FACULTY OF ENGINEERING OF THE UNIVERSITY OF PORTO

A live IDE for deep learning architectures

Mário Gustavo Fernandes

Integrated Master in Informatics and Computing Engineering

Supervisor: Hugo Sereno Ferreira

Second Supervisor: André Restivo

July 20, 2019

A live IDE for deep learning architectures

Mário Gustavo Fernandes

Integrated Master in Informatics and Computing Engineering

Abstract

During modern times deep neural networks have achieved a level of performance that rivals human accuracy, although explanation on how they make certain decisions is still lackluster. This becomes an issue when there is a need to justify how and why a network is obtaining its results, especially when its representation is treated as a black box.

Visual programming can be defined by the process of illustrating a solution to a problem using visual metaphors such as blocks or flowcharts. Concepts of abstractions and interactivity go hand in hand with visual programming languages (VPLs) and are often the focus of visualization tools. Considering this topic the concept of liveness also fits in quite nicely since the entire idea of VPLs is to provide a responsive user experience along with constant visualization of both the problem and the achieved solution. Liveness, in its essence, is the degree of how responsive and *conscious* a determined software can be. This ranges anywhere from basic pushes of buttons to activate events to a program capable of predicting what a user wants to perform.

Combining all of these fields we have created a tool capable of generating deep learning architectures through the use of visualization and live interaction. This type of software not only has the objective of demystifying the concept of a black box through a basic visual representation which can be interacted with ease, but also creates opportunities for every kind of additional feature and extension.

The main objective of this tool is to diminish the barrier between people who want to work and study deep learning and the minimum required expertise to deal with such projects. Having said this, a quasi-experiment with 16 participants was done in order to have empirical evidence to conduct a study to try and validate the idea that visualization and liveness can make the field of deep learning easier to get into.

Upon the completion of the statistical analysis of the results, it was safe to conclude that the developed software had a positive impact on the ability of people to create both simple and complex DL architectures. Furthermore, it was observed that during the experiment, participants tended to change their natural behavior of programming when using the visual software, adding yet another interesting factor to analyze.

Resumo

Nos dias de hoje, *deep neural networks* alcançaram um nível de desempenho que faz concorrência à precisão humana em vários aspetos, embora a compreensão sobre como são tomadas determinadas decisões ainda seja complicada. Tudo isto torna-se um problema quando surge a necessidade de justificar os resultados obtidos de uma *neural network*, especialmente quando a sua representação é tratada como uma caixa negra.

O conceito de *visual programming* pode ser definido pelo processo de ilustrar uma solução para um problema com recurso a metáforas visuais, como blocos ou fluxogramas. Conceitos de abstração e interatividade andam de mãos dadas com linguagens de programação visual (LPVs) e são frequentemente o foco das ferramentas de visualização. Considerando este tópico, o conceito de *liveness* também se encaixa muito bem, já que toda a ideia das LPVs é fornecer uma experiência responsiva aos utilizadores, juntamente com uma visualização constante do problema e da solução obtida. *Liveness*, na sua essência, é a capacidade de reação e o grau *consciência* que um determinado programa possa ter. Isto varia desde *clicks* básicos em botões para acionar eventos até um programa capaz de prever o que um utilizador deseja fazer.

Juntando todas estas áreas, criámos uma ferramenta capaz de gerar arquiteturas de *deep learning* através do uso de visualização e interatividade *live*. Este tipo de software não só tem o objetivo de desmistificar o conceito de caixa negra através do uso de uma representação visual básica com a qual se pode interagir com facilidade, como também consegue abrir portas para a criação de novas funcionalides de forma a estender a ferramenta atual.

O principal objetivo desta ferramenta é diminuir o grau de conhecimento mínimo necessários para conseguir interagir com trabalhos e estudos sobre *deep learning*. Dito isto, foi feita uma *quasi-experiment* com 16 participantes para conseguir obter dados que sirvam como base para um estudo cujo objetivo é validar a ideia de que a visualização e *liveness* possam facilitar o acesso à area de *deep learning*.

Após a conclusão da análise estatística dos resultados, é seguro concluir que a ferramenta desenvolvida teve um impacto positivo na capacidade das pessoas conseguirem criar arquiteturas simples e complexas de *deep learning*. Além disso, observou-se que durante a experiência, o comportamento dos participantes ao programar visualmente mudou até um certo ponto, quando comparado a maneiras mais tradicionais, acrescentando assim um tópico que merece ser estudado mais afincadamente.

Acknowledgements

Firstly I would like to thank my supervisors Hugo Sereno Ferreira and André Restivo for all the help, support and motivation they gave me each day. It was truly a remarkable experience to work with them, not only due to their continuous inputs on the project each week but also because of their friendship towards me and my colleagues, pressuring us to not become lone wolves.

My heartfelt gratitude towards my parents and my sister who never stopped supporting me throughout my life while helping me in any and every way they could.

To my friend António Gonçalves, whose perseverance I admire. Thank you for your friendship, support and car rides as you have helped me become a better me.

To Inês Gomes and Cat R, whom I hold dear to me, for their immeasurable patience. I would never be able to have so much fun and experience so many adventures without you during all these years and for that, both of you have my utmost gratitude and my everlasting friendship. You have really helped me to appreciate so many things in life, especially coffee breaks. My liver is not really fond of any of you though.

To Miguel Freitas for his trustworthiness, friendship and for all the company and help during all those hackathons. I still want to implement that upper downer system-like infinite program.

To my computer for not being like my printer.

Many thanks to Cátia Ramos and Luís Silva for all their friendship and ideological discussions at 3 am. I love to argue.

To Sara Fernandes and Diogo Campos for listening to my rants and keeping me *mildly* sane. One of you greatly inspired and helped me throughout this semester.

To all my friends from Bámos. You're all amazing and together you're everything a friend could ever ask for.

Mário Gustavo Fernandes

"Nevermore"

Edgar Allan Poe, The Raven

Contents

List of Figures

LIST OF FIGURES

List of Tables

Abbreviations

ML	Machine Learning
DL	Deep Learning
IDE	Integrated Development Environment
VPL	Visual Programming Language

Chapter 1

Introduction

Contents

In this chapter an introductory context is given in § 1.1, briefly explaining how the topics addressed in this dissertation are relevant to the current state of the world. An overview of what led the authors to do this project and its main expected objectives are described in § 1.2 and § 1.3, respectively. Finally, a summary of how this document is structured is presented in § 1.4.

1.1 Context

The world as we know it is being progressively changed by science and technology each day. Every place people go to, every building they enter and everywhere they look technology will be there. It has become such an intrinsic part of our lives that it is actually hard to imagine living in a world without a phone or access to the internet.

A result of this is a shift in people's needs and expectations. Nowadays a person no longer expects a phone to only be able to make and receive calls: it needs to be able to take pictures and run several applications. The same can be said for almost any consumable good, be it a smart fridge that automatically lists the items it's missing, a television that can open a web browser or even a watch that measures your heart-rate. The conclusion is clear, people want more.

A more recent improvement on everyday items is their integration with software capable of directly interacting with people, specifically facial recognition programs. These are now commonly used, for example, to unlock a phone. Commercial products using this kind of technology were made available roughly around the middle of the decade this work is being written on. This, together with many other factors of the same nature caused a spike in the interest pertaining to an artificial intelligence area: deep learning (DL). This can be proved analyzing Google's trend of a popular ML method: deep learning, as seen in figure Figure 1.1.

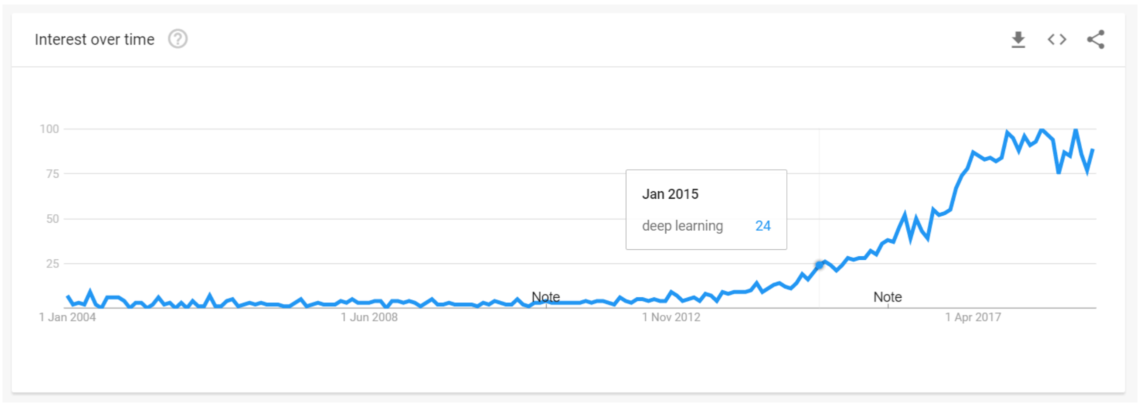


Figure 1.1: Google Trend on Deep Learning

1.2 Problem and Motivation

As it will be reviewed in Chapter 2 (p. 5), in recent years a lot of work has been done with the objective of diminishing the barrier between those that want to contribute to ML and DL and those that have the expertise to do it, some of the most popular being DL libraries such as TensorFlow[1]. These types of projects that create abstractions of concepts of a determined field of study are of great importance but generally have some inconveniences, especially for people that are either not used to developing programs or already know how to use similar libraries and do not want to change. In addition to this, more specific problems such as visualizing of how a neural network works in-depth is a very difficult task to perform if the only available tools are the aforementioned libraries. This constitutes a problem in the development of a product using DL, in its presentation and in its explanation, possibly resulting in adversities for people that wish to experiment with such algorithms.

With the sudden interest in ML and DL, an increase of scientific work on this field has been produced lately, namely in tools to analyze neural networks in many different ways, as can be checked in systematic literature reviews such as [GTC+18]. These works mainly focus on niche visualization concepts of a determined DL model and have very narrow use-case scenarios. The resulting tools of a lot of these projects also expose how bad the architecture of this type of software which in most cases can't be scaled or extended.

Having this in mind, the creation of a deep learning visual and interactive development tool, capable of supporting multiple DL libraries and providing the option of being extended for niche visualization tasks was deemed a pertinent topic to explore. The interest in this matter culminates

[1] https://www.tensorflow.org/ (Retrieved: 02/07/2019)

when a tool such as the one being conceptualized, provides features such as live development or the ability to customize the software, integrating it with other tools.

1.3 Goals

The main goals of this work can be divided into 4 requirements pertaining to the developed software:

R1 : The tool should be able to create represent and edit DL models and allow direct interaction with their visualizations.

R2 : The tool should have a modular architecture and should not force a user to depend on it.

R3 : The tool should have integrated concepts of liveness, namely stepping into level 4 of Tanimoto's hierarchy.

R4 : The tool's additional features should be based on gathered conclusions from a previously made literature review.

The stated requirements cover in an abstract manner the expected work to be developed in this dissertation.

R1 states that the focus of this tool will reside in the creation of architectures rather than the training of models and analysis of the outputs of the algorithms. A model creation environment will be made as well as the possibility of interacting with their layers and hyperparameters.

R2 explains that the architecture of the tool to be developed should be able to be extended. This means that the editor should be able to support the addition of new features without destroying its core objective. It also states that when using the tool, a user should be given the option of exporting their results at any point.

R3 states that the development of the tool needs to have in consideration level 4 of Tanimoto's liveness hierarchy [Tan15]. Objectively this means that besides being interactive, the tool needs to explore concepts of continuously and automatically performing changes to itself as well as the possibility of giving suggestions to the user.

To finalize and considering R4, an in-depth exploration of earlier works needs to be done in order to justify the features of the tool. A comparison between gathered machine learning visualization tools needs to be done so that concepts that have already been thoroughly worked on are clear and topics lacking scientific work are investigated upon.

1.4 Document Structure

This document is divided into 6 chapters.

Introduction, Chapter 1 (p. 1). The introductory chapter, this one, gives insight to the context of the problem as well as what it aims to accomplish.

Literature Review, Chapter 2 (p. 5). The second chapter explains all of the main concepts related to this work as well as reviewing a collection of projects that have been done in this field. It also correlates these projects with the Tanimoto's liveness hierarchy and compares them so that it becomes possible to identify what is the focus of these projects and which areas could benefit from research such as the one being proposed to be done.

Problem Statement, Chapter 3 (p. 19). This chapter describes the problem by formulating a hypothesis and associated research questions which the following chapters will provide the needed information to answer them.

Proposed Solution, Chapter 4 (p. 23). The proposed solution chapter explains in-depth how the project was conceptualized and how it was implemented. All of the difficulties, features and architectural choices are described along with the software's capabilities in terms of scalability and extension.

Evaluation, Chapter 5 (p. 37). This chapter provides an in-depth explanation of the performed quasi-experiment and statistical analysis of the obtained results. This is one of the main focus of the dissertation as it serves as the only basis which conclusions can be drawn from regarding the entire project.

Conclusions, Chapter 6 (p. 63). Concerning the final chapter, a summary of the dissertation is made, along with answers to the research questions based on the study that was made. As a final note, it is also referred to what kind of possibilities arise with the conclusion of this thesis, namely possible work that can be done using this work as a basis.

Chapter 2

Literature Review

Contents

This chapter exposes the main concepts of this work by explaining their backgrounds, their importance and how they relate to each other. The first section clarifies what deep learning is and gives focus to the networks that are used on this work: convolutional neural networks. The second section gives the input on what liveness in software development is by presenting its associated hierarchy, the main criticism around it and how it is applied in the modern world. The third section explains what visual programming is as well as relating it to liveness and how generic, present-day software uses those concepts. The fourth and most critical section on this chapter relates all of the previously mentioned concepts, essentially by reviewing modern tools that incorporate said concepts.

2.1 Deep Learning

In this section, the concept of deep learning is explored. A brief context is provided in subsection § 2.1.1 as well as an explanation of what kind of problems it solves. Subsection § 2.1.2 provides an introduction to Convolutional Neural Networks and why is it a relevant part to expand on.

2.1.1 Background

In 1943 Warren S. McCulloch and Walter Pitts introduced the McCulloch-Pitts Neuron which is a highly simplified model of a neuron [MP43]. This model consists of a set of inputs I, their assigned weights W and a single output y and can be mathematically described with the following equations:

$$Sum = \sum_{i=1}^{N} I_i W_i \tag{2.1}$$

$$y = f(Sum) \tag{2.2}$$

The y variable is seen as a binary output which changes if the sum achieves a higher value than the defined threshold for that neuron. The function f simply compares the value of the sum with the threshold and assigns the correct value to y. This can be called the activation function.

This model was extremely simplistic since it only translated inputs into a binary result yet had a considerable computing potential, notably because of how generic it is, having a multitude of applications in diverse fields [Zur92]. Thus, interest in developing a more flexible neural model emerged.

In 1958, Frank Rosenblatt enhanced McCulloch's model [Ros58] by merging it with Donald Hebb's principle of having neurons change their weights according to the behaviors of other neurons. This improvement was denominated as the perceptron. In today's vocabulary, it is said that this approach consists of a set of neurons in one layer. By having a single layer, this architecture is only capable of separating data with a single line, meaning non-linear problems, such as ones with XOR inputs, are impossible to be resolved with the presented model, as can be exemplified below 2.1.

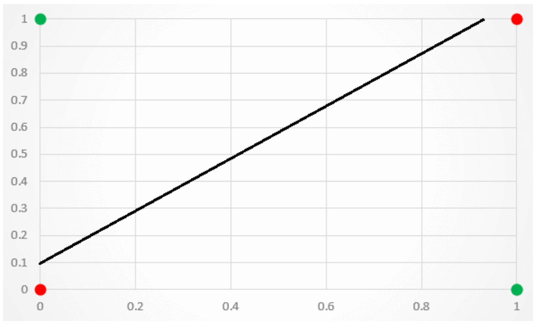


Figure 2.1: The perceptron: Single Layer version

Problems like these can be resolved by adding more layers, which are referred to as hidden layers, and inevitably adding more complexity. An example of the added complexity is the added difficulty of finding suitable weights to assign to each neuron in order to always have a correct output from the network. To solve this issue David E. Rumelhart proposed the backpropagation algorithm in 1986, which automatically solves this issue by adjusting every weight of every neuron by comparing the current output value to the expected value; this process is run indefinitely or until a satisfactory output is attained. This architecture (or customized versions of it) is used in the present-day when any problem needs to be solved with deep learning due to how general it is and how broad the spectrum of problems it can solve is. An example of what this improvement does to the previous case can be viewed in figure Figure 2.2

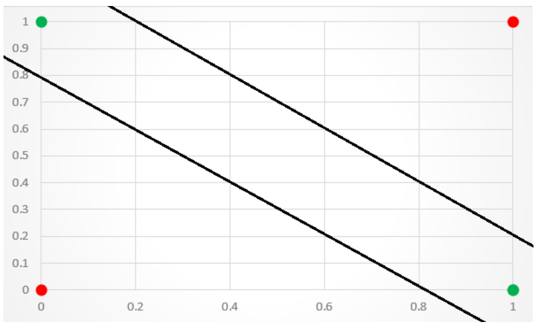


Figure 2.2: The perceptron: Multiple Layers version

2.1.2 Convolutional Neural Networks

Convolutional Neural Network (ConvNet or CNN) is a deep learning algorithm and a popular choice for problems dealing with images as an input such as image recognition and classification.

The first real viable application of CNNs was done by Yann LeChun et al. [LBBH01] with the introduction of the LeNet5 model. This architecture's applicability was based on reading and identifying images with digits, such as zip codes. However, the computer vision community disregarded this work at the time deeming it a non-scalable approach to read "real-world" images, mainly due to computational limitations.

In 2012 Alex Krizhevsky, Ilya Sutskever and Geoff Hinton removed the stigma around the referred architecture by using an adaptation of it (AlexNet [KSH12]) to win the ImageNet Large Scale Visual Recognition Challenge with error margins 10% lower than the second entry [Ima12]. The main difference, besides a large quantity of data, was the use of the computational power of GPUs.

A general idea of a CNN architecture is presented below and in figure 2.3.

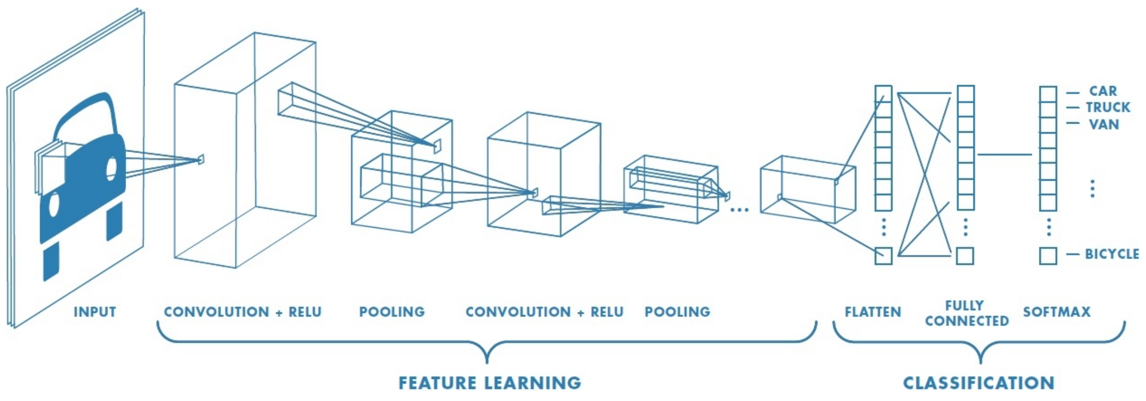


Figure 2.3: Simplified view on a CNN architecture

2.1.2.1 Convolution Layer

This is the starting point of any CNN. After processing the image and converting it to a matrix of dimensions height x width x dimension (the dimension having the value 3, for red, green and blue), this layer receives that information as input and applies filters to it. By applying a filter to the image, operations such as detecting edges or blurring the image are made, thus creating a new output. These are usually applied multiple times in a CNN with the objective of learning features of an input.

2.1.2.2 Rectified Linear Unit

Rectified Linear Unit (ReLU) is an operation that is generally used after every convolution. Its purpose is to add non-linearity to the network by mapping negative values to 0, commonly referred to activation, thus making the training faster and more effective since only the activated features are carried out to the next layer. ReLU is usually selected as the default function for this step due to performance issues although other non-linear functions exist such as tanh or sigmoid.

2.1.2.3 Pooling

These layers have the objective of reducing the number of parameters of a given input and its associated operation, spatial pooling, can also be called subsampling or downsampling. Different types of pooling exist such as max pooling, average pooling, and sum pooling although they all try to improve the input representations by making it smaller. This step usually controls overfitting, reduces the chances of the network to react to smaller and insignificant changes and helps to achieve an almost scale-invariant representation of the image, allowing to identify objects independently of their position.

2.1.2.4 Flatten

Flattening is a simple operation which transforms the multiple pooled features from previous layers into a vector of data, ready to be classified.

2.1.2.5 Full Connection

A fully connected layer has the goal of associating classes to the data it receives. Taking the example of single digits, this layer would have 10 possible classes to use for classification (digits from 0 to 9). The layer then correlates its given input to said classes, attributing a level of confidence according to comparisons made between the features it's processing to the ones the output classes have.

2.1.2.6 Softmax

Softmax is a classifying function generally used in CNNs. Its purpose is simply to normalize a given vector into a probability distribution and is usually the last step to take in a CNN.

2.2 Live Development

In this section, the concept of liveness in a program is analyzed with the main focus on why it's relevant, the effect that it has on developers and how modern programs use it.

2.2.1 Liveness Hierarchy

As Steven L. Tanimoto explains in [Tan13], replacing a light switch with a dimmer can be done in one of two ways: "(1) first turn off the circuit breaker, or (2) wire it hot". The second case is obviously more dangerous but also faster, especially since there is no need to turn the circuit breaker back on every time it is needed to see if the job was done successfully. When it comes to programming, the danger associated with techniques of this nature is heavily diminished.

With this in mind, Tanimoto created a liveness hierarchy Figure 2.4 which can be used to evaluate how live a system is.

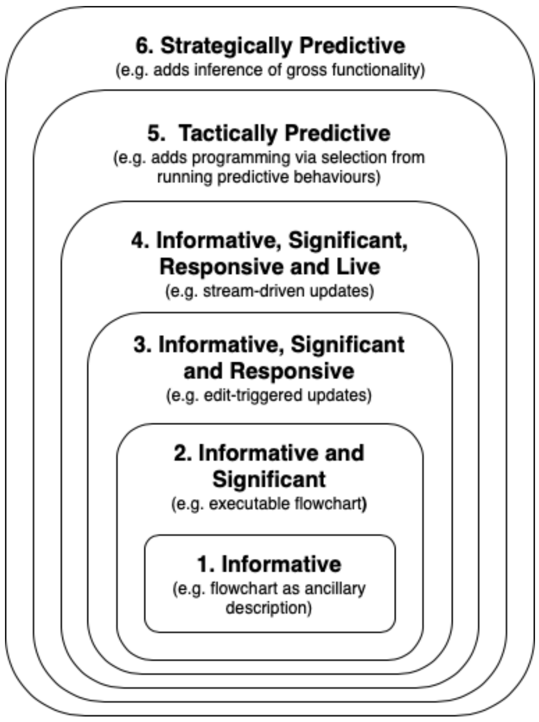

6. Strategically Predictive
(e.g. adds inference of gross functionality)
5. Tactically Predictive
(e.g. adds programming via selection from running predictive behaviours)
4. Informative, Significant, Responsive and Live
(e.g. stream-driven updates)
3. Informative, Significant and Responsive
(e.g. edit-triggered updates)
2. Informative and Significant
(e.g. executable flowchart)
1. Informative
(e.g. flowchart as ancillary description)

Figure 2.4: Liveness hierarchy proposed by Seteven L. Tanimoto

Considering the proposed hierarchy, level 2 refers to programs that require the user to ask the computer for responses, much like the first case of the earlier given example of turning off the circuit breaker. In level 3, as opposed to level 2, programs don't end until the user specifically orders it to, but waits for user's events, usually by clicking or typing, and gives some sort of output. Any level higher than third is closely related to the example of wiring it hot, such as level 4 which removes the need for explicit events of the user. In this level, programs are kept running as well as changing its behavior in a just-in-time (JIT) manner according to the user's changes. Levels 5 and 6 explore automation regarding changes to the program itself. Programs in liveness level 5 will suggest changes the user can make and finally level 6 automates those changes, usually by producing and running a separate program with changes which it has predicted.

For a better understanding of this hierarchy, a use-case of coding a program can be analyzed. Level 1 can be exemplified as a unified modeling language (UML) class diagram: it's merely information on how to implement a program and can't be executed. Level 2 environments can be viewed as writing a program in C compiling it through the invocation of its associated Makefile. Level 3 greatly improves the programmer experience, for example, with the use of an IDE which will be continuously running, waiting for the user to click a compile or run button. Any modern IDE such as Eclipse [Fou01] and IntelliJ IDEA [Jet01] explore the concepts in liveness level 4 using JIT compilation to inform the programmer of errors in the code without the need of explicitly asking for the program to compile. With the aid of code generation techniques and user-defined settings, these IDEs also provide suggestions to the programmer in multiple cases, stepping up to level 5, such as finishing a variable or method name or in some specific cases, generating an entire code block, albeit usually being simple ones such as getter and setter methods.

2.2.2 Criticism

Tanimoto himself acknowledges that liveness "is not a panacea for programming environments" [Tan13], notably referring to criticism related to the possible uselessness of liveness in short programs, execution having passed the location of the updates and the strain live development puts on a computer's resources and computational power.

All of these critics are discussed and solutions to all three of the mentioned issues are presented in [Tan13]. The first and second critique can be refuted with the concepts of level 4: changes made to a program, no matter how small it is, can be viewed nearly instantaneously and, with the use of breakpoints, adjusting an "auto-repeat" mode or setting a start and stop section, full control of the liveness of a program can be achieved. The third and final major issue could have been a problem in earlier times but any modern computer nowadays can deal with the added overhead of lively updating a program, especially with the use of threads and parallelism between tasks.

2.2.3 Modern Live Development

As it already been established in subsection § 2.2.1, modern IDEs already make heavy use of many live concepts explored mainly on levels three, four and five, with the main objective of heavily reducing feedback loops [ARC+19]. Further establishing the potential of live development, during modern times different techniques and tools can be used for different languages to make use of liveness levels. Clear examples of this can be examined in some JavaScript plugins like nodemon [web10] which simply reloads the entire program automatically according to user-settings or plugins based on code injection such as the live reload (commonly referred to as hot reload) features in Android Studio and Xcode of mobile applications.

The usability and interest of this topic go even further when studies related to the efficiency of live programming are made. [KKKB14] is one of these studies which analyze both the behavior of developers using live coding as well as their performance. The experiment described in this paper consisted of having two similar test groups of developers fix bugs in a program, one using live coding and the other group not. After a total of 205 bugs corrected by each developer, the paper's first hypothesis was verified which referred to the group using live coding would have an overall shorter time fixing all of the bugs. This happens not only because of the benefits live development brings to the table but also how developers adapt themselves to work with these technologies. Rather than a standard sequential approach of fixing the bugs by writing the entire source code and then debugging, the majority of the developers in the group using live coding used an incremental approach, fixing the code as it was written since the output would be viewed instantly.

2.3 Visual Programming

This subsection presents an overview of what visual programming is and its advantages and disadvantages as well as some examples. It also explains how some it's connected to liveness and how modern and sophisticated tools make use of these concepts.

2.3.1 Concept and Background

Visual programming refers to a different way of creating software: through interaction with a visual representation of code rather than writing the code itself. The main reason behind the usage of this approach is to abstract the user of the existing code, making the creation of the program usually faster and simpler. This becomes evident when analyzing studies such as [SLRGVC16] which besides proving that visual programming languages (VPLs) provide an easier way to program, they often make programming itself more engaging, especially towards newer developers. This last aspect also improves the learning capability of users of a given topic.

Having said this, visual programming is not just an improvement on programming itself, possibly limiting what a programmer can do or even providing an abstraction layer above what is intended, making the user learn concepts on a very superficial level. However tools like Scratch [MRR+10], Alice [Con97] and Greenfoot [Köl08] have become very popular choices to teach younger audiences computational thinking regarding some paradigms. According to [UCK+10] and [Goo11], Alice and Greenfoot are good choices to learn object-oriented programming, preparing a user to deal with languages such as Java and Greenfoot is also a good choice to learn concepts such as public-key cryptography. Scratch and Alice are also good choices to learn animations and easy to use for storytelling. Teaching using tools like the ones mentioned above is often referred to project-based learning and create environments for students to demonstrate their capabilities by solving complex problems, thus giving them better preparation for "real-world" situations.

2.3.2 Blocks, Liveness and Modern Applicability

All the previously mentioned software as well as the majority of visual programming tools make use of block-based programming which is a methodology referent to the combination of virtual blocks, that represent an excerpt of code, in order to create a program. This is usually accompanied by an interaction-based dragging and dropping events as can be seen on an online example such as Google's Blockly [webb].

Tools that use block-based coding usually also make use of liveness concepts which enhance the capabilities of this methodology by further reducing the cognitive load of programming [Tan15]. The process of incorporating liveness in these tools, such as warning a user of potential bugs in the program while it is being created not only caters to short attention spans but also reduces the plausible causes for said bugs.

A good example of a tool with these features is RapidMiner [KD14] which is a block-based tool with a certain degree of liveness. RapidMiner is a sophisticated tool which people can make use of it to learn and understand overall concepts and algorithms of data science but can also easily be used for "real-world" problems as can be seen in [Wow15] and [KSK14].

Another usage of visual programming can be seen in modern game engines such as Unity [uni17] which has a drag-and-drop, interactive environment that can be used to create, edit and debug games, as well as a typical text-based coding section. As the referred white paper title states "Make games - not tools", clearly inciting users to focus on their main objective of creating games rather than spending valuable time on creating a platform and tools to assist in the creation of the said game. This is one of the main focus of using visual development: abstracting large portions of code which virtually end up always being similar so that the developer focuses on reaching a solution.

2.3.3 Conclusions

In this section, it was possible to understand what visual programming is and why it is relevant to use. It was also possible to review multiple examples of different use-cases that use visual programming, both in academic and industrial environments, and how it is connected to liveness. For these reasons, the usage of visual programming concepts in an IDE for deep learning is a pertinent topic to explore.

2.4 Visual aided Machine Learning

In previous subsections, deep learning was presented as a solid solution to some complex and non-linear problems as well as how live development and visualization greatly aids a person during all phases of the development of a project. In this subsection, all of these concepts are used to describe the current state of the art of visualization tools for machine learning.

At the time this work is being written, the topic at hand has been receiving a lot of contribution over the last few years [GTC^{+}18] [YS18] [HKPC18]. Although, most tools presented in the referred surveys could be improved upon when it comes to their usage, especially by increasing their reaction to the user's behavior.

To further explain this, the table Table 2.1 presents all of the encountered tools and compares them based on their main features, general liveness level when comparing to the previously explored Tanimoto's liveness hierarchy and overall description. The compared features are composed of key elements that could make part of an ideal live IDE for machine learning and are described below.

- Model Creation (MC) - the tool gives a user an environment to create and edit custom ML models rather than using provided examples or importing previously created ones with the aid of other software.
- Feature Understanding (FU) - the tool allows a user to understand why a feature is being considered to be relevant by the model; can be evaluated with a • if the tool allows for a

basic understanding and with a + for a more in-depth analysis such as checking the output of a neuron in a given iteration and its effect on the model.

- Heatmap (HM) - the tool is able to generate relevant heatmaps to the problem at hand.
- Evolution Representation (ER) - the tool presents relevant information on how a sample was interpreted from the start-up to a determined iteration. A basic example of this is a visualization of the considered labels attributed to the sample and when they suffered changes.
- Neurons Interaction (NI) - the tool allows a user to directly interact with neurons performing low-level adjustments such as moving a neuron to another layer, deleting it or even change its weight.
- Data Flow Visualization (DFV) - the tool allows a user to view precise statistics regarding the data of a model such as averages and variances of activations of neurons in a given iteration.
- Real-Time Interaction (RTI) - the tool allows a user to interact with its machine learning capabilities in real-time. A use case of this is stopping the training of a model, altering a specification like a neuron's weight and resuming it with the new changes. A tool that has this capability is automatically considered to have a liveness level of 4.
- Liveness Level (LL) - the liveness level conferred by the author according to Tanimoto's liveness hierarchy explained in section 2.2. This level portrays how reactive and interactive the tool is when analyzing its capabilities of creating and editing projects in real-time.

2.4.1 Individual Tool Review

In this subsection, a brief analysis of every tool that was presented in table Table 2.1 is made by specifying what makes them unique as well as aspects that could be improved, regarding the topic of this work.

CNNComparator, 2017 [ZHP+17] - Web-based tool that allows a user to import two CNN model snapshots and generates statistics such as histograms comparing them both.

TensorFlow GraphVis., 2018 [WSW+18] - Tool that allows a user to create a deep learning model diagrams in detail.

LRP Toolbox, 2016 [GRNT16] - Python scripts which can be edited and need to be run separately each time.

FeatureVis, 2016 [FG16] - MATLAB library which can be used to generate heatmaps regarding selected features.

RNNVis, 2017 [MCZ+17] - Tool in development that closely details the evolution of (limited) pre-trained RNNs.

Grad-CAM, 2016 [SCD+16] - Python-based software that provides a reliable way for object detection and labeling through CNN interpretations. It can be run multiple times as a video

Tool	MC	FU	HM	ER	NI	DFV	RTI	LL
CNNComparator, 2017 [ZHP+17]		+	●					1
TensorFlow GraphVis., 2018 [WSW+18]	●							1
LRP Toolbox, 2016 [LBM+16]		●	●					2
FeatureVis, 2016 [FG16]		●	●					2
RNNVis, 2017 [MCZ+17]		●		●				2
Grad-CAM, 2016 [SCD+16]		●	●					2
CNNVis, 2017 [LSL+16]		+		●	●			3
Rauber et al., 2017 [RFFT17]		●		●	●			3
Zahavy et al., 2016 [ZBM16]		●	●					3
DGMTracker, 2018 [LSC+18]		●	●	●	●	●		3
Activis, 2018 [KAKC17]		+	●	●				3
DeepEyes, 2018 [PHG+18]		●	●	●				3
Deep View, 2017 [ZXZ+17]		+	●	●				3
RNNbow, 2017 [CPM+18]						●		3
Yosinski et al., 2015 [YCN+15]		●		●				3
Alsallakh et al., 2018 [AJY+17]		+	●	●				3
LSTMVis, 2018 [SGPR16]		●	●	●				3
ShapeShop, 2017 [HHC17]		●		●				3
Adversarial-Playground, 2017 [NQ17]		●						3
Olah, et al., 2018 [OSJ+18]		●		●				3
GANViz, 2018 [WGYS18]		+	●	●				3
TensorFlow Playground, 2017 [SCS+17]			●	●	●		●	4
ReVACNN, 2016 [CSP+16]		●	●	●	●		●	4
Harley, 2015 [Har15]		●		●			●	4
BIDViz, 2017 [QLZT17]		●		●			●	4
LAMVI, 2016 [RA16]	●	●	●	●			●	4
Seq2Seq-Vis, 2018 [SGB+18]		●		●			●	4
Carter, et al., 2016 [CHJO16]		●	●				●	5

Table 2.1: Comparison between all the relevant tools encountered in [GTC+18], [YS18] and [HKPC18]. All works that did not have a good enough description of the used or developed visualization tool were discarded since liveness level is closely related to the behaviour of the software.

processing software to simulate a stream-driven application but in reality, only python scripts are continuously being invoked.

CNNVis, 2017 [LSL+16] - Low-level CNN analysis tool that allows a user to view what features each neuron learned and how they are translated to high-level features. The tool responds to clicking and dragging events when interacting with neurons.

Rauber et al., 2017 [RFFT17] - Private tool that allows the visualization of multiple phases of training a model.

Zahavy et al., 2016 [ZBM16] - Tool for analyzing DQNs in detail through heatmaps.

DGMTracker, 2018 [LSC+18] - Tool that allows users to improve their current model training process by closely analyzing its evolution.

Activis, 2018 [KAKC17] - Tool deployed in Facebook that closely details training of a model with a responsive UI both in the model representation and its evolution. Future work is expected to tackle higher liveness levels with features such as the suggestion of unneeded neurons.

DeepEyes, 2018 [PHG+18] - C++-based tool that allows a user to analyze the evolution of training a model as well as identify optimizations such as unused layers.

Deep View, 2017 [ZXZ+17] - Framework for deep learning visualization that allows a user to evaluate the layer's and neuron's evolution.

RNNbow, 2017 [CPM+18] - Web application used to visualize the relative gradient contribution from an RNN, giving insight on how the network is learning. Responsive UI.

Yosinski et al., 2015 [YCN+15] - Tool that allows a user to feed images or a video stream and have it processed. During that phase, the activations of the layers can be seen and interactively enlarged.

Alsallakh et al., 2018 [AJY+17] - Tool that uses GoogLeNet that allows a user to view statistics of a model such as its confusion matrix and select parts of it so see result samples. The tool also generates neuron's response maps and has a visual feature detector.

LSTMVis, 2018 [SGPR16] - Python and JavaScript-based tool that allows a user to examine and filter hidden states of an RNN. The user can directly select an extract of text from a hypothesis and see its associated hidden states. Data sets are loaded by editing a YAML file.

ShapeShop, 2017 [HHC17] - A web-based application that allows a user to select simple shapes that act as training data, train the model itself and visualize what the network has learned.

Adversarial-Playground, 2017 [NQ17] - Web-based educational tool that allows a user to understand how adversarial examples are generated and their toll on DNN networks.

Olah, et al., 2018 [OSJ+18] - Tool that enables a user to interactively hover the mouse through different parts of an image and see the activation of neurons related to that specific part. These activations, rather than being represented through a typical abstract vector, are shown as a semantic dictionary, ordering the most relevant activations and showing associated labels and images.

GANViz, 2018 [WGYS18] - Tool that allows a user to interpret with detail CNN models and evaluate them. It also shows comparisons between features in each image such as real/fake and positive/negative.

TensorFlow Playground, 2017 [SCS+17] - Live neural network analyzer for pre-processed datasets. Allows a user to lively visualize the development of a dataset and directly interact in real-time with some of the network's features such as the neuron's weight.

ReVACNN, 2016 [CSP+16] - Simple web-based tool that allows a user to interact with a neural network in real-time. Limited statistics are presented.

Harley, 2015 [Har15] - Web-based tool that allows a user to draw numbers and have them be classified in real-time. Responsive visualization.

BIDViz, 2017 [QLZT17] - Scala-based application that allows a user to visualize a simple evolution of a neural network and interact with it in real-time. Statistics and visualization are very limited and its interaction is based on the user coding.

LAMVI, 2016 [RA16] - Although this tool has many features and a relatively high liveness level, most of the features are very superficial. All the interaction with the model is done by editing text parameters and the only real-time interaction it has is analyzing the evolution of a model when querying the result with a specific word. The program enables a user to query multiple words at the same time obtaining a graph in real-time for each word as well as giving the option of resetting and starting the training on demand.

Seq2Seq-Vis, 2018 [SGB$^+$18] - Web-based tool that allows a user to visually debug sequence to sequence models. A demo with the intent of translating Dutch to English shows the tool generating samples with a given confidence level for a given phrase. From then on, the user can interactively see the context captured by the encoder and decoder and even correct the output by selecting translations that do not have the best confidence.

Carter, et al., 2016 [CHJO16] - Interactive web-based tool that combines predictive analysis with generation of output. Although it steps into the level 5 liveness hierarchy, it severely lacks any type of basic customization such as altering the model or feeding it more samples.

2.4.2 Commercial Tools

In relation to commercial products on the topic at hand, some polished tools exist being the main ones the Deep Cognition [weba] and Sony's Neural Network Console [webc].

Deep Cognition is a full-fledged deep learning IDE providing the user with the ability of interactively creating a model, training said model and view statistics on it. It also supports multiple frameworks such as Keras and Caffe[1] and allows lives performance monitoring when training a model.

Sony's Neural Network Console does not have so many features as Deep Cognition but some of its features are more polished. This can be seen when creating a model, which Neural Network Console allows to select copy and paste multiple layers from a model, or when analyzing previously trained models, which is possible to navigate through multiple previous models and view individual iterations of each model.

Overall these tools are much more complete than the ones found in scientific works, but also have some issues. For starters, they are closed source, meaning they have 0 extensibility for anyone to develop on top of those tools, for example, to test new visualization techniques. Secondly, most of them are either region-restricted or require payments to either access the tool or some of its features. It is also worth mentioning that these tools mainly focus on usability since, besides quality-of-life characteristics, the tools merely provide features that already exist in previously mentioned scientific works.

2.4.3 Discussion

After analyzing the previously presented machine learning visualization tools it is possible to retrieve some information on this topic.

[1]`https://caffe.berkeleyvision.org/` (Retrieved: 02/07/2019)

First off, it is easy to understand that the majority of the existing tools are focused on the third liveness level, meaning they work based on the user's actions such as clicking and writing. Some tools are already including a stream-driven architecture further increasing their liveness level but they are usually developed as a proof-of-concept, meaning that they are made for niche use cases. These specific features are taken to the extreme both on liveness level five and one. On level five a handwriting generation work is presented and even though it produces gibberish it works for the purpose that has been made. On level one a software that compares the output of multiple models at the same time but only in a single iteration is presented as well as a tool that has the sole purpose of helping to draw machine learning models.

Secondly, the presented tools focus essentially on understanding features, generating heatmaps and representing the evolution of models. People that use the tools for these features will need to interpret the results, tune the model in separate software, load the new model and repeat the process until they are satisfied with it. A good example of this is the DGMTracker which in its associated paper is described a use-case with invited experts; these experts visualized the training of a model and associated statistics and, although they could diagnose some aspects of the model they could not directly apply their suggested changes and see the outcome [LSC+18]. This can clearly be improved by having the tool include other interactive features such as the ones presented. An excellent example of a tool that puts most of the analyzed features together is LAMVI albeit is very simplistic and some interaction with it is based on editing text [RA16].

On a final note, it is important to refer that none of the presented tools present the user with clear suggestions on how to improve a given model. This means that although a person can use tools like these as means of debugging a machine learning model they need to know what they are looking for in the statistics generated by the tools, which might not always be the case.

2.5 Summary

This chapter provided the reader with enough information to understand the context of the developed project and how the implemented features are distinguished from the existing tools. The last section also shed light to the current state-of-the-art works on the topic at hand, directly relating them to the concept of liveness, further explaining how this technique can constitute a substantial improvement on most programs.

With the conclusion of this chapter, the reader should have a general understanding that deep learning can be used to solve non-linear problems and that liveness and visualization greatly improve a user's interaction with the software. It is also important to realize the main differences between the problems both the scientific community and the industry try to solve. Most scientific works try to explore new visualization techniques on neural networks or improve previously made ones; Commercial products tend to be a general solution more focused on usability with little to no extensibility.

Chapter 3

Problem Statement

Contents

This chapter describes the problem at hand with more detail and what type of work can help solve such issues or at least improve the current way of creating deep learning architectures.

In addition to this, it also presents the problem under the form of research questions that this thesis attempts to give answers to, justifying said answers with empirical evidence.

3.1 Current Issues

As referred in Chapter 1 (p. 1), the interest in deep learning has been increasing, meaning that tools that aid the creation of DL architectures is a prominent topic to do research in.

In Chapter 2 (p. 5) multiple interactive tools that provide help regarding this subject have been presented and analyzed in terms of features and liveness level. The main conclusion that it was reached on said chapter was that there exists a lack of a tool capable of completely satisfying all the following topics to their fullest extent:

Architectures of DL models. Almost every tool presented is based on either *hard coded* examples in order to satisfy a specific use case or focus on a single DL architecture, such as [NQ17] and [SGB$^+$18]. This means that most tools only present results with a very limited amount of models, often only having a single model with no possibility of editing it.

Visual Interactive Environment Some tools such as the one presented in [SCS+17] give the user the ability to make specific and often complex changes to a model in a simple and interactive manner. This feature greatly reduces the requirements to use this type of software by not obliging users to have knowledge such as writing code for a specific library.

Support of different DL frameworks One of the tools that had a very interesting structure was the one described in [WSW+18], but the main issues with it are that besides not generating code or compiling a model, it supports exclusively TensorFlow. This aspect of supporting a single framework is verified throughout all of the analyzed tools, which is not ideal since the implementation of the same algorithms differs in most libraries, which lead to different results.

Generation of code Most of the referred tools use models for a specific objective which usually involve analyzing their training statistics and their output. This means that transforming a high-level visual architecture of a model into actual code that can be executed is most times not an objective of the tools. This is deemed as a very important feature to develop since it completely frees a user of the tool they are using giving them the possibility of extracting the code and manually modifying it without relying on third-party software. The possibility of having such a feature is considered to be interesting to have on projects such as [RA16].

Liveness A few presented tools give the possibility of changing parameters that influence their output on the fly, meaning that the time needed to test multiple instances of a problem, theoretically is greatly reduced. In terms of constructing a DL architecture, it is assumed that the ability to have a mode in which the model is incrementally compiled and validated has the potential to improve a user's performance.

3.2 Hypothesis

This dissertation proposes that the process of creating a DL architecture can be improved with the aid of a visual tool that gives live feedback to the user. In Chapter 2 (p. 5) it is presented a collection of different projects that provide visual and live aids that can make a difference in the process of working with DL projects, with an emphasis on the interpretation of the results. With this being said, the creation of DL architectures might be a process that can take advantage of this type of support. Moreover, this dissertation can support other prominent topics such as the importance of creating a tool that can integrate multiple different aspects of DL projects such as the ones solved individually by the projects referred in Chapter 4 (p. 23). To be more specific, this dissertation attempts to verify the following hypothesis:

> *When people are provided with a live, visual tool that enables them to create and edit a deep learning architecture, they will be able to produce a given model's architecture faster (i) and with a better understanding of their solution (ii), when compared to traditional alternatives.*

It is worth noting that the concepts of building a model's architecture faster (i) and have a better grasp on solutions that are reached (ii) can be subjective interpretability. In order to be more objective and precise on the hypothesis, both referred expressions and other meaningful topics are discussed below according to the interpretation of the authors:

(i) - *To reach an architecture faster.* This expression refers to the capability of a person to follow different types of instructions in order to produce a DL architecture. Interpretations such as improving a model's performance based on their training and outputs are not considered for this dissertation.

(ii) - *To have a better understanding of the problem.* This part of the hypothesis refers to the process of understanding how some aspects of a DL model change when editing it and what are the main things that affect its validity.

What defines *people that have little experience in deep learning*? This population is considered to be composed of people that have decent experience in developing software but either is new to deep learning or have had experience in low complexity projects of that area. Examples of these populations can be assumed to be undergraduate students or junior developers.

What is *a live, visual tool* that aids the creation and edition of DL architectures? The closest example encountered to this idea is a tool with a visual similar to the one presented in [WSW+18]. The tool should also allow the creation of a DL model from scratch, the edition of every possible parameter of any layer that it contains, the support of different DL libraries and at the same time give the user the ability to be independent of the software at any point.

At the time this dissertation is being written, it can be said that there are ways to visually represent a DL architecture and ways to create them, but software capable of integrating these two where the visual representation directly controls the end architecture is still in an embryonic state. Having said this, the main focus of this project is the creation and study of the impact of software capable of decoupling the visual metaphor from the end result, through an intermediate representation that can be interchanged between the two phases. Such decoupling process naturally leads to extensibility since neither the visual representation nor the architecture generation is dependant on one another, possibly creating situations such as the usage of the same visual representation to generate DL architectures using different DL frameworks or even different programming languages.

For a better understanding, the proposed hypothesis can be translated into the following research questions:

1. **How does the usage of interactive visual software affect one's ability to converge to an acceptable DL architecture, when instructions are provided? What if instructions are not provided?** Can this type of software help to recreate DL models? What features have the most impact?

2. **What are the consequences of varying the complexity of DL architectures for users of interactive, visual software?** Does software like the referred one have a higher impact on larger models? How does it affect the capability of users to change a given model's hyperparameters?

3. **To what extent does the expertise of a person in DL have an effect on their capability of creating DL architectures, when using interactive visual software?** Are people with more experience in this area not affected as much by these types of tools as people without any experience?

3.3 Validation Process

One of the main objectives of this dissertation is to study how people react while using a tool like the one that is being proposed. To accomplish this, the authors believe that the best course of action is to rely on an empirical study by performing controlled quasi-experiments where the use of such software is compared to a more traditional way of creating DL architectures.

In order to perform such a study, it is important to create an experimental design with a process that tests the various aspects of the creation of a DL architecture. At the same time, each aspect that is being tested should provide simple and independent results so that some validation threats can be mitigated. Having this in mind, the entire study design will be created from scratch, with tasks attempting to assess individual aspects of DL architectures, as well as creating the possibility of studying the different responses of the participants to different challenges.

3.4 Summary

This chapter makes reference to some of the existing solutions to different problems and indicates what could potentially be improved. With an analysis of the current issues a hypothesis is formulated, that suggests that a live, visual and interactive type of software can improve the performance of the creation of DL architectures in some situations. This is broken down into 3 main topics which are presented under the form of research questions: the ability to create DL architectures, the impact of changing the complexity of a DL model and the impact of software of this type based on the proficiency of a person in this type of field.

On the final section, an empirical study based on quasi-experiments is proposed to be made regarding the validation of the research questions. The reason behind such a proposition is due to how abstract and complex the process of assessing people's behaviors is, so evidence-based on real-life interactions is assumed to be the best validation method of this project.

Chapter 4

Proposed Solution

Contents

This chapter formally explains both the high-level overview and implementation details concerning the development of the proposed tool. A description of the final product is presented as well as an analysis of the different options taken into account during the tool's creation phase.

4.1 High-level Overview

As stated in the previous chapter, the main objective of this thesis is to study how a live, visual and interactive tool affects the ability of a person to manipulate a DL model, when compared to a regular coding strategy. Having said this, the tool requires the creation of an environment where the decoupling between a visual metaphor of a DL model and the actual model is possible. This means that even though the model depends on its visual representation, it can be separated from it and treated as a module. This conceptualization of the tool naturally leads to a highly scalable and extensible tool which gives the option to support features such as the creation of models using different DL libraries or even different programming languages with the same visual representation.

In order to guide the creation of the tool, some basic functional requirements were set and deemed as the minimum requirements that the final product should incorporate in order to be a viable validation tool for the study:

- No user should be *stuck* with the tool. This means that if at any point, a user would prefer to continue his work without the usage of the tool, he should be able to retrieve his current results.
- The tool should provide a way to directly interact with the visual representation of a DL model.
- The tool should allow a user to customize a DL model with hyperparameters associated with different layers and should generate code according to the model's structure and parameters.
- The tool should verify the integrity of the model created, assuring the user that the model is, in fact, valid.
- While the visual representation of a DL model is being manipulated, the option to lively view the output being updated together with its validity should exist.

4.2 Implementation Details

4.2.1 Client Side

The client in Figure 4.1 is a web-based application that uses JavaScript to receive and interpret the user's input.

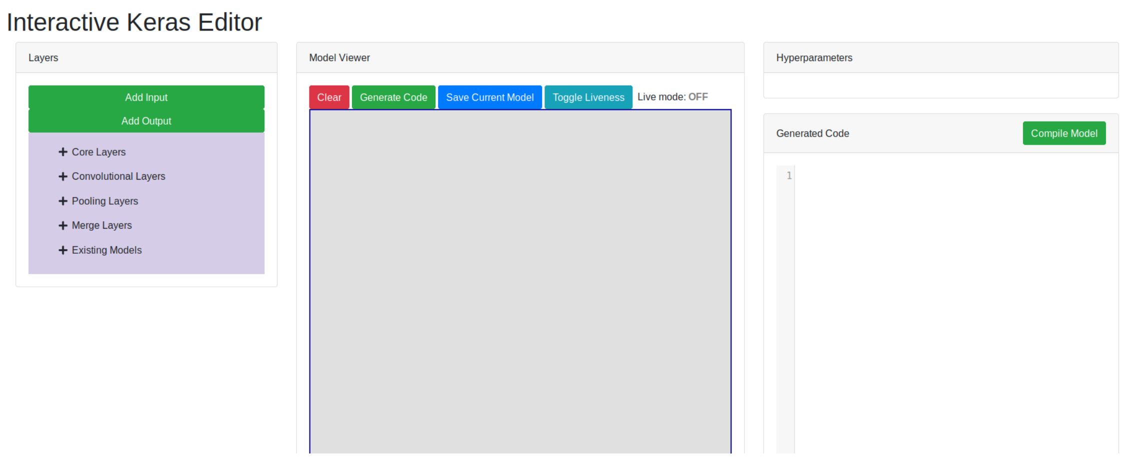


Figure 4.1: The Interactive Deep Learning Editor, currently supporting Keras.

As can be analyzed on the image, the client is composed of 4 sections:

- The leftmost container called "Layers" has buttons which, once they are clicked, add a layer to the model presented in the middle container. Every menu such as the "Core Layers" menu refers to the different sections presented on the Keras official documentation
- The middle container called "Model Viewer" holds the visualization of the current model and allows a user to directly interact with it. Once some layers are added to a model, they

can be connected to each-other via dragging the lower section of a layer to a top section of another layer, as can be seen in the image below. Any layer can be dragged anywhere within this container's bounds making it possible to rearrange a model on the fly.

- The upper rightmost container called "Hyperparameters" shows the parameters of a layer once said layer is selected by the user. These parameters can be inspected for a more detailed understanding of what they are and can also be modified.
- Once a user requests the code of a valid model to be generated, that code appears on the rightmost bottom container called "Generated Code". This section expects python code and is also editable, which can be useful for any desired custom alteration.

In order to provide a relevant output when generating code and verifying a model's integrity, it was assumed that it would be best to produce code using an existing DL open-source library that has already been tested and has had its validity accepted by a relevant enough community. Considering the project of this dissertation, only the Keras library was supported due to its simplicity and ease of use, although the entire process of generating and validating a model is not Keras-dependant and can be adapted for any other library.

4.2.2 Server Side

The server has as its main objective to receive a DL model architecture in JSON format, validate it and generate the associated code. For compatibility reasons, it started as a JavaScript application but was quickly changed to a python script so that it can integrate a lot of ML and DL libraries such as Keras and Theano. This decision was made not only because it assured that the generated code was, in fact, valid, by actually running it, but also future-proofed the project for any possible additions such as training a DL model or comparing it to existing ones.

This module of the software can also be hosted allowing multiple clients to connect to it and actually sharing models between them by using the save and load feature which is explained in § 5.3.

4.3 Model Creation

Once the client and the server are both running, an architecture of a DL model can be created. To start this process, a user can add some layers to the editor and connect them. When a user is satisfied with the architecture they can then change the parameters of each of the layer's parameters and finally attempt to generate the code. If a model is valid, besides the associated code being shown, a notification indicating the success of the process appears. A valid model follows the rules shown below:

- A model has at least one input and one output but can have multiple of either one.

- Every layer can have its output connected to multiple layers but can only receive one input, except the merge layers, which, by definition, handle multiple inputs.
- Every layer must have all of its mandatory parameters set.
- Layers that are connected to each other must have compatible shapes.

If any part of the process of generating the code encounters a problem the user will be notified. The process can fail either due to a connection issue, such as the server being offline or the integrity of the model being invalid. When such a case presents itself, the tool attempts to mimic the behavior of a debugger, informing the user what error occurred and its location. In the case which a layer receives too many inputs or doesn't have all of its mandatory parameters set, a notification appears describing that error, referring the name of said layer. If a connection between 2 layers is invalid, for example, because of their output/input shapes, an error message will appear inside that layer block. This feature is further explained in § 4.3.2.

Once a valid model has its code generated, the user has a codebase to work with, which enables him to directly modify it. The code can then be compiled using the "Compile Model" button which will run the code on the server. A detailed analysis of the model will then be shown, and on this menu which can be observed in Figure 4.2, the user can download 4 different files: A validated visual representation of the model, a YAML or JSON file containing every detail of the model and the model's associated hdf5 file which can be imported directly on projects using different libraries (such as TensorFlow).

4.3.1 Client Structure

Since the entirety of the visual editor is an abstraction of DL models and this topic has an extended amount of variable parameters, there exists not only a need for the application to support concepts of different DL libraries as well as a generic way to interact or modify such concepts.

Addressing the first issue, the ubiquitous concepts of a DL model concerning the client are the layers and their hyperparameters. Both of these concepts greatly differ when using different DL libraries, therefore everything related to them must be dynamic. Layers are, essentially, composed by a name and parameters, which can be mandatory or optional. Parameters are similar to layers in this context, but also have a type associated with them, such as a number or tuple. To deal with such problem the client has 2 objects: `layersObj` and `hyperparamsObj`.

The `layersObj` as shown in Listing 18 contains all information of each layer separated by module, which includes the layer's name and a reference to each of their hyperparameters separated by whether they are optional or not; each optional hyperparameter can have a default value set. Upon initialization of the software, this object is parsed and directly changes the visual structure of the client. This object also controls every possible layer that can be added to a model and have code generated with it, as can be seen in listing Listing 7 and figure Figure 4.3.

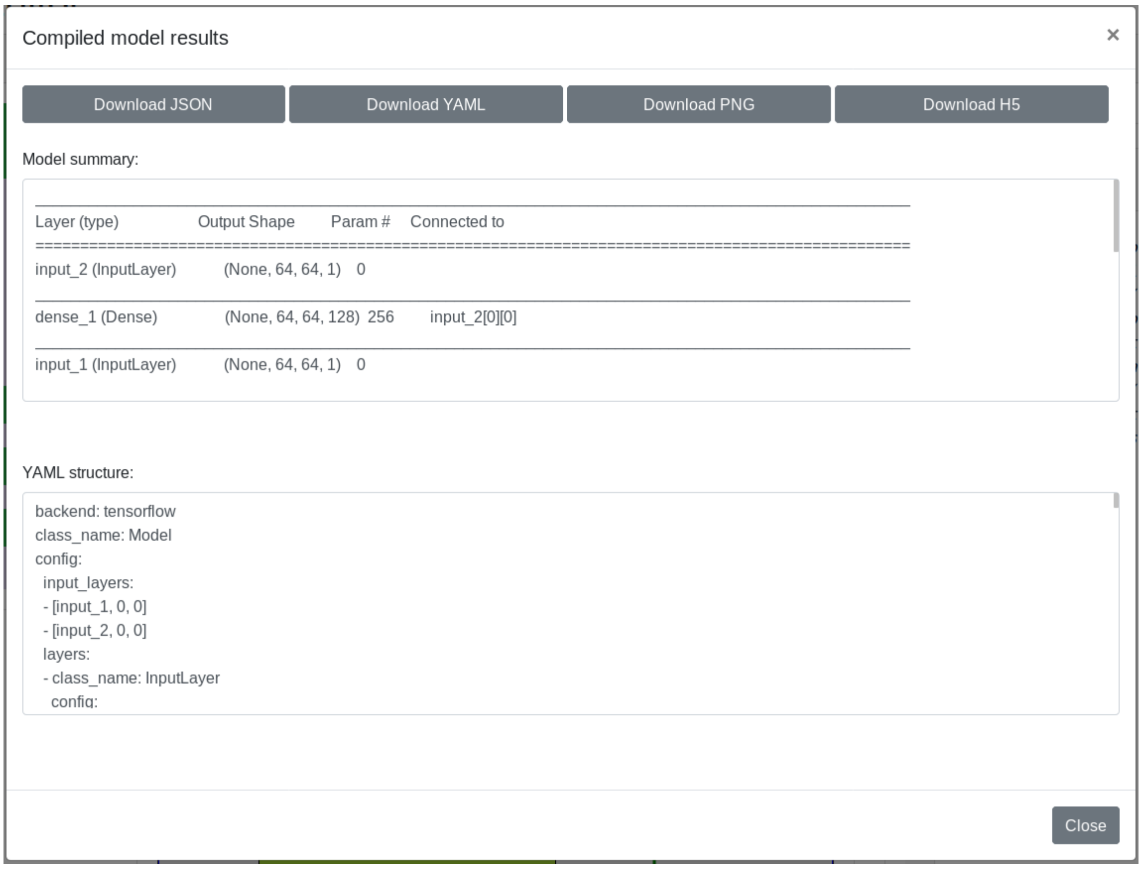
The `hyperparamsObj` contains all the hyperparameters, which can be referenced by any layer, along with one of the following types and specific parameters:

```
let layersObj = {
    "Core" : {
        "Dense" : {
            "required":[
                "units"
            ],
            "optional" : {
                "activation" : {
                    "default" : "linear"
                },
                "use_bias" : {
                    "default" : true
                }
            }
        }
    }
};
```

Listing 4.1: Extract of the `layersObj`, which shows an abbreviation of the *Dense* layer inside the *Core* module, having a reference to the hyperparameters *units*, *activation* and *use_bias*. As can be seen, *units* is the only required parameter of this layer and the rest are optional, having a default value set.

```
layersObj["Core"] = {
    "CustomDense" : {
        "required" : [],
        "optional" : {}
    }
}
```

Listing 4.2: An example that adds a layer called *CustomDense* with no parameters to the *Core* module, giving a user the possibility of using it in a model as shown in the following figure.



Model summary:

```
Layer (type)              Output Shape        Param #     Connected to
==================================================================================================
input_2 (InputLayer)      (None, 64, 64, 1)    0

dense_1 (Dense)           (None, 64, 64, 128)  256        input_2[0][0]

input_1 (InputLayer)      (None, 64, 64, 1)    0
```

YAML structure:

```
backend: tensorflow
class_name: Model
config:
  input_layers:
  - [input_1, 0, 0]
  - [input_2, 0, 0]
  layers:
  - class_name: InputLayer
    config:
```

Figure 4.2: Menu that appears after successfully compiling a model.

number , used for any numeric parameter and can be seen in Listing 2. It requires a specific parameter called `step`, which regulates which numbers are accepted.

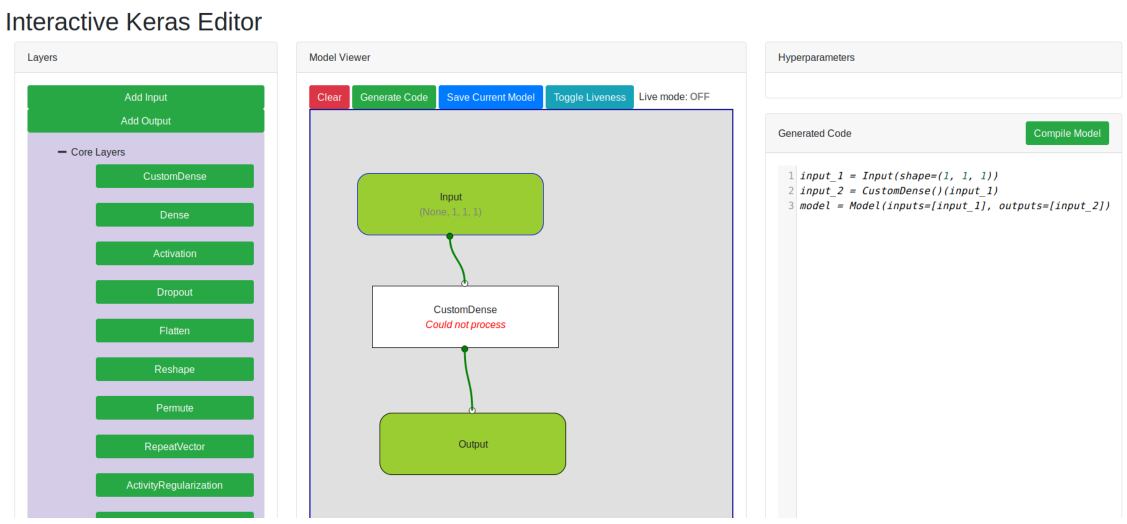
```
hyperparamsObj["units"] = {"type":"number", "step":1};
```

Listing 4.3: An example of a number hyperparameter called `units` defined in `hyperparamsObj`. If its default value, defined by the layer referencing it, is 0, then the input will only accept numbers with no decimal places.

tuple , used for any parameter defined by a numeric list and can be seen in Listing 2. It requires parameters similar to the `step` parameter presented above. The number of `step` parameters that are set influence the size of the tuple.

```
hyperparamsObj["shape"] = {"type":"tuple", "step_1":1, "step_2":1, "step_3"
    :1};
```

Listing 4.4: An example of a tuple hyperparameter called `shape` defined in `hyperparamsObj`. Since 3 `step` parameters are set, the tuple is a numeric array with a fixed length of 3.



```
input_1 = Input(shape=(1, 1, 1))
input_2 = CustomDense()(input_1)
model = Model(inputs=[input_1], outputs=[input_2])
```

Figure 4.3: The result of adding the code example shown in the previous listing. Note that *CustomDense* can't be processed since the server using Keras does not have any layer with that name *CustomDense*.

boolean and string , used for any parameter defined by either true/false or text values respectively and can be seen in listing Listing 3

```
hyperparamsObj["use_bias"] = {"type":"boolean", "value":true};
hyperparamsObj["name"] = {"type":"string"};
```

Listing 4.5: An example of a boolean and a string hyperparameter called `use_bias` and `name` respectively, defined in `hyperparamsObj`.

dropdown , used for any parameter defined by a value from a predefined set of possibilities, and can be seen in listing Listing 2

```
hyperparamsObj["padding"] = {"type":"dropdown", "values": ["valid", "same"]};
```

Listing 4.6: An example of a dropdown hyperparameter called `padding` defined in `hyperparamsObj`.

dropdown-dynamic , a special type used for any parameter that is defined by a function from a predefined set of possibilities, and the possible arguments that each function takes. Each one of these arguments has another type of parameter from the ones presented in this description and can have default values. This is described in the following listing.

The problem of interacting or modifying the mentioned concepts is solved by dynamically generating the inputs associated with the parameters referenced by layers. Once a layer is selected, each of their parameters is interpreted and based on their type, an input is generated. Every change

```
hyperparamsObj["activation"] = {"type": "dropdown-dynamic",
    "values": {
        "linear": {},
        "relu": {
            "alpha": {
                "type": "number",
                "step": 0.1,
                "default": 0.0
            },
            "max_value": {
                "type": "number",
                "step": 0.1,
                "default": "None"
            },
            "threshold": {
                "type": "number",
                "step": 0.1,
                "default": 0.0
            }
        }
    }
};
```

Listing 4.7: An abridged example of a dropdown-dynamic hyperparameter called `activation` defined in `hyperparamsObj`.

a generated input receives it records its current value on the model's object which can then be transmitted to the server for interpretation.

4.3.2 Model Interpretation and Code Generation

Once a model's architecture is finished a user can click the *Generate Code* button in order to build a JavaScript object containing all of the information of the currently built model and send it to the server for interpretation and feedback. The sent object is built by gathering the information of each layer of the model which is updated with each change to the model or a layer's parameters. As a starting point, this process was done by identifying the model's input and sequentially iterating through the connected layers, therefore, getting the correct order of a sequential model. An example of a DL model with a dense layer followed by a dropout and a flatten layer respectively would generate the following code:

```
model = Sequential()
model.add(Dense(128, input_dim=(64,64,1)))
model.add(Dropout(0.5))
model.add(Flatten())
```

Listing 4.8: Example of code generated by the editor using the sequential abstraction of Keras. This type of code is not scalable in terms of supporting higher degrees of complexity such as multiple inputs and outputs and will always take a single input and generate one output.

Upon further analysis, it was understood that this solution did not scale to more complex models nor did it manage to easily deal with issues such as non-connected inputs. In order to deal with models with multiple inputs and outputs, it was required to change this approach.

Since a complete DL model can always be assumed to have the form of a directed graph in which the layers are the vertexes and the connection between them are the edges, the ordering process can be performed using an adaptation of the topological sorting algorithm which, by definition "is a linear ordering of all its vertices such that if G contains an edge (u, v), then u appears before v in the ordering" [CLRS09].

This also leads to a difference in the code generation which in order to support multiple inputs and outputs the keras functional API is used, leading to the above code now being generated like the following listing:

```
1 input_1 = Input(shape=(64,64,1))
2 input_2 = Dense(units=128)(input_1)
3 input_3 = Dropout(rate=0.5)(input_2)
4 input_4 = Flatten()(input_3)
5 model = Model(inputs=[input_1], outputs=[input_4])
```

Listing 4.9: Example of code generated by the editor using the keras functional api, capable of supporting multiple inputs and outputs (by defining them in line 5).

When the server receives a request to generate the code it expects to receive an ordered model under the form of a JavaScript object as well. This object essentially contains an ordered identification of every layer that the model has, together with their associated parameters and their values. Each layer also has a unique identifier and a reference to the layers that are their direct inputs. With this structure, it is easy to generate the code since all information is well grouped and if the target DL library has a generic way of constructing a model's architecture then it becomes even easier.

For the Keras library, the functional API is used since all models can be structured the same way and all information for each layer is present on mentioned object: the layers, their parameters' values, and their inputs. The generation of the code can be broken down into 3 different steps:

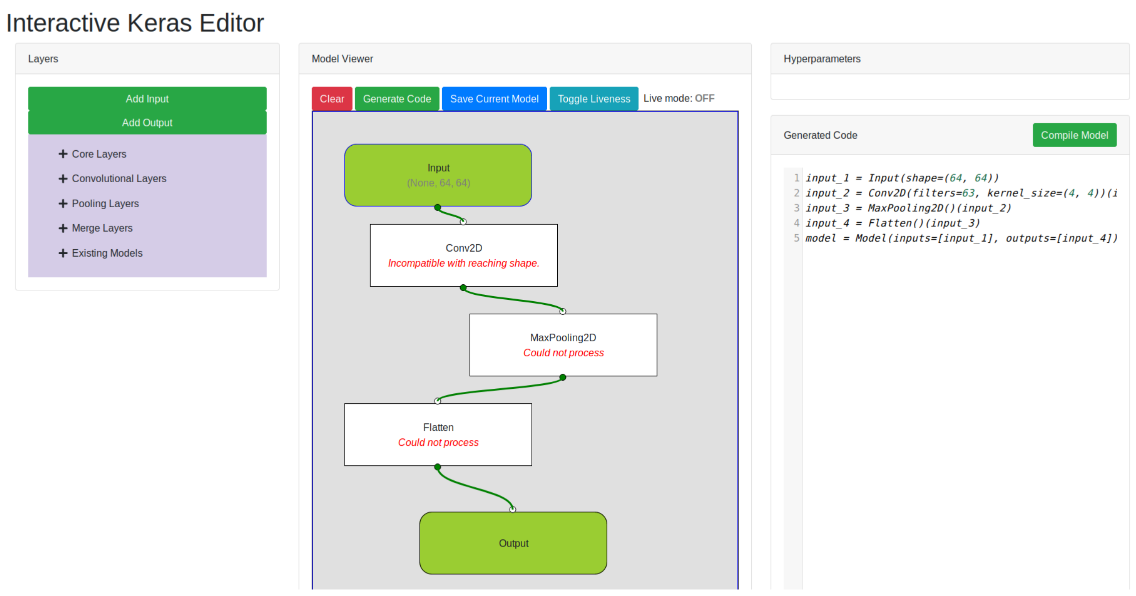
1. **Mapping the layers.** By receiving automatically generated unique identifiers for each layer the first step is to iterate through every layer and, besides replacing the identifier for an actual name to be used such as `input_1`, it is required to map the identifiers to said names. The mapped names of the layers are used every time a reference to a layer is required, such as in the inputs and outputs list on the model definition.

2. **Parsing the hyperparameters.** The choice of generating code that uses the python language really shines on this step because, when adding parameters to a layer, the order of said parameters is not needed to be taken into account by simply stating the name of the argument that is being defined, such as `units=128`. With this taken into account,

when generating the code of a layer all arguments are passed as a string with a form of `param_name=param_value,` and once all parameters are added, the last comma will be removed.

3. **Validating the model.** Once the code is generated it is required to validate it so that every output presented in the client is assured to be valid. This is done by individually executing every line of code with the `exec` function, and catching possible exceptions which will be transmitted to the client. This process not only assures the quality of the code that the client displays but also makes it possible to interpret the exceptions, indicating, for example, if a determined layer was incapable of being processed due to the compatibility between the input and output shapes.

As seen in figure Figure 4.4, it is possible to take note of the errors presented directly associated with each layer. This model has an input of 64x64 and the first layer to which this input is transmitted is a 2d convolutional layer, which usually expects to receive inputs with shapes of 3 dimensions. Naturally, the validating process fails at that point, and the rest of the layers that depend on that convolutional layer can not be processed. All of this information is presented with a specific description which allows a user to conclude exactly where the error is located in, regardless of their knowledge of DL, since it is clearly visible that the input layer was processed but the convolutional layer was not.



```
input_1 = Input(shape=(64, 64))
input_2 = Conv2D(filters=63, kernel_size=(4, 4))(i
input_3 = MaxPooling2D()(input_2)
input_4 = Flatten()(input_3)
model = Model(inputs=[input_1], outputs=[input_4])
```

Figure 4.4: Example of errors shown by the editor when the model is invalid.

4.4 Quality of Life Features

With the creation of this tool, all sorts of additional features can be thought of in order to improve the software's utility. Besides simple aspects such as alerts, four features were implemented and deemed to be of great interest both for the validation process and for regular use.

Live mode is one of the focus of this thesis and of great importance to this project as well as an appropriate addition to the creation of DL models. Instead of making the user create the entire model and then correct the problems, such as incompatibility of shapes, this feature aims to change this traditional way of programming. This is done by incrementally generating and executing the code associated to the model at hand, with the assumption that the last layer will be the output, and that allows the user to receive validation on the fly of their model. This mode is activated by clicking the *Toggle Liveness* button.

Save/Load is an extremely useful feature that allows a user to save and load their models. This is done by transmitting to the server the current model under the form of an ordered JavaScript object and saving said object. Every time a model is saved or the page is reloaded the *Existing Models* section on the *Layers* container is updated, making it possible opening it and selecting the newly created entry to load the model

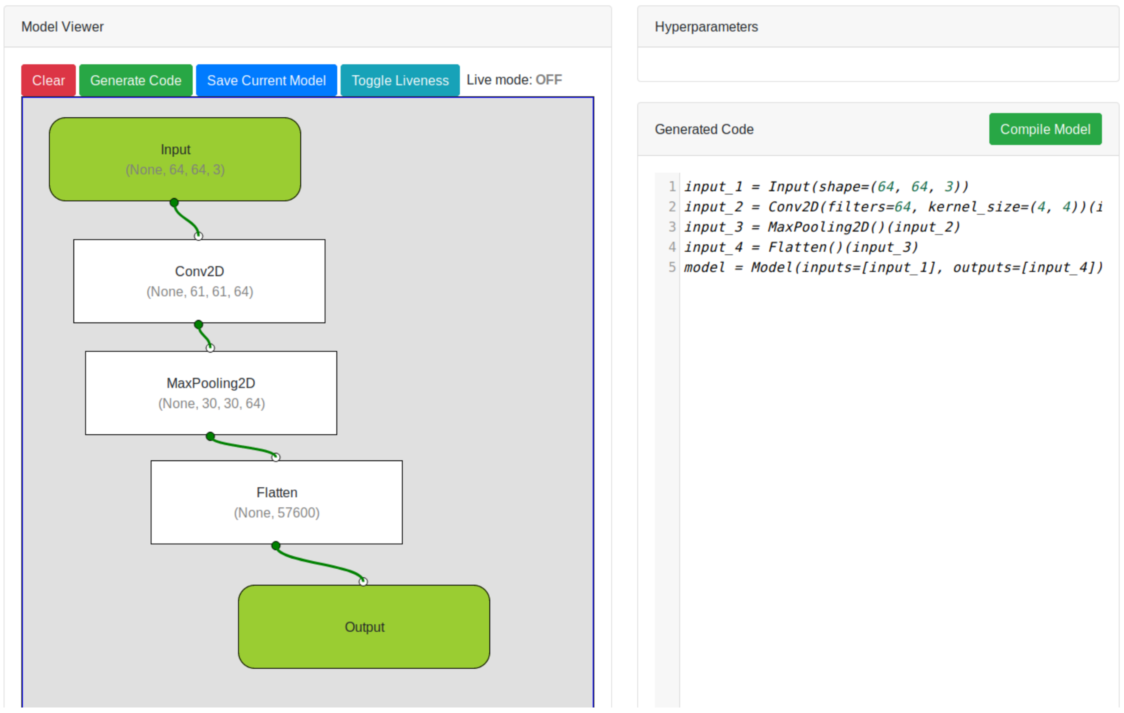
Creation and usage of patterns is a feature that derives from saving and loading. When creating a deep learning model it is common to have patterns and repeat them, such as a common pattern of having a convolutional layer followed by a pooling layer. With this in mind, once a model is loaded a single layer appears in the viewer, being an abstraction of a saved pattern; if the previously saved model is valid, then the loaded pattern can be used as many times as wished, as seen in figures Figure 4.5 and Figure 4.6.

Overwriting saved models is the complimentary feature to the previously mentioned one. Since invalid models can be saved, the ability to overwrite them is a must. By double-clicking a loaded model, the viewer will be updated with the previously saved model, and the user can inspect it, change its layers or parameters or even use individual pieces of it for a different model. The user can then save it again and the model will be updated.

4.5 Summary

By reading this chapter it is possible to understand the architecture of the developed tool as well as what its features try to accomplish. The tool is divided into a web-based application which a user is expected to interact with, and a python script that runs Keras (or other DL library if such is desired) which generates and validates code regarding the JavaScript objects that it receives. The entire project is aimed towards both usability and scalability since both the client and server-side are modular and can be replaced at any time, meaning that, for example, if a developer wishes to either create a mobile application to replace the client-side or a program in C with DL libraries other than Keras, they are able to.

The entire project was also created by taking into account the concept of liveness and how it can affect the users. This is notable during the creation and editing of a DL model whereas the

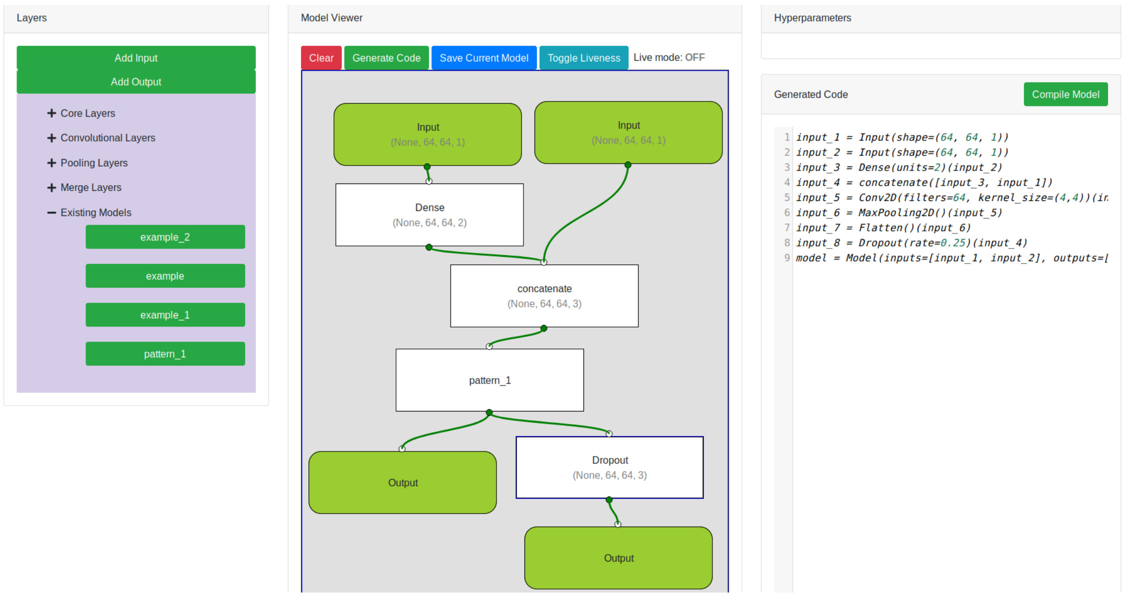


```
input_1 = Input(shape=(64, 64, 3))
input_2 = Conv2D(filters=64, kernel_size=(4, 4))(i
input_3 = MaxPooling2D()(input_2)
input_4 = Flatten()(input_3)
model = Model(inputs=[input_1], outputs=[input_4])
```

Figure 4.5: An example of a valid model, ready to be saved a re-used in the future.

user can change hyperparameters and immediately see the results on the model. Such feature is considered to be especially useful when changing the shapes of the layers of a DL model since errors, such as incompatibility between two layers, are presented instantly and associated with the problematic layer.

Lastly, it is important to refer that another concern of the project was to provide a way for users to not be dependent of the tool, meaning that at any point the user should be able to fully customize his results, either by editing them on the tool or export them to use in another way. All of these features are provided by the tool which allows a user to compile a DL model and export its results or even directly editing the generated code.



```
1 input_1 = Input(shape=(64, 64, 1))
2 input_2 = Input(shape=(64, 64, 1))
3 input_3 = Dense(units=2)(input_2)
4 input_4 = concatenate([input_3, input_1])
5 input_5 = Conv2D(filters=64, kernel_size=(4,4))(in
6 input_6 = MaxPooling2D()(input_5)
7 input_7 = Flatten()(input_6)
8 input_8 = Dropout(rate=0.25)(input_4)
9 model = Model(inputs=[input_1, input_2], outputs=[
```

Figure 4.6: An example of a valid model, using the pattern created previously as shown in the previous figure, which was saved with the name `pattern_1`. As can be seen on the generated code, lines 5 through 7 refer to the pattern in the aforementioned figure.

Chapter 5

Evaluation

Contents

This chapter details every aspect of the quasi-experiment that was made with the intent of getting a better understanding of the effect that the developed visual editor has on the process of building DL architectures. This was accomplished by performing a study on groups of computer science students, which consisted of challenging said people to tackle different aspects of DL models using Keras and the developed editor.

5.1 Research Design

With the introduction of a tool like the one developed, the way of constructing DL models changes drastically from writing code to visually editing them. With this being said, the usage of the visual editor needs to be subjugated to an evaluation in order to comprehend if it contains the potential to improve how a person can create DL architectures. In order to do this, a quasi-experiment was performed since it is believed to be the best overall method of validating the proposed hypothesis, especially because of the complexity that evaluating behaviors of human beings poses. The experiment was performed with the help of 16 students from the Integrated Masters in Informatics and Computing Engineering course of the Faculty of Engineering of the University of Porto.

As a starting point, all participants were issued to sign a consent Appendix A (p. 75), thus providing the necessary authorization for the collection and analysis of their data. The experiment was composed by 16 individual sessions that had a duration of around one hour and each one included two sets of 4 tasks and questionnaires Appendix B (p. 77) that had to be filled after every step, as can be seen in figure Figure 5.1. The four tasks a student had to complete are virtually the same in order to reduce any validation threats such as difficulty or interpretation issues albeit some instructions are adapted to the environment the person is using. Each student was given one of two versions so that once the experiment was over, 8 people would have used version A, and the remaining 8, version B. The difference between the versions was simply the order of the group of tasks, since following the same order of tasks could threaten the validity of the results (and such phenomenon is verified in section § 5.3).

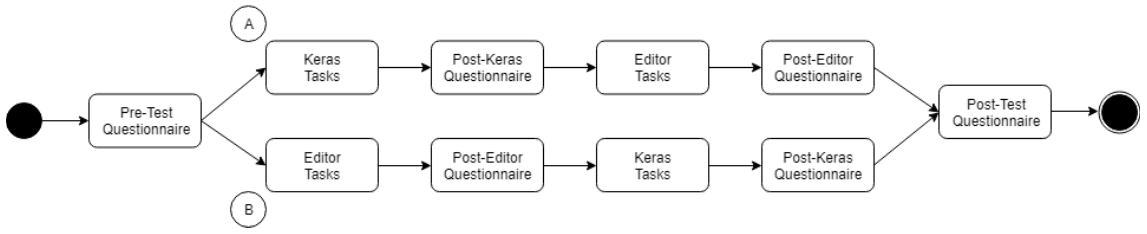


Figure 5.1: Diagram explaining the process of the quasi-experiment. A and B are referent to version A and version B.

During this study, the baseline for comparison was the tasks completed with the Keras framework which provided students with a traditional way of interacting with a DL architecture: through regular coding. During the execution of these tasks, regular developer tools were provided which included:

Code Editor. Students were given the option of using one of three generic code editors: *Visual Studio Code*[1], *Atom*[2] or *Notepad++*[3]. This ensured that they were in a familiar and comfortable developing environment and could use their preferred features such as shortcuts, code selection, integrated terminals, etc.

Python Linter. Since all code that needed to be implemented needed to be in *Python* the linter *tslint*[4] was provided with the sole purpose of providing semantic error detection and auto-completion, especially for the Keras framework.

Access to the Internet. During these tasks students were able to search the internet for any answers to problems that they could encounter as well as each task giving a direct link to the official Keras documentation.

The experimental tool was used in the editor tasks which had the same objectives. In these tasks, the students were given access to the tool and a short guide on how to use its basic features.

[1]https://code.visualstudio.com/ (Retrieved: 02/07/2019)
[2]https://atom.io/ (Retrieved: 02/07/2019)
[3]https://notepad-plus-plus.org/ (Retrieved: 02/07/2019)
[4]https://palantir.github.io/tslint/ (Retrieved: 02/07/2019)

The end result of each task was simple and the same throughout each one: generate an image of a DL model's architecture. To accomplish this in the tool, instructions were provided on how to access the postcode compilation menu and download the model's image. In the tasks that required the use of the Keras framework, template python files were provided which included all needed imports as well as the code line that generated the image of a model. This meant that what students were required to perform was just the construction of a model's architecture and nothing else.

5.1.1 Statistical Analysis Used

Regarding the statistical analysis that is made on the following sections, it is important to understand what kind of analysis can be done, which is the most pertinent concerning the results obtained and to what extent is this study valid and significant.

There exist three types of data that was collected: time that the participants required to complete every task, whether or not a participant managed to complete certain parts of a task and Likert-scale based answers [Lik32] to the questionnaires. In order to make a statistical analysis of this data, four different statistical tests were taken into account in order to check if there exists a significant difference between the comparisons made between two populations with different factors affecting them. These tests are the Independent-Samples T-tests [KHL11], Mann-Whitney U tests [MN10], Chi-Square tests [McH13], and Fisher's Exact tests [Bow03]. The first two were considered for analyzing the differences between the times needed to complete determined tasks and the third and fourth ones were considered for binary comparisons, being these the ones analyzing whether or not participants could complete a determined topic of a task. For the questionnaires, the only part which was considered to be relevant enough to make a statistical analysis was the first part with the objective of making a more in-depth demographic analysis. The other 3 parts of the questionnaire were assumed to be affected by too many threats and have a heavy bias so the only analysis that was made was a descriptive one.

According to [dWD10] and to the tests that were made, very minor differences were obtained when performing either the Independent-Samples T-tests and the Mann-Whitney U tests. The only assumption that was required to mathematically verify in order to make the correct choices between these two tests was the fifth assumption, stating that the data should be approximately normally distributed for each group considered [Sta], even though this was only to make sure the results of the Independent-Samples T-tests were more robust. This was tested with the Shapiro-Wilk test [SW65] and all results of each considered group from the first and second task did not reject the hypothesis of having a normal distribution, therefore the Independent-Samples T-tests were used. For task 3 and 4, the method of evaluation was related to the correctness of the answers rather than the time each participant took to complete it since almost all participants were unable to complete the entire task while using Keras.

Considering now the Chi-Square tests and Fisher's Exact tests, the main factor that leads to the choice of either one of these tests is the size of the populations which is advised to be higher than 25 when using the Chi-Square. Since this was never the case, Fisher's Exact test was assumed to be the best choice in order to evaluate if two populations have significant differences.

The general way of understanding the results of these tests is to assume a null hypothesis of having two populations that present results that have no significant differences. After reading the results of the applied test, namely the 2-tailed significance value, such result should be compared to 0.05; if the obtained result is higher than 0.05, then the null hypothesis can not be rejected, but if it is lower, than the null hypothesis is rejected, resulting in the conclusion that the two populations have a significant statistical difference. These tests do not necessarily mean that one population is better or faster than the other, it simply states that under the conditions that they are being compared, they have a significant difference; that conclusion can be reached after analyzing the means and the results. For example, if a population has an average time of 5 minutes to complete a task and another population has an average time of 10 minutes, if an Independent-Samples T-test concludes that the populations that are being compared have a significant statistical difference, then a conclusion such as the population that had an average time of 5 minutes is faster than the other one since their average time is lower.

For better interpretability, an example of how to interpret the results of the Independent-Samples T-tests is presented in § 5.3 is shown below. Considering the results presented in Table 5.1 and in Table 5.2 are referent to time group A and B required to complete a task, when analyzing Table 5.2, the first 3 columns of the results section (*Eq. Var.*, *F* and *Sig.*) are referent to the Levene's Test [Lev61]. This test has as its null hypothesis the fact that the compared groups have similar population variances; if the significance value of this test is below 0.05, this hypothesis is rejected and the row to consider is the second one, which assumes an unequal variance. As the significance value (a) is 0.883 which is above 0.05 the row to consider is the first one and the significance of the Independent-Samples T-test is (b), 0.002, which is below 0.05. This means that in this particular comparison there is a significant enough difference between the results of group A and B, meaning that for this scenario, since group A as a lower mean than group B, it can be concluded that group A was faster than group B.

Table 5.1: Example of descriptive results of a comparison called *Example*.

Comp.	**Results**			
	Group	Mean	Std. Deviation	Std. Error Mean
Example	A	1.63	1.061	0.375
	B	3.88	1.246	0.441

Table 5.2: Example of an Independent-Samples T-Test of a comparison called *Example*.

Comp.	**Results**					
	Eq. Var.	F	Sig.	T	DF	Sig. (2-tailed)
Example	True	0.022	0.883 (a)	3.888	14	0.002 (b)
	False			3.888	13.651	0.002

5.1.2 Demographic analysis

To ensure the entire validation process produces meaningful results, one of the first things that needs to be done is to guarantee that the group of participants is somewhat homogeneous both in terms of skill level and knowledge. This is needed in order to mitigate the discrepancies that might occur due to the different background of the participants, rather than the different software that they are using. With this being said, all participants were students from the same course meaning that their backgrounds had strong similarities between them.

5.1.2.1 Experienced participants in ML

A concern that was taken into account was the fact that even by selecting students with what was being assumed of having a similar range of skills in programming, a part of the participants had specific characteristics that could influence their results such as already having had contact with the python language before. Therefore, a pre-test questionnaire was given to the participants so that those that had some knowledge on relevant topics such as python and ML could be analyzed differently. Six questions were posed, for this reason, each with answers based on a Likert scale, which explored the abilities of a participant regarding using python, understanding basic ML and DL concepts and having experience with Keras. Once the experiment was done, it was possible to assume that 2 groups of 8 people each could be made:

Knowing python (P1) and Knowing Keras (P6) To assess this point, a single Likert scale based question of whether or not the participant was experienced in python or Keras was made, respectively.

Knowing basic concepts of ML and DL This point was explored in 4 different questions. The first two (P2) (P3) simply assessed with Likert scale based questions if the participants were familiar with basic concepts of ML and DL respectively; the third and fourth Likert scale based questions (P4 and P5) posed a false and true statement, requiring the participants to concur or not with them. The answers to the last two questions of this topic were deemed correct if the participant slightly agreed or completely agreed with the true statements and vice-versa for the false ones.

With this concern in mind, after the participants filled the pre-test questionnaire, they were given version A or B with an attempt of having the same amount of experienced people performing each version. After analyzing the results and dividing the participants into two groups, it was possible to validate and conclude that 8 of the participants had some experience in this field and 4 of them were given version A while the rest, version B.

For readability and validation sake the results of P4 and P5 were transformed into binary results of whether the answer was correct or not and aggregated into one single result (by performing the `AND` logic operation) corresponding of whether a participant answered correctly both questions. To validate this entire process, Independent-Samples T-Tests were made and the results of P1, P2, P3, and P6 and for the aggregation of the results of P4 and P5 a Fisher's Exact test was conducted. Note that for all of the tests the null hypothesis is the following:

> **H0** - *For the question at hand, the answers from both populations have no significant difference.*

Table 5.3: Statistical descriptive values of a part of the pre-test questionnaire. The raw data that led to these values was used for the Independent-Samples T-Tests in table Table 5.4 and the Fisher's Exact test in Table 5.5

Topics	Results			
	Group	Mean	Std. Deviation	Std. Error Mean
P1	1	1.63	1.061	0.375
	2	3.88	1.246	0.441
P2	1	1.88	1.126	0.398
	2	3.75	1.282	0.453
P3	1	1.75	0.886	0.313
	2	3.38	1.061	0.375
P4	1	2.88	0.744	0.263
	2	1.63	0.354	0.125
P5	1	3.75	0.756	0.267
	2	4.50	0.886	0.313
P6	1	1.13	0.354	0.125
	2	3.13	0.991	0.350

Table 5.4: Results of the Independent-Samples T-Tests for 4 of the Likert scale based questions of the pre-test questionnaire. By assuming a minimum significance of 95% (2 tailed significance <= 0.05) it can be concluded that the two populations that are being compared have a significant difference, therefore we can reject the null hypothesis.

Null Hypothesis	Population has no significant difference					
Topics	Results					
	Eq. Var.	F	Sig.	T	DF	Sig. (2-tailed)
P1 - Python Experience	True	0.022	0.883	3.888	14	0.002
	False			3.888	13.651	0.002
P2 - Knowledge of ML	True	0.000	1.000	3.108	14	0.008
	False			3.108	13.771	0.008
P3 - Knowledge of DL	True	0.014	0.908	3.325	14	0.005
	False			3.325	13.572	0.005
P6 - Keras Experience	True	2.733	0.121	5.376	14	0.000
	False			5.376	8.753	0.000

By analyzing the results of these tests, assuming in all cases a minimum significance of 95%, all cases reject H0, therefore we can assume that the two populations that are being compared are, in fact, different. Although, with this conclusion, it is not possible to assume that population from Group G2 will have different results from the population of Group G1, nor that they are more

Table 5.5: Results of the logic operation *AND* between the results of the Likert scale based question 4 (P4) and 5 (P5). A correct answer to P4 and P5 was considered to be slightly agree or completely agree and slightly disagree or completely disagree, respectively. The result of Fisher's Exact test based on these results gives a 2-tailed exact significance of 0.010, which we can use to invalidate the null hypothesis of these two population not having any significant difference.

Topic	Count			Total
	Group	0	1	
P4 AND P5	1	7	1	8
	2	1	7	8

experienced since these questions merely ask the participants about their own feelings towards the fields of ML and DL.

An interesting remark can be made by analyzing the means from the answers to P4 and P5 and the results of the Fisher's Exact test pertaining the correctness of the answers to these 2 questions: although most of the answers from group 1 were considered to be incorrect, the majority of them tended to the correct answer. As an example, in P5 the answers which were considered correct were either slightly agree and completely agree (4 and 5 in the Likert scale) and the mean of the answers for group 1 was 3.75; this suggests that population from group 1 had some correct answers but the majority were unsure and selected the value 3, which checks out if the raw data is analyzed. The same case occurs for P4 but with opposite values (correct answers were 1 and 2 in the Likert scale and the mean of group 1 was 2.88).

In order to summarize the conclusions of this subsection, the diagram in Figure 5.2 was created and throughout the rest of this chapter, all groups will be referred to by their associated labels.

5.1.2.2 Experienced participants in live and visual software

Another important factor taken into account of the participants of this experiment was that there could be a significant difference between the experience when dealing with live or visual software. While in the previous section the population was analyzed to see if one group could have more proficiency in the Keras experiments because of their backgrounds, this section details if there exists a group of participants that could be more proficient when dealing with a visual editor such as the one developed because of their background.

In the pre-test questionnaire, question 7 and 8 assessed the opinion participants had about their previous experience with visual, interactive tools, such as Scratch [MRR+10], and live developing software such as nodemon [web10], respectively. All attempts of Independent-Samples T-Tests on the population pertaining these two questions proved to be ineffective so, for the entirety of this experiment, all possible differences on these subjects were not taken into account.

5.1.3 Process Overview

During the experiment, several aspects of the construction of a DL model were being tested. These ranged from building a simple, linear model to the construction of an architecture of a model with

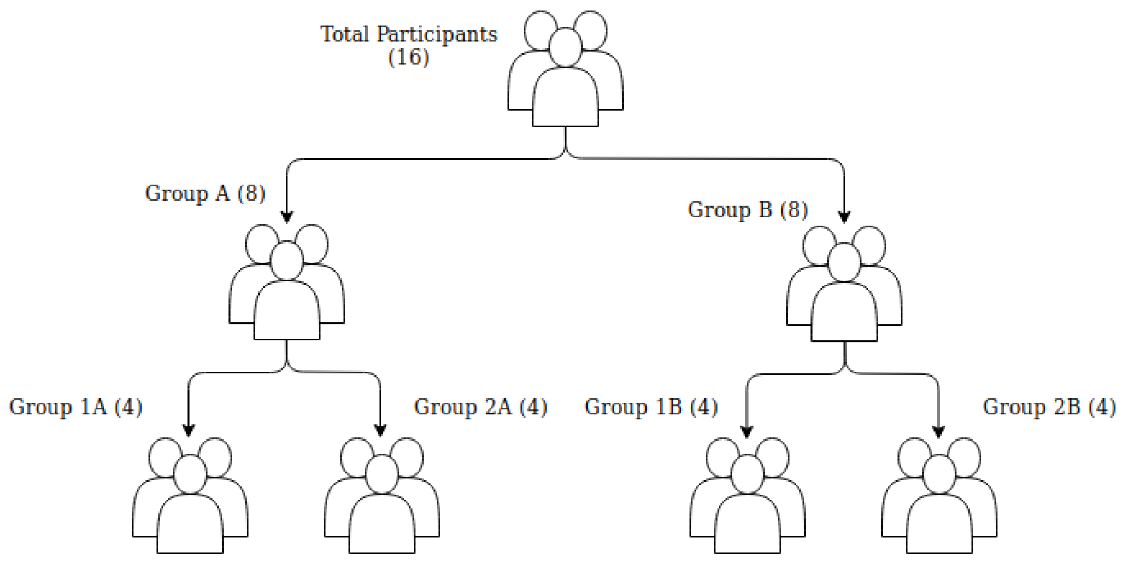


Figure 5.2: Diagram explaining the groups of participants along with their labels and their quantity. Groups GA and GB refer to the group of participants that were given versions A and B, respectively. Groups G1A and G2A are participants from Group GA which were deemed to have no experience and some experience in ML and DL, respectively. Groups G1B and G2B are participants from Group B which were deemed to have no experience and some experience in ML and DL, respectively.

several layers and multiple inputs and outputs, parameterization and even edition of models made earlier. To retrieve important data from the experiments and analyze it after the completion of all sessions, all students were required to save the results that they reached, either by saving the python file they edited or by using the save function of the tool. Their method of solving the tasks was also being individually checked so that other types of data could be retrieved such as their usage of the editor's live mode or how they tackled each task. To help this retrieval, questionnaires after each group of tasks were required to be filled with the intent of assessing how a person felt while using a determined environment.

To start a session, a student would be indicated to access a *readme* file which contained all information they required for the tasks such as the location of the necessary files, relevant links and a small background of what deep learning is and what the tasks test. From then on students accessed each task, each contained within a specific folder, and followed the provided instructions under that same folder, with absolutely no need for contact with the person supervising them.

Each task was expected to be completed in a short amount of time with the considered hardest one to reach up to 10 minutes, although 15 minutes were given for each task which at that point the student would be issued to save their progress and advance to the next task. At any point, the possibility of giving up on a task and going to the next one was also valid for any student. In order to analyze the time a student takes to successfully perform a task, dropouts are deemed to have the same value as incorrect or incomplete answers, although the motive of why the output of a task was not the correct one was also registered and analyzed.

5.2 Tasks Descriptions

This section details every task each student had to complete along with what they focused on testing, what solution was expected and how the process of solving it differed between the two given environments.

5.2.1 Task 1: A Starting Guide

The first task was designed to be very simple with the objective of getting the users comfortable with the developing environment. The objective of the exercise was to build a very simple architecture of a DL model based on a generic guide found on the web, with the intent of simulating the experience of a person's first contact with DL. The participant would access the guide follow its instructions, in Keras by either writing or copying and pasting the lines of code and in the editor by building the model which is shown in figure Figure 5.3.

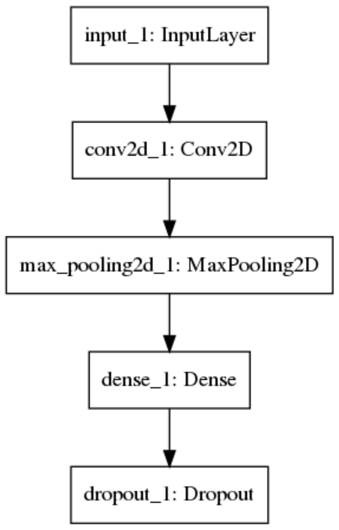


Figure 5.3: Expected output of the execution of Task 1.

5.2.2 Task 2: Simple DL Architectures

In this task, you will have to build a model based on your interpretation of the following text:

- *The model at hand should have an input with dimensions of [32x32x3] and a single output.*
- *The first layer should be a 2D convolutional layer with 32 output filters, a kernel size of [3x3] and a relu activation.*

- *At this point a pooling layer is needed with a pool size of [2x2] and then another layer should help to prevent overfitting by dropping out 25% of the data.*
- *Let's flatten the data and, for the final layer, use a dense layer with 128 units and a softmax activation.*

The second task was the first real challenge the participants had to face since no guide was provided and simple, text-based instructions were provided on how to build this task's model. The expected model to be generated is somewhat similar to the one in task 1 with some minor variations on the layers and their parameters and is presented in Figure Figure 5.4.

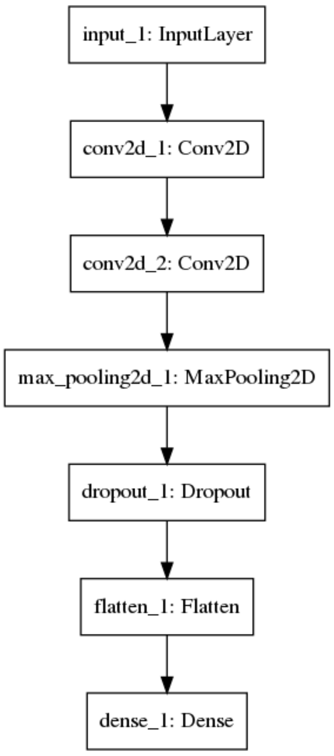


Figure 5.4: Expected output of the execution of Task 2.

5.2.3 Task 3: Complex DL Architectures

The third task was assumed to be the hardest one since the only provided information on the model was Figure 5.5. This model tested the ability of a participant to understand and adapt to the fact that the model is not sequential and to perceive the existence of input and output shapes between layers and realize they are transformed based on some parameters. This task was divided into 2 topics: the ability to create a model with multiple inputs and outputs and the ability to adapt a model's shapes based on the interpretation of an image.

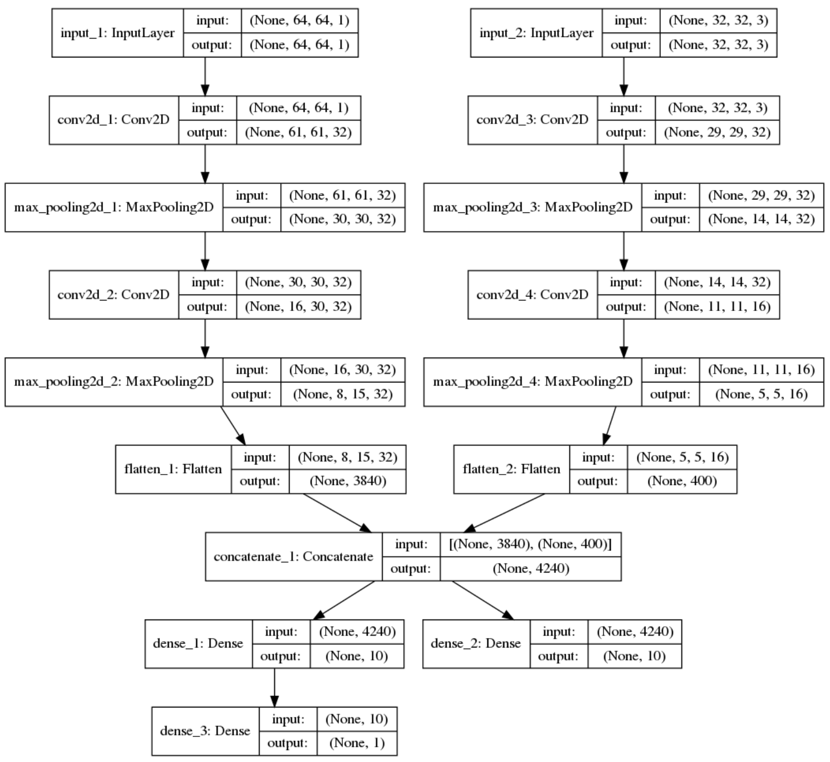

input_1: InputLayer
input: (None, 64, 64, 1)
output: (None, 64, 64, 1)
input_2: InputLayer
input: (None, 32, 32, 3)
output: (None, 32, 32, 3)
conv2d_1: Conv2D
input: (None, 64, 64, 1)
output: (None, 61, 61, 32)
conv2d_3: Conv2D
input: (None, 32, 32, 3)
output: (None, 29, 29, 32)
max_pooling2d_1: MaxPooling2D
input: (None, 61, 61, 32)
output: (None, 30, 30, 32)
max_pooling2d_3: MaxPooling2D
input: (None, 29, 29, 32)
output: (None, 14, 14, 32)
conv2d_2: Conv2D
input: (None, 30, 30, 32)
output: (None, 16, 30, 32)
conv2d_4: Conv2D
input: (None, 14, 14, 32)
output: (None, 11, 11, 16)
max_pooling2d_2: MaxPooling2D
input: (None, 16, 30, 32)
output: (None, 8, 15, 32)
max_pooling2d_4: MaxPooling2D
input: (None, 11, 11, 16)
output: (None, 5, 5, 16)
flatten_1: Flatten
input: (None, 8, 15, 32)
output: (None, 3840)
flatten_2: Flatten
input: (None, 5, 5, 16)
output: (None, 400)
concatenate_1: Concatenate
input: [(None, 3840), (None, 400)]
output: (None, 4240)
dense_1: Dense
input: (None, 4240)
output: (None, 10)
dense_2: Dense
input: (None, 4240)
output: (None, 10)
dense_3: Dense
input: (None, 10)
output: (None, 1)

Figure 5.5: Expected output of the execution of Task 3.

5.2.4 Task 4: Simple and Complex Hyper-Parameterization

The fourth and final task had the objective of testing the ability of a participant to edit an existing model's parameters. The editions were considered to be simple apart from the third point which requires a user to edit an activation of a layer to have a different parameter from its default. Although in the editor it is similar to editing another regular parameter because of the usage of the type `dropdown-dynamic`, in regular Keras code a more complex approach is required. The names of the layers that the task is referring to are the ones in figure Figure 5.5 and independently from the ability of the participant to finish task 3, an answer to the said task, both in Keras and in the editor, was provided and could be used at any point.

In this task you will have to overwrite the model from the third task.

The new model should have the following changes (note that the names of the layers are referent to the names shown in the auxiliary image, i.e. input_1 is the leftmost InputLayer):

- *conv2d_2 should use a relu activation, its padding mode should be "same" and its data format should be "channels_first".*
- *max_pooling_2d_2 should use "same" and "channels_first" as its padding and data format, respectively*

- *conv2d_1 should have a relu activation with a maximum value of 30*
- *dense_1 and dense_2 should both have relu activations*
- *dense_3 should have a sigmoid activation and should use bias, with an initial value of a Zeros matrix.*

5.3 Results

In this section, all the results obtained during the quasi-experiment are presented in seven subsections. The first subsection describes the overall concerns taken into account about the data and some general information about it. The next four describe the results of each task individually, the fifth makes a global analysis together with the post-tasks questionnaires and the final subsection describes some interesting remarks about the behavior and feedback of the participants.

5.3.1 General Overview

All the gathered data about the times taken by the participants, on both versions, and with both tools, in Tasks 1 and 2 has been tested with the Shapiro-Wilk test and were concluded to be normally distributed. Therefore, in these cases, the authors believed the Independent-Samples T-Tests to be the best choice of statistical analysis of the population's difference since all assumptions of the test are met. The null and alternative hypothesis for the tests are the following:

Time used in each task	
H0 - Both groups needed a similar amount of time to complete the task.	H1 - There exists a significant difference in the average time each group needed to complete the task.
Correctness of each challenge of each task	
H0 - Both groups correctly completed similar amount of challenges of the task.	H1 - There exists a significant difference in the average amount of challenges that each group answered correctly in the task.

During this chapter, 14 different comparisons are made in order to assess the results of this experiment. These comparisons attempt to classify the reason for significant discrepancies in the results based on three main alternatives: the different tool used, the different expertise of the population and the order which the tasks have been performed. For readability purposes the following notations are used:

GA and GB - Groups A and B, used to refer to the group of participants that were given versions A and B, respectively.

G1 and G2 - Groups 1 and 2, used to refer to the group of participants which were considered to not have any experience in ML and DL and to those that have basic knowledge of these fields as seen in section § 5.1.2, respectively.

GA_1, GA_2, GB_1, and GB_2 - An intersection between GA and G1, GA and G2, GB and G1 and GB and G2 respectively. This means that, for example, GA_1 is the group of participants that were considered to not have any experience in ML and DL from the participants that were given version A.

S1 and S2 - The first and second set of tasks given to a group of participants, respectively.

T1, T2, T3, and T4 - Annotations referent to Task 1, Task 2, Task 3 and Task 4, respectively.

C1-C14 - All 14 comparisons are annotated with the form of C# which the # is referent to each comparison. An individual description of what is being compared in these 14 topics is further explained under the form of a table in Table 5.6.

All the comparisons presented in the next sections were considered to be the most relevant to analyze. These make an attempt to analyze what is the main factor that is affecting the participant's performance from three possible answers: (i) the tool being used, (ii) the different expertise of the participants or (iii) the order the tasks are being performed in.

The authors consider that the best way to interpret them is to first understand that the important factor taken into account from comparisons C1 through C6 are the versions, which leads to having a more heterogeneous population (of both participants from Groups 1 and 2) creating the possibilities of reaching general conclusions about the usage of the visual editor.

The rest of the comparisons are expertise-based, which means that only half of the population is being taken into account per comparison, leading to more niche conclusions, but interesting nevertheless. These last comparisons have the main objective of assessing how experience influences the usage of the developed editor when compared to the usage of Keras.

All these comparisons are summarized in Table § 5.6. As an example of how to read this table, in the C7 comparison, we are comparing the time required to complete the first set of tasks (S1) between Groups G1A and G2A.

5.3.2 Environment Comparisons: All Test Subjects

When reviewing the comparisons that are referent to the performances of the entire population, and when the only varying factor is the tool that it is being used (C1 through C6), it is possible to draw some conclusions not only from the descriptive statistics as shown by Table 5.7 but also from the tables that provide the results of the statistical analysis, being them Table 5.8 and Table 5.9. Next, we will analyze and discuss these results for each one of the four tasks:

Task 1: This task was the one that differed the most in terms of execution from all four; as even though the end result was the same, participants using Keras had to access a specific online guide while the ones using the editor were given a very intuitive textual guide. The objective of this task

Table 5.6: Representation of the analyzed comparisons between the time each group required during the experiment. On the first and second row, all groups and sets of tasks are presented, respectively.

Comparisons	GA		GB		G1A		G2A		G1B		G2B	
	S1	S2	S1	S2	S1	S2	S1	S2	S1	S2	S1	S2
C1	●		●									
C2		●		●								
C3	●	●										
C4			●	●								
C5	●			●								
C6		●	●									
C7					●		●					
C8						●		●				
C9									●		●	
C10										●		●
C11					●				●			
C12						●				●		
C13							●				●	
C14								●				●

was simply to prepare the user for the following ones by making them comfortable with different developing environment aspects. On a final note regarding this task and as it has already been mentioned, the copy and paste feature provided participants the ability to achieve overwhelmingly fast completion times and should have been disabled.

With this in mind, it was expected that the participants using Keras would be faster than the ones that had to create the models with the editor. Such phenomenon is possible to be seen when considering the results presented in Table 5.8, namely (a) (b) and (c). All values are below 0.05 which means the null hypothesis, in regards to the populations completing the task in similar speed, can be rejected. The first two values (a) and (b) refer to comparisons between participants using different environments (C2 and C4) and when taking into account the *Mean T1* value of Group GB using Keras in Table 5.7 (1.24 minutes), it is safe to say that this is the outlier when considering only Groups GA and GB. We assume that this was due to both the combination of the possibility of using the *copy-and-paste* feature and the fact that the experiment had been going on for about 30 minutes, threats such as social pressure were already mitigated for the participants of Group GB.

Task 2: The second task had a similar structure to Task 1 but no guide or detailed instructions were provided so it was considered more difficult. Participants were expected to struggle with the interpretation of some concepts, especially those that have limited experience with ML, and most likely would need to search the Keras documentation or carefully interpret the editor.

When taking into account all of the participants it is not much apparent this distinction between the expertise. By comparing the performance of Groups GA and GB when not performing tasks in the same development environment (C1 through C4) all of these significances from the

Table 5.7: Number of participants (N), average task completion time, and respective standard deviation for each group of participants. In some cases, not all participants managed to finish a task (thus having a different value for N).

Group	*N*	*Env.*	*Mean T1 (mins)*	*Mean T2 (mins)*	*Mean T3 (mins)*	*Mean T4 (mins)*
GA	8	Keras	2.02 ($\sigma = 0.65$)	3.29 ($\sigma = 0.98$)	10.2 ($\sigma = 0, N = 1$)	9.27 ($\sigma = 0.21, N = 2$)
	8	Editor	2.28 ($\sigma = 0.19$)	1.24 ($\sigma = 0.65$)	6.45 ($\sigma = 0.83$)	2.85 ($\sigma = 0.63, N = 2$)
GB	8	Keras	1.24 ($\sigma = 0.65$)	2.28 ($\sigma = 0.19$)	13.27 ($\sigma = 0, N = 1$)	6.85 ($\sigma = 2.20$)
	8	Editor	2.38 ($\sigma = 0.57$)	2.38 ($\sigma = 0.62$)	6.18 ($\sigma = 0.77$)	3.06 ($\sigma = 0.44$)
G1A	4	Keras	2.25 ($\sigma = 0.71$)	4.01 ($\sigma = 0.77$)	(N = 0)	(N = 0)
	4	Editor	2.27 ($\sigma = 0.22$)	2.71 ($\sigma = 0.38$)	6.31 ($\sigma = 0.65$)	2.85 ($\sigma = 0.39$)
G1B	4	Keras	2.02 ($\sigma = 0.14$)	4.94 ($\sigma = 1.39$)	(N = 0)	8.40 ($\sigma = 0, N = 1$)
	4	Editor	2.28 ($\sigma = 0.15$)	2.50 ($\sigma = 0.61$)	2.27 ($\sigma = 0.15$)	3.20 ($\sigma = 0.42$)
G2A	4	Keras	1.78 ($\sigma = 0.59$)	2.56 ($\sigma = 0.48$)	10.2 ($\sigma = 0, N = 1$)	9.27 ($\sigma = 0.21, N = 2$)
	4	Editor	2.28 ($\sigma = 0.19$)	2.45 ($\sigma = 0.39$)	6.59 ($\sigma = 1.06$)	2.85 ($\sigma = 0.88$)
G2B	4	Keras	1.22 ($\sigma = 0.97$)	2.63 ($\sigma = 0.85$)	13.27 ($\sigma = 0, N = 1$)	5.29 ($\sigma = 0, N = 1$)
	4	Editor	2.50 ($\sigma = 0.84$)	2.26 ($\sigma = 0.71$)	2.50 ($\sigma = 0.84$)	2.93 ($\sigma = 0.47$)

Independent-Samples T-Tests are akin and close to rejecting the null hypothesis but only (d) and (e) manage to do so. The reason why there aren't more significant differences, and the values that do reject the null hypothesis barely do it (since they are very close to 0.05), is assumed to be due to the heterogeneity of the population, mixing those that have experience with Keras and those who do not. This is analyzed in more detail in § 5.3.3.

Task 3: During the execution of this task it was noticed that participants using Keras greatly struggled with the creation of a DL model with multiple inputs and outputs. Keras is a library that has a multitude of guides explaining how to work with it but the majority of them (as can be observed, at the time this dissertation was written, with a simple web search) use the popular *Sequential* abstraction. Although abstractions like this one are easy to use, a lot of knowledge and understanding of the subject is hidden from users. This was exactly what happened since some of the participants, especially the ones with no experience on this field, were not capable of understanding that a DL model could have multiple inputs and outputs during the experiment. Since barely any participants managed to complete this task while using Keras, Independent-Samples T-Tests were expected to be ineffective due to the lack of data to compare. Therefore it was assumed that the option to analyze a binary comparison of what participants managed to successfully complete in this task as seen in Table 5.9 would be wiser.

By analyzing Table 5.9 it is immediately possible to see that when comparisons in which the development environment is different for the two groups are made (C1 through C4) all results provide extremely high significance since all are below 0.01. This is because from 16 participants, when using Keras, only 3 managed to complete the first part and only 1 person successfully completed the second step. The first part involved the creation of a DL model with multiple inputs and outputs, and the second the participants were required to adapt the model's shape to previously given values. The complete opposite of this reality is portraited when viewing the same participants using the developed editor which the only issue was related to 1 person that could not complete the second part of this task.

To conclude, based on the obtained results, it is possible to say that the null hypothesis referent to both compared populations achieving a similar degree of correct responses can safely be rejected and for the case of Task 3, it can be assumed that when participants used the editor they achieved better results.

On a final note on this task, two particular and somewhat surprising behaviors of the participants using the editor were observed:

- Firstly, the majority of the participants explored the liveness feature of the editor during this task, especially when attempting to reach the desired shapes of the model on all of the layers. Since every time a parameter was changed the participant would be required to generate the code again in order to view the model's current shapes, it made sense to lively update the model with each change in the model. This led to participants quickly realizing characteristics of layers such as what parameters changed a layer's output shape and how it was affected.

- Secondly, and very interestingly, the core behavior of constructing DL models changed for many users when creating such a big and complex model when compared to the previous ones. Instead of a participant adding a layer, connecting it to the model at hand and edit its parameters, many participants chose to add all layers, connect them and only then edit all of the model's hyperparameters. Such behavior was not predicted during the research design, therefore no significant conclusions can be drawn from the simple individual observations of the participant's performance. Nevertheless, the authors believe this change in behavior is interesting to study as it was never observed in the Keras environment.

Task 4: Similarly to Task 3, a lot of participants could not entirely finish this task, therefore it was assumed that an assessment of how many participants of each group managed to successfully complete each part of this task had a greater relevance rather than compare the task's execution time by those that managed to finish it. The Fisher's Exact test (Table 5.9) was again used to compare a participant rather than the chi-square test due to the population's size and the task was divided into 2 topics: simple and complex hyper-parameterization of complex models. The main point of this problem is to show, yet again, how some DL libraries conceal features behind abstractions to make their usage easier and how an interactive application can both be easy to use and present all the features in the same way.

When analyzing Table 5.7 and Table 5.9 it is possible to view right away, in the first two comparisons that take into account the existence of two versions (C1 and C2), very similar cases. Regarding the first part of the task, simple hyper-parameterization, every participant managed to successfully complete it but the opposite can be seen in the second part of the task when looking at the participants that were using Keras. The results of the Fisher's Exact tests on those two comparisons expose this phenomenon, namely (f) and (g) which by having values below 0.05 reject the hypothesis of the two compared populations being similar in terms of correctness of the task.

Table 5.8: Independent-Samples T-Tests of Task 1 and 2 regarding the comparisons between participants subjected to different development environments.

Comp.	T1					T2				
	F	Sig	T	DF	Sig. (2-tailed)	F	Sig	T	DF	Sig. (2-tailed)
C1	0.084	0.776	-1.202	14.00	0.249	1.088	0.315	2.212	14.00	0.044(d)
			-1.202	13.75	0.250			2.212	11.90	0.047
C2	1.956	0.184	4.361	14.00	0.001(a)	10.994	0.005	-2.043	14.00	0.060
			4.361	8.19	0.002			-2.043	7.77	0.076
C3	5.113	0.040	-1.086	14.00	0.296	3.331	0.089	1.909	14.00	0.077
			-1.086	8.16	0.309			1.909	9.09	0.088
C4	0.023	0.881	3.758	14.00	0.002	6.994	0.019	-2.282	14.00	0.039
			3.758	13.80	0.002(b)			-2.282	9.00	0.048(e)
C5	0.149	0.706	2.387	14.00	0.032(c)	2.685	0.124	-0.747	14.00	0.468
			2.387	14.00	0.032			-0.747	11.45	0.470
C6	13.478	0.003	0.796	14.00	0.439	0.516	0.485	-0.557	14.00	0.586
			0.796	8.30	0.448			-0.557	11.80	0.586

5.3.3 Expertise Comparisons: Same Environment

This section describes the results of the comparisons between groups G1A, G2A, G1B and G2B regarding the situations in which the participants are using the same development environment (C7 through C10). Basic descriptive results can be consulted in Table 5.7 and the results of the statistical analysis are in Table 5.10 and Table 5.11 for T1 and T2, and T3 and T4 respectively.

Task 1: As it has been analyzed in § 5.3.2, the performance of the participants during the execution of Task 1 was largely influenced by the use of Keras and the *copy-and-paste* command. For this reason and as can be seen in Table 5.10 no significant results are reached when comparing populations using the same environment and with the different levels of expertise (C7 through C10).

Task 2: Upon comparing people with different levels of expertise and having the same development environment the significant results are (h) and (i) from Table 5.10. The two comparisons that led to these results (C7 and C10) have one aspect in common: they are both comparing people that have less experience in ML and DL to other participants of the same version while using Keras. This means that after comparing the average time of both populations in Table 5.7 (2.25 minutes compared to 1.78 minutes and 2.02 minutes compared to 1.22 minutes) it is possible to conclude that participants that had prior experience in DL perform significantly better than those that did not. Another interesting result regarding this topic is that those using the editor have no significant difference between the means of the times needed to complete this task, therefore suggesting that expertise does not have much of an impact when using the editor.

Task 3: Regarding Task 3 and Table 5.11, and after understanding how many people actually managed to complete it and in which conditions they did so, the situation is not surprising. When comparing experiences between two populations in the same environment the result is similar throughout the different expertise levels of the participants: when using the editor almost everyone

Table 5.9: Fisher's Exact Tests of Task 3 and 4 regarding the comparisons between participants subjected to different development environments. Each comparison is divided into two separate moments. Regarding Task 3, the first and second moments respectively refer to the ability to create DL models with multiple inputs and outputs and to adapt the shapes of the layers of a DL model. Regarding Task 4, the first and second moments respectively refer to the ability to perform simple and complex hyper-parameterization.

Comp.	Group	Tool	T3				T4			
			Inc.	Corr.	N	Sig. (2-sided)	Inc.	Corr.	N	Sig. (2-sided)
C1	GA	Keras	6	2	8	0.007	0	8	8	1.000
	GB	Editor	0	8	8		0	8	8	
	GA	Keras	7	1	8	0.010	6	2	8	0.007(f)
	GB	Editor	1	7	8		0	8	8	
C2	GA	Editor	0	8	8	0.001	0	8	8	1.000
	GB	Keras	7	1	8		1	7	8	
	GA	Editor	0	8	8	0.000	0	8	8	0.007(g)
	GB	Keras	8	0	8		6	2	8	
C3	GA	Keras	6	2	8	0.007	0	8	8	1.000
	GA	Editor	0	8	8		0	8	8	
	GA	Keras	7	1	8	0.001	6	2	8	0.467
	GA	Editor	0	8	8		8	0	8	
C4	GB	Editor	0	8	8	0.001	0	8	8	1.000
	GB	Keras	7	1	8		1	7	8	
	GB	Editor	1	7	8	0.001	0	8	8	0.007
	GB	Keras	8	0	8		6	2	8	
C5	GA	Keras	6	2	8	1.000	0	8	8	1.000
	GB	Keras	7	1	8		1	7	8	
	GA	Keras	7	1	8	1.000	6	2	8	1.000
	GB	Keras	8	0	8		6	2	8	
C6	GA	Editor	0	8	8	1.000	0	8	8	1.000
	GB	Editor	0	8	8		0	8	8	
	GA	Editor	0	8	8	1.000	0	8	8	1.000
	GB	Editor	1	7	8		0	8	8	

managed to reach a correct answer but when using Keras the opposite occurred, independently of the expertise with some exceptions that are not statistically significant.

Task 4: Finally the situation in Task 4, when the environment is the same, is very similar to the performance of the participants in Task 3. This means that when comparing different levels of expertise there are no statistically significant discrepancies between the groups being compared.

5.3.4 Expertise Comparisons: Different Environments

In this section, the comparisons between groups G1A, G2A, G1B, and G2B are made, specifically in the situations which different development environments are being compared (C11 through C14). For reference, the descriptive results of the experiment are presented in Table 5.7 and the

Table 5.10: Independent-Samples T-Tests of Task 1 and 2 regarding the comparisons between participants of different expertise subjected to the same development environment.

Comp.	T1					T2				
	F	Sig	T	DF	Sig. (2-tailed)	F	Sig	T	DF	Sig. (2-tailed)
C7	0.000	0.989	1.007	6	0.353	0.641	0.454	3.212	6	0.018(h)
			1.007	5.825	0.354			3.212	5.050	0.023
C8	0.129	0.732	-0.087	6	0.934	0.151	0.711	0.945	6	0.381
			-0.087	5.896	0.934			0.945	5.999	0.381
C9	39.539	0.001	-0.523	6	0.620	0.022	0.887	0.514	6	0.626
			-0.523	3.182	0.635			0.514	5.878	0.626
C10	3.462	0.112	0.102	6	0.922	0.253	0.633	2.832	6	0.030(i)
			0.102	3.125	0.925			2.832	4.956	0.037

results of the statistical analysis are in Table 5.12 and in Table 5.13 for T1 and T2, and T3 and T4 respectively.

Task 1: If the previous analysis concerning Task 1 is being taken into account then it is not much of a surprise when finding that when comparing participants without much experience in DL from GA to the same type of participants from GB (using the editor) details a significant difference in execution times. These participants were already about 30 minutes into the experiment and naturally were more confident and less stressed. This led to the abuse of the *copy-and-paste* command without taking into account the code that was being written whereas the participants using the editor needed to build the model normally without any shortcuts.

Task 2:When analyzing the differences of Task 2 in Table 5.12 it is possible to understand that the comparisons with statistical significances are those between the participants with not much experience in ML and DL during both their sets of tasks. This is interesting to analyze because, as can be seen in Table 5.7 (4.01 minutes compared to 2.50 minutes and 2.71 minutes compared to 4.94 minutes), participants without much experience in these fields perform significantly faster in the editor.

Task 3: As it was discussed in § 5.3.2, participants performed much better using the editor than Keras both to create complex DL models and to change its different shapes. The only non-significant statistical results were due to some people with prior experience managing to complete at least 1 part of the task. This leads to believe that it is definitely a matter of comparison between different development environments whilst the results obtained from the comparison of equal expertise levels is secondary. Nevertheless, it is important to note that, according to the results, the more expert and familiar with Keras a user is, the less significant statistical results it is possible to obtain concerning the correctness of the answers.

Task 4: Finally, by analyzing the results of Task 3 it is possible to assume that the threat of the size of the population played a big part here, since the only possible way of obtaining a significant statistical comparison is to have all participants fail a topic of a task, whilst the opposing group's participants all correctly answered that topic. As seen in Table 5.13 there are multiple cases in which 3 participants incorrectly answered a topic and the other participant guessed it correctly

Table 5.11: Fisher's Exact Tests of Task 3 and 4 regarding the comparisons between participants of different expertise subjected to the same development environment. Each comparison is divided into two separate moments. Regarding Task 3, the first and second moments respectively refer to the ability to create DL models with multiple inputs and outputs and to adapt the shapes of the layers of a DL model. Regarding Task 4, the first and second moments respectively refer to the ability to perform simple and complex hyper-parameterization.

Comp.	Group	Tool	T3				T4			
			Inc.	Corr.	N	Sig. (2-sided)	Inc.	Corr.	N	Sig. (2-sided)
C7	GA	Keras	4	0	4	0.429	0	4	4	1.000
	GA	Keras	2	2	4		0	4	4	
	GA	Keras	4	0	4	1.000	4	0	4	0.429
	GA	Keras	3	1	4		2	2	4	
C8	GA	Editor	0	4	4	1.000	0	4	4	1.000
	GA	Editor	0	4	4		0	4	4	
	GA	Editor	0	4	4	1.000	0	4	4	1.000
	GA	Editor	0	4	4		0	4	4	
C9	GB	Editor	0	4	4	1.000	0	4	4	1.000
	GB	Editor	0	4	4		0	4	4	
	GB	Editor	0	4	4	1.000	0	4	4	1.000
	GB	Editor	1	3	4		0	4	4	
C10	GB	Keras	4	0	4	1.000	0	4	4	1.000
	GB	Keras	3	1	4		1	3	4	
	GB	Keras	4	0	4	1.000	3	1	4	1.000
	GB	Keras	4	0	4		3	1	4	

and the participants from the other group also correctly guessed that topic. Nevertheless, the point made in § 5.3.2: participants struggled they had to use Keras to perform complex hyper-parameterization.

5.3.5 Post-Tasks and Post-Experiment Questionnaire

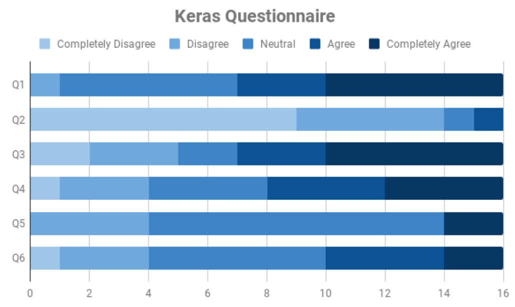
The answers to the questionnaires analyzed in this section follow a more descriptive analysis since there was a lack of divergence in the results. This means that overall the population had similar experiences and opinions regarding the use of the interactive tool when compared to Keras under the circumstances of the experiment.

By analyzing the descriptive results of the post-tasks questionnaires in Table 5.6, it is possible to say that overall, people's opinions diverged more in Q2, Q3, and Q4 which are referent to the ability to create complex models and the need of consulting online documentation or guides. The results of the questionnaires suggest that when using Keras, people are more dependant on the access to either the library documentation or online guides and have more trouble of creating models with multiple inputs and outputs. Such results are expected since, as observed in the previous sections, participants using Keras had more trouble completing certain aspects of some tasks, namely those that involved the creation and edition of complex models.

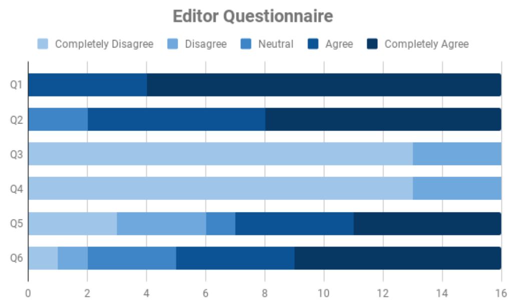
Table 5.12: Independent-Samples T-Tests of Task 1 and 2 regarding the comparisons between participants of different expertise subjected to different development environments.

Comp.	T1					T2				
	F	Sig	T	DF	Sig. (2-tailed)	F	Sig	T	DF	Sig. (2-tailed)
C11	2.613	0.157	-0.069	6	0.947	0.139	0.723	3.091	6	<u>0.021</u>
			-0.069	3.256	0.949			3.091	5.713	0.023
C12	0.611	0.464	7.787	6	<u>0.000</u>	1.537	0.261	-3.097	6	<u>0.021</u>
			7.787	5.148	0.000			-3.097	3.450	0.044
C13	2.602	0.158	-1.386	6	0.215	0.323	0.590	0.702	6	0.509
			-1.386	5.404	0.220			0.702	5.297	0.512
C14	3.123	0.128	2.151	6	0.075	2.603	0.158	-0.397	6	0.705
			2.151	3.226	0.114			-0.397	4.201	0.711

Concerning the rest of the results of these questionnaires, they are fairly similar. Q1 refers to the ability to create simple models which participants managed to do in both Keras and the visual editor. Q5 asks if participants manage to correct mistakes that they made and Q6 inquires participants whether they felt like they have successfully completed each task. Naturally, those that did not manage to complete some tasks would disagree with this last question.



(a) Results of the questionnaire referent to the tasks completed using Keras.



(b) Results of the questionnaire referent to the tasks completed using the developed tool.

Figure 5.6: Results of the post-tasks questionnaires.

In regards to the post-experiment questionnaire the obtained results corresponded to the expectations. Albeit being heavily biased because of the experiment they had just taken part of, participants showed a positive response when asked if they preferred to use an interactive tool rather than the Keras option, as can be seen in Q1 and Q2. Q3, Q4, and Q5 reference the use of some features of the editor being them the default values in the optional parameters, the save and load feature and the liveness, respectively, and as can be seen in Figure 5.7, the general population agrees that they were indeed handy to have. Q6 and Q7 reiterate the questions about how easy it was to correct mistakes in the editor and in Keras, respectively, in an attempt to see a change of heart but alas the participants were consistent. In the final three questions, it was asked if the participants joined a project in which there was a need to create a DL architecture if they would

Table 5.13: Fisher's Exact Tests of Task 3 and 4 regarding the comparisons between participants of different expertise subjected to different development environments. Each comparison is divided into two separate moments. Regarding Task 3, the first and second moments respectively refer to the ability to create DL models with multiple inputs and outputs and to adapt the shapes of the layers of a DL model. Regarding Task 4, the first and second moments respectively refer to the ability to perform simple and complex hyper-parameterization.

Comp.	Group	Tool	T3				T4			
			Inc.	Corr.	N	Sig. (2-sided)	Inc.	Corr.	N	Sig. (2-sided)
C11	GA	Keras	4	0	4	0.029	0	4	4	1.000
	GB	Editor	0	4	4		0	4	4	
	GA	Keras	4	0	4	0.029	4	0	4	0.029
	GB	Editor	0	0	4		0	4	4	
C12	GA	Editor	0	4	4	0.029	0	4	4	1.000
	GB	Keras	4	0	4		0	4	4	
	GA	Editor	0	4	4	0.029	0	4	4	0.143
	GB	Keras	4	0	4		3	1	4	
C13	GA	Keras	2	2	4	0.429	0	4	4	1.000
	GB	Editor	0	4	4		0	4	4	
	GA	Keras	3	1	4	0.486	2	2	4	0.143
	GB	Editor	1	3	4		0	4	4	
C14	GA	Editor	0	4	4	0.143	0	4	4	1.000
	GB	Keras	3	1	4		1	3	4	
	GA	Editor	0	4	4	0.029	0	4	4	0.143
	GB	Keras	4	0	4		3	1	4	

like to use a tool such as the one they had access too in different states of development. Q8 asks specifically if they would use the tool as is and, surprisingly, the participants agreed, even though the tool lacks a lot of user experience features and could be further improved. In Q9 it is asked if the participants in the previously mentioned scenario would use an improved version of the tool, essentially with better user experience, and as expected the answers were very positive, and Q10 proposes the use of a tool that integrates not only the creation of the architecture of a DL model but also other phases such as the training or the data preparation; in this final question the results were also similarly positive.

5.3.6 Discussion

During the execution of each task, some notes were taken about individual aspects of each participant that were not explored or taken into account during the design of this experiment. A discussion of these characteristics and a general discussion are presented on the following topics:

- As a rule of thumb, those that used the developed editor performed better than the ones that used Keras, both in terms of correct answers and used time. The only case which such did not occur was during Task 1, specifically the participants from Group B, but such exception was assumed to be caused by how the task was presented to the participants.

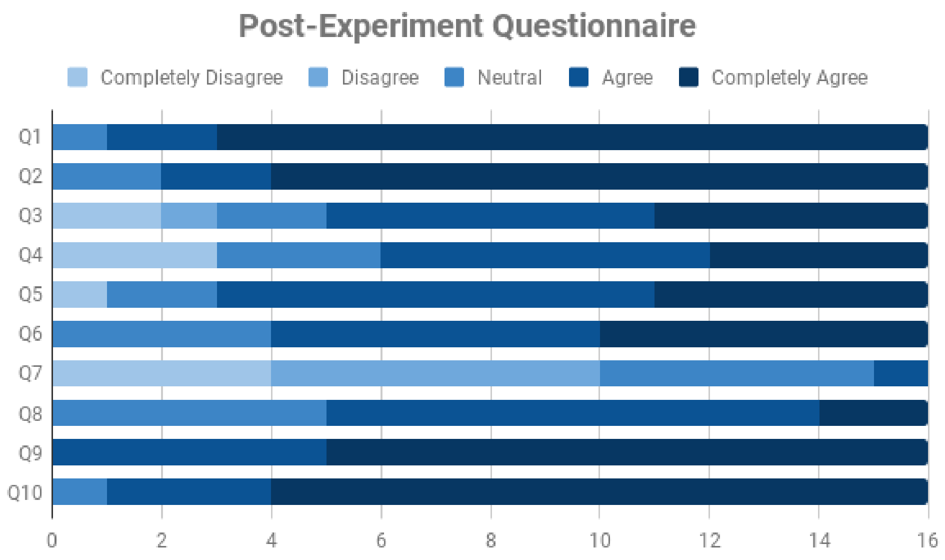


Figure 5.7: Results of the post-experiment questionnaire.

- Regarding short and simple tasks, participants had no issues in completing them but Keras users were significantly slower. Such difference could be due to the fact that many participants tried to and sometimes successfully guessed some of the hyperparameters names and values rather than searching their implementation on the documentation. Such behavior could make some tasks easier and faster to do when comparing to the performance of those using the editor *if* all guesses were correct. As expected, guessing method's invocations is not a reliable way to solve the tasks resulting in time being wasted both in the act of guessing and the act of correcting such mistakes.

- While using the visual editor, especially during complex tasks such as Tasks 3 and 4, participants altered their usual way of creating DL architectures: on Tasks 1 and 2 participants would usually follow each instruction by creating a layer and changing its parameters; this behavior continued when using Keras in Tasks 3 and 4 but when the visual editor was used, most of the participants chose to add every layer to the model, and only then accessed specific layers and changed their parameters. This was one of the most interesting phenomenons observed because it was a change in the behavior of the participants when dealing with the tasks. This change can be compared to a change from a depth-first approach to a breadth-first one; as many layers are being put into the model but none are being fully completed right away. Such behavior requires further study before any major conclusion can be drawn from it, but for tools such as the developed editor, it can be very beneficial since, for example, with the live mode, a user will receive feedback (of errors) of a model with all layers connected but no parameters set as opposed to having an incomplete model with a few layers fully characterized.

- As seen in the results of Task 3, in the Keras tasks almost no participant was able to understand how shapes were affected. On the other hand, while using the visual editor many participants were able to understand what was happening once they lively edited a model's hyperparameters and saw changes to the model happening on the fly. This is especially interesting to analyze during the sessions of the participants with little to no experience with ML, as they barely understood what an output shape is but managed to realize, for example, that filters and kernel size of a 2d convolutional layer have an effect on it.
- When taking into account both the results of the tasks and the feedback from the questionnaires, although some statistical tests prove to be either inconclusive or have provided low confidence, especially due to the size of the population, it is safe to assume that the participants of this experiment definitely preferred to use the developed tool. The ability to visualize and interact with an abstraction of the creation of an extremely complex algorithm definitely has its pros. One of the most prominent advantages is the lack of needing to continuously check the documentation or an online guide which all participants needed at some point while performing the tasks with Keras. When using the developed editor many participants did not require any guide whatsoever, and instead consulted information blocks associated to each parameter, providing them with information similar to the documentation but without the overhead of navigating online.
- In the visual editor, the majority of the participants criticized some user experience issues although all managed to adapt to it, and some even incorporated shortcuts such as the use of tabs to quickly navigate through labels' parameters.
- Both the Independent-Samples T-Tests and the fisher's exact tests are useful to analyze this type of experiments and results since what was observed during the sessions complies with the results of the referred statistical tests. Sadly, in cases such as the comparison between the expertise of groups, whereas the size of the compared populations is only four participants, the validity and significance of the given results are dubious. Nevertheless, instead of taking these specific results as irrefutable proof of the developed editor having a better performance than Keras for these tasks, it can rather be taken as evidence that suggests that the behavior of people using an interactive, visual and live tool is significantly different than the ones using a traditional code-based approach.

5.4 Validation Threats

The following list describes the relevant issues encountered while performing the experiment that could potentially invalidate a part of the validation process. Nevertheless, all threats were taken into account while analyzing the results and some were even predicted and mitigated during the preparation of the tasks.

Social pressure. It is believed that during the experiment the participants felt somewhat unease due to the possible pressure that they could have been feeling while doing the tasks. This is a regular problem with quasi-experiments and one of the best ways to mitigate such issue is to perform the test with multiple participants so that outliers that falter in these conditions have less impact on the final result.

Quantity of participants. The main idea behind this dissertation is to give evidence that a visual editor such as the one developed creates impact when applied to fields of ML, namely DL. Therefore if such hypothesis is correct it is expected that when two populations, one using the editor and the other using other means, try to create architectures of DL models there will be a significant difference between them. Although the quantity of the participants is not ideal, in some cases of the experiment it is proved mathematically that there is a significant enough difference between the two populations. Nevertheless, this is one of the main threats and further validation with more participants would be relevant to make.

Framework used. For the baseline results, the participants used the Keras library. The choice of this library was mainly based on its popularity and the ease of working with it, although no evidence is presented that a different DL library could change the results of this experiment. Although this is true, the fact that the presented results invalidate the proposed null hypothesis on some aspects when compared to the results obtained with one of the most popular DL frameworks has a high degree of relevance for this dissertation.

Extent of the experiment. The tasks of the experiment were somewhat short and each tested different aspects of the creation process of a DL model. Although the tasks can be used for assessing the performance of participants without a lot of experience in this area it would be relevant to examine their evolution when performing multiple tasks that test the same problematic situations but with different use-cases.

5.5 Summary

In this chapter, an in-depth evaluation was made based on the results obtained from a quasi-experiment that was performed with the help of 16 participants. This experiment is described in the first section along with an analysis of the participants. During this analysis, it was seen that the expertise of the participants regarding the fields of ML and DL could constitute a potential threat to the evaluation of the obtained results. For this reason, the participants were divided into two groups: those that already had previous contact with ML and Keras and those who did not. It was also explained in this section that the focus of the analysis of the results was based on how fast the participants managed to complete the tasks and how correct their answers were.

The process of evaluation is defined as having two groups of 8 participants, each being given the same tasks and the same two environments but one group starts by using Keras and the other start with the editor. This has the objective to reduce the threat of potentially always using one environment before the other, thus producing biased results. All of the tasks are also explained on this chapter, namely the distinction between simple tasks such as the first and second ones, a task that requires interpretation of an image of an architecture with multiple inputs and outputs and a final task that involved the edition of this complex model. Such division is important in order to decrease other potential threats to the experiment such as an overlap of problems in the same task.

In the third section, the results are discussed together with a statistical analysis. This analysis mainly breaks down to the comparison between the usage of the editor and Keras, and comparisons between the groups of participants with different degrees of expertise. This last point enables critical thinking regarding the correlation between different results of two groups and the expertise of the participants that constitute them. The two main conclusions concerning this population and this experiment were that, overall, participants using the editor performed the given tasks faster and more correctly than the ones using Keras. Also, although expertise had a great impact on the Keras usage it made no significant difference in those using the editor.

One of the most important factors that were observed during multiple experiments and is presented in the discussion of the results is the fact that participants, at a certain point, started adapting the usage of the tool to their needs rather than adapting themselves to the editor. This was verified by viewing how people were creating the models, especially the complex ones such as the one in the third task. The process consisted of using a breadth-first-like approach to create a DL architecture, starting with creating all the layers, connecting them and only then editing their parameters. This was assumed to be the main reason why people using the editor had such significantly faster times than those using Keras although the authors find it relevant to study this change in behavior further.

It is important to realize that this experiment suffered from multiple validation threats, being the most relevant one the reduced size of the population. This means that all obtained results, especially the ones where two groups of 4 participants each were compared, should be considered with a grain of salt since, although some tests indicate how much better the editor was when compared to the Keras alternative, a more robust experiment where the number of participants is higher and more tasks are performed could potentially change some of the conclusions that were reached. Nevertheless, for this dissertation's purpose, which is mainly to indicate that a visual, interactive and live editor can have a significant and positive impact on the development of DL architectures, it was assumed that the experiment and associated evaluation was relevant enough.

Chapter 6

Conclusions

Contents

This chapter aims to give an overview of the project that was made in § 6.1 along with the main conclusions and finally the scientific contributions that can be extracted from this dissertation in § 6.2. On some final notes, in section § 6.3, a guideline of what doors this project might have opened for further research is presented.

6.1 Summary

At the start of this dissertation, multiple studies concerning deep learning and visualization tools were explored in Chapter 2 (p. 5). All of the information provided had as its main goal to make the readers realize two things: deep learning algorithms have been steadily gaining notoriety as possible and good solutions to some problems and tools that provide visualization over a problem and can be interacted with in a live manner are becoming the norm due to their helpfulness and possible ease of use. A prime example of a visual and interactive tool, as it was referred to in the aforementioned chapter, is Unity [uni17]. This program and others like it provide constant visualization of the software that is being created and features such as being able to see different parts of a project at the same time and even editing some parameters while it's being executing in order to see the possible changes being applied on the fly. Naturally, when comparing developers that are given such powerful software to make a project to others that are expected to do the same but in a more traditional coding way the manner of creating software of these two groups will be different. Without extensive research and experiments regarding the two previous cases, it is impossible to say that one approach will be better than the other, in fact, in the authors' opinions, it is impractical to reach such conclusion because the nature of the two approaches is just too different.

After taking into account the topics of deep learning and live, visualization tools, the interest in associating the two fields rises and further research concerning the current state of the art of the intersection of these two topics are made. As was seen in § 2.4 many projects that correlate parts of a DL project with visualization have been made, namely statistical analysis of the training process of DL models and end results of such projects. Although this is true, the visualization tools are analyzed and the majority are not meant to be scaled or have any other purposes other than the one that they were created for. This leads to two main conclusions:

- There exists an interest in an in-depth visualization of parts of DL projects. This idea is conveyed by the multiple projects that require the use (or even the creation in some cases) of software to view things such as graphs being lively updated to analyze issues such as the convergence of an algorithm to the desired solution. Some projects that were covered in the literature review also explain cases in which live interaction with such software are extremely helpful, for example, to view how changing a parameter affects an algorithm.

- The architectural phase of DL projects has been made easier with python libraries such as Keras and Theano but could possibly be improved with the use of interactive, live software. A program like this one could be scaled and extended to provide many of the different features of the different reviewed projects that make use of visualization tools, therefore potentially making it easier for less experienced people to experiment with deep learning.

Having made this research it is possible to construct a hypothesis which is shown in Chapter 3 (p. 19). This hypothesis states the possibility of people performing certain aspects of a DL project faster and more correctly when provided with a live, visual and interactive tool when compared to others using more traditional, code-based approaches. This is further broken down into research questions which both the implementation and the evaluation chapter attempt to answer.

In Chapter 4 (p. 23) a complete review of the created software was made, namely the architecture and why and how some features were implemented. The main focus of the software was the creation of DL architectures in a scalable and lively interactive way. The way that this was done was by identifying the two very different aspects of a DL model which is its conceptualization and creation and joining them together with a visual metaphor for what a DL algorithm. This visual metaphor is simply layers connected to other layers, inputs and outputs and all of the associated parameterization. The main challenge of the software was to decouple this visualization from the code that would be generated by the interpretation of the model in order to have two modules: the visualization and the generation of the code to be executed. This architecture created scaling opportunities such as supporting multiple libraries, creating custom layers and parameters and even being able to extend such project to include different phases of a DL project other than the conceptualization of the algorithm.

Finally, in Chapter 5 (p. 37) an evaluation is done in order to get a grasp of the potential impact of the created software. This was done with empirical evidence extracted from a quasi-experiment that was performed with 16 participants and although there are many threats to this study the authors assumed that the results were relevant enough to have conclusions drawn from them regarding the hypothesis of this dissertation. By analyzing the results it was possible to conclude that when comparing the ability to create a DL architecture between a group of people using the developed software and a typical DL library such as Keras, those using Keras were slower and struggled harder to correctly implement some aspects of a DL model. The results are further explored when taking into account the expertise of the participants in this field, reaching the conclusion that when using the developed deep learning editor, participants that had more experience in the fields of ML and DL presented little to no difference when compared to those that had almost no experience. The opposite result was reached, as expected when the participants used the Keras libraries since participants with more experience performed better than the others.

6.2 Main Conclusions

Regarding the hypothesis of this project and the resulting research questions, according to the obtained results, it is possible to validate it up to a certain extent. Although the experiment was not extremely robust and the obtained results had some threats to their validity, it is quite possible to understand that people that are subjected to the developed software perform differently than the ones using code-based approaches such as using Keras.

When taking into account the research questions and the study that was made, it is possible to conclude that to up to a certain extent, people using a visual software converge better and faster to an acceptable DL architecture both when detailed instructions and simple visual abstractions of the solution are provided. Thus, tools such as the one created can help to recreate DL models, especially with the use of liveness and interactive capabilities. When taking into account the use of the editor regarding simple and complex DL architectures the results end up being similar, obtaining the desired model easier and faster when compared to the use of DL libraries. The same can be said for the edition of hyperparameters. Finally, when considering people with different degrees of expertise in fields such as ML and DL, although this type of experience has a great impact on the use of libraries such as Keras, it provides no significant advantage when creating DL models on the developed editor. This is of great importance since the barrier between people that want to use DL and the expertise required to explore projects of this nature is greatly diminished.

As a final note, one of the most important factors that led to faster creation of DL models was assumed to be the change observed in the behavior of creating such models. As it has been explained in § 5.3.6, there was a noticeable change in the methodology of performing the tasks by the participants, which can be compared to the difference in changing from a depth-first to a breadth-first approach. This change in behavior has points in common with the approaches of creating software in programs such as Unity as it was mentioned before, therefore strongly indicating that live, visual and interactive tools have the potential to have a different impact when compared to traditional code-based approaches.

6.3 Main contributions and Future Work

As it has been stated in some parts of this project, this dissertation's goal is not to create a full-fledged deep learning visual IDE but to study how some visualization aspects impact the performance of people when creating DL architectures. Having this said, this dissertation aims to achieve three scientific contributions:

State of the Art. In Chapter 2 (p. 5) an analysis of the research that has been and is currently being conducted of three topics is done, being them deep learning, live development, and visual programming. This is explored in more depth by doing an analysis of recent surveys on tools that aid deep learning projects with visual representations and by individually correlating the encountered projects with the levels presented in Tanimoto's liveness hierarchy.

Deep Learning Visual Editor. During this project a tool capable of creating DL code based on user-created visual metaphors of a DL model was created. This tool currently generates code using the Keras library but not only can customize it as well as supporting completely different libraries. This means that a user can create one visualization of a model and generate code, for example, for Keras, Tensorflow, and Theano. The tool also offers live development which as seen from the evaluation, has a positive impact when comparing to traditional, code-based approaches.

Evaluation of the Editor. In Chapter 5 (p. 37) a detailed evaluation of a quasi-experiment that compared the use of the developed tool to the use of existent DL libraries, specifically Keras, is performed. The results suggest that overall, people tend to perform better when using the editor. During this evaluation, not only the use of different environments to create DL architectures are assessed but also how different levels of expertise impact the use of the tool when taking into account the performance of the participants when they used Keras.

This project also creates opportunities to delve further into some of the explored topics, noticeably regarding the extension of the developed tool and execution of a more robust study.

Polish the developed tool. As it is now, the created software is stable and the most problematic issues are related to its usability. Although the participants of the quasi-experiment managed to adapt themselves to the tool, the objective is the opposite: to have a tool capable of being used as a user wants. This means issues such as the visual aspect of the tool, the animations and even presenting more details of the layers can be addressed to improve the tool's usability. This topic can be explored even up to adding more small features such as the ability to create 2 models simultaneously.

Extension of the developed tool. As it was seen in Chapter 2 (p. 5), many of the projects that make use of visual aided deep learning approaches often have standalone niche features. The developed tool has at its core the creation of DL models that often could make use of these specific niches. This means that extending the developed tool in order for it to harbor such features is of great interest, namely having the possibility of feeding a model data, training it and make an analysis of the output. This topic can be explored for the creation of a tool, much like popular examples that have been referred such as the Eclipse IDE, which could be capable of enabling researchers to make use of its modularity to extend it for the purpose of their own investigations.

A more robust evaluation. Both during and after the evaluation process the authors identified a set of validation threats § 5.4. Addressing them in further studies would make our conclusions more robust.

All in all, the authors consider that visual aided deep learning through the use of live and interactive software is a promising topic for future research.

References

[AJY+17] Bilal Alsallakh, Amin Jourabloo, Mao Ye, Xiaoming Liu, and Liu Ren. Do Convolutional Neural Networks Learn Class Hierarchy? *arXiv e-prints*, page arXiv:1710.06501, October 2017.

[ARC+19] Ademar Aguiar, André Restivo, Filipe Figueiredo Correia, Hugo Sereno Ferreira, and João Pedro Dias. Live software development — tightening the feedback loops. In *Proceedings of the 5th Programming Experience (PX) Workshop*, apr 2019.

[Bow03] Keith M Bower. When to use fisher's exact test. In *American Society for Quality, Six Sigma Forum Magazine*, volume 2, pages 35–37, 2003.

[CHJO16] Shan Carter, David Ha, Ian Johnson, and Chris Olah. Experiments in handwriting with a neural network. *Distill*, 2016.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[Con97] Matthew J. Conway. *Alice : Easy-to-Learn 3D Scripting for Novices*. PhD thesis, 1997.

[CPM+18] D. Cashman, G. Patterson, A. Mosca, N. Watts, S. Robinson, and R. Chang. Rnnbow: Visualizing learning via backpropagation gradients in rnns. *IEEE Computer Graphics and Applications*, 38(6):39–50, Nov.-Dec. 2018.

[CSP+16] Sung Gon Chung, Sangho Suh, Cheonbok Park, Kyeongpil Kang, Jaegul Choo, and Bum Chul Kwon. Revacnn : Real-time visual analytics for convolutional neural network. 2016.

[dWD10] Joost de Winter and Dimitra Dodou. Five-point likert items: t test versus mann–whitney–wilcoxon. *Practical Assessment, Research and Evaluation*, 15, 01 2010.

[FG16] Nassir Navab Federico Tombari Felix Grün, Christian Rupprecht. A taxonomy and library for visualizing learned features in convolutional neural networks. In *ICML Visualization for Deep Learning Workshop*, 2016.

[Fou01] Eclipse Foundation. Eclipse, 2001. Last accessed 02 February 2019.

[Goo11] Judith Good. Learners at the Wheel. *International Journal of People-Oriented Programming*, 1(1):1–24, jan 2011.

[GRNT16] Felix Grün, Christian Rupprecht, Nassir Navab, and Federico Tombari. A Taxonomy and Library for Visualizing Learned Features in Convolutional Neural Networks. *arXiv e-prints*, page arXiv:1606.07757, June 2016.

[GTC$^+$18] Rafael Garcia, Alexandru C. Telea, Bruno Castro da Silva, Jim Tørresen, and João Luiz Dihl Comba. A task-and-technique centered survey on visual analytics for deep learning model engineering. *Computers & Graphics*, 77:30–49, dec 2018.

[Har15] Adam W Harley. An interactive node-link visualization of convolutional neural networks. In *ISVC*, pages 867–877, 2015.

[HHC17] Fred Hohman, Nathan Hodas, and Duen Horng Chau. Shapeshop: Towards understanding deep learning representations via interactive experimentation. *Extended abstracts on Human factors in computing systems . CHI Conference*, 2017:1694—1699, May 2017.

[HKPC18] Fred Hohman, Minsuk Kahng, Robert Pienta, and Duen Horng Chau. Visual Analytics in Deep Learning: An Interrogative Survey for the Next Frontiers. *arXiv e-prints*, page arXiv:1801.06889, January 2018.

[Ima12] ImageNet. Ilsvrc2012, 2012. Last accessed 04 February 2019.

[Jet01] JetBrains. Intellij idea, 2001. Last accessed 02 February 2019.

[KAKC17] Minsuk Kahng, Pierre Y. Andrews, Aditya Kalro, and Duen Horng Chau. ActiVis: Visual Exploration of Industry-Scale Deep Neural Network Models. *arXiv e-prints*, page arXiv:1704.01942, April 2017.

[KD14] Vijay Kotu and Bala Deshpande. *Predictive analytics and data mining: concepts and practice with rapidminer*. 2014.

[KHL11] Damir Kalpić, Nikica Hlupić, and Miodrag Lovrić. Student's t-tests. *International encyclopedia of statistical science*, pages 1559–1563, 2011.

[KKKB14] J. Kramer, J. Kurz, T. Karrer, and J. Borchers. How live coding affects developers' coding behavior. In *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)(VLHCC)*, volume 00, pages 5–8, July 2014.

[Köl08] Michael Kölling. Greenfoot. *ACM SIGCSE Bulletin*, 40(3):327, 2008.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[KSK14] Zehra Karapinar Senturk and Resul Kara. Breast cancer diagnosis via data mining: Performance analysis of seven different algorithms. *Computer Science & Engineering: An International Journal*, 4:35–46, 02 2014.

[LBBH01] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351. IEEE Press, 2001.

[LBM$^+$16] Sebastian Lapuschkin, Alexander Binder, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. The lrp toolbox for artificial neural networks. *Journal of Machine Learning Research*, 17(114):1–5, 2016.

[Lev61] Howard Levene. Robust tests for equality of variances. *Contributions to probability and statistics. Essays in honor of Harold Hotelling*, pages 279–292, 1961.

[Lik32] Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.

[LSC+18] M. Liu, J. Shi, K. Cao, J. Zhu, and S. Liu. Analyzing the training processes of deep generative models. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):77–87, Jan 2018.

[LSL+16] Mengchen Liu, Jiaxin Shi, Zhen Li, Chongxuan Li, Jun Zhu, and Shixia Liu. Towards better analysis of deep convolutional neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23, 04 2016.

[McH13] Mary L McHugh. The chi-square test of independence. *Biochemia medica: Biochemia medica*, 23(2):143–149, 2013.

[MCZ+17] Yao Ming, Shaozu Cao, Ruixiang Zhang, Zhen Li, Yuanzhe Chen, Yangqiu Song, and Huamin Qu. Understanding Hidden Memories of Recurrent Neural Networks. *arXiv e-prints*, page arXiv:1710.10777, October 2017.

[MN10] Patrick E. McKnight and Julius Najab. *Mann-Whitney U Test*, pages 1–1. American Cancer Society, 2010.

[MP43] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943.

[MRR+10] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. The Scratch Programming Language and Environment. *ACM Transactions on Computing Education*, 10(4):1–15, nov 2010.

[NQ17] Andrew P. Norton and Yanjun Qi. Adversarial-playground: A visualization suite showing how adversarial examples fool deep learning. *2017 IEEE Symposium on Visualization for Cyber Security (VizSec)*, pages 1–4, 2017.

[OSJ+18] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018. https://distill.pub/2018/building-blocks.

[PHG+18] N. Pezzotti, T. Höllt, J. Van Gemert, B. P. F. Lelieveldt, E. Eisemann, and A. Vilanova. Deepeyes: Progressive visual analytics for designing deep neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):98–108, Jan 2018.

[QLZT17] Han Qi, Jingqiu Liu, Xuan Zou, and Allen Tang. Bidviz: Real-time monitoring and debugging of machine learning training processes. Master's thesis, EECS Department, University of California, Berkeley, May 2017.

[RA16] Xin Rong and Eytan Adar. Visual Tools for Debugging Neural Language Models. In *Proceedings of ICML Workshop on Visualization for Deep Learning*, 2016.

[RFFT17] P. E. Rauber, S. G. Fadel, A. X. Falcão, and A. C. Telea. Visualizing the hidden activity of artificial neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):101–110, Jan 2017.

[Ros58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[SCD$^+$16] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. *arXiv e-prints*, page arXiv:1610.02391, October 2016.

[SCS$^+$17] Daniel Smilkov, Shan Carter, D. Sculley, Fernanda B. Viégas, and Martin Wattenberg. Direct-Manipulation Visualization of Deep Networks. *arXiv e-prints*, page arXiv:1708.03788, August 2017.

[SGB$^+$18] Hendrik Strobelt, Sebastian Gehrmann, Michael Behrisch, Adam Perer, Hanspeter Pfister, and Alexander M. Rush. Seq2Seq-Vis: A Visual Debugging Tool for Sequence-to-Sequence Models. *arXiv e-prints*, page arXiv:1804.09299, April 2018.

[SGPR16] Hendrik Strobelt, Sebastian Gehrmann, Hanspeter Pfister, and Alexander M. Rush. LSTMVis: A Tool for Visual Analysis of Hidden State Dynamics in Recurrent Neural Networks. *arXiv e-prints*, page arXiv:1606.07461, June 2016.

[SLRGVC16] José-Manuel Sáez-López, Marcos Román-González, and Esteban Vázquez-Cano. Visual programming languages integrated across the curriculum in elementary school: A two year case study using “Scratch” in five schools. *Computers & Education*, 97:129–141, jun 2016.

[Sta] Laerd Statistics. Independent t-test using spss statistics. Last accessed 01 June 2019.

[SW65] Samuel Sanford Shapiro and Martin B Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[Tan13] Steven L. Tanimoto. A perspective on the evolution of live programming. In *2013 1st International Workshop on Live Programming, LIVE 2013 - Proceedings*, pages 31–34. IEEE, may 2013.

[Tan15] Steven L. Tanimoto. Transparency and liveness in visual programming environments for novices. In *Proceedings - 2015 IEEE Blocks and Beyond Workshop, Blocks and Beyond 2015*, pages 113–114. IEEE, oct 2015.

[UCK$^+$10] Ian Utting, Stephen Cooper, Michael Kölling, John Maloney, and Mitchel Resnick. Alice, Greenfoot, and Scratch – A Discussion. *ACM Transactions on Computing Education*, 10(4):1–11, nov 2010.

[uni17] Make games-not tools. Technical report, 2017.

[weba] Deep cognition. Last accessed 08 February 2019.

[webb] Google for education - blockly. Last accessed 05 February 2019.

[webc] Sony's neural network console. Last accessed 08 February 2019.

[web10] nodemon, 2010. Last accessed 02 February 2019.

[WGYS18] Junpeng Wang, Liang Gou, Hao Yang, and Han-Wei Shen. Ganviz: A visual analytics approach to understand the adversarial game. *IEEE transactions on visualization and computer graphics*, 24(6):1905—1917, June 2018.

[Wow15] Izabela Wowczko. A Case Study of Evaluating Job Readiness with Data Mining Tools and CRISP-DM Methodology. *International Journal for Infonomics (IJI)*, 8(3):1066–1070, 2015.

[WSW$^+$18] K. Wongsuphasawat, D. Smilkov, J. Wexler, J. Wilson, D. Mané, D. Fritz, D. Krishnan, F. B. Viégas, and M. Wattenberg. Visualizing dataflow graphs of deep learning models in tensorflow. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):1–12, Jan 2018.

[YCN$^+$15] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding Neural Networks Through Deep Visualization. *arXiv e-prints*, page arXiv:1506.06579, June 2015.

[YS18] Rulei Yu and Lei Shi. A user-based taxonomy for deep learning visualization. *Visual Informatics*, 2(3):147 – 154, 2018.

[ZBM16] Tom Zahavy, Nir Ben Zrihem, and Shie Mannor. Graying the black box: Understanding DQNs. *arXiv e-prints*, page arXiv:1602.02658, February 2016.

[ZHP$^+$17] Haipeng Zeng, Hammad Haleem, Xavier Plantaz, Nan Cao, and Huamin Qu. CNNComparator: Comparative Analytics of Convolutional Neural Networks. *arXiv e-prints*, page arXiv:1710.05285, October 2017.

[Zur92] J. Zurada. *Introduction to Artificial Neural Systems*. West Publishing Co., St. Paul, MN, USA, 1992.

[ZXZ$^+$17] Wen Zhong, Cong Xie, Yuan Zhong, Yang Wang, Wei Xu, Shenghui Cheng, and Klaus Mueller. Evolutionary Visual Analysis of Deep Neural Networks. In *International Conference on Machine Learning (ICML)*, number August, 2017.

REFERENCES

Appendix A

Participation Consent Declaration

This appendix contains the declaration each participant was required to sign before performing the experiment. The main objective was to safely collect and use data from the participant without any legal repercussions.

DECLARAÇÃO DE CONSENTIMENTO

(Baseada na declaração de Helsínquia)

No âmbito da realização da tese de Mestre no Mestrado Integrado em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto, intitulada **A live IDE for deep learning architectures**, realizada pelo estudante **Mário Gustavo Fernandes**, orientada pelo Professor Hugo Sereno Ferreira e sob a coorientação do Professor André Restivo, eu abaixo assinado, ____, declaro que compreendi a explicação que me foi fornecida acerca do estudo que irei participar, nomeadamente o carácter voluntário dessa participação, tendo-me sido dada a oportunidade de fazer as perguntas que julguei necessárias.

Tomei conhecimento de que a informação ou explicação que me foi prestada versou os objetivos e os respetivos métodos a serem empregues, assim como que será assegurada a máxima confidencialidade dos meus dados.

Explicaram-me, ainda, que poderei abandonar o estudo em qualquer momento, sem que daí advenham quaisquer desvantagens.

Por isso, consinto participar no estudo e na recolha de imagens necessárias, respondendo a todas as questões propostas.

Porto, __ de ____________ de _____

____

(Participante ou seu representante)

Appendix B

Questionnaire used in the Quasi-Experiment

The following document constitutes the questionnaire that was expected to be filled by the participants of the quasi-experiment. Note that one post-task questionnaire refers to the Keras questionnaire and the other refers to the Editor questionnaire. Participants would have a version associated to them and according to it the order of these 2 questionnaires was taken into consideration.

Interactive Deep Learning Editor: Pre-Test Questionnaire

*Required

1. **Age** *

2. **Gender**
Mark only one oval.

◯ Male
◯ Female

3. **I consider myself an experienced python programmer** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	◯	◯	◯	◯	◯	Completly agree

4. **I understand basic concepts of machine learning such as models, classification and feature engineering** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	◯	◯	◯	◯	◯	Completly agree

5. **I am familiar with some concepts of deep learning such as neural networks, layers and hyperparameterization** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	◯	◯	◯	◯	◯	Completly agree

6. **When comparing Deep Learning (DL) to traditional Machine Learning methods, DL usually needs a lot more feature engineering, therefore both the algorithm and the results are easier to understand** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	⬭	⬭	⬭	⬭	⬭	Completly agree

7. **Convolutional neural networks (CNNs) can be used in image recognition and classification essentially by using filters to classify different, smaller parts of an image.** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	⬭	⬭	⬭	⬭	⬭	Completly agree

8. **I consider myself an experienced Keras user** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	⬭	⬭	⬭	⬭	⬭	Completly agree

9. **I have considerable experience with visual programming languages or tools such as Scratch or Node-RED** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	⬭	⬭	⬭	⬭	⬭	Completly agree

10. **I regularly use live development tools such as nodemon** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	⬭	⬭	⬭	⬭	⬭	Completly agree

Interactive Deep Learning Editor: First Test Questionnaire

11. **Creating architecture of deep learning models with a single input and a single output was easy to do** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	◯	◯	◯	◯	◯	Completly agree

12. **Creating architecture of deep learning models with multiple inputs and multiple outputs was easy to do** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	◯	◯	◯	◯	◯	Completly agree

13. **While doing the task I used official Keras documentation multiple times for answers to my problems** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	◯	◯	◯	◯	◯	Completly agree

14. **While doing the task I used other websites other than Keras.io multiple times for answers to my problems** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	◯	◯	◯	◯	◯	Completly agree

15. **While doing the task I made multiple mistakes and then corrected them** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	◯	◯	◯	◯	◯	Completly agree

16. **I felt that I have successfully completed all exercises** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	◯	◯	◯	◯	◯	Completly agree

Interactive Deep Learning Editor: Second Test

Questionnaire

17. **Creating architecture of deep learning models with a single input and a single output was easy to do** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	⬭	⬭	⬭	⬭	⬭	Completly agree

18. **Creating architecture of deep learning models with multiple inputs and multiple outputs was easy to do** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	⬭	⬭	⬭	⬭	⬭	Completly agree

19. **While doing the task I used official Keras documentation multiple times for answers to my problems** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	⬭	⬭	⬭	⬭	⬭	Completly agree

20. **While doing the task I used other websites other than Keras.io multiple times for answers to my problems** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	⬭	⬭	⬭	⬭	⬭	Completly agree

21. **While doing the task I made multiple mistakes and then corrected them** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	⬭	⬭	⬭	⬭	⬭	Completly agree

22. **I felt that I have successfully completed all exercises** *
Mark only one oval.

	1	2	3	4	5	
Completly disagree	⬭	⬭	⬭	⬭	⬭	Completly agree

Interactive Deep Learning Editor: Post-Tests Questionnaire

23. **Overall, the visual editor was easier to use**
Mark only one oval.

	1	2	3	4	5	
Completly disagree	⬭	⬭	⬭	⬭	⬭	Completly agree

24. **I felt that I understood better what was going on when I used the visual editor**
Mark only one oval.

	1	2	3	4	5	
Completly disagree	⬭	⬭	⬭	⬭	⬭	Completly agree

25. **While using the visual editor I noticed that some default values were set so I did not need to change them, whereas while using Keras I set them either way.**
Mark only one oval.

	1	2	3	4	5	
Completly disagree	⬭	⬭	⬭	⬭	⬭	Completly agree

26. **While using the visual editor I made good use of the saving and loading feature and felt that they helped me**
Mark only one oval.

	1	2	3	4	5	
Completly disagree	⬭	⬭	⬭	⬭	⬭	Completly agree

27. **While using the visual editor I made good use of the live mode feature and felt that it helped me**
Mark only one oval.

	1	2	3	4	5	
Completly disagree	⬭	⬭	⬭	⬭	⬭	Completly agree

28. **While using the visual editor I encountered some errors while creating a model but I easily corrected them**
Mark only one oval.

	1	2	3	4	5	
Completly disagree	◯	◯	◯	◯	◯	Completly agree

29. **While using Keras I encountered some errors while creating a model but I easily corrected them**
Mark only one oval.

	1	2	3	4	5	
Completly disagree	◯	◯	◯	◯	◯	Completly agree

30. **If I wished to partake in a deep learning (DL) project I would use this visual editor as is to create DL architectures, taking into account that it can support DL libraries other than Keras**
Mark only one oval.

	1	2	3	4	5	
Completly disagree	◯	◯	◯	◯	◯	Completly agree

31. **If I wished to partake in a deep learning (DL) project I would use a more polished version of this visual editor to create DL architectures, taking into account that it can support DL libraries other than Keras**
Mark only one oval.

	1	2	3	4	5	
Completly disagree	◯	◯	◯	◯	◯	Completly agree

32. **If I wished to partake in a deep learning (DL) project I would use a visual editor that would not only aid me in the creation of a DL architecture but also give me the ability to feed it data, train it and analyze the output**
Mark only one oval.

	1	2	3	4	5	
Completly disagree	◯	◯	◯	◯	◯	Completly agree

33. **Comments or suggestions**

Powered by Google Forms

Questionnaire used in the Quasi-Experiment