POLITECNICO DI TORINO
Repository ISTITUZIONALE

Towards A Visual Programming Tool to Create Deep Learning Models

Original
Towards A Visual Programming Tool to Create Deep Learning Models / Calò, T., De Russis, L.. - ELETTRONICO. - (2023), pp. 38-44. (ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '23) Swansea (UK) June 27–30, 2023) [10.1145/3596454.3597181].

Availability:
This version is available at: 11583/2978363 since: 2025-08-04T15:57:46Z

Publisher:
Association for Computing Machinery

Published
DOI:10.1145/3596454.3597181

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
ACM postprint/Author's Accepted Manuscript

(Article begins on next page)

28 November 2026

Towards A Visual Programming Tool to Create Deep Learning Models

Tommaso Calò
Politecnico di Torino
Torino, Italy
tommaso.calo@polito.it

Luigi De Russis
Politecnico di Torino
Torino, Italy
luigi.derussis@polito.it

ABSTRACT

Deep Learning (DL) developers come from different backgrounds, e.g., medicine, genomics, finance, and computer science. To create a DL model, they must learn and use high-level programming languages (e.g., Python), thus needing to handle related setups and solve programming errors. This paper presents DeepBlocks, a visual programming tool that allows DL developers to design, train, and evaluate models without relying on specific programming languages. DeepBlocks works by building on the typical model structure: a sequence of learnable functions whose arrangement defines the specific characteristics of the model. We derived DeepBlocks' design goals from a 5-participants formative interview, and we validated the first implementation of the tool through a typical use case. Results are promising and show that developers could visually design complex DL architectures.

CCS CONCEPTS

• **Human-centered computing** → **Graphical user interfaces**; Empirical studies in HCI; • **Computing methodologies** → *Machine learning*; • **Software and its engineering** → **Application specific development environments**.

KEYWORDS

deep learning, visual programming, debugging, user interface

ACM Reference Format:
Tommaso Calò and Luigi De Russis. 2023. Towards A Visual Programming Tool to Create Deep Learning Models. In *Companion of the 2023 ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '23 Companion), June 27–30, 2023, Swansea, Wales, UK*. ACM, New York, NY, USA, 7 pages.

1 INTRODUCTION AND BACKGROUND

Recently, deep learning (DL) experienced rapid progress and achieved competitive performance in numerous areas such as image recognition, natural language processing, autonomous driving, medical diagnosis, and drug discovery. As such, DL developers can choose between many popular frameworks for development, including Tensorflow [1], Keras [4], and PyTorch [7]. These frameworks, however, require developers to have a certain level of programming skills, which many developers from diverse domains (e.g., medicine, genomics, or finance) need to master before being able to create and evaluate a DL model. Thus, many challenges arise [10] as developers experience this steep learning curve. In 2017, Sankaran et al. [10] studied the challenges that DL developers face through a quantitative survey among 113 software engineers and researchers from various backgrounds and experiences. The authors showed that DL frameworks exhibit a lack of needed features for quicker and more efficient implementation and prototyping. As a solution, 89% suggested the need for a system able to suggest hyper-parameters and assist in debugging the DL model, while 72% of the respondents suggested that a **visual programming tool** would be useful to speed up the overall development process.

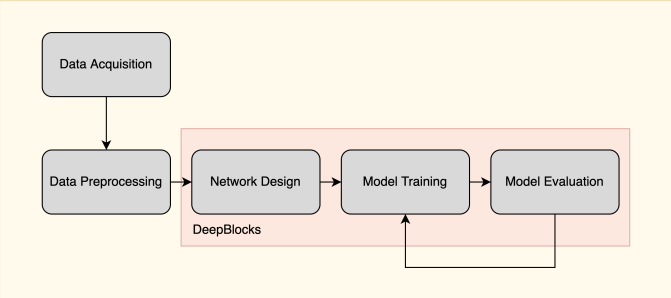

Figure 1: The typical deep-learning development workflow, the highlighted part indicates the processes covered by DeepBlocks.

In traditional software development, visual programming is a paradigm that lets users create programs by manipulating elements graphically rather than by specifying them textually. DARVIZ [10], DL-IDE [12], DeepVisual [13], and ModelTracker [2] were the first attempts to introduce visual programming IDE enabling "no-code" intuitive way of designing deep learning models. They, however, exhibit limitations that do not allow the design of complex and scalable models, such as the impossibility of merging, connecting, and reusing layers and the impossibility of customizing the training procedure. In addition, they do not include important features for the complete development process, such as debugging features. Such limitations must be overcome to build larger and more complex networks, which can accomplish the recent design requirements that emerged in the community [3]. UMLAUT [11] is, instead, an example of a tool targeted to non-expert developers that focuses on debugging.

Neural Network Console developed by Sony [6] is the only available web application that overcomes most of the above limitations. However, the DL library it supports has quite a limited user scope,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
EICS '23 Companion, June 27–30, 2023, Swansea, Wales, UK
© 2023 Association for Computing Machinery.

with less than 1% usage over the DL community [12]. Moreover, it is available in the cloud, only, thus limiting the possibility of using in-house machines and introducing several privacy concerns.

To address the limitations of the available visual programming tools, and to reduce the gap between visual tools' capabilities and the freedom of expression of pure coding, we propose *DeepBlocks*, a visual programming tool for DL that integrates training, debugging, and evaluation of neural network models under a single user interface. In this way, the tool covers the main steps of the DL development workflow (Fig. 1). DeepBlocks allows DL programmers to design neural networks by adding, connecting, and merging layers, which we refer to as "blocks", to create more complex layers and architectures. In addition, DeepBlocks allows users to create personalized blocks and add custom functions to easily adapt the tool to their application domain. DeepBlocks also allows developers to schedule and process multiple inputs and to design networks with multiple branches, being this a novel feature the existing tools, which only allows building layers with a single forward connection. DeepBlocks uses PyThorch, which is a DL library widely used in the DL community, adopted by up to 75% of the papers available in the literature.

Starting from a formative interview with 3 Machine Learning engineers and 2 Ph.D. students in the field of deep learning, we obtained nine crucial functionalities to be implemented in DeepBlocks, to make it useful, efficient, and versatile. We then present the design and implementation of the tool, and report a use case to validate it. Finally, we discuss possible advantages and limitations and conclude with future work.

2 FORMATIVE INTERVIEWS AND DESIGN GOALS

We conducted five semi-structured interviews, online and in-person, to five DL developers in October 2022. In the interviews, we focused on three main questions: 1) how they perform the design of deep learning architectures; 2) the main difficulties they face throughout the process; and 3) we discussed the possible advantages and disadvantages of the adoption of a visual programming interface to develop deep learning architectures.

We interviewed three data scientists who frequently develop deep learning models at a large, data-driven software company, and two artificial intelligence Ph.D. students, who both develop and apply deep learning models in their research field. 3 participants (P1–P5) self-identified as male and 2 self-identified as females, their age was between 25 and 29 years old, and they signed a consent form before starting the interview. We synthesized the results of the interviews into nine design goals, which guided the design of DeepBlocks. In a subsequent phase, participants were asked to rank the goals by importance and comment on their decisions.

2.1 Results and Design Goals

For P1, P2, and P4 the main difficulty in programming deep neural networks is to understand from the code how the network is structured; they argue that the code structure, in a large number of cases, does not reflect the structure of the network, leading to inefficient design.

For P3 and P5 the most important issue is to locate bugs in the architecture, along with the fact that they can only spot them at the start of the training phase. From this insight, we set our first design goal to be *Interactive Debugging*. Furthermore, P1 and P3 point out that it takes a lot of effort to monitor the inputs and outputs for every layer of the network during training and that simplifying such procedure would be useful to better understand the behavior of the model. We summarize this finding in a second design goal: *Visualization of blocks inputs and outputs*. Where we refer as a 'block' to an elementary architectural layer.

P4 and P5 add that the reuse of layers between different architectures is particularly prohibitive due to a lack of compatibility and difficulties in separating the individual layers from their context. From this insight, we obtain the third design goal: *Load existing and custom blocks*.

P1, P2, P4, and P5 are concerned about the possibility that a visual tool would be actually able to convey the same freedom of design that coding does; on the other hand, all participants agree that visual programming could be particularly suitable for the deep learning domain, due to the repetitiveness of layers that composes the architectures and the limited procedures schemes to train a network. Furthermore, P2, P4, and P5 argue that a simple and intuitive user interface is preferable over a more sophisticated one, even at the cost of implementing fewer customization options; this insight led us to derive a fourth design goal: *Simple and Intuitive Interface*. In addition, for P1, P2 and P5, in a visual programming interface default blocks should be organized in structured menus; from this observation, we obtain our fifth goal: *Blocks Organized in structured menus*. P1, P3, and P4 pointed out that such a visual tool should be capable of designing the same architectures that can be programmed with code, both in terms of complexity and scalability; from this point, we derive the sixth, seventh, and eighth design goals: *Hierarchical Aggregation of blocks*, *Customizable blocks* and *Personalize optimization strategy*. P3 and P4 evidenced that not only such a visual tool should be capable of designing the same architectures that can be programmed with code but also it should be able to do the same kind of model evaluation; this led us to set the ninth design goal: *Visualization for model evaluation*.

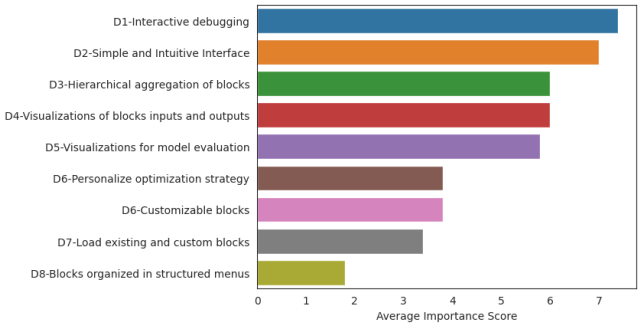

Figure 2: The design goals we derived from formative interviews, ordered by the average importance given by every participant.

In a second phase, participants were asked to rank the resulting design goals, with a score from 1 (lowest importance) to 9 (highest

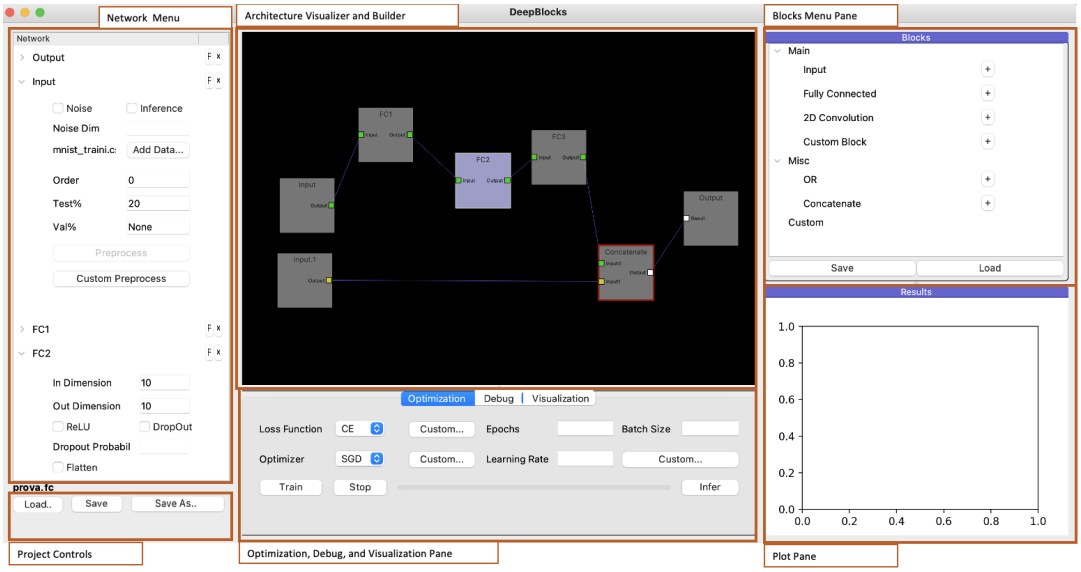

Figure 3: DeepBlocks Layout

importance) in order for us to focus our efforts on the most relevant design features. The results shown in Figure 2 reflect the findings of the formative interviews, where participants agree to the need for a simple interface to visually program the architectures and a proper interactive debugging procedure that can easily notify users about the location and the type of bug.

3 DEEPBLOCKS

DeepBlocks is a visual programming tool intended for Deep Learning engineers that need to develop and test complex deep learning architectures. We designed DeepBlocks to accomplish the needs extracted from the formative interviews. In this section, we list the main features of DeepBlocks and describe its design and implementation.

3.1 An Example Scenario

We introduce an example scenario that will be developed throughout all Section 3 to illustrate to the reader how the implemented functionalities can be used by a programmer to achieve the design of a simple DL architecture.

> *John is a Computer Engineering student, he is in his third year of B.S., and he is following the Artificial Intelligence course. In the second assignment, he has been charged with developing, to classify an image dataset, a simple neural network composed of five Fully Connected Layers, using DeepBlocks.*

3.2 Layout of DeepBlocks

> *John downloads DeepBlocks installs its dependencies and launches the program. To design the network, he adds an input block and five fully connected blocks by clicking the "+" button on the respective voice in the right panel. By clicking on the "Add Data…" button in the "Input" voice on the tree menu in the left panel, John can select the dataset file from a dialog. John is not required to custom preprocess the dataset, since it is already in the format specified in the tool's documentation.*

The layout of DeepBlocks is illustrated in Fig. 3. DeepBlocks consists of six main panes: Network Menu, The Architecture Visualizer and Builder, the Blocks Menu Pane, the Project Controls, the Results Visualization Pane, and the Optimization, Debug, and Visualization Pane. The Network Menu, located on the upper left, lists the current blocks present in the architectures and allows access to their specific controls and parameters. The Architecture Visualizer and Builder is the core of the visual programming capabilities of DeepBlocks; it visualizes the blocks and allows the user to connect the input and outputs terminal of different blocks. By right-clicking on a specific block, users can save it under the "custom" menu in the Blocks Pane, or, when multiple blocks are selected, users can merge them into an abstract "SuperBlock". In addition, the Architecture Visualizer and Builder provides debug tips: when a block is not correctly processed, its contours are colored red, and when a signal is not processed, due to an error in the block, the input and output terminals are colored yellow as shown in Fig. 3. The Blocks Pane lists the default available blocks, which are subdivided into main blocks, which are the most typical deep learning processing layers, as well as miscellaneous blocks, such as the one to concatenate data or to do a logical OR between two inputs. In addition, the Blocks Pane contains the controls that allow to saving and load one or multiple custom blocks. The Optimization, Debug, and Visualization Pane include the optimization controls to train and test the network, the debug information of the selected block, and a visualization section to visualize the inputs and outputs of every block. In the Results Visualization Pane are plotted the train and

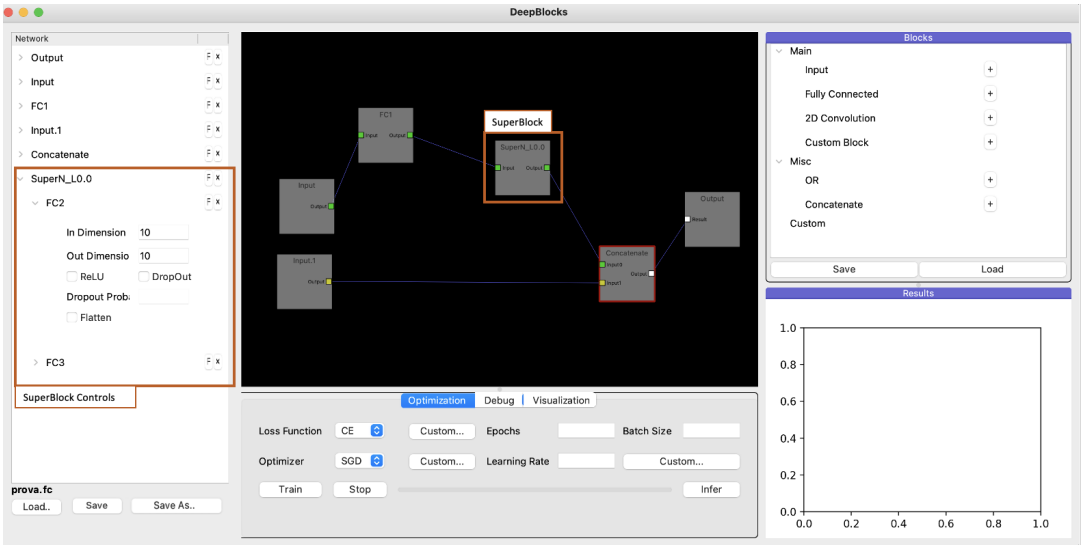

Figure 4: Illustration of a SuperBlock and its hierarchical visualization in the Network Menu

test accuracy. Finally, the Project Controls Pane, allows you to save or load a project.

3.3 Visual Programming in DeepBlocks

John connects the blocks he previously added starting from the Input block, and sequentially through every Fully Connected Block until the output block. He then sets the right input and output dimension for every Block, checking the output dimension of every Block in the Visualization pane. John then merges the five fully connected Blocks into a SuperBlock and renames it "Backbone". He then saves the "Backbone" block in the custom blocks tree, by right-clicking over it and selecting the "Save" option.

With respect to the literature, DeepBlocks provides a fully scalable way to visually design DL architectures and the possibility to design complex, multi-branch architectures. Large practical cases could be modeled in deep blocks thanks to the possibility of adding custom blocks, merging multiple blocks into hierarchical "SuperBlocks" as well as the possibility of scheduling multiple inputs and creating multi-branch connections.

A Block is composed of one or multiple input and output terminals. Every Block contains a Python function that characterizes it; users can add custom blocks by adding "Custom Block" from the Blocks Pane and specifying its custom function, or can directly load existing custom blocks; the latter can be useful for non-expert users, which when reusing a custom made block could only focus on its input and outputs and not on the underlying logic. The blocks specific properties can be set in the Network Menu (see Fig. 4).

To scale up the designed architecture, selected blocks can be merged into more abstract "SuperBlocks" by invoking the right-click menu over the Architecture Visualizer and Builder and selecting the "Merge" option. SuperBlocks sub-blocks can be hierarchically visualized, along with their controls, in the Network Menu. To train or execute the architecture, a computational tree is generated, starting from input blocks and recurrently through every connection; if two branches converge on the same block, they are guaranteed to be processed sequentially before the successive computations. Cycles are not allowed in our setting. In order to expand the capabilities of the training procedure, we introduced a notion of "order" for every input. Inputs can be assigned to one or multiple orders. At every training step, orders are executed sequentially, and, for each order, only the signals coming from input blocks that belong to the specified order are passed downstream, while for the others is passed a null value. This allows the designing of many practical architectures that require alternation of different input signals. An input signal is a dictionary composed of the input value, the ground truth, the current order, and a flag indicating if the current is a test or a training step. By accessing the dictionary, the different custom functions can be programmed depending on the order currently executing. We believe that the above-proposed features, which are mostly missing in the literature, can make a step forward in allowing developers to design with our visual programming tool most of the variety of the typical architectural designs.

3.4 Debug Features and Validation

John notices that the contours of the "Backbone" SuperBlock are painted red; he inspects the debug pane to check what error is causing the block not to process, and understands that there is a problem in the input dimensions since the image must be flattened before passing it to the Fully Connected Block. He flattens the input of the first fully connected block on the left pane, and the red contour disappears. He then sets up the optimization parameters, and finally trains the network. He checks the training meters by looking at the Train Metrics Plot on the right.

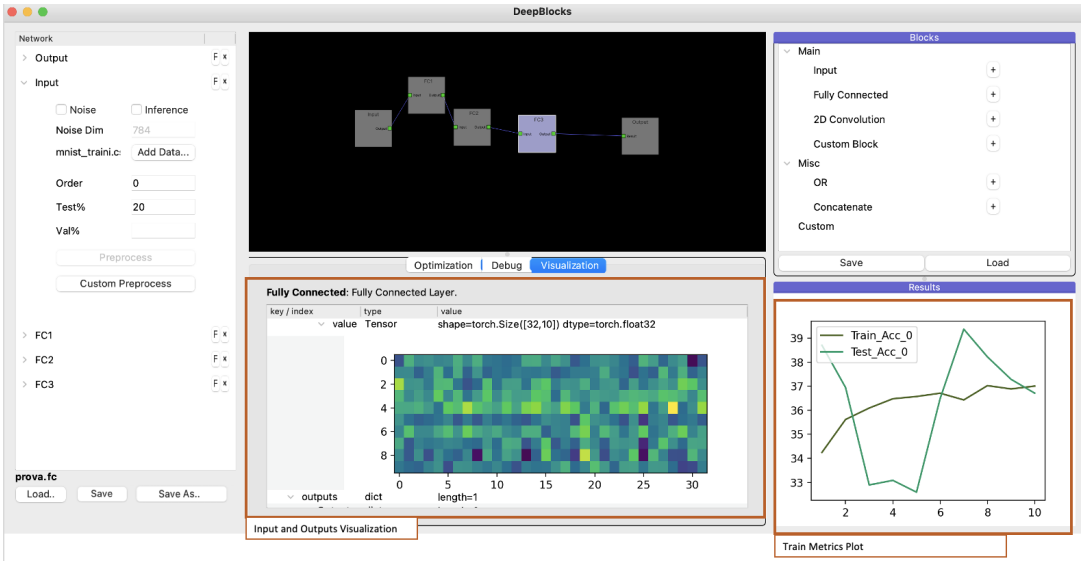

Figure 5: Visualization Features of DeepBlocks.

Although the model evaluation is not the main focus of DeepBlocks, which is rather more engineered on the architecture design, we added two main features to monitor the model results: the Visualization and the Train Metric Plot panes (Fig. 5). The former reports, for the selected block, the input and output dimensions, as well as an heatmap of their values. The latter shows training reports evaluation metrics over the training and test data.

In addition, as in John's story, a Debug Pane is available, showing the type and dimension of the various Blocks' inputs and outputs.

3.5 Implementation Details

DeepBlocks has been implemented in Python 3.8 using PyTorch [7] for Deep Learning modeling, PyQT5 [8] for the user interface design, and PyQTgraph [9] for the visual programming features.

4 USE CASE: DOMAIN-ADVERSARIAL NEURAL NETWORK

This section demonstrates the applicability of DeepBlocks in a typical use cases: how to visually design a domain-adversarial neural network (DANN) [5]. DANN takes as inputs labeled samples from a source distribution and unlabelled samples from a target distribution and it learns how to extract the features to solve the task for both the source and target domains.

We start by adding two Inputs Blocks and load on the first the source and the second the target domain data. We then concatenate the outputs. To design the feature extractor, we add three Convolutional Blocks to the Architecture Visualization and Builder Pane, and connect them; we then select them and merge them into a single SuperBlock that we rename "Feature Extractor". From the miscellaneous blocks, we add the Copy Block that simply copies the input to one or more outputs, and we connect it to the outputs of the Feature Extractor. To design the label classifier, we add three Fully Connected Blocks, connect them and merge them as done with the Feature Extractor. We repeat the process for the Domain Classifier. To reverse the gradients of the Domain classifier, we add a Custom Block where we simply replace the predefined "backward" function returning its negative. Fig. 6 reports the resulting architecture along with the one in the reference paper [5].

5 CONCLUSIONS AND FUTURE WORK

In this paper, we introduced DeepBlocks, a visual programming tool for deep learning software development. DeepBlocks allows developers to design and implement DL architectures visually. The tool provides several development features including model designing from scratch, interactive debugging, model training, and model inference. In addition, with respect to the previously available tools, DeepBlocks allows the designing of more complex, scalable, and custom architectures.

We designed DeepBlocks with the support of a formative interview with 5 participants, and we preliminary validated it through a use case. With the use case, in particular, we showed that just allowing a little customization of blocks, we can permit developers to visually design complex and experimental architectures. Clearly, there is a trade-off between customization capabilities and the actual automation that DeepBlocks provides in the process of DL programming. Allowing customization without losing automation is design-challenging, but it could augment the complexity of the tool; for example, when adding the notion of "order" — to let users customize the training procedure — we require the users to specify it for every training procedure. The right balance should be found between customization capabilities, complexity, and automation of the tool.

As future work, DeepBlocks can tame the problem of visualizing a large number of inputs and outputs (in the order of billion). Currently, there is also no way in the tool to understand the behavior of sub-blocks that composes a SuperBlock. This applies to debug as well; in fact, from a faulty SuperBlock you cannot visually locate the

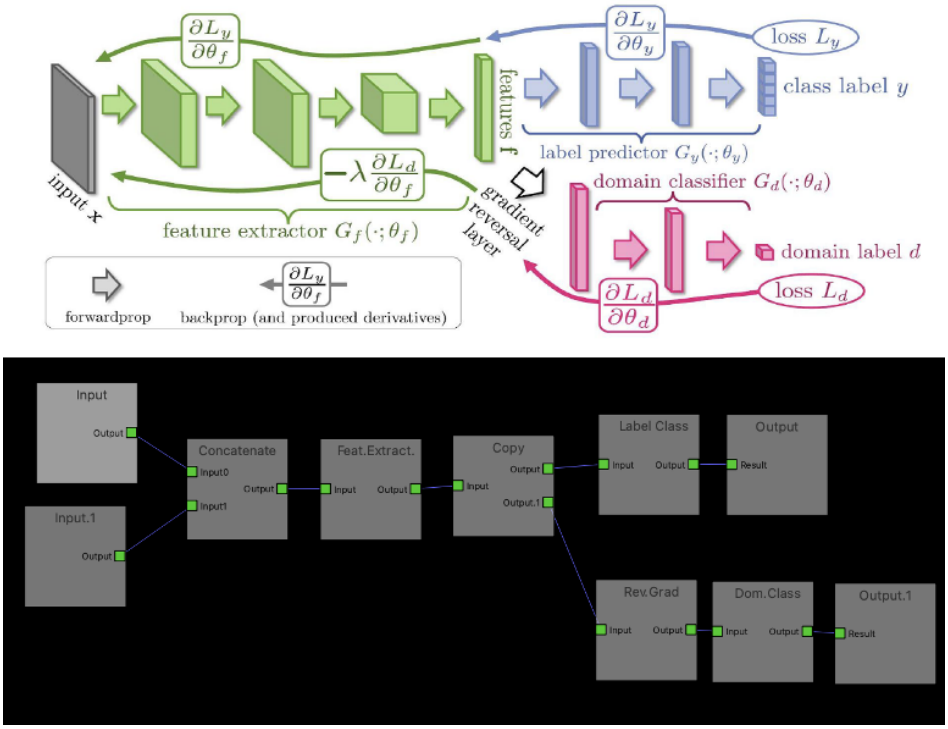

Figure 6: Domain-Adversarial Neural Network as described in [5] (above) and visualized in DeepBlocks (below).

bug in the sub-blocks without expanding it in the Debug Pane, and this becomes unfeasible with a very large number of hierarchies.

Among the next steps, DeepBlocks needs to undergo a series of user studies, involving its usability and effectiveness against similar tools and traditional programming approaches. Finally, once the tool is consolidated, we plan to release it and further evaluate the tool in a large-scale, in-the-wild study, e.g., with machine learning students.

6 ACKNOWLEDGEMENTS

This study was carried out within the FAIR - Future Artificial Intelligence Research and received funding from the European Union Next-Generation EU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) – MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.3 – D.D. 1555 11/10/2022, PE00000013). This manuscript reflects only the authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. USENIX Association, USA, 265–283.

[2] Saleema Amershi, Max Chickering, Steven M. Drucker, Bongshin Lee, Patrice Simard, and Jina Suh. 2015. ModelTracker: Redesigning Performance Analysis Tools for Machine Learning. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea). Association for Computing Machinery, New York, NY, USA, 337–346. https://doi.org/10.1145/2702123.2702509

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[4] François Chollet et al. 2018. Keras: The python deep learning library. *Astrophysics source code library* (2018), ascl–1806.

[5] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-Adversarial Training of Neural Networks. *J. Mach. Learn. Res.* 17, 1 (jan 2016), 2096–2030.

[6] Yoshiyuki Kobayashi. 2020. Neural Network Console that Accelerates the Use of Deep Learning. *Medical Imaging and Information Sciences* 37, 1 (2020), 1–4. https://doi.org/10.11318/mii.37.1

[7] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. 2017. Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration. *PyTorch: Tensors and dynamic neural networks in Python with strong GPU acceleration* 6, 3 (2017), 67.

[8] PyQT [n. d.]. https://riverbankcomputing.com/software/pyqt/.

[9] PyQTgraph [n. d.]. https://pyqtgraph.readthedocs.io/.

[10] Anush Sankaran, Rahul Aralikatte, Senthil Mani, Shreya Khare, Naveen Panwar, and Neelamadhav Gantayat. 2017. DARVIZ: Deep Abstract Representation, Visualization, and Verification of Deep Learning Models. In *2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER)*. 47–50. https://doi.org/10.1109/ICSE-NIER.2017.13

[11] Eldon Schoop, Forrest Huang, and Bjoern Hartmann. 2021. UMLAUT: Debugging Deep Learning Programs Using Program Structure and Model Behavior. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*

(Yokohama, Japan). Association for Computing Machinery, New York, NY, USA, Article 310, 16 pages. https://doi.org/10.1145/3411764.3445538

[12] Srikanth G Tamilselvam, Naveen Panwar, Shreya Khare, Rahul Aralikatte, Anush Sankaran, and Senthil Mani. 2019. A Visual Programming Paradigm for Abstract Deep Learning Model Development. In *Proceedings of the 10th Indian Conference on Human-Computer Interaction* (Hyderabad, India) *(IndiaHCI '19)*. Association for Computing Machinery, New York, NY, USA, Article 16, 11 pages. https://doi.org/10.1145/3364183.3364202

[13] Chao Xie, Hua Qi, Lei Ma, and Jianjun Zhao. 2019. DeepVisual: A Visual Programming Tool for Deep Learning Systems. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. 130–134. https://doi.org/10.1109/ICPC.2019.00028