

VISUALIZATION-BASED NEURAL NETWORK INTROSPECTION

ALEX BÄUERLE

VISUALIZATION-BASED
NEURAL NETWORK INTROSPECTION

ALEX BÄUERLE
FROM FRIEDRICHSHAFEN

A cumulative thesis submitted to attain the degree of Dr. rer. nat.
of the Faculty of Engineering, Computer Sciences and Psychology
at Ulm University

July, 2022

Alex Bäuerle: *Visualization-based
Neural Network Introspection* © July, 2022
This work is licensed under the Creative Commons **Attribution-
NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**
License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/4.0/>.
Typeset: PDF-L^AT_EX 2 _{ε}

Acting Dean:
Prof. Dr. Anke Huckauf

Referees:
Prof. Dr. Timo Ropinski
Prof. Dr. Martin Wattenberg

Date of Defense:
December 21, 2022

ABSTRACT

Artificial intelligence (AI) and the use of neural networks have become omnipresent in recent years. Neural networks model complex mathematical functions that can be based on billions, or even trillions, of parameters. At the same time, neural networks make decisions that can deeply impact our lives. Therefore, understanding, testing, and interpreting these networks and their decisions is an integral part of model development and deployment. While there exist introspection techniques that support such understanding, testing, and interpretation, their adoption for diagnosing systems and explaining decisions can be difficult. Current problems with the adoption of introspection techniques are that they are not easily implemented in one's framework, do not work well in combination to create more meaningful analyses, and are difficult to interpret.

Through the integration of existing and novel introspection techniques into visualization interfaces, extensive analysis, actionable insights, and effective diagnosis are made widely available. These visualization interfaces can be incorporated into existing development workflows and are designed to support bespoke analysis needs, which makes the interpretation of findings easier. In this thesis, we present published visualization interfaces in three different areas, namely quality assurance, communication, and AI education. These publications include a visualization approach for correcting mislabeled training data, an interface for automatic network figure generation to communicate network architectures, and an educational environment for recurrent neural networks (RNNs). Finally, to unify the diverse landscape of AI visualization interfaces, we also present a framework for composing, reusing, exploring, and sharing such interactive machine learning (ML) interfaces.

Working on this thesis was a learning endeavor, both scientifically and personally. I want to express my gratitude to everyone who supported me throughout this time, especially:

My supervisor, **Timo**, for believing in me when I had no clue about a field of research that I now think about every day and allowing me to work on what I found most interesting once I was ready for it.

All the **MI-colleagues** who were always supportive during work times and became great friends outside of work.

My **colleagues and mentors** from my internships at Google and Apple. These times were the most demanding, but also the most exciting and formative phases of my PhD.

All my **friends** who were there when I needed a break from work. Without you, I don't believe I could have completed this task while staying sane.

Most importantly, my parents, **Paul** and **Sigi**, for making sure I had every opportunity and giving me the freedom to do what interested me the most, as well as my brother **Kai** and my partner **Alina**, who were incredibly supportive.

CONTENTS

I Thesis

1	Introduction	3
1.1	Introspection Techniques	3
1.2	ML Visualization Interfaces	5
1.2.1	Quality Assurance	5
1.2.2	Communication	7
1.2.3	Education	7
1.2.4	Unification	9
1.3	Structure of this Work	9
1.4	Publications included in this Work	10
1.5	Other Publications	10
2	Contributions	13
2.1	Classifier-Guided Visual Error Correction	14
2.2	Net2Vis	18
2.3	exploRNN	23
2.4	Symphony	27
3	Conclusion	31
3.1	Future Work	31
3.2	Summary of Contributions	33
	Bibliography	35

II Publications

Part I
THESIS

INTRODUCTION

ML systems, especially deep learning (DL) algorithms, are becoming omnipresent and nowadays influence many parts of our everyday lives. DL has already been integrated into various domains such as navigation [Chi+19; Wor+19; RC21], recommender systems [PAC18; Rag20], medicine [RDK19; WLW20], and many more [JM15]. However, as is often the case, with great power comes great responsibility, and ML systems have been shown to be discriminative, error-prone, and unreliable [BG18; Sno18; AH+18; SC18; WHM19; Obe+19]. As a result, practitioners [DVK17; JIV19; DJL21; Sav22] as well as policy-makers [OEC19; AIEC20] call for explanations and control provided through ML introspection techniques.

1.1 INTROSPECTION TECHNNIQUES

To facilitate the need for analysis tools, myriad introspection techniques to investigate DL models or data sets have been proposed by the research community [MCB20; MV20]. In this work, we use introspection techniques as an umbrella term for explainability techniques and other approaches designed to foster an understanding of the problems, limitations, and capabilities of ML models or data sets, e.g., data set analysis methods or model architecture descriptions. As such, introspection techniques can be loosely defined as tools that help human observers understand elements of an otherwise opaque DL pipeline.

Before going into what introspection techniques are used for, we first need to define what the term introspection techniques does encompass and, perhaps more importantly, what it does not include. The terms interpretability and explainability are often used interchangeably in the context of DL. Multiple definitions for both terms can be found online and in research papers. In this work, we adopt the definition of Rudin [Rud19], who defines interpretable models as those that are inherently interpretable, whereas explainability only includes post-hoc explanation methods for black-box models. Thus, interpretability generally describes models that can be interpreted without auxiliary explanation, whereas black-box models require introspection techniques to be understood. As such, we do not consider interpretable models in this thesis but focus instead on the introspection techniques and visualization interfaces needed to facilitate explainability.

One of the most well-known explainability techniques is attribution, where activations of neural network units are attributed to input

features. Attribution methods can be either perturbation-based, e.g., [ZF14; FV17], or backpropagation-based, e.g., [ZTF11; STY17]. Another well-researched explainability technique is feature visualization, where the activation of a neuron is maximized by changing the input image, often through gradient descent [Erh+09]. Feature visualization can be used to reason about the features that individual neurons or combinations of neurons, such as channels or layers inside a neural network, have learned. There are also several data-based explainability methods, the most prominent of which might be counterfactual explanations [WMR17]. For a more complete overview of explainability methods for DL, the *field guide* published by Ras and Xie et al. serves as a good starting point [Ras+22]. In our work, we use explainability techniques as a technical foundation of the presented visualization interfaces.

In addition to explainability, our definition of introspection also includes the numerous tools that are not targeted directly at explaining the decisions of ML models or what they have learned. The most common techniques here might be analysis techniques related to training data sets. In this context, labeling approaches might be evaluated [KSA11; Lea11], or training data set fairness tested [Aka+21]. Another example for introspection techniques aside from explainability is documentation methods [Arn+19; BF18]. The most prominent of these documentation methods is Datasheets for Datasets [Geb+21]. In this work, Gebru et al. apply the idea of datasheets in electrical engineering to the domain of DL. Their Datasheets for Datasets describe important attributes of a data set, e.g., overview statistics, collection methods, and intended uses. A similar approach but for model reporting has been proposed by Mitchell et al. [Mit+19]. Similar to explainability, such introspection techniques are used in our work as a technical foundation for visualization interfaces.

Now that the scope of introspection methods has been clarified, we also need to specify the rationale behind the development of introspection techniques. In their review, Marcinkevičs et al. list trust, causality, reliability/robustness/transferability, fairness, and privacy as potential goals of explainability methods [MV20]. Additionally, we have seen that documentation methods can be important for reusability [Mit+19; Geb+21]. This list describes some of the most common objectives of introspection techniques but does not claim to be exhaustive. Furthermore, these objectives typically vary depending on the application area and target users of introspection techniques [Hoh+18]. As such, introspection techniques can be, e.g., roughly assigned to the spectrum between developer-focus and end-user-focus [Ras+22].

As mentioned before, we use introspection techniques as a technical foundation for our work. To this end, our work also includes novel introspection approaches. In this thesis, we mainly focus on data

collectors, developers, researchers, and decision-makers in the ML pipeline, but not end-users as our target audience. More importantly, our work is focused on how introspection techniques can be made more accessible through visualization. The following section explains the role of visualization interfaces in the context of DL introspection and provides an overview of the work in this area.

1.2 ML VISUALIZATION INTERFACES

While the necessity of introspection techniques for analyzing ML models and data sets is apparent [DVK17; Sav22], their existence alone is not sufficient. Introspection techniques need to be understandable for their target audience and usable in their specific problem context. If not designed and presented adequately, introspection techniques can fail their users [För+20] or even have adverse effects [Rud19; Sto+22]. We have found that, despite their existence, many introspection techniques are rarely used and shared between collaborators working on ML systems, as it can be hard to integrate them into their workflow and problem context [Bäu+22c].

One way to counteract this lack of broad adoption of introspection techniques is to improve their presentation, usability, and integratability [Wan+22]. Visualization is one way to achieve better usability and understandability of introspection methods [Yan+20; Str+21; Mes+22]. Data visualization transforms abstract data into meaningful representations, thus fostering knowledge communication and discovery [Hoh+18]. Although visualization research is relatively new to the domain of ML, it has already had a remarkable impact on the domain. In their interrogative survey, Hohman et al. refer to the *short yet impactful history* of visual analytics in the domain of ML [Hoh+18] while providing an overview of the visualization work in the domain of DL.

Despite not being the first to apply visualization to DL, Zeiler and Fergus [ZF14] are often considered to have popularized the use of visualization for neural network introspection. They presented an approach to project learned features back to input images through deconvolution. Since then, countless papers at the intersection of visualization and DL have been published. In the following, we will discuss recent publications in the three areas that our contributions can be attributed to, namely quality assurance, communication, and education.

1.2.1 Quality Assurance

When the predictions that DL systems make have a large impact on our lives, it is just natural that practitioners, affected end users, and independent organizations ask for quality guarantees [Ham+20;

[AIEC20](#)]. To investigate the quality of DL systems, visualizations that support human overview are often employed [[Mes+22](#)].

Those focusing on neural network prediction quality assurance are amongst the most popular of such visualization interfaces. With Boxer, model predictions can be interactively compared [[Gle+20](#)]. This comparison supports model diagnosis and selection. Combining this comparative visualization approach with set algebra further enables subgroup analysis. There even exists a testing framework, Vatun, for CNNs [[Par+21](#)]. Those tests can reveal differences between models and surface reasons for neural network predictions. Similarly, the what-if tool [[Wex+20](#)] and the language interpretability tool [[Ten+20](#)] enable probing into an ML model through what-if-based analysis.

There are also visualization interfaces for the detection and mitigation of the fairness issues that many ML models have. With FairVis, predictions for intersectional subgroups can be compared as a means to evaluate model fairness [[Cab+19](#)]. In a later publication, Wang et al. introduce DistriLens [[Wan+20](#)]. They propose improved means by which subsets of data that are worth investigating can be found. During the time of working on this thesis, we have also worked on a visualization interface for bias detection [[Bäu+22a](#)]. In our work, which is not included as one of the main contributions of this thesis, fairness for large label spaces can be visually investigated. As bias detection can be a labor-intensive process, modern approaches also use crowdsourcing to reliably discover and scale the bias detection process [[Cab+21](#)]. In this approach by Cabrera et al., systematic failures can subsequently be discovered and analyzed using a visualization interface.

Aside from model analysis and bias detection, ML data can also be analyzed and improved. To this end, Dataset Cartography helps diagnose data sets throughout the training process [[Swa+20](#)]. Here, samples within the data set are tracked so that one can resolve which of these samples are hard to learn for the model. Other visualizations that are focused on data iteration help track how a data set evolves over the course of the development process [[Hoh+20](#)]. To correct mislabeled data, Xiang et al. use trusted data samples to guide users to potentially mislabeled data [[Xia+19](#)]. Many visualization interfaces help resolve labeling errors that are introduced by less experienced crowd workers. Liu et al. designed such a visualization interface by incorporating expert validation to improve crowdsourced image labels [[Liu+18](#)]. Similar approaches have been proposed for domains other than image classification, such as for object detection [[Che+21](#)], often using an iterative process for data correction.

In this thesis, we also present an approach to correct mislabeled training data for image classifiers. Our work introduces an introspection technique that is based on using the classifier to guide practitioners to potentially mislabeled data samples [[BNR20](#)]. Additionally, this

work includes a visualization interface that facilitates the adoption and usage of this introspection technique.

1.2.2 *Communication*

While quality assurance has seen more attention from the research community, visualization can also be employed for communicating DL introspection. Such communication methods can create a broader adoption of DL models or data sets and foster a shared understanding of the involved concepts.

One line of research for communicating findings about ML systems are ML documentation visualizations. In this area, Holland et al. [HHN20] proposed the concept of Dataset Nutrition Labels. Dataset Nutrition Labels are modular graphics describing different aspects of a data set. These labels can be seen as a more visualization-focused approach to the previously mentioned Datasheets for Datasets [Geb+21]. Know Your Data takes this approach of broadly sharing insights about data sets one step further and displays data distributions and introspection results for many ML data sets in an interactive web-dashboard [Inc21a].

Other communication approaches are aimed at conveying model architectures. Many DL frameworks even have such model architecture visualization techniques directly built in, such as TensorBoard [Won+17], Caffe [Jia+14], and Keras [Cho+15]. However, these detailed visualizations are designed for debugging purposes rather than for broad communication. In contrast, visualization interfaces that are designed to help create abstract publication figures, such as drawconvnet [Din18] and convnetdrawer [Uch19], can be used to generate publication figures and, in turn, broadly communicate neural network architectures. Similarly, NN-SVG can also be used to generate publication-tailored neural network visualizations [Len18].

This thesis includes Net2Vis, a visualization interface that can be used to generate publication-tailored CNN architecture visualizations [BOR21]. These visualizations can be created automatically from the modeling code, employing a unified design grammar and reducing the time needed to create such visualizations. In contrast to the aforementioned related projects, Net2Vis supports creating visualizations for modern, large CNN architectures.

1.2.3 *Education*

With the growing power of DL comes an increased popularity of DL techniques. As such, many students, as well as practitioners from other research or product fields, are eager to learn about DL techniques. Catering to this need, modern explorable explanations have been developed. Such explorable explanations are interactive inter-

faces that communicate learning material, often through visualization in combination with text. For fields other than DL, such explorable explanations have existed for a long time, preceding the DL era. Hundhausen et al. investigated the effectiveness of such general explorable explanations [HDS02; HBo7].

The work most closely related to DL education is focused on programming education. In this area of work, Guo proposed an online Python tutor [Guo13]. Building on their earlier work, Guo et al. subsequently proposed Codechella, where pair programming for educational purposes is supported through an online visualization interface [GWZ15]. Ynnerman et al. later proposed to summarize these ideas under the term exploranation, which covers explorable explanations for educational purposes [YLT18].

In the domain of DL, these explanations shed light on the properties of DL models or data sets, their functionality, or their application domain. Harley proposed a visualization interface where users can experiment with their own inputs to a neural network model [Har15]. While it is not yet possible in their approach to train a model, visualizing how it processes data educates learners about the model’s internals, nonetheless. To explain how adversarial examples deceive DL systems, Norton and Qi propose the Adversarial Playground [NQ17].

In RevaCNN, Chung et al. focus more on exploring the activations of a neural network rather than explaining the training procedure [Chu+16]. While a model can be trained in their approach, their main visualization is focused on the network’s activation space. The most well-known example of conveying the training process of a neural network might be Tensorflow Playground. In Tensorflow Playground, a simple neural network can be trained in the browser [Smi+17]. GanLab provides a similar in-browser training experience with explanations for generative adversarial networks [Kah+18b]. However, GanLab targets a different, more complex type of neural network, namely generative adversarial networks. Both Tensorflow Playground and GanLab include simple data representations, animated data transformations, and explanatory text to support learners.

In contrast to these technical explanations, ValueCards educate students about the social impacts of ML systems [She+21]. Targeting AI-novices, Shen et al. tested their ValueCards on college students, which improved the students’ understanding of different AI concepts.

In this thesis, we present an interactive visual education environment for a different type of neural network, namely recurrent neural networks [Bäu+22b]. Additionally, we are the first to quantitatively evaluate the effectiveness of such a web-based DL educational environment.

1.2.4 Unification

Similar to the aforementioned research publications, we also use visualization techniques to make neural network introspection more usable, understandable, and applicable. To support these needs, we developed both introspection techniques and visualization interfaces as part of the work included in this thesis. The visualization interfaces presented in this work can be categorized by their target usage of quality assurance, communication, or education, as outlined above.

While visualization interfaces that integrate introspection techniques bring great benefits in these areas of AI explainability, “*scholars from different disciplines focus on different objectives and fairly independent topics of Explainable AI research*” [MZR21]. In turn, visualization interfaces can often not be shared between different practitioners, reused across domains, or combined into more powerful explainability tooling. To address this issue of non-unified visualization interfaces, frameworks for the composition of interfaces, such as ipywidgets [Jup21] and Streamlit [Inc21b], can be used. However, these frameworks lack the flexibility that many practitioners require in order to create bespoke visualization interfaces that address their analysis needs [Koe+19; Zha+20]. To this end, Marcelle presents one possible architecture for composing such ML interfaces [FCS21]. Similar to Marcelle, we present Symphony, which can also be used for the composition of interactive ML interfaces [Bäu+22c]. However, Symphony not only lets practitioners compose multiple interfaces but also makes them available across platforms, including reactive interaction techniques.

1.3 STRUCTURE OF THIS WORK

We just described how introspection techniques that are made accessible through visualization are needed in ML to ensure quality, communicate findings, and educate learners. To support this need, we set our research agenda for this dissertation as follows:

Develop methods for ML introspection and make them accessible through visualization.

Our work is detailed in [Chapter 2](#) in the form of four individual research paper contributions. This includes both novel introspection techniques as well as visualization interfaces designed to make these tools accessible to the respective target audience. Whereas introspection techniques can be used to gain insights into an ML model or data set, visualization interfaces can communicate these insights and make them actionable. We begin [Chapter 2](#) with a visualization interface that has been designed to support the correction of mislabeled training data with the help of the trained classifier itself. Next, we

describe a visualization approach that fosters research communication through an interactive, code-based generation of publication-tailored CNN architecture visualizations. Third, we present a visual and interactive learning environment for RNNs, which we compared to classical learning environments in a quantitative evaluation. To bring these diverse visualization interfaces for neural network introspection closer together, we have worked on a framework called Symphony. Symphony can be used to compose visualization interfaces in different environments and share them with the various stakeholders working on separate aspects of the ML pipeline.

Finally, Chapter 3 concludes this thesis. In this conclusion, we briefly summarize our contributions and explore directions in which further improvements could bring great benefits.

1.4 PUBLICATIONS INCLUDED IN THIS WORK

This thesis is a cumulation of previous publications that are listed here in order of their appearance in the thesis.

- [BNR20] **Alex Bäuerle**, Heiko Neumann, and Timo Ropinski. "Classifier-Guided Visual Correction of Noisy Labels for Image Classification Tasks." In: *Computer Graphics Forum* 39.3 (2020), pp. 195–205.
- [BOR21] **Alex Bäuerle**, Christian van Onzenoodt, and Timo Ropinski. "Net2Vis—A Visual Grammar for Automatically Generating Publication-Tailored CNN Architecture Visualizations." In: *IEEE Transactions on Visualization and Computer Graphics* 27.6 (2021), pp. 2980–2991.
- [Bäu+22a] **Alex Bäuerle**, Patrick Albus, Raphael Störk, Tina Seufert, and Timo Ropinski. "exploRNN: Understanding Recurrent Neural Networks through Visual Exploration." In: *The Visual Computer* (2022).
- [Bäu+22b] **Alex Bäuerle¹**, Ángel Alexander Cabrera¹, Fred Hohman, Megan Maher, David Koski, Xavier Suau, Titus Barik, and Dominik Moritz. "Symphony: Composing Interactive Interfaces for Machine Learning." In: *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (2022), pp. 1–14.

1.5 OTHER PUBLICATIONS

During the work on this thesis, several other research papers were published. While these publications are not described in detail in this dissertation, they also influenced this thesis:

¹ Shared first co-authorship

- [BJR22] **Alex Bäuerle¹**, Daniel Jönsson¹, and Timo Ropinski. “Neural Activation Patterns (NAPs): Visual Explainability of Learned Concepts.” In: (2022). arXiv: [2206.10611 \[cs.AI\]](https://arxiv.org/abs/2206.10611).
- [Bäu+22a] **Alex Bäuerle¹**, Christian van Onzenoodt¹, Simon Der Kinderen, Jimmy Johansson Westberg, Daniel Jönsson, and Timo Ropinski. “Where did my Lines go? Visualizing Missing Data in Parallel Coordinates.” In: *Computer Graphics Forum* (2022).
- [Bäu+22b] **Alex Bäuerle**, Aybuke Gul Turker, Ken Burke, Osman Aka, Timo Ropinski, Christina Greer, and Mani Varadarajan. “Visual Identification of Problematic Bias in Large Label Spaces.” In: (2022). arXiv: [2201.06386 \[cs.AI\]](https://arxiv.org/abs/2201.06386).
- [BW20] **Alex Bäuerle** and James Wexler. “What does BERT dream of?” In: *VISxAI 2020* (2020).
- [Aka+21] Osman Aka, Ken Burke, **Alex Bäuerle**, Christina Greer, and Margaret Mitchell. “Measuring model biases in the absence of ground truth.” In: *Proceedings of the 2021 AAAI /ACM Conference on AI, Ethics, and Society*. 2021, pp. 327–335.
- [Web+20] Matthias Weber, **Alex Bäuerle**, Matthias Schmidt, Matthias Neumann, Marcus Faendrich, Timo Ropinski, and Volker Schmidt. “Automatic identification of crossovers in cryo-EM images of murine amyloid protein A fibrils with machine learning.” In: *Journal of Microscopy* 277.1 (2020), pp. 12–22.

¹ Shared first co-authorship

2

CONTRIBUTIONS

Following our mission to **develop methods for ML introspection and make them accessible through visualization**, we published several visualization interfaces. Some of these publications include novel introspection techniques that build the foundation of these visualization interfaces. Altogether, our work enables ML model introspection through visualization interfaces, fostering the understanding of different aspects within the ML pipeline.

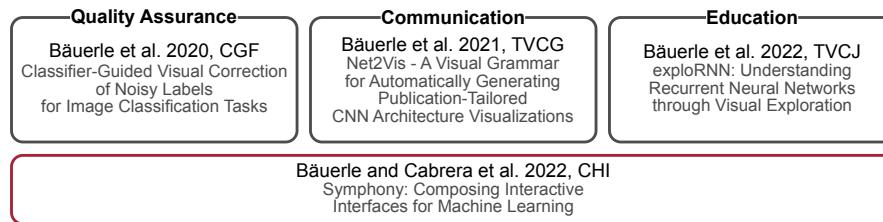


Figure 2.1: We developed visualization interfaces in three different categories.

Quality assurance is supported by our approach to correct mislabeled training data. Research communication is improved by Net2Vis, which simplifies and unifies the generation of CNN architecture visualizations. exploRNN supports learners who are interested in understanding RNN architectures. Finally, to provide a unified framework for ML visualization interfaces, we developed Symphony.

[Figure 2.1](#) depicts how our publications included in this work address three aspects of the ML development pipeline. These aspects are quality assurance, communication, and ML education. To support quality assurance, we present a method to correct mislabeled training data for image classification tasks [BNR20]. Here, we use the classifier itself to guide practitioners to potentially mislabeled data samples. Then, human observers can correct the data set by changing labels or removing data points. Second, to enhance research communication, we developed a method for generating CNN architecture visualizations directly from the code that defines them [BOR21]. This visualization interface provides a design language that can be used for different models and supports practitioners in their paper writing process. Third, we introduce exploRNN which is an interactive visualization interface to teach the concepts behind recurrent neural networks [Bäu+22b]. This interactive learning environment has been developed following predefined educational objectives and challenges. We also tested the usefulness of learning environments like exploRNN by comparing the learning outcome, required cognitive resources, and joy while learning to a conventional text-based learning environment.

In sum, visualization interfaces such as the ones we developed for quality assurance, communication, and education help make introspection techniques accessible to their intended target users.

Finally, while such visualization interfaces can support investigating specific aspects of the ML pipeline, they are rarely designed to be reused in different contexts or by different stakeholders. We developed Symphony [Bäu+22c] to support such reusability and shareability while maintaining the explorability of visualization interfaces for ML introspection. The Symphony framework is designed to unify the landscape of ML interfaces. With Symphony, ML interfaces can be combined and reused in different environments, such as computational notebooks and web-based dashboards.

2.1 CLASSIFIER-GUIDED VISUAL CORRECTION OF NOISY LABELS FOR IMAGE CLASSIFICATION TASKS [BNR20]

Modern DL models require large training data sets to unfold their generalization potential. As such, training data sets for image classification models nowadays contain up to 10 million images [Kuz+20]. For classification models, these large data sets need to be labeled so that the model can be trained on the loss measured between a model prediction and the ground truth label of a training data sample. Assigning ground truth labels for such a large number of images is not feasible for any single person, and paying domain experts to label large numbers of images is often too expensive. Therefore, labeling is often outsourced to crowdworkers who do not necessarily have the domain knowledge to correctly label every single image [KLA17; Ras+10]. Sometimes, automatic approaches are even used to obtain labels for image data sets [Usa+11]. These approaches take into account available information about the images in the data set, e.g., captions or descriptions, to assign labels to a large number of images automatically. Just like obtaining labels from non-expert human annotators, automatically extracting labels from metadata is error-prone [Zha+21]. However, when the training data contains incorrect labels, model performance can degrade quite drastically [NOPF10; Pec+06]. To address this issue, we worked on quality assurance for ML applications through a visualization interface. This visualization interface uses the trained classifier as a guide to potentially mislabeled data. Practitioners can make use of this guidance for interactive label error correction.

CONTRIBUTIONS

In this work, we make three main contributions towards detecting and correcting errors in labeled training data. First, we propose a categorization of common types of errors in image classification train-

ing data. Second, we develop measures that take these error types into account. With these measures, users can be guided to potential errors in the data set. Third, we provide a visualization interface that incorporates these metrics as means for user guidance and allows for a direct manipulation of labels. Altogether, our approach is based on the classifier output itself and requires no additional information to facilitate error correction.

IMPLEMENTATION

We identified three main types of errors that may occur in labeled image classification training data. Class Interpretation Errors (CIEs) occur when a labeler confuses two classes (e.g., the way of writing ones and sevens between different western cultures). Since entire classes are confused for this type of error, there exists a set of training samples where all images are mislabeled in the same way. To detect candidates for this type of error, we search for unusually large numbers of data samples that are labeled as \hat{y} but classified as y . In contrast, Instance Interpretation Errors (IIEs) occur when only one single data point has been mislabeled. One way these types of errors typically get introduced is through plain misclicks. To guide users to these types of errors, we search for data samples that are confidently misclassified. Mathematically, this means that the assigned classification probability for label y is much larger than that for \hat{y} . For spotting Similarity Errors (SEs), we extract images that are highly similar through similarity metrics. We use SSIM as a similarity metric [Wan+04], but any similarity metric for images would work in this context. To reduce the number of images we need to calculate similarity scores for, we only measure similarity between images that the classifier assigned the same label to. If the classifier already sees two images as different enough to assign different classification results, we assume that these images must be sufficiently different.

While these error types helped us design metrics that can guide users to error candidates, there is no guarantee that error candidates extracted through these metrics actually correspond to labeling errors. Instead, for CIEs and IIEs, there is always the possibility that the model is just not performing well on these samples, while SEs might be intentionally added to augment the data distribution or emphasize certain important image features. Therefore, extracted error candidates require human oversight before being resolved. We support such human oversight in a three-step process. Practitioners first need to detect potential errors in classification training data, then they can reason about these error candidates, before eventually resolving them.

In our visualization interface, detecting error candidates is supported through visual guidance. This guidance is implemented through the visualization of the aforementioned metrics. For both CIEs and

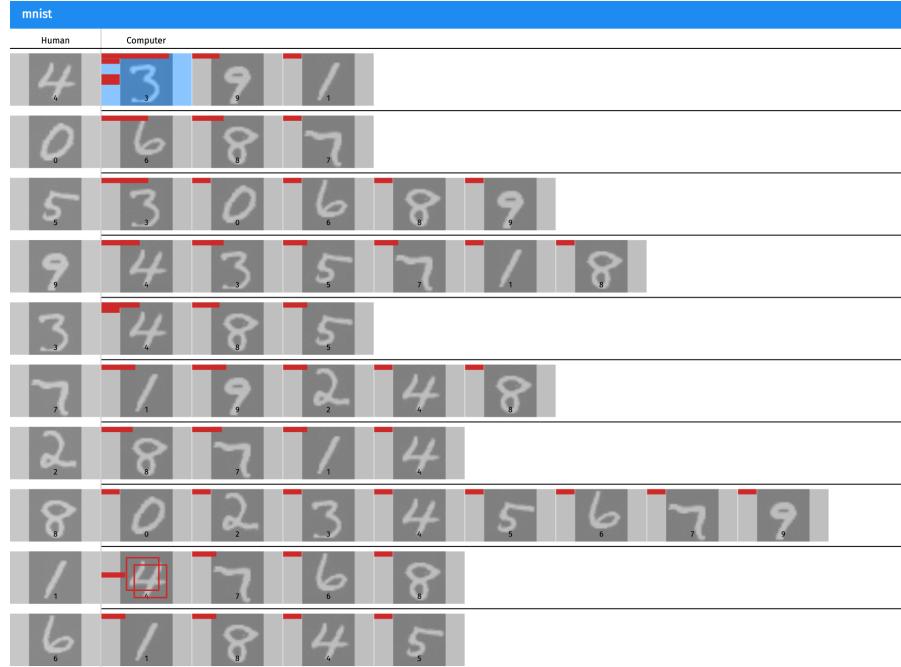


Figure 2.2: A modified confusion matrix is used to guide practitioners to potential labeling errors in the training data set. Cells are sorted according to their CIE score, bars indicate IIE scores, and duplicate icons depict high SE scores. Figure taken from Bäuerle et al. [BNR20].

IIEs, we are only interested in data samples where the classifier disagrees with the human annotator. To guide users to these samples, we organize the data in a modified confusion matrix as shown in Figure 2.2. Cells that contain correct classifications (agreement between annotator and classifier) are not as important for error detection. Therefore, we can use these correctly classified samples as visual anchors, placing them in the first column of the matrix to resolve the label that was assigned for all data samples in the respective row. To highlight potential CIEs, we sort the rows so that rows that contain more misclassified images appear at the top. Within each row, we further sort all cells from left to right by the number of misclassifications they represent. As this reorganization removes the columnar mapping of columns to model predictions, we place a representative data sample in each cell to clarify the model prediction it represents. To further highlight potential CIEs, cells that represent large numbers of misclassifications are colored in blue. For IIEs, we are interested in the most confident misclassifications, i.e., where the model assigns a much higher probability to y than \hat{y} . To visualize this confidence score, we display a histogram within each cell, whereas the vertical positioning of a histogram bar indicates misclassification confidence. The most confident misclassifications are thus represented by bars close to the top of a cell. SEs can occur in any cell regardless of whether it rep-

resents misclassifications. To guide users to these error candidates, we place a duplicate icon over any cell that contains samples with a high similarity score. Altogether, this overview visualization guides practitioners to all three types of typical labeling errors we identified for image classification training data.

To reason about error candidates, one can look more closely at the data samples that one of the cells in this matrix visualization contains. For large numbers of samples, i.e., for CIEs, we display a list of all samples within one cell. To spot IIEs, we use a projection visualization of the images. IIEs typically represent a different data distribution than plain misclassifications as they display a different class. Therefore, IIEs can often be found by looking at outliers that are well visible using such a projection [McI+18]. For SEs, the most similar images of the cell are displayed separately in a list so that they can be easily compared. Looking at these images in detail allows practitioners to make an informed decision on the correctness of individual data samples.

Finally, to resolve error candidates, labels can be either confirmed or changed directly in our visualization interface. Additionally, if duplicates have been found or a data sample does not appropriately represent any label, data samples can also be removed entirely from the data set. The whole process of training a classifier, obtaining error scores, and visually investigating potential errors can be repeated multiple times, as with each iteration, different images might be suggested as error candidates. Once data quality is sufficient, the data set and model can go into productive use (see [Figure 2.3](#)).

FINDINGS

During this research, we found several errors in real-world benchmark data sets such as MNIST [[LC10](#)] and CIFAR 10 [[Kri+09](#)]. Additionally, to evaluate our approach, we conducted a qualitative user study with 10 participants. After a brief introduction, participants were asked to correct an intentionally corrupted version of the MNIST data set within a 15 minute session of using our visualization interface. We introduced all three error types, which resulted in 3,300 incorrect labels and 5 duplicates in total. All participants resolved all duplicates and changed the labels of 2,902 images on average. Of these changed labels, an average of only 27.5 labels were incorrectly changed. In total, they managed to correct 85.65% of the mislabeled data in just this one 15 minute session. On average, this increased classifier accuracy from 94.37% to 99.05%. In turn, we conclude that this method works for the intended use case of correcting mislabeled image classification training data. When looking at the images that were incorrectly changed, we found that many of them could be argued to be incorrectly labeled in the original data set instead.

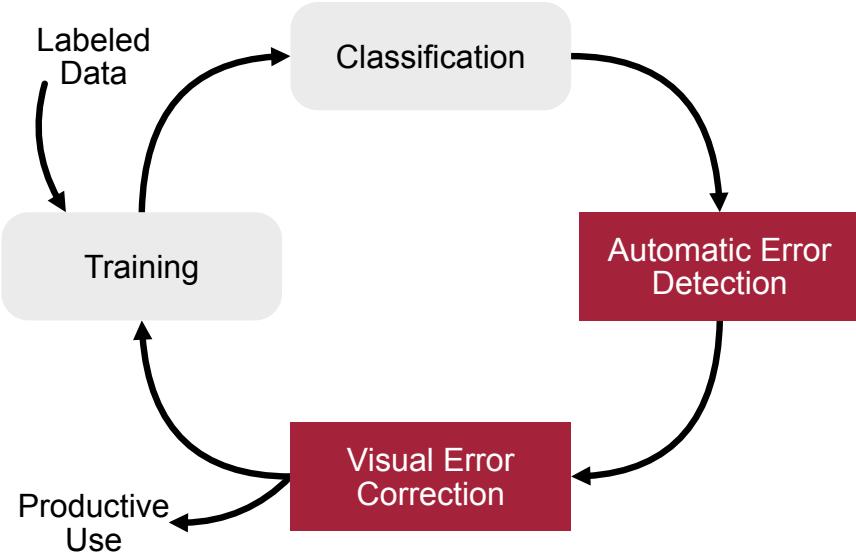


Figure 2.3: We use the classifier to guide ML practitioners to error candidates in a training data set. Then, errors can be reasoned about and resolved within a visualization interface. After this correction process, the data set and model can either be used productively or refined through further iterations of data correction. Grey boxes are existing steps of the process, while red boxes represent what we propose to add. Figure taken from Bäuerle et al. [BNR20].

Overall, this project presents a technique to extract error candidates. These error candidates can then be interactively corrected through a visualization interface to ensure classifier quality. To support error correction, we propose a categorization of typical error types for image classification data. Through measures that are based on classifier response analysis, we provide means to identify candidates for these error types. Finally, to detect, reason about, and resolve potential errors, we implemented a visualization interface that incorporates the aforementioned metrics. In a qualitative user study, we saw that the proposed approach helped correct errors in a corrupted image classification data set.

2.2 NET2VIS – A VISUAL GRAMMAR FOR AUTOMATICALLY GENERATING PUBLICATION-TAILORED CONVOLUTIONAL NEURAL NETWORK (CNN) ARCHITECTURE VISUALIZATIONS [BOR21]

The second focus area in this thesis was the communication of findings about ML data or models. As shown in Figure 2.4, we saw a clear trend toward including such figures in research papers at prime conferences for vision-related ML tasks. However, when looking at how visualizations of ML model architectures are used in publications, we found that mostly one-off, handcrafted visualizations are employed.

In turn, these figures lack a common visual grammar as different researchers create figures based on their gusto and capabilities rather than agreed-upon conventions. Creating such figures also requires a significant time investment that could otherwise be directed towards improving the main content of the publication. Finally, handcrafted figures are error-prone since they are often generated just before publication. Based on these findings that we extracted from a survey of existing publication figures, we set out to develop Net2Vis. Net2Vis simplifies and unifies the creation of publication-tailored CNN model architecture visualizations.

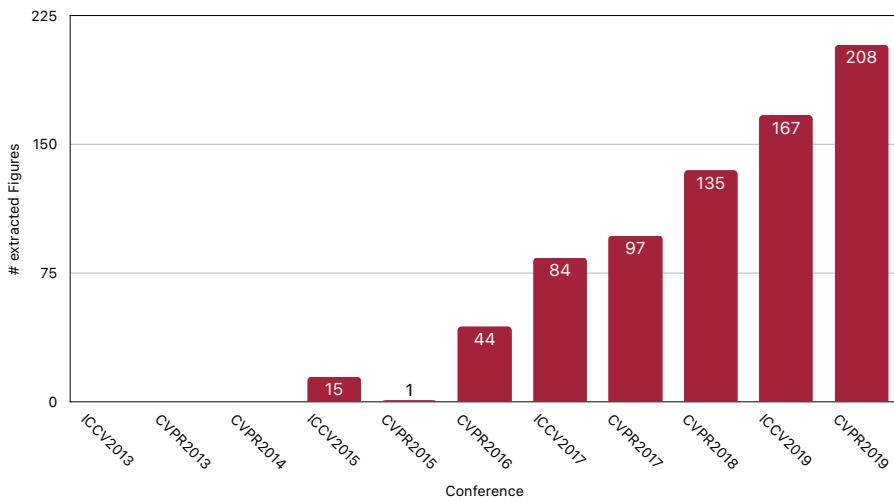


Figure 2.4: Number of CNN architecture figures we extracted from all ICCV and CVPR papers between 2013 and 2019. We searched for pages of papers containing figures and the words *figure* and *architecture* in the same line to extract these. Then we manually filtered them to obtain only neural network architecture visualizations. Figure recreated based on Bäuerle et al. [BOR21].

CONTRIBUTIONS

In this work, we make three contributions towards simplifying and enhancing the presentation of CNN model architectures. First, based on expert feedback and the evaluation of a set of existing visualizations, we propose requirements for an effective communication of ML model architectures. Second, we developed a visual grammar for CNN architecture visualizations and provide an online platform where ML practitioners can obtain publication-tailored visualizations directly from the code that defines their architectures. Third, we release a data set of 751 annotated neural network architecture visualizations that have been extracted from all papers published at ICCV and CVPR between 2013 and 2019.

IMPLEMENTATION

While there exist other approaches to automatically generate visualizations of neural network architectures, neither of them is designed for the compact visualization of modern neural network architectures that is required for publication figures. Visualization approaches such as TensorBoards graph visualizer [Won+17], Cafe’s Netscope [Gsc17], and Netron [Roe18] are powerful visualization interfaces that support all kinds of architectures. However, they are designed for use during the development of neural network models. As such, these interfaces show a detailed, vertical visual encoding, which can help investigate an architecture in detail but would not fit into a publication. There are also more abstract visualization approaches such as convnetdrawer [Uch19] and drawconvnet [Din18]. However, these visualization environments do not support modern neural network architectures as they cannot display parallel execution paths or aggregations of layers. To fill this gap, we developed Net2Vis, an interactive interface for the generation of visualizations based on the Net2Vis visual grammar. An example of a figure generated with Net2Vis can be seen in [Figure 2.5](#).

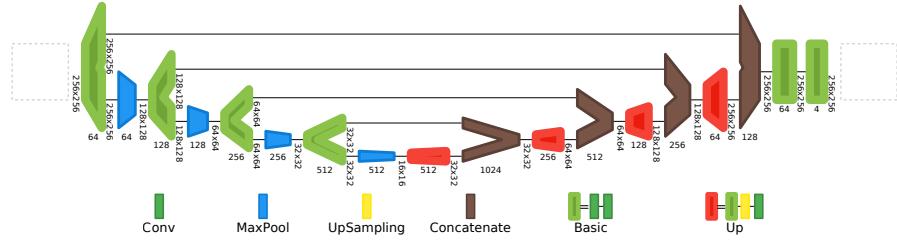


Figure 2.5: Visualization of a U-Net variant. It was automatically generated using Net2Vis based on the Keras code describing the architecture. Data flows from left to right. Glyphs represent layers or aggregates, while lines represent connections. Glyph widths communicate feature size, while heights communicate the spatial resolution. Both values are also given through labels. Dashed boxes on the left and right serve as placeholders to provide input and output samples. The legend communicates layer types and the composition of aggregates. Figure taken from Bäuerle et al. [BOR21].

As a foundation for our proposed visual grammar, we analyzed and annotated 751 neural network architecture figures, which we extracted from ICCV and CVPR papers that were published between 2013 and 2019. We coded each visualization based on different attributes, such as what dimensionality the encoding uses, how layers are laid out, how connections are visualized, and more. Additionally, we talked to ML practitioners who worked on some of these publications to obtain direct feedback about their needs and wants. Based on this analysis, our visualization design incorporates considerations for both

visualizing the overall model structure and how to encode individual layers.

For an overview, the most important piece of information that needs to be communicated about a neural network architecture is the data flow through a model’s layers. We follow the reading direction of most western cultures and position the layers of the neural network architecture from left to right. As modern architectures can contain parallel execution steps, we draw lines between a layer that sends data to another layer. However, such a line-based encoding can introduce size-related attention bias, where the centerline of the visualization is seen as more important than separate paths of execution. As there is no importance ranking in neural network architectures, our goal was to mitigate this size-related attention bias. Therefore, we add multiple handles to layers that start or combine parallel execution steps. With this visual encoding, the flow of data is communicated in a way that supports and promotes CNN architectures. However, the resulting visualized graphs can exceed the space that is typically available for such a figure in a publication. To compact the resulting graphs, ML practitioners can combine repeated structures in their networks into groups. Such combinations are replaced with a single layer glyph and resolved in a legend that maps layer types to colored glyphs.

The visual encoding of an individual layer in a neural network model in Net2Vis is focused on communicating the important transformation into and out of latent space. For this data transformation to be communicated effectively, we map the height of a layer glyph to the spatial resolution it operates on. However, assigning one fixed height to a layer glyph leaves the question of how a layer that changes the spatial dimension of the data through its operations should be depicted as the incoming data dimensionality does not match the dimensionality after the layer has been applied. To remove this ambiguity, we assign different height values to the respective ends of a layer, thus mapping the height on the left side of a layer glyph to the incoming data dimensionality and the height on the right side of a layer glyph to the transformed data dimensionality. Closely related to the spatial resolution of data is the number of features that are used to encode the data. Therefore, we use a similar encoding to what is employed for the spatial resolution and communicate the number of feature channels through the width of a layer glyph. Since the number of feature channels is based on one fixed value that the programmer of a neural network architecture sets upon initializing a layer, there is no need to differentiate between unprocessed and processed numbers of feature channels. In turn, we can use one fixed width value to encode the number of feature channels that a layer operates on. In addition to visualizing the data transformation of a neural network layer, we also communicate the layer type in our visual grammar for neural network layers. While many current visualization approaches use a textual en-

coding of the layer type, layers are typically repeated throughout the network structure. Thus, to reduce repetition, we color-code layers and resolve their types in a legend below the main network visualization.

While this encoding on both a network and layer level provides a common visual grammar for CNN architectures, it does not necessarily simplify the creation of such visualizations. To address the problem, which is that authors of publications need to spend valuable time on creating and refining such CNN architecture visualizations, we additionally provide an interactive visualization interface with which such publication figures can be created. In this interface, ML practitioners can obtain visualizations by simply pasting their CNN model code, which gets automatically transformed into a visualization using the aforementioned visual grammar. After parametrizing the visualization, e.g., through adding groups, changing colors, or adjusting sizes, ML practitioners can then download a publication-tailored visualization of their model architecture.

FINDINGS

To evaluate our visual grammar, we conducted a quantitative user study with 10 participants. Hereby, we used visualizations of architectures from well-cited publications that were obtained through Net2Vis, TensorBoard, and the paper figures themselves. We then showed these visualizations one after the other to our participants and asked them to answer questions such as “*how much downsampling does this model do?*” or “*how many convolution layers does this model include?*”. In this evaluation, visualizations generated with Net2Vis were shown to be the most accurate. Additionally, in a follow-up questionnaire where we showed the same architecture using the three aforementioned encoding variants (Net2Vis, TensorBoard, paper figure), Net2Vis was preferred by most participants. We also already saw the adoption of Net2Vis in newly published research [[Sar+21](#); [Ver+22](#)].

In summary, Net2Vis improves research communication through automatic visualization generation. Net2Vis is supported by a visual grammar that is based on the analysis of existing CNN architecture visualizations, visualization knowledge, and expert feedback. To simplify the creation of said CNN architecture visualizations, we implemented a visualization interface that provides such visualizations using only the code that defines an architecture as input. Our quantitative evaluation that compares Net2Vis with other common visual encodings further indicates that Net2Vis works better than other methods currently used for communicating CNN architectures in publication figures.

2.3 EXPLORNN: UNDERSTANDING RECURRENT NEURAL NETWORKS THROUGH VISUAL EXPLORATION [BÄU+22B]

The third focus area for visualization interfaces that has been addressed in this thesis is educating learners about the concepts behind neural networks. Education becomes ever more important as DL is adopted in diverse fields, bringing new learners with different backgrounds to neural network education. To support this growing need for learning material, we present exploRNN, an interactive learning environment that teaches the concepts behind RNNs and long short-term memory (LSTM) cells. Other network types such as feed-forward neural networks [Smi+17] and generative adversarial networks [Kah+18b] have already been targeted by researchers developing interactive learning environments. However, education in the growing field of sequential data processing, which is often performed with RNNs, is still mostly based on conventional methods such as lectures or textual learning material. The advantage of interactive learning environments compared to lectures is that they don't need teacher supervision. However, it is not clear how interactive learning experiences compare to the similarly self-paced text-based education, which we investigated in our research.

CONTRIBUTIONS

exploRNN is the first interactive educational visualization interface designed specifically to convey the unique architecture and functionality of RNNs to learners. The development of exploRNN was driven by educational objectives and design challenges that arise in the context of RNNs. Our quantitative user study which compares exploRNN against text-based learning evaluates the learning outcome, required cognitive resources, motivation, and joyfulness of the learning experience. The results provide insights for future interactive learning environments and indicate that more learners are willing to spend more time learning with an environment like exploRNN.

IMPLEMENTATION

In this work, our goal was twofold, namely, to implement an interactive learning environment for RNNs and to conduct the first quantitative evaluation comparing such interactive learning environments to text-based learning.

To inform our implementation of exploRNN, we first defined educational objectives for the learning experience. These educational objectives were created to both guide our visualization design and serve as a foundation for what to test in our evaluation. We defined four educational objectives for exploRNN which are briefly described

in the following. First, the justification for when to use RNNs over other network architectures should be clarified by exploRNN. This also includes the backpropagation mechanisms for RNNs, namely backpropagation through time (BPTT). Second, learners should be taught how LSTM cells are built to understand how temporal information can be captured by RNNs. We focused on the LSTM cell architecture as one of the most frequently used [WGY18] and best performing cell architectures [Bri+17]. However, the learned ideas are expected to be transferable to other cell types. Third, the RNN training process should be communicated by exploRNN so that learners can apply the gained knowledge in their own DL projects. Finally, learners should get an idea of the different tasks that can be solved with RNNs. Taken together, these educational objectives are designed to give learners an overview of the techniques from which further exploration and experimentation are possible.

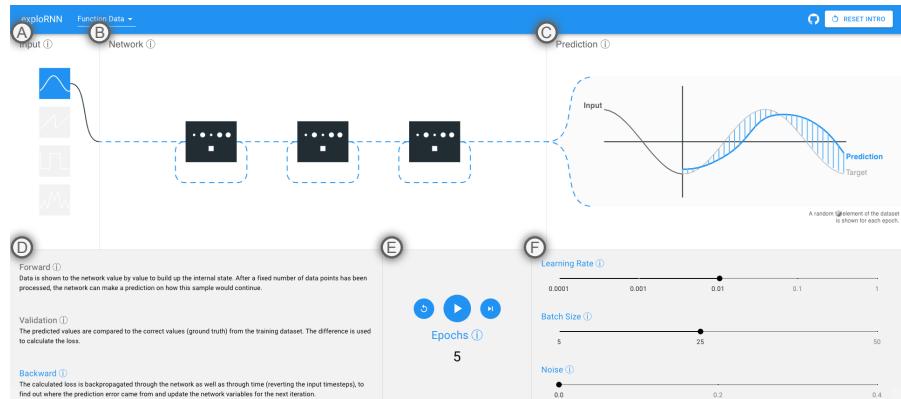


Figure 2.6: (A) Simple input types illustrate the abstract concepts behind RNNs. (B) An animated, modifiable network architecture shows the data flow. (C) The prediction visualization shows the network input, prediction, ground truth, and error bars, all animated to communicate their temporal nature. (D) Text helps explain the training process. (E) RNNs can be interactively trained. (F) Training parameters can be interactively explored. Figure taken from Bäuerle et al. [Bäu+22b].

exploRNN itself is an interactive learning environment designed to transport these educational objectives. We implemented exploRNN using a multiscale approach. Here, the overall training process is communicated as shown in Figure 2.6, but we also educate about details of the computation inside of individual cells (see Figure 2.7). Furthermore, we provide textual explanations for many elements on both levels that allow learners to inspect specifically those components that are important to them (see Figure 2.7). This multiscale approach of information transportation adopts the *overview first, zoom and filter, details on demand* visualization mantra proposed by Shneiderman [Shn03].

The network overview which is shown in Figure 2.6 consists of six main elements. The input data to train the network is visualized on the

left of this view. Users can select simple functions but also text snippets as training input. Including both function and text data facilitates knowledge transfer to new, more realistic settings. The selected type of training data is animated to symbolize the flow of data into the network. We depict the model itself via glyphs for individual layers. To communicate recurrence, we add a loop to each layer glyph. Lines that transport data are animated so that they move in the direction of data flow (forward during inference, backward during training). The input data and predictions are shown on the right of this view. In this visualization, the ground truth and error are also shown. Thus, users can learn how the model improves its prediction over the course of the training process. Below these main visualizations are more detailed explanations, controls for the training process, and settings for training hyperparameters.

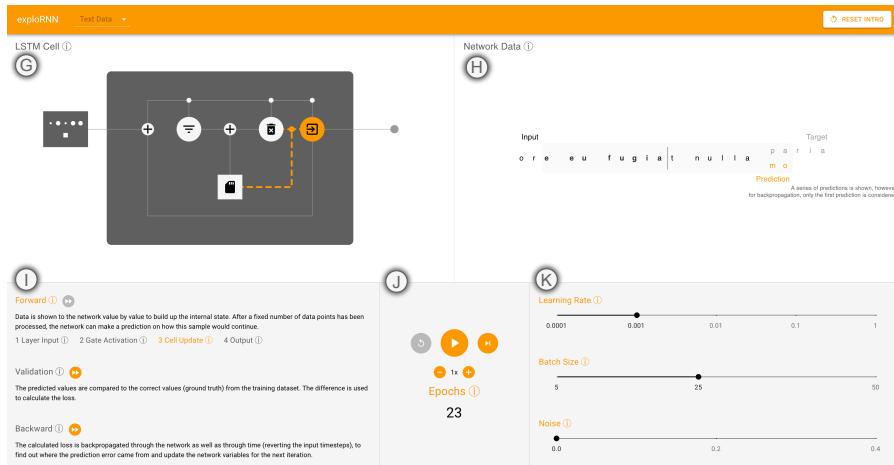


Figure 2.7: (G) Visualization of data flow through the cell. (H) Input to the network and its prediction. Visualization of the training error computation. A grey sliding window indicates which data points are needed to initialize the cell state. (I) Explanations with more detailed steps for the forward direction of data flow. (J) The speed at which the visualization for cell steps advances can be changed. (K) Just as in the network overview, users can modify training parameters. Figure taken from Bäuerle et al. [Bäu+22b].

In the LSTM cell view, which learners can access through the selection of one layer in the network overview, we show an LSTM cell with all its computation units. Additionally, the data processed by the network and, similar to the network overview, controls for the training process (see Figure 2.7) are displayed. In contrast to the network overview, training progresses more slowly in this visualization to show the individual computation steps inside the cell gates. Again, connections between cell components that transport data are animated, and the input, prediction, ground truth, and prediction error are visualized.

Memory Cell

The **cell state** is the heart of any LSTM cell. By having a cell state, and deciding how to update it based on **the filtered input and previous state**, LSTM cells are able to **capture long-term dependencies**.

$$c^t = \text{filtered_input} + \text{filtered_state}$$

Symbols:

c^t : the cell state at timestep t

Figure 2.8: Users can access more detailed explanations for many elements of our visualizations, such as training steps, hyper-parameters, and operations in a cell. Figure taken from Bäuerle et al. [Bäu+22b].

We implemented exploRNN as a web application using TensorflowJS [Smi+19]. Throughout our visualizations, we use animation to convey the sequential flow of data inside the RNN. To prevent the need for teacher supervision, we implemented an onboarding process, which explains the functionality and concepts behind exploRNN. Textual explanations are used throughout exploRNN to provide details on demand.

FINDINGS

To evaluate exploRNN, we conducted a quantitative user study with 37 participants. Our study was set up as the last lecture of a DL course, so students already knew about feed-forward NNs. Each participant either used exploRNN or received a learning text to learn about RNNs. We evaluated the learning outcome divided into recall, comprehension, and transfer [Blo+56]. Furthermore, we measured the cognitive load that the learning experience inflicted on learners, divided into intrinsic, extraneous, and germane cognitive load [Swe11]. Finally, we also used the System Usability Scale (SUS) [Bro+96] to evaluate the usability of exploRNN and asked the participants qualitative questions about the learning environment.

We could not find a significant difference in learning outcome between exploRNN and text-based learning. In fact, for superficial knowledge acquisition (recall), we even found that the text condition led to better learning results. However, learning with exploRNN required fewer cognitive resources. exploRNN also proved to create a more likable and fun learning experience. While we did not test this hypothesis, reduced cognitive load in combination with a more

enjoyable learning experience might lead to more time spent learning. In turn, visual and interactive learning environments might still lead to a better learning outcome over a longer period. We hope that these insights can guide the development of future learning environments and motivate further comparative evaluations of visual and interactive vs. classical learning.

In this work, we present an interactive learning environment for RNN education. With our visualization approach, we provide an overview of the training process but also let learners explore individual components in detail. exploRNN is designed based on educational objectives for RNN education. Our evaluation is the first to qualitatively compare an interactive NN learning environment with classical learning approaches.

2.4 SYMPHONY: COMPOSING INTERACTIVE INTERFACES FOR MACHINE LEARNING [BÄU+22C]

The aforementioned projects are all standalone visualization interfaces that address one specific task or problem. Apart from the interfaces presented in this thesis, there exist numerous ML visualization interfaces. Examples of such interfaces are documentation methods (e.g., Model Cards [Mit+19], Datasheets [Geb+21]), visualization dashboards (e.g., What-if Tool [Wex+20], ActiVis [Kah+18a]), and interactive programming widgets (e.g., ipywidgets [Jup21], Streamlit [Inc21b]). All of these ML interfaces support practitioners in specific aspects of their model or data analysis needs. Despite the benefits of these interfaces, recent studies [Zha+20; Koe+19] and interviews we conducted with practitioners revealed that these ML interfaces have limited adoption in practice. To address this issue and simplify the adoption, combination, and reusability of ML interfaces, we developed Symphony. The Symphony framework is designed to unify the landscape of ML interfaces, making them accessible across platforms and for different stakeholders.

CONTRIBUTIONS

With Symphony, we designed a framework to unify the landscape of ML interfaces. Our design process included formative interviews, participatory design sessions, and case studies on deployed ML workflows. Altogether, we collaborated with 39 ML practitioners across 15 product and engineering teams. Symphony can be used to compose data-driven ML interfaces that include task-specific visualizations and allow for interactive exploration. These interfaces can be reused by different stakeholders and shared across environments, such as computational notebooks and standalone web dashboards.

IMPLEMENTATION

Our formative interviews with 9 ML practitioners surfaced three main problems that hinder the adoption of current ML interfaces. First, practitioners often use ad-hoc tools and analyses that cannot be easily reused and require extensive manual labor to create. This is because no visualization interface exists that matches the specific needs of ML practitioners. Second, existing ML interfaces are limited as most of them require time-consuming data preprocessing before they can be used. Often, this even requires a move to new platforms or environments, disrupting the workflow that practitioners have. Furthermore, sometimes existing interfaces are not designed for the data types that ML practitioners are working with. Finally, we found that there is a lack of communication between different stakeholders working on an ML project. As those stakeholders often work on different platforms, it is hard to share insights gained through ML analysis.

Based on these insights, we set out to build Symphony, a framework to unify the landscape of ML interfaces. The goal of Symphony is to provide ML interfaces that are connected to the ML system's backing data, in turn supporting the task-specific visualizations that are needed by practitioners. On top of that, Symphony is designed to provide interactive exploration tools that allow for flexible discovery and validations, and to make components reusable across different environments, domains, and tasks.

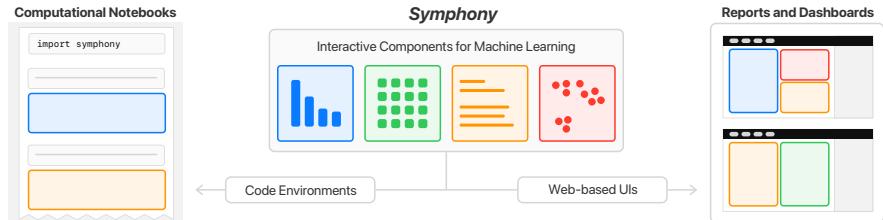


Figure 2.9: Symphony provides interactive visualization components which can be used in different environments, namely Jupyter notebooks and web-based dashboards. Figure taken from Bäuerle and Cabrera et al. [Bäu+22c].

The primary concept behind Symphony, which is depicted in [Figure 2.9](#), is based on an expandable set of components with synchronized, interactive visualizations. These components can be used across different mediums, such as computational notebooks or web-based dashboards and reports. This allows engineers to do exploratory model development while other stakeholders, such as decision-makers or policymakers, may view the same analyses on a web-based dashboard.

As shown in [Figure 2.10](#), Symphony's visualizations are informed directly by a practitioner's data. This data includes both metadata of an ML model or data set as well as the raw data samples. Based on this

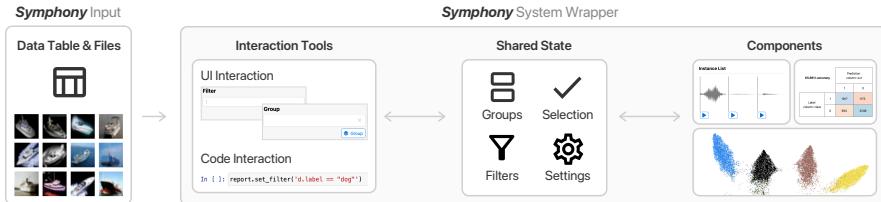


Figure 2.10: Backing data is used by Symphony’s system wrapper to provide interactive, synchronized visualizations in different environments. Figure taken from Bäuerle and Cabrera et al. [Bäu+22c].

backing data, Symphony includes modular visualization components that can be combined into Symphony reports. In our implementation, visualization components are provided as Python packages, which makes them shareable and reusable across teams or institutions via PyPi. A shared state functions as a representation of that data for all visualization components that are in use. Interaction tools, which can be UI-based or code-based, provide means to modify said shared state. New visualization components can be implemented in JavaScript, which makes them more flexible compared to other common charting libraries such as Matplotlib [Hun07] or Altair [Van+18].

System wrappers make components available in different environments so that stakeholders can use them on their preferred platforms. These wrappers pass data from a backing environment to Symphony and render Symphony components within the environment’s UI. We implemented wrappers for Jupyter notebooks as well as web-based UIs. An example of how the same visualization components are made available by Symphony can be seen in Figure 2.11. As such, ML practitioners can start their model or data exploration in computational notebooks, assembling the visualization components required for their analysis. Once they want to share insights or add other stakeholders to the analysis process, they can export the generated symphony report as a standalone, statically hosted web dashboard. Altogether, this allows for reuse and combination of visualization components, enables exploration through interactivity, and fosters communication through its shareability.

FINDINGS

We evaluated Symphony through three case studies with ML practitioners working on real-world ML products. In think-aloud studies that lasted 60 minutes, ML practitioners used Symphony’s Jupyter environment as well as the standalone dashboard. The case studies were conducted with three different teams, where the tasks were *validating and sharing data patterns of a data set creation team*, *debugging training data of an accessibility team*, and *promoting data exploration for ML novices in an education team*, respectively. Throughout these case

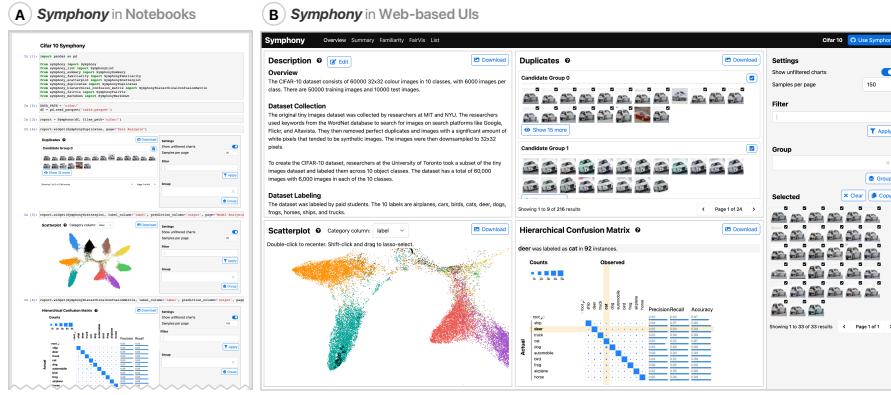


Figure 2.11: The same visualization components can be used both in computational notebooks (A), as well as in web-based dashboard UIs (B). Figure taken from Bäuerle and Cabrera et al. [Bäu+22c].

studies, Symphony enabled practitioners to discover issues with their data or models and encouraged them to share their insights with other stakeholders.

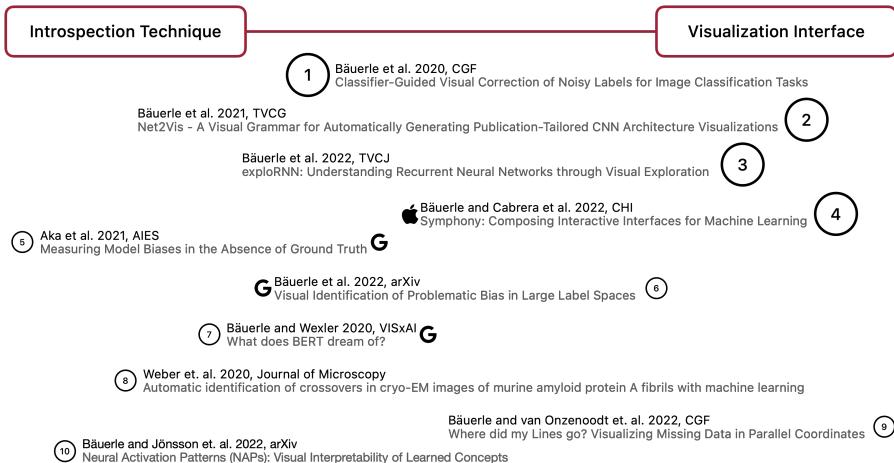
Altogether, Symphony adopts the idea of task-specific visualization components that can be reused in different environments and by different stakeholders. Furthermore, Symphony brings the landscape of ML interfaces closer together, as different interfaces can be flexibly composed, all operating on the same shared data. In turn, Symphony fosters a culture of shared ML understanding and encourages the creation of accurate, responsible, and robust AI products.

3

CONCLUSION

The work that influenced this thesis is summarized in [Figure 3.1](#). This thesis includes four main contributions, which are highlighted with large circle markers. These main contributions include three visualization interfaces covering the areas of quality assurance, communication, and education. Furthermore, Symphony, a framework that brings different visualization interfaces together and fosters shareability of analyses, is included as one of the main publications in this work.

Apart from these main contributions, this thesis was also influenced by further publications. These cover the wide range from pure introspection techniques to pure visualization interfaces. Some of this additional work was done during internships at Google and Apple.



[Figure 3.1](#): Overview of the contributions that influenced this thesis. Large marks represent the main contributions that are part of this thesis while small marks depict contributions that influenced this thesis but are not part of its main content. Some of this work has been conducted during internships at Google and Apple.

In the following, we talk about future work at the intersection of introspection techniques and visualization interfaces. Finally, we will briefly summarize the contribution that this thesis makes to the field.

3.1 FUTURE WORK

Our work brings improvements to various elements of the ML pipeline across different application areas. However, the need for better introspection techniques and visualization interfaces is still more present than ever. In the following, we summarize some of the most promising directions in which future work could bring great improvements.

Introspection Techniques.

Various model attributes have not yet been studied exhaustively. For example, in our preprint on neural activation patterns [BJR22], we search for clusters in the activation space of neural networks. This is only one example of how the attributes of a neural network can be used to provide introspection techniques. Further investigations into the role of activations, connection weights, and their change during training might bring new introspection techniques that help explain the functionality of a neural network. Furthermore, introspection techniques are often used in isolation. Combining multiple introspection techniques effectively and understandably could lead to a more complete picture of a neural network and its decisions.

Additionally, while a lot of focus has been put on introspecting CNN-based image classification models, other model architectures are still missing a similar set of introspection techniques. If we, e.g., take the field of feature visualization, there are numerous publications on how to use this technique to systematically probe into and understand CNN models [OMS17; Ola+18; Car+19; Cam+20]. These techniques do not yet work as well for other network architectures [PRS18] such as transformers [BW20]. Work on the systematic explainability of transformers is still in its early stages [Elh+21]. However, introspection techniques are important for any network architecture, which calls for cross-functionality. If we are unable to provide introspection techniques that are architecture-independent, the landscape of introspection techniques will always struggle to keep up with the newest architectural trends.

Finally, the introspection techniques we develop need to be evaluated to ensure they live up to their promises. For attribution techniques, a substantial amount of auditing research has been conducted, e.g., [Ade+18; Ade+20; DS22]. These auditing methods show that many of the regularly employed introspection techniques have serious flaws, which calls for more reliable approaches. Additionally, for introspection techniques to be widely used and reliable, auditing approaches that systematically test their efficacy are essential. Especially introspection techniques that are designed to explain what a model has learned instead of focusing directly on an input sample and its corresponding prediction are lacking auditing methods.

Visualization Interfaces.

In the area of visualization interfaces, we have seen a lot of focus on specific settings. Namely, ML developers performing classification tasks on structured or image data have seen most attention. However, modern ML systems can encompass complex data types and target end users without ML knowledge. Visualization interfaces need to account for this need by supporting different types of data and ML models.

Furthermore, it is crucial to enhance the interaction with ML systems, display prediction confidences, and highlight the limitations of an ML system's capabilities for domain experts who lack ML knowledge. As such, visualization designs for ML novices have to be easier to understand and must explain concepts on a different, higher level.

A lot of the visualization work so far has also focused on individual, one-off visualization interfaces that are hard to reuse, combine, and share. Therefore, we see limited adoption of these tools in practice [Bäu+22c]. As a visualization and human-computer interaction community, we should strive to improve the usability of these visualization interfaces to foster their adoption in the field. With Symphony, we made a first step towards that goal [Bäu+22c], but to fully address practitioners' ML analysis needs, many hurdles still need to be overcome. Future work might be able to remove these hurdles by streamlining explainables and promoting their integration into any ML workflow. We have seen similar advancements in information visualization, where tools like Tableau [LLC21], D3 [BOH11], and Vega [Sat+15] greatly simplify visualization authoring. A similar improvement in the domain of ML visualization could both improve our understanding of ML systems and cement the importance of visualization as a research field in connection with ML.

3.2 SUMMARY OF CONTRIBUTIONS

In this dissertation, we set out to:

**Develop methods for ML introspection
and make them accessible through visualization.**

To this end, we present three publications that target different areas in which they support the ML development pipeline through introspection and visualization. Through such introspection-based visualization interfaces, we can learn about these ML systems, assure their quality, and communicate our findings when developing or investigating such systems. The first publication that we presented in this work explains how a CNN classifier can be used to guide practitioners to potentially mislabeled samples for data correction. Net2Vis, the second publication that was presented, provides means to automatically generate publication-tailored CNN architecture visualizations for more unified and less time-consuming research communication. Finally, exploRNN, which provides an interactive learning environment for RNNs, can reduce the cognitive resources required during learning and provides a learning experience that is more enjoyable and fun.

After we obtained experience in supporting practitioners with such introspection-based visualization interfaces, we noticed that there was still a gap that prevented a broad adoption of existing visualization

interfaces. Thus, to bring these diverse visualization interfaces closer to the practitioners that need them, we developed Symphony. With the Symphony framework, visualization interfaces can be reused, combined, explored, and shared, fostering collaborative ML analyses.

While there are still many open research questions, our publications show how visualization can help many aspects of ML introspection become more accessible, creating a broader and deeper understanding of different areas of ML systems. Furthermore, we made first steps towards unifying the landscape of ML interfaces, which is important to make the work of the visualization community more visible, usable, and applicable for ML practitioners.

BIBLIOGRAPHY

- [Ade+18] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. “Sanity checks for saliency maps.” In: *Advances in neural information processing systems* 31 (2018).
- [Ade+20] Julius Adebayo, Michael Muelly, Ilaria Liccardi, and Been Kim. “Debugging tests for model explanations.” In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. 2020, pp. 700–712.
- [AH+18] Lisa Anne Hendricks, Kaylee Burns, Kate Saenko, Trevor Darrell, and Anna Rohrbach. “Women also snowboard: Overcoming bias in captioning models.” In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 771–787.
- [Arn+19] Matthew Arnold, Rachel KE Bellamy, Michael Hind, Stephanie Houde, Sameep Mehta, Aleksandra Mojsilović, Ravi Nair, K Natesan Ramamurthy, Alexandra Olteanu, David Piorkowski, et al. “FactSheets: Increasing trust in AI services through supplier’s declarations of conformaty.” In: *IBM Journal of Research and Development* 63.4/5 (2019), pp. 6–1.
- [AIEC20] High-Level Expert Group on Artificial Intelligence European Commission. *Assessment List for Trustworthy Artificial Intelligence (ALTAI)*. 2020. URL: <https://ec.europa.eu/digital-single-market/en/news/assessment-list-trustworthy-artificial-intelligence-alta-self-assessment> (visited on 11/27/2020).
- [BF18] Emily M Bender and Batya Friedman. “Data statements for natural language processing: Toward mitigating system bias and enabling better science.” In: *Transactions of the Association for Computational Linguistics* 6 (2018), pp. 587–604.
- [Blo+56] Benjamin S Bloom et al. “Taxonomy of educational objectives. Vol. 1: Cognitive domain.” In: *New York: McKay* 20 (1956), p. 24.
- [BOH11] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. “D³ data-driven documents.” In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), pp. 2301–2309.

- [Bri+17] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. "Massive Exploration of Neural Machine Translation Architectures." In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2017, pp. 1442–1451.
- [Bro+96] John Brooke et al. "SUS-A quick and dirty usability scale." In: *Usability evaluation in industry* 189.194 (1996), pp. 4–7.
- [BG18] Joy Buolamwini and Timnit Gebru. "Gender shades: Intersectional accuracy disparities in commercial gender classification." In: *Conference on fairness, accountability and transparency*. PMLR, 2018, pp. 77–91.
- [Cab+21] Ángel Alexander Cabrera, Abraham J Druck, Jason I Hong, and Adam Perer. "Discovering and validating ai errors with crowdsourced failure reports." In: *Proceedings of the ACM on Human-Computer Interaction* 5.CSCW2 (2021), pp. 1–22.
- [Cab+19] Ángel Alexander Cabrera, Will Epperson, Fred Hohman, Minsuk Kahng, Jamie Morgenstern, and Duen Horng Chau. "FairVis: Visual analytics for discovering intersectional bias in machine learning." In: *2019 IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE, 2019, pp. 46–56.
- [Cam+20] Nick Cammarata, Shan Carter, Gabriel Goh, Chris Olah, Michael Petrov, Ludwig Schubert, Chelsea Voss, Ben Egan, and Swee Kiat Lim. "Thread: Circuits." In: *Distill* (2020). URL: <https://distill.pub/2020/circuits>.
- [Car+19] Shan Carter, Zan Armstrong, Ludwig Schubert, Ian Johnson, and Chris Olah. "Activation Atlas." In: *Distill* (2019). URL: <https://distill.pub/2019/activation-atlas>.
- [Che+21] Changjian Chen, Jing Wu, Xiaohan Wang, Shouxing Xiang, Song-Hai Zhang, Qifeng Tang, and Shixia Liu. "Towards better caption supervision for object detection." In: *IEEE Transactions on Visualization and Computer Graphics* 28.4 (2021), pp. 1941–1954.
- [Chi+19] Hao-Tien Lewis Chiang, Aleksandra Faust, Marek Fiser, and Anthony Francis. "Learning navigation behaviors end-to-end with autorl." In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 2007–2014.
- [Cho+15] François Chollet et al. *Keras*. <https://keras.io>. 2015.
- [Chu+16] Sunghyo Chung, Sangho Suh, Cheonbok Park, Kyeongpil Kang, Jaegul Choo, and Bum Chul Kwon. "Revacnn: Real-time visual analytics for convolutional neural network." In: *KDD 16 Workshop on Interactive Data Exploration and Analytics*. 2016.

- [DJL21] Alex J DeGrave, Joseph D Janizek, and Su-In Lee. “AI for radiographic COVID-19 detection selects shortcuts over signal.” In: *Nature Machine Intelligence* 3.7 (2021), pp. 610–619.
- [DS22] Jean-Stanislas Denain and Jacob Steinhardt. “Auditing Visualizations: Transparency Methods Struggle to Detect Anomalous Behavior.” In: *arXiv preprint arXiv:2206.13498* (2022).
- [Din18] Weiguang Ding. *Draw Convnet*. https://github.com/gwding/draw_convnet. 2018.
- [DVK17] Finale Doshi-Velez and Been Kim. “Towards a rigorous science of interpretable machine learning.” In: *arXiv preprint arXiv:1702.08608* (2017).
- [Elh+21] Nelson Elhage et al. “A Mathematical Framework for Transformer Circuits.” In: *Transformer Circuits Thread* (2021). URL: <https://transformer-circuits.pub/2021/framework/index.html>.
- [Erh+09] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. “Visualizing higher-layer features of a deep network.” In: *University of Montreal 1341.3* (2009), p. 1.
- [FV17] Ruth C Fong and Andrea Vedaldi. “Interpretable explanations of black boxes by meaningful perturbation.” In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 3429–3437.
- [Für+20] Maximilian Förster, Mathias Klier, Kilian Kluge, and Irina Sigler. “Evaluating explainable Artificial intelligence – What users really appreciate.” In: *Proceedings of the European Conference on Information Systems* (2020).
- [FCS21] Jules Françoise, Baptiste Caramiaux, and Téo Sanchez. “Marcelle: Composing interactive machine learning workflows and interfaces.” In: *The 34th Annual ACM Symposium on User Interface Software and Technology*. 2021, pp. 39–53.
- [Geb+21] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé Iii, and Kate Crawford. “Datasheets for datasets.” In: *Communications of the ACM* 64.12 (2021), pp. 86–92.
- [Gle+20] Michael Gleicher, Aditya Barve, Xinyi Yu, and Florian Heimerl. “Boxer: Interactive comparison of classifier results.” In: *Computer Graphics Forum*. Vol. 39. 3. Wiley Online Library, 2020, pp. 181–193.
- [Gsc17] David Gschwend. *Netscope Quickstart*. <http://dgschwend.github.io/netscope/quickstart.html>. 2017.

- [Guo13] Philip J Guo. “Online python tutor: embeddable web-based program visualization for cs education.” In: *Proceeding of the 44th ACM technical symposium on Computer science education*. 2013, pp. 579–584.
- [GWZ15] Philip J Guo, Jeffery White, and Renan Zanelatto. “Codechella: Multi-user program visualizations for real-time tutoring and collaborative learning.” In: *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2015, pp. 79–87.
- [Ham+20] Koichi Hamada, Fuyuki Ishikawa, Satoshi Masuda, Tomoyuki Myojin, Yasuharu Nishi, Hideto Ogawa, Takahiro Toku, Susumu Tokumoto, Kazunori Tsuchiya, Yasuhiro Ujita, et al. “Guidelines for Quality Assurance of Machine Learning-based Artificial Intelligence.” In: *SEKE*. 2020, pp. 335–341.
- [Har15] Adam W Harley. “An interactive node-link visualization of convolutional neural networks.” In: *International Symposium on Visual Computing*. Springer, 2015, pp. 867–877.
- [Hoh+18] Fred Hohman, Minsuk Kahng, Robert Pienta, and Duen Horng Chau. “Visual analytics in deep learning: An interrogative survey for the next frontiers.” In: *IEEE transactions on visualization and computer graphics* 25.8 (2018), pp. 2674–2693.
- [Hoh+20] Fred Hohman, Kanit Wongsuphasawat, Mary Beth Kery, and Kayur Patel. “Understanding and visualizing data iteration in machine learning.” In: *Proceedings of the 2020 CHI conference on human factors in computing systems*. 2020, pp. 1–13.
- [HHN20] Sarah Holland, Ahmed Hosny, and Sarah Newman. “The dataset nutrition label.” In: *Data Protection and Privacy, Volume 12: Data Protection and Democracy* 12 (2020), p. 1.
- [HBo07] Christopher D Hundhausen and Jonathan L Brown. “What You See Is What You Code: A “live” algorithm development and visualization environment for novice learners.” In: *Journal of Visual Languages & Computing* 18.1 (2007), pp. 22–47.
- [HDS02] Christopher D Hundhausen, Sarah A Douglas, and John T Stasko. “A meta-study of algorithm visualization effectiveness.” In: *Journal of Visual Languages & Computing* 13.3 (2002), pp. 259–290.
- [Hun07] J. D. Hunter. “Matplotlib: A 2D graphics environment.” In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95.

- [Inc21a] Google Inc. *Know Your Data*. 2021. URL: <https://knowyourdata.withgoogle.com/> (visited on 08/30/2021).
- [Inc21b] Streamlit Inc. *Streamlit*. 2021. URL: <https://streamlit.io/> (visited on 07/29/2021).
- [Jia+14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. “Caffe: Convolutional architecture for fast feature embedding.” In: *Proceedings of the 22nd ACM international conference on Multimedia*. 2014, pp. 675–678.
- [JIV19] Anna Jobin, Marcello Ienca, and Effy Vayena. “The global landscape of AI ethics guidelines.” In: *Nature Machine Intelligence* 1.9 (2019), pp. 389–399.
- [JM15] Michael I Jordan and Tom M Mitchell. “Machine learning: Trends, perspectives, and prospects.” In: *Science* 349.6245 (2015), pp. 255–260.
- [Jup21] Jupyter. *IPyWidgets*. 2021. URL: <https://ipywidgets.readthedocs.io/en/stable/> (visited on 08/31/2021).
- [Kah+18a] Minsuk Kahng, Pierre Y. Andrews, Aditya Kalro, and Duen Horng Polo Chau. “ActiVis: Visual exploration of industry-scale deep neural network models.” In: *IEEE Transactions on Visualization and Computer Graphics* 24 (1 2018), pp. 88–97.
- [Kah+18b] Minsuk Kahng, Nikhil Thorat, Duen Horng Polo Chau, Fernanda B Viégas, and Martin Wattenberg. “Gan lab: Understanding complex deep generative models using interactive visual experimentation.” In: *IEEE transactions on visualization and computer graphics* 25.1 (2018), pp. 1–11.
- [KSA11] Faiza Khan Khattak and Ansaf Salleb-Aouissi. “Quality control of crowd labeling through expert evaluation.” In: *Proceedings of the NIPS 2nd Workshop on Computational Social Science and the Wisdom of Crowds*. Vol. 2. 2011, p. 5.
- [KLA17] Ashish Khetan, Zachary C Lipton, and Anima Anandkumar. “Learning from noisy singly-labeled data.” In: *arXiv preprint arXiv:1712.04577* (2017).
- [Koe+19] Laura Koesten, Emilia Kacprzak, Jeni Tennison, and Elena Simperl. “Collaborative practices with structured data: Do tools support what users need?” In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 2019, pp. 1–14.
- [Kri+09] Alex Krizhevsky et al. “Learning multiple layers of features from tiny images.” In: (2009).

- [Kuz+20] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, et al. “The open images dataset v4.” In: *International Journal of Computer Vision* 128.7 (2020), pp. 1956–1981.
- [LLC21] Tableau Software LLC. *Tableau*. 2021. URL: <https://www.tableau.com/> (visited on 07/29/2021).
- [LC10] Yann LeCun and Corinna Cortes. “MNIST handwritten digit database.” In: (2010). URL: <http://yann.lecun.com/exdb/mnist/>.
- [Lea11] Matthew Lease. “On quality control and machine learning in crowdsourcing.” In: *Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence*. 2011.
- [Len18] Alex Lenail. NN-SVG. <https://github.com/zfrenchee/NN-SVG>. 2018.
- [Liu+18] Shixia Liu, Changjian Chen, Yafeng Lu, Fangxin Ouyang, and Bin Wang. “An interactive method to improve crowd-sourced annotations.” In: *IEEE transactions on visualization and computer graphics* 25.1 (2018), pp. 235–245.
- [MV20] Ričards Marcinkevičs and Julia E Vogt. “Interpretability and explainability: A machine learning zoo mini-tour.” In: *arXiv preprint arXiv:2012.01805* (2020).
- [McI+18] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. “UMAP: Uniform Manifold Approximation and Projection.” In: *Journal of Open Source Software* 3.29 (2018).
- [Mes+22] Christian Meske, Enrico Bunde, Johannes Schneider, and Martin Gersch. “Explainable artificial intelligence: objectives, stakeholders, and future research opportunities.” In: *Information Systems Management* 39.1 (2022), pp. 53–63.
- [Mit+19] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. “Model cards for model reporting.” In: *Proceedings of the Conference on Fairness, Accountability, and Transparency*. 2019, pp. 220–229.
- [MZR21] Sina Mohseni, Niloofar Zarei, and Eric D Ragan. “A multidisciplinary survey and framework for design and evaluation of explainable AI systems.” In: *ACM Transactions on Interactive Intelligent Systems (TiiS)* 11.3-4 (2021), pp. 1–45.

- [MCB20] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. "Interpretable machine learning—a brief history, state-of-the-art and challenges." In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2020, pp. 417–431.
- [NOPF10] David F Nettleton, Albert Orriols-Puig, and Albert Fornells. "A study of the effect of different types of noise on the precision of supervised learning techniques." In: *Artificial intelligence review* 33.4 (2010), pp. 275–306.
- [NQ17] Andrew P Norton and Yanjun Qi. "Adversarial-Playground: A visualization suite showing how adversarial examples fool deep learning." In: *2017 IEEE Symposium on Visualization for Cyber Security (VizSec)*. IEEE, 2017, pp. 1–4.
- [OEC19] OECD. *Recommendation of the Council on Artificial Intelligence*. 2019. URL: <https://legalinstruments.oecd.org/en/instruments/OECD-LEGAL-0449> (visited on 12/11/2020).
- [Obe+19] Ziad Obermeyer, Brian Powers, Christine Vogeli, and Sendhil Mullainathan. "Dissecting racial bias in an algorithm used to manage the health of populations." In: *Science* 366.6464 (2019), pp. 447–453.
- [OMS17] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. "Feature Visualization." In: *Distill* (2017). URL: <https://distill.pub/2017/feature-visualization>.
- [Ola+18] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. "The Building Blocks of Interpretability." In: *Distill* (2018). URL: <https://distill.pub/2018/building-blocks>.
- [Par+21] Cheonbok Park, Soyoung Yang, Inyoup Na, Sunghyo Chung, Sungbok Shin, Bum Chul Kwon, Deokgun Park, and Jaegul Choo. "VATUN: Visual Analytics for Testing and Understanding Convolutional Neural Networks." In: *Eurographics Conference on Visualization (EuroVis)-Short Papers*. The Eurographics Association. 2021.
- [Pec+06] Mykola Pechenizkiy, Alexey Tsymbal, Seppo Puuronen, and Oleksandr Pechenizkiy. "Class noise and supervised learning in medical domains: The effect of feature extraction." In: *19th IEEE symposium on computer-based medical systems (CBMS'06)*. IEEE, 2006, pp. 708–713.

- [PRS18] Nina Poerner, Benjamin Roth, and Hinrich Schütze. "Interpretable Textual Neuron Representations for NLP." In: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Association for Computational Linguistics, 2018, pp. 325–327.
- [PAC18] Ivens Portugal, Paulo Alencar, and Donald Cowan. "The use of machine learning algorithms in recommender systems: A systematic review." In: *Expert Systems with Applications* 97 (2018), pp. 205–227.
- [Rag20] Prabhakar Raghavan. *How AI is powering a more helpful Google*. 2020. URL: <https://blog.google/products/search/search-on/> (visited on 05/24/2022).
- [RDK19] Alvin Rajkomar, Jeffrey Dean, and Isaac Kohane. "Machine learning in medicine." In: *New England Journal of Medicine* 380.14 (2019), pp. 1347–1358.
- [Ras+22] Gabrielle Ras, Ning Xie, Marcel van Gerven, and Derek Doran. "Explainable Deep Learning: A Field Guide for the Uninitiated." In: *Journal of Artificial Intelligence Research* 73 (2022), pp. 329–397.
- [Ras+10] Cyrus Rashtchian, Peter Young, Micah Hodosh, and Julia Hockenmaier. "Collecting image annotations using amazon's mechanical turk." In: *Proceedings of the NAACL HLT 2010 workshop on creating speech and language data with Amazon's Mechanical Turk*. 2010, pp. 139–147.
- [Roe18] Lutz Roeder. *Netron*. <https://github.com/lutzroeder/Netron>. 2018.
- [RC21] Priya Roy and Chandreyee Chowdhury. "A survey of machine learning techniques for indoor localization and navigation systems." In: *Journal of Intelligent & Robotic Systems* 101.3 (2021), pp. 1–34.
- [Rud19] Cynthia Rudin. "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead." In: *Nature Machine Intelligence* 1.5 (2019), pp. 206–215.
- [Sar+21] Cristina L Saratxaga, Jorge Bote, Juan F Ortega-Morán, Artzai Picón, Elena Terradillos, Nagore Arbide del Río, Nagore Andraka, Estibaliz Garrote, and Olga M Conde. "Characterization of optical coherence tomography images for colon lesion differentiation under deep learning." In: *Applied Sciences* 11.7 (2021), p. 3119.

- [Sat+15] Arvind Satyanarayan, Ryan Russell, Jane Hoffswell, and Jeffrey Heer. “Reactive vega: A streaming dataflow architecture for declarative interactive visualization.” In: *IEEE Transactions on Visualization and Computer Graphics* 22.1 (2015), pp. 659–668.
- [Sav22] Neil Savage. “Breaking into the black box of artificial intelligence.” In: *Nature* (2022).
- [She+21] Hong Shen, Wesley H Deng, Aditi Chattopadhyay, Zhiwei Steven Wu, Xu Wang, and Haiyi Zhu. “Value cards: An educational toolkit for teaching social impacts of machine learning through deliberation.” In: *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*. 2021, pp. 850–861.
- [Shn03] Ben Shneiderman. “The eyes have it: A task by data type taxonomy for information visualizations.” In: *The craft of information visualization*. Elsevier, 2003, pp. 364–371.
- [Smi+17] Daniel Smilkov, Shan Carter, D Sculley, Fernanda B Viégas, and Martin Wattenberg. “Direct-manipulation visualization of deep networks.” In: *arXiv preprint arXiv:1708.03788* (2017).
- [Smi+19] Daniel Smilkov, Nikhil Thorat, Yannick Assogba, Charles Nicholson, Nick Kreeger, Ping Yu, Shanqing Cai, Eric Nielsen, David Soegel, Stan Bileschi, et al. “Tensorflow.js: Machine learning for the web and beyond.” In: *Proceedings of Machine Learning and Systems* 1 (2019), pp. 309–321.
- [Sno18] Jacob Snow. “Amazon’s face recognition falsely matched 28 members of Congress with mugshots.” In: *American Civil Liberties Union* 28 (2018).
- [SC18] Pierre Stock and Moustapha Cisse. “Convnets and imagenet beyond accuracy: Understanding mistakes and uncovering biases.” In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 498–512.
- [Sto+22] Veda C. Storey, Roman Lukyanenko, Wolfgang Maass, and Jeffrey Parsons. “Explainable AI.” In: *Communications of the ACM* (2022).
- [Str+21] Dirk Streeb, Mennatallah El-Assady, Daniel A Keim, and Min Chen. “Why visualize? Arguments for visual support in decision making.” In: *IEEE Computer Graphics and Applications* 41.2 (2021), pp. 17–22.
- [STY17] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. “Axiomatic attribution for deep networks.” In: *International conference on machine learning*. PMLR, 2017, pp. 3319–3328.

- [Swa+20] Swabha Swayamdipta, Roy Schwartz, Nicholas Lourie, Yizhong Wang, Hannaneh Hajishirzi, Noah A. Smith, and Yejin Choi. “Dataset Cartography: Mapping and Diagnosing Datasets with Training Dynamics.” In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2020, pp. 9275–9293.
- [Swe11] John Sweller. “Cognitive load theory.” In: *Psychology of learning and motivation*. Vol. 55. Elsevier, 2011, pp. 37–76.
- [Ten+20] Ian Tenney, James Wexler, Jasmijn Bastings, Tolga Bolukbasi, Andy Coenen, Sebastian Gehrmann, Ellen Jiang, Mahima Pushkarna, Carey Radebaugh, Emily Reif, et al. “The Language Interpretability Tool: Extensible, Interactive Visualizations and Analysis for NLP Models.” In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. 2020, pp. 107–118.
- [Uch19] Yusuke Uchida. *Convnet Drawer*. <https://github.com/yu4u/convnet-drawer>. 2019.
- [Usa+11] Yu Usami, Han-Cheol Cho, Naoaki Okazaki, and Jun’ichi Tsujii. “Automatic acquisition of huge training data for bio-medical named entity recognition.” In: *Proceedings of BioNLP 2011 Workshop*. 2011, pp. 65–73.
- [Van+18] Jacob VanderPlas, Brian Granger, Jeffrey Heer, Dominik Moritz, Kanit Wongsuphasawat, Arvind Satyanarayan, Eitan Lees, Ilia Timofeev, Ben Welsh, and Scott Sievert. “Altair: Interactive statistical visualizations for python.” In: *Journal of Open Source Software* 3 (32 2018), p. 1057.
- [Ver+22] Ina Vernikouskaya, Dagmar Bertsche, Wolfgang Rottbauer, and Volker Rasche. “Deep learning-based framework for motion-compensated image fusion in catheterization procedures.” In: *Computerized Medical Imaging and Graphics* 98 (2022), p. 102069.
- [WMR17] Sandra Wachter, Brent Mittelstadt, and Chris Russell. “Counterfactual explanations without opening the black box: Automated decisions and the GDPR.” In: *Harv. JL & Tech.* 31 (2017), p. 841.
- [WLW20] Linda Wang, Zhong Qiu Lin, and Alexander Wong. “Covid-net: A tailored deep convolutional neural network design for detection of covid-19 cases from chest x-ray images.” In: *Scientific Reports* 10.1 (2020), pp. 1–12.

- [Wan+22] Qianwen Wang, Kexin Huang, Payal Chandak, Marinka Zitnik, and Nils Gehlenborg. "Towards Usable Explanations: Extending the Nested Model of Visualization Design for User-Centric XAI." In: (2022).
- [Wan+20] Qianwen Wang, Zhenhua Xu, Zhutian Chen, Yong Wang, Shixia Liu, and Huamin Qu. "Visual analysis of discrimination in machine learning." In: *IEEE Transactions on Visualization and Computer Graphics* 27.2 (2020), pp. 1470–1480.
- [Wan+04] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. "Image quality assessment: from error visibility to structural similarity." In: *IEEE transactions on image processing* 13.4 (2004), pp. 600–612.
- [WGY18] Gail Weiss, Yoav Goldberg, and Eran Yahav. "On the Practical Computational Power of Finite Precision RNNs for Language Recognition." In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2018, pp. 740–745.
- [Wex+20] James Wexler, Mahima Pushkarna, Tolga Bolukbasi, Martin Wattenberg, Fernanda Viegas, and Jimbo Wilson. "The what-if tool: Interactive probing of machine learning models." In: *IEEE Transactions on Visualization and Computer Graphics* 26 (1 2020), pp. 56–65.
- [WHM19] Benjamin Wilson, Judy Hoffman, and Jamie Morgenstern. "Predictive inequity in object detection." In: *arXiv preprint arXiv:1902.11097* (2019).
- [Won+17] Kanit Wongsuphasawat, Daniel Smilkov, James Wexler, Jimbo Wilson, Dandelion Mane, Doug Fritz, Dilip Krishnan, Fernanda B Viégas, and Martin Wattenberg. "Visualizing dataflow graphs of deep learning models in tensorflow." In: *IEEE transactions on visualization and computer graphics* 24.1 (2017), pp. 1–12.
- [Wor+19] Mitchell Wortsman, Kiana Ehsani, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. "Learning to learn how to learn: Self-adaptive visual navigation using meta-learning." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 6750–6759.
- [Xia+19] Shouxing Xiang, Xi Ye, Jiazhi Xia, Jing Wu, Yang Chen, and Shixia Liu. "Interactive correction of mislabeled training data." In: *2019 IEEE Conference on Visual Analytics Science and Technology (VAST)*. 2019, pp. 57–68.

- [Yan+20] Fumeng Yang, Zhuanyi Huang, Jean Scholtz, and Dustin L Arendt. “How do visual explanations foster end users’ appropriate trust in machine learning?” In: *Proceedings of the 25th International Conference on Intelligent User Interfaces*. 2020, pp. 189–201.
- [YLT18] Anders Ynnerman, Jonas Löwgren, and Lena Tibell. “Exploramation: A new science communication paradigm.” In: *IEEE computer graphics and applications* 38.3 (2018), pp. 13–20.
- [ZF14] Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks.” In: *European conference on computer vision*. Springer, 2014, pp. 818–833.
- [ZTF11] Matthew D Zeiler, Graham W Taylor, and Rob Fergus. “Adaptive deconvolutional networks for mid and high level feature learning.” In: *2011 international conference on computer vision*. IEEE, 2011, pp. 2018–2025.
- [Zha+21] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. “Understanding deep learning (still) requires rethinking generalization.” In: *Communications of the ACM* 64.3 (2021), pp. 107–115.
- [Zha+20] J M Zhang, M Harman, L Ma, and Y Liu. “Machine learning testing: Survey, landscapes and horizons.” In: *IEEE Transactions on Software Engineering* (2020).

Part II
PUBLICATIONS

CLASSIFIER-GUIDED VISUAL CORRECTION OF NOISY LABELS FOR IMAGE CLASSIFICATION TASKS

Alex Bäuerle, Heiko Neumann, and Timo Ropinski. "Classifier-Guided Visual Correction of Noisy Labels for Image Classification Tasks." In: *Computer Graphics Forum* 39.3 (2020), pp. 195–205

This work is published under the terms of the Creative Commons Attribution 4.0 License (CC BY 4.0), <https://creativecommons.org/licenses/by/4.0/>

Classifier-Guided Visual Correction of Noisy Labels for Image Classification Tasks

A. Bäuerle , H. Neumann , and T. Ropinski 

All authors are with Ulm University. E-mail: alex.baerle@uni-ulm.de.

Abstract

Training data plays an essential role in modern applications of machine learning. However, gathering labeled training data is time-consuming. Therefore, labeling is often outsourced to less experienced users, or completely automated. This can introduce errors, which compromise valuable training data, and lead to suboptimal training results. We thus propose a novel approach that uses the power of pretrained classifiers to visually guide users to noisy labels, and let them interactively check error candidates, to iteratively improve the training data set. To systematically investigate training data, we propose a categorization of labeling errors into three different types, based on an analysis of potential pitfalls in label acquisition processes. For each of these types, we present approaches to detect, reason about, and resolve error candidates, as we propose measures and visual guidance techniques to support machine learning users. Our approach has been used to spot errors in well-known machine learning benchmark data sets, and we tested its usability during a user evaluation. While initially developed for images, the techniques presented in this paper are independent of the classification algorithm, and can also be extended to many other types of training data.

CCS Concepts

• *Information systems* → *Expert systems*; • *Human-centered computing* → *User centered design; Information visualization;*

1. Introduction

While most of the latest breakthroughs in deep learning have been achieved by means of supervised algorithms, these algorithms have one essential limitation: they require large amounts of labeled training data. When learning image classification tasks, this means that a large set of correctly labeled images needs to be available [NOPF10, PTPP06]. Since the labeling process is time-consuming and labor-intensive, acquiring labeled training data is, however, a cumbersome process. To speed this process up, labeling is often outsourced to less experienced annotators or crowd workers, for instance via Amazon's Mechanical Turk [KLA17, RYHH10]. In the context of deep learning, crowd workers are human labor, getting paid for labeling large data sets. Sometimes, even automatic label assignment tools are used [UCOT11]. Unfortunately, such a label acquisition process usually leads to noisy labels, i.e., a training data set which contains many wrongly assigned labels. This can compromise training results [ZBH*16]. Thus, to be able to benefit from these approaches for training data acquisition, dedicated quality control mechanisms must be in place.

To address the problem of noisy labels, we propose a classifier-guided visual correction approach, which combines automatic error detection with interactive visual error correction (see Figure 1). To enable the automatic detection, we have systematically categorized

error types, that can be potentially present in noisy label data sets. Our categorization led to three such error types: *Class Interpretation Errors*, *Instance Interpretation Errors*, and *Similarity Errors*. Tailored towards these error types, we further introduce detection measures, which are based on the classifier's response. Therefore, we first train with the potentially noisy labels, and subsequently classify all training and validation images with the trained classifier. The classifier's response can then be analyzed using our error detection measures to guide the user to potential errors. These potential errors are visualized in a way that supports an interactive visual correction. To visually guide the user during the correction, we propose to employ linked list visualizations with importance sorting. By using our approach, the number of required inspections is bound by – and usually much lower than – the classification error, i.e., for a classifier that reaches an accuracy of 92%, only 8% of the data has to be reviewed at maximum. While this is the upper bound for investigated training samples per iteration, all samples that have already been inspected can additionally be ignored in the error detection process in future iterations. This means that for each subsequent iteration of data-cleanup, only those samples where the classifier disagrees with the label and that have not been already revisited need to be reviewed. Without our classifier-guided approach, instead, an inspection of the entire labeled data set would be necessary.

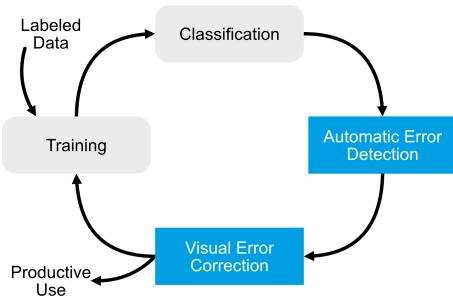


Figure 1: We propose a classifier-guided **Automatic Error Detection** for noisy labels, to visually guide the user to erroneous labels, which can then be inspected and corrected during the proposed **Visual Error Correction**. The two proposed components seamlessly integrate with standard machine learning workflows, as they operate downstream from **Training** and **Classification**. After a visual inspection, the classifier can be deployed for **Productive Use**, or trained again to be iteratively improved through the proposed process.

As illustrated in Figure 1, the proposed approach can be used iteratively to further improve classification accuracy, whereas users have to inspect fewer images for each subsequent iteration, as already inspected images do not require further consideration. While the contributions made in this paper address automatic error detection and visual error correction, no modifications are necessary for collecting labels, or training and testing the classifier, as our approach is to correct training data independent of the labeling or training process, allowing data experts to review data sets that have been fully labeled. This is in contrast to active learning, which modifies the label acquisition process during training [SOS92, Set10], as well as more recent fully automatic techniques, which modify the training process, and also reduce the amount of training data by sorting out noisy labels [TIYA18, LHZY18, HQJZ19]. We propose an error correction approach that is based solely on classification results of the trained model, and integrates seamlessly with modern deep learning workflows without reducing the size of the training data set.

To this end, we make the following contributions throughout this paper:

- Categorization of label error types potentially occurring in classification training data.
- Measures to identify error candidates by means of classifier response analysis.
- Interactive visual error guidance and correction by means of classifier result measure visualization.

We have realized these contributions within an interactive visualization system, with which we were able to identify errors in standard machine learning benchmark data sets, such as MNIST and CIFAR10 (see Figure 2). We have further evaluated this system, whereby our findings indicate, that it enables users to intuitively clean noisy label data in order to achieve higher classification accuracies.

2. Related Work

Work on handling noisy labels for datasets can be delineated into two main categories. On one side, some approaches aim at inspecting datasets, often through visualization. On the other, there are training setups that aim at providing robust classifiers that cope with noisy labels. The following will provide an overview of both those lines of research.

Data labeling. One area of deep learning where data labeling is a central aspect is active learning [Set10]. Here, candidates for labeling assignments are selected, often through a query-by-committee strategy, where the output of several classifiers is used to inform candidate selection [SOS92]. The line of work by Bernard et. al. [BHZ^{*}17, BZL^{*}18, BHR^{*}19] investigates how label acquisition in active learning scenarios can be improved. They also employ the classifier directly to suggest new items to be labeled and use dimensionality reduction techniques to visualize these proposed items and their distribution. What separates active learning from our work is, that active learning does not aim at improving noisy data sets, but rather works towards improving the labeling process itself. Thus, active learning is placed before label acquisition has been performed, while our approach is designed to work with readily labeled data sets.

There also exist numerous techniques to ensure a better quality of crowdsourced training data while labels are being generated [HKBE12, CAK17, Set11]. They use multiple workers [KH16], provide monetary benefits for good work and specialized task framing [RKK^{*}11], or select workers with predefined requirements [MHG15]. All of these approaches are focused on quality assurance while labels are acquired. Approaches to examining data quality after labeling through crowd services are analyzing how the worker interacted with the system [RK12, LCL^{*}19], or having workers review the work of other workers [HSC^{*}13]. A work

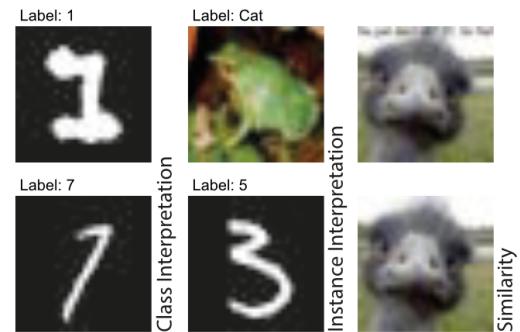


Figure 2: Examples of errors we discovered by applying our techniques to widely used machine learning benchmark data sets. On the left, one can see possible **Class Interpretation Errors**. While the top one was labeled as one, the bottom one was labeled to be a seven. The frog in the center is labeled as a cat and the three as a five, thus, single instances were clearly misinterpreted. On the right, one can see almost equal images. One might question if they should both be in the data set. (Original resolution of 32 by 32 for Cifar10/animals and 28 by 28 for MNIST/digits)

published by Chang et. al. combines multiple of these aspects to ensure data quality by grouping multiple workers and letting them interactively reason about label decisions [CAK17]. However, they do not incorporate the classifier feedback in their visualizations, which is the building block of our guidance system and can help reduce the samples to be revisited. Additionally, their techniques are only applicable if all annotations are present and can be assigned to individual workers. Yet, correcting labels for readily provided data sets where original labelers are not accessible anymore can be valuable to support already processed data sets. Current tools are targeted more towards analyzing worker performance than correcting already labeled data sets. However, it can often be of great value for domain experts to be able to validate and correct their training data themselves as sometimes the data is specific and cannot be perfectly labeled by laymen. Additionally, for all of these data improvement methods in the context of crowdsourcing, one needs to either hire more crowdworkers, refine the requirements or conduct a separate, second task to verify the generated labels, which comes with a greater investment of money and time during label acquisition [SPI08, IPSW14], and sometimes even makes crowdsourcing more expensive than conventional approaches [KLA17]. In these scenarios, it is therefore helpful if domain experts can review and resolve label errors quickly. Our approach is thus focused on correcting erroneous labels.

Visualization has been used for data cleaning in several publications, which shows how effective visualization can be when data is to be cleaned. Kandel et. al. worked on improving data by visually exploring data sets and directly manipulating them whenever a user spots a problem in the data [KPHH11, KPP*12]. Gschwandtner et. al. [GAM*14] as well as Arbesser et. al. [ASMP16] use visualization to clean up time-oriented data. Wilkinson developed measures and visualizations to detect and inspect outliers in data sets [Wil17]. However, these and related [PNMK16, WGS*13] tools are not tailored towards use with machine learning data sets, which often exceed the amount of data used in these contexts, contain labels that are to be corrected instead of direct data properties and offer additional guidance usable for visualization designs, such as classification results.

In a publication by Xiang et. al., visualization is directly used to improve the quality of neural network training data sets [XYX*19]. They use a projection of the whole high dimensional data set to define trusted items, which are then propagated to more items using an approach by Zhang et. al. [ZZW18]. However, while this approach combines human interaction with network-based label correction, they do not use the network predictions as guidance to potential errors. Similarly, Alsallakh et. al. [AJY*18] developed a visualization method to analyze class hierarchies in training scenarios. The purpose of this approach is to identify class hierarchies that are often confused by the classifier, and upon this knowledge, improve the classifier or label definitions. As a side-product, they were also able to find labeling errors in the data. However, their visualization design and especially the lack of tools to directly investigate and correct mislabeled samples shows, that this is not the main goal of their application.

Robust training. One way to approach noisy data sets is to train a classifier that is robust against such noisy labels. Here, some ap-

proaches rely on modifications of said classifier to introduce features that can filter noisy labels [TIYA18, ZS18, HQJZ19]. This introduces additional overhead and does not improve the general label quality so that the data set remains erroneous. Others rely on additional, clean data to filter for noisy samples [PRKM*17, HMWG18]. These methods remove potentially noisy labels from the data set entirely [NNL*19], or reduce the importance of potentially false labels for training [RLA*14, JZL*17, RZYU18], which might reduce diversity in the data set. Such approaches can help circumvent some of the downsides of data sets that contain labeling errors, however, they do not tackle the underlying problem. Cleaning up data sets is still fundamental, as this is the only way a data set can be reliably reused, shared and published. At the same time, these approaches effectively make the data set smaller, which is not desirable. Some of these approaches also require using adjusted classifiers, which is neither desirable nor easy to use, especially by data-experts who are less experienced in ML.

Other authors introduce additional label cleaning networks to be trained to remove or relabel potentially compromised samples [VAC*17, LHZY18]. Han et. al. even propose to use a self learning approach to clean up noisy labels using extracted features from the data points [HLW19], however, all these automatic approaches do not guarantee correct labels. They either reduce the data set size, require modified training with another classifier, or both. Additionally, they do not allow data-experts to verify their data sets.

We propose an approach to improve the training data set without having to look at every individual sample by using the classifier as a guide to mislabeled samples. Our user-centered approach does not only focus on the final classifier performance, but is also targeted at cleaning up the training data at the same time, as it does not simply reweight or remove training samples. As this permanently corrects training data, it additionally makes the data reusable, publishable, and shareable. Also, the approach we propose can directly be integrated into any training process, as it does not require any manipulation of the classifier or additional data. Users simply use their trained classifier for permanent data-cleanup. It additionally provides insights about the training data, e.g. which classes are typically confused, biased, or seen similar.

3. Automatic Error Detection

To be able to tailor the visual user guidance towards relevant errors in labeled data sets, a characterization is required to differentiate error types potentially occurring in such labeling scenarios. Based on a systematic analysis of the image labeling process, we have identified three such error types.

Whenever annotators assign an incorrect label to an image, this can stem from two fundamentally different problems. Either, they just mislabel the one image at hand, while they have in general understood the task; or they have a wrong mental image of a class, and thus assign incorrect labels to all data points of that class. While these are problems that occur during the labeling of data points, another source for corrupted data sets may already be the data acquisition process. Similar or equal data points are sometimes added to the data set more than once, which can shift the data-distribution

away from real-world scenarios. While the aforementioned error types mostly stem from human errors, the addition of highly similar data points can be a problem especially when automatically collecting data, e.g. from online sources. To summarize, noise in training data can be introduced when:

1. A labeler confuses two classes (Class Interpretation Error)
2. A labeler mislabels one data point (Instance Interpretation Error)
3. Data points get added to the data set multiple times (Similarity Error)

These error types all introduce unique challenges for how to resolve them. Nevertheless, this categorization also enables the invention of novel approaches, to guide the user to potential instances of these errors. Therefore, to suggest further inspection of labeled data points, we propose the following measures for the three error types:

1. Class Interpretation Error Score (Equation (1))
2. Instance Interpretation Error Score (Equation (2))
3. Similarity Error Score (Equation (3))

For the first two scores, we use the classification results in combination with the labels, which might be incorrect, as the basis for computing them. The Class Interpretation Error Score is computed for each label/classification (lbl/cls) subset of the data, whereas the Instance Interpretation Error Score is computed on individual instances. The Similarity Error Score is computed for each instance pair with the same classification. We assume that, although the labeled data may contain errors, the convolutional neural network (CNN) is still able to differentiate between different classes, such that in general incorrectly labeled data points get classified into their original class. This assumption has been tested on an intentionally corrupted data set, which is described in Section 5. Since this makes the classification result and the original label differ, these data points can be detected by looking at misclassifications in the data set. The similarity error score instead, can be calculated by exploiting similarity measures between training samples. As every part of the data-split can contain errors, we classify all samples in the data set once after the network has been trained. This includes train, test, and validation data, which can then subsequently be corrected. In the following, we introduce these three scores and their computation in detail.

3.1. Class Interpretation Errors

Class Interpretation Errors are introduced when data points from class a were assumed to be of class b by one, or few, of the labelers. This error type is conceptual, and leads to multiple or all labels assigned by one, or a few, labelers and belonging to class a ending up with the wrong label b (e.g., labelers considering gooses to be ducks throughout the entire data set). However, as long as the majority of data points are correctly labeled, our presented approach is able to guide to these errors, as the classifier will still be able to correctly classify most of the data points with incorrect labels, see Section 5. Fortunately, the fact that multiple data points are labeled incorrectly makes Class Interpretation Errors easy to detect. We make use of the amount of resulting misclassifications to find

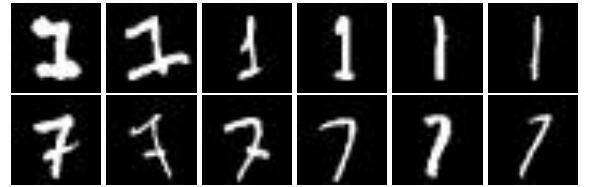


Figure 3: Images from the original MNIST data set (original resolution 28 by 28). The top row shows images labeled as one. The bottom row contains images labeled as seven. Here, Class Interpretation Errors might occur, since those digits are written differently in the US and Europe.

candidates for Class Interpretation Errors. Thus, we analyze lbl/cls combinations by the amount of missclassifications in them as:

$$CIES_{y,\hat{y}} = |\{x \mid x \in D, \text{argmax}(\text{cls}(x)) = y, \text{lbl}(x) = \hat{y}\}| \quad (1)$$

Which means that the Class Interpretation Error Score $CIES$ given a prediction class y and a ground truth class \hat{y} is defined as the cardinality of the subset of data points x in the data set D for which the classification result $\text{cls}(x)$ equals y and the label $\text{lbl}(x)$ equals \hat{y} . Thus, this measure is designed to analyze entire lbl/cls subsets of the data. An interesting occurrence of this type of error in the widely used MNIST data set is the interpretation of the North-American and European way of writing the digits '7' and '1', as shown in Figure 3.

3.2. Instance Interpretation Errors

When single items in the data set get labeled incorrectly, the situation is more difficult, as these errors cannot be spotted by analyzing the ratio of misclassifications of one lbl/cls pair. At the same time, however, they have less influence on classification accuracy as compared to Class Interpretation Errors. To provide means to identify and remove Instance Interpretation Errors, we employ the classification confidence as an indication for labeling errors. This works well for all probabilistic models, such as neural networks, where prediction probabilities are an implicit output. When data points are misclassified confidently, they might as well be incorrectly labeled. This can be used to guide the user to these samples in the data set. To enhance this guidance, we go one step further and analyze the relation of the classification confidence and the classification probability assigned to the ground-truth label of a data point. On these means, Alsallakh et. al [AJY*18] state:

[...] detecting mislabeled samples such as an image of a lion labeled as a monkey. We found such cases by inspecting misclassified samples having high prediction probability and low probability assigned to the ground-truth.

We, therefore, propose the following measure to guide users to these error types:

$$IIES_x = \frac{\max(\text{cls}(x)) + (1 - \text{cls}(x))_{\hat{y}}}{2} \quad (2)$$

Here, we calculate the Instance Interpretation Error Score *IIES* for a data point x as the normalized relation between the class that the classifier assigned the highest classification probability to, and the probability the classifier assigned to the ground-truth label \hat{y} . Thus, this score provides guidance on an individual instance level. This score is used as an indicator for how certain the classifier is wrt. the misclassification of a data point, and can be used to recognize potential labeling errors. Applying this approach to the widely used Cifar10 as well as the MNIST data set, revealed previously unknown labeling errors, which we discuss in Section 5.

3.3. Similarity Errors

When data points occur more than once in the labeled data set, this can lead to an unintended shift away from the real-world data distribution. Such errors can be introduced when data points are taken from online sources or when an overview of the data set is not always present during data acquisition. It is important to differentiate between intentionally augmented data and data points that might over-represent certain features during training, as data-augmentation can lead to better training results. However, having multiple similar data points unintentionally in the labeled data set can compromise the training results in multiple ways. When they are in the training set, a higher priority is assigned to this representation, which can lead to bias, where some features are considered more important than other features. This is a problem when this over-representation is not expected in the productive use of the classifier. When, in contrast, several instances are in the validation data set, validation accuracy has a higher variation depending on the correctness of the classification of these data points, which in turn might compromise the performance measure of the classifier. If similar data points exist across training and validation data sets, validation is performed on data points that the classifier has been trained on, which can also compromise validation results, and at the same time introduce bias to the training data. Gladly, guiding users to similar data points is also possible, as similarity measures can be computed for each pair of elements in the data set that are assigned the same classification result:

$$\begin{aligned} SES_{x_1, x_2} &= \text{sim}(x_1, x_2), \quad \text{for } x_1, x_2 \in M \\ M &:= \{x_1, x_2 \in D \mid x_1 \neq x_2, \text{argmax}(\text{cls}(x_1)) = \text{argmax}(\text{cls}(x_2))\} \end{aligned} \quad (3)$$

The Similarity Error Score *SES* for a pair of data points x_1, x_2 can be obtained using similarity measures, which exist for many types of data. The *SES* is calculated for all pairs of data points in the data set D that were classified into the same class, whereas the *sim* function represents a similarity measure for two data points. For images, this function could be the Structural Similarity Index Measure (SSIM) [WBSS04]. While proposing candidates with this measure is not complex, Similarity Errors require the most experience of all error types to be resolved, as highly similar images are not always a problem for training a classifier. They are only harmful if either, they do not represent the real-world distribution, or, if they originate from both the training and validation data sets because then, validation does not test generalizability. This makes

expert revision, which our approach is targeted towards, even more important.

By calculating the measures presented in this section, we are able to analyze the training data set and extract potential labeling errors using only the trained classifier. In our visual error correction approach, we make use of the suggested error characterization and treat these three error types differently, both, by calculating specialized error measures, and employing tailored visual guidance systems.

3.4. Workflow Integration

As we exploit a pre-trained classifier for error detection, a few considerations need to be made in order to integrate our approach into a standard classification workflow. Before analyzing the data set, the classifier needs to be trained. Here the classifier and the training process do not need to be altered at all. The user can then reinspect misclassified samples based on our proposed visual guidance. Additionally, if the number of data points to be reinspected is too small, experienced users can employ strict regularization or early stopping if they intend to control the number of training samples to reinspect, as the classification accuracy directly influences this number. To be able to use the classification results as guidance towards possible errors, we assume that the network still has enough correctly labeled data to learn from, and guide the user towards incorrect labels. While this assumption is likely to be true for most scenarios, if the data set is too small or contains too much noise, our approach will not function anymore as it relies on the classification results of the neural network.

To then get an idea about which items should be inspected again, all samples in the data set are classified once using the trained classifier. In a typical neural network setting, this would include training, test, and validation data, as all of them can contain errors. It is important to note that no evaluation of the model or further training is done at this point, so the data-split or training setup is not corrupted in any way. This way, each data point is assigned a probability distribution over all classes. We then present only misclassified samples through our visual guidance approach which we introduce in the next section. This way, the user has to look at far fewer items than if they would have to inspect all data points again. Our evaluation shows that this approach works well even when a large number of incorrect labels are present (see Section 5).

4. Visual Error Correction

While obtaining potential error candidates, as described above, is essential for improving training data sets, only through visual guidance users can *detect* potential errors, and *reason* about them. Our visual guidance approaches help to do this for all three error types that typically occur in labeling processes. Once errors have been reasoned about, they can directly be *resolved*. Again, the visual correction of data points, which involves the user tasks of detecting, reasoning about, and resolving potential errors, should be in line with the error types we propose. This interplay of user tasks and error types is shown in Table 1.

Table 1: User tasks involved when improving training data. The user has to first, detect potential errors, then try to reason them, before he/she can resolve them. The table shows how these tasks are completed for the three identified error types.

	Class Interpretation Error	Instance Interpretation Error	Similarity Error
Detect	Many samples misclassified from a to b	Samples confidently misclassified	Similar/ identical samples
Reason	Error or bad classifier performance?	Error or bad classifier performance?	Error or intentional?
Resolve	Reassign multiple labels	Reassign individual label	Remove item

4.1. Error Detection

Through the error measures we propose, it is possible to support users through visual guidance to the most critical items in the data set. For all three error types, users should have indications of which data points to review. In Section 3, we showed how candidates for these error types can be extracted from the data set based on classification results. Thus, the user should be guided to lbl/cls pairs that contain a large number of misclassifications for Class Interpretation Errors. For Instance Interpretation Errors, they should see which samples have been most confidently misclassified. Additionally, users should be given an indication of where to find very similar images to be able to resolve Similarity Errors. In the following, we present visual guidelines that support all of these requirements. To give users an overview of those measures, we propose a visualization of the data set that contains information about the amount, probability distribution, and similarity score for each lbl/cls pair. In line with our approach of guiding the user only to samples that the network misclassified, and thus might be labeled incorrectly, we only highlight misclassifications in this view, while correct classifications are depicted in the leftmost column. The resulting visualization can be seen in Figure 4.

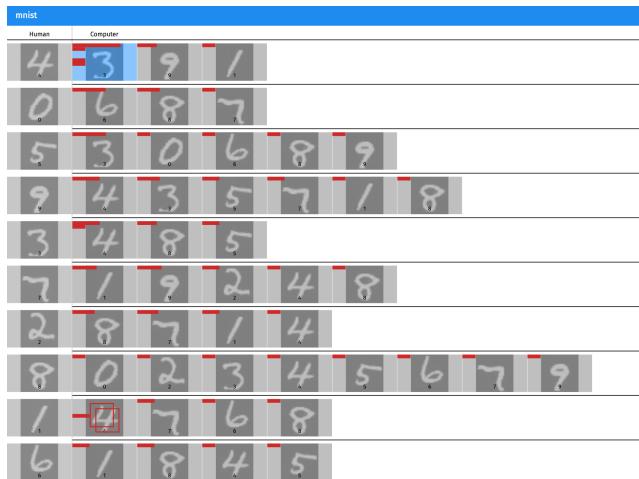


Figure 4: The list view of classifications shows problematic lbl/cls combinations at a glance. The number of misclassifications for each cell is encoded in the blue background. The red horizontal bars in each cell show, how confidently the images have been misclassified as computed through Equation (2). Visual separation of rows makes clear, that this list should be read from left to right. On the left, one can see cells for correctly classified samples.

We propose a visualization approach that employs a modified version of confusion matrices. To search for possible Class Interpretation Errors, users need to be able to investigate lbl/cls combinations containing many misclassifications. We support this requirement by sorting matrix cells based on the number of data points they contain, while the distribution of Instance Interpretation Scores is displayed within each cell. We first sort by the number of misclassifications across different labels (rows), before sorting classification results within each label (columns). This places the most critical classes at the top of this matrix. Additionally, we omit cells that do not contain any items, which removes unnecessary clutter and makes the visualization more sparse. In our implementation, we also highlight lbl/cls combinations with many misclassifications in blue, where the saturation of this color depends on the number of samples. This guides the visual attention of users directly to these, most critical lbl/cls combinations.

To also embed the IIES-distribution of those misclassifications in this overview, which is helpful for spotting potential Instance Interpretation Errors, we propose to show this distribution using horizontal bar-charts within each list item. Here, the y-position of the bars represents the IIES-distribution scaled from $1.0/\text{num_classes}$ (lowest bar) to 1.0 (top bar) while the length of the bars signals the number of items in an IIES-range.

The third user guidance system, which shows if similar items are present in a lbl/cls combination, is indicated by a duplicate icon within cells that contain highly similar data points. With these visual indicators across the entire data set, this view serves as an overview that guides users to all three error types we defined in Section 3.

Traditional approaches, such as confusion matrices [AJY*18, KH09] or the confusion wheel [AHH*14], which are commonly used to provide such an overview have major limitations for the task of spotting potential errors in the labeled data set. Confusion matrices always require understanding and combining both, the label and the classification axis, which proved to be too complex for depicting the source and destination for misclassifications when presented to domain experts [RAL*17]. At the same time, most of the confusion wheels screen real estate is taken up by class overviews and it provides no clear entry point. This renders both of these visual approaches suboptimal for guiding users to potential errors in the data set, which our approach is explicitly designed for.

4.2. Error Reasoning

When the user decides to inspect a potentially problematic lbl/cls combination, they naturally want to inspect individual data points and the distribution of data points in this subset of the data. This

way, they can reason about the potential errors to decide if they are problematic, and should be acted upon. To inspect one such lbl/cls combination in detail, users select one of the items in our overview visualization.

Reasoning about potential errors includes comparing samples, and extracting outliers as well as investigating similar samples for a lbl/cls combination. Thus, we propose to guide the user by visualizing similarity-based embeddings of the selected lbl/cls combination. Therefore, to inspect Instance Interpretation Errors, as well as Class Interpretation Errors, dimensionality reduction techniques that preserve high-dimensional relations are helpful. If many similar items have been misclassified, users can quickly reason about potential Class Interpretation Errors as these items, which differ from plain misclassifications, will cluster when dimensionality reduction is applied. On the other hand, outliers can be an indication for Instance Interpretation Errors, as can be seen in Figure 5. When dealing with images, we propose to use UMAP [MHM18] to show clusters of data points, as well as outliers in this lbl/cls combination, which can be seen in Figure 6. Here, either direct image pixels can be used as projection features. An even more sophisticated approach, which we used to generate these projections is, to use saliency visualizations of those images as a projection basis to also incorporate what the model looks for in these images. While labeling errors will not always be projected as outliers, users can iteratively remove items from the visualizations by confirming or changing their labels, which eventually reveals label errors. However, if there are few data points, or the user wishes to scan the data sequentially, there is also the option to switch to a grid-based view on the items. To also support the inspection of Similarity Errors, the most similar images per /c combination should additionally be presented to the user. In our implementation, those data points are shown below the projection-view.

Apart from showing data points with dimensionality reduction or sorted by similarity, their properties should also be inspectable in detail individually. This can further help to decide upon whether a proposed error candidate was indeed labeled incorrectly. Thus, in our proposed visualization approach, the final reasoning step on an individual data point level should be performed by selecting individual samples to view them in detail. Additionally, for selected items, we show the probability distribution that stems from the classifier response to provide the user with another tool to reason about a potential labeling error. In our implementation, enlarged images and classifier responses are shown on the right of the projection view (see Figure 5).

While each of these visual guides is targeted towards satisfying a specific user-need, in combination, they provide the visual tools necessary to reason about the three error types we propose.

4.3. Error Resolving

The final step in our proposed iterative data-correction approach is resolving potential errors that have been found within the data set. Once error candidates have been reasoned about, it is important to directly be able to resolve them. This can mean assigning new labels, but also confirming labels that are correct to remove items from the error correction process. For resolving Similarity Errors,

data points should also be permanently removable from the data set. To enable a correction, confirmation, and removal of labels for data points, we show actionable buttons on the lower right of the GUI (see Figure 6). Whenever data points are selected and subsequently resolved using these buttons, all visualizations are updated as resolved data points are removed from all guidance measure calculations and visualizations. The effect of this can be seen in Figure 5. Thus, by resolving error candidates, users can interactively process the visualizations and work their way through the data set until all error candidates are resolved, and thus removed from the guidance approach.

After one iteration of data-correction has been completed, users can reiterate and restart the process by training a new classifier on the partially cleaned data set (see Figure 1). With training a new classifier, proposed error candidates may change, and new error candidates can be inspected. For subsequent iterations, our proposed measure calculation and user guidance can thus be kept as is, with the exception that all previously relabeled, removed, or confirmed data points are not included in the guidance system anymore, as they have already been resolved.

In our approach, users are guided to confident misclassifications, large quantities of misclassifications, and almost equal images through a data set overview, which helps to investigate potential errors. To reason about error candidates, clustering mechanisms and outlier visualization are of great help. It is also essential to directly be able to act upon inspected items to remove them from the process. Through the translation of the three user tasks of detecting, reasoning about, and resolving potential labeling errors into our visualization guidelines, this approach can be implemented to fit any classifier as well as data type to be cleaned. Thus, our approach enables a user-centered data cleaning that utilizes the trained classifier to propose error candidates. The proposed visual design directly follows the principles of our approach to resolve the error types we introduced in Section 3, and obeys to the user tasks we defined for the visual correction process. Our implementation along with the user-study which we present in Section 5 shows, that our concepts are applicable to network-supported data-cleanup, and could be adopted in many application scenarios.

5. Evaluation

To test the proposed approach, we implemented a web-based application that realizes the proposed visualizations, and focuses on image data in combination with CNNs as classifiers. The general idea of using the classifier as a guide to potential labeling errors is, however, not limited to such data or classification algorithms. The following will present both, the application of our approach to renowned data sets, as well as a user study that tests the applicability of our approach.

5.1. Analyzing Benchmark Data Sets

Using our approach, we were able to spot errors in well-known machine-learning benchmark data sets. Here, we analyzed both, the Cifar10 [KH09], and MNIST [LC10] data sets.

MNIST. The MNIST data set [LC10] is one of the most popular machine learning benchmark data sets. It contains greyscale im-

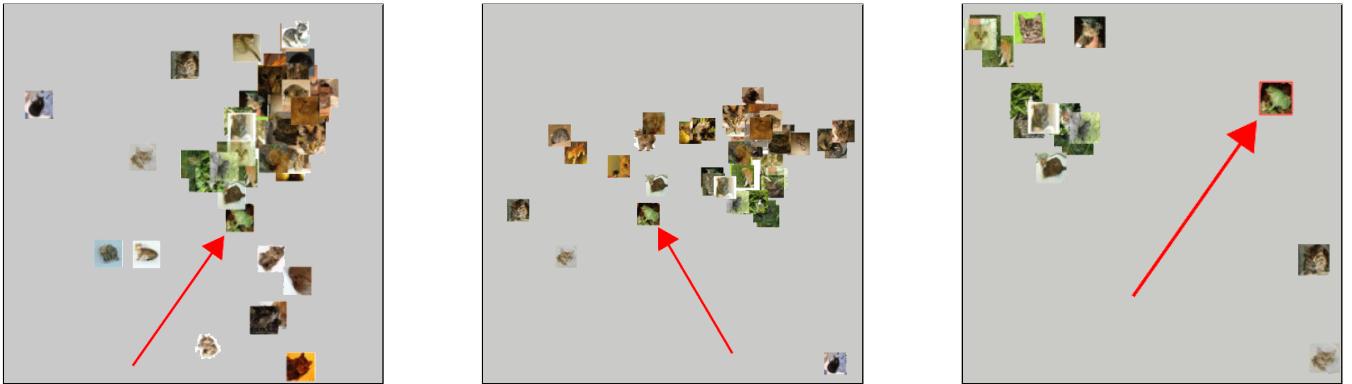


Figure 5: UMAP [MHM18] projection of the label cat and classification frog. One can see that dimensionality reduction helps to spot outliers in these lbl/cls combinations. The red arrows were added to indicate the position of the frog image. The three subsequent steps during interactive isolation of the frog wrongly labeled as cat show how after removing some data points, reprojecting the remaining data helps to isolate outliers. By iteratively removing outliers and through the nondeterministic nature of UMAP, the frog is embedded further away from the cats. (Images are from Cifar10, original resolution 32 by 32)

ages of handwritten digits from zero to nine with a size of 28 by 28 pixels. We used a simple architecture for training a CNN on that data set. It consisted of two convolutional layers, each followed by a max-pooling layer. For obtaining classification results on top of this embedding, one dense layer was used, followed by a dropout layer and the final softmax layer. Our classifier reached an accuracy of 99.3 percent. To review the data, we then inspected label classification pairs marked as suspicious in the overview visualization. Since only 0.7 percent of the data set was misclassified, our visual-

ization allowed us to only look at these images as potential errors. Thus, instead of looking at all 70,000 images in a file explorer, we had to look at only 490 misclassified images through a guided process.

When looking at the classes seven and one, some samples are almost impossible to distinguish while being from different classes. This can be seen in Figure 3. Here, different cultural interpretations of said classes might lead to Class Interpretation Errors. We found that the US-American versus European writing style of these digits might introduce problems to this data set. We also discovered individual instances that are mislabeled in the MNIST data set. Figure 2 shows a data point that clearly shows a three, but was labeled as a five. More of such examples can be found in our supplementary material.

Cifar10. The Cifar10 data set [KH09] consists of 32 by 32 pixel colored images from ten different classes. The model used for training on this data set was built by two blocks, each containing two convolutional layers followed by a pooling and a dropout layer. This was then followed up by two dense layers each also preceding a dropout layer, before the final softmax classification layer was added. With this intentionally simplistic network, we reached an accuracy of 77.13 percent, which is representative of real-world training scenarios on new, domain-specific data sets. Even with the classification comparably low accuracy we reached, we only had to look at 22.87 percent of the data.

As can be seen in Figure 5, for Cifar10, we were able to spot an image that was incorrectly labeled as cat, while showing a frog. When performing an in-detail inspection of the lbl/cls combination of the label cat and the classification frog, we found this incorrectly labeled image by iteratively removing outliers from the embedding visualization. Additionally, we found a set of very similar bird images as shown in Figure 7. While this is not a clear error in the data set, having multiple highly similar images of an ostrich in this data set is at least debatable.

Our approach is generally targeted towards domain-experts that get their data labeled and then train a classifier on that data or use

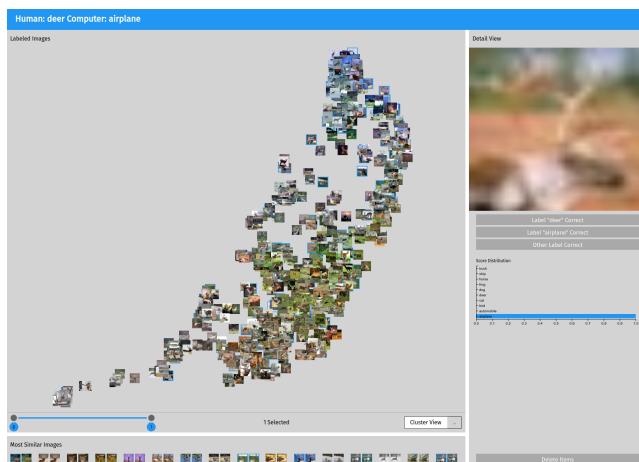


Figure 6: After gaining an overview of the classification results, the user can inspect the content of individual cells to analyze classification results in detail. Images are embedded by applying projection, e.g. UMAP. Filtering can be done by selecting IIES ranges. Once one or more images have been selected, the according probability distribution is visualized. Using the buttons on the right, users can change or confirm the label of the selected images. (Data set: Cifar10, resolution of images 32 by 32)



Figure 7: At the bottom of our in-detail visualization, we show pairs of similar images. The user can then decide whether these should stay in the labeled data set (e.g. in cases where data augmentation is used) or if they should be removed (in case of unwanted duplicates). The images show five similar images of a bird discovered in the Cifar10 data set (original resolution 32 by 32).

online services such as AutoML [Goo19] for training classifiers. Here, these control mechanisms are even more important, as data quality can be worse than in benchmark datasets. However, the fact that we were even able to find errors in two of the most popular benchmark data sets in the machine learning community shows how important approaches as the one we propose are.

5.2. Qualitative Evaluation

Based on our implementation, we additionally conducted a qualitative study to test the applicability of our approach. In our user study, 10 participants had to find and resolve errors in a labeled data set. Participants were recruited in a university setting, whereby out of the 10 participants, only two had experience with neural networks and none of them had seen or heard of our approach before. This shows, that no ML background is needed to use our visualization guidelines.

To generate a setup in which we could recruit participants in a university setting while still reflecting a real-world scenario, where data-experts would correct their noisy data set using our approach, we chose to use the MNIST data set in our study. This dataset requires no prior knowledge to review, as it consists of hand-drawn digits, which anyone can identify. To be able to verify which items have been changed by a participant, we corrupted the data set by introducing five errors of each type. For Class Interpretation Errors, we changed 1,400 images from nine to six, 700 images from one to four, 700 images from three to one, 350 images from eight to two and 175 images from seven to three. For Instance Interpretation Errors, we changed the labels of five images from different classes. With this, we tried to reflect real-world scenarios, where CIEs would introduce many more incorrect labels than IIEs. Similarity Errors were introduced by duplicating five images. In this study, we told the participants to remove all duplicates, as reasoning about if they are actually harmful could not be done in this setting. In total, we introduced 3,330 mislabeled images and five duplicates.

We then trained on this data set and visualized the results using our implementation. The classification accuracy for this manipulated data set was at 94.37 percent, hence, participants were only presented the 5.63 percent that were misclassified. This equals to about 4,000 out of the 70,000 images. We provided a short introduction of about 10 minutes which showed our idea for data-cleanup and explained the task, which was to resolve as many errors as possible in 15 minutes. We then let them use the approach we propose in this paper to resolve all errors they spotted.

With our similarity guidance, all participants were able to resolve all duplicates. We mainly attribute this to our visually prominent similarity indicators in the data set overview, and the fact, that the most similar items in a lbl/cls combination are shown separately when inspecting such combinations in detail. On average, every participant changed the labels of 2,902 images, of which only 27.5 were incorrectly changed. They thus managed to bring the number of incorrect labels down by 85.65 percent on average. This is a reduction to 477 errors from 3,330 after only one iteration of our approach. We then used the corrected data sets to train the classifier once again for each participant. On average, the validation accuracy rose to 99.05 percent, which shows the enormous impact of such data-cleanup. This shows the applicability of our approach to cleaning noisy labeled datasets.

Looking at the images that we initially considered as incorrectly changed also provided an interesting insight. When investigating them, we found that some of them seemed to be mislabeled in the original data set. The participants thus found new errors in the well-established MNIST data set by using our approach. Examples of these errors are included in the supplementary material.

To also evaluate the usability of our techniques, we asked the participants to rate the helpfulness of our approach. They had to rate the helpfulness of the visualizations from one, not helpful at all, to five, helped a lot, all of them rated the visualizations between four and five, with an average of 4.4. When asked what they found most helpful, most of them said the overview guidance approaches were helpful for spotting errors in the data set. Some additionally mentioned that it is also essential to be able to inspect individual samples for resolving errors. When asked what was bad and could be improved, many said that the latency was a problem. This, however, was a problem specific to the study setup and not to our approach perse.

As all participants were able to improve the data set by a large margin and thus greatly improve classification accuracy, this study shows that our proposed approach can, in fact, be a valuable tool to clean up labeled data. Also, as our participants stated, our guidance system helps users focus on critical training samples which greatly reduces samples that need to be reinspected.

6. Limitations

Currently, the approach we present within this work is limited to classification problems. For other problems, different error measures, as well as visual guidance systems, would have to be invented, which remains an open research question. Additionally, the error types we present within this paper cannot be applied outside the domain of classification problems. While our approach is model-agnostic and does not depend on the data that is used, the exemplar implementation we provide is focused on image-data in combination with CNNs. We propose three types of errors, which our analysis of labeling processes suggests are most common. However, one could think of other error cases, for example, if a labeler assigns completely random labels to all images. We did not include such error cases, as most of them could be filtered by traditional quality assurance methods. Nonetheless, investigating and handling other potential labeling errors remains an open challenge. Also, while matrix views are a common metaphor for getting

an overview of classification results for a data set, and our proposed matrix is even more condensed than others, it cannot scale indefinitely. We tested our approach with data sets containing up to more than 20 classes. A data set with 22 different classes containing animal skull X-Ray images, can be seen in our supplementary material. Yet, for data sets that contain even more classes, matrix views are not optimal. In this case, users would have to look at a subset of classes rather than viewing the whole class-pool right away. However, this is a general research question and is not tied only to our approach.

7. Conclusion

After introducing the problems that mislabeled training data for classification algorithms bring with them, we formulate a novel categorization of error types that typically occur in labeling settings for classification tasks. While there are other approaches that aim at improving noisy labels in training data, ours introduces the concept of using the trained classifier as a support for resolving these three different error types. The proposed visual correction approach can be performed at any point in the lifetime of a training data set, and permanently and reliably improves training data sets after the labeling process has been finished. Contrary to other approaches, our visual error correction tightly couples automated approaches with user interaction to ensure data quality. To model this visual correction approach, we define the user-tasks of first, detecting errors, then, reasoning about them, and finally resolving them, which users typically perform for cleaning up data sets. Our method fits especially well into the context of crowdsourced data-labels. With the ongoing automation of data acquisition, as well as classifier training, we imagine such data-cleanup techniques to be picked up in these contexts. Our approach could be a candidate to be plugged in directly into services such as AutoML [Goo19], where labels and classifiers can be obtained automatically, and correctly labeled data is crucial.

Acknowledgments

This work was funded by the Carl-Zeiss-Scholarship for PhD students.

References

- [AHH*14] ALSALLAKH B., HANBURY A., HAUSER H., MIKSCH S., RAUBER A.: Visual methods for analyzing probabilistic classification data. *IEEE transactions on visualization and computer graphics* 20, 12 (2014), 1703–1712. [6](#)
- [AJY*18] ALSALLAKH B., JOURABLOO A., YE M., LIU X., REN L.: Do convolutional neural networks learn class hierarchy? *IEEE transactions on visualization and computer graphics* 24, 1 (2018), 152–162. [3](#), [4](#), [6](#)
- [ASMP16] ARBESSER C., SPECHTENHAUSER F., MÜHLBACHER T., PIRINGER H.: Visplause: Visual data quality assessment of many time series using plausibility checks. *IEEE transactions on visualization and computer graphics* 23, 1 (2016), 641–650. [3](#)
- [BHR*19] BERNARD J., HUTTER M., RITTER C., LEHMANN M., SEDLMAIR M., ZEPPELZAUER M.: Visual analysis of degree-of-interest functions to support selection strategies for instance labeling. [2](#)
- [BHZ*17] BERNARD J., HUTTER M., ZEPPELZAUER M., FELLNER D., SEDLMAIR M.: Comparing visual-interactive labeling with active learning: An experimental study. *IEEE transactions on visualization and computer graphics* (2017). [2](#)
- [BZL*18] BERNARD J., ZEPPELZAUER M., LEHMANN M., MÜLLER M., SEDLMAIR M.: Towards user-centered active learning algorithms. In *Computer Graphics Forum* (2018), vol. 37, Wiley Online Library, pp. 121–132. [2](#)
- [CAK17] CHANG J. C., AMERSHI S., KAMAR E.: Revolt: Collaborative crowdsourcing for labeling machine learning datasets. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (2017), ACM, pp. 2334–2346. [2](#), [3](#)
- [GAM*14] GSCHWANDTNER T., AIGNER W., MIKSCH S., GÄRTNER J., KRIGLSTEIN S., POHL M., SUCHY N.: Timecleanser: A visual analytics approach for data cleansing of time-oriented data. In *Proceedings of the 14th international conference on knowledge technologies and data-driven business* (2014), pp. 1–8. [3](#)
- [Goo19] GOOGLE: Cloud AutoML BETA: Train high-quality custom machine learning models with minimal effort and machine learning expertise. <https://cloud.google.com/automl/>, October 2019. [Online; accessed 29-October-2019]. [9](#), [10](#)
- [HKBE12] HEIMERL F., KOCH S., BOSCH H., ERTL T.: Visual classifier training for text document retrieval. *IEEE Transactions on Visualization and Computer Graphics* 18, 12 (2012), 2839–2848. [2](#)
- [HLW19] HAN J., LUO P., WANG X.: Deep self-learning from noisy labels. *arXiv preprint arXiv:1908.02160v2* (2019). [3](#)
- [HMWG18] HENDRYCKS D., MAZEKA M., WILSON D., GIMPEL K.: Using trusted data to train deep networks on labels corrupted by severe noise. In *Advances in neural information processing systems* (2018), pp. 10456–10465. [3](#)
- [HQJZ19] HUANG J., QU L., JIA R., ZHAO B.: O2u-net: A simple noisy label detection approach for deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision* (2019), pp. 3326–3334. [2](#), [3](#)
- [HSC*13] HANSEN D. L., SCHONE P. J., COREY D., REID M., GEHRING J.: Quality control mechanisms for crowdsourcing: peer review, arbitration, & expertise at familysearch indexing. In *Proceedings of the 2013 conference on Computer supported cooperative work* (2013), ACM, pp. 649–660. [2](#)
- [IPSW14] IPEIROTIS P. G., PROVOST F., SHENG V. S., WANG J.: Repeated labeling using multiple noisy labelers. *Data Mining and Knowledge Discovery* 28, 2 (2014), 402–441. [3](#)
- [JZL*17] JIANG L., ZHOU Z., LEUNG T., LI L.-J., FEI-FEI L.: Men-tornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. *arXiv preprint arXiv:1712.05055* (2017). [3](#)
- [KH09] KRIZHEVSKY A., HINTON G.: Learning multiple layers of features from tiny images. [6](#), [7](#), [8](#)
- [KH16] KAIRAM S., HEER J.: Parting crowds: Characterizing divergent interpretations in crowdsourced annotation tasks. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing* (2016), ACM, pp. 1637–1648. [2](#)
- [KLA17] KHETAN A., LIPTON Z. C., ANANDKUMAR A.: Learning from noisy singly-labeled data. *arXiv preprint arXiv:1712.04577* (2017). [1](#), [3](#)
- [KPHH11] KANDEL S., PAEPCKE A., HELLERSTEIN J., HEER J.: Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2011), pp. 3363–3372. [3](#)
- [KPP*12] KANDEL S., PARIKH R., PAEPCKE A., HELLERSTEIN J. M., HEER J.: Profiler: Integrated statistical analysis and visualization for data quality assessment. In *Proceedings of the International Working Conference on Advanced Visual Interfaces* (2012), pp. 547–554. [3](#)

- [LC10] LECUN Y., CORTES C.: MNIST handwritten digit database. URL: <http://yann.lecun.com/exdb/mnist/> [cited 2016-01-14 14:24:11]. 7
- [LCL*19] LIU S., CHEN C., LU Y., OUYANG F., WANG B.: An interactive method to improve crowdsourced annotations. *IEEE transactions on visualization and computer graphics* 25, 1 (2019), 235–245. 2
- [LHZY18] LEE K.-H., HE X., ZHANG L., YANG L.: Cleannet: Transfer learning for scalable image classifier training with label noise. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 5447–5456. 2, 3
- [MHG15] MITRA T., HUTTO C. J., GILBERT E.: Comparing person-and process-centric strategies for obtaining quality data on amazon mechanical turk. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (2015), ACM, pp. 1345–1354. 2
- [MHM18] MCINNES L., HEALY J., MELVILLE J.: Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426* (2018). 7, 8
- [NNL*19] NGUYEN D. T., NGO T.-P.-N., LOU Z., KLAR M., BEGGEL L., BROX T.: Robust learning under label noise with iterative noise-filtering. *arXiv preprint arXiv:1906.00216* (2019). 3
- [NOPF10] NETTLETON D. F., ORRIOLS-PUIG A., FORNELLIS A.: A study of the effect of different types of noise on the precision of supervised learning techniques. *Artificial intelligence review* 33, 4 (2010), 275–306. 1
- [PNMK16] PARK J. H., NADEEM S., MIRHOSSEINI S., KAUFMAN A.: C 2 a: Crowd consensus analytics for virtual colonoscopy. In *2016 IEEE Conference on Visual Analytics Science and Technology (VAST)* (2016), IEEE, pp. 21–30. 3
- [PRKM*17] PATRINI G., ROZZA A., KRISHNA MENON A., NOCK R., QU L.: Making deep neural networks robust to label noise: A loss correction approach. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 1944–1952. 3
- [PTPP06] PECHENIZKIY M., TSYMBAL A., PUURONEN S., PECHENIZKIY O.: Class noise and supervised learning in medical domains: The effect of feature extraction. In *19th IEEE Symposium on Computer-Based Medical Systems (CBMS'06)* (2006), IEEE, pp. 708–713. 1
- [RAL*17] REN D., AMERSHI S., LEE B., SUH J., WILLIAMS J. D.: Squares: Supporting interactive performance analysis for multiclass classifiers. *IEEE transactions on visualization and computer graphics* 23, 1 (2017), 61–70. 6
- [RK12] RZESZOTARSKI J., KITTUR A.: Crowdscape: interactively visualizing user behavior and output. In *Proceedings of the 25th annual ACM symposium on User interface software and technology* (2012), ACM, pp. 55–62. 2
- [RKK*11] ROGSTADIUS J., KOSTAKOS V., KITTUR A., SMUS B., LAREDO J., VUKOVIC M.: An assessment of intrinsic and extrinsic motivation on task performance in crowdsourcing markets. *ICWSM 11* (2011), 17–21. 2
- [RLA*14] REED S., LEE H., ANGUELOV D., SZEGEDY C., ERHAN D., RABINOVICH A.: Training deep neural networks on noisy labels with bootstrapping. *arXiv preprint arXiv:1412.6596* (2014). 3
- [RYHH10] RASHTCHIAN C., YOUNG P., HODOSH M., HOCKENMAIER J.: Collecting image annotations using amazon's mechanical turk. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk* (2010), Association for Computational Linguistics, pp. 139–147. 1
- [RZYU18] REN M., ZENG W., YANG B., URTASUN R.: Learning to reweight examples for robust deep learning. *arXiv preprint arXiv:1803.09050* (2018). 3
- [Set10] SETTLES B.: Active learning literature survey. *University of Wisconsin, Madison* 52, 55–66 (2010), 11. 2
- [Set11] SETTLES B.: Closing the loop: Fast, interactive semi-supervised annotation with queries on features and instances. In *Proceedings of the conference on empirical methods in natural language processing* (2011), Association for Computational Linguistics, pp. 1467–1478. 2
- [SOS92] SEUNG H. S., OPPER M., SOMPOLINSKY H.: Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory* (1992), ACM, pp. 287–294. 2
- [SPI08] SHENG V. S., PROVOST F., IPEIROTIS P. G.: Get another label? improving data quality and data mining using multiple, noisy labelers. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (2008), ACM, pp. 614–622. 3
- [TIYA18] TANAKA D., IKAMI D., YAMASAKI T., AIZAWA K.: Joint optimization framework for learning with noisy labels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 5552–5560. 2, 3
- [UCOT11] USAMI Y., CHO H.-C., OKAZAKI N., TSUJII J.: Automatic acquisition of huge training data for bio-medical named entity recognition. In *Proceedings of BioNLP 2011 Workshop* (2011), Association for Computational Linguistics, pp. 65–73. 1
- [VAC*17] VEIT A., ALDRIN N., CHECHIK G., KRASIN I., GUPTA A., BELONGIE S.: Learning from noisy large-scale datasets with minimal supervision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 839–847. 3
- [WBSS04] WANG Z., BOVIK A. C., SHEIKH H. R., SIMONCELLI E. P.: Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13, 4 (2004), 600–612. 5
- [WGS*13] WILLETT W., GINOSAR S., STEINITZ A., HARTMANN B., AGRAWALA M.: Identifying redundancy and exposing provenance in crowdsourced data analysis. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2198–2206. 3
- [Wil17] WILKINSON L.: Visualizing big data outliers through distributed aggregation. *IEEE transactions on visualization and computer graphics* 24, 1 (2017), 256–266. 3
- [XYX*19] XIANG S., YE X., XIA J., WU J., CHEN Y., LIU S.: Interactive correction of mislabeled training data. 3
- [ZBH*16] ZHANG C., BENGIO S., HARDT M., RECHT B., VINYALS O.: Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530* (2016). 1
- [ZS18] ZHANG Z., SABUNCU M.: Generalized cross entropy loss for training deep neural networks with noisy labels. In *Advances in neural information processing systems* (2018), pp. 8778–8788. 3
- [ZZW18] ZHANG X., ZHU X., WRIGHT S.: Training set debugging using trusted items. In *Thirty-Second AAAI Conference on Artificial Intelligence* (2018). 3

NET₂VIS – A VISUAL GRAMMAR FOR
AUTOMATICALLY GENERATING
PUBLICATION-TAILORED CNN ARCHITECTURE
VISUALIZATIONS

Alex Bäuerle, Christian van Onzenoodt, and Timo Ropinski. "Net2Vis—A Visual Grammar for Automatically Generating Publication-Tailored CNN Architecture Visualizations." In: *IEEE Transactions on Visualization and Computer Graphics* 27.6 (2021), pp. 2980–2991

© 2021 IEEE. Reprinted, with permission, from Alex Bäuerle, Christian van Onzenoodt, and Timo Ropinski, Net2Vis – A Visual Grammar for Automatically Generating Publication-Tailored CNN Architecture Visualizations, IEEE Transactions on Visualization and Computer Graphics, June 2021.

Net2Vis – A Visual Grammar for Automatically Generating Publication-Tailored CNN Architecture Visualizations

Alex Bäuerle^{ID}, Christian van Onzenoort^{ID}, and Timo Ropinski^{ID}

Abstract—To convey neural network architectures in publications, appropriate visualizations are of great importance. While most current deep learning papers contain such visualizations, these are usually handcrafted just before publication, which results in a lack of a common visual grammar, significant time investment, errors, and ambiguities. Current automatic network visualization tools focus on debugging the network itself and are not ideal for generating publication visualizations. Therefore, we present an approach to automate this process by translating network architectures specified in Keras into visualizations that can directly be embedded into any publication. To do so, we propose a visual grammar for convolutional neural networks (CNNs), which has been derived from an analysis of such figures extracted from all ICCV and CVPR papers published between 2013 and 2019. The proposed grammar incorporates visual encoding, network layout, layer aggregation, and legend generation. We have further realized our approach in an online system available to the community, which we have evaluated through expert feedback, and a quantitative study. It not only reduces the time needed to generate network visualizations for publications, but also enables a unified and unambiguous visualization design.

Index Terms—Neural networks, architecture visualization, graph layouting

1 INTRODUCTION

PAPERS utilizing CNNs are published on a daily basis. An essential aspect of all these publications is to communicate the used or developed network architecture. Accordingly, a rising number of architecture visualizations can be observed from year to year (see Fig. 2). Authors, who often may lack visualization expertise, mostly use handcrafted, non-standardized visualizations. As a consequence, generating visualizations takes significant time, and authors often employ suboptimal visual encodings that are sometimes even inaccurate or erroneous.

We argue, as backed by our expert questionnaire (see Section 6), that the time invested in suboptimal visualizations would be better used to improve training results. Nevertheless, such abstract visualizations are generally considered to be of great importance. Therefore, automated approaches that obey to a common visual grammar are required. We argue that, ideally, such a visual grammar should be informed by three factors: current practice, expert demands, and visualization expertise. Accordingly, we have analyzed properties of existing architecture visualizations, which we scraped from all ICCV and CVPR papers published between 2013 and 2019 – which led to a pool of

751 such visualizations. ICCV and CVPR are prime conferences on machine learning for vision-related tasks and, thus, reflect the great need for such automated visualization approaches. Additionally, we contacted authors of highly cited papers encompassing architecture visualizations, in order to assess their demands. Last but not least, we brought in established rules from the data visualization literature to inform our visual grammar. Based on this, we propose the first method to automatically generate abstract, publication-tailored visualizations of complex, modern CNN architectures, obeying a unified visual grammar, which we refer to as *Net2Vis*. To this end, we make the following three main contributions:

- 1) We propose a set of requirements for effectively communicating neural network architectures, based on expert feedback and the analysis of existing visualizations.
- 2) Based on these requirements, we propose a new visual grammar for CNN architecture visualizations, which we make available via an online platform that transforms Keras code into visualizations tailored to the use in publications.
- 3) We release a data set of 751 neural network architecture visualizations, which we have extracted from all papers published at ICCV and CVPR between 2013 and 2019.

Fig. 1 shows an example visualization of a U-Net variant generated using our approach. To evaluate our approach, we conducted both a quantitative user study and a qualitative usability evaluation. The obtained results indicate that our techniques are beneficial for creating and reading CNN

• The authors are with the Visual Computing Group at Ulm University, 89081 Ulm, Germany. E-mail: falex.baerle, christian.van-onzenoort, timo.ropinski@uni-ulm.de.

Manuscript received 28 May 2020; revised 26 Jan. 2021; accepted 3 Feb. 2021.
Date of publication 8 Feb. 2021; date of current version 29 Apr. 2021.

(Corresponding author: Alex Bäuerle.)

Recommended for acceptance by S. Liu.

Digital Object Identifier no. 10.1109/TVCG.2021.3057483

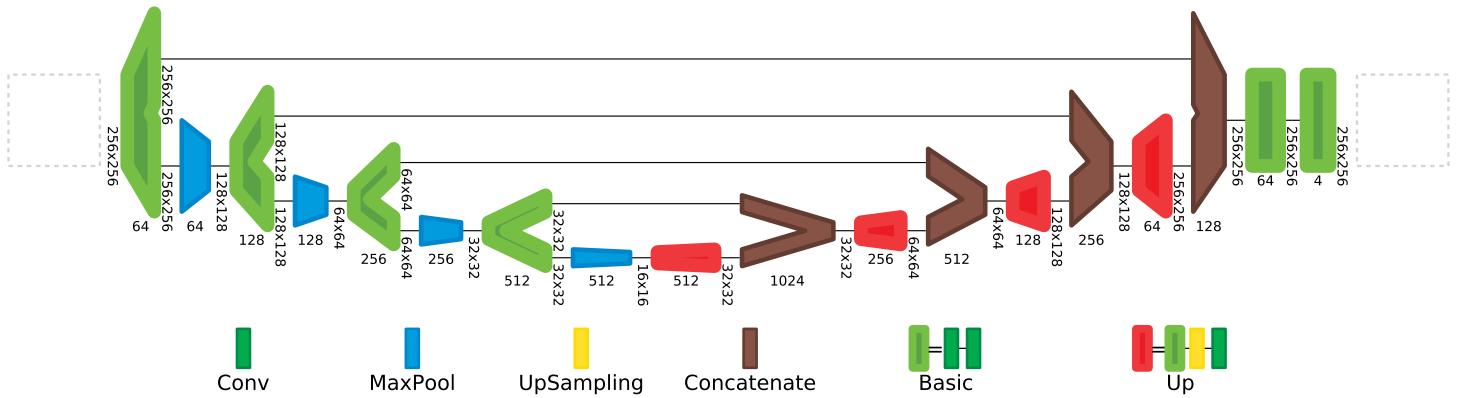


Fig. 1. Visualization of a U-Net variant. It was automatically generated using our approach based on the Keras code describing the architecture. Data flows from left to right. Glyphs represent layers or aggregates, while lines represent connections. Glyph widths communicate feature size, while heights communicate the spatial resolution. Both values are also given through labels, while dashed boxes on the left and right serve as placeholders to provide input and output samples. The legend communicates layer types and the composition of aggregates.

architecture visualizations, which is important for broad acceptance and unambiguous CNN architecture communication. Our techniques can be used in the form of an online platform at: <https://viscom.net2vis.uni-ulm.de>.

2 RELATED WORK

Handcrafted visualizations are part of many research papers that use neural networks in their publications [1], [2], [3], [4]. However, they differ greatly in their visual appearance, which complicates transferring knowledge between them, e.g., [5], [6], [7]. In addition, they sometimes contain errors, as can be observed in work done by Henzler *et al.* [8], where visual glyph encoding and glyph labeling diverge. Thus, automatically visualizing network architectures to convey their underlying ideas is an extensive field of research. In the following, we divide related research into approaches for debugging and investigating network architectures, and approaches targeted towards communicating these.

Debugging Approaches. Demonstrating the importance of visualization for the field, most deep learning frameworks, such as Tensorflow [9] with TensorBoard [10], Caffe [11] with Netscope [12], and also Keras [13], directly provide visualization toolkits.

All of these are clearly designed for online use. They are based on vertical layouts for detailed visualizations

including all layers and parameters and provide some information only on interaction. Their consumption of visualization space and required user interaction are perfect for debugging the network architecture, but it renders them inapplicable for use in publications.

Network visualization tools similar but unrelated to these frameworks such as ANNvisualizer [14] and Netron [15] suffer comparable shortcomings. Their glyphs do not convey any information apart from layer type, whereby additional information is displayed by overlaying textual annotations on top of the used glyphs. Additionally, their vertical layout, along with spacing between layers makes even small networks appear relatively large.

Another interesting visualization approach along this line was presented by Wang *et al.* [16]. Here, the focus is on comparing different neural network architectures. The approach can be used to identify differences in neural architecture design, compare the number of parameters, and draw conclusions for one's own architecture choice. However, while this approach allows for in-depth comparisons through interactive visualizations, it is not designed to convey network architecture details in a compressed, static way.

Communication Approaches. Some visualizations convey neural network architectures to explain their functionality to novices [17], [18], [19], [20], [21], or are targeted towards analyzing what a network has learned [22], [23], [24], [25]. These visualizations clearly fulfill their purpose to support education or interpretability, but are not designed for use in publications. They all display basic network architectures limited to a specific use case and are not generalizable to more complex architectures.

One visualization technique that is specifically targeted towards use in scientific papers is Drawconvnet [26]. Convnet-drawer [27], which builds on the aforementioned, provides such visualizations, and even allows visualization generation from source code. Similarly, NN-SVG also claims to create publication-ready network visualizations [28]. While these techniques can be used for small and simple networks, they all face major problems. First, they do not scale to modern, large network architectures since no aggregation technique is implemented. Second, they visualize layer connections simply by placing the layers from left to right, which means that parallel network parts cannot be

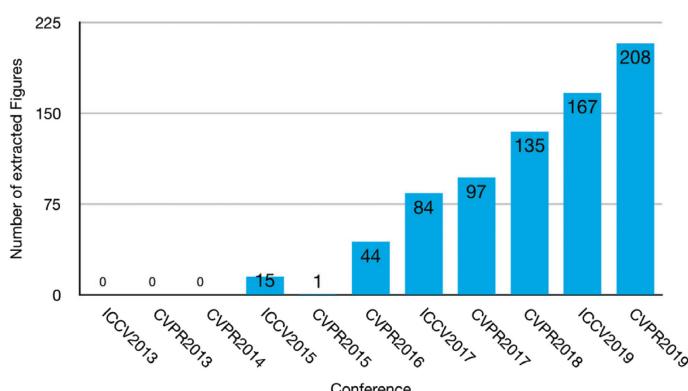


Fig. 2. Number of CNN architecture figures we extracted from all ICCV and CVPR papers between 2013 and 2019. We searched for pages of papers containing figures and the words *figure* and *architecture* in the same line to extract these. Then we manually filtered them to obtain only neural network architecture visualizations.

represented. Additionally, in Drawconvnet and NN-SVG, users have to invest the time to rebuild their network architecture to obtain visualizations.

The surveys by Hohman *et al.* [29] and Yuan *et al.* [30] discuss many of these graph visualization techniques. One important downside of all currently available approaches is that they struggle to visualize large networks in a compact way. Thus, it remains an open challenge to generate publication-tailored visualizations, despite the existence of the visualization systems described above. Current state-of-the-art visualizations [10], [12], [15] allow to inspect operations in great detail. However, these visualizations lack abstractions to make the general network structure comprehensible at a glance. For a demonstration of this problem, see our supplementary material, which can be found on the Computer Society Digital Library at <http://doi.ieee.org/10.1109/TVCG.2021.3057483>, which contains a comparison between Netron, Netscope, TensorBoard, and our approach. Other visualization techniques that aim at providing publication-ready visualizations cannot handle modern network architectures [26], [27], [28] and lack important features requested by experts. Thus, in research papers, these complex networks are usually simplified and drawn manually [4], [7], [8].

Besides the extra time effort related to this manual drawing process, the field lacks guidelines to create such visualizations, as is observable in our review of papers from CVPR and ICCV. Some properties in existing visualizations are ambiguously interpretable, e.g., where downsampling happens, and, for the lack of a common visual grammar, knowledge can hardly be transferred between different publications. Therefore, we propose a novel visualization technique for abstract architecture visualizations that are optimized for use in scientific publications, where display-space is limited and interaction is impossible.

3 DESIGN REQUIREMENTS ABSTRACTION

We argue that for abstract CNN visualizations, both model properties and layer properties need to be visualized. Model properties are important for the layout and arrangement of the network architecture, while layer properties visualize parameters of individual layers, or groups of layers. In the following sections, we describe how we used the analysis of current practice and insights from visualization research to inform our design proposition for CNN architecture visualizations.

Data Collection. To support this analysis, we interviewed multiple ML practitioners and reviewed 751 figures of neural network architectures extracted from papers of all CVPR and ICCV conferences between 2013 and 2019. This data was gathered by crawling <http://openaccess.thecvf.com/> for all 7988 main conference papers using scrapy [31]. We then filtered for neural network architecture visualizations and extracted all 1168 pages that contain a figure and had a line of text containing both the words *figure* and *architecture* using PyPDF2 [32]. We used pdffigures2 [33] to then extract the figures from these pages. This yielded 1027 images which we manually coded with respect to their visualization design choices. Since some of the used tools did not work perfectly, we had to delete images that did not

contain, or contained corrupted versions of architecture visualizations, leaving a total of 751 figures. To our knowledge, this is the first data set of such visualizations. The visualization properties we analyzed are italicized in the following sections and listed in the data set we release alongside this paper. The following observations are based on both, expert feedback and this figure analysis. When based on figure analysis, we indicate how many of the analyzed papers used a certain encoding as a percentage value.

3.1 Model Properties

In the following, we discuss which model properties of a neural network need to be communicated to quickly assess relevant architectural decisions within a neural network, as derived from the collected data.

Layout. Maybe the most important factor when visualizing neural network architectures is the interconnection of layers as it communicates the order in which the computation graph is executed. At the same time, there is also limited space for publication figures, which is to be taken into account when designing such visualizations. Thus, one has to find a layout that clearly communicates the order of computations, while also not wasting too much space for a single publication figure.

Connections. A consistent layout helps to resolve the order in which the network graph is traversed, but there is still important information missing, namely, connectivity. Without connectivity information, it is not clear which layer in a network routes data to which following layers. Especially if network graphs contain parallel execution steps, simply laying out the network layers in a consistent manner will not always resolve which layers actually interact with each other. Thus, communicating which layers are connected in the computation graph is essential.

Aggregations. When thinking about visualizing modern network architectures in publications, space and complexity is a major point of concern. It is a well-known fact in the visualization literature that hierarchical *aggregation* can help to simplify visualization designs [34]. Following this, as networks get more complex, authors manually aggregate layers to make their architectures fit on one page (63.4 percent). Here, *legends* sometimes indicate which layers are aggregated (15.2 percent). Both of these numbers were rising over the years, as shown in our supplementary material, available online. We, thus, argue that communicating modern network architectures requires a way of aggregating layer glyphs to create overview visualizations.

Omission. Another way of simplifying displayed network architectures is to just omit layers that are not important to convey the general idea of the network architecture. This is supported by our expert feedback, which indicates that simplification is a major feature for visualizations of neural network architectures.

Input and Output Samples. Several of the network visualizations incorporate *input* or *output examples* (73.1 percent). However, samples are mainly useful for image or shape related tasks and do not provide additional information concerning the network architecture. We thus argue that such samples can help for some application areas, while they should not be presented for others.

3.2 Layer Properties

Layers are the building blocks of the computation graph that defines any network architecture. Thus, visualizing properties that parametrize these layers is important to convey the structure and architectural decisions of said network. In the following, we discuss important layer properties and explain which of those should be visualized for obtaining a general overview of a network architecture.

Layer Type. We consider the *layer type* to be the most important layer-specific variable. Together with the model properties, it already helps greatly in determining the functionality of a neural network. It is thus important to encode layer types as a visually prominent variable.

Spatial Resolution. Next, to determine the functionality of a neural network architecture, it is important to be able to follow the transformation of data into or out of the latent space, which is often referred to as the change of spatial resolution. Thus, the dimensionality of data is another variable to be considered when visualizing neural network architectures. While only slightly more than half of the surveyed visualizations encode the *spatial resolution* (53.7 percent), expert feedback, and the fact that this variable can be encoded within the layer glyphs, suggests that visualizing it brings great benefit at no cost. We thus advocate for always providing information about spatial resolution.

Feature Channels. All previously mentioned attributes apply to almost all types of neural networks. Feature channels, on the other hand, are mainly important for CNNs. Thus, most visualizations do not at all encode the number of feature channels (56.7 percent). Since feature channels match the variable type of the spatial resolution, and since they are tightly coupled across the network, they are often viewed in combination. Thus, they support the assessment of data transformation, and our evaluations surfaced that their display is important for architecture visualizations of CNNs.

3.3 Properties Not to Be Visualized

We propose a set of properties to be visualized for providing an overview of a network architecture. However, there are other properties we explicitly advise to omit in non-interactive visualizations of neural network architectures.

Kernel Size. Kernel sizes can be found in many visualizations as textual descriptions of layers (24.4 percent) or as encodings in the layer glyphs (5.0 percent), but are not encoded in most of the visualizations (73.5 percent). When analyzing why kernel sizes are not displayed in most visualizations, two factors were apparent. First, kernel sizes often are consistent across multiple layers leading to repeated information. This is in contrast with the request for reduced complexity by domain experts (see Section 6). However, the biggest problem with them is that they are in stark contrast with layer aggregations, which are important to reduce the complexity of network visualizations. For aggregations, there is no such thing as one kernel size as it may differ for layers contained in them. Additionally, as aggregations are reused, kernel sizes may be different again. We aim at reducing repetition and embracing aggregation. Thus, we propose not to display kernel sizes in overview visualizations for neural networks. This is in line with the expert

feedback of domain experts, e.g. *I'm very interested in great visualization tools that emphasize function and architecture over the kind of "obtuse prettiness" [...]*.

Additional Layer Properties. Neural network layers contain many more features such as weights, strides, padding, and others. Yet, most of them only exist for certain layer types and are thus rarely communicated. Some printed visualizations include features such as activation maps, e.g., [5], or receptive fields, e.g., [35], but they are only provided for specialized use cases. In contrast, we argue that for obtaining a general network architecture overview, such features are not necessary. This is supported by our expert feedback and in line with our goal of providing an abstract overview of the network architecture, where detailed information can be obtained by reading the publication it is contained in.

Dimensionality. Three-dimensional visualizations are helpful if the reader's task includes shape understanding, but less so for any relational task [36], [37]. Thus, the relation of layers and their spatial position would suffer from being visualized in 3D, while the benefit of using three-dimensional layer glyphs would only be to resemble the shape of data in case it is three-dimensional. Another important argument against three-dimensional visualizations in publications is that the viewpoint cannot be changed. Contrary, exploration is essential for utilizing the benefits of three-dimensional visualizations [36], [38]. About half of the analyzed network visualizations display layers in 3D (50.2 percent), however, this added dimension does not convey additional information for most of them. Since the reduction of spatial resolution is almost always applied to all spatial dimensions equally, the third dimension is not needed for the visualization of such changes. For those reasons, we advise not to visualize layer glyphs in 3D.

3.4 Summary

Based on these requirements, we propose to use the following properties in any visualization that aims at providing a general overview of a CNN architecture in non-interactive visualization environments:

Model Properties

- 1) Layout
- 2) Connections
- 3) Aggregations
- 4) Omission
- 5) Input and Output Samples (for some tasks)

Layer Properties

- 1) Layer Type
- 2) Spatial Resolution
- 3) Feature Channels (for convolutional layers)

We, thus, elaborate on *what* to visualize in this section, defining the dimensions of our proposed design space. This assessment is based on our analysis of figures extracted from the top conferences in the field, as well as expert feedback both before and during the development of our approach. It is strictly tailored towards conveying the overall idea of a network architecture, and does not cover cases in which specific features of a network are to be communicated. Therefore, we also provide guidelines for which features not to visually encode for abstract architecture

visualizations, thus preserving simplicity and preventing the need for interaction. By always visualizing the aforementioned parameters in the same way, users can transfer knowledge between different visualizations. We thus advise not to render the mapping of variables customizable. Instead, we propose a unified visualization design for these parameters. In the following sections, we map these visualization properties (i.e., *what* to visualize) to specific visual encodings (i.e., *how* to visualize them), describing the proposed design space inspired by similar design space descriptions [39], [40].

4 VISUALIZATION OF MODEL PROPERTIES

Based on our assessment of important visualization aspects for communicating CNN architectures as described in the previous section, we now explain *how* we propose to visually encode the model properties. For these global properties, the design space consists of the aforementioned dimensions, i.e., Layout, Connections, Aggregations, Omission, and Data Samples, which we describe in the following subsections.

4.1 Layout

The representation of this design space dimension mainly concerns the spatial arrangement of layers which can be vertical or horizontal, and in any direction. Most investigated neural network visualizations layout their layers from left to right (81.6 percent). Our figure analysis as well as the collected expert feedback indicate that for publications, this layout is preferable over vertical layouts, which are often used in online tools. It does not only preserve the reading direction of western cultures, but visualizations also nicely fit across the width of a page. Furthermore, perceptual rankings indicate that position best encodes ordered data [41], [42], [43], such as the order of network layers. Follow these insights, and the fact that space taken up by publication figures is important, we propose to employ a narrow horizontal layout, in which parallel execution steps of the network are stacked vertically on the same horizontal position.

To layout CNN graphs, we propose to use the network simplex algorithm [44] which is explicitly targeted towards drawing directed rank-based graphs. The rank-based nature of this algorithm perfectly fits our use case in which parallel layers are to be placed at the same x-coordinate, and sequential parts of the network tend to be drawn on the same vertical level. Using an algorithm that only layouts series-parallel graphs is not an option, since Keras operations are not restricted to those (e.g., $[a \rightarrow b, a \rightarrow c, b \rightarrow d, c \rightarrow d, d \rightarrow e, b \rightarrow e]$).

4.2 Connections

For this dimension of the design space, possible representations are an implicit connectivity without explicit visualizations and different forms of connecting lines. In existing figures, connections between layers are visualized either using lines (73.4 percent) or by simply placing layers next to each other. Some visualizations additionally add arrowheads to clarify the *direction* of data flow (65.9 percent). Following most visualizations, we also propose to use lines as connections between layers to emphasize the graph structure

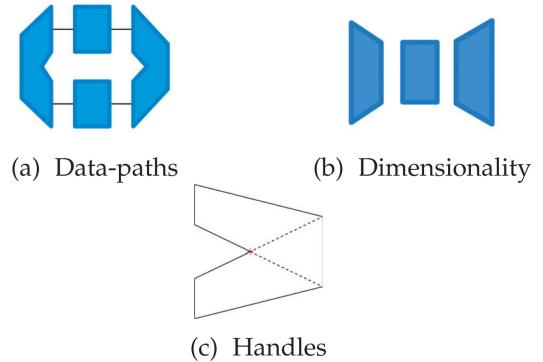


Fig. 3. (a): Two simultaneous data-paths. The layer displayed on the left has two outgoing connections, while the layer displayed on the right fuses these paths back together. (b): Left: layer that reduces the spatial resolution; Center: same spatial resolution for input and output; Right: a layer that increases the spatial resolution (c): Multi-handled glyphs are always connecting corners on the input side with corners on the output side, here depicted by a dotted line.

of neural networks. However, we propose not to add arrowheads to these connections, as the forward data flow direction is uniformly left-to-right in our visualizations.

Many architectures contain *skip connections* visualized by lines between distant layers (55.9 percent, rising over the years, see supplementary material available online). Displaying splits in the execution graph only through lines has the negative implication that size-related attention bias is induced [45], [46]. Thus, we propose a glyph design that prevents such issues. Whenever a layer has multiple outgoing or incoming connections, we modify the glyph that represents it as shown in Fig. 3a. This way, there might be multiple ends on the left or the right side of the glyph each having the same visual prominence. At the same time, splits and joins of the data flow, which are important features of the architecture, are highlighted. A visual representation of such multi-handled glyphs is illustrated in Fig. 3c.

One might think that this layer shape induces problems with edge crossings. However, edge-crossings are uncommon to neural network architectures. We have only found planar network graphs which consequently can be laid out to avoid crossing edges.

4.3 Aggregation

The aggregation of multiple layers is another dimension in our design space. Here, representations can be no aggregation of layers, vertical stacking, or replacement with a group-representing glyph. As modern networks become increasingly complex, the aggregation of layers is inevitable. Such aggregations can contain many layers at once, and even parallel paths, which is why we do not employ a stack-based visualization. Instead, we adopt the paradigm of providing ways to aggregate multiple layers and replacing them with a new, single glyph. As this removes direct insight into the content of aggregations, we resolve them in a legend below the network graph. Furthermore, domain experts and users frequently requested a visual separation of aggregations. Thus, in our approach, their border is drawn thicker, and their color scheme is inverted (lighter border than center).

Aggregation Constraints. As aggregations substitute all occurrences of selected layer sequences throughout the

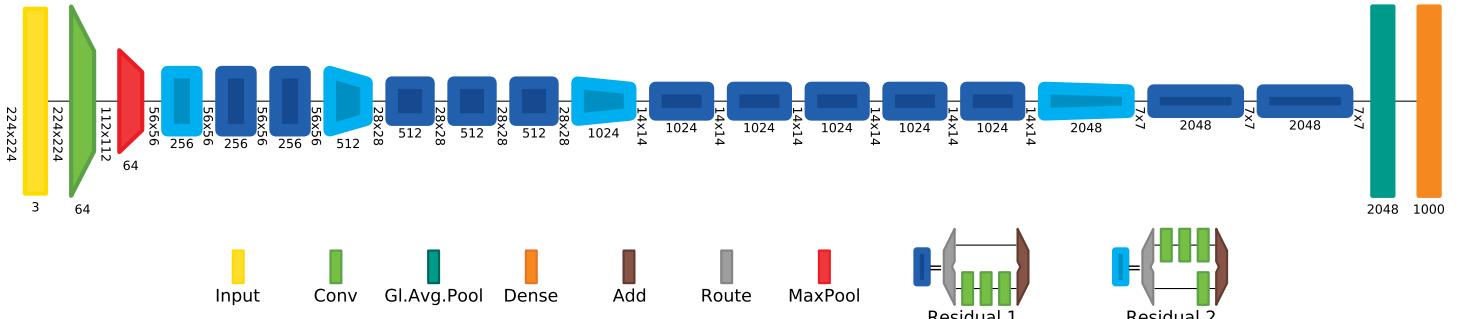


Fig. 4. With ResNet50, the removal of activation and batch-normalization layers from the visualization, which do not add information about the network structure, along with the repetition of residual blocks in ResNet allows us to reduce the number of glyphs that need to be drawn from 177 to just 21, even though we added routing layers to clearly define the beginning of a residual block.

graph, only parts of the network graph that can be represented by one sequential layer can be aggregated. We, thus, restrict aggregation to sequentializable segments. This way, no deformed aggregation layers can occur, where two outputs or inputs might differ in their spatial resolution. This ensures visual consistency such that layers always manipulate the data the same way for all of their connections and that all connections of each layer end on the same horizontal level in the graph.

Automatic Aggregation. To generate aggregations, the layers to be aggregated can either be selected by the user or more conveniently be selected automatically. To obtain automatic aggregations, we propose to analyze all sequential parts of the network. We then search for recurring sequences of layers. The most frequent of these sequences is then assumed to be the preferred aggregation.

Interacting With Aggregations. We argue that visualization designers should be able to remove or temporally deactivate aggregations of the network, which expands abstracted layers back to their initial layout. Deactivated aggregations are preserved in the legend for later reuse to be able to explore the visualization without losing information. To visually convey the state of aggregation, we propose to draw active ones with a dark outline and black description text, while outlines and text are drawn in light gray for inactive aggregations and layer types that are hidden by the user.

This way, the effect of different aggregation levels can be explored while preserving all aggregation information.

Split Layers. While there are dedicated layers to fuse computation paths, routing the data to multiple outputs is done implicitly. Thus, it is possible, that, e.g., an activation layer feeds data to two different paths. Visually, this is a problem as splits and merges of the computation graph are often seen as blocks and frequently get aggregated by the user. Activations are orthogonally seen as the end of a computation group rather than a start of one. For such dedicated groups, we argue that the user should be able to add special routing layers. This way, they can clearly communicate the special role of such multi-path aggregations while also assigning more importance to data path splits, e.g., as shown in Fig. 4.

4.4 Omission

In the computation graph, any function that has been added by the developer is considered a network layer. However,

these graphs can be defined at different levels of detail (e.g., activation within layer or separately). Thus, in our approach, this graph can be thinned by the user to better convey the underlying architecture rather than each individual computation step by hiding individual layer types entirely. Showing a graph overview, and then allowing the user to filter it is in line with Shneiderman’s mantra [47].

4.5 Input and Output Samples

As described in the property analysis of Section 3, input and output samples may, for some networks, be helpful. Directly integrating such samples would, however, require the user to provide training or testing data, and thus interfere with the automatic nature of our visualization design. We thus propose not to include them directly in any programmatic CNN architecture visualizer. Instead, we suggest to optionally provide placeholders for input and output samples which users can replace during post-processing with actual samples.

5 VISUALIZATION OF LAYER PROPERTIES

In the following, we describe the design space dimensions of the visual layer encoding we propose based on the design discussion presented in Section 3, in other words, *how* to visualize layer properties. Our visualization design supports the direct encoding of layer type, spatial resolution, and the number of feature channels. By reducing the visualization space to these three variables, we are able to encode all of them in simple glyphs that represent the layers of the network architecture to be visualized. For the spatial resolution and the number of feature channels, percentages only consider the 464 visualizations that contain convolutional layers, as we explicitly tailored our glyph design to work well for CNNs.

5.1 Layer Type

The design space dimension for the layer type can take the representations of textual display, color-coding, and shape-encoding. Most visualizations only use textual descriptions (52.7 percent), or text along with glyph color (14.6 percent) to convey layer types. However, it can be repetitive to encode the layer type as text below each layer. We, thus, propose to provide this textual encoding optionally (see Fig. 6) while not being displayed in the default setting. The layer type is a categorical attribute and consequently best visualized using a channel that is optimized for such data [36], [48]. In Mackinlay’s ranking [41], color ranks just

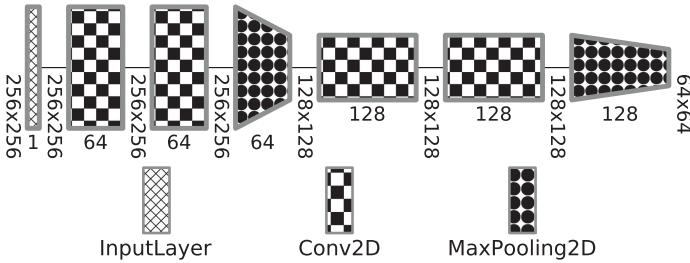


Fig. 5. Accessible encoding of the layer type to also support readers with monochromatic vision and publications without color.

behind position, which we already employ to communicate the data flow of the network. Perception research also shows that color is a visually dominant channel [49]. Thus, we propose to use a color encoding to convey the layer types in the visualized network.

Color Assignment. In our approach, colors for new layers are automatically preset in accordance to one of two alternative approaches. The first approach is motivated by farthest point sampling. It finds unused colors in hsv color space by searching for the biggest gap between any two hue values of already used colors. This is the most functional approach, as it optimizes for color difference. Unfortunately, it might result in colors that are indistinguishable by color-blind users and unpleasant color choices. Therefore, the second option for color proposition is palette-based and serves as the default. We suggest to use two color palettes, one from materialuicolors [50] for visually pleasing color mappings, and one adapted to users with trichromatic or dichromatic vision [51]. Additionally, visualization designers should always be able to customize the color that is used to encode a layer type.

To also make our visualizations accessible to readers with monochromatic vision, and support publications without colored images, we also propose a texture-based encoding of layer types, see Fig. 5. We provide twelve distinguishable patterns that can be extended upon when needed.

Legend Generation. Since we use color-coding to differentiate between layer types, a legend that maps these color codes back to layer names is needed. This legend contains a glyph for each layer type in the network and displays the name of its layer as shown in Fig. 4. Based on expert feedback, legend items are sorted from simple to complex. This complexity is determined by analyzing the dependency-tree of aggregations, whereas nested aggregations are seen as more complex.

5.2 Spatial Resolution

The spatial resolution is a design space dimension that might be represented by glyph height, glyph width, glyph color, and textual annotations. As the spatial resolution is a quantitative and sequential variable, length, angle, slope, and area are the best remaining options for encoding it [41], [42], [52]. In some visualizations, the spatial resolution is represented by the shape of the layer glyph in combination with textual information (10.8 percent) or just textual representations (10.1 percent). However, mostly only glyph shape (32.8 percent) is used. We propose to use height in combination with text as a direct mapping of the spatial

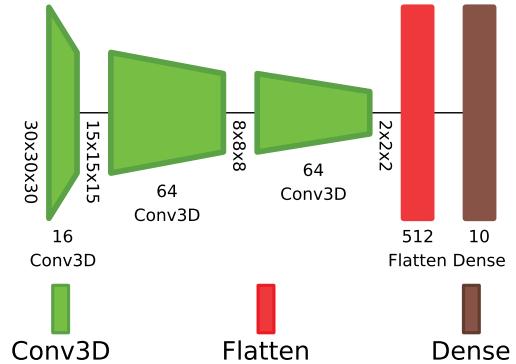


Fig. 6. Net2Vis can also be used to generate visualizations of multi-dimensional network architectures.

resolution, as this does not interfere with our network layout and can be encoded in the glyph design directly.

However, not all visualizations map glyph height in the same way. Sometimes the height of the downsampling layer already encodes the changed resolution, while in other visualizations, the next layer is first affected by this change. This ambiguity makes the interpretation of such visualizations hard, as one needs to determine which representation was chosen for each visualization approach. Furthermore, the transformation of the resolution is determined by multiple parameters (e.g., stride, kernel size, padding). Thus, the output resolution is a result of the inner working of a layer rather than a fixed parameter. We, therefore, propose to visualize the spatial resolution as a change within the layer. To convey the underlying transformation, we set the height on the left edge of the glyph to match the input resolution, while the height on the right edge of the glyph reflects the output resolution, resulting in trapezoid-shaped glyphs, as shown in Fig. 3b. This conveys the change of resolution as a result of the mathematical operations within the layer while at the same time removing its ambiguity. In addition, with this encoding, the relation of input and output dimension from layer to layer as well as across the whole network is clearly visible by horizontally scanning the visualization [53].

To draw these glyph shapes, in our approach visualization designers can define a minimal and maximal height for the glyphs. We then obtain the spatial resolution for the input and output tensors for each layer. The highest and lowest value of all spatial extents gets mapped to the extremes of the user-defined height values. Values between these extremes are interpolated linearly to convey the actual spatial resolution for the input and output of each layer in the network. Using these interpolations, each input and output of every layer gets a height value assigned, which maps this important quantitative variable to the vertical length of the glyph ends, as Mackinlay's ranking suggests [41], [52].

We found that it is mostly not important to convey the exact spatial resolution by means of textual descriptions, as only 20.9 percent do so. Since many modern architectures further allow inputs to be arbitrarily shaped, e.g., [3], [7], [54], the spatial dimension is not necessarily fixed at any given layer for many network architectures. We, therefore, advocate for the option to toggle labels that display the exact spatial resolution between the layers, following our visualization design, in which the resolution is fixed between layers but changes within them.

5.3 Feature Channels

Similar to the spatial resolution, feature channels may be represented by glyph height, glyph width, glyph color, and textual annotations. Textual descriptions (20.0 percent), glyph shapes (13.8 percent), or a combination of both (9.1 percent) are common for conveying the number of feature channels. The number of feature channels, just as the spatial resolution, represents the important transformation into or from latent space, tightly coupling these two variables. Thus, we propose to employ a similar visual encoding to convey them, again, mapping a perceptually preferable length parameter, in this case, glyph width, to this variable [41], [43]. Feature channels are different from the spatial resolution in that they are fixed properties of a layer and not derived from the previous dimensions. We, therefore, propose to visualize this variable as a direct property of the layer rather than a change within it. In our approach, the number of feature channels can additionally be represented as text, displayed below each layer.

While the spatial resolution and its change within a layer is represented by the heights at both ends of the layer glyph, combining this representation with the number of feature channels as encoded in the glyph width can reveal important overall information on the processed data. Together, they present the total amount of data that is processed by the layer. In our visualization design, this amount of data is implicitly encoded in the area covered by the glyph.

Accordingly, with this glyph design, variable importance is perfectly aligned with Mackinley's ranking [41], [42], [53]. At the same time, these glyph shapes fit nicely into the horizontal network layout.

Dense layers only have one intrinsically specified dimension of data. Since this is a fixed dimensionality, it is more similar to feature channels than the spatial resolution. Additionally, dense layers are one-dimensional and commonly visualized as vertical chains of neurons. Thus, for dense layers, the number of neurons is also mapped to the glyph height. Additionally, for better differentiation of these simpler layer types, we also propose to employ a simpler color-coding, using the same color for the border as well as for the body of the glyph.

For all of these visual encodings, we propose default specifications, such as minimal and maximal width and height of layer glyphs, which can be customized by the user.

6 EVALUATION

To evaluate the proposed concepts, we have visualized multiple well-known architectures with our techniques, gathered expert feedback to inform and evaluate our visualization design, and conducted a quantitative user study.

6.1 Application Examples

To provide the community with means to incorporate our visual encoding, we implemented *Net2Vis* as an online application which is available at <https://viscom.net2vis.uni-ulm.de>. Here, users can paste Keras code to obtain ready-to-use architecture visualizations and download those as PDF figures for direct use in publications, as well as SVG images, which can be edited in hindsight. To demonstrate *Net2Vis'* capabilities, we applied it to several

commonly used network architectures. Fig. 1 shows a variation of U-Net [55] which is frequently used for semantic segmentation. Fig. 4 shows a visualization of ResNet [56] where we show a reduction from 177 to just 21 glyphs through our aggregation techniques. Finally, in Fig. 6, we demonstrate that we also support multi-dimensional network architectures. Even more application examples where we show that our techniques can even visualize networks such as InceptionV3 [57] can be found in our supplementary material, available online. Here we show popular published neural network architectures [1], [6], [55], [56], [57], [58], [59], [60], [61], [62], [63], and two network visualizations we used for our own publications and presentations.

6.2 Comparison to TensorBoard

TensorBoards graph visualization is based on the computation graph as defined in the program code for the network architecture. Aggregations in the graph visualizer can, thus, only be made for those nodes that contain sub-nodes. This can provide more detail as every operation in the graph can be examined. However, TensorBoard's aggregation approach is far less flexible and not tailored towards publication figures. For publication visualizations, users typically want arbitrary aggregations of multiple layers without having to specify parent nodes for those in the code first. Our flexible graph-based aggregation methods support that exact need, in that they allow for tailoring aggregations in a way that best communicates the overall architectural ideas. This also leads to much smaller and easier to understand figures, which we will elaborate more on in the following evaluations.

6.3 Expert Interviews

After the implementation of the first version of *Net2Vis* was complete, we conducted qualitative interviews with experienced machine learning researchers.

Expert Selection. We used *Net2Vis* to generate replications of visualizations from published papers. These visualizations were then emailed together with our questionnaire to the authors of the respective papers (in the following referred to as experts). In total, we contacted 7 experts in this manner who had novel papers or high citation count and published at different conferences.

Three of those answered our questions, one replied to have other obligations, and three did not get back to us. Two of the papers from which the respective experts gave feedback are highly cited (i.e., Noh *et al.* [1]: > 2300 and Long *et al.* [6]: > 14000), the third is a more recent publication from 2018 [59].

Questionnaire. Our questions were designed following Munzner's nested evaluation model [64]. Thus, we assessed the need for such automatic visualizations (Q1, Q5), analyzing the threat of targeting a wrong problem. We also investigated why 3D visualizations are so common (Q3), and asked about our visualization design (Q2, Q4) to evaluate the abstraction and encoding technique, which are the second and third possible pitfalls [64]. Our implementation proves interactivity, the fourth possible visualization pitfall [64].

We intentionally asked only five questions to keep the time needed to answer our survey relatively low, and thus

maximize the chance for responses from these well-known researchers.

However, we encouraged the experts to add any comments to their replies.

Feedback. All replies were positive and emphasized the importance of such automatic visualizations. Furthermore, they gave positive feedback on our glyph and graph design. Their main concerns were scalability to different architectures since they only received visualizations for their papers. However, as can be seen in Section 6.1 as well as the supplementary material, available online, *Net2Vis* is designed to work with a wide variety and high complexity of convolutional neural networks.

Q1: How Much Time Did You Spend Creating Your Figure? All experts stated that creating figures of their architecture took too much time. While initial versions seemed to take about one hour on average, they all noted that they needed multiple iterations.

This supports our claim that this task can be greatly optimized as it takes valuable research and paper-writing time from the experts.

Q2: How Do You Understand the Mapping of Number of Feature Channels and Spatial Resolution in the Visualizations We Sent You? All three responded that they understand that and how the spatial dimension and feature channels are mapped onto the glyphs correctly. Thus, the mapping of spatial dimension and feature channels seems comprehensible even though this representation is more abstract than the ones used in their papers.

Q3: Why Did You Pick a 3D Visualization for the Layers and Which Information Did You Want to Convey? All experts said this was done since the data was three-dimensional. None of them conveyed additional information this way. One expert also admitted that 3D visualizations introduce the problem that layers cannot always be evenly spaced because of occlusion.

All three also noted that this makes the visualization more complex. Our choice of visualizing the network architecture in 2D was preferred also by these experts.

Q4: What Do You Think of Visualizing The Transformation That Happens During Pooling/Unpooling as a Transformation of the Layer Itself (Trapezoid Glyphs) Rather Than In-Between? One expert said *I found many people complained about not drawing in-between relation between pooling/unpooling*, which indicates that this implicit transformation used in the existing visualizations is confusing to the reader. Another expert mentioned that *the trapezoids seem like a nicely simple way to indicate where and how much downsampling is going on*. However, he also noted that this is dual to the way Alex-Net visualizes network architectures which has been picked up by many researchers. While it is a valid concern in that readers have to differentiate between these approaches, we think that the mentioned benefits outweigh the downsides which naturally come with adopting a new visualization approach.

Q5: Would You Use Such a Tool For Your Projects, if Available? All experts agreed that they would be users of network visualization generators as proposed in this paper. One expert additionally mentioned that he would still want to have the possibility to modify the visualization to his will which he did not know is possible in *Net2Vis*.

Other remarkable comments that clearly show the need for such automatic visualizations were: *I have been ranting about this for a while and have been waiting for somebody to ask, and I've been hoping someone would make better automatic visualization toolkits for a while.*

Based on the comments and free-form texts within the expert feedback, we further refined our visualization design, e.g., by visually separating groups from standard layers through different border styles or increased vertical spacing between parallel execution steps.

6.4 Quantitative User Study

To evaluate our final visualization design, we then conducted a quantitative study with 10 participants (3 female, 7 male, $M_{age} = 25.9$ $SD = 2.27$).

These participants were recruited in a university setting. Requirements for participation were that the participants knew what a CNN is, knew what elements CNNs consist of, and knew about feature- and spatial dimensions. Machine learning expertise of the participants varied between less than half a year and less than five years, which nicely reflects the broad audience which we expect for our visualization design. Based on internal piloting, we found that our study takes participants around an hour to complete. We compensated them with a 10 Euro Amazon gift card.

Methods. The participants were presented with different well-known machine learning architectures. Each of these architectures was visualized using three different visualization approaches, *Net2Vis*, TensorBoard, and a visualization taken from the original publication. We implemented the network architectures for each paper in Keras, which enabled us to directly export visualizations using *Net2Vis*. For TensorBoard, we had to manually screenshot and stitch the visualizations as the export functionality in TensorBoard did not work for us. For each visualization, participants had to answer eight questions by extracting information about the architecture. These questions included the following tasks: *How many convolutional layers does this architecture contain?, What is the maximal feature depth for the convolutional part?, What is the minimal spatial resolution of the convolutional part?, What are the input dimensions for this network?, What are/is the output dimension(s) of this network?, How many times does downsampling happen in this network?, How many steps are performed to increase the feature dimension?, Is this Architecture "Fully Convolutional"?*. Participants entered the answers to these questions into a text field and were instructed to answer -1 if they could not extract the information from the visualization. Each participant was presented with every network architecture (6) using all three visualization techniques (3), resulting in 18 stimuli presented in a randomized order. Afterward, participants were presented with each architecture using all three visualization techniques, side by side, and were asked three comparison questions: *Which of the above visualizations contains the most useful information?, Which of the above visualizations was easiest to interpret?, and Which of the above visualizations did you find most visually appealing?*.

Analysis. To analyze the performance of the different visualization approaches, we computed the mean accuracy over all eight questions for each of the presented

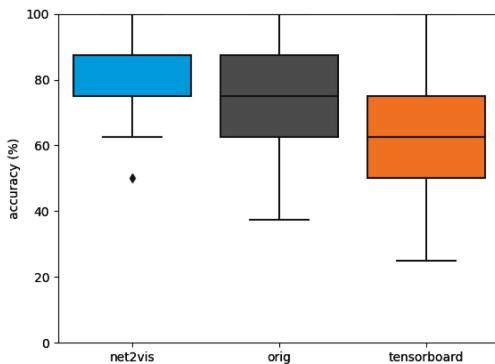


Fig. 7. This plot shows a comparison of accuracies for each condition. Accuracies are the mean of all eight questions asked for each network. For *Net2Vis*, the median is the same as the upper IQR. One can see that our approach led to the best accuracy, even though the other approaches contained questions that participants did not even need to answer. We additionally found significant differences between TensorBoard and the other two conditions, indicating that TensorBoard is the worst approach of the three for an abstract display of network architectures.

architectures for each condition (*Net2Vis* versus TensorBoard versus handcrafted), as can be seen in Fig. 7. We compared these conditions using Friedman ANOVA and found a significant difference in accuracy for the visualizations drawn using *Net2Vis* ($M = 83.75$ percent, $SD = 12.02$ percent) when compared to the handcrafted ($M = 75.83$ percent, $SD = 15.75$ percent), and TensorBoard ($M = 63.13$ percent, $SD = 17.29$ percent), $\chi^2(2) = 38.75$, $p < .001$. During post-hoc analysis using Nemenyi's test, we found a significant difference between *Net2Vis* and TensorBoard ($p = .001$), as well as TensorBoard and the handcrafted version ($p = .001$). While the accuracy for *Net2Vis* was higher compared to the handcrafted version, we could not find a significant effect between these conditions ($p = .11$). When analyzing the time our participants took to complete all eight questions for each visualization, we found a significant effect between our conditions. As with accuracy, TensorBoard showed the worst performance ($M = 27.1$ s, $SD = 30$ s), followed by *Net2Vis* ($M = 16.98$ s, $SD = 15$ sec), and the handcrafted version ($M = 13.53$ s, $SD = 8.95$ s), $\chi^2(2) = 17.1$, $p < .001$. During post-hoc analysis, we found this effect to be significant between TensorBoard versus *Net2Vis* ($p < .05$), and TensorBoard versus handcrafted ($p < .001$). However, when comparing *Net2Vis* and the handcrafted version, we could not find a significant effect between these conditions ($p = .23$).

For the questionnaire about which visualization technique our participants would prefer with respect to informativeness, interpretability, and design, our technique outperformed the other conditions again. For informativeness, participants preferred our approach in 86.6 percent of cases, while they favored the original version in 13.3 percent of cases. The interpretability of our design was ranked highest as well, as in 75 percent of all cases, our approach was favored, whereas the original versions were favored in 25 percent of cases. The design of our approach was favored in 70 percent of all cases, against 30 percent in favor of the original visualization. Looking at the individual networks, our approach was rated better or evenly good in all conditions except for the design of U-Net, where 60 percent favored U-Net. Remarkably, across all six conditions and

all ten participants, TensorBoard did not get voted as preferable once. Used visualizations, plots, and raw study data can be found in our supplementary material, available online.

6.5 Usability Evaluation

In another evaluation with 16 participants (13 male, 1 female, 2 did not report, $M_{age} = 28.06$ $SD = 4.23$), we also evaluated our approach from the view of a visualization designer. These participants took part in our study right after a one-week full-time deep learning course, where we recruited the participants. Thus, they were familiar with the underlying concepts. Participants were given a brief introduction to *Net2Vis* before they had the chance to visualize one of their own architectures. Then, they filled a questionnaire regarding the system, including a system usability scale questionnaire (*SUS*), and a demographic questionnaire. The usability analysis through the *SUS* questionnaire resulted in a mean score of 83.44 points ($SD = 6.25$) indicating *excellent* usability [65].

6.6 Discussion

The main goal of *Net2Vis* was to introduce a unified design for neural network architectures as a replacement for handcrafted visualizations. While we could not find a significant effect between *Net2Vis* and the handcrafted version during our quantitative user study, we still argue that our approach outperforms the handcrafted versions in some ways. While not significant, *Net2Vis* showed overall higher accuracies, which might indicate that the effect between the conditions would become significant, given a larger number of participants. Furthermore, we saw higher accuracies in key aspects of these types of visualizations, particularly in the direct comparisons such as interpretability and design of the visualization. Besides that, for the TensorBoard visualizations and original paper figures not all information needed for answering the questions was always present during our quantitative user study. In such cases, users could answer with -1 whenever information was not available. Thus, in these cases, users could not give a wrong answer. We still counted these answers in our evaluation, essentially putting our approach at a disadvantage. Nonetheless, our techniques outperformed the others in almost all metrics. Compared to the original paper figures, which ranked better than TensorBoard, our approach has the additional advantage of a unified design and automation, which can save time and makes knowledge transfer possible.

Referring back to the nested model of evaluating visualization design [64], first, our evaluation indicates that we indeed work on a relevant problem for our target users. Moreover, the abstraction level we chose seems to be appropriate as supported by our quantitative user study. Our expert interviews clarified that it is important to keep such visualizations simple and minimalist as none of the experts complained about missing information in our visualizations. One expert even explicitly stated that it is important to *emphasize function and architecture especially over obtuse prettiness that you see in many of the tools that visualize activations or things like layer gradients*. Third, the evaluation of our

expert interviews, as well as the quantitative study, suggest that our glyph design is easily interpretable and adds valuable information as it directly visualizes the transformation of data. Fourth, the interactivity of our implementation shows that we do not have a problem with the speed of the algorithm. The results of our quantitative user study support this, with an excellent usability score for our system. While this evaluation is only a first indication of the applicability of *Net2Vis* and only adoption of the concepts in the field can prove its value, the evaluation clearly supports the need for such a tool as well as our design choices.

7 CONCLUSION

In this paper, we propose a visualization design for communicating neural network architectures which is based on expert feedback, state of the art visualizations, and user studies. Additionally, we provide an automated approach for visualizing CNN architectures and release a data set which contains an analysis of 751 paper figures of neural network architectures from all CVPR and ICCV conference papers since 2013. Currently, such visualizations are mostly handcrafted which consumes time in the paper writing process. Our novel visual grammar for visualizing CNNs, called *Net2Vis*, is informed by an analysis of the current practice, expert feedback, as well as widely accepted data visualization guidelines. Our proposed visual grammar incorporates visualization requirements for neural network architecture visualization, network layout, aggregation, legend generation, and a novel glyph design. *Net2Vis* represents the first visualization technique for modern and complex CNNs that is tailored towards use in publications, while the results of our quantitative user study indicate that *Net2Vis* improves both visualization generation and reading. For wide adoption, *Net2Vis* can be used as an online service at <https://viscom.net2vis.uni-ulm.de>, where users can obtain CNN architecture visualizations tailored towards use in publications directly from their Keras code.

ACKNOWLEDGMENTS

This work was supported by the Carl-Zeiss Scholarship for Ph.D. students.

REFERENCES

- [1] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 1520–1528.
- [2] D. Teney and M. Hebert, "Learning to extract motion from videos in convolutional neural networks," in *Proc. Asian Conf. Comput. Vis.*, 2016, pp. 412–428.
- [3] H. Jiang, D. Sun, V. Jampani, M.-H. Yang, E. Learned-Miller, and J. Kautz, "Super slomo: High quality estimation of multiple intermediate frames for video interpolation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 9000–9008.
- [4] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, Dec. 2017.
- [5] G. Strezoski and M. Worring, "Plug-and-play interactive deep network visualization," *Proc. Vis. Anal. Deep Learn.*, 2017, pp. 100–106.
- [6] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 3431–3440.
- [7] O. Nalbach, E. Arabadzhyska, D. Mehta, H.-P. Seidel, and T. Ritschel, "Deep shading: Convolutional neural networks for screen space shading," *Comput. Graph. Forum*, vol. 36, no. 4, pp. 65–78, 2017.
- [8] P. Henzler, V. Rasche, T. Ropinski, and T. Ritschel, "Single-image tomography: 3D volumes from 2D cranial x-rays," *Comput. Graph. Forum*, vol. 37, no. 2, pp. 377–388, 2017. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13369>
- [9] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Conf. Operating Syst. Des. Implementation*, 2016, pp. 265–283.
- [10] K. Wongsuphasawat, "Visualizing dataflow graphs of deep learning models in tensorflow," *IEEE Trans. Vis. Comput. Graphics*, vol. 24, no. 1, pp. 1–12, Jan. 2018.
- [11] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, 2014, pp. 675–678.
- [12] D. Gschwend, "Netscope quickstart," 2017. [Online]. Available: <http://dgschwend.github.io/netscope/quickstart.html>
- [13] F. Chollet *et al.*, "Keras," 2015. [Online]. Available: <https://keras.io>
- [14] T. Gheorghiu, "Annvisualizer," 2018. [Online]. Available: <https://github.com/Prodicode/ann-visualizer>
- [15] L. Roeder, "Netron," 2018. [Online]. Available: <https://github.com/lutzroeder/Netron>
- [16] Q. Wang, J. Yuan, S. Chen, H. Su, H. Qu, and S. Liu, "Visual genealogy of deep neural networks," *IEEE Trans. Vis. Comput. Graphics*, vol. 26, no. 11, pp. 3340–3352, Nov. 2020.
- [17] M. Crowe, "Neurovis," 2018. [Online]. Available: <http://neurovis.mitchcrowe.com/>
- [18] D. Smilkov, S. Carter, D. Sculley, F. B. Viégas, and M. Wattenberg, "Direct-manipulation visualization of deep networks," *CoRR*, vol. abs/1708.03788, 2017. [Online]. Available: <http://arxiv.org/abs/1708.03788>
- [19] A. W. Harley, "An interactive node-link visualization of convolutional neural networks," in *Proc. Int. Symp. Vis. Comput.*, 2015, pp. 867–877.
- [20] M. Kahng, N. Thorat, D. H. P. Chau, F. B. Viégas, and M. Wattenberg, "GAN lab: Understanding complex deep generative models using interactive visual experimentation," *IEEE Trans. Vis. Comput. Graphics*, vol. 25, no. 1, pp. 310–320, Jan. 2019.
- [21] M. Liu, J. Shi, K. Cao, J. Zhu, and S. Liu, "Analyzing the training processes of deep generative models," *IEEE Trans. Vis. Comput. Graphics*, vol. 24, no. 1, pp. 77–87, Jan. 2018.
- [22] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu, "Towards better analysis of deep convolutional neural networks," *IEEE Trans. Vis. Comput. Graphics*, vol. 23, no. 1, pp. 91–100, Jan. 2017.
- [23] H. Zeng, H. Haleem, X. Plantaz, N. Cao, and H. Qu, "CNNcomparator: Comparative analytics of convolutional neural networks," 2017, *arXiv* : 1710.05285.
- [24] D. Bruckner, "ML-o-scope: A diagnostic visualization system for deep machine learning pipelines," Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2014-99, 2014.
- [25] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. P. Chau, "ActiVis: Visual exploration of industry-scale deep neural network models," *IEEE Trans. Vis. Comput. Graphics*, vol. 24, no. 1, pp. 88–97, Jan. 2018.
- [26] W. Ding, "Draw convnet," 2018. [Online]. Available: https://github.com/gwding/draw_convnet
- [27] Y. Uchida, "Convnet drawer," 2019. [Online]. Available: <https://github.com/yu4u/convnet-drawer>
- [28] A. Lenail, "Nn-svg," 2018. [Online]. Available: <https://github.com/zfrenchee/NN-SVG>
- [29] F. M. Hohman, M. Kahng, R. Pienta, and D. H. Chau, "Visual analytics in deep learning: An interrogative survey for the next frontiers," *IEEE Trans. Vis. Comput. Graphics*, vol. 25, no. 8, pp. 2674–2693, Aug. 2018.
- [30] J. Yuan, C. Chen, W. Yang, M. Liu, J. Xia, and S. Liu, "A survey of visual analytics techniques for machine learning," *Comput. Vis. Media*, vol. 7, no. 1, pp. 1–34, 2021.
- [31] Scrapinghub, "scrapy," 2020. [Online]. Available: <https://scrapy.org/>
- [32] PyPDF2, "PyPDF2," 2020. [Online]. Available: <https://github.com/mstamy2/PyPDF2>
- [33] C. Clark and S. Divvala, "PDFFigures 2.0: Mining figures from research papers," in *Proc. IEEE/ACM Joint Conf. Digital Libraries*, 2016, pp. 143–152.

- [34] N. Elmquist and J.-D. Fekete, "Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines," *IEEE Trans. Vis. Comput. Graphics*, vol. 16, no. 3, pp. 439–454, May/Jun. 2010.
- [35] H. Shi, J. Dong, W. Wang, Y. Qian, and X. Zhang, "SSGAN: Secure steganography based on generative adversarial networks," in *Proc. Pacific Rim Conf. Multimedia*, 2017, pp. 534–544.
- [36] T. Munzner, *Visualization Analysis and Design*. Natick, MA, USA/Boca Raton, FL, USA: AK Peters/CRC Press, 2014.
- [37] M. St. John, M. B. Cowen, H. S. Smallman, and H. M. Oonk, "The use of 2D and 3D displays for shape-understanding versus relative-position tasks," *Hum. Factors*, vol. 43, no. 1, pp. 79–98, 2001.
- [38] A. Cockburn and B. McKenzie, "An evaluation of cone trees," in *People and Computers XIV-Usability or Else!* Berlin, Germany: Springer, 2000, pp. 425–436.
- [39] M. Brehmer, B. Lee, B. Bach, N. H. Riche, and T. Munzner, "Timelines revisited: A design space and considerations for expressive storytelling," *IEEE Trans. Vis. Comput. Graphics*, vol. 23, no. 9, pp. 2151–2164, Sep. 2017.
- [40] C. Felix, S. Franconeri, and E. Bertini, "Taking word clouds apart: An empirical investigation of the design space for keyword summaries," *IEEE Trans. Vis. Comput. Graphics*, vol. 24, no. 1, pp. 657–666, Jan. 2018.
- [41] J. Mackinlay, "Automating the design of graphical presentations of relational information," *ACM Trans. Graph.*, vol. 5, no. 2, pp. 110–141, 1986.
- [42] M. Causse and C. Hurter, "The physiological user's response as a clue to assess visual variables effectiveness," in *Proc. Int. Conf. Hum. Centered Des.*, 2009, pp. 167–176.
- [43] J. Heer and M. Bostock, "Crowdsourcing graphical perception: Using mechanical turk to assess visualization design," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 2010, pp. 203–212.
- [44] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo, "A technique for drawing directed graphs," *IEEE Trans. Softw. Eng.*, vol. 19, no. 3, pp. 214–230, Mar. 1993.
- [45] M. J. Proulx and H. E. Eggeth, "Biased competition and visual search: The role of luminance and size contrast," *Psychol. Res.*, vol. 72, no. 1, pp. 106–113, 2008.
- [46] M. J. Proulx, "Size matters: Large objects capture attention in visual search," *PLoS One*, vol. 5, no. 12, 2010, Art. no. e15293.
- [47] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *The Craft of Information Visualization*. Berlin, Germany: Elsevier, 2003, pp. 364–371.
- [48] C. Ware, *Information Visualization: Perception for Design*. Berlin, Germany: Elsevier, 2012.
- [49] R. E. Christ, "Review and analysis of color coding research for visual displays," *Hum. Factors*, vol. 17, no. 6, pp. 542–570, 1975.
- [50] P. Network, "materialuicolors," 2018. [Online]. Available: <https://materialuicolors.co>
- [51] B. Wong, "Points of view: Color blindness," *Nat. Methods*, vol. 8, 2011, Art. no. 441.
- [52] W. S. Cleveland and R. McGill, "Graphical perception: Theory, experimentation, and application to the development of graphical methods," *J. Amer. Statist. Assoc.*, vol. 79, no. 387, pp. 531–554, 1984.
- [53] Y. Kim and J. Heer, "Assessing effects of task and data distribution on the effectiveness of visual encodings," *Comput. Graph. Forum*, vol. 37, no. 3, pp. 157–167, 2018.
- [54] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3D object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4490–4499.
- [55] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assisted Interv.*, 2015, pp. 234–241.
- [56] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
- [57] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.
- [58] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: <http://arxiv.org/abs/1704.04861>
- [59] J. Ott, A. Atchison, P. Harnack, A. Bergh, and E. Linstead, "A deep learning approach to identifying source code in images and video," in *Proc. IEEE/ACM 15th Int. Conf. Mining Softw. Repositories*, 2018, pp. 376–386.
- [60] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Representations*, 2015. [Online]. Available: <http://arxiv.org/abs/1409.1556>
- [61] E. Nehme, L. E. Weiss, T. Michaeli, and Y. Shechtman, "Deep-storm: Super-resolution single-molecule microscopy by deep learning," *Optica*, vol. 5, no. 4, pp. 458–464, 2018.
- [62] G. Urban *et al.*, "Deep learning for drug discovery and cancer research: Automated analysis of vascularization images," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 16, no. 3, pp. 1029–1035, May/Jun. 2018.
- [63] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 630–645.
- [64] T. Munzner, "A nested model for visualization design and validation," *IEEE Trans. Vis. Comput. Graphics*, vol. 15, no. 6, pp. 921–928, Nov./Dec. 2009.
- [65] A. Bangor, P. Kortum, and J. Miller, "Determining what individual sus scores mean: Adding an adjective rating scale," *J. Usability Stud.*, vol. 4, no. 3, pp. 114–123, 2009.



Alex Bäuerle received the master's degree in media informatics from Ulm University, in 2017. He is currently working as a research associate with the Visual Computing Group at Ulm University. His current research interests include visualization of neural networks to generate better understanding and tooling around these techniques.



Christian van Onzenoodt received the master's degree in media informatics from Ulm University, in 2017. He is currently working as a research associate with the Visual Computing Group at Ulm University. His current research interests include information visualization with a focus on the perception of visualizations.



Timo Ropinski received the PhD degree in computer science, in 2004, and the Habilitation degree, in 2009, both from the University of Münster. He is currently a professor with Ulm University, where he is heading the Visual Computing Group. Before moving to Ulm he was professor in Interactive Visualization at Linköping University, in Sweden, where he was heading the Scientific Visualization Group.

▷ For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csl.

EXPLORNN: UNDERSTANDING RECURRENT NEURAL NETWORKS THROUGH VISUAL EXPLORATION

Alex Bäuerle, Patrick Albus, Raphael Störk, Tina Seufert, and Timo Ropinski. "exploRNN: Understanding Recurrent Neural Networks through Visual Exploration." In: *The Visual Computer* (2022)

This work is published under the terms of the Creative Commons Attribution 4.0 License (CC BY 4.0), <https://creativecommons.org/licenses/by/4.0/>



exploRNN: teaching recurrent neural networks through visual exploration

Alex Bäuerle¹ · Patrick Albus¹ · Raphael Störk¹ · Tina Seufert¹ · Timo Ropinski¹

Accepted: 10 June 2022
© The Author(s) 2022

Abstract

Due to the success and growing job market of deep learning (DL), students and researchers from many areas are interested in learning about DL technologies. Visualization has been used as a modern medium during this learning process. However, despite the fact that sequential data tasks, such as text and function analysis, are at the forefront of DL research, there does not yet exist an educational visualization that covers recurrent neural networks (RNNs). Additionally, the benefits and trade-offs between using visualization environments and conventional learning material for DL have not yet been evaluated. To address these gaps, we propose *exploRNN*, the first interactively exploratory educational visualization for RNNs. *exploRNN* is accessible online and provides an overview of the training process of RNNs at a coarse level, as well as detailed tools for the inspection of data flow within LSTM cells. In an empirical between-subjects study with 37 participants, we investigate the learning outcomes and cognitive load of *exploRNN* compared to a classic text-based learning environment. While learners in the text group are ahead in superficial knowledge acquisition, *exploRNN* is particularly helpful for deeper understanding. Additionally, learning with *exploRNN* is perceived as significantly easier and causes less extraneous load. In conclusion, for difficult learning material, such as neural networks that require deep understanding, interactive visualizations such as *exploRNN* can be helpful.

Keywords Neural network education · Recurrent neural networks · Sequential data · Visual education

1 Introduction

With its recent advances, DL has gained immense traction in research, industry, and education. As job opportunities related to machine learning are unprecedented, many want to learn about and understand DL technologies.

While initial progress in DL was mainly possible due to the rise of convolutional neural networks (CNNs), large training data sets, and GPU training in the context of image recognition [1–3], other network architectures, such as RNNs, which are able to process sequential data, are becoming increasingly important. At the same time, these more advanced learning architectures are more difficult to comprehend, as they employ concepts that are fundamentally different from classical computer science. Thus, by making the process behind RNNs transparent and easy to understand, research

in sequential learning tasks can be accelerated as the field opens up to additional users and contributors.

Along this line, the visualization community has shown how interactive visual explorables can be effective for learning about DL concepts [4–7]. Since different architectures come with their unique challenges, existing educational applications usually focus on one type of architecture. Unfortunately, the set of existing applications still does not cover RNNs. This is despite the fact that RNNs are widely adopted in tasks such as speech processing [8,9], handwriting recognition [10], and machine translation [11], among many others. While RNNs are capable of solving such sequential tasks, they also bring their unique architectures and concepts to capture temporal information. As these concepts differ from other network types, RNN education could be of great benefit. To facilitate RNN education, we propose *exploRNN*, an interactive exploratory visualization for RNNs that runs directly in any modern web browser (Fig. 1).

The focus of *exploRNN* is to make learning about these abstract and complex network types easier, more motivating, and more applicable to real problems. By presenting

Alex Bäuerle
alex@a13x.io

¹ Ulm University, Ulm, Germany

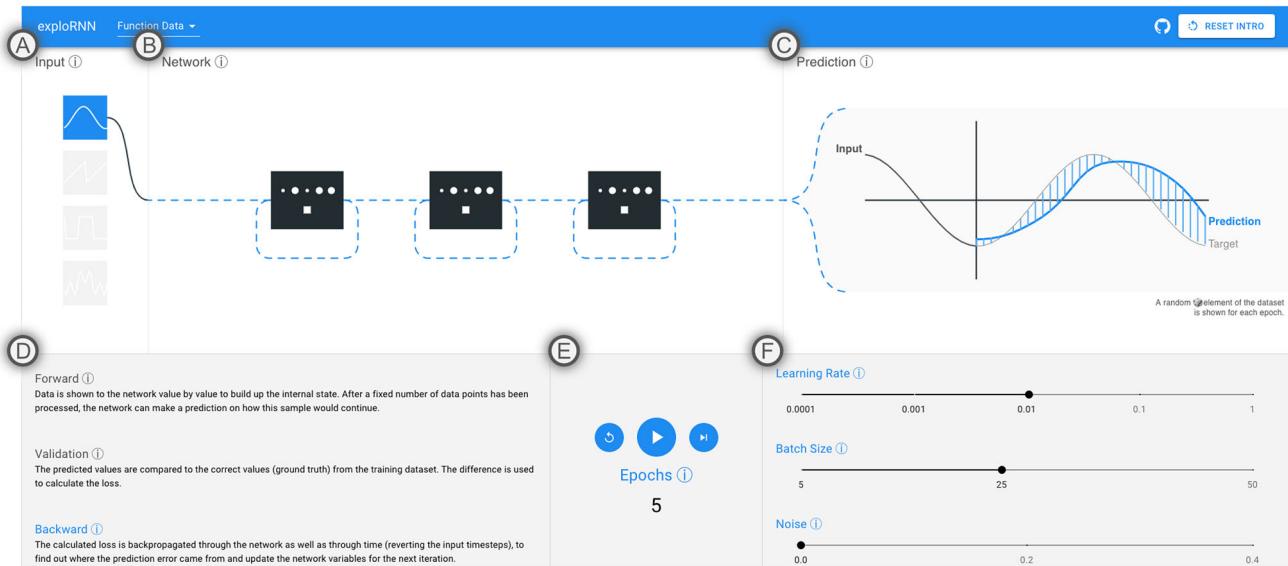


Fig. 1 **a** Simple input types illustrate the abstract concepts behind RNNs. **b** An animated, modifiable network architecture shows the data flow. **c** The prediction visualization shows the network input, prediction, ground truth, and error bars, all animated to communicate their

temporal nature. **d** Text helps explain the training process. **e** RNNs can be interactively trained. **f** Training parameters can be interactively explored

learning material in a way that is conducive to learning, learners should need fewer unnecessary cognitive resources (CR) [12]. These freed-up resources are then available to be used for a deeper understanding (DU) of the learning material. We also expect that this would result in a more motivating and joyful (MJ) learning experience compared to traditional learning methods, such as text. In turn, learners might be willing to spend more time learning, and more learners could be attracted in general, effectively increasing overall knowledge gain. To assess these hypotheses, we compare *exploRNN* with text-based learning in a between-subjects study with 37 participants. Our evaluation provides insights into when, and under which conditions, visual interactive learning environments can outperform conventional learning material.

Along this line, we make the following contributions:

- *Educational Objectives and Design Challenges* for educational RNN visualizations, informing our visualization design.
- *An interactive visualization approach for RNN education*, enabling investigation at different levels of granularity.
- *A quantitative, comparative evaluation*, investigating the effectiveness of our approach and providing hints for other interactive educational visualizations.

exploRNN can be accessed online at: <https://mi-pages.informatik.uni-ulm.de/explornn>, contributing to a fast-growing corpus of visualization work in the field of DL. To our

knowledge, *exploRNN* is the first educational visualization interface that is targeted at RNNs, an important and growing class of neural networks (NN). Additionally, our study is the first to compare conventional learning material to a visual, interactive learning environment for DL education.

2 Background: RNNs

We would like to invite readers who want to refresh their knowledge on RNNs to use *exploRNN* at <https://mi-pages.informatik.uni-ulm.de/explornn> as an interactive learning experiment. This chapter contains a brief summary of the knowledge that is communicated in *exploRNN*.

CNNs and multi-layer perceptrons (MLPs), which are used for most classical DL tasks, process data in a feed-forward manner. On the contrary, RNNs provide a cyclical architecture in which the output of the previous timestep is used in combination with new inputs to inform the activation of a cell. The main difference in training RNNs is backpropagation through time (BPTT), where the prediction error is propagated not only back through the layers but also within the recurrent connections of the layers.

We visualize the LSTM architecture (cf. Fig. 2). Although this is not the most simple recurrent architecture that exists, it is superior in capturing long-range dependencies, as it mitigates the vanishing gradient problem [13–15], and is thus widely used. The main features of an LSTM cell are the gating mechanisms and the cell state. The three gates within an

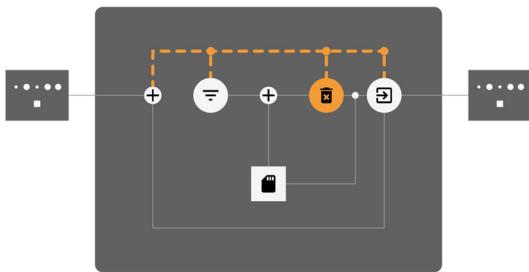


Fig. 2 LSTM cell with all operations visualized. The input is added to the output of the previous time step and then used by the three gates for the gate activation

LSTM cell are computed based on the input at time step t , x^t and the activation of the cell at time step $t - 1$, a^{t-1} as follows:

☒ Input Gate: $i^t = \text{sigmoid}(W_{ix}x^t + W_{ia}a^{t-1} + b_i)$, what new information to use to update the cell state c^t .

☒ Forget Gate: $f^t = \text{sigmoid}(W_{fx}x^t + W_{fa}a^{t-1} + b_f)$, what information in the cell state c^t can be forgotten.

☒ Output Gate: $o^t = \text{sigmoid}(W_{ox}x^t + W_{oa}a^{t-1} + b_o)$, what part of the cell state c^t is used to compute the activation.

The cell state ☐ at timestep t is then computed as $c^t = f^t \circ c^{t-1} + i^t \circ \tanh(W_{cx}x^t + W_{ca}a^{t-1} + b_c)$, where \circ is the hadamard product.

While there are other architectures that also use the concept of cell state and modular updating, such as gated recurrent units (GRUs) [16], their underlying idea does not greatly differ. However, since LSTMs were the first to introduce the explained concepts, are more general in their application [17], and often outperform GRUs [18], we focus on conveying the LSTM architecture.

3 Related work

In this section, we first give a brief overview of the explorable explanation literature before elaborating on the corpus of related work in the area of educational visualizations for DL non-experts and RNN visualizations.

3.1 Explorable explanations

Explorable learning environments were invented long before DL raised awareness in the broader public. Their effectiveness was investigated in the line of work by Hundhausen et al. [19,20]. Schweitzer and Brown then described design characteristics and an evaluation of active learning settings in classrooms by using visualization [21]. There also exists a line of work on the use of visualization for programming education [22–24]. These approaches show how visualization can communicate algorithmic thinking effectively. We combine these ideas with more recent concepts, which have

been proposed under the term *exploranation* in the area of science education [25], where explorable explanations provide benefits for learning.

There are numerous helpful visualizations conveying properties of NN architectures, their functionality, or application scenarios [5,6,26]. However, we will focus on educational, explorable visualization approaches that have been proposed for different network types. One of the most prominent interactive educational visualization approaches has been proposed by Smilkov et al. [7]. In their explainable *Tensorflow Playground*, one can select the properties of a NN to be trained. They also allow the customization of certain training parameters and deployed their approach as a web-based application. Similarly, in *Revacnn*, users can explore the activations of a CNN by modifying the network structure and training the network in the browser [27]. While these approaches help teach the most basic concepts of MLPs and CNNs, respectively, more advanced architectures need further, specialized visualizations. Another approach that is closely related to ours, but works on a different type of NNs, namely GANs, is *GanLab* [4]. They focus on how the generator and discriminator are used adversarially to yield synthetic data that resembles the data distribution it was trained on. However, GANs bring their own visualization challenges, which are fundamentally different from those we found for RNNs. Additionally, neither of these systems was systematically and quantitatively evaluated.

As none of these visualization approaches is designed to help non-experts understand how RNNs function, with their unique concepts of memory and temporal dependence, our aim is to fill this gap in the literature with *exploRNN*. Additionally, we shed light on the usefulness of such interactive learning environments through our quantitative user study. We specifically examine the difference in learning outcomes across different learning hierarchies [28], the complexity of the learning experience by means of cognitive load [12], and qualitative assessments such as motivation, perceived quality of content, and joy throughout the learning process. We hope that our findings in this area can be an indication for similar learning environments and motivate others to conduct similar experiments.

3.2 RNN visualizations

Apart from educational visualizations, there is another line of visualization work targeted toward investigating RNNs. These approaches are designed mainly for researchers who want to understand and debug their models. An early approach toward visualizing RNNs was proposed by Karpathy et al., who visualize the activation of RNN cells for expert analysis [29]. Strobelt et al. published *LSTMVis*, in which the hidden state dynamics of RNNs is investigated [30]. They specifically demonstrate how text understanding can

be analyzed through investigating the structure and change of the cell state. They also presented *Seq2Seq-Vis*, in which sequence-to-sequence models can be probed to reveal errors and learned patterns [31]. Along the same line, Ming et al. introduced RNNVis [32]. They analyze the functionality of individual hidden state units by observing their reaction to specific text segments. With RNNbow, Cashman et al. published a visualization, in which the gradients of RNNs can be analyzed [33]. They attribute the gradient to individual letters in a textual input sequence. This way, researchers can inspect how their models learn. In another approach, Shen et al. proposed visualizations for RNNs [34] operating on multi-dimensional sequence data. Here, developers can inspect hidden unit responses to get insight into different networks. Similarly, Garcia and Weiskopf proposed a visualization for the inspection of hidden states of RNNs [35]. However, all approaches described here are expert tools that help during development. Contrary to this, we aim to convey the general idea of RNNs to novices in this area of DL.

Insights on the effects of using exploration and visualization for learning in general, as well as present educational visualizations for NNs, show how interactive exploration can help a broader audience with access to learning experiences. Therefore, we propose *exploRNN*, which provides insight into the function of RNNs for users who know the basics of DL, but are laymen in the area of sequential learning. Our evaluation also provides the first comparative analysis of interactive learning environments and classical learning approaches for NN education.

4 Educational objectives

To inform the visualization design of a learning experience, educational objectives are needed, which we defined based on Bloom's taxonomy [28]. Our target users already understand the fundamental concepts of DL and know about feed-forward NNs. Without this background knowledge, the theory behind those techniques would first need to be explained, which would extend the scope of *exploRNN*. As our target audience aims to learn the yet unknown concepts of RNNs, we focused on *recall* O1&2, *comprehension* O2&3, and *transfer* O3&4 of the learned information. Later, this learning can be applied in the wild to access levels four to six (*analyze, evaluate, create*) of Bloom's taxonomy. Formulated on this basis, our educational objectives are:

O1 Justification. Users should know that RNNs, in contrast to other network types, can be used for sequential data. This also includes BPTT, through which RNNs can learn temporal dependencies, which classical feed-forward networks cannot.

O2 Cell Structure. Users should then understand how LSTM cells are built and what functionality their individ-

ual components have. Here, the cell gates, as well as the memory element, are of special importance, as they enable the processing of sequential data.

O3 Training Setup. To understand the training process of such networks, users should know important parameters for the setup of RNNs. This includes the network structure, training parameters, and how data are fed to the network.

O4 Task. Finally, to transfer this theoretical knowledge about RNNs to real applications, users should learn about different application areas and data types that can be used with RNNs. In the end, they should be able to describe how RNNs could be used for their own application scenarios.

Similarly to a lecture at a university or a textbook, our learning environment is designed to provide an introduction to RNNs from which interested users can start experimenting with the techniques. Accordingly, our educational objectives not only motivate the importance of RNNs, but are also aimed at providing insights about the input data and related tasks, as well as how the training process and LSTM cells work.

5 Design challenges

Since RNN cells are a special form of NN layers, they open up unique challenges for visualization-based education. We observed both visualization design challenges and technical challenges, which we describe in this section.

5.1 Visualization design challenges

We first discuss the following visualization design challenges that we identified in the context of an interactive learning environment aimed at RNNs and illustrate how they relate to our educational objectives:

V1 Complexity. As mentioned in Sect. 1, one of our central goals is to simplify learning by reducing cognitive load [12]. However, RNNs are typically trained on a large amount of complex data that can be difficult to grasp O1 O4 [36,37]. The same holds for network architectures, which are also often too complex to fully comprehend in their entirety O3 [38,39]. Consequently, all visualizations must be interpretable and intuitive, but realistic enough to form a compelling use case [40,41].

V2 Dynamics. An educational system to teach RNN concepts should clarify the dynamics of the sequential data on which these networks operate O1, as well as the dynamics of the training process O3. These dynamic processes must be visually communicated, including data type and data processing, both forward (inference) and backward (back-propagation), within the network [42].

V3 Multiscale. RNN structures need to be communicated at different granularities, i.e., network, cell, and cell components O2. These multiple scales need to be fluidly

inspectable, while at the same time, the granularity at which the user currently operates must be communicated [41,43].

V4 Supervision. In classical learning settings, teacher supervision or other opportunities to seek further information is provided. Contrary to this, *exploRNN* is designed as a standalone learning environment that does not require external guidance O1-4. Thus, supervision has to be substituted by visual guidance [44,45].

5.2 Technical challenges

Whereas the visualization design challenges are based directly on our educational objectives, the following technical challenges relate to the development of such an interactive, explorable learning environment:

T1 Training Time. Typically, training processes can take up to several days to convergence [46,47]. However, for an interactive learning experience, waiting days for convergence is not feasible. To provide direct feedback to the user, our networks thus have to converge in minutes instead of hours or days.

T2 Training Steps. Normally, computation is done as fast as possible to minimize the time it takes for the network to converge. However, we want the user to be able to follow the training process and observe individual training steps [44]. Thus, training steps should be separated temporally from the visualization.

T3 Deployment. Modern-day learning is often conducted via online courses, blog posts, or explainable web pages [48]. Although this makes such learning environments accessible to a broad audience, it also limits the technical freedom of such applications [49]. Therefore, educational environments should be deployed to a broad audience, while also providing diverse functionality.

6 Visualization design

In the following, we discuss the visualization design of *exploRNN*.

We explain how we tackle the aforementioned visualization challenges Vx and learning psychology goals CR/DU/MJ while targeting the educational objectives Ox defined in Sect. 4. We first describe the overall visualization concepts we implemented for *exploRNN*. Then, we elaborate on the different views of our environment in the upcoming subsections.

Scales To show both an overview of the training process O3 and give detailed insight into the computation that is performed within one recurrent cell O2 DU, we employ an overview first, zoom and filter, then details on demand visualization design, following Shneiderman's mantra V3 [43]. Therefore, *exploRNN* consists of two main views, the network overview (Fig. 1), which displays the training progress

on the network scale, and the LSTM cell view (Fig. 6), which allows for a detailed inspection of an LSTM cell. This is in line with our goal of reducing complexity CR by focusing on individual steps of the learning process rather than presenting everything at once.

Animation Animation has shown to be effective in visualizing data relationships and algorithms O1-3 [42,50,51]. Furthermore, animation has shown to be associated with fun and excitement [52], which is in line with our goal of making learning more enjoyable MJ. Thus, to visually communicate how the network operates on sequential data, we use animation throughout our visualizations V2.

Onboarding Novel visualizations and interactive systems can be hard to understand [53]. We designed *exploRNN* in a way that allows exploration without running the risk of making irreversible errors or needing teacher supervision V4. However, instructional aids may be important to understand such complex content DU [54]. Therefore, we use an onboarding process for our educational environment [55] (cf. Fig. 3). With this process, we aim to further reduce the cognitive load during learning compared to classical learning environments CR [56]. For example, the sequential nature of RNNs O1 V2 and the data and tasks that RNNs can be used for O4 are communicated in *exploRNN*.

Textual explanations In contrast to other learning environments, which show static textual explanations below the main visualization [4,7], we instead provide such additional information as *details on demand* V3 [43]. This way, users can access more information for exactly the components they want to learn more about DU, while not having to read a lot of text CR. Our interactively explorable dialog boxes, as shown in Fig. 4, provide information about all important elements of the learning environment. Such dialogs exist for all headings and are anchored through an ⓘ icon, and for all components of an LSTM cell, which is referred to in our onboarding process.

6.1 Network overview

In the network overview, following the natural reading direction of western cultures, as well as related work on NN architecture visualization [57–59], we arrange the network from left to right. On the left, one can see the input type that is currently used to train the network A. Centered, we present an abstracted visualization of the network, where users can see how many layers the network contains B. On the right, a visualization of the prediction along with the prediction error shows how training progresses C. Below these visualizations, information about the training process, controls for the training process, and means to change training parameters are shown D-F.

A Input. To experiment with the network, users can select the input data that is used to train the network from

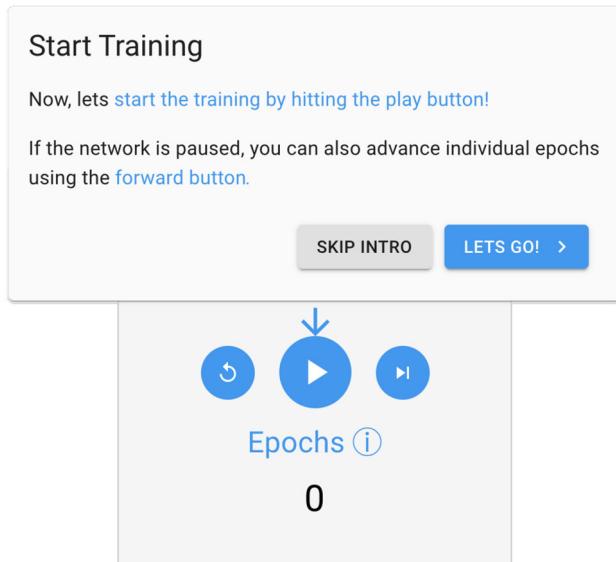


Fig. 3 Onboarding dialogs guide the user through our visualizations, so that no manual introduction is needed, and the user can explore *exploRNN* on their own. Textual descriptions with highlights provide detailed explanations for individual components. Positioning and arrows reveal associations between dialogs and components

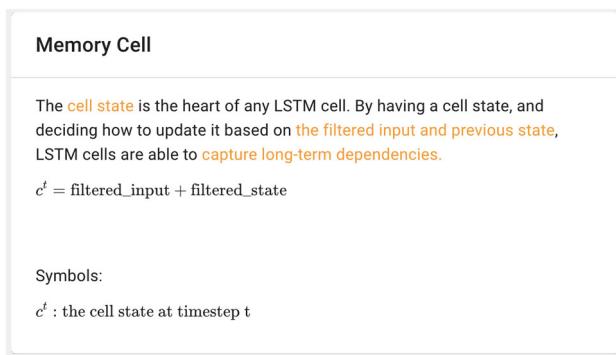


Fig. 4 Users can access more detailed explanations for many elements of our visualizations such as training steps, hyper-parameters, and operations in a cell

a set of explanatory input types. Data for an interactive and explainable visualization of NNs needs to both explain the network functionality (O1) and be easy to understand (V1). Therefore, current educational visualizations use an abstract, two-dimensional distribution of points to train their networks on [4,7]. With *exploRNN*, we follow this approach of employing data that is as simple as possible (CR). As RNNs are focused on sequential data, we decided to use periodical mathematical functions and simple text snippets, which map nicely to the sequential nature of RNNs (V2). The functions that can be used as training data in *exploRNN* are a sinusoidal function, a sawtooth function, an oscillating function, and a composite sinusoidal function and vary in their periodicity. To demonstrate the sequential and dynamic nature of these

input functions, we animated those that are in use so that they seem to flow while being input to the network (MJ).

In addition to abstract function continuation, we also provide text-based data to train the network on (DU). To allow for interactive training, we employ rather simplistic text samples. These include a recurring character sequence (*ababab...*) and the well-known text *lorem ipsum*. Here, we employ a similar design language as with function data, to show that most ideas can be transferred across tasks. By incorporating this text learning scenario, users of *exploRNN* get to learn and inspect not only abstract problems, but can also experience more realistic scenarios (O4) (MJ).

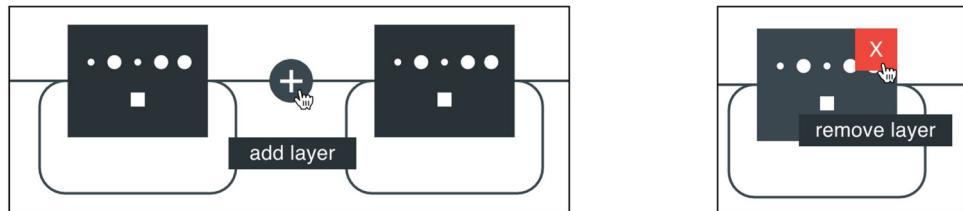
(B) *Network*. In the network visualization, we want to communicate the recurrent nature of our network (O1), but at the same time, show all layers. Thus, instead of the more frequently used unrolling of RNN layers [60], we add a loop to the layer glyph to symbolize this recurrence. This symbolizes the feedback loop of information output at t back to the input of a cell at $t + 1$ (V2), which enables BPTT.

Our network visualization is animated as data flows through its layers (O3) (MJ). For the prediction step, dashed lines flow in the forward direction to symbolize forward data processing. For the backpropagation step, they flow backward to resemble the backpropagation of the error (V2). Dashed lines are moving from input to output during the prediction phase, and from output to the first network layer during training, because backpropagation is not applied to the input domain.

Users can also investigate how the training progresses differently depending on the number of recurrent layers in the network (O3). Therefore, layers can be added or removed from the network to be trained, as shown in Fig. 5 (DU). As with most exploratory components, we explain the implications of this in our introduction, and users can click the (i) next to the network heading.

(C) *Predictions*. Commencing the top row of visualizations is the data plot, where we visualize an input sample and the prediction of the network along with its ground truth (O3). Here, multiple data points that are processed by the network one after the other are used to inform a prediction, which is visualized by sliding a gray box over the input data that is currently processed (V2). Additionally, the prediction values slowly build up with animations to clarify that this prediction is building up sequentially (MJ). We then use vertical lines in the function plots, which slowly emerge between the prediction and the target value. This vertical line encoding is in analogy with the way we calculate errors, namely, by looking at the prediction values and calculating the difference to the ground truth (DU). The error calculation is embedded temporally between the inference (forward network animation) and backpropagation (backward network animation) phases of the training process (V2). Altogether, through this animated component, while not being interactive itself, users

Fig. 5 left: Adding a layer between two existing layers. right: Removing a layer from the network



can inspect the results of modifications that have been made in other places (O3).

(D) *Process*. According to the typical NN training setup, we divide the training process into three distinct steps: inference (forward), validation (error calculation), and back-propagation (backward). The explanation pane in the lower left of the network overview (see Fig. 1) displays which step is currently executed and provides an explanation of what happens in each of these steps (DU). Through this, the user can learn more about the training dynamics of the network (O3). As described previously, animations in other components complement this dynamic nature of the training process (V2).

(E) *Controls*. In the network overview, the network is trained by means of epochs to *first* provide an *overview* [43] of the training process (V3) (CR). To experiment (O3), users can interact with the control area in the bottom center of our environment (MJ). In addition to automatically advancing epochs, which can be controlled with the ▶ and II buttons, users can also trigger network training for a single epoch, by pressing the ▶ button (DU). The training process can always be reset using the ⌛ button. A back button to go to a previous epoch is not included in *exploRNN* as this would require saving multiple previous states of the network parameters, which would require significant browser memory. Therefore, and as individual epochs normally do not change the network behavior completely, going back one training epoch during training is not a common operation during neural network training, so we think users will not miss such functionality.

(F) *Training Parameters*. To communicate the training setup of an RNN (O3), a trade-off between completeness and simplicity must be made (V1). Thus, we let the user freely choose some training parameters, but employ restrictions for others (CR). As mentioned, users can add or remove individual network layers and use different preset training inputs. In addition, they can change the learning rate, batch size, and noise (DU). The learning rate and batch size allow for exploration of different training settings (O3). Noise can be added to make the training data more realistic, resembling real-world scenarios of imperfect measurements (O4). Parameter changes can be made through sliders, which are positioned on the bottom right. To provide an intuition about the influence of these parameters, we include pretrained models that are loaded during the onboarding steps which explain each indi-

vidual parameter (V4). Other parameters, such as units per layer or optimization strategies, cannot be changed in our implementation. This trade-off between freedom of exploration and simplicity proved to be effective in educating users about the influence of different training parameters and keeping their cognitive load low (V1).

Hierarchical aggregation can help simplify visualization designs (CR) [61]. Thus, after getting an overview of the network, the user can inspect another hierarchy level in detail, namely individual LSTM cells (O2) [43]. When selecting one of the layers in the network overview a zooming transition onto one of the network layers gradually reveals the structure of an LSTM cell to support the user's mental image of looking into one of the layers (V3) (CR). With this multiscale approach, where users can navigate between views, orientation is important (CR). Therefore, a color coding indicates the current level of detail. This highlight color is blue for the network overview, whereas orange is used for the LSTM cell view. Orange and blue are complementary colors, which makes them easily distinguishable, and they can be differentiated by vision-impaired users [62].

6.2 LSTM cell view

In the LSTM cell view, we show a detailed visualization of the selected cell on the left, embedded in small pictograms of neighboring cells (G). On the right of this cell visualization, one can see the input, target, and prediction values of the network, where new points are added as they flow through the cell (H). Below these visualizations, we show information about the training process, controls for the training process, and means to change training parameters, similar to the network overview (I-K).

(G) *Cell Architecture*. To convey the functionality of one recurrent unit (O2), we show all computational elements within a cell (DU). Wherever information is combined, we show a + icon. Icons for the input (=), forget (☒), and output gates (☒) visualize the gating functionality of an LSTM cell. While all gates that transform the data are depicted with circular icons, the cell state, which represents the saved state of the cell, is represented by a squared ☒ icon, illustrating the semantic difference between these components. Each of these cell components can be selected to get a detailed expla-

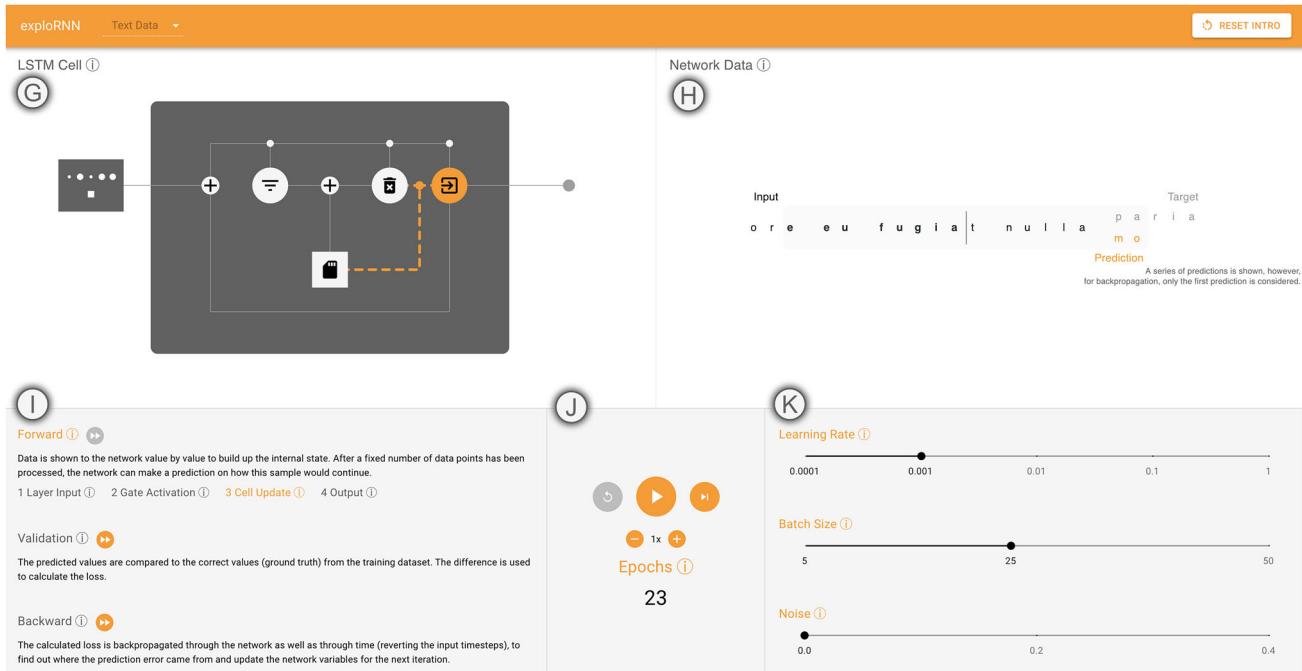


Fig. 6 LSTM cell view. **G** Visualization of data flow through the cell. **H** Input to the network and its prediction. Visualization of the training error computation. A gray sliding window indicates which data points are needed to initialize the cell state. **I** Explanations with more detailed

steps for the forward direction of data flow. **J** In addition to interactively training the network, users can change the speed at which the visualization for cell steps advances. **K** Just as in the network overview, users can modify training parameters

nation of its functionality, as shown in Fig. 4, marking another level of detail in this visualization **V3** **CR**.

Data flow is visualized through connecting lines **O2** and step-by-step animations of the cell components **MJ**. Here, elements that process data in the currently visualized computation step are highlighted. As in the network overview, connections moving data are symbolized with dashed lines. Those lines flow forward during inference and backward during backpropagation. This way, we communicate how the hidden state and output of these cells is computed and visualize how the data flows from one to the next operation or gate **V2** **DU**.

The reverse data flow of BPPT occurs not only once within a cell to backpropagate to the previous layer, but multiple times, for all input time steps **O1**. The connections within the cell also clarify that there are two recurrent cycles, one from the output of the cell back to the input, and one within the cell to update the cell state based on its state in the previous iteration **DU**. As a result, while other visualizations require *unrolling*, where time steps are visualized by displaying multiple cells in concatenation [60], we communicate recurrence through step-by-step animation. This removes the ambiguity of stacked layers vs. unrolled cells, which was shown to hinder learners in our first experiments **V1** **CR**.

H Data Plot. Right of the cell visualization, we show the input data, network prediction, and ground truth all in

one graph. In contrast to the network overview, where the network is directly connected to this output graph, the cell is disentangled from this visualization. As the depicted cell typically receives data from previous cells and outputs data to subsequent cells, this visualization, where animation steps are synchronized but not visually connected on both the input and output side, better reflects the network architecture of RNNs **V3** **DU**. Users can inspect this view during interactively controlled training to see how the network processes input data to make predictions sequentially and how it calculates the training loss in relation to the processing steps within a cell **O1&2** **V2**.

I Training Process. The three steps of inference, validation, and backpropagation are just as relevant in the LSTM cell view as they are in the network overview **O3**. As the training speed is lower in the LSTM cell view, users can skip part of the data processing and go directly to the processing step of interest **V1&2**. For the forward pass, we add additional explanations for the different processing steps of *receiving the layer input*, *calculating the gate activations*, *updating the cell state*, and *outputting the activation value* **DU**. These explanations are highlighted in synchronization with the processing steps during the forward pass to the data flow in the cell visualization above **MJ**, allowing users to draw links between the processing steps and the explanations they are interested in **V2&4** **CR**.

J Controls. In the LSTM cell view, processing is done by means of compute steps, showing a much more detailed processing pipeline than in the network overview **V3**. As in the network overview, the control area can be used to experiment with the training process **O3**. The more fine-grained advancement of the visualization is also adopted by the degree to which the animation advances with the forward button, since it only executes the next compute step within a cell **DU**. In addition to what can be done in the network overview, the speed of the animations for data processing within this cell can be adjusted, so that users can explore the processing steps at their own pace **V2**.

We want to emphasize the buildup of state within a cell based on multiple input time steps. Thus, we show how the network processes these inputs in great detail, whereas we made the animation of the backpropagation take less time than forward processing. As *exploRNN* is not designed to represent accurate timings anyway, this is our way of visualizing cell processes in detail, while also preserving the ability to observe multiple epochs.

K Training Parameters. Training parameters can be adjusted in the LSTM cell view just as in the network overview, giving the user even more control over the training process and room for experimentation **O3**.

To get back to the network overview, one can click anywhere outside of the LSTM cell in Fig. 6 **G** **V3**.

7 Technical realization

While the visualization design described above has been carefully crafted to meet the educational objectives described in Sect. 4 and the visualization design challenges outlined in Sect. 5.1, its technical realization needs to take into account the technical challenges identified in Sect. 5.2. In this section, we detail how we tackled these technical challenges.

T1 Training Time. While an RNN for a complex application cannot be trained live in the browser, we simplify the problem in multiple ways. By employing simplistic data sets, the model can converge after relatively few epochs. Additionally, we limit the number of data points that are fed to the network per epoch. Therefore, epochs are processed sufficiently fast for our interactive visualization approach. We also limit the network size to at most seven layers, so that memory consumption and processing time are reduced. In turn, users can see the training progress and get visible prediction improvements after only a few epochs, while one such epoch takes seconds to compute.

T2 Training Steps. A key aspect of our approach is the decoupling of computation and visualization. Through this decoupling, we are able to show the training steps in an observable manner and enable exploration at the user's own

pace. This helps users understand how the model processes input data and predicts new data points.

T3 Deployment. To be able to make *exploRNN* publicly available for a large audience, we implemented it as an interactive browser application using HTML and JavaScript. To train the RNN, we use TensorflowJS [49], for animated visualizations of the trained network, we use P5.js [63]. This way, we are able to provide an interactive, web application that visualizes the training dynamics of RNNs through animation, which is accessible at: <https://mi-pages.informatik.uni-ulm.de/explornn/>.

8 Limitations

While *exploRNN* provides a novel environment for learning about RNNs, there is still room for more advanced visualization designs that could be explored in the future. Some of these limitations are explained in the following.

Explanations *exploRNN* offers a lot of experimentation that is complemented by textual explanations. However, the number of textual explanations that fit into the context of such an educational system, which is designed to provide an overview of this complex topic, is insufficient to fully explain RNNs. For specific questions that are not addressed by our interactive system, we refer to developer documentation and scientific papers.

Drill-Down *exploRNN* explains RNNs on both a network and a cell level. Apart from seeing the data flow on these granularities and textually describing the components of a cell, visualizing the workings of these components could further benefit the learning experience. However, these components are just mathematical functions to which neither the input nor output have a directly discernable meaning. If we were to, e.g., visualize the internals of a memory component, users could only see matrices of seemingly meaningless numbers flowing through these cells. This would not add any benefit and might even result in confusion about such a visualization. To explain these internal components, novel interpretability techniques might help. Inventing and implementing those is beyond the scope of this work.

Component change To see the influence of individual components in a cell, changing or removing them could be an interesting addition to the workflow. We did not implement this capability for two reasons. First, adding such functionality goes deep into the working of individual cells, which would exceed the learning objective of getting an overview of RNNs and LSTM cells. In turn, we assume that changing single components in individual cells is unlikely to have a measurable and interpretable effect on the overall learning outcome. Second, we would have needed to implement our own DL library for this to be possible, as TensorflowJS has predefined LSTM layer implementations.

Degrees of freedom While users can change some hyperparameters and network settings in our environment, we deliberately do not expose all possible settings to our users. The goal of this limited exploration setting is that users can get an overview of important manipulations to be made, while at the same time not overwhelming our target audience. As for limited explanations, we refer to developer documentation and scientific papers for users that want to explore these details.

Layer types In our implementation, we focused on conveying LSTM cells. However, there are numerous other cell architectures for RNNs. Although we do not think this limited focus hinders learners with understanding RNNs on a high level, it would, nonetheless, be helpful for users specifically interested in certain cell types to include these in *exploRNN*.

9 User study

To evaluate the effectiveness of our approach, we conducted a user study with 37 participants (30 male, seven female) aged between 21 and 32. Participants were recruited from a DL course at our local university. Our study was a lecture at the end of the course, after students had already learned about feed-forward NNs. Participants were randomly assigned to one of two groups. The *exploRNN* group received the interactive application, and the text group was presented a text-based learning environment.

To look at learning outcome in detail, our evaluation was divided into the first three distinct, hierarchical cognitive learning goals according to Bloom's taxonomy [28], namely recall, comprehension, and transfer. We expect higher learning outcomes for the *exploRNN* group compared to the text group at all three levels. For a closer look at the cognitive processes involved in learning [CR], we also collected data for the three types of cognitive load [12]. Intrinsic cognitive load (ICL) results from the natural complexity that underlies the learning content. Since the difficulty does not differ, there should be no difference between the two groups. Extraneous cognitive load (ECL) is caused by inadequate instruction or presentation of information. Due to the step-by-step presentation of information and the direct connection of textual information and explanatory figures in the *exploRNN* group, we expect lower ECL for the *exploRNN* group compared to the text group. Lastly, germane cognitive load (GCL) represents the invested learning-related load. GCL is connected to the processes that are needed to construct and automate mental representations [12]. Following the reduced ECL in the *exploRNN* group, learners should have more free cognitive capacity in working memory to invest in learning-related GCL.

9.1 Hypotheses

Based on the described theory, we hypothesize the following. We expect a higher learning outcome, differentiated by recall [H1], comprehension [H2] and transfer [H3] in the *exploRNN* group than in the text group. Furthermore, we expect no differences between the groups for ICL [H4]. We expect a lower ECL in the *exploRNN* group than in the text group [H5]. For the GCL, we expect it to be higher in the *exploRNN* group compared to the text group [H6].

9.2 Method

Our study was split into different steps, which we explain in the order they were presented to the participants.

Prior knowledge. Prior knowledge was measured with seven open-ended questions on NNs and DL techniques (e.g., *Name two activation functions used in deep learning.*). The questions were developed by a domain expert. All answers were rated by a domain expert, following a predefined solution to ensure objectivity. A total of one point could be scored for each question, with partial points of .5. The maximum score for the prior knowledge test was seven.

Motivation (MSLQ). To assess motivation, the MSLQ [64] subscale for motivation was used. The MSLQ is a self-report questionnaire designed for an academic setting. Motivation was measured with twelve items (e.g., *I'm confident I can do an excellent job on the tests in this study.*). Learners were instructed to respond as accurately as possible, reflecting their attitudes and behaviors toward the learning module. Responses were given on a 7-point Likert scale ranging from 1 *strongly disagree* to 7 *strongly agree*. Cronbach's Alpha was computed for the internal consistency of the measures [65], and the reliability was $\alpha = .95$.

Learning material. The learning material was presented either as a text with illustrating figures, formulas, and graphs (see our supplementary material) or through *exploRNN* (see website). For both conditions, the information was the same. The only difference was the presentation medium and the lack of interactivity in text-based learning.

Learning outcome. To assess learning outcome, a domain expert developed a posttest with 11 open questions on the content of the learning session. To better understand cognitive processes, the posttest was differentiated by the first three levels of Bloom's taxonomy [DU] [28]. Recall was measured with four questions (e.g., *Name the backpropagation algorithm that is used for RNNs.*). Comprehension was also measured with four questions (e.g., *Explain the meaning of this formula: $c_t = \text{filtered_input} + \text{filtered_state}$.*) The main purpose of these questions was to test how well people could explain and discuss the learning content. Transfer was measured with three questions (e.g., *Assuming you have a poem continuation network and training data with poems from the*

internet. If your network now makes a prediction, how do you determine if it is correct, to calculate the loss?). These questions were designed to test the ability of learners to draw inferences from the learning content and apply it to new contexts. Similarly to the prior knowledge test, each question was rated by a domain expert, following a predefined solution to ensure objectivity. A total of one point could be scored for each question, with partial points of .5. The maximum score for recall and comprehension was four each, and for transfer, it was three, so the total maximum score for the posttest was eleven. We did an ANOVA on the learning outcome to test for statistical significance.

Cognitive load. To measure cognitive load (CR), the differentiated cognitive load questionnaire was used [66]. It contains two items for ICL, three items for ECL, and three items for GCL, all measured as self-reports on a 7-point Likert scale from 1 *strongly disagree* to 7 *strongly agree*. To measure internal consistency, the Cronbach Alpha was calculated [65]. Reliability was $\alpha = .66$ for ICL, $\alpha = .81$ for ECL, and $\alpha = .77$ for GCL. As for learning outcome, we tested for significance with an ANOVA.

System usability To quantitatively measure the system usability, the System Usability Scale (SUS) was used [67]. This scale is a self-report measurement consisting of 10 items related to the usability of *exploRNN* (e.g., *I found the system very cumbersome to use*). Responses to the items were given on a 7-point Likert scale ranging from 1 *strongly disagree* to 7 *strongly agree*. The internal consistency of this scale was $\alpha = .74$.

Qualitative questions For an impression of the quality of the learning material, further questions were implemented. Three open-ended questions were related to likeability (*What about the learning experience did you like especially, what did you not like?*), missing functionality (*Was there something you would have liked to do but could not?*), and additional comments (*Other remarks*) (MJ). For liking (*I would like to use this learning material*) and recommendation (*I would recommend this learning material to my friends*) of the material, two items could be rated on a 5-point Likert scale from *very unlikely* to *very likely*. Content (*How was the quality of the content?*) and design (*How was the design of the learning experience?*) could be rated with 0–5 stars.

9.3 Results

In the following, we present the results of the user study.

Descriptive data. The analysis of the descriptive statistics showed that subjects of the text group and the *exploRNN* group did not differ in most of the variables. T tests (variances were equal for all variables) with respect to age ($p = .33$), gender (*exploRNN* group 21% females, text group 16.67% females) ($p = .74$), MSLQ ($p = .11$), self-efficacy (MSLQ) ($p = .16$) and duration ($p = .79$) revealed no significant

differences. Motivation (MSLQ) showed a significant *t* test ($p = .02$), indicating that learners in the text group had a significantly higher score. Descriptive data for all variables per condition are given in Table 1.

To analyze whether prior knowledge and MSLQ should be used as covariates, we conducted a correlation analysis with learning outcomes and cognitive load. Significant correlations could be found for GCL with the MSLQ ($r = .37$, $p = .024$) and for the recall of the posttest with the MSLQ ($r = .44$, $p = .006$). Therefore, they were included as a covariate in the following calculations concerning GCL and recall. No other significant correlations for the potential covariates could be found.

Learning outcome. Against our hypotheses, we found a significant difference regarding recall ($F(1, 34) = 3.91$, $p = .028$, $\eta_p^2 = .103$) in favor of the text group but not for comprehension ($F < 1$, n.s.) or transfer ($F < 1$, n.s.).

Cognitive load. Contrary to our expectations, we found a significant difference between text and *exploRNN* group for ICL ($F(1, 34) = 3.85$, $p = .029$, $\eta_p^2 = .099$). ECL showed the hypothesized effect: The *exploRNN* group showed a significant lower score than the text group ($F(1, 34) = 4.33$, $p = .023$, $\eta_p^2 = .113$). Against our hypothesis, GCL was not significantly higher in the *exploRNN* group than in the text group ($F < 1$, n.s.).

System usability. The SUS questionnaire indicates an *excellent* usability ($M = 84.47$, $SD = 9.45$) [68]. Participants also rated our approach as significantly more likable ($F(1, 30) = 10.52$, $p = .003$, $\eta_p^2 = .260$), more recommendable ($F(1, 30) = 11.75$, $p = .002$, $\eta_p^2 = .281$), and better designed ($F(1, 30) = 20.711$, $p < .001$, $\eta_p^2 = .408$) compared to the learning text.

Qualitative questions. We also got some qualitative feedback in our free-form fields. Participants liked our introduction, which apparently made it easy for them to get started with *exploRNN* (*the tutorial was nice and the platform was easy to use*). They also mentioned that the graphical support of these textual explanations was helpful for them to form a mental image of the setting: (*the mental bridge the graphical presentation helped build was helpful in memorizing and understanding*). Some participants said that they did not remember specific names, as it was not important during the usage of *exploRNN*: (*I later did not remember the name of the algorithm that was used, since it was not important during the usage of the tool*). Some participants asked for something similar for other types of networks, e.g., (*I would like to have similar resources to cover other topics from the basics such as MLPs up to advanced topics and more complicated kinds of networks*). As described in Sect. 8, we only support a limited set of interactions, which some participants commented on, e.g., (*[I missed] changing the activation function of the LSTM gates*).

Table 1 Means and standard deviations for our study results separated by groups for all variables

Variable	Text		<i>exploRNN</i>	
	M	SD	M	SD
Duratiton (min):	43.6	26.78	40.87	35.67
Age (years):	24.94	2.96	24.05	2.51
Pre-T: (%)	83.14	13.86	80.0	14.57
Post-T: (%)	68.48	20.46	65.47	14.41
Post-T recall*: (%)	79.75	26.30	63.25	29.64
Post-T comp.: (%)	65.00	51.54	65.50	37.26
Post-T trans.: (%)	58.00	69.54	68.33	41.46
ICL*: (%)	61.51	18.51	48.82	20.51
ECL*: (%)	48.14	23.32	37.09	15.52
GCL: (%)	77.51	13.94	75.94	17.13
MSLQ*: (Max: 7)	5.17	0.95	4.56	1.30
SUS: (Max: 100)	—	—	84.47	9.45
Qualit. Like*: (Max: 5)	3.27	1.03	4.35	0.86
Qualit. Recomm.*: (Max: 5)	3.00	1.20	4.24	0.83
Qualit. Content: (Max: 5)	4.00	0.76	4.35	0.61
Qualit. Design*: (Max: 5)	3.20	0.94	4.47	0.62

Numbers annotated with * indicate a significant difference between the two conditions

9.4 Discussion

In the following, we will refer back to the hypotheses we had before conducting the study and discuss the study outcome. *Learning outcome* We looked at both recall and understanding regarding the learning outcome. In contrast to [H1](#), we found that learners in the text group showed significantly better results for recall. While we found no significant differences between the groups regarding comprehension [H2](#) and transfer [H3](#) the results are interesting nonetheless. Although not significant, the descriptive statistics indicate that the score for transfer is about 10% higher for *exploRNN* compared to text [DU](#). This could be a first indication that learning environments such as *exploRNN* can help learners build a deeper understanding of the subject compared to learning with classic text. However, significant results and further research are needed to support this statement. Compared to recall, these results may indicate that while learners are better at learning terms by heart (surface learning) when they learn with text than with *exploRNN*.

A possible explanation for the better recall performance in the text group could be that learners have more experience with text-reading strategies [\[69\]](#). This might help with the complex terms explained here, as learners might find it easier to find information that was previously presented [\[70\]](#). Thus, designing ways to easily retrieve previously presented information could be an interesting direction of future research for such interactive explorables. Another possibility is that learn-

ing with an interactive environment, which is affirmative and provides information step by step, might infuse the illusion of knowing [\[71\]](#). Learners may think that after a few experiments in *exploRNN*, they have acquired enough knowledge, while there is still much more to explore and learn. In the text group, it is immediately clear to the learner when the text is finished. On the contrary, *exploRNN* can require user initiative for information acquisition. As the learning experience was self-controlled, participants could decide for themselves when to go from the learning content to the posttest. Referring to the illusion of knowing, learners might have felt too competent as they experienced this more guided experience. However, even though learners may be able to transfer what they have learned to other application areas, they may be missing important basic terminology that was presented in the learning material to reflect their knowledge gain in a classical learning test.

Cognitive load Against [H4](#), the *exploRNN* group showed significantly lower ICL than the text group. The perceived difficulty of the learning material is 12.71% lower for *exploRNN* even though the text content was identical in both conditions. This suggests that *exploRNN* makes the learning material appear easier [CR](#). The reason for this could be that the content is presented step by step in the tutorial of *exploRNN*, instead of all at once as in the text condition [\[56\]](#).

The results regarding [H5](#) are consistent with our assumptions. With a large effect size, the *exploRNN* group showed lower ECL than the text group [CR](#). Therefore, *exploRNN* reduced the extraneous cognitive load compared to the text content, although the content was the same in both conditions and there were no unnecessary figures or information in the text. Combined with lower ICL, more cognitive capacity remains for GCL, which is important for learning.

For GCL, we did not find the significant difference we hypothesized in [H6](#). Although ICL and ECL indicate that more cognitive capacity should be free in the *exploRNN* condition, participants did not invest that cognitive capacity into GCL. This could be because there might already be high investment in GCL in both groups. Another explanation could be that since participants in the text group perceived the learning content as more difficult, they might have invested more GCL to compensate for said difficulty.

System usability The results of the SUS questionnaire indicate that our system is easy to use. This supports our proposed visualization and interaction design and shows that our design choice of creating an interactive environment as a learning experience on RNNs matches our target audience well. Additionally, as participants rated our approach as significantly more likeable, recommendable, and better designed, users are likely to experience more joy, and be more motivated when learning [MJ](#). In combination with the reduced cognitive load *exploRNN* inflicts on users, we hope that this could result in a larger number of users willing to

spend their time learning and more time spent learning per user. In turn, we think that this might outweigh the advantage in some areas of learning outcome with the more familiar text-based learning environment. Further longitudinal studies on NN learning systems need to be done to investigate this in more detail.

Qualitative feedback. The open feedback forms also provided interesting insights. In general, participants seemed to like *exploRNN* as a learning experience and even asked for similar interfaces in other contexts. Furthermore, the amount of information and our onboarding process seemed to make *exploRNN* easily usable. Most of the criticism was related to limitations regarding the freedom of interaction, which we deliberately implemented to provide an overview rather than in-depth technical details. Future work might reveal how both an overview and full depth could be combined in NN learning environments. We also learned why recall might be better in the text condition. As participants mentioned, they did not feel they needed to memorize specific terms to be able to use RNNs. This seems natural, as when programming or using RNNs in the wild, remembering specific terms is also not essential as they can be searched for. On the other hand, transfer tends to be much more important when tasked with solving real problems.

For learning material where transfer is important [DU](#), as in recurrent networks, our descriptive results suggest that interactive visualizations such as *exploRNN* might be helpful. Additionally, the lower cognitive load and higher perceived likeability of our interactive environment might result in more learners spending more time with *exploRNN*. Although we extensively evaluated *exploRNN* in this study, it remains to be seen whether our insights are transferable to other learning environments. If so, the development of future explorables could be much better informed, indicating what is important, what could be discarded, and what needs to be improved on. While this study provides first insights into the effectiveness of such educational NN exploration environments, we hope that similar evaluations of other applications can broaden our insights.

10 Conclusions and future work

This paper presents the first interactive learning environment specifically designed for RNNs. We propose a new visualization approach for inspecting RNNs where different levels of granularity are employed. To inform our visualization design, we first introduced educational objectives for this setting. Based on these objectives, we identified design challenges, which we tackle in the proposed learning environment. We hope that this process can be helpful for the development of future interactive learning environments.

Subsequently, we tested the learning outcome, cognitive load, and usability of this learning environment in an empirical study. Our study is the first quantitative evaluation of an interactive NN learning interface and, as such, provides helpful insights and directions for future work. The results of the user study indicate that while the raw learning outcome was not improved compared to conventional methods, *exploRNN* makes learning easier and more fun since cognitive load was significantly reduced by *exploRNN*, while subjective likeability was significantly improved. Based on these findings, we propose to specifically design interactive NN learning environments so that cognitive load is reduced. For broadly accessible education, *exploRNN* can be used in any modern browser at <https://mi-pages.informatik.uni-ulm.de/explornn/>.

As mentioned, more user studies for similar educational explorables could further advance the field and better inform future visualization designs. One such possible research direction is the suspected advantage of the text condition for going back to previously presented information. Here, eye-tracking studies and novel interaction designs might provide new insights. Additionally, it would be interesting to investigate whether such systems indeed lead to more voluntary learning and how that affects learning outcome.

Acknowledgements This work was funded by the Carl-Zeiss Scholarship for Ph.D. students. The datasets generated during and analyzed during the presented study are available from the corresponding author on reasonable request.

Funding Open Access funding enabled and organized by Projekt DEAL.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
2. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
3. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with con-

- volutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9 (2015)
4. Kahng, M., Thorat, N., Chau, D.H.P., Viégas, F.B., Wattenberg, M.: Gan lab: understanding complex deep generative models using interactive visual experimentation. *IEEE Trans. Vis. Comput. Graph.* **25**(1), 1–11 (2018)
 5. Norton, A.P., Qi, Y.: Adversarial-playground: A visualization suite showing how adversarial examples fool deep learning. In: 2017 IEEE Symposium on Visualization for Cyber Security (VizSec), pp. 1–4. IEEE (2017)
 6. Karpathy, A.: ConvnetJS mnist demo. <https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html> (2020)
 7. Smilkov, D., Carter, S., Sculley, D., Viégas, F. B., Wattenberg, M.: Direct-manipulation visualization of deep networks. *arXiv:1708.03788* (2017)
 8. Graves, A., Mohamed, A.-R., Hinton, G.: Speech recognition with deep recurrent neural networks. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 6645–6649. IEEE (2013)
 9. Sak, H., Senior, A.W., Beaufays, F.: Long short-term memory recurrent neural network architectures for large scale acoustic modeling (2014)
 10. Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., Schmidhuber, J.: A novel connectionist system for unconstrained handwriting recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* **31**(5), 855–868 (2008)
 11. Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K. et al.: Google’s neural machine translation system: bridging the gap between human and machine translation. *arXiv:1609.08144* (2016)
 12. Sweller, J.: Cognitive load theory. In: Psychology of Learning and Motivation, vol. 55, pp. 37–76. Elsevier (2011)
 13. Hochreiter, S., Schmidhuber, J.: Long short-term memory. *Neural Comput.* **9**(8), 1735–1780 (1997)
 14. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* **5**(2), 157–166 (1994)
 15. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: International Conference on Machine Learning, pp. 1310–1318 (2013)
 16. Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1724–1734 (2014)
 17. Weiss, G., Goldberg, Y., Yahav, E.: On the practical computational power of finite precision RNNs for language recognition. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pp. 740–745 (2018)
 18. Britz, D., Goldie, A., Luong, M.-T., Le, Q.: Massive exploration of neural machine translation architectures. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pp. 1442–1451 (2017)
 19. Hundhausen, C.D., Brown, J.L.: What you see is what you code: a “live” algorithm development and visualization environment for novice learners. *J. Vis. Lang. Comput.* **18**(1), 22–47 (2007)
 20. Hundhausen, C.D., Douglas, S.A., Stasko, J.T.: A meta-study of algorithm visualization effectiveness. *J. Vis. Lang. Comput.* **13**(3), 259–290 (2002)
 21. Schweitzer, D., Brown, W.: Interactive visualization for the active learning classroom. In: Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education, pp. 208–212 (2007)
 22. Guo, P.J.: Online python tutor: embeddable web-based program visualization for CS education. In: Proceeding of the 44th ACM Technical Symposium on Computer Science Education, pp. 579–584 (2013)
 23. Guo, P.J., White, J., Zanelatto, R.: “Codechella: multi-user program visualizations for real-time tutoring and collaborative learning. In: 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 79–87. IEEE (2015)
 24. Drosos, I., Barik, T., Guo, P. J., DeLine, R., Gulwani, S.: Wrex: A unified programming-by-example interaction for synthesizing readable code for data scientists. In: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, pp. 1–12 (2020)
 25. Ynnerman, A., Löwgren, J., Tibell, L.: Exploramation: a new science communication paradigm. *IEEE Comput. Graph. Appl.* **38**(3), 13–20 (2018)
 26. Harley, A.W.: An interactive node-link visualization of convolutional neural networks. In: International Symposium on Visual Computing, pp. 867–877. Springer (2015)
 27. Chung, S., Suh, S., Park, C., Kang, K., Choo, J., Kwon, B.C.: Revacnn: real-time visual analytics for convolutional neural network. In: KDD 16 Workshop on Interactive Data Exploration and Analytics (2016)
 28. Bloom, B.S., et al.: Taxonomy of educational objectives. vol. 1: Cognitive domain, vol. 20, p. 24. McKay, New York (1956)
 29. Karpathy, A., Johnson, J., Fei-Fei, L.: Visualizing and understanding recurrent networks. *arXiv:1506.02078* (2015)
 30. Strobelt, H., Gehrmann, S., Pfister, H., Rush, A.M.: Lstmvis: a tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE Trans. Visual Comput. Graph.* **24**(1), 667–676 (2017)
 31. Strobelt, H., Gehrmann, S., Behrisch, M., Perer, A., Pfister, H., Rush, A.M.: Seq2seq-vis: a visual debugging tool for sequence-to-sequence models. *IEEE Trans. Visual Comput. Graph.* **25**(1), 353–363 (2018)
 32. Ming, Y., Cao, S., Zhang, R., Li, Z., Chen, Y., Song, Y., Qu, H.: “Understanding hidden memories of recurrent neural networks,” in *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE, pp. 13–24 (2017)
 33. Cashman, D., Patterson, G., Mosca, A., Watts, N., Robinson, S., Chang, R.: Rnrbow: visualizing learning via backpropagation gradients in rnns. *IEEE Comput. Graph. Appl.* **38**(6), 39–50 (2018)
 34. Shen, Q., Wu, Y., Jiang, Y., Zeng, W., LAU, A. K., Vianova, A., Qu, H.: Visual interpretation of recurrent neural network on multi-dimensional time-series forecast. *IEEE Transactions on Visualization and Computer Graphics* (2020)
 35. Garcia, R., Weiskopf, D.: Inner-process visualization of hidden states in recurrent neural networks. In: Proceedings of the 13th International Symposium on Visual Information Communication and Interaction, pp. 1–5 (2020)
 36. Soltau, H., Liao, H., Sak, H.: Neural speech recognizer: acoustic-to-word LSTM model for large vocabulary speech recognition. *arXiv:1610.09975* (2016)
 37. Manh, H., Alaghband, G.: Scene-LSTM: A model for human trajectory prediction. *arXiv:1808.04018* (2018)
 38. Graves, A., Jaitly, N., Mohamed, A.-R.: Hybrid speech recognition with deep bidirectional LSTM. In: 2013 IEEE Workshop on Automatic Speech Recognition and Understanding, pp. 273–278. IEEE (2013)
 39. Park, S.H., Kim, B., Kang, C.M., Chung, C.C., Choi, J.W.: Sequence-to-sequence prediction of vehicle trajectory via LSTM encoder-decoder architecture. In: 2018 IEEE Intelligent Vehicles Symposium (IV), pp. 1672–1678. IEEE (2018)
 40. Gleicher, M.: Explainers: expert explorations with crafted projections. *IEEE Trans. Visual Comput. Graph.* **19**(12), 2042–2051 (2013)
 41. Munzner, T.: *Visualization Analysis and Design*. CRC Press, Boca Raton (2014)

42. Beck, F., Burch, M., Diehl, S., Weiskopf, D.: The state of the art in visualizing dynamic graphs. In: EuroVis (STARs). Citeseer (2014)
43. Shneiderman, B.: The eyes have it: a task by data type taxonomy for information visualizations. In: Proceedings 1996 IEEE Symposium on Visual Languages, pp. 336–343. IEEE (1996)
44. Amadieu, F., Mariné, C., Laimay, C.: The attention-guiding effect and cognitive load in the comprehension of animations. *Comput. Hum. Behav.* **27**(1), 36–40 (2011)
45. De Koning, B.B., Tabbers, H.K., Rikers, R.M., Paas, F.: Attention guidance in learning from a complex animation: Seeing is understanding? *Learn. Instr.* **20**(2), 111–122 (2010)
46. Khomenko, V., Shyshkov, O., Radyvonenko, O., Bokhan, K.: Accelerating recurrent neural network training using sequence bucketing and multi-gpu data parallelization. In: 2016 IEEE First International Conference on Data Stream Mining and Processing (DSMP). IEEE, pp. 100–103 (2016)
47. Zhu, H., Akrout, M., Zheng, B., Pelegris, A., Jayarajan, A., Phanishayee, A., Schroeder, B., Pekhimenko, G.: Benchmarking and analyzing deep neural network training. In: 2018 IEEE International Symposium on Workload Characterization (IISWC). IEEE, pp. 88–100 (2018)
48. Peters, O.: Digital learning environments: new possibilities and opportunities. *Int. Rev. Res. Open Distrib. Learning* **1**(1), 1–19 (2000)
49. Smilkov, D., Thorat, N., Assogba, Y., Yuan, A., Kreeger, N., Yu, P., Zhang, K., Cai, S., Nielsen, E., Soergel, D. et al.: Tensorflow.js: Machine learning for the web and beyond. [arXiv:1901.05350](https://arxiv.org/abs/1901.05350) (2019)
50. Bartram, L.: Perceptual and interpretative properties of motion for information visualization. In: Proceedings of the 1997 Workshop on New Paradigms in Information Visualization and Manipulation, pp. 3–7 (1997)
51. Chevalier, F., Riche, N. H., Plaisant, C., Chalbi, A., Hurter, C.: Animations 25 years later: new roles and opportunities. In: Proceedings of the International Working Conference on Advanced Visual Interfaces, pp. 280–287 (2016)
52. Robertson, G., Fernandez, R., Fisher, D., Lee, B., Stasko, J.: Effectiveness of animation in trend visualization. *IEEE Trans. Visual Comput. Graph.* **14**(6), 1325–1332 (2008)
53. Lee, S., Kim, S.-H., Hung, Y.-H., Lam, H., Kang, Y.-A., Yi, J.S.: How do people make sense of unfamiliar visualizations?: A grounded model of novice's information visualization sensemaking. *IEEE Trans. Visual Comput. Graph.* **22**(1), 499–508 (2015)
54. Berthold, K., Renkl, A.: Instructional aids to support a conceptual understanding of multiple representations. *J. Educ. Psychol.* **101**(1), 70 (2009)
55. Kang, H., Plaisant, C., Shneiderman, B.: New approaches to help users get started with visual interfaces: multi-layered interfaces and integrated initial guidance. In: Proceedings of the 2003 Annual National Conference on Digital Government Research, pp. 1–6 (2003)
56. Brachten, F., Brünker, F., Frick, N.R., Ross, B., Stieglitz, S.: On the ability of virtual agents to decrease cognitive load: an experimental study. *Inf. Syst. e-Bus. Manag.* **18**(2), 187–207 (2020)
57. Ding, W.: Draw convnet. https://github.com/gwding/draw_convnet (2018)
58. Uchida, Y.: Convnet drawer. <https://github.com/yu4u/convnet-drawer> (2019)
59. Bauerle, A., Van Onzenoodt, C., Ropinski, T.: Net2vis-a visual grammar for automatically generating publication-ready CNN architecture visualizations. *IEEE Trans. Visual. Comput. Graph.* (2021)
60. Olah, C.: Understanding LSTM networks. <https://colah.github.io/posts/2015-08-Understanding-LSTMs/> (2015)
61. Elmqvist, N., Fekete, J.-D.: Hierarchical aggregation for information visualization: overview, techniques, and design guidelines. *IEEE Trans. Visual Comput. Graph.* **16**(3), 439–454 (2010)
62. Jenny, B., Kelso, N.V.: Color design for the color vision impaired. *Cartograph. Perspect.* **58**, 61–67 (2007)
63. McCarthy, L.: P5.js. <https://p5js.org/> (2020)
64. Pintrich, P.R. et al.: A manual for the use of the motivated strategies for learning questionnaire (MSLQ) (1991)
65. Cronbach, L.J.: Coefficient alpha and the internal structure of tests. *Psychometrika* **16**(3), 297–334 (1951)
66. Klepsch, M., Schmitz, F., Seufert, T.: Development and validation of two instruments measuring intrinsic, extraneous, and germane cognitive load. *Front. Psychol.* **8**, 1997 (2017)
67. Brooke, J., et al.: Sus-a quick and dirty usability scale. *Usability Eval. Industry* **189**(194), 4–7 (1996)
68. Bangor, A., Kortum, P., Miller, J.: Determining what individual SUS scores mean: adding an adjective rating scale. *J. Usability Stud.* **4**(3), 114–123 (2009)
69. Kolić-Vrhovec, S., Bajšanski, I., Rončević Zubković, B.: The role of reading strategies in scientific text comprehension and academic achievement of university students. *Rev. Psychol.* **18**(2), 81–90 (2011)
70. Kürschner, C., Seufert, T., Hauck, G., Schnotz, W., Eid, M.: Konstruktion visuell-räumlicher repräsentationen beim hör-und leseverstehen. *Z. Psychol./J. Psychol.* **214**(3), 117–132 (2006)
71. Glenberg, A.M., Wilkinson, A.C., Epstein, W.: The illusion of knowing: failure in the self-assessment of comprehension. *Mem. Cognit.* **10**(6), 597–602 (1982)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Alex Bauerle received the master's degree in media informatics from Ulm University in 2017 and is now working at a research associate at the Visual Computing Group at Ulm University. His current research interests are the visualization of neural networks to generate better understanding around these techniques.



Patrick Albus completed his master's degree in psychology at Ulm University with a focus on Work and Organizational Psychology and Clinical Psychology. Since September 2018, Patrick Albus has been working as a research assistant at the Institute of Psychology and Education, Department Learning and Instruction, conducting research in the field of instructional design and virtual reality.



Raphael Störk studied media informatics at the Ulm University and received his bachelor's degree in 2020. His main interests involve the development of learning applications and the concepts of neural networks and other forms of artificial intelligence.



Tina Seufert is Professor and head of the Department for Learning and Instruction since 2008. Her research interests deal with theoretical and practical questions on how to foster learning processes by instructional means. These include the areas of multiple representations, virtual reality, aptitude-treatment interactions and cognitive load. She also develops and tests pedagogical concepts for various learning scenarios such as e-learning, HCI, and continuing education.



Timo Ropinski is a professor at Ulm University, where he is heading the Visual Computing Group. Before moving to Ulm, he was Professor in Interactive Visualization at Linköping University in Sweden, where he was heading the Scientific Visualization Group. He has received his PhD in computer science in 2004 from the University of Münster, where he has also completed his Habilitation in 2009.

SYMPHONY: COMPOSING INTERACTIVE INTERFACES FOR MACHINE LEARNING

Alex Bäuerle¹, Ángel Alexander Cabrera¹, Fred Hohman, Megan Maher, David Koski, Xavier Suau, et al. "Symphony: Composing Interactive Interfaces for Machine Learning." In: *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (2022), pp. 1–14

This work is published under the terms of the Creative Commons Attribution 4.0 License (CC BY 4.0), <https://creativecommons.org/licenses/by/4.0/>

Symphony: Composing Interactive Interfaces for Machine Learning

Alex Bäuerle^{*}[†]

Ulm University
Ulm, Germany

alex.baerle@uni-ulm.de

Megan Maher

Apple
Cupertino, CA, USA
megan_maher@apple.com

Angel Alexander Cabrera^{*}[†]

Carnegie Mellon University
Pittsburgh, PA, USA
cabrera@cmu.edu

Fred Hohman

Apple
Seattle, WA, USA
fredhohman@apple.com

Titus Barik

Apple
Seattle, WA, USA
tbarik@apple.com

David Koski

Apple
Cupertino, CA, USA
dkoski@apple.com

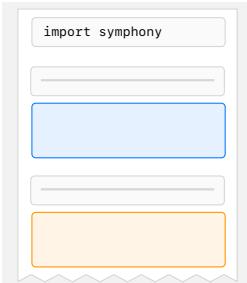
Xavier Suau

Apple
Barcelona, Spain
xsuaucuadros@apple.com

Dominik Moritz

Apple
Pittsburgh, PA, USA
domoritz@apple.com

Computational Notebooks



Symphony

Interactive Components for Machine Learning



Code Environments

Web-based UIs

Reports and Dashboards

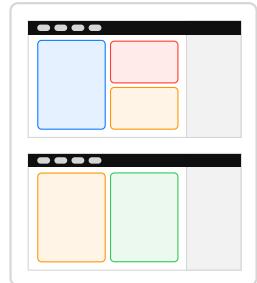


Figure 1: *Symphony* applies techniques from machine learning (ML) documentation, data visualization, and interactive programming to create ML interfaces with interactive, task-specific components. Diverse ML practitioners can explore their data and analyze their models where they work, both in computational notebooks and in web-based dashboards.

ABSTRACT

Interfaces for machine learning (ML), information and visualizations about models or data, can help practitioners build robust and responsible ML systems. Despite their benefits, recent studies of ML teams and our interviews with practitioners (n=9) showed that ML interfaces have limited adoption in practice. While existing ML interfaces are effective for specific tasks, they are not designed to be reused, explored, and shared by multiple stakeholders in cross-functional teams. To enable analysis and communication between

different ML practitioners, we designed and implemented *Symphony*, a framework for composing interactive ML interfaces with task-specific, data-driven components that can be used across platforms such as computational notebooks and web dashboards. We developed *Symphony* through participatory design sessions with 10 teams (n=31), and discuss our findings from deploying *Symphony* to 3 production ML projects at Apple. *Symphony* helped ML practitioners discover previously unknown issues like data duplicates and blind spots in models while enabling them to share insights with other stakeholders.

CCS CONCEPTS

- Human-centered computing → Interactive systems and tools; Visual analytics;
- Computing methodologies → Machine learning; Artificial intelligence.

KEYWORDS

Machine learning, AI, visualization, documentation, interactive programming, computational notebooks

^{*}Both authors contributed equally to this research.

[†]Work done at Apple.



This work is licensed under a Creative Commons Attribution International 4.0 License.

ACM Reference Format:

Alex Bäuerle, Ángel Alexander Cabrera, Fred Hohman, Megan Maher, David Koski, Xavier Suau, Titus Barik, and Dominik Moritz. 2022. Symphony: Composing Interactive Interfaces for Machine Learning. In *CHI Conference on Human Factors in Computing Systems (CHI '22), April 29-May 5, 2022, New Orleans, LA, USA*. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3491102.3502102>

1 INTRODUCTION

Successfully deploying machine learning systems in production is a complex, collaborative process that involves a wide range of ML practitioners, from data scientists and engineers to domain experts and product managers. A substantial amount of research has gone into creating ML interfaces for analyzing and sharing insights about ML systems that practitioners can use to better understand and improve deployed ML products. We describe *machine learning interfaces* as static or interactive artifacts, visualizations, and information that communicate details about ML data and models. ML interfaces include documentation methods (e.g., Model Cards [46], Datasheets [17]), visualization dashboards (e.g., What-if Tool [62], ActiVis [33], among many others [21]), and interactive programming widgets (e.g., ipywidgets [32], Streamlit [31]) that give practitioners insights into what their datasets contain and how their models behave. Despite the benefits and breadth of ML interfaces, recent studies have found that they are not as widely used and shared in practice as expected [38, 68]. This underuse can lead to missed data errors and model failures, a lack of shared team understanding of model behavior, and, ultimately, deployed ML systems that may be biased [8] or unsafe [47].

To understand why ML interfaces are not used more frequently, we interviewed 9 ML practitioners at Apple about their current machine learning practice and workflows. We found that while ML practitioners want to use them, current interfaces have limitations that make them either insufficient or too time consuming to use. One category of ML interfaces are *ML documentation* methods, such as Model Cards [46] and Datasheets [17], which describe the details and records the provenance of an ML system's data and model. Documentation methods often lack the interactive tools and visualizations necessary for specific analyses and have to be manually authored and updated separately from where ML development happens. Another category of interfaces, *visualization dashboards*, consist of multiple coordinated views tailored to specific domains and tasks. ML practitioners must learn a new platform and wrangle their data into the right format in order to use these bespoke systems, which also require significant work to reuse for different tasks. Finally, *interactive programming widgets* can render web-based ML visualizations directly in code environments. However, widgets typically cannot be used outside of the platform in which they were created and often lack complex visualizations required by modern ML models and unstructured data—non-tabular data types such as images, videos, audio, point-clouds, sensor data, etc. Overall, we found that while current ML interfaces work well for specific tasks and platforms, they are not designed to be reused, explored, and shared by diverse stakeholders in cross-functional ML teams.

Our formative research showed that ML work requires bespoke visualizations for complex models and data types which work across the different platforms ML practitioners use. To address these needs,

we combined the affordances of existing ML interfaces to design and implement *Symphony*, a framework for creating and composing interactive ML interfaces with task-specific, data-driven visualization components. *Symphony* supports two popular platforms used by ML practitioners, code environments such as Jupyter notebooks and no-code environments such as web-based UIs (Figure 1). *Symphony* components are JavaScript modules that use custom code or existing libraries to create task-specific visualizations of structured and unstructured data. Each component is also fully interactive: users can filter, group, or select instances either through a UI toolbar or code. These interactions are reactively synchronized across *Symphony* components, enabling linked visualizations. *Symphony*'s cross-platform availability enables ML practitioners to use the same components for both exploring and sharing insights about their ML systems (Figure 2).

We worked with ML teams at Apple to both design *Symphony* and apply it to deployed ML projects. To collect the diverse requirements and use cases for ML interfaces, we conducted participatory design sessions with 10 ML teams with a total of 31 ML practitioners. Informed by these sessions, we implemented a set of 11 components supporting a range of different models and data types. We then worked with 3 teams from the design sessions to deploy *Symphony* in their machine learning workflows and ran a think-aloud study with them to qualitatively evaluate *Symphony*.

Teams using *Symphony* with their real-world data and models found surprising insights which they had not previously known, such as duplicate instances, labeling errors, and model blind spots. Participants also described a variety of use cases for *Symphony*, from creating automated dataset reports to analyzing model performance in computational notebooks. Moreover, participants that did not previously share their analyses also showed interest in using *Symphony* in their teams to better communicate the state of their ML system with other stakeholders.

The main contribution of this work is *Symphony*, a framework for composing interactive ML interfaces with task-specific, data-driven visualization components. To design *Symphony*, we conducted formative interviews, participatory design sessions, and case studies on deployed ML workflows with a total of 39 ML practitioners across 15 teams. *Symphony* enabled ML practitioners to discover significant issues like dataset duplicates and model blind spots, and encouraged them to share their insights with other stakeholders. *Symphony* combines the following principles to improve upon existing ML interfaces:

- **Data-driven ML interfaces** derived from and updated with ML data and models.
- **Task-specific visualizations** for unstructured data and modern machine learning models.
- **Interactive exploration tools** for exploring different dimensions of an ML system.
- **Reusable components** that can be used, composed, and shared across different platforms.

2 BACKGROUND AND RELATED WORK

Symphony bridges three areas of related work: ML documentation methods, data visualization dashboards, and interactive programming environments. First, the *Symphony* framework can be used

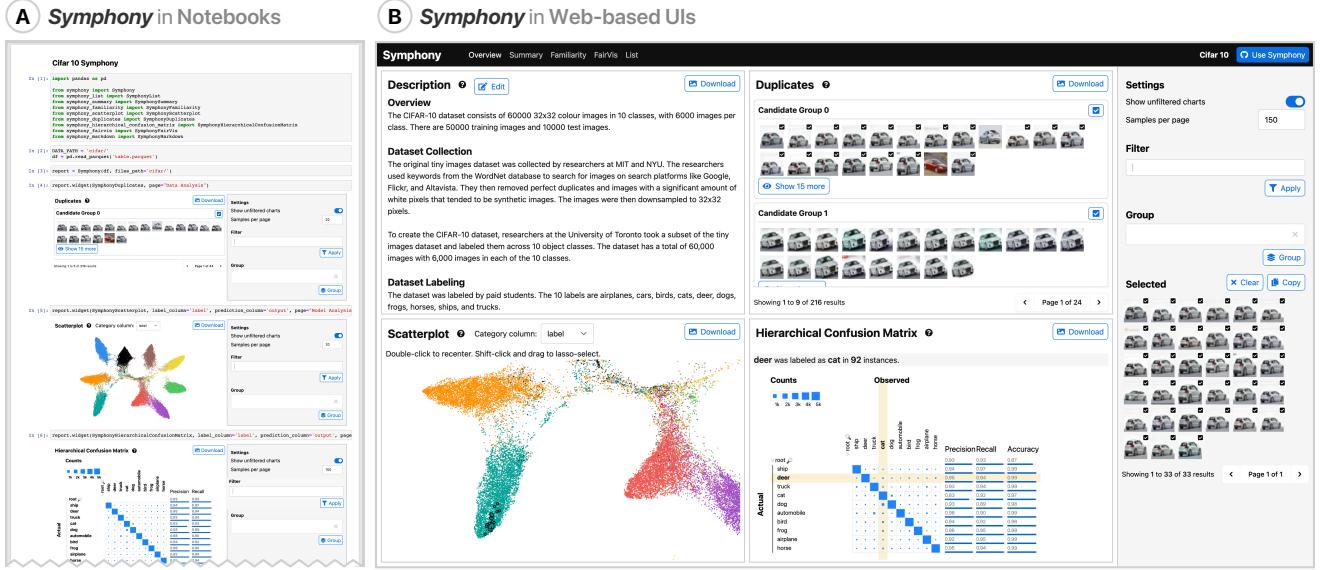


Figure 2: A demonstration of *Symphony* running in both (A) a computational notebook and (B) a web-based UI with the same visualization components and code. In a computational notebook, an ML practitioner passes their data and model outputs directly from Python variables like Pandas Data Frames [45] to *Symphony* components. The ML practitioner can then export the components to a self-contained, web-based UI. This example shows *Symphony* loaded with the CIFAR-10 [40] dataset and a trained image classification model. After reading a textual description of the dataset, a user found and selected duplicate car instances which were reactively highlighted in the projection component and the confusion matrix. The user then explored the confusion matrix to determine if the duplicates could be impacting model performance.

to write and share ML documentation. Second, *Symphony* components can show complex visualizations and be composed into visual analytics dashboards to help ML practitioners make sense of ML data and models (Section 2.2). Lastly, *Symphony* components can be used in and exported from interactive programming environments, like computational notebooks, which are often used by ML practitioners (Section 2.3).

2.1 Documenting Data and Models

A variety of documentation methods exist to help ML practitioners track and communicate details about their data and models. Without knowing what a dataset contains or what a model has learned, teams can inadvertently release AI products with issues like safety concerns and biases [13, 15, 53, 54], as seen in numerous deployed systems [8, 19, 55, 63].

Since machine learning models are a direct result of the data they were trained on, it is important to first understand the data behind an ML system. Datasheets for Datasets [17] applies the idea of datasheets in electrical engineering to describe important attributes of a dataset, such as collection methods and intended uses. Similar work has focused on specific types of data, for example, Data Statements [7] are tailored to natural language processing datasets. These guidelines describe *what* should be included in documentation, not *how* an author can create or share the resulting artifact [38]. Additionally, these documents are static L^AT_EX or text documents that are disjoint from the backing data and models and have to be manually updated. Since there is heterogeneity in

what information is important for each dataset, Holland et al. [23] proposed the more general concept of Dataset Nutrition Labels, modular graphs describing different aspects of a dataset. Like *Symphony*, these labels use modular visualizations, however, they focus on simple aggregate visualizations without displaying data samples and do not support platforms where ML practitioners do their work.

A parallel line of research has focused on documenting machine learning models. Model Cards [46] and FactSheets [4] are similar concepts to Datasheets that can include important information and details about machine learning models. These model reports include information ranging from the model type and hyperparameters to aggregate metrics and ethical considerations. Similar to Datasheets, these types of documentation are disjoint from the backing data and do not include interactive visualizations of model details and performance metrics.

2.2 Visualization for Machine Learning

There are a growing number of visualization systems that help ML practitioners make sense of modern ML systems with unstructured datasets and machine learning models [21]. Visualizations can help ML practitioners in tasks such as auditing models for bias [9], understanding the internals of deep learning architectures [22], and guiding automatic model selection [10]. A full review of this literature is out of scope for this work, but we provide a sample of representative systems to highlight the types of visualizations that could be implemented as *Symphony* components.

Data science work often starts with and leads back to understanding the backing data. Modern machine learning models and tasks use *unstructured* data like images and audio that cannot be visualized and explored with tables and histograms. Systems like Know Your Data [28] and Facets [27] are visualization dashboards for exploring unstructured data. Other visual analytics systems process the data further to derive insights like outliers [11], biases in a dataset [61], or mislabeled data instances [66]. With a deeper understanding of their data, ML practitioners can more effectively debug and improve their models.

The models ML practitioners use are often large, complex black-box models like deep learning systems. Visualization systems like Summit [22] and Seq2Seq-Vis [57] can help ML practitioners develop a better mental model of how their machine learning systems work and what they are learning. Another set of systems, including Model Tracker [3], Squares [50], AnchorViz [12], ConfusionFlow [20], What-if Tool [62], and MLCube [34], focus on performance analysis and provide different views of a model's errors. Lastly, there are tools for detecting potential biases [1] or systematic errors [6, 64] in training data. These various of visualizations can be repackaged as *Symphony* components, for example, we implement a version of FairVis [9] as a component for auditing classifiers for bias.

Lastly, there are integrated systems that help ML practitioners both implement and visualize ML models. One of the first systems describing such an integrated system is Gestalt [48], a development environment with visualizations for training and analyzing classification models. A subsequent system focused on interactive machine learning is Marcell [16], which uses composable stages and visualizations to create interactive ML interfaces. In contrast to Gestalt and Marcell, *Symphony* is focused on the analysis stage of ML systems, and includes important features such as cross-platform support, reactivity, and a consistent data API which are not available in Gestalt and Marcell.

ML data and model visualizations are often deployed as visual analytics dashboards that are separate from both interactive programming environments that ML practitioners work with and ML documentation shared with other stakeholders. This separation limits who can use visualizations to understand ML data and models. *Symphony* aims to bridge these worlds by bringing visualizations both into notebooks where data work happens and into the documentation shared with other stakeholders.

2.3 Interactive Programming Environments

ML practitioners often use interactive programming environments for exploring and modeling data since they can interact with and iterate on their ML systems [35]. These environments are most commonly implemented as computational notebooks like Jupyter [37], DataBricks [26], and Observable [29]. While computational notebooks have extensions for creating interactive visualizations, such as the ipywidgets API [32] for Jupyter, they are often underused [5] and hard to share [18, 35].

Several libraries exist for interactively visualizing data in notebooks. Graphing libraries such as Altair [58] and Plotly [30] allow users to create interactive charts but only support a finite set of graphs and require users to manually define what visualizations

they want to use. Lux [41] and B2 [65] lower the cost of using visualizations in notebooks by automatically providing relevant charts for users' data frames. These approaches help analyze tabular data, but they lack the specific visual representations needed for machine learning development.

A separate challenge is sharing visualizations and other notebook outputs outside of the notebook context. Voilà [60] tackles this challenge directly by exporting full Jupyter notebooks to a hosted website. ML practitioners can use Voilà to share notebooks that contain *Symphony* components, but it requires a Python kernel to be running and Voilà does not provide any visualizations itself. Two visualization frameworks similar to *Symphony*, Panel [24] and Plotly Dash [49], use independent components to create visualizations that can be used in both Jupyter notebooks and standalone websites. However, these tools also have limitations for creating complete ML interfaces: Panel visualizations are tied to the Jupyter ecosystem and lack interactivity without a Python backend, while Plotly Dash primarily supports Plotly charts and does not easily extend to custom visualizations. *Symphony* provides components that are fully interactive in both notebooks and web UIs, and support any JavaScript-based visualization. Additionally, *Symphony*'s shared state synchronizes its components, enabling reactive brushing and linking between views.

More recent interactive programming environments have moved away from the notebook paradigm. For example, in the Streamlit [31] platform, users write Python scripts using a library that renders interactive components in a separate website. While Streamlit supports interactive components like Jupyter notebooks, it is primarily an environment focused on designing web applications rather than exploratory data science or ML reporting. Exploratory analysis is still often done in notebooks, and Streamlit requires users to learn a new platform. Other platforms are moving away from programming altogether, such as Glinda [14], a declarative language that lets ML practitioners describe analysis steps in a domain-specific language. Glinda does not define any specific visualizations, but it could be complemented by *Symphony* components. Since *Symphony* components are standalone JavaScript modules, future wrappers could integrate *Symphony* components into data science environments like Streamlit and Glinda.

3 FORMATIVE INTERVIEWS

To understand how ML interfaces are used in practice, we conducted 7 semi-structured interviews with 9 participants at Apple. We recruited participants through internal emails and messaging boards and selected participants across a range of different roles, including engineers, designers, researchers, and testing roles that work on teams to build and deploy ML systems. Each interview was conducted over a video call and lasted about an hour. First, we asked participants about how they currently create and use different ML interfaces like documentation, visualization dashboards, and widgets. We then asked them what the main limitations and pain points are in current tools and what types of improvements they would find helpful. From these need-finding interviews we identified the following themes.

Use cases for ML interfaces. All participants agreed that creating and sharing ML interfaces can help them build more robust and

capable ML products. Participants described use cases of ML interfaces in myriad tasks, such as “flagging failures for review,” (P2) “detecting systematic failures,” (P4) and “fairness and bias education.” (P1) Participants also mentioned stages across the entire ML process in which ML interfaces can be useful, from “dataset curation and sharing” (P5) to analysis “after an ML model has been trained,” (P7) or “in all stages” (P1). Consequently, since different stakeholders involved in an ML product need specific views of the data and models, ML interfaces must be flexible enough to support analysis across numerous tasks and domains.

Ad-hoc tools and analyses. While all participants detailed clear use cases for ML interfaces, they also mentioned limitations preventing them from using existing tools or sharing insights. One participant bluntly stated “right now, we basically have no tools” (P3) for analyzing ML systems. Instead, participants rely on ad-hoc, hand-crafted visualizations for their specific analyses. For example, one of our participants said their process for looking at instances is to “manually examine icons in a file explorer.” (P9) Another participant “looks at handcrafted summaries of select data subsets” (P4) to do model analysis. Larger teams with more resources may have bespoke tools, such as one participant that “use[s] a team-internal tool to analyze data” (P6). Overall, a lack of adequate tooling leads to ML practitioners using one-off, manual tools or ML teams investing in their own, custom visualization systems.

Limitations of existing ML interfaces. Participants detailed a variety of technical roadblocks and time-consuming processes preventing them from using existing ML interfaces. Many tools require users to wrangle and export their data into a specific format before loading it into a custom system or dashboard. However, as one participant stated, “we do not have a lot of time for creating such visualizations:” (P1) ML practitioners simply do not have the bandwidth to do the setup and data wrangling work necessary to use separate systems. ML practitioners’ main priority is working on data and models, and “if it takes longer than 5-10 minutes, I am not going to [use an ML interface] immediately” (P6).

Five participants mentioned explicitly that they do not use ML interfaces because they are not available in the environments where they work, and that “people would want to use easier tools.” (P3) For example, “many data scientists want to explore their data in notebooks” (P2) without having to open a separate system. Additionally, since data and models update frequently, one participant wanted to “start a job with checkboxes and buttons” (P6) and produce a self-updating web UI that they would not have to manually author.

Lastly, the teams we talked to work with myriad data types, such as video, 3D point cloud, tabular, image, and audio data, and desired bespoke visualizations supporting their analysis needs. One participant mentioned running and visualizing specific data analyses, and “would want to specify algorithms because our problems are very specialized.” (P8) However, current data science tools often only provide visualizations for a limited set of data types and models.

Lack of communication between stakeholders. As a consequence of limited, isolated interfaces, participants described various challenges for communicating and sharing insights. Since different stakeholders prefer different environments, such as code-based notebooks or standalone dashboards, it can be challenging to share

insights with others. In addition to sharable interfaces, participants also wanted cross-platform support for themselves, as one participant put it, “I would like both an environment for experimentation and always there reliable visualizations.” (P2)

It can also be difficult to transfer visualizations and findings between platforms that different stakeholders work with. One participant lamented that “I am often not invited to the table until things go wrong,” (P4) and in some teams “designers often times don’t have access to data and model results.” (P3) In turn, decisions about ML systems are made without all team members having a shared understanding of the current state and limitations of the project. Despite these current limitations, participants thought that “fostering a culture of sharing insights would be great.” (P3)

4 DESIGN GOALS

Based on the challenges we identified in the formative interviews, we found that a successful framework for ML interfaces must fulfill the following:

Enable data-driven ML interfaces. ML interfaces are often disconnected from an ML system’s backing data and model outputs [17, 46]. ML practitioners should be able to create visualizations that are up-to-date and reflect an ML systems’ current state.

Support task-specific visualizations. Specialized visualizations are often needed to make sense of the unstructured data and deep learning models increasingly used in machine learning [51, 59]. ML interfaces should support these task-specific visualization needs.

Provide interactive exploration tools. Static ML interfaces only show a fixed subset of the possible analyses stakeholders may need [38]. Interactive visualizations let different stakeholders discover and validate the patterns most relevant to their goals.

Make components reusable. Different stakeholders explore ML systems in different environments, such as computational notebooks and web-based UIs. ML interfaces should be available across environments and reusable for different domains and tasks.

5 SYMPHONY: A FRAMEWORK FOR COMPOSING INTERACTIVE INTERFACES FOR MACHINE LEARNING

Based on these design goals we built *Symphony*, a framework for composing ML interfaces from interactive visualization components. ML practitioners can explore their data and models using *Symphony* components in a computational notebook and then combine and transform them into web-based UIs. *Symphony* consists of three primary features: modular components (Section 5.1), environment wrappers (Section 5.2), and interaction tools (Section 5.3). In the following, we describe the specific design and implementation choices we made to support these goals.

5.1 Modular Components

The building blocks of *Symphony* are independent, modular components designed for task-specific visualizations (Figure 3, right). A *Symphony* component is a JavaScript module that renders a web-based visualization. We use the Svelte¹ web framework as the base

¹<https://svelte.dev>

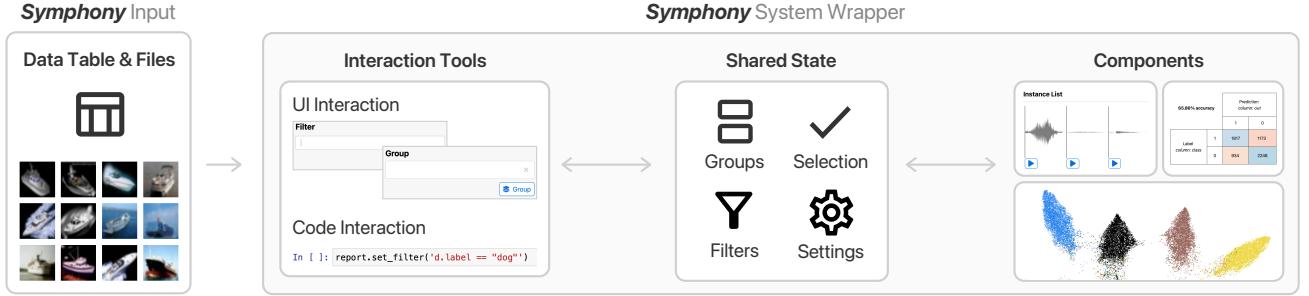


Figure 3: The technical overview of the *Symphony* framework. A dataset and files are passed into the *Symphony* wrapper for a particular platform. The wrapper holds the shared state which is reactively updated and modified either by standardized interaction tools or components themselves.

of *Symphony* components, but visualizations can be written using any JavaScript code or library. JavaScript has a rich ecosystem of libraries and APIs for creating interactive visualizations, like D3 and ThreeJS, which can be used to create *Symphony* components. This flexibility is important for visualizing unstructured ML datasets, something that is not supported by common charting libraries like Matplotlib [25] or Altair [58].

Each *Symphony* component is passed three parameters: a metadata table, derived state variables like grouped tables, and references to raw data instances like images. The metadata table contains a row for each instance from which a set of *state* variables, such as filtered and grouped tables are derived (state variables are described in detail in Section 5.3). Components are also passed a URL from which to fetch raw data samples such as images or audio files. *Symphony* controls these three parameters, synchronizing and reactively updating them across components.

New components can be created using a cookiecutter template that generates all the boilerplate code needed to integrate components with *Symphony*. In the cookiecutter code, a component developer modifies the front-end JavaScript to create their custom interactive visualization. They can make use of the parameters provided by *Symphony* to base their visualization on the data provided by a ML practitioner. In the following Subsection we show how these modular, reactive components can then be composed by a *Symphony* wrapper to be used across different platforms.

5.2 Platform Wrappers

The primary goal of using self-contained components is to compose and share them as flexible interfaces across different platforms. This is done using *Symphony*'s next main feature, wrappers, which connect components with a particular backing platform. These wrappers have two primary functions - first, passing data from a platform to *Symphony* in the correct format, and second, rendering *Symphony* components in the platform's UI. To support both exploring and sharing ML interfaces, we implemented wrappers for the two platforms most requested in our formative study, Jupyter notebooks and web UIs. These platforms are also representative of the two environments we found to be most used by ML practitioners: programming environments for exploratory analysis and web-based UI interfaces for sharing insights.

The Python wrapper bundles *Symphony* components as packages which can be published to a package index like PyPI for use in notebooks and Python scripts. To make *Symphony* interfaces available in Jupyter notebooks, *Symphony*'s Python wrapper also makes each component an ipywidget [32]. The ipywidgets API renders web-based widgets in the Jupyter notebook UI and synchronizes its variables with the Python kernel. Data tables like Pandas DataFrames or Apache Arrow tables, along with an endpoint for raw instance files, can be passed to *Symphony*'s Python wrapper to connect components to the data.

```
# Using Symphony in Python (e.g. a notebook)
import pandas as pd
from symphony import Symphony

# Import three Symphony components
from symphony_summary import SymphonySummary
from symphony_list import SymphonyList
from symphony_duplicates import SymphonyDuplicates

# Load data
IMAGE_PATH = 'images/cifar/'
metadata_table = pd.read_parquet('table.parquet')

# Initialize Symphony
symph = Symphony(metadata_table, files_path=IMAGE_PATH)

# Use Symphony components
symph.widget(SymphonySummary)
symph.widget(SymphonyList)
symph.widget(SymphonyDuplicates)
```

The second wrapper we implemented is for standalone, web-based dashboards. To support this, each *Symphony* component overrides an export function which is used by *Symphony* to transform selected visualization components from Python code into web-based UIs. Components can be configured before export to be placed on different subpages and arranged within these pages to fit particular use cases, as shown in Figure 5. These dashboards can be authored in programming environments and then exported as a statically hosted websites. The wrapper for web-based UIs provides an HTML file which imports components as independent JavaScript (ES6) modules. Since *Symphony* components are compiled to pure JavaScript files, the standalone dashboard does not need a dedicated backend and can be hosted on a static file server.

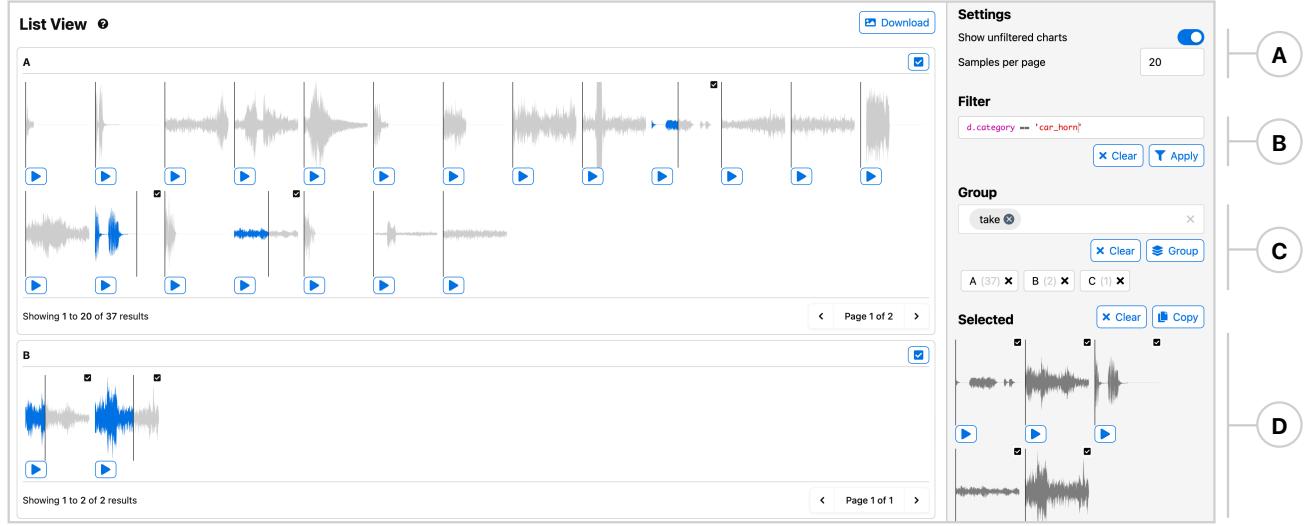


Figure 4: A list component looking at audio samples from the ESC-50 environmental noise classification dataset. The toolbar on the right has UI elements for the different interaction tools available in *Symphony*. The user (A) has increased the number of instances shown per page, and then (B) filtered to see only car horn noises. They then (C) grouped by the “take” feature, and (D) selected a set of interesting instances. In a notebook a user can also set these parameters from code.

```
# Compose a Symphony web dashboard
sympf.widget(SymphonySummary, page="Overview")
sympf.widget(SymphonyList, page="Overview", width="M")
sympf.widget(SymphonyDuplicates, page="Data Analysis",
            width="M", height="L")

# Export Symphony as a standalone web dashboard
sympf.export('./standalone', name="Cifar 10")

# Run the Symphony dashboard in a web browser
sympf.serve_static('./standalone')
```

New wrappers can be written to include *Symphony* components in other platforms. For example, we began to explore how we can enable users without programming experience to create *Symphony* UIs using a drag-and-drop dashboard builder. We have also experimented with integrating *Symphony* components in other interactive programming environments like Streamlit [31] or Glinda [14].

5.3 Interactive Exploration Tools

The final key feature of *Symphony* is a set of tools for interacting with and exploring data. Each component has the same interaction tools, and changes are reactively synchronized between components both in Jupyter notebooks and in web-based UIs. For the web-based UI, state changes are also saved in the URL, allowing stakeholders to share specific findings. *Symphony*'s interaction tools were derived both from common interactions described by participants in the formative study and findings from visualization research [2, 67]. We included a subset of tools that we found to be important for the specific components we implemented. These tools include data filtering, grouping, and instance selection. Additional interaction tools can be added to *Symphony* by updating the main *Symphony* package and platform wrappers with the new tool, which is then available on different platforms and synchronized

across components. New interaction tools can then be accessed and modified by individual *Symphony* components.

Users have three ways of using *Symphony*'s interaction tools: through a UI toolbar, *Symphony* components themselves, or code. The UI toolbar (Figure 4, right) is available both in interactive programming environments (Figure 2, left) and the web-based dashboards (Figure 2, right). We implemented this toolbar as another *Symphony* component, which is shown alongside each component in Jupyter notebooks for convenient access, and displayed as a consistent sidebar for the web-based dashboard. Apart from the UI toolbar, components not only have direct access to the global *Symphony* state but can also modify it based on user interaction. For example, individual data samples can be selected from whichever component they are viewed in. Thus, component developers can add custom controls to manipulate *Symphony*'s state. Lastly, ML practitioners may want to make more complex data transformations that cannot be mapped to UI components. For such use cases, *Symphony*'s state can also be directly manipulated within Python. Whether in a notebook or Python script, users can set and retrieve any of the state variables. In Jupyter notebooks, this allows for fluid interactions between UI and code in the style of Kery et al. [36]. Additionally, *Symphony*'s state can be extracted from the web-based UI and loaded into Python-based notebooks, making findings from shared *Symphony* dashboards available to the ML practitioners in code-based environments.

```
# Get selected items in GUI as a Python list
selected_items = sympf.get_selected()

# Set selected items in GUI from a Python list
sympf.set_selected(python_list)
```

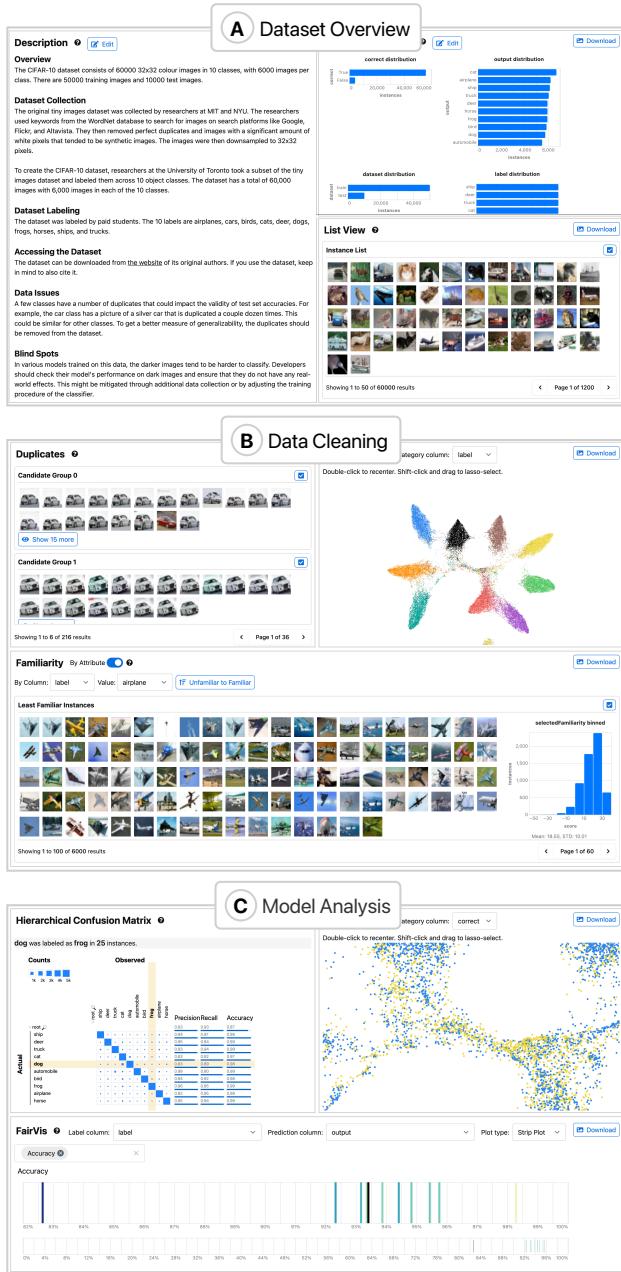


Figure 5: Symphony supports a diverse range of ML tasks. Here we show examples of three distinct dashboards: (A) A dataset overview with a textual description of the data's origin, distribution plots, and example data instances. ML practitioner can use this report to understand what a dataset contains and what tasks they can use it for. (B) A data validation dashboard to help ML practitioners track issues during data collection, such as duplicate or out-of-distribution instances. (C) A model analysis dashboard for exploring the performance of an ML system. Users can find groups of incorrectly classified instances in the embedding and drill down into fairness metrics with respect to different data subgroups.

6 PARTICIPATORY DESIGN SESSIONS

With the initial *Symphony* framework, we conducted a series of participatory design sessions to understand the specific needs of ML teams and design and develop an initial set of *Symphony* components. We conducted 10 sessions where each session had between 1 and 7 people, with a total of 31 people across all sessions. We recruited and contacted teams via internal mailing lists, and the sessions lasted between 30 minutes and an hour. The first half of each session consisted of a demonstration of a *Symphony* prototype based on a mock dataset. In the second half of each session, we asked participants to reflect on and describe their own work and asked them about what additional features would be necessary to integrate *Symphony* into their workflows.

6.1 Expanding *Symphony*'s Technical Capabilities

From these participatory design sessions, we extracted a set of additional needs and wants for *Symphony*. Rather than the high-level goals presented in Section 4, the findings from the participatory design sessions are more technical and tied to the implementation of *Symphony*.

While displaying images directly in computational notebook components was greatly appreciated by the participants working in computer vision, the teams working in different domains expressed interest in previewing and visualizing other data types. To demonstrate *Symphony*'s ability to support other unstructured data types, we made the display of data sample modular and added audio data as an additional supported data type. To visualize other types of data, a developer just has to implement a rendering function for the new data which all components can use.

Some teams work with large models trained on big data, which originally exceeded *Symphony*'s ability to scale and led to long load times. In response, we implemented pagination for all the components that display raw data. Depending on the data type, the number of samples per page can be adjusted, allowing *Symphony* to scale to millions of data samples. For even larger datasets, where a ML practitioner wants to load and visualize hundreds of millions of data points, the browser memory becomes a limiting factor for holding the backing metadata table. For these truly large datasets, we suggest users select representative subsets for detailed analysis; however, scaling beyond millions of data instances is described in Figure 8.

Interactive exploration is a powerful analysis technique when developing ML systems. However, for ML projects that contain many datasets, compounded when data or models are rapidly changing, participants expressed interest in automatically generating shareable dashboards and reports to support streaming data and automatic model retraining. Apart from providing *Symphony* as an authoring tool in computational notebooks, ML practitioners can also write Python scripts that consume ML data and model outputs, assemble a selection of components, and create and export a standalone *Symphony* web UI.

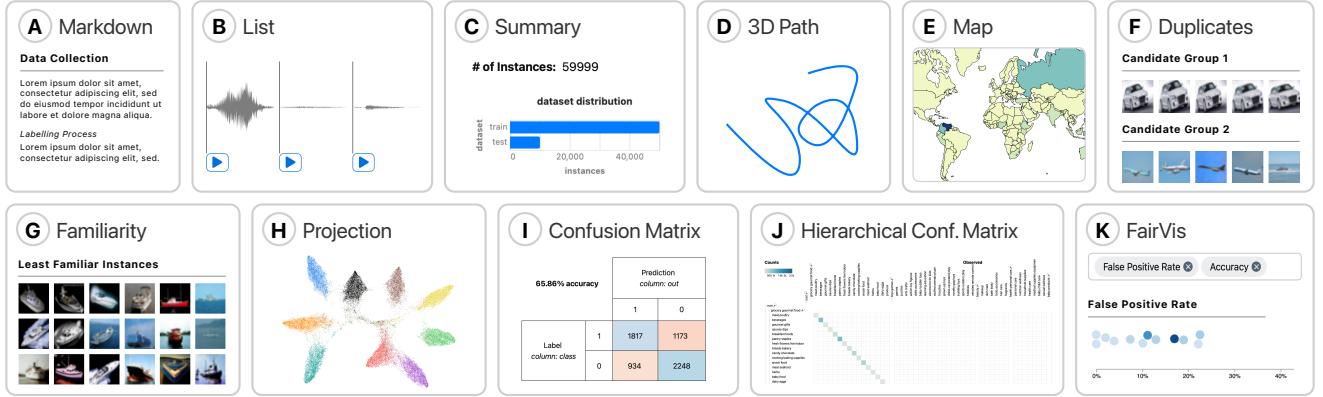


Figure 6: The *Symphony* components we implemented as a result of the participatory design sessions. (A) Markdown text for data and model details. (B) A paginated list of instances. (C) Distribution charts for metadata columns. (D) 3D path visualizations for sensor and inertial measurement unit (IMU) data. (E) Map visualizations for geographic data. (F) Potential duplicate instances. (G) Familiar and unfamiliar instances in a dataset. (H) 2D projection for model embeddings. (I) Binary confusion matrix. (J) Confusion matrix for hierarchical classification models. (K) Fairness analyses of intersectional subgroups.

6.2 Implemented *Symphony* Components for Data and Model Analysis

Informed by the feedback and needs expressed in the participatory design sessions, we implemented an initial set of 11 components shown in Figure 6. These initial components cover various data and model analysis tasks, from finding potential duplicates in a dataset to auditing models for biases. We created all components using the component cookiecutter template described in Figure 5.1.

The first set of components created cover *overview descriptions and summaries of an ML dataset*. The markdown component (A) lets *Symphony* replicate existing documentation methods like Datasheets and Model Cards by writing rich text content. Users can follow existing guidelines to document essential information about a dataset or model often overlooked or not described. The list component (B) shows a paginated list of data instances, with support for a variety of data types like images and audio. Multiple ML practitioners requested this feature, since they currently use file explorers outside of a notebook or one-off functions to look at individual instances. Distribution charts and counts in the summary component (C) provide a high-level overview of data and can help detect potential biases or skews in a dataset. Lastly, we developed two additional components, a 3D path component (D) and map component (E), for exploring specific data types like health sensor data and geographic distributions.

We also implemented a set of components for more *complex analysis of unstructured datasets* that were important to multiple teams. We first compute a model embedding from a deep learning model on the provided data instances, from which different metrics are calculated. For the first of these components we use a nearest neighbors algorithm based on cosine distance in embedding space to find potential groups of duplicate instances (F), which could impact training performance or the validity of test set accuracy. In the next component, we fit a Mixture of Gaussians model on the embeddings to calculate a familiarity score for each data point. We find the most and least familiar instances in a dataset (G) by sorting

by familiarity score. Instances with low familiarity scores can be outliers or mislabeled instances, while high familiarity instances can show over-represented types of data. Finally, there is a 2D projection embedding (H) that shows a dimensionality reduced representation of the embeddings. The embedding can be used to find various interesting data and model patterns and is especially useful when used to explore insights found in other components.

The last set of components we implemented focus on *analyzing and debugging ML models*. The classic confusion matrix component (I) is important for initial debugging of classification models. Other classification tasks that use data with hierarchical or multi-label data can be explored using a hierarchical confusion matrix component (J). We primarily implemented this component for a team in the participatory design sessions that was working on hierarchical classification models. Lastly, we built a set of visualizations for analyzing model performance across intersectional subgroups (G) based on a system by Cabrera et al. [9]. The visualization can help users audit their models for biases, something which multiple product teams were interested in.

We used three different methods for implementing the above *Symphony* components. *Symphony* components are Svelte and JavaScript (JS) files, so authors can create new visualizations with their preferred front-end libraries. For components without existing libraries, we used JavaScript in combination with visualization packages such as Vega and D3. *Symphony* can also use off-the-shelf JS libraries, for example, we used REGL Scatterplot [42], a WebGL library, to create the projection component. Lastly, since *Symphony* components are made with Svelte, we can also directly use Svelte components, which is what we did with the Hierarchical Confusion Matrix. These different strategies for creating components provide the flexibility to implement custom visualizations while also allowing developers to use off-the-shelf libraries and visualizations.

7 CASE STUDIES ON DEPLOYED ML SYSTEMS

Lastly, we evaluated *Symphony* with ML practitioners and stakeholders working on real-world ML products. We worked with three ML teams at Apple, drawn from the participatory design sessions, to integrate *Symphony* with their data and ML pipelines. The teams focus on different machine learning tasks, namely dataset creation and labeling, accessibility research, and ML education. To understand the affordances and limitations of *Symphony*, we conducted think-aloud studies lasting 60 minutes where a member of each team used *Symphony* to explore a Jupyter notebook and create a web-based dashboard for their data and model. While field studies like ours excel at capturing how participants actually work, this data has to be collected opportunistically. We believe these case studies capture the target audience of *Symphony*, cross-functional teams working on modern ML models trained on unstructured data, but may have some insights specific to organizational workflows.

Before the study, we sent a member of each team, the main participant, a Jupyter notebook that imported their data and displayed a set of *Symphony* components applicable to their domain and task. The study was split into three main sections. For the first third of the study, we asked the team to think aloud while the main participant used the notebook and *Symphony* components to explore the data and model freely. In the second part of the study, we asked the main participant to export the *Symphony* components (using a command in the notebook) to a standalone dashboard and continue exploring in the exported web UI. For the final part of the study, we asked the team for feedback on *Symphony* and discussed what types of use cases or limitations they found.

7.1 Case Study I: Validating and Sharing Data Patterns on a Dataset Creation Team

For the first case study, we worked with a team that assembles and labels large machine learning datasets. Their datasets are composed of labeled images and videos which they publish to an internal data repository. The team was interested in using *Symphony* in two ways, first, using it during dataset creation to detect errors in the data and labels, and second, as a reporting tool to give consumers of the dataset details about the data. Given these requirements, we loaded *Symphony* with the list (Figure 6 (B)), summary (Figure 6 (C)), duplicates (Figure 6 (F)), familiarity (Figure 6 (G)), projection (Figure 6 (H)), and map (Figure 6 (E)) components.

The main participant started in the notebook and used multiple components and interaction tools in concert to spot unexpected patterns in their data. They made extensive use of *Symphony*'s toolbar to combine filters and select subsets of data in which they were interested. When using the notebook, they commented that “*there are a lot of neat things here, first, the filter carried over, and it is so cool to see the data samples and metadata within the notebook.*” The synchronized, reactive state let them validate insights from the filtered summary charts with the actual raw instance previews in the list view. Next, the main participant moved on to the duplicates and familiarity components, where they found a couple of labeling errors that they suspected existed in their dataset but had not been able to validate previously. After transitioning to the standalone dashboard, the first component they looked at was the projection visualization. They used the projection to find a closely clustered

group of instances where a few highlighted points that the model had misclassified. In the standalone dashboard, they also dubbed the map visualization “*very useful*”, especially when sharing reports of their data collection efforts with managers or policymakers.

Overall, the team found “*a lot of value here*” when using *Symphony*. They mentioned that the workflow they would most prefer would be automatically generating shareable reports for every dataset they published: “*programmatic generation and live visualizations are awesome, being able to pop these charts into all our READMEs would be amazing.*” They saw the standalone dashboard that they created with *Symphony* as a “*great starting point*” for analyzing their datasets, and that they could see people use the notebooks for more detailed analysis: “*if people want to drill down more, and get exact specific access, summon the notebook.*” Being able to create different interfaces with subsets of visualization components was important for them as well, as different audiences have different needs and they “*do not want customers to do the data cleanup*” for them.

The team also identified usability issues and limitations in *Symphony*. When initially using the projection component, the main participant was not sure what it showed and thought that “*this component would need some introduction, as it has complex controls.*” They also requested additional components, such as heatmaps and other 2D graphs, to do a more detailed analysis of distributions. Lastly, the main limitation for directly using *Symphony* was not being able to attach the raw data files to a *Symphony* interface as their data samples are often not hosted and too large to duplicate.

7.2 Case Study II: Debugging Training Data on an Accessibility Team

In the second case study, we worked with a team that uses ML to make software applications more accessible. They have a large dataset of icon screenshots for which we assembled a similar set of components to the dataset creation team. We included the summary (Figure 6 (F)), duplicates (Figure 6 (A)), familiarity (Figure 6 (B)), and projection (Figure 6 (H)) components .

When exploring the notebook, the participant found the duplicates, familiarity, and scatterplot components to be the most interesting. Since they use an automated approach to collect their data, the participant assumed that there were likely duplicates in the dataset but had protocols to ensure they would not be across the training and testing set. Using the duplicates component, they confirmed that a significant number of icons were duplicates, but when they used the grouping interaction to split the data by testing and training they found that a significant number of instances were duplicated across the two datasets. The combination of the duplicates visualization and grouping interaction tool helped them discover that they “*were cheating learning on samples we test for.*” The participant identified the problematic duplicates and selected them in the notebook to remove from the test set with a Python command later. Next, the participant explored the familiarity component and found a large number of similar grey icons, based on which they wondered if “*the model might overfit on these samples.*” Finally, using the projection visualization, they found a dispersed cluster of instances with different labels. When they selected the group, they found that the instances were all PNG images in the

test set, while the training set only contained JPEG images. The participant then mentioned they “*want to test their model specifically on PNG images to assess how the model generalized.*”

Overall, the participant mentioned that they would “*want to try and use this to share insights within the team.*” Additionally, they found the notebook-based visualizations personally useful to “*look into the data,*” which they had previously done manually using a file explorer outside of the notebook. They mentioned that they would likely use a computational notebook to explore data, and only use the standalone dashboards to share insights or when they wanted more visualization space. The main feature the team wanted was to combine data and model findings to understand the impact of data changes: “*it would be super helpful to also add models and combine model analysis with existing components.*” While this analysis is possible with existing model analysis components, future components could specifically combine data and model information.

7.3 Case Study III: Promoting Data Exploration for ML Novices on an Education Team

For the final case study, we collaborated with a team focused on ML education. They teach courses about ML principles and techniques to engineers, and also teach their audience about data and model analysis tools. They sent us a list of datasets they commonly explore with students from which we selected two representative datasets, one audio dataset for data analysis and one image dataset for model analysis. For the audio dataset we used the same components as in the previous evaluation. To support model analysis for the image dataset, we used the summary (Figure 6 (F)), hierarchical confusion matrix (Figure 6 (D)), FairVis (Figure 6 (K)), and projection (Figure 6 (H)) components.

The team was interested in how they could use separate components in concert. They used the cross-filtering and grouping heavily to combine, for example, the projection visualization with the summary component to spot misclassified samples. They also used the confusion matrix visualization in combination with our filtering tool. For example, they filtered out the correctly classified data samples from the metadata table to highlight misclassifications and described the resulting confusion matrix as “*a fantastic graphic.*” They were also intrigued by being able to display a list of data samples in notebooks or a standalone dashboard, as they “*constantly tell [their] students to look at a lot of examples*” but are currently limited to seeing one or two instance at a time and “*just graph things using matplotlib or pillow.*”

Overall, the team found *Symphony* to be a valuable tool for ML tasks and thought it could play a part in one of their lessons, as “*promoting looking at data is extremely important.*” They remarked that “*in Python, its very easy to ignore the data, anything you can do to bring the data to the forefront is great.*” Thus, they wanted to use *Symphony* during their courses in multiple ways, namely using the “*notebook for generating interfaces, then exporting them to teach a group such that they can open the website and everyone can explore on their own or follow my instructions.*” This way, they hoped that “[students] can play with it and experiment talk about how to communicate results for ML models.” They also particularly liked the option to assemble visualizations, for example when their

students learn how to “*communicate findings to executives*” and “*graphing the relevant, and hiding the irrelevant.*”

As for limitations, they wanted to be able to unlink the state of different components to experiment with them independently. They also mentioned that they would like to load more data types than just the currently implemented audio, images, and tabular data, namely text data. While this is not possible right now, *Symphony* could be extended to more data types by augmenting the data sample adapter we provide.

8 LIMITATIONS AND FUTURE WORK

In both the pilot studies and case studies, we found ways in which *Symphony* could be further improved.

Authoring components. *Symphony* components are written using JavaScript code and web-based visualization libraries. Programming these visualizations requires expertise in web development and visualization, which limits who can create new components. Future work could explore ways to lower the barrier to authoring new visualization components. Potential strategies to make component creation more accessible include using grammars for interactive graphics, such as Vega [52], or UI-based visualization builders like Tableau [43]. Additional research would be needed to make these tools more expressive for unstructured data and ML models.

Scaling past millions of data points. *Symphony* currently loads the backing metadata table used for *Symphony* into web browser memory. This scales to tens of millions of data points, which, while sufficient for many modern ML tasks, does not cover all domains. In our design sessions, we spoke to teams with terabyte-scale metadata tables that do not fit in browser memory. Future work could explore ways to support this scale while still providing direct interactivity with the underlying data and models. Using an external API or backend for data processing combined with more efficient data queries could support massive data but would limit where the web-based UI could be used.

Beyond conventional data science platforms. In this work, we implemented *Symphony* wrappers for computational notebooks, programming environments, and web-based dashboards. While these platforms cover a significant portion of where ML work happens, future work could explore how *Symphony* could be incorporated into other platforms, especially those which are currently isolated from data science work. For example, *Symphony* interfaces could be included in messaging services, documents, presentations, or issue trackers to further bring the benefits of *Symphony* to more people. New design studies could be conducted to understand how users in common communication platforms like instant messaging would benefit from and use *Symphony* components.

Guided usage of Symphony. ML practitioners can use *Symphony* for a wide array of ML analyses, from dataset debugging to auditing models for bias. This gamut of uses stands in contrast to more *prescriptive* approaches like Datsheets [17], Model Cards [46], and checklists [44] which define an ordered list of what an ML interface should show. While ML practitioners can use *Symphony* for more types of analyses, it does not provide any guidance to users about which components might be the most adequate or useful for a given

task. Future work could look at combining *Symphony*'s open-ended, exploratory approach with more prescriptive guidance.

Scope of case study findings. Lastly, our case studies were conducted with ML practitioners at a single institution that works on large ML models trained on unstructured data, often using notebooks and visualization dashboards. While we believe these tools and ML development practices exist widely in industry and academia, we recognize some of our findings may not generalize to other organizations or types of users such as machine learning enthusiasts, hobbyists, or small teams. Further studies could explore *Symphony*'s affordances and drawbacks in these distinct settings.

9 DISCUSSION

Symphony provides a common substrate for ML interfaces that enables both exploratory analysis and sharable ML interfaces. By meeting different users where they work, *Symphony* empowers each member of an ML team to have direct access and knowledge of the data and models powering an AI product.

While the case studies described scenarios where ML practitioners work in programming environments and then transition to web-based UIs, we also observed in our studies that ML practitioners can benefit from going the opposite direction: transitioning from a web-based UI back to a programming environment. When a user finds an interesting insight in a standalone *Symphony* dashboard, they can copy their findings to the programming environments along with state variables like filters and groups. Existing analysis tooling often suffers from an “expressiveness cliff”, where only a fixed set of visualizations and data manipulations is available. *Symphony* allows users to return to programming environments where they have more flexible analysis tools.

ML practitioners' desire to use *Symphony* for exploration could also encourage them to share their insights more frequently. If ML practitioners are using a set of *Symphony* components for exploratory analysis in a notebook, no additional work is needed for them to export it as a standalone, shareable UI. Participants mentioned the ability to programmatically combine components as a major benefit, allowing them to go from exploration to an interactive, web-based UI without using a different tool. Additionally, *Symphony* interfaces can be redeployed continuously whenever the data and model are updated, supporting ML tasks with streaming data or automatic model retraining.

By integrating with existing data science platforms, *Symphony* could also encourage broader use of task-specific ML visualizations. ML visualization systems are often implemented as one-off web dashboards [1, 9, 10, 22, 39, 64] that require users to wrangle and export their data into systems separate from where they do ML development. *Symphony* includes task-specific visualization components directly in data since platforms like Jupyter notebooks, and the components can consume data from standard data APIs like Pandas Data Frames. In turn, implementing ML visualizations as independent components in a framework like *Symphony* could increase their use and longevity.

Beyond helping individuals understand ML systems, *Symphony* is intended to foster a shared organizational understanding [69] between stakeholders on an ML team. *Symphony* interfaces act

as *boundary objects* for large, cross-functional ML teams. Boundary objects are artifacts that are “*both plastic enough to adapt to local needs and the constraints of the several parties employing them, yet robust enough to maintain a common identity across sites*” [56]. *Symphony* can serve as a boundary object for ML teams, providing interfaces that adapt to the different needs of stakeholders. At the same time, “*The creation and management of boundary objects is a key process in developing and maintaining coherence across intersecting social worlds*” [56]. *Symphony* aids in this creation and management process, bridging the gap between the intersecting worlds of different ML stakeholders such as engineers, designers, and product managers.

10 CONCLUSION

In this work, we designed and implemented *Symphony*, a framework for composing interactive ML interfaces with data-driven, task-specific visualization components. *Symphony*'s visualizations helped ML teams find important issues such as data duplicates and model blind spots. Additionally, We found that by providing ML interfaces in the data science platforms where ML practitioners work, *Symphony* can encourage ML practitioners to *want* to use and share insights. With data-driven components that diverse stakeholders across an ML team can use, *Symphony* fosters a culture of shared ML understanding and encourages the creation of accurate, responsible, and robust AI products.

ACKNOWLEDGMENTS

We thank our colleagues at Apple for their time and effort integrating our research with their work. We especially thank Kayur Patel for his guidance and Mary Beth Kery for her generosity reviewing early drafts of this work.

REFERENCES

- [1] Yongsu Ahn and Yu-Ru Lin. 2019. Fairsight: Visual analytics for fairness in decision making. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (2019), 1086–1095.
- [2] Robert Amar, James Eagan, and John Stasko. 2005. Low-level components of analytic activity in information visualization. In *IEEE Symposium on Information Visualization, INFO VIS*. IEEE, 111–117.
- [3] 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*. 337–346.
- [4] Matthew Arnold, Rachel KE Bellamy, Michael Hind, Stephanie Houde, Sameep Mehta, Aleksandra Mojsilović, Ravi Nair, K Natesan Ramamurthy, Alexandra Olteanu, David Piorkowski, et al. 2019. FactSheets: Increasing trust in AI services through supplier's declarations of conformity. *IBM Journal of Research and Development* 63, 4/5 (2019), 6–1.
- [5] Andrea Batch, Niklas Elmquist, and Senior Member. 2018. The interactive visualization gap in initial exploratory data analysis. *IEEE Transactions on Visualization and Computer Graphics* 24 (2018), 278–287.
- [6] Alex Bäuerle, Heiko Neumann, and Timo Ropinski. 2020. Classifier-guided visual correction of noisy labels for image classification tasks. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 195–205.
- [7] Emily M Bender and Batya Friedman. 2018. Data statements for natural language processing: Toward mitigating system bias and enabling better science. *Transactions of the Association for Computational Linguistics* 6 (2018), 587–604.
- [8] Joy Buolamwini and Timnit Gebru. 2018. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on Fairness, Accountability and Transparency*. PMLR, 77–91.
- [9] Ángel Alexander Cabrera, Will Epperson, Fred Hohman, Minsuk Kahng, Jamie Morgenstern, and Duen Horng Chau. 2019. FairVis: Visual analytics for discovering intersectional bias in machine learning. In *2019 IEEE Conference on Visual Analytics Science and Technology*. IEEE, 46–56.

- [10] Dylan Cashman, Adam Perer, Remco Chang, and Hendrik Strobelt. 2019. Ablate, variate, and contemplate: Visual analytics for discovering neural architectures. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (2019), 863–873.
- [11] Changjian Chen, Jun Yuan, Yafeng Lu, Yang Liu, Hang Su, Songtao Yuan, and Shixia Liu. 2020. Oodanalyzer: Interactive analysis of out-of-distribution samples. *IEEE Transactions on Visualization and Computer Graphics* 27, 7 (2020), 3335–3349.
- [12] Nan-Chen Chen, Jina Suh, Johan Verwey, Gonzalo Ramos, Steven Drucker, and Patrice Simard. 2018. AnchorViz: Facilitating classifier error discovery through interactive semantic data exploration. In *23rd International Conference on Intelligent User Interfaces*. 269–280.
- [13] European Commission. 2019. *Ethics guidelines for trustworthy AI*. <https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai>
- [14] Robert A DeLine. 2021. Glinda: Supporting data science with live programming, GUIs and a Domain-specific Language. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–11.
- [15] Luciano Floridi. 2019. Establishing the rules for building trustworthy AI. *Nature Machine Intelligence* 1, 6 (2019), 261–262.
- [16] Jules Françoise, Baptiste Caramiaux, and Téo Sanchez. 2021. Marcelle: Composing interactive machine learning workflows and interfaces. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 39–53.
- [17] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, and Kate Crawford. 2021. Datasheets for datasets. *Commun. ACM* 64, 12 (2021), 86–92.
- [18] Andrew Head, Fred Hohman, Titus Barik, Steven M Drucker, and Robert DeLine. 2019. Managing messes in computational notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–12.
- [19] Lisa Anne Hendricks, Kaylee Burns, Kate Saenko, Trevor Darrell, and Anna Rohrbach. 2018. Women also snowboard: Overcoming bias in captioning models. In *Proceedings of the European Conference on Computer Vision*. 771–787.
- [20] Andreas Hinterreiter, Peter Ruch, Holger Stitz, Martin Ennemoser, Jurgen Bernard, Hendrik Strobelt, and Marc Streit. 2020. Confusionflow: A model-agnostic visualization for temporal analysis of classifier confusion. *IEEE Transactions on Visualization and Computer Graphics* (2020).
- [21] Fred Hohman, Minsuk Kahng, Robert Pienta, and Duen Horng Chau. 2018. Visual analytics in deep learning: An interrogative survey for the next frontiers. *IEEE Transactions on Visualization and Computer Graphics* (2018). <https://doi.org/10.1109/TVCG.2018.2843369>
- [22] Fred Hohman, Haekyu Park, Caleb Robinson, and Duen Horng Polo Chau. 2019. Summit: Scaling deep learning interpretability by visualizing activation and attribution summarizations. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (2019), 1096–1106.
- [23] Sarah Holland, Ahmed Hosny, Sarah Newman, Joshua Joseph, and Kasia Chmielinski. 2018. The dataset nutrition label: A framework to drive higher data quality standards. *arXiv preprint arXiv:1805.03677* (2018).
- [24] Holoviz. 2021. *Panel*. <https://panel.holoviz.org/>
- [25] J. D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95. <https://doi.org/10.1109/MCSE.2007.55>
- [26] DataBricks Inc. 2021. *DataBricks*. <https://databricks.com/>
- [27] Google Inc. 2021. *Facets*. <https://pair-code.github.io/facets/>
- [28] Google Inc. 2021. *Know Your Data*. <https://knowyourdata.withgoogle.com/>
- [29] Observable Inc. 2021. *Observable*. <https://observablehq.com/>
- [30] Plotly Technologies Inc. 2015. *Collaborative data science*. Montreal, QC. <https://plotly.com/>
- [31] Streamlit Inc. 2021. *Streamlit*. <https://streamlit.io/>
- [32] Jupyter. 2021. *IPyWidgets*. <https://ipywidgets.readthedocs.io/en/stable/>
- [33] Minsuk Kahng, Pierre Y. Andrews, Aditya Kalro, and Duen Horng Polo Chau. 2018. ActiVis: Visual exploration of industry-scale deep neural network models. *IEEE Transactions on Visualization and Computer Graphics* 24 (2018), 88–97. Issue 1. <https://doi.org/10.1109/TVCG.2017.2744718>
- [34] Minsuk Kahng, Dezhi Fang, and Duen Horng Chau. 2016. Visual exploration of machine learning results using data cube analysis. *HILDA 2016 - Proceedings of the Workshop on Human-In-the-Loop Data Analytics* (2016). <https://doi.org/10.1145/2939502.2939503>
- [35] Mary Beth Kery, Marissa Radensky, Mahima Arya, Bonnie E John, and Brad A Myers. 2018. The story in the notebook: Exploratory data science using a literate programming tool. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–11.
- [36] Mary Beth Kery, Donghao Ren, Fred Hohman, Dominik Moritz, Kanit Wong-suphasawat, and Kayur Patel. 2020. mage: Fluid moves between code and graphical work in computational notebooks. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 140–151.
- [37] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. 2016. Jupyter Notebooks – a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt (Eds.). IOS Press, 87 – 90.
- [38] Laura Koesten, Emilia Kacprzak, Jeni Tennison, and Elena Simperl. 2019. Collaborative practices with structured data: Do tools support what users need?. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–14.
- [39] Josua Krause, Adam Perer, and Kenney Ng. 2016. Interacting with predictions: Visual inspection of black-box machine learning models. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 5686–5697.
- [40] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).
- [41] Doris Jung-Lin Lee, Dixin Tang, Kunal Agarwal, Thyne Boonmark, Caitlyn Chen, Jake Kang, Ujjaini Mukhopadhyay, Jerry Song, Micah Yong, Marti A. Hearst, and Aditya G. Parameswaran. 2022. Lux: Always-on visualization recommendations for exploratory data science. *Proceedings of the VLDB Endowment* 15, 3 (2022), 727–738.
- [42] Fritz Lekschas. 2021. *Regl Scatterplot*. <https://github.com/flekschas/regl-scatterplot>
- [43] Tableau Software LLC. 2021. *Tableau*. <https://www.tableau.com/>
- [44] Michael A Madaio, Luke Stark, Jennifer Wortman Vaughan, and Hanna Wallach. 2020. Co-designing checklists to understand organizational challenges and opportunities around fairness in ai. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–14.
- [45] Wes McKinney et al. 2010. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, Vol. 445. Austin, TX, 51–56.
- [46] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2019. Model cards for model reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*. 220–229.
- [47] Luke Oakden-Rayner, Jared Dunmon, Gustavo Carneiro, and Christopher Ré. 2020. Hidden stratification causes clinically meaningful failures in machine learning for medical imaging. In *Proceedings of the ACM Conference on Health, Inference, and Learning*. 151–159.
- [48] Kayur Patel, Naomi Bancroft, Steven M Drucker, James Fogarty, Andrew J Ko, and James Landay. 2010. Gestalt: Integrated support for implementation and analysis in machine learning. In *Proceedings of the 23nd annual ACM Symposium on User Interface Software and Technology*. 37–46.
- [49] Plotly. 2021. *Dash*. <https://plotly.com/dash/>
- [50] Donghao Ren, Saleema Amershi, Bongshin Lee, Jina Suh, and Jason D. Williams. 2017. Squares: Supporting interactive performance analysis for multiclass classifiers. *IEEE Transactions on Visualization and Computer Graphics* 23 (2017), 61–70. Issue 1. <https://doi.org/10.1109/TVCG.2016.2598828>
- [51] Dominik Sacha, Michael Sedlmair, Leishi Zhang, John A Lee, Jaakkko Peltonen, Daniel Weiskopf, Stephen C North, and Daniel A Keim. 2017. What you see is what you can change: Human-centered machine learning by interactive visualization. *Neurocomputing* 268 (2017), 164–175.
- [52] Arvind Satyanarayan, Ryan Russell, Jane Hoffswell, and Jeffrey Heer. 2015. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (2015), 659–668.
- [53] Ben Shneiderman. 2020. Bridging the gap between ethics and practice: Guidelines for reliable, safe, and trustworthy Human-Centered AI systems. *ACM Transactions on Interactive Intelligent Systems* 10, 4 (2020), 1–31.
- [54] Jake Silberg and James Manyika. 2019. Notes from the AI frontier: Tackling bias in AI (and in humans). *McKinsey Global Institute (June 2019)* (2019).
- [55] Jacob Snow. 2018. Amazon's face recognition falsely matched 28 members of congress with mugshots. *American Civil Liberties Union* 28 (2018).
- [56] Susan Leigh Star and James R Griesemer. 1989. Institutional ecology, translations' and boundary objects: Amateurs and professionals in Berkeley's Museum of Vertebrate Zoology, 1907–39. *Social Studies of Science* 19, 3 (1989), 387–420.
- [57] Hendrik Strobelt, Sebastian Gehrmann, Michael Behrisch, Adam Perer, Hanspeter Pfister, and Alexander M. Rush. 2019. Seq2seq-VIs: A visual debugging tool for sequence-to-sequence models. In *IEEE Transactions on Visualization and Computer Graphics*, Vol. 25. 353–363. <https://doi.org/10.1109/TVCG.2018.2865044>
- [58] Jacob VanderPlas, Brian Granger, Jeffrey Heer, Dominik Moritz, Kanit Wong-suphasawat, Arvind Satyanarayan, Eitan Lees, Ilia Timofeev, Ben Welsh, and Scott Sievert. 2018. Altair: Interactive statistical visualizations for python. *Journal of Open Source Software* 3 (2018), 1057. Issue 32. <https://doi.org/10.21105/joss.01057>
- [59] Alfredo Vellido. 2020. The importance of interpretability and visualization in machine learning for applications in medicine and health care. *Neural Computing and Applications* 32, 24 (2020), 18069–18083.
- [60] Voila. 2021. *Voila*. <https://github.com/voila-dashboards/voila>
- [61] Angelina Wang, Arvind Narayanan, and Olga Russakovsky. 2020. REVISE: A tool for measuring and mitigating bias in visual datasets. In *European Conference on Computer Vision*. Springer, 733–751.
- [62] James Wexler, Mahima Pushkarna, Tolga Bolukbasi, Martin Wattenberg, Fernanda Viégas, and Jimbo Wilson. 2020. The what-if tool: Interactive probing of machine learning models. *IEEE Transactions on Visualization and Computer Graphics* 26 (2020), 56–65. Issue 1. <https://doi.org/10.1109/TVCG.2019.2934619>

- [63] Benjamin Wilson, Judy Hoffman, and Jamie Morgenstern. 2019. Predictive inequity in object detection. *arXiv preprint arXiv:1902.11097* (2019).
- [64] Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel S Weld. 2019. Errudite: Scalable, reproducible, and testable error analysis. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 747–763.
- [65] Yifan Wu, Joseph M Hellerstein, and Arvind Satyanarayanan. 2020. B2: Bridging code and interactive visualization in computational notebooks. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 152–165.
- [66] Shouxing Xiang, Xi Ye, Jiazhixia, Jing Wu, Yang Chen, and Shixia Liu. 2019. Interactive correction of mislabeled training data. In *2019 IEEE Conference on Visual Analytics Science and Technology*. IEEE, 57–68.
- [67] Ji Soo Yi, Youn ah Kang, John Stasko, and Julie A Jacko. 2007. Toward a deeper understanding of the role of interaction in information visualization. *IEEE Transactions on Visualization and Computer Graphics* 13, 6 (2007), 1224–1231.
- [68] J M Zhang, M Harman, L Ma, and Y Liu. 2020. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering* (2020), 1. <https://doi.org/10.1109/TSE.2019.2962027>
- [69] Ángel Cabrera and Elizabeth F. Cabrera. 2002. Knowledge-sharing dilemmas. *Organization Studies* 23, 687–710. Issue 5. <https://doi.org/10.1177/0170840602235001>

VISUALIZATION-BASED NEURAL NETWORK INTROSPECTION

ALEX BÄUERLE

20.07.2022

