia applications with Machine Learning models interested all their participants. Most ML researchers, in fact, do not have enough skills to write prototypes and so providing them a VP tool to rapidly create ML pipelines would be beneficial. They observed also that comparing model outputs and results is a crucial activity in the day-to-day life of a ML researcher by providing visualizations able to gain insights and make evaluations on the fly.

Visual Blocks also offers intriguing insights into the freedom users have to connect blocks and explore their functionality. Indeed, tests have revealed how users interact with the blocks and whether this fosters creativity.

2.3 DL-IDE

DL-IDE by Tamilselvam et al. [8] is another tool to design Deep Learning models based on a 'no-code' paradigm.

During their qualitative survey among software engineers they discovered that the 83% of the participants took about 3-4 days to implement a Deep Learning model even if 86% of those respondents rated themselves with very good programming skills. Thus, even good programmers find it challenging to implement DL models. Another result of the survey shows that more than 92% of the respondents wanted an interoperability framework to convert the model in one library into another one. The most suitable existing framework determined by the authors of DL-IDE is the Open Neural Network Exchange (ONNX) [13] which comes with an online visualizer, the Netron web app [1].

The authors identified the main requirements for any DL tool:

- Design and Visualization: intuitive design and construction of Deep Learning models.
- Elimination of need to code: automatic source code generation.

[1] `https://netron.app/`

- Production-ready capabilities: efficiently training the model, fast and scalable deployment.

The application is also designed so that for every update on the model design in the drag-and-drop interface, the validation component checks the correctness of the whole model.

The user test designed by the authors comprised three distinct tasks, each targeting the creation of a unique Deep Learning model. This was achieved by employing the provided layers through plain coding as well as a visual method involving the addition and connection of nodes.

Upon concluding the experiments, the researchers highlighted that the average duration for coding significantly exceeds that of using visual tools for designing Deep Learning models. They also noted that the visual interface is well-adapted for users of varying expertise, although the time required to construct a model is contingent upon the individual technical abilities of the participants.

2.4 DeepVisual

DeepVisual by C. Xie et al. [14] is an application for designing and deploying DL systems.

The goal of the research was to open the way to the technological innovations and help developers from diverse domains to start to take advantage of DL models. DeepVisual is another example of Visual Programming tool based on the drag-and-drop paradigm to create visually a Neural Network.

One important feature of DeepVisual is the Code pane which is a tab of the application in which is possible to visualize the generated code for the built network. Any modification to the model affects the Code pane which is dynamically updated to keep track of the changes. In the Code pane, the user is also able to export the generated code in various DL framework supported.

Users have the option to upload a file with code, after which the application's backend will take responsibility for generating the Neural Network's graphical structure and displaying it within the drag-and-drop interface.

2.5 UMLAUT

UMLAUT by E. Schoop et al. [6] is a tool to debug ML pipelines providing useful errors suggestions or link to online resources.

This tool attaches to the DL program runtime, running heuristic checks of model structure and behavior that encode the tacit knowledge of experts. UMLAUT then displays results of checks as error messages that integrate program context, explain

best practices, and suggest code recipes to address the root causes. The crucial concept is that UMLAUT assists users before the evaluation phase.

The errors are categorized in three classes: Warning, Errors and Critical depending on the impact on model performances. Error messages are presented in a web interface that tightly couples errors with visualizations of model output, linking root causes to their symptoms. Error messages include descriptions of their underlying theoretical concepts, and suggest potential debugging strategies to bridge theoretical and practical knowledge gaps. To translate these strategies into actionable changes, UMLAUT errors include code snippets which implement suggested fixes, outbound links to curated Stack overflow and documentation searches, and links to the suspect lines of code in source.

2.6 Model Tracker

ModelTracker by Amershi et al. [7] introduces an interactive visualization tool that integrates information from various traditional summary statistics and graphs. This tool not only displays example-level performance but also facilitates direct error examination and debugging.

A notable aspect of this research, which distinguishes it from previous works, is the incorporation of user feedback. One significant suggestion was to display not only the changed predictions but also the magnitude of these changes by directly comparing them with the previous results. This enhancement allows for a more nuanced understanding of model performance and aids in more effective debugging and model improvement. In addition to this, other user feedback that influenced the development of ModelTracker included better integration with other Machine Learning tools and platforms.

2.7 Discussion

The research of the existing literature and tools reveals several important insights that have influenced the development of the new version of DeepBlocks. The DeepBlocks desktop version served as the foundation for this work, and its core features, such as the creation of superblocks, custom loss function uploads, and model correctness verification, have been preserved in the new version to maintain consistency and functionality. These features are essential for users to efficiently construct, debug, and train Deep Learning pipelines, providing a balance between simplicity and depth in network design.

The Visual Blocks tool highlighted the importance of providing users with the freedom to explore and connect different blocks, which fosters creativity and deeper understanding of Machine Learning principles. This insight underscores the value

of an intuitive, flexible visual interface that caters to both novice and experienced users. The ability to compare model outputs and results in real-time, as emphasized by Visual Blocks, is crucial for iterative development and refinement of ML models.

DL-IDE's contribution to this work lies in its focus on the 'no-code' paradigm, which aligns with the goal of making Deep Learning accessible to a broader audience. The study presented in the article revealed that even experienced programmers find it challenging to implement DL models quickly. Therefore, integrating a drag-and-drop interface with automatic code generation not only accelerates the development process but also lowers the entry barrier for non-programmers. The interoperability framework highlighted by DL-IDE, particularly the use of ONNX for model conversion, has also been a consideration in the new version of DeepBlocks, ensuring that the tool can be integrated with various ML libraries and frameworks.

DeepVisual introduced the concept of dynamic code generation and visualization, which allows users to see and export the generated code corresponding to their visual model. This feature provides transparency and flexibility, enabling users to understand the underlying code structure and modify it if necessary. The idea of converting code into a graphical model representation is particularly intriguing and presents an opportunity for future enhancements in DeepBlocks, potentially allowing users to switch seamlessly between code and visual modes.

The UMLAUT tool brought forward the critical importance of advanced debugging features in Deep Learning models. The ability to run heuristic checks and provide context-aware error messages that include practical solutions and links to online resources is a powerful aid for both novice and expert users. This approach to debugging, where errors are not only identified but also contextualized with theoretical explanations and practical fixes, has influenced the design of error-handling features in the new DeepBlocks version.

Finally, the ModelTracker tool demonstrated the effectiveness of integrating user feedback into the development process. The ability to track changes in model performance and visualize the magnitude of these changes, as well as the importance of flexibility in integrating with other ML tools, has informed the design of the User Interface and the feedback mechanisms in DeepBlocks. This focus on user experience ensures that the tool remains adaptable and responsive to the needs of its users.

In summary, this discussion demonstrates how the insights and features from each of the researched tools and studies have significantly influenced the development of the new DeepBlocks version. By integrating best practices - such as flexible visual interfaces and debugging mechanisms - DeepBlocks is designed to enhance usability, accessibility, and flexibility in Deep Learning model development. The incorporation of these elements not only preserves the core strengths of the original DeepBlocks but also introduces innovative features, ultimately advancing the capabilities of Visual Programming in the Machine Learning domain. Additionally, features like

seamless code-to-graphical model conversion and enhanced interoperability across ML frameworks, as seen in some of the researched tools, present promising directions for future versions of DeepBlocks, further extending its utility and appeal.

Chapter 3

Materials and methods

This chapter presents the concept of Machine Learning (ML), the primary frameworks, the phases of ML pipelines, diverse ML methods, the fundamentals of Visual Programming, gRPC, and the theoretical concepts behind the techniques employed in application testing.

3.1 Machine Learning

This section provides a concise introduction to Machine Learning, examining both the ML pipelines and the methods involved as well as the main framework to code DL models.

3.1.1 Introduction

Machine Learning (ML) is a subset of Artificial Intelligence (AI) that concentrates on using data and algorithms to empower AI systems to mimic the way humans learn, thereby progressively enhancing their accuracy.

Machine Learning (ML), Deep Learning (DL), and Neural Networks (NNs) are distinct yet interconnected sub-fields of Artificial Intelligence. Neural Networks form a subset of Machine Learning, and Deep Learning is a specialized branch of Neural Networks.

Deep Learning algorithms are designed to process unstructured data in its native form, such as text or images. They autonomously identify the features that differentiate various data categories, reducing the need for human input and allowing for the utilization of vast datasets.

Neural Networks, or Artificial Neural Networks (ANNs), consist of layers of nodes, including an input layer, several hidden layers, and an output layer. Each node, or artificial neuron, is connected to others and possesses an associated weight

and threshold. If the output of a node exceeds the threshold value, the node activates and transmits data to the next layer. If not, it does not pass data forward.

The term "deep" in Deep Learning refers to the number of layers within a Neural Network. A NN with more than three layers, including the input and output layers, is considered a Deep Learning algorithm or Deep Neural Network [15].

In this thesis the focus is on the **Deep Neural Networks**.

3.1.2 Pipelines

A Machine Learning pipeline is a series of interconnected data processing and modeling steps designed to automate, standardize and streamline the process of building, training, evaluating and deploying ML models.

ML pipelines consist of multiple sequential steps that do everything from data extraction and pre-processing to model training and deployment. Figure 3.1 shows an high level view of the stages of a ML pipeline.

The data processed throughout the pipeline is a dataset which can consist of any kind of data type, like images, audio, text etc.

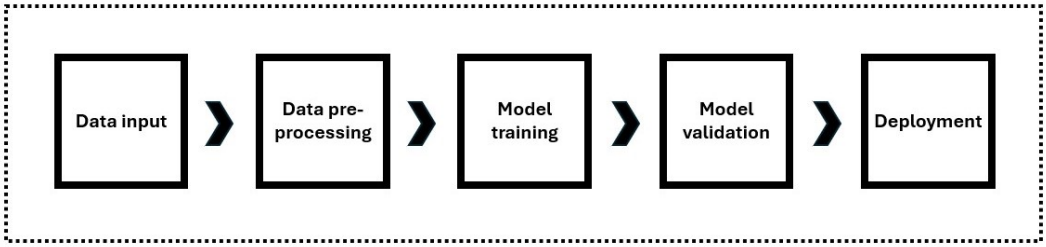

Figure 3.1: Stages overview of a ML pipeline

The first stage is the **Data Input** in which data is organized and processed so that can be used by the following steps.

Data pre-processing is an essential step because the input dataset can produce unexpected results during the training phase if not processed correctly. This stage involves various operations, including data cleaning, data fusion, data reduction, and data transformation. To prepare a final dataset for the training process, several methods are available for pre-processing data, such as normalization, aggregation, and numerosity reduction.

Model training is the process where the Machine Learning model receives data and begins the learning process. It is a fundamental phase in a ML pipeline, during which the model is trained to predict outcomes with the highest possible accuracy.

Model validation is the process of assessing a predictive model's performance and reliability by comparing its outputs against actual outcomes or using statistical techniques to ensure it generalizes well to unseen data.

The **Deployment** of the model involves integrating the trained model into a production environment where it can process real-world scenarios and provide predictions or insights to end-users.

3.1.3 Methods

ML methods fall in three main categories [16]:

- **Supervised Learning** is a type of Machine Learning methods where a model is trained on a labeled dataset, meaning that each training example is paired with an output label. The goal is for the model to learn the relationship between the input data and the corresponding labels so that it can make accurate predictions on new, unseen data. Common algorithms used in supervised learning include linear regression, decision trees, and Neural Networks.

  More details in the following paragraph.

- **Unsupervised Learning** employs Machine Learning algorithms to analyze and cluster unlabeled datasets. These algorithms uncover hidden patterns in the data autonomously, without human intervention. The capability of this method to identify similarities and differences in data makes it well-suited for exploratory data analysis, cross-selling strategies, customer segmentation, as well as image and pattern recognition.

- **Semi-supervised Learning** provides a middle ground between supervised and unsupervised learning. It employs a smaller set of labeled data during training to guide the classification and feature extraction from a larger set of unlabeled data.

Supervised Learning

Supervised Learning is a fundamental approach in ML where a model is trained using a labeled dataset. In this context, "labeled" means that each training example is paired with an output label, which serves as the ground truth for the model to learn from.

The primary objective of Supervised Learning is to learn a mapping from inputs to outputs, enabling the model to make predictions on new, unseen data. This process involves feeding the model a set of input features along with their corresponding labels, allowing it to identify patterns and relationships within the

data. Once trained, the model can then be evaluated on a separate test dataset to assess its performance and generalization capabilities.

The Supervised Learning training process is chosen based on the problem type. The two main type of problems are:

- **Classification** problems involve algorithms that accurately categorize test data into distinct groups. For instance, they can differentiate between apples and oranges. In practical applications, supervised learning algorithms are employed to filter spam emails into a designated folder, separate from the main inbox.

  Common classification algorithms include linear classifiers, support vector machines, decision trees, and random forests.

- **Regression** in supervised learning is a statistical method used to model the relationship between a dependent variable and one or more independent variables. It aims to predict continuous outcomes by fitting a mathematical function to the observed data. Common techniques include linear regression, which assumes a linear relationship, and more complex methods like polynomial regression and support vector regression.

During the training phase, the model learns from the labeled data by adjusting its parameters to minimize the difference between its predictions and the actual labels. Finally, the model's performance is evaluated using various metrics, such as accuracy, precision, recall, and F1 score, to determine how well it can predict outcomes on unseen data.

Supervised learning is widely applied across various domains, including finance, healthcare, marketing, and Natural Language Processing.

Furthermore, it plays a crucial role in image recognition tasks, where models are trained to classify images into predefined categories. The versatility and effectiveness of supervised learning make it a cornerstone of modern ML applications.

3.1.4 Main Libraries and Frameworks

In the rapidly evolving field of Machine Learning (ML), various libraries and frameworks have been developed to streamline the process of building, training, and deploying models. These tools provide developers with pre-built modules, optimized algorithms, and scalable solutions, thereby reducing the complexity of ML development.

PyTorch

Developed by Facebook's AI Research lab, PyTorch[1] is a prominent open-source Deep Learning framework. PyTorch has gained popularity for its dynamic computation graph, which allows for real-time model definition and modification, making it particularly suited for research and development. It offers a user-friendly interface, strong support for GPU acceleration, and seamless integration with Python, making it accessible to both beginners and experienced ML practitioners. PyTorch's ecosystem includes tools such as TorchVision for computer vision tasks and TorchText for Natural Language Processing, further extending its utility across various domains [17].

ONNX

The Open Neural Network Exchange (ONNX)[2] is an open-source format designed for the interchangeability of Deep Learning models between different frameworks. Developed by Facebook and Microsoft, ONNX allows developers to train a model in one framework (e.g., PyTorch) and then transfer it to another framework (e.g., TensorFlow) for inference. This interoperability is crucial in scenarios where different tools or platforms are better suited for different stages of the ML pipeline. ONNX is also supported by a wide range of hardware vendors, making it easier to deploy models across diverse environments.

Hugging Face Transformers

Hugging Face's Transformers library[3] has become a go-to resource for working with state-of-the-art Natural Language Processing (NLP) models. It provides pre-trained models for tasks such as text classification, translation, summarization, and more, built on architectures like BERT, GPT, and T5. However, the library's capabilities extend beyond NLP and have increasingly embraced computer vision tasks as well.

For image-related tasks, Hugging Face Transformers supports a variety of cutting-edge models such as Vision Transformers (ViT), DeiT (Data-efficient Image Transformers), and CLIP (Contrastive Language-Image Pretraining). These models enable powerful image classification, object detection, and image-text alignment capabilities. The Vision Transformer (ViT) models treat images as sequences of patches and apply the Transformer architecture, originally designed for NLP, to

[1] `https://pytorch.org/`

[2] `https://onnx.ai/`

[3] `https://huggingface.co/docs/transformers/index`

these sequences. This approach has shown competitive performance compared to traditional Convolutional Neural Networks (CNNs), especially when trained on large datasets.

In addition to pre-trained models, the Hugging Face ecosystem provides tools for fine-tuning these models on custom datasets, making it accessible for developers to adapt these powerful models to their specific use cases. The library also supports seamless integration with other tools and platforms within the Hugging Face ecosystem, such as the 'datasets' library, which simplifies the process of loading and pre-processing large image datasets.

Furthermore, Hugging Face's Transformers library includes utilities for image pre-processing, such as resizing, normalization, and augmentations, which are essential steps in preparing images for model training and inference. The library also offers straightforward APIs for converting images into the appropriate input formats required by the Transformer models.

The Hugging Face platform also includes a model hub, where users can explore, share, and deploy image models. This hub allows users to upload their trained models and make them available to the broader community, fostering collaboration and accelerating the development of image-related AI applications [17].

TensorFlow

TensorFlow[4] is an open-source Machine Learning framework developed by Google. It is highly versatile, supporting a wide range of tasks, including Neural Network training, Deep Learning, and even reinforcement learning. TensorFlow is particularly known for its scalability, making it suitable for both small projects and large-scale production systems. It offers a comprehensive ecosystem of tools, such as TensorBoard for visualization and TensorFlow Serving for deploying models. TensorFlow also supports both high-level APIs like Keras, which simplify model building, and low-level operations, providing fine-grained control over model architecture and optimization [17].

Scikit-learn

Scikit-learn[5] is a powerful library built on top of NumPy, SciPy, and Matplotlib, providing simple and efficient tools for data mining and data analysis. It is one of the most popular libraries for implementing traditional Machine Learning algorithms such as classification, regression, clustering, and dimensionality reduction. Scikit-learn is designed with an emphasis on simplicity and accessibility, making it an

[4] `https://www.tensorflow.org/`

[5] `https://scikit-learn.org/stable/index.html`

ideal choice for those new to ML. It also supports various pre-processing techniques, model evaluation methods, and pipelines, facilitating end-to-end ML workflows [17].

Keras

Keras[6] is a high-level neural networks API, written in Python, that runs on top of TensorFlow, Theano, or CNTK. It is designed to enable fast experimentation with deep neural networks, allowing users to build and train models with just a few lines of code. Keras emphasizes modularity, minimalism, and extensibility, providing a simple interface for defining complex models. Due to its ease of use and focus on user experience, Keras has become a preferred choice for those who want to quickly prototype models without delving into the complexities of the underlying frameworks [18].

CAFFE

Caffe[7], short for Convolutional Architecture for Fast Feature Embedding, is a Deep Learning framework developed by the Berkeley Vision and Learning Center (BVLC). Known for its speed and modularity, Caffe is particularly well-suited for tasks that require Convolutional Neural Networks (CNNs), such as image classification and object detection.

One of the key strengths of Caffe is its efficiency. It is written in C++ and supports CUDA, allowing it to leverage GPU acceleration for faster training and inference. This makes Caffe an attractive choice for deploying Deep Learning models in real-time applications, where performance is critical.

Caffe's architecture is highly modular, allowing users to define complex neural networks by assembling layers such as convolution, pooling, and activation functions. The framework also provides a "Model Zoo", a collection of pre-trained models that can be used out-of-the-box for a variety of tasks or fine-tuned for specific applications. This library includes models like AlexNet, VGGNet, and GoogLeNet, which have been pivotal in advancing the field of computer vision [19].

Although Caffe is highly efficient, it has some limitations. The framework does not natively support dynamic Neural Network structures, making it less flexible compared to other modern frameworks like PyTorch or TensorFlow. However, its speed and ease of use continue to make it a valuable tool for specific use cases, particularly in industry settings where performance is paramount [20].

[6] `https://keras.io/`

[7] `https://caffe.berkeleyvision.org/`

Apache Spark

Apache Spark[8] is an open-source distributed computing system that provides an efficient platform for big data processing and Machine Learning. Its ability to process large datasets in parallel makes it an invaluable tool for Machine Learning applications that require handling massive amounts of data.

Spark's MLlib, the Machine Learning library built on top of Spark, offers a range of scalable algorithms for classification, regression, clustering, collaborative filtering, and more. It also includes utilities for data processing, feature engineering, and model evaluation, making it a comprehensive tool for developing Machine Learning pipelines.

One of the significant advantages of Apache Spark is its integration with other big data tools, such as Hadoop, Hive, and HBase, allowing it to easily process data stored in a variety of formats and locations. This integration is crucial for Machine Learning applications that operate on large datasets distributed across multiple storage systems.

Spark also supports a wide range of programming languages, including Java, Scala, Python, and R, making it accessible to a broad audience of data scientists and engineers.

For Deep Learning tasks, Apache Spark can be integrated with other frameworks such as TensorFlow and Keras, enabling distributed training of Neural Networks across a cluster of machines [21].

Conclusion

The libraries and frameworks discussed in this section represent the cutting edge of tools available for Machine Learning development. Each offers unique features and advantages, catering to different aspects of the Machine Learning life-cycle, from data pre-processing and model training to deployment and inference. By leveraging these tools, developers and researchers can accelerate their workflows, achieve higher performance, and focus more on innovation rather than the underlying complexities of ML implementation.

During this project, the PyTorch library, the ONNX format and the HuggingFace Transformers were used.

3.2 Visual Programming

End-User Development (EUD) has emerged as a field that is concerned with tools and activities allowing end users who are not professional software developers

[8] `https://spark.apache.org/`

to write software applications. One advantage is that as end users, they know their own domain and needs more than anyone else, and they are often aware of specificities in their respective contexts.

Visual Programming, among other EUD techniques, allows end users to create a program by piecing together graphical elements rather than textually specifying them.

According to the analysis given by the paper Characterizing Visual Programming Approaches by Kuhail et al. [22], the VPL can be split in four categories:

- **Block-based** VP enables users to construct programs by dragging and dropping "blocks" of code from a predefined command list into a development space. These blocks fit together to form a complete program. This method eradicates syntax errors, thus lessening the cognitive load on users and enabling them to focus on conceptual comprehension instead of the intricacies of code syntax.

  One very popular tool which can serve as an example is Scratch[9].

- **Icon-based** languages leverage icons, which are graphical symbols that represent objects or actions.

- **Form-based** VP enables developers to design a form by adding triggers and actions through text-based drop-down menus or visual drag-and-drop interfaces. While some form-based methods are predominantly visual, others incorporate textual specifications.

- **Diagram-based** VP involves the connection of graphical elements, such as boxes, with arrows, lines, or arcs to denote relationships. Understanding a diagram-based program requires users to navigate through the diagram, which employs various forms of perceptual coding to depict the program's flow.

In this thesis, the objective is to leverage the use of the diagram-based VP to build a Deep Neural Network.

3.3 gRPC

Like many RPC systems, gRPC is based around the idea of defining a service, specifying the methods that can be called remotely with their parameters and return types. By default, gRPC uses protocol buffers, which are language-neutral mechanism serializing structured data [23], as the Interface Definition Language (IDL) for describing both the service interface and the structure of the payload

[9] `https://scratch.mit.edu/`

messages. gRPC can define four kinds of service method: **Unary** RPCs where the client sends a single request to the server and gets a single response back. **Server streaming** RPCs where the client sends a request to the server and gets a stream to read a sequence of messages back. The client reads from the returned stream until there are no more messages. gRPC guarantees message ordering within an individual RPC call. **Client streaming** RPCs where the client writes a sequence of messages and sends them to the server, again using a provided stream. Once the client has finished writing the messages, it waits for the server to read them and return its response. Again gRPC guarantees message ordering within an individual RPC call. **Bidirectional streaming** RPCs where both sides send a sequence of messages using a read-write stream. The two streams operate independently, so clients and servers can read and write in whatever order they like: for example, the server could wait to receive all the client messages before writing its responses, or it could alternately read a message then write a message, or some other combination of reads and writes. The order of messages in each stream is preserved [24].

In this thesis project, gRPC was selected due to its simplicity, efficient communication structure, and ease of scalability. Its lightweight nature enables fast and reliable communication between services, while its support for multiple programming languages ensures flexibility across different platforms. Additionally, gRPC's ability to easily add new services without major architectural changes makes it highly scalable, allowing for seamless expansion and integration as the project grows. This combination of simplicity and scalability makes gRPC an ideal choice for building a robust and future-proof service architecture. Among the several service methods, the *Unary* has been used.

The language used for handling the communication is Python through its `grpcio-tools`[10] package, and the protocol buffer language has been used to define the server interface and the structure of the payload messages.

3.4 Testing techniques

There are many ways to test an application and its interface. The main categories to distinguish are the **Between-subjects** where different people test each condition, so that each person is only exposed to a single user interface, or the **Within-subjects** where the same person tests all the conditions [25].

The *between-subjects* design offers several advantages in usability studies. First, it minimizes learning and the transfer of knowledge between conditions. For example, when a participant completes a series of tasks on a car-rental site, they become familiar with the domain, such as learning about fees for drivers under

[10] `https://pypi.org/project/grpcio-tools/`

21 or insurance coverage. However, in a *between-subjects* design, this knowledge transfer is not a problem since participants are not exposed to multiple levels of the same independent variable. Another benefit is the shorter session duration: testing a single application results in shorter sessions, which are less tiring for participants and more suitable for unmoderated remote testing. Finally, this design is easier to set up compared to *within-subjects* design, which requires randomization of stimuli to avoid order effects [25].

The *within-subjects* design presents several significant advantages. First, it requires fewer participants to achieve statistically significant results. Since each participant provides data for each level of the independent variable, a single participant can contribute multiple data points. For example, in a study comparing two car-rental sites, 40 participants will generate data for both sites, while a *between-subjects* design would need twice the number of participants, increasing the cost. Thus, *within-subjects* studies are more cost-effective. Additionally, the *within-subjects* design minimizes noise in the data, reducing the likelihood that real differences between conditions are obscured by random variability. Participants bring their own history, knowledge, and emotional state to the test. For example, a happy participant will affect all conditions similarly, whereas in a *between-subjects* design, such an effect might impact only one group, requiring additional effort to balance such variables between groups. In practice, controlling for these factors across groups is difficult, making *within-subjects* design more robust to such variations [25].

3.4.1 SEQ Questionnaire

Single Ease Question (SEQ) is a standard user experience metric that researchers use in usability studies to figure out how hard or easy it is for user attempts to do a task. It lets you get more qualitative information and determine how users feel about the task. Instead of measuring how well an interface or prototype is used, SEQ measures how hard the users respond differently and think it is to do a specific task [26].

SEQs offer invaluable benefits to user experience research. They enable to pinpoint problematic areas within the tested interface or workflow by collecting user feedback immediately after each task. This real-time insight helps identify the most challenging aspects, allowing to prioritize improvements effectively. Additionally, SEQs provide fresh and unbiased insights, as participants' experiences are still vivid in their minds right after completing a task. This timeliness minimizes the risk of other tasks influencing their memory or distorting their perception, leading to more accurate feedback on the user experience [27].

3.4.2 SUS Questionnaire

The System Usability Scale (SUS) is a widely used 10-item questionnaire designed to assess the usability of a system. Each item is rated on a 5-point Likert scale ranging from 1 (Strongly Disagree) to 5 (Strongly Agree). A Likert scale is a rating scale used to measure opinions, attitudes, or behaviors. It consists of a statement or a question, followed by a series of five or seven answer statements. Respondents choose the option that best corresponds with how they feel about the statement or question. Because respondents are presented with a range of possible answers, Likert scales are great for capturing the level of agreement or their feelings regarding the topic in a more nuanced way [28].

The questionnaire covers a range of usability aspects, with alternating positive and negative statements to ensure balanced feedback. The questions are as follows:

- I think that I would like to use this system frequently.
- I found the system unnecessarily complex.
- I thought the system was easy to use.
- I think that I would need the support of a technical person to be able to use this system.
- I found the various functions in this system were well integrated.
- I thought there was too much inconsistency in this system.
- I would imagine that most people would learn to use this system very quickly.
- I found the system very cumbersome to use.
- I felt very confident using the system.
- I needed to learn a lot of things before I could get going with this system.

SUS has a strong record of consistently being a reliable and valid tool for measuring usability. It's shown to effectively provide valuable insights to smaller sample-size audiences, making them easier to leverage than larger commercial techniques. SUS has also proven to consistently provide valid measurements around perceived usability, meaning any researcher or team can trust the results [29].

Scoring System

The scoring system for SUS involves adjusting user responses to ensure consistency between positive and negative statements. For odd-numbered items, subtract one from the user's response, while for even-numbered items, subtract the user's response from 5. This method scales all values to a range from 0 to 4, where a score of 4 represents the most positive usability rating.

After adjusting the responses, sum the total for each user. To convert this total to a standardized score, multiply the result by 2.5. This transformation scales the score to a range of 0 to 100, with higher scores indicating better usability. According to established benchmarks, a SUS score above 68 is considered above average in terms of usability [30].

Chapter 4

Design

This chapter outlines the design process for the user interface of the new version of DeepBlocks.

Previously, DeepBlocks was developed as a desktop application by Tommaso Calò and Luigi de Russis [11]. Building upon the features implemented in the desktop version, the design phase for the new version was organized into three key steps. First, a review of existing literature was conducted to identify similar applications and extract useful ideas and features (see Section 2). Next, the appropriate architecture for the web application was selected. Finally, the user interface (UI) was designed using Figma, a collaborative interface design tool [31].

4.1 Features design

The design of the new features was carried out by evaluating the existing features in the desktop version of the DeepBlocks application and by exploring the literature for new functionalities to implement in the upcoming version of the app.

4.1.1 From the desktop version

The concept began with the creation of a Visual Programming tool aimed at constructing, debugging, and evaluating Deep Neural Networks. The core objective was to preserve the distinctive features that define the application, ensuring it remains innovative and beneficial for its intended users. The primary functionality involves defining the network by adding, connecting, and parameterizing blocks. A novel feature introduced in DeepBlocks is the capability to forge a Superblock, which encapsulates a variety of heterogeneous blocks within. You can see an example of the Superblock design in Figure 4.1.

An essential feature of the application is pre-training network verification, which

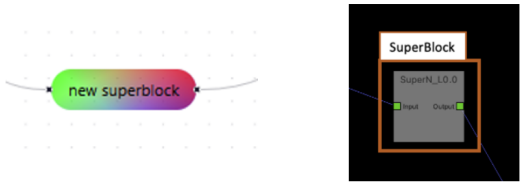

Figure 4.1: Superblock design comparison: the new version is depicted on the left, while the previous one is shown on the right.

identifies and highlights erroneous nodes in the User Interface, serving as an instructive guide for users. In the previous version of DeepBlocks, errors were indicated by highlighting the branch where the issue originated. In the current version, the specific node causing the error is circled with a red dashed line, and an error message is displayed. This enhancement helps users quickly identify and resolve issues, as shown in Figure 5.2.

One feature of the desktop version retained in the new version of the application is the organization of the training settings menu. This menu includes fields for setting the number of epochs, learning rate, batch size, optimizer, and the loss function for the training process. Regarding the loss function, users can upload a file with a custom loss function or select from a list of predefined ones, such as Cross Entropy (CE), Binary Cross Entropy (BCE), and Mean Squared Error (MSE) loss functions. For the optimizer, two options are available: the Adam optimizer and Stochastic Gradient Descent (SGD).

Compared to the previous version, the new release has expanded the functions available for users to build their DL model. Additionally, the node design displayed on the canvas has been updated to clarify its function. As illustrated in Figure 4.2, the left side shows the new block design, labeled with the PyTorch function, whereas the older version displays a node named after a Fully Connected (FC) block.

4.1.2 From the literature

The literature research pinpointed many interesting functionalities that could match with the scope of DeepBlocks.

Some suggestions were reserved for potential inclusion in future versions of DeepBlocks, while others were considered during the current project's implementation phase. Among those assessed for this version was the proposal of a resolution phase that could be based on standard error or utilize online resources when the server returns an error in the model.

Another aspect underscored by numerous scientific studies is the potential to

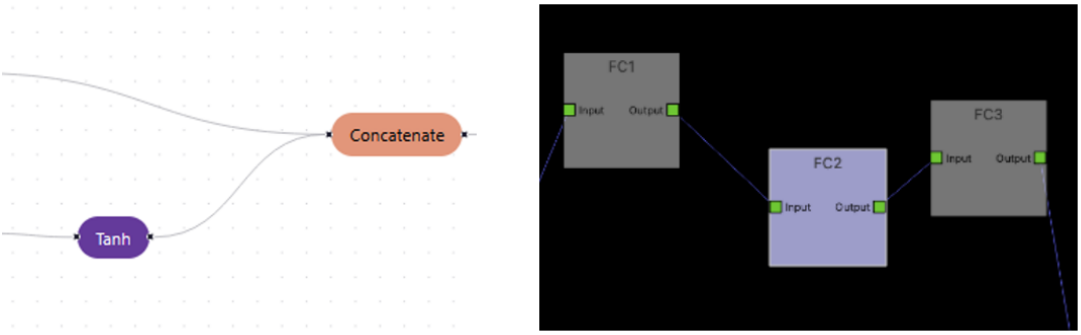

Figure 4.2: To the left is the new block design, and to the right is the older version.

develop a flexible system that can also extend compatibility to other frameworks. In this regard, the targeted open system for constructing DL models that can be utilized across various frameworks is the Open Neural Network Exchange (ONNX) format. Users can export their models in ONNX format and then employ them in multiple frameworks to alter the training process or evaluation phase or visualizing the model graphically on the Netron website[1].

The final feature evaluated during the project's design phase was the ability to compare current results with those from previous models when there have been changes in node parameters. This allows users to swiftly determine whether their parameter adjustments have effectively enhanced the model's overall performance.

4.2 Architecture

To determine the most suitable architecture model for the second version of DeepBlocks, a comprehensive evaluation of all potential options was conducted. After an initial round of deliberations, the decision to continue with a desktop application was discarded. This choice was made due to the inherent limitations of desktop applications, including restricted accessibility, challenges in maintaining up-to-date functionalities, and the difficulty in providing seamless user experiences across different platforms.

The final architecture selected for DeepBlocks is a web-based application utilizing a three-tier model. This architecture offers enhanced scalability, accessibility, and ease of maintenance, ensuring that users can access the application from anywhere with an internet connection. The three tiers of the architecture are as follows:

[1] `https://netron.app/`

- Web User Interface (UI): This tier handles the frontend logic and is responsible for managing user interactions. It is designed to provide a user-friendly and responsive interface that facilitates smooth communication between the user and the application. It also includes the logic to download the DL model in JSON format and to upload a JSON file that contains a model.

- Middleware Service: Acting as an intermediary layer, the middleware service connects the frontend with the backend. It processes incoming requests, performs optional data validation and checks, and manages access to external resources and services. This tier ensures efficient communication and data flow between the UI and the backend.

- Backend: This tier is responsible for the core functionality of building and training DL models. It manages the computational processes and algorithms necessary for DeepBlocks to deliver accurate and reliable results. The backend is designed to be scalable and robust, allowing for the efficient handling of complex computations and large datasets.

This three-tier architecture not only enhances the overall performance and reliability of DeepBlocks but also provides a flexible foundation for future updates and feature expansions. A schematic representation of this architecture is illustrated in Figure 4.3, showcasing the interaction between the different components of the system.

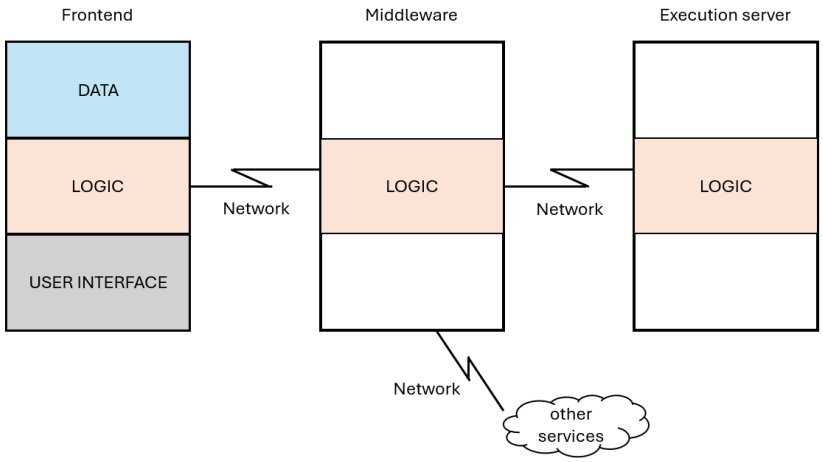

Figure 4.3: DeepBlocks' web architecture

By adopting this web-based three-tier architecture, DeepBlocks is well-positioned to meet the evolving needs of its users, offering a modern solution that is both powerful and adaptable.

4.3 UI design

After finalizing the application's architecture, the next step involved creating a Medium-Fidelity prototype (Med-Fi). This prototype served as a preliminary design, featuring a series of interconnected screens that provide a clear visual representation of the application's layout and flow. Although not finalized, this prototype offers a more detailed view than low-fidelity sketches, as it permits to understand the overall structure and functionality of the application [32]. The focus was on usability and user interaction, ensuring that the design concept aligns with the intended user experience.

Building on the features identified in the previous version of the application, as referenced in Section 2.1, and those found in other scientific works detailed in Section 2, an initial set of requirements has been established. Subsequently, given that the selected architecture is web-based, the design has been carried out with consideration for a web browser environment.

The DeepBlocks web application's design was crafted using Figma [31], a web-based tool for creating prototypes. The web UI was conceptualized to mirror the layout of an Integrated Development Environment (IDE) used for coding.

The DeepBlocks' interface includes a vertical menu bar on the left side with icons, providing navigation through the app's various menus: a selection of blocks to add to the network (N), a list of already incorporated blocks with available operations (V), a training menu for setting parameters and initiating the training process (T), a separate menu for import/export tasks (S), and an icon to access the training results window (A). The Medium-Fidelity design of the DeepBlocks application, showcasing the left menu and the open list of available blocks, is depicted in Figure 4.4.

Nodes are represented as compact blocks on the interface, ensuring clarity and a degree of scalability with growing network sizes. Superblocks, introduced in Section 2.1, further enhance scalability. Upon creating and opening a Superblock, a new tab is generated at the top of the webpage, enabling seamless navigation. This tab allows the user to effortlessly switch between the Superblock and the main tab, where all individual blocks and Superblocks are displayed. This functionality enhances the user experience by providing a streamlined way to access and manage different components of the application.

Users have the flexibility to personalize block names by relabeling them.

The web UI features zoom controls located at the bottom-right, allowing users to zoom in, zoom out, and fit the entire network within a single screen view as efficiently as possible.

To configure any node parameters, the user can open the node bar which is displayed in the bottom part of the screen. This interface provides a description of the node, displays any specific parameters that can be adjusted for the specific

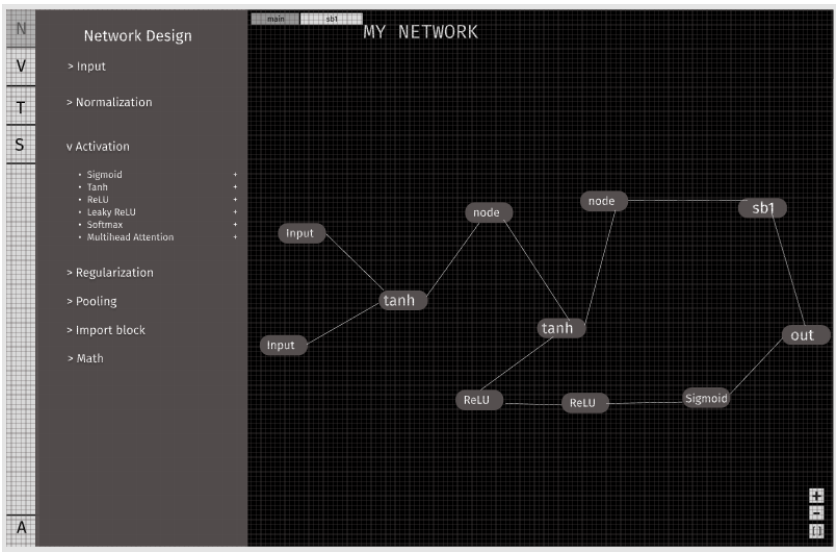

Figure 4.4: DeepBlocks Med-Fi prototype: vertical menu bar on the left side with icons and the open list of available blocks with an example network

PyTorch function, and includes a tab for error messages if any issues arise. This setup ensures that users have all the necessary information and tools to effectively manage node configurations. Figure 4.5 depicts the design of this component.

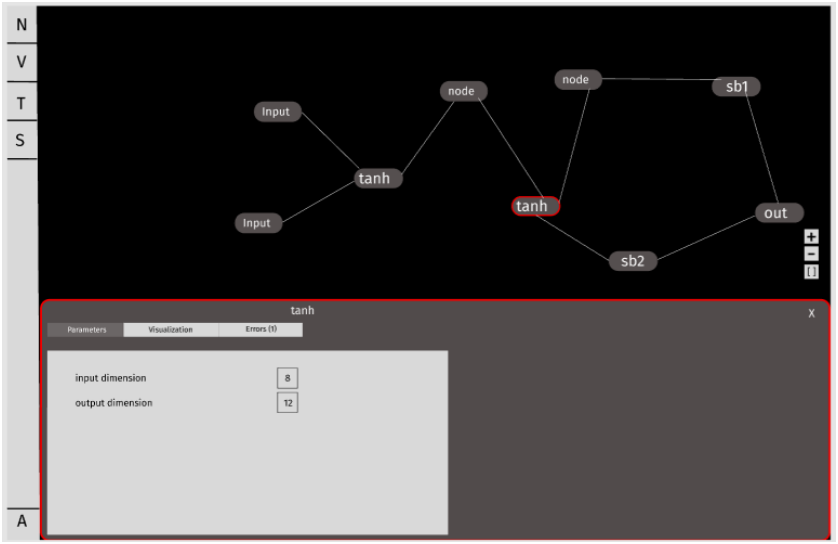

Figure 4.5: DeepBlocks Med-Fi prototype: node info bar to set parameters and display errors, if any

The menu displaying the training results is designed to show the status of the

training process and includes controls for managing execution. The training results are presented in a tabular format, with key metrics such as accuracy, precision, recall, and F1-score visualized through tables and plots. This approach provides a clear and concise overview of the model's performance. Figure 4.6 shows the design of that component.

Figure 4.6: DeepBlocks Med-Fi prototype: a results menu featuring various plots and corresponding metric values

Chapter 5

Implementation

In this chapter, the various components of the application are detailed, including the implementation choices, the libraries utilized, and the rationale behind the development of each component of the web application. This section is crucial as it provides insights into the technical architecture and the decision-making process that underpins the overall functionality of the application.

The source code of the application is available on GitLab at the following link `https://git.elite.polito.it/filippocaste/deepblocks-msthesis.git`.

5.1 Architecture

This section provides a comprehensive overview of the application's architecture from a technical perspective. The architecture is composed of three main components: the frontend, the middleware service, and the backend server responsible for building and training Deep Learning models. These components work in tandem to deliver a seamless user experience, from interacting with the Graphical User Interface (GUI) to executing complex Deep Learning tasks.

Frontend The frontend is the user-facing layer of the application, responsible for rendering the Graphical User Interface (GUI) and managing user interactions. The frontend has been developed using the React framework, which is known for its efficiency in building dynamic and responsive User Interfaces.

Middleware The middleware serves as a bridge between the frontend and the backend, facilitating communication and ensuring that requests from the frontend are appropriately handled and routed to the backend services.

Backend The backend is the core of the application, responsible for the heavy lifting involved in building, training, and managing Deep Learning models. It exposes a range of services through a gRPC interface, allowing the middleware to invoke these services as needed.

5.1.1 Data Management

One of the unique aspects of the application is that it does not incorporate user account management or persistent data storage. As a result, the data handled by the application is transient and session-based. The main data components include the available building blocks and PyTorch functions that users can utilize to construct their models. This data is stored directly in the frontend, making it readily accessible for user interactions.

Future Enhancements

In future iterations of the application, it might be beneficial to incorporate user accounts and persistent storage. This would allow users to save their work, share models with others, and return to their projects at a later time without the need to download the model and later uploading it. Additionally, implementing a database to store user data could open up new possibilities, such as tracking user progress, and providing analytics on the types of models users are building.

5.2 Communication Flow

The communication flow between the frontend, middleware, and backend is designed to be efficient and robust. Each component communicates with the others using well-defined protocols and data formats, ensuring that information is transferred accurately and quickly.

The frontend communicates with the middleware using API calls. These calls are made using HTTP, and the data is serialized in JSON format or sent as binary data.

The middleware uses gRPC to communicate with the backend. This choice was made to leverage gRPC's performance advantages, such as lower latency. gRPC also allows for more complex data structures to be transferred between the middleware and backend, which is essential for the complex tasks handled by the backend.

5.2.1 Error Handling

The communication flow includes robust error-handling mechanisms. For example, if the backend encounters an issue during model training, it sends an error message

back to the middleware, which then relays this information to the frontend. The frontend can then display an appropriate error message to the user, providing guidance on how to resolve the issue.

5.3 Frontend

5.3.1 Folder organization

The Figure 5.1 illustrates the organizational structure of the repository's directory that contains the frontend code.

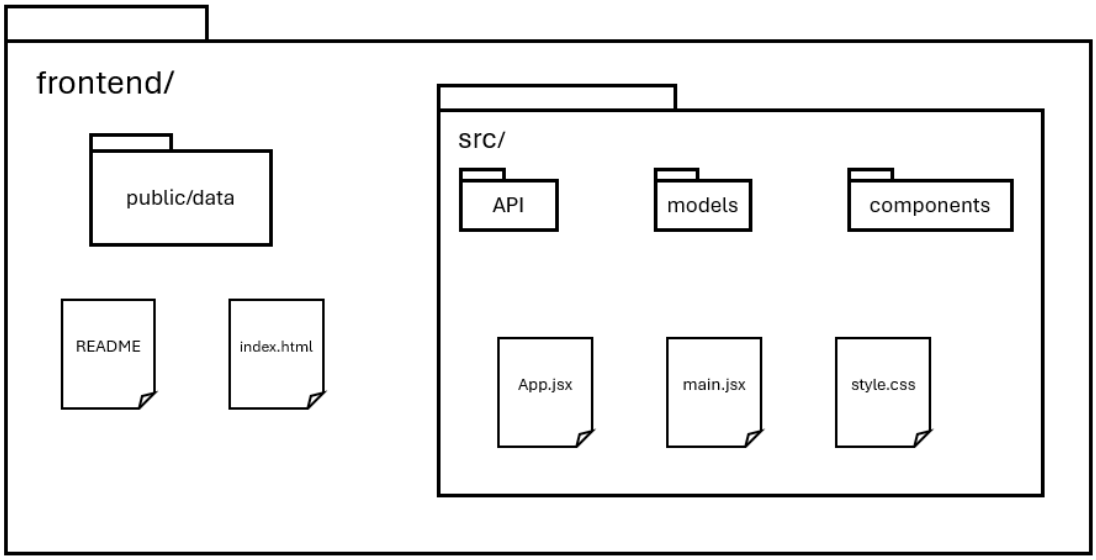

Figure 5.1: The folder structure for the frontend implementation, including the main sub-folders and files.

The `frontend` directory is structured to efficiently organize the web application's components and resources. At the root level, it contains essential files such as `index.html` for the main webpage structure, `package-lock.json` for managing dependencies, and `README.md` for documentation purposes. Within the `public` folder, there is a sub-directory `data`, which stores static data resources accessible by the application. The `src` folder is the core of the project, housing various sub-directories: `API`, which contains the API logic and service calls; `components`, which includes reusable UI components; and `models`, which holds data models used throughout the application which are mainly the `Blocks.js` and `SuperBlock.js` files. Additionally, the `src` directory contains key files like `App.jsx`, the main application component; `style.css`, which manages the application's styling; and

`main.jsx`, which serves as the entry point for the JavaScript code. This structure is designed to keep the codebase modular and maintainable, facilitating development and future enhancements.

5.3.2 Functionality

The frontend is more than just a static interface; it plays a pivotal role in the overall workflow. Users can construct Neural Network architectures by dragging and dropping various components onto the canvas. These components represent different layers or operations, such as convolutional layers, activation functions, or pooling layers. The frontend provides real-time feedback, allowing users to visualize the connections between different layers and observe how changes affect the overall architecture.

Additionally, the frontend offers the ability to create *superblocks*, which are composite blocks designed to encapsulate multiple blocks or even other superblocks. This feature enhances modularity and reusability within the Neural Network design.

The frontend also enables users to export the created model in various formats: as a PyTorch model, as an ONNX file, or as a JSON document. The JSON format is specifically designed to allow users to upload the model back into DeepBlocks in the future, enabling them to continue working on it. Since this functionality pertains exclusively to the frontend, the logic for converting the JavaScript objects created by the user is handled entirely within the frontend layer without involving the backend. However, for the conversion to ONNX format and the creation of the PyTorch model, backend processing is required.

Whenever an error is returned by the middleware server during network validation or training, it is the responsibility of the frontend to analyze the server's error message. The goal is to identify the node responsible for the error and adjust its style, enabling immediate visualization by the user as it is depicted in Figure 5.2.

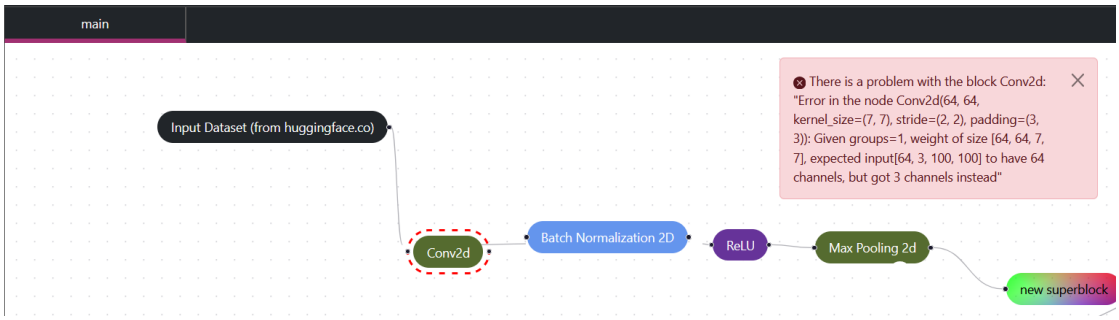

Figure 5.2: The node causing the error is highlighted, allowing the user to easily identify and correct it based on the error message

JSON Structure of a Node (Block)

The structure of a `Block` node in the JSON format includes the following fields:

- **id** (String): A unique identifier for the block, automatically generated as a string based on an incrementing counter.
- **type** (String): Specifies the type of the block, which determines its behavior and characteristics within the Neural Network architecture.
- **position** (Object): Contains information about the block's position within the architecture. This could include coordinates or other positioning details and it is a default parameter from Reactflow.
- **data** (Object): Stores additional data associated with the block. This includes:
  - **openInfo** (Boolean): Indicates whether the block's detailed information is currently open.
  - **isSelected** (Boolean): Indicates whether the block is currently selected.
- **parameters** (Object): Holds the specific parameters that define the block's configuration. These parameters vary depending on the type of block.
- **hidden** (Boolean): A flag indicating whether the block is hidden. By default, this is set to `false`.
- **fn** (Function): The PyTorch function associated with the block that defines its specific computational behavior.
- **description**: A brief description about the PyTorch function job.

The `Block` class includes a static method to update the ID counter:

- `static updateIdCounter(newIdCounter)`: Updates the internal counter used to generate unique IDs for new blocks.

JSON Structure of a Superblock

The structure of a `Superblock` in the JSON format includes the following fields:

- **id** (String): A unique identifier for the superblock, generated as a string with an appended 's' based on an incrementing counter.
- **type** (String): Specifies the type of the superblock, which categorizes its role within the architecture.

- **position** (Object): Contains information about the superblock's position. This could include coordinates or other positional attributes.
- **data** (Object): Stores additional data related to the superblock, including:
  - **openInfo** (Boolean): Indicates whether the superblock's detailed information is currently open.
  - **isSelected** (Boolean): Indicates whether the superblock is currently selected.
  - **isOpenInSheet** (Boolean): Indicates whether the superblock is open in a separate sheet or detailed view.
  - **hasSheet** (Boolean): Indicates whether the superblock has an associated sheet for detailed configuration.
- **children** (Array of Strings): An array of child block IDs associated with this superblock. Each child is referenced by its unique ID.

The `Superblock` class also includes a method to manage its children:

- `addChild(child)`: Adds a new child block to the superblock. The child is identified by its unique ID and appended to the `children` array.

Training parameters

The training parameters required to send the created model to the backend layers include the **number of epochs**, **learning rate**, **batch size**, **loss function**, and **optimizer**. The number of epochs defines how many times the entire training dataset will pass through the model during the training process. The learning rate is a crucial factor that controls how much the model's weights are adjusted with respect to the loss gradient during optimization. The batch size determines the number of training samples processed before the model updates its weights. The loss function is a critical element in training, as it measures the difference between the predicted output and the actual label. It can be chosen from options such as Binary Cross Entropy (BCE), Cross Entropy (CE), or Mean Squared Error (MSE). Alternatively, a custom loss function may be provided through an uploaded file containing a function named `custom_loss`. Lastly, the optimizer is responsible for updating the model's parameters to minimize the loss, with the options being the Adam optimizer or Stochastic Gradient Descent (SGD). Each of these parameters plays a fundamental role in determining the behavior and success of the training process [33].

Available blocks

The following blocks are the ones currently supported by the application developed during this project to create a DL model based on the PyTorch documentation [1].

- **Input Blocks**
  - Input Dataset (from huggingface.co)
    - Parameters:
      input_dataset: Input dataset from `huggingface.co/datasets` (e.g., `'dair-ai/emotion'`)
      dataset_type: Dataset type from `huggingface.co/datasets` (currently either `'text'` or `'image'`)
      dataset_config: Dataset configuration (e.g., `'default'`)
- **Normalization Blocks**
  - Batch Normalization 1D
    - Parameters:
      num_features: Number of features
      momentum: Momentum
      eps: Epsilon
      affine: Whether to use learnable affine parameters
  - Batch Normalization 2D
    - Parameters:
      num_features: Number of features
      momentum: Momentum
      eps: Epsilon
      affine: Whether to use learnable affine parameters
  - Layer Normalization
    - Parameters:
      normalized_shape: Shape of the tensor to be normalized
      eps: Epsilon
      elementwise_affine: Whether to apply affine transformation element-wise
- **Element-wise Operations**
  - Addition

[1] `https://pytorch.org/docs/stable/nn.html`

  - Subtraction
  - Multiplication
  - Division

- **Operations on Blocks**
  - Concatenate
  - Split

- **Reshaping Layers**
  - Flatten
    * Parameters:
      layer_type: Type of the layer
      start_dim: The dimension to start flattening (0-based index)
      end_dim: The dimension to stop flattening (exclusive)
  - Linear
    * Parameters:
      layer_type: Type of the layer
      in_features: Size of each input sample
      out_features: Size of each output sample
      bias: If set to False, the layer will not learn an additive bias

- **Regularization Layers**
  - Dropout
    * Parameters:
      kernel_size: Size of the convolving kernel
      p: Dropout probability

- **Activation Functions**
  - Sigmoid
  - Tanh
  - ReLU
  - Leaky ReLU
    * Parameters:
      negative_slope: Negative slope
  - Softmax

    * Parameters:
      dim: Dimension along which softmax will be computed
  - Multihead Attention
    * Parameters:
      embed_dim: Total dimension of the model input
      num_heads: Number of parallel attention heads
      dropout: Dropout probability
      bias: Whether to add bias to key and value sequences
      add_bias_kv: Whether to add bias to the key and value sequences
      add_zero_attn: Whether to add a new batch of zeros to key and value sequences at each forward call
      kdim: Total number of features in key
      vdim: Total number of features in value

- **Convolutional Layers**
  - Conv1d
    * Parameters:
      in_channels: Number of input channels (e.g., number of word embeddings for text data)
      out_channels: Number of output channels from the convolution
      kernel_size: Size of the convolutional filter
      stride: Stride of the convolution (step size)
      padding: Padding added to the input (0 for no padding)
      dilation: Spacing between elements in the filter
      bias: If False, then no bias term is added
  - Max Pooling 1d
    * Parameters:
      kernel_size: Size of the max pooling window
      stride: Stride of the pooling operation
      padding: Padding added to the input (0 for no padding)
      ceil_mode: If True, will use ceil instead of floor to compute the output shape
  - Average Pooling 1d
    * Parameters:
      kernel_size: Kernel size
      stride: Stride of the window
      padding: Implicit zero padding to be added on both sides

ceil_mode: Whether to use ceil instead of floor to compute the output shape

count_include_pad: Whether to include the zero-padding in the averaging calculation

- Conv2d
  - Parameters:

    in_channels: Number of input channels (e.g., number of word embeddings for text data)

    out_channels: Number of output channels from the convolution

    kernel_size: Size of the convolutional filter

    stride: Stride of the convolution (step size)

    padding: Padding added to the input (0 for no padding)
- Max Pooling 2d
  - Parameters:

    kernel_size: Size of the max pooling window

    stride: Stride of the pooling operation

    padding: Padding added to the input (0 for no padding)

    ceil_mode: If True, will use ceil instead of floor to compute the output shape
- Average Pooling 2d
  - Parameters:

    kernel_size: Kernel size

    stride: Stride of the window

    padding: Implicit zero padding to be added on both sides

    ceil_mode: Whether to use ceil instead of floor to compute the output shape

    count_include_pad: Whether to include the zero-padding in the averaging calculation

Invoking the *forwarding* process

The `useEffect` hook in `App.jsx` manages the validation and checking of a network model whenever certain dependencies change.Initially, the hook defines an array `paramObj` containing the current values of several parameters (`learningRate`, `epochs`, `batchSize`, `loss`, and `optimizer`). This array is used to send the parameters to the backend for validation.

A `debounceEffect` function is defined to handle the actual validation process. It starts by checking if the network is currently being trained and if there is an input node in the configuration. The input node is identified as a node of type `customNode` with a specific parameter named `'input_dataset'` and must be connected by an

edge. If the network is not being trained and the input node is present, the function verifies that all necessary parameters are set. If any parameters are missing or set to default values, a warning message is displayed to the user to set all parameters. The function then sends a request to the backend using `BLOCKS_API.forwardBlock` to validate the network configuration with the provided parameters. Upon receiving a successful response, the checking state is reset, and a success message is shown to the user. Any previous error states or messages are cleared.

In case of an error during the validation request, the function processes the error message to extract the problematic node's ID and its details. It also identifies if the node is part of a superblock and updates the error states accordingly. An error message is then displayed to the user, specifying the issue with the block and the associated node.

The hook uses a debounce mechanism to delay the execution of the `debounceEffect` function by 4 seconds. If another change occurs within this period, the previous timeout is cleared and reset to ensure that the validation is not triggered excessively. The cleanup function clears the timeout when the component unmounts or dependencies change, preventing potential memory leaks.

The implementation code can be found at `frontend/src/App.jsx`.

Invoking the training process

The `handleTrain` function is responsible for managing the initiation of a training session based on provided parameters. It begins by verifying that all necessary parameters are set and are of the correct type. Specifically, it checks whether the parameters `learningRate`, `epochs`, and `batchSize` are non-zero, the `loss` is not set to an empty string or 'Custom', and the `optimizer` is not empty. If any of these conditions are not met, the function will display an error message indicating that all parameters need to be filled. Next, the function checks whether the `learningRate`, `epochs`, and `batchSize` values are numeric. If any of these values are not numbers, an error message is generated, informing the user of the specific non-numeric parameter. The error state is set, and training is halted. If the parameters are valid, the function creates an object that holds key-value pairs representing each parameter. This object is prepared for submission to the backend.

If a custom loss function is defined, it is first uploaded to the backend via an API call. Once this is done (or if no custom loss is set), the function sends the training network's nodes, edges, and parameters to the backend using another API request. During this process, the function checks whether the network architecture has changed from previous sessions. If it has, the new network configuration and metrics are stored. If the network remains the same, the training metrics are appended to the previous session. The function then informs the user that training has completed and displays the results in the dedicated menu tab. In case of an

error during the API call, appropriate error messages are shown, and the training process is stopped.

If the initial pre-training check fails, the user is prompted to fix the errors before proceeding. If any required parameters are missing, the user is notified to complete the parameter input.

The code for this function is located at `frontend/src/components/Sidebar.jsx`

Duplicating a node

The `handleDuplicateNode` function is responsible for creating a duplicate of a given node in a node graph. This function handles two cases: duplicating a `superBlockNode` and duplicating a standard `Block`. The process for each case is as follows:

For a node of type `superBlockNode`, the function first identifies and collects the IDs of its child nodes, excluding those with 'i' (input) or 'o' (output) in their IDs. It then retrieves the corresponding child nodes from the `nodes` array.

The function creates new blocks for each child node by copying their type, position, data, parameters, and function. Each new block is marked as hidden to limit the visibility only in the case of the superblock sheet is opened and a mapping of original to new IDs is maintained. These new blocks are collected into an array.

Next, a new `Superblock` is created with updated properties. It includes a modified position and a label indicating it is a copy of the original. Two invisible nodes, an `invisibleInputNode` and an `invisibleOutputNode`, are also created and added to the new superblock. The IDs of these invisible nodes are appended to the list of children for the new superblock, and they are included in the nodes array with hidden status.

The function then copies the edges associated with the original children nodes. It maps the original edge sources and targets to the new block IDs, replacing any references to the invisible nodes accordingly. The new edges are added to the existing edges in the graph.

For a node that is not of type `superBlockNode`, the function creates a new `Block` with a similar process: duplicating its type, position, data, parameters, and function, while updating the label to indicate it is a copy. If there is an open superblock in the graph, the new block is added as a child of this superblock and made visible.

Finally, the function updates the node graph by adding the newly created nodes (blocks and/or superblocks) to the existing list of nodes.

The implementation code is located at `frontend/src/App.jsx`.

Eliminating nodes

The `handleDeleteNodes` function manages the deletion of nodes from a graph. It starts by creating copies of the current `nodes` and `edges` arrays to avoid direct modification. It also identifies all superblocks (nodes of type `superBlockNode`) in the graph. For each node specified in the parameter of type array, the function processes it based on its type. If the node is a superblock, it retrieves the IDs of its child nodes and removes both the child nodes and the superblock itself from the `updatedNodes` array. Additionally, it deletes the edges associated with these child nodes and those connected to or from the superblock. It also removes any sheets related to the superblock from the `sheets` state. If the node is not a superblock, the function checks each superblock to see if the node to be deleted is one of its children. If so, it removes the node from the list of children. It then removes any edges connected to or originating from the node and finally removes the node from the `updatedNodes` array. The function updates the state with the modified lists of nodes and edges.

The code can be found at `frontend/src/App.jsx`.

Renaming nodes

The `handleRenameNode` function updates the label of a specific node and reflects this change across the application. It begins by creating a new array of nodes where the node with the matching ID is updated with the new name provided, while other nodes remain unchanged. The function then updates the sheet names associated with the node, replacing the old name with the new one if the node's ID matches. Finally, it updates the state with the modified array of nodes. This ensures that both the node label and any related sheets are synchronized with the new name.

The code is located at `frontend/src/App.jsx`.

Downloading the network in the JSON format

The `handleDownload` function manages the downloading of network data based on the specified file type. If the `fileType` parameter is `'json'`, the function converts the nodes, edges, and network parameters into a JSON string and creates a downloadable file using a `Blob`. It then generates a temporary URL for the file, creates a link element, and simulates a click on this link to trigger the download. After the download, it cleans up by revoking the temporary URL and removing the link element. If the `fileType` is not `'json'`, the function calls an API to export the network in the requested file format. It then creates a temporary URL for the received blob, sets up a download link, and simulates a click to initiate the

download. On successful download, it displays a success message, and on failure, it shows an error message with the exception details.

The code for this function can be found at `frontend/src/App.jsx`

Uploading a network

The `handleUpload` function handles the upload of a network file. It begins by checking if an `inputFile` is provided. If so, it uses a `FileReader` to read the file's contents as text. On successful reading, the function attempts to parse the file's content as JSON. If parsing is successful and the JSON contains `nodes`, `edges`, and `params`, it proceeds to set the nodes and edges state to the parsed data. The function also updates the application name to the file's name and initializes the sheets state. It then iterates over the parameters to set various configuration values such as `learningRate`, `epochs`, `batchSize`, `loss`, and `optimizer`. If an error occurs during parsing or reading, it catches the error and displays an appropriate error message. On successful upload, it displays a success message. The function also handles edge cases where errors occur while reading the file by providing error messages. The commented-out code seems to handle more complex cases of node mapping and updating, which is not currently in use.

The code is located at `frontend/src/App.jsx`

Training results

The `Results` component in this React application displays the results of the DL training session. It receives two props: `metrics` and `parameters`. The component is structured into three main sections. First, it presents a table of the training results, where it lists the latest values of various metrics, such as accuracy and loss, extracted from the most recent metrics data. Second, it provides a table of parameters used during training, including details such as learning rate and batch size. Finally, the component incorporates the `Plots` component, which visualizes the training metrics. The `Plots` component generates line charts for precision and recall, loss over epochs, and accuracy over epochs, using different colors for each session to distinguish between them. The charts are designed to aid in understanding the performance of the model throughout the training process.

The component is able to understand whether a model have been changed from the previous training session or not. If the model is the same as before, or with just some different parameters, the plots are made by showing the results with different colors and labels to provide a clean manner to compare the results with respect to the previous iteration.

Help page

The help page provides a comprehensive guide on how to use the application, including creating and training the networks, visualizing results, and exporting models.

5.3.3 Libraries

- **React**[2]: The choice of React for the frontend was driven by its component-based architecture, which allows for the modularization of the UI. This modularity is crucial for managing the complex interactions and the dynamic nature of the application's interface.

- **Reactflow**[3]: For handling the graphical representation of networks, including nodes and edges, the Reactflow library was employed. Reactflow offers a powerful and flexible way to visualize complex structures, making it an ideal choice for applications that require the manipulation of graphical elements, such as Neural Network layers in this case. The library also provides built-in functionalities like zooming, panning, and drag-and-drop, which enhance user interaction and experience.

- **Bootstrap**[4]: Bootstrap was utilized for quickly styling the user interface elements, ensuring that the application is not only functional but also visually appealing. The grid system, responsive design components, and pre-designed UI elements provided by Bootstrap allowed for a consistent user experience across different devices and screen sizes.

- **Chart.js**[5]: Chart.js was integrated for visualizing data in the form of interactive charts. This library is particularly useful for displaying the performance metrics of the trained model. Its ability to create various types of charts with minimal configuration made it a good fit for the application's needs.

[2] `https://react.dev/`

[3] `https://reactflow.dev/`

[4] `https://getbootstrap.com/`

[5] `https://www.chartjs.org/`

5.4 Middleware

5.4.1 Folder organization

The Figure 5.3 illustrates the organizational structure of the repository's directory that contains the middleware service code.

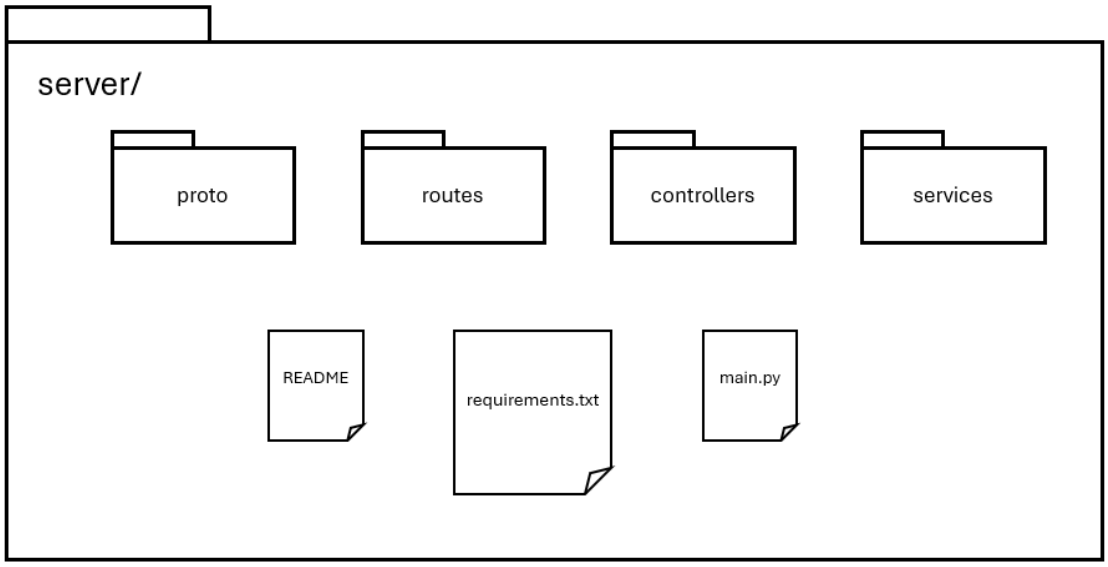

Figure 5.3: The folder structure for the middleware server, including the main sub-folders and files.

The `server` directory is organized to facilitate the development and deployment of the backend services. At the root of the directory lies essential files like `main.py`, which serves as the entry point for the application, and `requirements.txt`, which lists the necessary Python dependencies. The `README` file provides documentation for the server, explaining its setup and usage. Within the `server` folder, there are several sub-directories: `controllers` contains the logic that handles the incoming requests, `routes` defines the various API endpoints, and `services` manages the business logic and interactions with the data layer. The `proto` directory is designated for protocol buffer files, which are used for defining the structure of data and services in a gRPC context. This structure is designed to keep the backend code modular and scalable, allowing for easy maintenance and future enhancements.

5.4.2 Functionality

The middleware acts as a translator, converting HTTP requests from the frontend into gRPC calls that the backend can understand. It also manages sessions, handles

user requests, and routes them to the appropriate backend services. For instance, when a user requests to train a model, the middleware processes this request and forwards it to the backend, where the training process is initiated.

In addition to serving as a communication hub, the middleware also interacts with external services. For example, it may fetch resolution steps in case of errors, if the query is successfully processed by the external service. It also handles error management, ensuring that any issues in communication between the frontend and backend are gracefully managed and communicated back to the user.

Proto Conversion

The middleware server plays a critical role in the data processing pipeline by receiving serialized JSON data from the frontend, which encapsulates all the nodes and their connections. Its primary function is to transform this JSON data into a format that is compatible with the gRPC server's message definitions, ensuring that the backend server can effectively process the information.

For detailed information on the message definitions, refer to the protocol buffer file discussed in Section 5.5.2.

The structure sent by the frontend contains information about blocks, connections between blocks (`edges`), and global parameters (`params`). Blocks represent individual nodes in the network, each with a unique `id`, a node type (`type`), a position (`position`), specific data (`data`), and parameters (`parameters`). Each block may also include a specific function (`fn`), such as "torch.nn.Conv2d" for a convolution block. Additionally, special blocks like superblocks (`superBlockNode`) contain references to child nodes (`children`) to represent more complex structures.

The middleware service processes each block and converts it into a protocol buffer format where each block is represented as a node in the following way:

- Each block is assigned an `id` corresponding to its original identifier to keep the reference in case of errors.
- The block's parameters are translated into key-value pairs, where the parameter name becomes the key (`key`) and its value is stored as the `value`. If a parameter has no defined value, it is assigned `None`.
- For blocks representing specific operations (like PyTorch functions), a `function` field is added containing the function name (e.g., `torch.nn.Conv2d`).

`superBlockNode`s, which aggregate child nodes, are translated into protocol buffer blocks containing additional parameters. Each child node is represented as a parameter with the key `childId` and a value corresponding to the child's `id`. Invisible input and output nodes associated with complex blocks are also

translated as independent nodes in the protocol buffer, with their relationship preserved through edges.

The connections between blocks, defined in the `edges` section of the structure sent from the frontend, are translated into direct relationships between nodes in the protocol buffer. Each connection contains a source (`source`) and a target (`target`), represented by the corresponding node `id`s. These are added to the `edges` section of the protocol buffer.

Lastly, global parameters of the network, such as *learningRate*, *epochs*, and *batchSize*, are also translated as key-value pairs in the protocol buffer. These parameters are taken directly from the `params` section sent from the frontend, retaining the user-defined or default values.

The final result of this conversion is a protocol buffer structure that contains a complete representation of the network, including the blocks, connections, and global parameters.

External services

The middleware server is equipped with an error guidance mechanism that connects to external resources, such as Stack Exchange [34], to assist users in troubleshooting issues encountered during the training process or network validation. When an error occurs, the middleware server analyzes the error message and context, and then queries the Stack Exchange website[6] for potential solutions or explanations. This information is then presented to the user in a link format, offering immediate guidance and recommended actions. This integration enhances the user experience by providing accessible support and reducing downtime caused by errors.

Future Enhancements

Future versions of the application could explore integrating access to a broader range of external resources to further support users during error resolution and network validation. Additionally, the middleware server could be enhanced to function as a load balancer, distributing requests across multiple backend servers to ensure scalability and maintain performance during periods of high demand. This approach would allow the application to handle larger user bases efficiently, providing a more robust and responsive user experience.

[6] `https://api.stackexchange.com/docs/advanced-search`

5.4.3 Libraries

- **Flask**: Flask is a lightweight web framework for Python that was chosen for its simplicity and flexibility in building the middleware server. It allows for easy creation of APIs, which are essential for handling HTTP requests and routing in the application. Flask's minimalistic design and modular nature make it ideal for rapid development and integration with other components of the system, particularly in environments where quick prototyping and adaptability are required [35].

- **grpcio-tools**: The `grpcio-tools` library is an essential component used in the gRPC client proof of concept. This library provides tools for generating Python code from protocol buffer ('.proto') files, which define the structure of the messages and services used in gRPC communication. By using `grpcio-tools`, the application can seamlessly convert protocol buffer definitions into Python classes, enabling efficient communication between the client and server through Remote Procedure Calls (RPCs). This is crucial for ensuring that the middleware server can interact with the backend in a structured and consistent manner [36].

5.5 Backend

The backend is built as a gRPC server, which allows for efficient communication between the middleware and the backend services. gRPC was chosen for its performance benefits and its support for multiple programming languages, which provides flexibility in how the backend can be extended in the future.

5.5.1 Folder organization

The Figure 5.4 illustrates the organizational structure of the repository's directory that contains the execution service code.

The `execution_service` directory is designed to handle the core functionalities related to network and model generation within the application. The directory is organized into several key components: the `network_generation` sub-directory contains scripts and logic dedicated to the generation of Neural Network architectures, while `model_generation.py` and `pytorch_functions.py` provide specific functions and classes for building and managing PyTorch models. The `proto` directory is included for protocol buffer files, which are likely used to define data structures and services for communication purposes. The root of the directory also contains `execution_service.py`, the main script that coordinates the various services, along with `requirements.txt`, which lists the dependencies necessary for

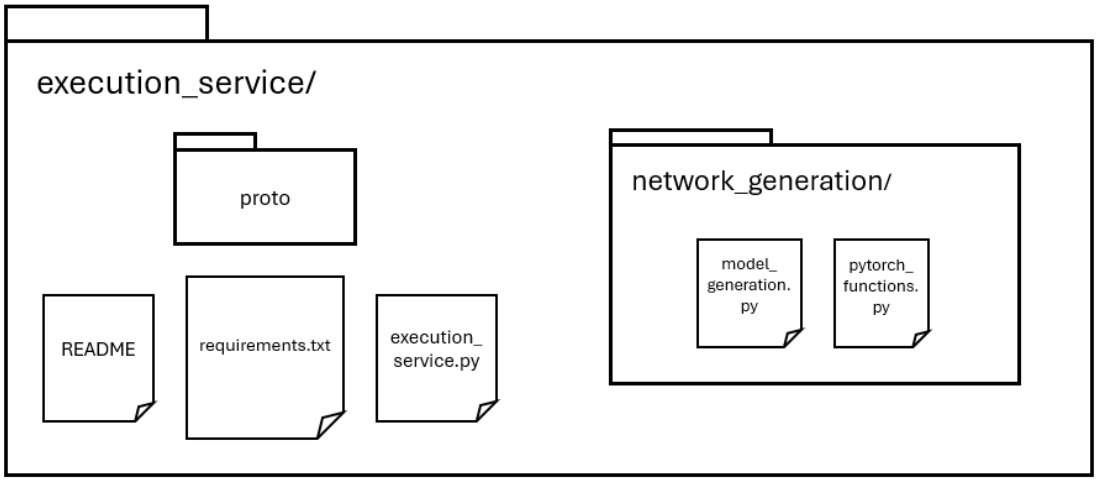

Figure 5.4: The folder structure for the execution service implementation, including the main sub-folders and files

the execution service to run, and a `README` file, which offers documentation and instructions related to the service.

5.5.2 Functionality

The backend's primary responsibility is to provide services related to Deep Learning model management. These services include:

- **Model training**: The backend can initiate and manage the training of Deep Learning models. This includes compiling the model, setting up the training loop, and managing resources such as GPUs.
- **Model export**: Once a model is trained, the backend can export the model in a format suitable for deployment, such as ONNX or PyTorch model.
- **Model pre-training check**: The backend provides tools for checking the model by forwarding all the layers one by one and returning the errors.

Message definition

In the context of a gRPC environment, the protocol file serves as a critical contract between the client and server, defining the communication structure and the available services.

The `Trainer` service provides three Remote Procedure Calls (RPCs) that the server implements:

- **TrainNetwork**: This RPC accepts a `Network` message and returns a `NetworkResult`. The `Network` message contains the Neural Network's architecture, defined by nodes and edges, and any associated parameters. The server's role is to process this structure, execute the training logic, and return the results encapsulated in the `NetworkResult` message, which includes metrics and status information.
- **ExportNetwork**: This RPC is designed to export the trained network to the ONNX file format or PyTorch's `.pth`. The server receives the `Network` message, processes the request to export the network, and returns a `File` message containing the serialized network data and file name.
- **ForwardBlock**: This RPC facilitates the forward propagation of a single block within the network. The server receives a `Network` message representing the block and returns a `ForwardResult`, which includes the output parameters and a status message.

The protocol file defines several message types to structure the data exchanged between the client and server:

- **Node**: Represents a computational unit within the network, containing an ID, a PyTorch function name, and associated parameters.
- **Edge**: Defines connections between nodes, indicating the data flow within the network.
- **Parameters**: Key-value pairs used to configure nodes or the network as a whole.
- **File**: Used to encapsulate file data and metadata, such as when exporting a network or uploading a custom loss function to the server.
- **Metric**: Represents performance metrics, such as accuracy or loss, tracked during the training process.
- **Network**: A composite message that encapsulates the entire network's structure, including nodes, edges, parameters, and any associated files.
- **NetworkResult**: Contains the outcome of the `TrainNetwork` RPC, including the status indicating a numerical code indicating the successful completion of the operation or that an error occurred, a message, and performance metrics obtained from the network evaluation phase.
- **ForwardResult**: Contains the outcome of the `ForwardBlock` RPC, including the status indicating a numerical code indicating the successful completion of the operation or that an error occurred and a message.

The protocol buffer definition is presented in the Appendix A.

Model creation

A key operation in the system is the creation of the model used to manage layers and execute the training process. The `create_model` function demonstrates how the model is generated in response to a request sent by the gRPC client. This request includes the network structure, encapsulated in the `Network` message (detailed in Appendix A), which defines all nodes, edges, network parameters and the custom loss function file if present.

The first step in the process is responsible for transforming the node specifications (received as strings) into actual PyTorch modules that will compose the Neural Network architecture. The function begins by processing a list of nodes that represent different layers in the model. Each node contains information about the type of operation (such as a convolutional or fully connected layer) and its parameters. The goal is to map these nodes to their corresponding PyTorch layers and configure them based on the provided parameters.

The core functionality starts with an internal helper function, `filter_params`, which is used to filter and convert the node parameters into the expected format for each layer. It checks the parameter type (e.g., int, float, tuple) and ensures that the values are correctly parsed from the node's string data. The function relies on a predefined dictionary which stores the expected types for each parameter of the corresponding PyTorch functions and can easily be expanded.

Next, for each node, the function determines the appropriate PyTorch class by extracting the function name (using the `node.function` field) and checking if it is a valid PyTorch function. If the function is valid, the code then proceeds to create the specific layer based on the node's parameters.

These layers are then passed to the `CustomModel` class, which extends `nn.Module`.

The `CustomModel` is designed to create and execute a DL model by combining nodes, edges, and layers (modules). These three components — nodes, edges, and modules — form the foundation of the Neural Network architecture. The core functionality involves mapping these nodes to corresponding computational layers, defining the relationships between nodes using edges, and finally constructing a model that can perform a forward pass by processing data from input to output.

At the initialization stage, the model is provided with three main inputs: nodes, edges, and modules. The nodes represent the individual computational blocks (layers) of the network, each identified by a unique identifier (`id`). The edges define the dependencies between the nodes, specifying how data should flow from one node to another. The modules are the layers or computational blocks of the Neural Network, such as `torch.nn.Conv2d`, `torch.nn.ReLU`, and so on.

Once the model is initialized, the first step is to construct a computational graph. This graph organizes the nodes based on the connections specified by the edges and ensures that each node is executed only after its required inputs are available. The

graph is built by iterating over all the edges and connecting the source and target nodes accordingly. The result is a directed acyclic graph (DAG) that represents the data flow through the network. In addition to the graph construction, the model establishes a *topological order*[7] of nodes. This ordering ensures that the nodes are executed in a sequence that respects their dependencies. Specifically, for any node, all preceding nodes in the graph (those providing inputs to this node) must be executed before the current node. This topological sort is critical for determining the correct sequence of operations during the forward pass.

Once the graph and topological order are in place, the model proceeds to map the nodes to their respective layers. To achieve this mapping, the model compares the function of each node to the available layers in the module list. It ensures that the parameters of the node (such as input/output shapes) match those of the layer through the `compare_parameters` function. If a match is found, the node is associated with the corresponding layer, ensuring that it will perform the correct computation during the forward pass.

The forward pass is the key phase where data flows through the network from input to output. The process begins with the input data being passed to the first node in the topological order. For each node, the model performs the following steps: first, it collects the outputs of all predecessor nodes (those providing inputs to the current node). If the node is a standard layer, such as a convolutional layer, it simply applies the layer's transformation to the input data. However, if the node is an aggregation function (e.g., addition, multiplication, concatenation), it combines the inputs from multiple predecessor nodes according to the specified operation.

The model handles several special cases during the forward pass. For example, certain nodes perform operations such as splitting, where the output of one node is sent to multiple successor nodes. Additionally, some nodes may represent attention layers, such as multi-head attention. In these cases, the model ensures that the input data is appropriately reshaped before applying the attention mechanism.

Throughout the forward pass, the outputs of each node are stored and made available for subsequent nodes. This ensures that the forward pass proceeds smoothly through the network, with each node receiving the required inputs from its predecessors. If any node encounters an error, the model provides a detailed error message, indicating which node and layer caused the issue. This makes debugging easier and allows for more robust network design.

[7] `https://www.geeksforgeeks.org/topological-sorting/`

Model pre-training check

The `forward_model` function is a critical component in the system, responsible for managing the initial data pre-processing and executing the Neural Network's forward pass based on the structure provided by the client.

The function identifies the input dataset configuration by parsing the nodes for specific parameters that describe the dataset name, type, and configuration. Depending on the dataset type — whether text or image — the function loads and pre-processes the dataset accordingly. For text datasets, a tokenizer is employed to convert the text into a format suitable for the model, while image datasets are resized and processed using a feature extractor like `ViTImageProcessor`[8].

Once the dataset is prepared and formatted, the function initializes an input tensor that matches the expected input size of the model. This tensor is crucial for simulating the input during the forward pass.

Following the data preparation, the function constructs the model by invoking the `create_model` method, which assembles the network based on the provided nodes and edges. If the model creation is successful, the function proceeds to execute the forward pass by iterating through the network's topologically ordered nodes. The output of each node is computed and stored, ensuring the correct data flow through the network.

Model training

The `train_model` function is the core component responsible for orchestrating the training process of the Neural Network models. It handles the loading and pre-processing of datasets, model creation, parameter initialization, and the iterative training and evaluation phases.

Initially, the function determines the appropriate computing device, utilizing a GPU if available. It then identifies and loads the input dataset based on the nodes' configuration, which includes the dataset name, type, and any specific configuration required. Depending on the dataset type —either text or image—different pre-processing steps are applied. For text data, tokenization is performed using the BERT tokenizer[9], while for image data, images are resized and processed using a pre-trained Vision Transformer (ViT) image processor. The pre-processed datasets are then converted into PyTorch tensors, ready for model input.

Once the dataset is prepared, the model is constructed using the `create_model` function, which initializes the network architecture based on the provided nodes and edges. The model is subsequently moved to the device for training. The

[8] `https://huggingface.co/docs/transformers/model_doc/vit`

[9] `https://huggingface.co/docs/transformers/model_doc/bert`

function then sets up essential training parameters, such as the number of epochs, learning rate, batch size, loss function, and optimizer, based on user-defined or default values.

The training loop involves multiple epochs, where the model iteratively processes batches of data, computes the loss using the specified loss function, performs backpropagation, and updates the model weights. The learning rate is adjusted dynamically using a scheduler to optimize the training process. After each epoch, the model's performance is evaluated on a validation dataset to compute key metrics, including accuracy, precision, recall, and F1-score. These metrics are tracked across epochs to monitor the model's improvement and generalization.

Throughout the training process, the function incorporates extensive error handling to catch and report issues, such as mismatched tensor shapes or failures in loss computation and backpropagation. This robust design ensures that any problems encountered during training are promptly identified and addressed, allowing for smooth model development and tuning.

At the end of the training, the function returns a set of metrics that summarize the model's performance over the training epochs, providing valuable insights into the effectiveness of the training process.

Export model to ONNX or pth file

The `ExportNetwork` RPC provides the capability to export trained models into industry-standard formats such as ONNX and `.pth`, ensuring compatibility with a wide range of deployment environments and tools. This functionality is encapsulated within the `export_to_onnx` and `export_to_pth` functions, which handle the conversion of PyTorch models based on the network structure defined by nodes and edges.

The `export_to_onnx` function converts a PyTorch model into the ONNX format, which is particularly useful for deploying models in environments that support ONNX, such as Tensorflow and other frameworks. The function first constructs the model using the provided nodes and edges, simulating an input tensor for the export process. If the initial export fails, the function attempts an alternative input tensor shape, ensuring robustness in handling different model configurations. Upon successful export, the ONNX file is saved to the specified directory, ready for being sent to the gRPC client (the middleware server).

Similarly, the `export_to_pth` function facilitates the export of models into the `.pth` format, which is the standard format for saving PyTorch models. This format preserves the model's state dictionary, including all the learned parameters, enabling seamless loading and inference in any PyTorch environment. The function constructs the model and saves the state dictionary to the specified file path. In case of any issues during the saving process, appropriate exceptions are raised to

inform the user.

Both functions are designed with error handling mechanisms to ensure the export process is reliable and user-friendly. Whether the target format is ONNX or `.pth`, these functions play a crucial role in the deployment and distribution of trained models.

Future Enhancements

The backend is designed to be extensible, allowing new services to be added as the needs of the application grow. For example, additional services could be added to support new types of models, integration with different ML frameworks, or more advanced debugging and monitoring tools.

5.5.3 Libraries

- **Torch**: Torch is a foundational library for DL, providing robust support for tensor operations, automatic differentiation, and Neural Network implementation. It is widely used in Machine Learning for tasks ranging from basic linear algebra to complex Neural Network training [37].

- **NumPy**: NumPy is a fundamental package for scientific computing in Python, offering powerful tools for working with arrays and matrices. It integrates seamlessly with Torch, allowing for efficient numerical computations, which are crucial in data preprocessing, manipulation, and mathematical operations within Machine Learning pipelines [38].

- **Torchvision**: Torchvision extends Torch by providing datasets, model architectures, and image processing utilities specifically tailored for computer vision tasks. It includes pre-trained models, transforms for image augmentation, and easy-to-use data loaders, facilitating the development of image-based Deep Learning models [39].

- **grpcio-tools**: The `grpcio-tools` library is vital for enabling gRPC communication in the server architecture. It allows for the generation of Python code from protocol buffer ('.proto') files, which define the structure of messages and services in gRPC. This ensures efficient and structured communication between different components of a distributed system, particularly in environments where performance and scalability are critical [36].

- **Hugging Face**: Hugging Face is renowned for its extensive collection of pre-trained models, particularly in Natural Language Processing (NLP). It provides tools for fine-tuning, deploying, and utilizing models in various NLP

tasks, simplifying the integration of state-of-the-art models into production systems and research workflows [40].

- **Evaluate**[10]: The `evaluate` library is used for assessing the performance of Machine Learning models by providing a wide range of evaluation metrics. It is particularly useful in the training and validation phases, offering metrics like accuracy, precision, recall, and F1-score to ensure that models meet desired performance criteria.
- **Transformers**[11]: Part of the Hugging Face ecosystem, the `transformers` library is focused on implementing transformer models, which are the backbone of many modern NLP systems. It provides easy access to pre-trained models and tools for fine-tuning them on specific tasks, making it a crucial tool for developing and deploying NLP applications.

[10] `https://huggingface.co/docs/evaluate/index`

[11] `https://huggingface.co/docs/transformers/index`

Chapter 6

User Testing

In this chapter, the testing phase is presented comprehensively, offering a clear breakdown of user categories, a detailed explanation of the testing scripts, and an in-depth analysis of the results. The user categories are carefully defined to simulate real-world usage scenarios. The testing script is meticulously crafted, with each step explained to ensure reproducibility and ease of understanding.

The test consisted of two parts: one focused on coding and discussing the implementation of a ResNet with two residual blocks, while the other one involved creating the same ResNet using DeepBlocks to streamline the process.

6.1 User categories

The identified user categories comprise both ML novices, such as Machine Learning students, and ML practitioners, ranging from freshly graduated Master's degree students to young researchers. This selection was made to ensure the tool or framework is accessible and valuable across a broad spectrum of experience levels. ML novices often need more guidance and simplified explanations, making it crucial to evaluate whether the system provides sufficient clarity and support for beginners. Meanwhile, ML practitioners, especially those with more experience, require a solution that is robust, scalable, and capable of handling complex tasks. By including this range of users, the testing phase can assess whether the system effectively caters to the diverse needs of both groups — offering intuitive functionality for learners while maintaining the flexibility and depth required by more advanced users. This comprehensive approach ensures that the system can support growth in proficiency and adapt to various skill levels.

Following this approach, a total of 16 participants were selected for testing, consisting of 8 Machine Learning students and 8 PhD candidates or research assistants. To minimize order effects as discussed in Section 3.4, randomization of

task order was introduced. Half of the students first completed the coding task before using the DeepBlocks application to create the same model, while the other half began with the DeepBlocks task. The same methodology was applied to the PhD candidates and research assistants. This randomization was implemented to ensure that the order of task presentation did not influence the participants' performance or their perception of task difficulty.

6.1.1 Screening survey

Testers were asked to complete a brief questionnaire aimed at gathering personal data and assessing their familiarity with Machine Learning (ML) concepts. The questionnaire consists of two sections.

In the first section, testers were asked for demographic information, including their name, age, gender, highest level of education attained (and the specific field of study), as well as their current occupation (student, young researcher, or PhD candidate). This section was designed to capture basic background information to understand the diversity of the participants and how different educational or professional experiences might influence their interaction with ML tools.

The second section focused on the tester's knowledge of Machine Learning, divided into two subsections: theoretical knowledge and practical coding experience. Testers were asked to assess their familiarity with various aspects of ML, starting with their knowledge of designing an ML pipeline and their confidence in using popular ML frameworks (such as PyTorch, TensorFlow, etc.). The familiarity with coding a Machine Learning pipeline was also evaluated. The answer scale ranged from *No knowledge* to *I am an expert*, with intermediary options like *I am new to ML concepts*, *I know a few things*, and *I have good knowledge*. This range of responses allows for a nuanced understanding of the user's competency level, ensuring that the system can cater to both novices and experts.

Additionally, participants were asked to indicate how frequently they engage in coding tasks related to ML models, with options such as *Less than once a month*, *Once a month*, *Less than once a week*, *Once a week*, and *More than once a week*. This was included to gauge their hands-on experience with coding ML models, which often differs from theoretical understanding.

The questions were chosen deliberately to ensure that the system was evaluated by a diverse range of users with varying levels of experience. The goal was to ensure the platform could address the needs of both theoretical learners and those more involved in the practical application of Machine Learning. Moreover, by understanding their coding frequency, we could better assess how familiar users were with implementing ML solutions, ensuring the feedback from occasional users was just as valuable as that from frequent practitioners.

6.2 Test script

Before beginning the usability testing, the DeepBlocks project was introduced to the users to explain the concept behind the application and its overall goals.

The test was divided into two segments: the first centered on coding and discussing the implementation of a ResNet featuring two residual blocks, accompanied by an intermediate questionnaire assessing the task's difficulty. The second segment entailed constructing the identical ResNet utilizing DeepBlocks to simplify the procedure, followed by a questionnaire that posed specific questions after certain tasks to appraise the entire process.

Finally, as last step, users were required to fill the SUS questionnaire concerning the usage of DeepBlocks and a qualitative survey mainly focus on what could be improved in further versions and what the testers really liked about the application.

6.2.1 DeepBlocks part

The usability testing conducted for the DeepBlocks application involved a series of tasks designed to evaluate user interaction and functionality comprehension. Users were instructed to open the training menu and configure the training parameters, including setting the number of epochs to 10, the learning rate to 0.0001, and the batch size to 64. In addition, the loss function was customized by uploading a file named `custom_loss.py`, and the Adam optimizer was selected. Successful completion of this task was determined by the correct configuration of all parameters through the appropriate menu navigation.

Following this, users were asked to add an input node and configure it by assigning the dataset `sasha/dog-food` (from HuggingFace datasets[1]). The task was deemed successful if the user correctly identified the node and applied the correct dataset.

To further evaluate the application's ease of use, users were tasked with building the model by adding the remaining blocks in sequence in order to build a ResNet. A successfully completed task resulted in the construction of the ResNet with two residual blocks.

Users were also encouraged to introduce *superblocks*, blocks containing other blocks inside, for scalability purposes, ensuring that the network remained organized on the canvas. Superblocks were connected to the network, indicating successful completion of the task.

In the next step, participants were prompted to investigate the result of a network check without modifying any blocks or parameters for a short duration.

[1] `https://huggingface.co/datasets`

If any errors were encountered, users were instructed to diagnose the problem by interpreting the error message, identifying the problematic node, and resolving the issue.

The following phase involved training the network. Users were required to locate and activate the *Train* button, allowing the training process to commence. If any errors occurred during training, participants were expected to follow a similar investigative process as in the previous task, correctly diagnosing the issue based on the server's response and implementing the necessary corrections.

After training, users were instructed to explore the results of the network, which involved interacting with the visualizations provided. This interaction included navigating through the results and examining details by hovering over the plots.

Finally, participants completed the test by downloading the constructed network in JSON format. This involved accessing the relevant menu and ensuring the correct format was selected for download.

Each task was structured to evaluate the intuitive use of the application, the clarity of the error messages, and the overall user experience in performing essential operations such as model construction, training, and results interpretation.

6.2.2 Coding part

In this part of the test, participants were tasked with coding a Deep Neural Network from scratch, specifically implementing the structure of a ResNet model. They were free to use any programming language or framework they preferred, such as PyTorch, TensorFlow, or others. The task began with initializing a new file and adding the necessary input datasets and nodes to build the network.

Next, they implemented the validation steps for the model. Participants were required to configure the network parameters by setting the same hyperparameters used for the DeepBlocks part. They also needed to define a custom loss function tailored to the task and select the Adam optimizer for training.

Once these configurations were in place, the participants trained the network and monitored its progress. This task evaluated their ability to manually code and configure a Deep Learning model, focusing on advanced aspects such as customizing loss functions and implementing validation procedures. The freedom to choose any framework or language allowed participants to leverage their familiarity with different tools, making the test more adaptable to their individual skills.

6.2.3 Post-task questionnaire

For the tasks related to the creation of superblocks and setting hyperparameters, a post-task questionnaire was provided. The question asked, "Overall, this task

was" with response options ranging from "Very difficult" to "Very easy" on a five-point scale. Additionally, a post-task questionnaire was administered for the task involving the coding of the ResNet. This question aimed to provide an overview of how both Machine Learning novices and practitioners perceive the difficulty of coding a Deep Learning model. These three tasks were specifically chosen for evaluation as they require more complex thinking and problem-solving compared to the others.

6.2.4 Qualitative survey

The questionnaire is designed to collect qualitative feedback on the usability, effectiveness, and areas for improvement in DeepBlocks. Participants were asked to compare their experience using DeepBlocks with traditional methods of building Deep Learning models, identifying any advantages or disadvantages. The questionnaire explores how DeepBlocks influences the model creation process, probing into specific features that users found intuitive or challenging. Additionally, it seeks to understand how DeepBlocks could affect users' workflow in the long term and how its Visual Programming approach compares to conventional coding practices in Deep Learning. Participants are also encouraged to describe any limitations they encountered, provide suggestions for improvement, and reflect on scenarios where DeepBlocks might be particularly beneficial. The ultimate goal is to gather insights that will help enhance the tool and understand its potential impact on the Deep Learning community.

6.3 Results

This section presents the results of the questionnaires along with the lessons learned from the feedback provided by the testers.

Table 6.2 presents the numerical results of the number of Critical and Non-Critical errors per task.

6.3.1 Comments

During the testing phase, several users highlighted the potential and usability of the application. T2 mentioned that they would be one of the first users if the application became available, while T3 compared it to a similar mechanism in Matlab[2], pointing out that Matlab's version lacks export capabilities in different formats. For academic use, T3 suggested that it would be helpful to visualize the

[2] `https://it.mathworks.com/help/deeplearning/ref/deepnetworkdesigner-app.html`

Task	Critical	Non-Critical
Add the input node	0	2
Add the blocks and set the parameters	0	18
Superblock creation	1	25
Set hyperparameters	0	6
Error check	0	0
Error correction	0	8
Train the network	0	2
Open training results	0	1
Download model as a JSON file	0	0

Table 6.2: Critical and Non-Critical Errors per Task

code generated by each block and view the output size for each edge. T4 was impressed with the application's potential impact on students, noting its ability to simplify the creation and visualization of Neural Networks. However, T4 also mentioned the absence of common commands like Ctrl+C and Ctrl+V for node manipulation. T5 quickly used the guide to create superblocks and duplicate nodes and commented that the application is suitable for users with limited programming knowledge, though a good understanding of the blocks and parameters is still necessary. T6 initially opened the guide but struggled to find it again after closing it, and also mentioned difficulty in finding the hyperparameter settings window and the way to start the training process. Once familiar with the process, creating a superblock was considered manageable. T7 quickly glanced through the guide, attempted to delete a node using keyboard shortcuts but failed, and found the 'play' icon for setting hyperparameters unintuitive. They noted that the superblock creation process is simple with the guide but not obvious otherwise. Additionally, T7 found the duplicate functionality insufficient and suggested the application would be highly beneficial for users learning Neural Networks. T8 expected a popup to choose the location and name of saved files after clicking the 'Download' button and found the numerous error messages bothersome as they did not disappear after re-training. T9 had trouble merging superblocks using selection boxes and emphasized the need for clearer saving mechanisms for node parameters. T10 appreciated the application's concept but suggested UI improvements, such as easier copying and automatic renaming of copied superblocks with different names. T11 suggested enlarging the block pins, adding keyboard shortcuts, and improving error handling. T12 and T13 explored the application with ease, using the guide for clarification when needed. They expected standard keyboard commands like Ctrl+C and Ctrl+V to work. T14 also found the persistent error messages annoying

but found the guide helpful for beginners. T15 recommended a clearer right-click menu for renaming superblocks, while T16 suggested larger junction points, noting the application is useful for those new to Machine Learning, helping lower the learning curve.

6.3.2 Screening survey

Testers were carefully selected to ensure an equal distribution between Machine Learning novices and more experienced users. The group of ML novices consists primarily of students pursuing a Master's degree, with only one participant enrolled in a Bachelor's degree program. Specifically, half of the students, totaling four individuals, are enrolled in Master's programs in Biomedical Engineering. Two are pursuing a Master's degree in Computer Engineering, one is a Bachelor's student in Computer Engineering, and one is enrolled in a Master's program in Electronic Engineering.

The more experienced testers, including PhD candidates and young professionals, are categorized as follows: one research assistant in Cinema and Media Engineering, one research assistant in Biomedical Engineering, and six individuals working in the field of Computer Engineering. Among these six, two are software developers, while the remaining four are PhD candidates.

Of the sixteen testers, thirteen identify as men, and the remaining three identify as women. The age of participants ranges from 19 to 30 years, encompassing a wide spectrum of educational and professional stages. This distribution reflects the common trajectory within technical fields, where individuals begin formal education in their late teens and continue through to professional roles by their late twenties or early thirties.

The results from the screening survey reveal varying levels of familiarity and confidence in Machine Learning (ML) competencies among testers. A significant portion of the respondents indicated that they "know a few things" about designing an ML pipeline, coding within an ML framework, and using ML frameworks, with a smaller group expressing "good knowledge" in these areas. A minority reported no knowledge or being new to ML concepts. In terms of coding frequency, participants' experience also varied, with some coding "more than once a week" while others indicated programming "less than once a month" for ML purposes. The variation in both ML knowledge and coding frequency highlights the range of experience levels among the individuals surveyed, which is crucial for understanding how users with different ML backgrounds approach and interact with Machine Learning tools and frameworks.

6.3.3 Post-task questionnaire

The results of the post-task questionnaire are categorized into three main tasks: coding (task 1), superblock creation (task 3), and setting hyperparameters (task 4). Each task was evaluated by participants based on the level of difficulty they experienced during the post-task phase.

Coding task

Participants' responses to task 1, which involved coding, showed a broad spectrum of difficulty levels. A significant portion of the respondents found the task to be "quite difficult" or even "very difficult", highlighting the challenges some users faced with coding. However, other participants rated the task as "neither difficult nor easy", indicating a neutral experience. A smaller subset of users found coding to be "quite easy" or "very easy", suggesting that a portion of the testers felt comfortable with the coding task. Overall, the range of experiences indicates varying levels of familiarity and proficiency with coding among the participants with the more expert users or student indicating this task as easy.

Superblock creation task

For task 3, which involved superblock creation, the majority of participants rated the task as either "quite easy" or "neither difficult nor easy". A few respondents found it "quite difficult", but overall, the superblock creation task was perceived as moderately easy to neutral by most. This suggests that participants were generally comfortable with this aspect of the system, though there were still some who found it challenging. Most participants noted that the process becomes smooth after consulting the help guide, yet they suggested it could be more intuitive, eliminating the need to refer to the guide when creating a superblock.

Hyperparameters setting task

Task 4, which focused on setting hyperparameters, was generally regarded as easier compared to the other tasks. Many participants rated it as "very easy" or "quite easy" demonstrating a high level of comfort and confidence in completing this task. Only a small fraction found it "quite difficult" or "neither difficult nor easy" mainly because they didn't find intuitive the menu icon. The ease of this task for most participants suggests that setting hyperparameters was intuitive and well integrated within the system, leading to more positive experiences.

6.3.4 SUS questionnaire

The answers to the SUS questionnaire are shown in Figure 6.1. It is possible to observe that the responses to the System Usability Scale (SUS) questionnaire provided a detailed overview of the system's usability. The results revealed that most participants agreed or strongly agreed that they would like to use the system frequently. Many also found the system easy to use, with several respondents expressing high confidence in their ability to use it. However, some participants felt that they would need the support of a technical person to operate it effectively.

The integration of system functions was generally well received, as many testers either agreed or strongly agreed that the system's various functions were well integrated. A few respondents found the system's design consistent, not highlighting much inconsistencies in the system. There was also a mix of opinions regarding the learning curve: while some participants believed that most users would quickly learn how to use the system, others felt they needed to learn many things before being able to use it effectively. Furthermore, while several users agreed that the system was not cumbersome, others found it somewhat challenging to use.

Overall, the SUS scores reflect a balance between positive usability experiences and areas where some users encountered difficulties, suggesting that while the system is generally user-friendly, certain aspects may require refinement to enhance consistency and ease of learning.

Following the rules to calculate the SUS score presented in Section 3.4.2, the overall **SUS score** is 79.53, which indicates that the system falls into the "good" range of usability. This score suggests that, on average, users had a positive experience, though there are still areas that could be improved for a more seamless and consistent user interaction.

6.3.5 Qualitative survey

The feedback on using DeepBlocks compared to traditional methods for creating Deep Learning models highlights both its advantages and limitations. Many users emphasized how DeepBlocks simplifies the process through its graphical UI, making it easier to visualize and connect different layers of the model. This visual approach allows users to translate their ideas into working models more intuitively and quickly, which is seen as a major advantage over writing code from scratch. Users appreciated the clear feedback provided on connections between layers, helping to avoid errors and ensuring that parameters between blocks were compatible. Features like superblocks, which allow for the reuse of components as well as improving scalability, further streamline the process. Beginners benefited from the drag-and-drop functionality, which made it easier to understand and experiment with different components. However, some users noted drawbacks, including a lack of flexibility in adjusting parameters directly compared to coding, and frustrations

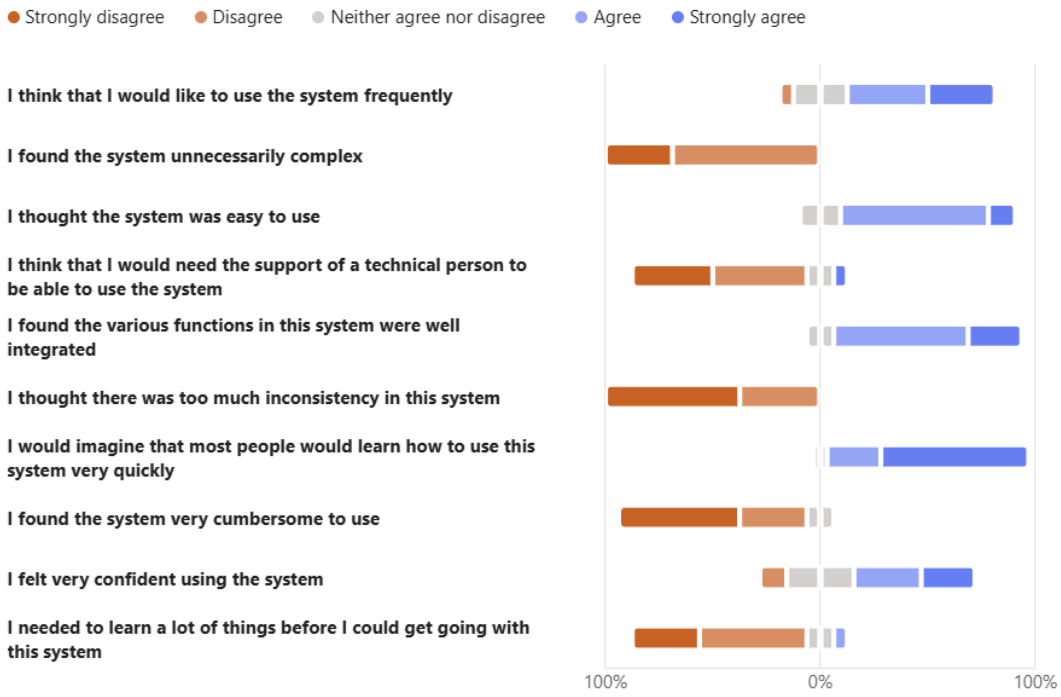

Figure 6.1: SUS questionnaire results

with the interface related to block selection and editing. Experienced developers found it slower and less efficient than traditional coding, particularly due to the absence of features like code completion or familiar shortcuts. While the tool was seen as intuitive and user-friendly for beginners and those focused on visualizing models, some experienced users still preferred the control offered by traditional coding methods.

Several features of DeepBlocks stood out as particularly useful, such as the ability to easily define necessary parameters by clicking on a block, ensuring no parameters were overlooked. The intuitive drag-and-drop interface for creating connections between blocks provided a clear, chronological representation of the model's workflow. The automatic checking of dimensional compatibility between blocks was highlighted as a valuable feature. Users also appreciated the integrated training function, which allowed them to start training models with minimal effort by entering hyperparameters and clicking a button, as well as the helper page providing clear instructions for navigating the platform.

Challenges encountered while using DeepBlocks included difficulties in making manual connections between blocks, duplicating or creating superblocks, and grouping blocks due to a confusing selection system. Users reported frustrations with the absence of keyboard shortcuts for copying and pasting, and noted that

personalized block creation was not intuitive. The right-click menu could benefit from clearer visual cues, and some users found error messages intrusive, needing to be manually cleared. The process of disconnecting or removing links between nodes was not clearly explained, and suggestions were made to increase the size of block junctions for easier navigation.

Many users expressed that using DeepBlocks regularly could enhance their workflow, particularly by improving the efficiency of creating and modifying Deep Learning models. The ability to visualize existing models and make graphical modifications with direct feedback was seen as a strong advantage. Users suggested that if bugs were fixed and features like easier copy-paste functionality were added, it could further speed up their workflow. However, some more programming-oriented users felt that DeepBlocks could slow them down and viewed it more as a learning tool than a programming tool.

While many users did not encounter significant limitations, some reported issues that affected task completion, such as the inability to unselect grouped blocks and difficulties with linking blocks correctly. Better access to guidelines and help resources within the app was suggested to streamline workflows. Users also identified hyperparameter tuning as an area for improvement.

To enhance DeepBlocks for developers, suggestions included displaying acceptable value ranges for block parameters based on connected layers, cleaning up existing bugs, incorporating new blocks, and linking documentation for frameworks like PyTorch and TensorFlow. Enhancements could also include greater customization of the training procedure and allowing users to download complete code, including the training procedure. Improved error visualization that aligns with the overall UI and automatic cleaning of error messages would enhance usability, particularly for non-developers.

DeepBlocks is particularly beneficial for educational projects, beginner-level Deep Learning tasks, and environments where users are starting to explore Machine Learning. Its scalability allows users to easily modify previous projects for new applications while receiving direct feedback on their adjustments, which is valuable for those with minimal knowledge of Deep Learning as it aids in visualizing and comprehending the underlying concepts.

6.3.6 Future enhancements

In future versions of DeepBlocks, several enhancements could significantly improve user experience and functionality. Firstly, the app could include a feature that displays acceptable value ranges for block parameters based on connected layers, making it easier for users to set parameters correctly. Addressing existing bugs would also enhance the overall user experience. Additionally, incorporating new blocks would expand the app's functionality, and linking documentation for frameworks

like PyTorch and TensorFlow directly within the app would provide valuable context for users. Another beneficial feature would be the option to download the complete code, including the training procedure, which would provide better transparency and usability. Improving error visualization to align with the overall UI and implementing automatic clearing of error messages once resolved would enhance usability, particularly for non-developers. Lastly, increasing the size of block junctions for easier navigation and improving access to guidelines and help resources within the app would streamline user workflow. Incorporating these enhancements could greatly improve the efficiency and usability of DeepBlocks for its users.

Chapter 7

Conclusions

The use of Machine Learning algorithms has expanded across various industrial sectors, gaining widespread popularity. However, constructing a Machine Learning pipeline depends on the programming skills of data scientists. Visual Programming applications have addressed this challenge by enabling the creation of such pipelines through a drag-and-drop interface of blocks that represent ML functionalities, although they have their limitations.

DeepBlocks aligns with the principles established by the creators of DL-IDE, highlighting the importance of a Visual Programming tool for developing Deep Learning models that features an intuitive interface, removes the necessity for coding, and enables efficient training along with rapid and scalable deployment [8].

DeepBlocks utilizes Visual Programming for the creation, debugging, and training of Deep Neural Networks. It allows users to place blocks that symbolize PyTorch functions onto a canvas, adjust their parameters, and link them together. DeepBlocks introduces *superblocks*, as termed in its prior version, which are blocks designed to encapsulate additional blocks within them. It also features a user-friendly interface that facilitates easy navigation within the web tool. Additionally, DeepBlocks performs checks after each modification to evaluate if the altered model is still prepared for training or if any errors have occurred. Moreover, DeepBlocks offers users the ability to download their designed model in multiple formats, enabling its use in other applications or its re-upload to the web tool for further development.

DeepBlocks is designed as a web application to ensure easy access from any device and from any location globally, eliminating the need for downloading an executable file to begin crafting a Deep Learning model.

The testing phase revealed that the tool features a user-friendly interface, which could significantly flatten the learning curve, particularly for novices. Moreover, the ability to visualize all the model components was well-received by the majority of users, allowing for a comprehensive view of the model and aiding in their

comprehension of its structure.

This thesis has laid the groundwork for developing an application that aids in building Deep Learning models. However, there remains significant room for future work that can expand its scope and capabilities. One potential avenue for expansion is to increase the range of functions available for model building, allowing users to design more complex and diverse models. This could include integrating a broader selection of layers, optimizers, loss functions, and pre-processing tools to give users more flexibility in their designs.

Additionally, it would be beneficial to incorporate more detailed documentation and educational resources within the application itself. By providing clear descriptions of each function's purpose, along with examples and explanations of how these functions interact within a Deep Learning pipeline, the application could serve as a powerful learning tool for users who are not only building models but also gaining a deeper understanding of the underlying principles.

Another important feature to consider for future work is the ability to export the generated models into code that is compatible with various Deep Learning frameworks such as PyTorch, TensorFlow, and Keras. By providing users with the option to download the model in the framework of their choice, the application would significantly improve its usability and versatility, catering to a wider range of preferences and project requirements. This capability would allow users to directly integrate the generated code into their workflows, facilitating seamless transitions between model prototyping and deployment.

Lastly, incorporating interactive tutorials or guided examples could further enhance the application's role as a teaching tool. By bridging the gap between model construction and theoretical understanding, these additions could make the platform more accessible to a wider range of users, from beginners to advanced practitioners. Expanding these aspects would make the application not only more versatile but also a more effective resource for learning and exploring Deep Learning techniques.

Source code for DeepBlocks is available on GitLab at the following link: `https://git.elite.polito.it/filippocaste/deepblocks-msthesis.git`.

Appendix A

Protocol Buffer definition

```
syntax = "proto3";

service Trainer {
  // Train the network
  rpc TrainNetwork (Network) returns (NetworkResult) {}
  // Export to file (onnx or pth)
  rpc ExportNetwork (Network) returns (File) {}
  // Forward single block to the network
  rpc ForwardBlock (Network) returns (ForwardResult) {}
}

message Parameters {
  string key = 1;
  string value = 2;
}

message Node {
  string id = 1;
  string function = 2; // is the pytorch function name
  repeated Parameters parameters = 3;
}

message Edge {
  string source = 1;
  string target = 2;
}

message File {
  bytes file_data = 1;
  string file_name = 2;
}

message Metric {
```

```
34        string name = 1;
35        repeated float value = 2;
36      }
37
38      message Network {
39          repeated Node nodes = 1;
40          repeated Edge edges = 2;
41          repeated Parameters parameters = 3;
42          repeated File files = 4;
43      }
44
45      message NetworkResult {
46        string status = 1;
47        string message = 2;
48        repeated Metric metrics = 3;
49      }
50
51      message ForwardResult {
52        string status = 1;
53        string message = 2;
54        repeated Parameters parameters = 3;
55      }
```

Bibliography

[1] *ML examples and use cases*. https://www.ibm.com/think/topics/machine-learning-use-cases. Accessed: 2024-08-31 (cit. on p. 1).

[2] *ML examples and use cases in healthcare*. https://www.coursera.org/articles/machine-learning-in-health-care. Accessed: 2024-08-31 (cit. on p. 1).

[3] Nikos Tsiknakis et al. «Deep learning for diabetic retinopathy detection and classification based on fundus images: A review». In: *Computers in Biology and Medicine* 135 (2021), p. 104599. ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2021.104599. URL: https://www.sciencedirect.com/science/article/pii/S0010482521003930 (cit. on p. 1).

[4] *What Is a Machine Learning Pipeline and Why It's Important*. https://plat.ai/blog/machine-learning-pipeline/#challenges-associated-with-ml-pipelines. Accessed: 2024-09-01 (cit. on p. 2).

[5] *Machine Learning Concept Drift*. https://www.seldon.io/machine-learning-concept-drift. Accessed: 2024-09-01 (cit. on p. 2).

[6] Eldon Schoop, Forrest Huang, and Bjoern Hartmann. «UMLAUT: Debugging Deep Learning Programs using Program Structure and Model Behavior». In: May 2021, pp. 1–16. DOI: 10.1145/3411764.3445538 (cit. on pp. 2, 9).

[7] Saleema Amershi, Max Chickering, Steven M. Drucker, Bongshin Lee, Patrice Simard, and Jina Suh. «ModelTracker: Redesigning Performance Analysis Tools for Machine Learning». In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. CHI '15. Seoul, Republic of Korea: Association for Computing Machinery, 2015, pp. 337–346. ISBN: 9781450331456. DOI: 10.1145/2702123.2702509. URL: https://doi.org/10.1145/2702123.2702509 (cit. on pp. 2, 10).

[8] Srikanth Tamilselvam, Naveen Panwar, Shreya Khare, Rahul Aralikatte, Anush Sankaran, and Senthil Mani. *A Visual Programming Paradigm for Abstract Deep Learning Model Development*. 2019. arXiv: 1905.02486 [cs.HC]. URL: https://arxiv.org/abs/1905.02486 (cit. on pp. 2, 3, 8, 75).

[9] *Teachable Machine*. https://teachablemachine.withgoogle.com/. Accessed: 2024-09-02 (cit. on p. 2).

[10] *Visual Blocks*. https://visualblocks.withgoogle.com/. Accessed: 2024-09-02 (cit. on p. 2).

[11] Tommaso Calò and Luigi De Russis. *Towards A Visual Programming Tool to Create Deep Learning Models*. 2023. arXiv: 2303.12821 [cs.HC]. URL: https://arxiv.org/abs/2303.12821 (cit. on pp. 2, 3, 7, 27).

[12] Ruofei Du et al. «Rapsai: Accelerating Machine Learning Prototyping of Multimedia Applications through Visual Programming». In: Apr. 2023, pp. 1–23. DOI: 10.1145/3544548.3581338 (cit. on pp. 2, 8).

[13] *ONNX: Open Neural Network Exchange*. https://onnx.ai/. Accessed: 2024-08-06 (cit. on p. 8).

[14] Chao Xie, Hua Qi, Lei Ma, and Jianjun Zhao. «DeepVisual: A Visual Programming Tool for Deep Learning Systems». In: *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. 2019, pp. 130–134. DOI: 10.1109/ICPC.2019.00028 (cit. on p. 9).

[15] *An overview of Machine Learning*. https://www.ibm.com/topics/machine-learning. Accessed: 2024-08-07 (cit. on p. 14).

[16] *Machine Learning methods*. https://www.ibm.com/think/topics/supervised-vs-unsupervised-learning. Accessed: 2024-08-07 (cit. on p. 15).

[17] *Machine Learning libraries and frameworks*. https://www.geeksforgeeks.org/machine-learning-frameworks/. Accessed: 2024-09-03 (cit. on pp. 17–19).

[18] *Keras*. https://www.geeksforgeeks.org/what-is-keras/. Accessed: 2024-09-03 (cit. on p. 19).

[19] *Caffe Model Zoo*. https://caffe.berkeleyvision.org/model_zoo.html. Accessed: 2024-09-03 (cit. on p. 19).

[20] *Caffe*. https://www.geeksforgeeks.org/difference-between-tensorflow-and-caffe/. Accessed: 2024-09-03 (cit. on p. 19).

[21] *Spark*. https://www.geeksforgeeks.org/overview-of-apache-spark/. Accessed: 2024-09-03 (cit. on p. 20).

[22] Mohammad Amin Kuhail, Shahbano Farooq, Rawad Hammad, and Mohammed Bahja. «Characterizing Visual Programming Approaches for End-User Developers: A Systematic Review». In: *IEEE Access* 9 (2021), pp. 14181–14202. DOI: 10.1109/ACCESS.2021.3051043 (cit. on p. 21).

[23] *Protocol Buffers*. serializingstructureddata. Accessed: 2024-09-06 (cit. on p. 21).

[24] *gRPC core concepts*. https://grpc.io/docs/what-is-grpc/core-concepts/. Accessed: 2024-09-05 (cit. on p. 22).

[25] *Between-Subjects vs. Within-Subjects Study Design*. https://www.nngroup.com/articles/between-within-subjects/. Accessed: 2024-09-16 (cit. on pp. 22, 23).

[26] *SEQ Questionnaire*. https://www.questionpro.com/blog/single-ease-question/. Accessed: 2024-09-17 (cit. on p. 23).

[27] *SEQ usefulness*. https://blog.uxtweak.com/single-ease-question/. Accessed: 2024-09-17 (cit. on p. 23).

[28] *What Is a Likert Scale?* https://www.scribbr.com/methodology/likert-scale/. Accessed: 2024-09-17 (cit. on p. 24).

[29] *System Usability Scale (SUS)*. https://blog.uxtweak.com/system-usability-scale/. Accessed: 2024-09-16 (cit. on p. 24).

[30] *SUS questionnaire*. https://measuringu.com/sus/. Accessed: 2024-09-16 (cit. on p. 25).

[31] *Figma: The Collaborative Interface Design Tool*. https://www.figma.com/. Accessed: 2024-08-06 (cit. on pp. 27, 31).

[32] *Medium Fidelity prototype*. https://polito-hci-2023.github.io/materials/slides/09-mid-fi-prototypes.pdf. Accessed: 2024-08-12 (cit. on p. 31).

[33] *Demystifying Training Parameters in Machine Learning: Batch Size, Iteration, Epoch, Learning Rate*. https://medium.com/@weidagang/understanding-parameters-in-ml-training-batch-size-iteration-epoch-learning-rate-a1217d8f80e1. Accessed: 2024-09-05 (cit. on p. 40).

[34] *Stack Exchange: all websites*. https://stackexchange.com/sites. Accessed: 2024-08-19 (cit. on p. 52).

[35] *Flask documentation*. https://flask.palletsprojects.com/en/3.0.x/. Accessed: 2024-08-18 (cit. on p. 53).

[36] *Package for gRPC Python tools*. https://pypi.org/project/grpcio-tools/. Accessed: 2024-08-19 (cit. on pp. 53, 60).

[37] *PyTorch: torch.nn documentation*. https://pytorch.org/docs/stable/nn.html. Accessed: 2024-08-18 (cit. on p. 60).

[38] *The fundamental package for scientific computing with Python*. https://numpy.org/. Accessed: 2024-08-19 (cit. on p. 60).

[39] *The torchvision package consists of popular datasets, model architectures, and common image transformations for computer vision*. https://pytorch.org/vision/stable/index.html. Accessed: 2024-08-19 (cit. on p. 60).

[40] *Hugging Face: State-of-the-Art Natural Language Processing*. https://huggingface.co/. Accessed: 2024-08-19 (cit. on p. 61).