Visualizing Examples of Deep Neural Networks at Scale

Litao Yan
Harvard University
Cambridge, MA, USA
litaoyan@g.harvard.edu

Elena L. Glassman
Harvard University
Cambridge, MA, USA
glassman@seas.harvard.edu

Tianyi Zhang
Harvard University
Cambridge, MA, USA
tianyi@seas.harvard.edu

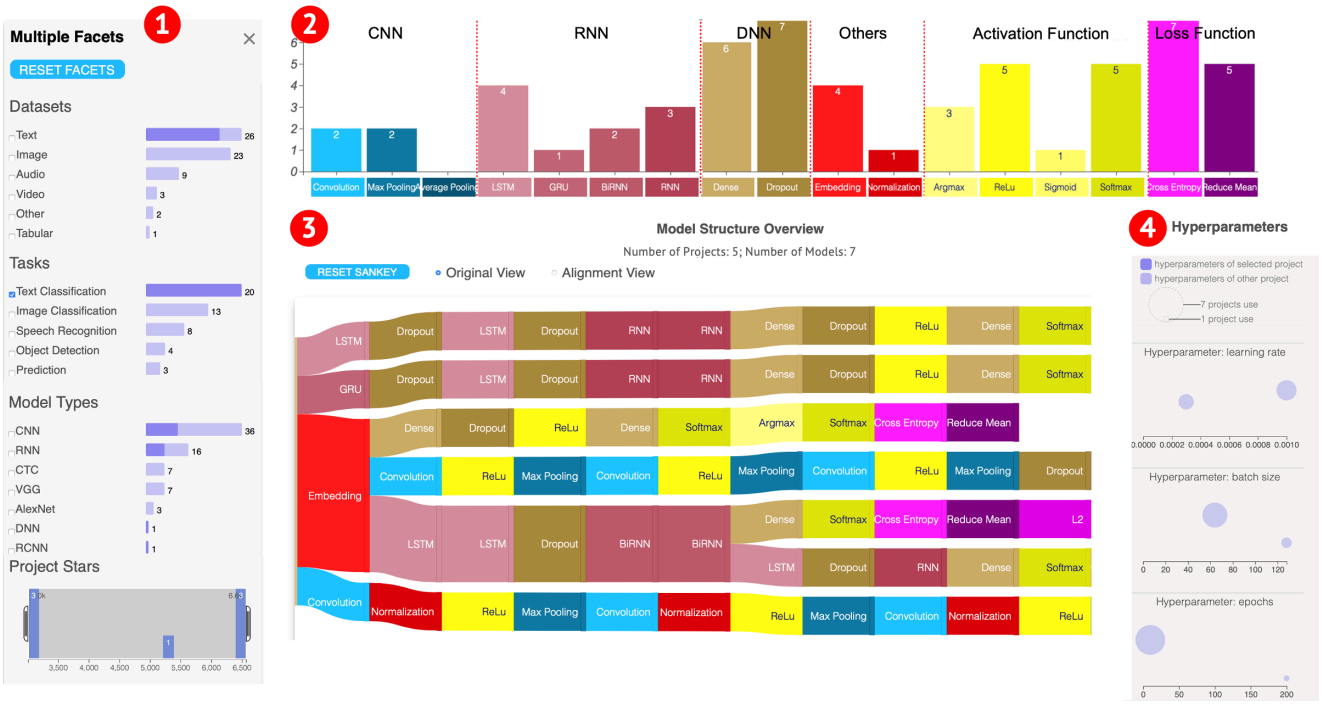


Figure 1: ExampleNet, an interface to explore the commonalities and variations in relevant neural network models built by other GitHub developers: (1) a faceted browser to identify relevant models, (2) the distribution of various layers used by other developers, (3) an overview diagram of various model structures, and (4) the distribution of hyperparameters used by others.

ABSTRACT

Many programmers want to use deep learning due to its superior accuracy in many challenging domains. Yet our formative study with ten programmers indicated that, when constructing their own deep neural networks (DNNs), they often had a difficult time choosing appropriate model structures and hyperparameter values. This paper presents ExampleNet—a novel interactive visualization system for exploring common and uncommon design choices in a large collection of open-source DNN projects. ExampleNet provides a holistic view of the distribution over model structures and hyperparameter settings in the corpus of DNNs, so users can easily filter the corpus down to projects tackling similar tasks and compare design choices made by others. We evaluated ExampleNet in a within-subjects study with sixteen participants. Compared with the control condition (i.e., online search), participants using ExampleNet were able to inspect more online examples, make more data-driven design decisions, and make fewer design mistakes.

CCS CONCEPTS

• **Human-centered computing** → **Human computer interaction (HCI)**; **Interactive systems and tools**.

KEYWORDS

Visualization; deep neural networks; code examples

ACM Reference Format:
Litao Yan, Elena L. Glassman, and Tianyi Zhang. 2021. Visualizing Examples of Deep Neural Networks at Scale. In *CHI Conference on Human Factors in Computing Systems (CHI '21), May 8–13, 2021, Yokohama, Japan.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3411764.3445654

CHI '21, May 8–13, 2021, Yokohama, Japan

ACM ISBN 978-1-4503-8096-6/21/05…$15.00
https://doi.org/10.1145/3411764.3445654

1 INTRODUCTION

In recent years, a particular form of machine learning (ML)—deep neural networks (DNNs)—has gained a lot of attention. More and more programmers now want to learn and tinker with deep learning models, mainly through online tutorials and blogs [4]. As tutorials and blogs tend to only include a few simple examples for illustration purposes, programmers often struggle to identify appropriate model architectures for their own usage scenarios. When seeing a specific design choice in an example model, they often wonder how common this design choice is, how suitable is it for their own tasks and datasets, and what other options are available.

In a formative study with ten participants, we found that most participants (9/10) said they searched online for tutorials, blogs, and example models when building their own models. However, searching online often makes them feel overwhelmed because of the enormous amount of online resources. All of them complained about the difficulty of searching and navigating online examples to find desired, relevant, easy-to-understand models. They expressed their needs to understand the network structures and hyperparameters used by other developers on related tasks and datasets. However, due to the tremendous resources available online, they are unable to quickly search, navigate, and assess such neural network design decisions using online search.

In addition to the information needs that these aspiring DNN programmers described to us in the formative study, we also consulted relevant theory, i.e., Variation Theory [16], a theory about how humans effectively learn concepts, like "what is a DNN?" directly from examples. Variation Theory suggests that, for every object of learning, there are critical dimensions of variation and critical values along that dimension that learners need to discern. These critical dimensions and values become discernable by showing examples that are similar and different to it along these critical dimensions. For example, an English speaker may be told that there are tones in tonal languages that change the meaning of a word, but until they hear two words that are identical except for the tone, they cannot discern what the concept of a tone refers to. Similarly, they may not be able to discern a particular tone until they hear multiple words that share the same tone but vary in other ways. In the context of building DNNs, they may need to see many similar and different examples of DNNs to understand all the different dimensions (e.g., types of layers, sequences of layers, etc.) that they can play with while constructing their own DNNs without it ceasing to be a DNN. Once they can discern these dimensions of variation, they may also want to anchor their initial choices on the common design choices of others, while knowing that they can vary these choices at least as far as the revealed uncommon choices.

Based on those identified information needs and the relevant theory, we summarize three design principles. First, our system should help users understand the relevance of any example DNN to their own task. Second, our system needs to help users compare and contrast DNN examples on the basis of the design decisions they care about, e.g., model type and structure, hyperparameter values for the network and individual chosen layers, etc. Third, our system needs to help users see commonalities and variations of these design choices over a large sample of available DNN examples.

In this paper, we introduce ExampleNet—a novel interactive visualization interface that (1) provides users a holistic view to explore common and uncommon design decisions, such as neural network architectures and hyperparameters, in a large collection of deep learning projects, (2) allows users to quickly filter a corpus of DNNs down to a subset that tackles similar tasks, and (3) compare and contrast the design decisions made by other developers represented in that corpus. The faceted browser (Figure 1 ①) in ExampleNet assists users in quickly selecting and filtering by the model's metadata. The overview diagram of model structures (Figure 1 ③) aggregates and aligns the structure of various models in a single view so that users can compare and contrast the commonalities and variations of layers and their arrangement. ExampleNet also includes the summative distribution of layer types (Figure 1 ②) for users to explore common layers used by others. The distribution of hyperparameters (Figure 1 ④) shows the different hyperparameter values from a large number of models so that users can have a comprehensive understanding of common and uncommon hyperparameter settings.

We conducted a within-subjects user study with sixteen DL programmers of various levels of expertise in computer vision and natural language process. Participants were asked to complete two DL tasks by using either online search or ExampleNet to design neural network structures and hyperparameter settings. We found that when using ExampleNet, participants (1) navigated more online examples, (2) made more data-driven design decisions, such as using more types of layers and hyperparameters, and (3) made fewer design mistakes, e.g., leaving out an activation function or loss function, setting the epoch to an unworkably large value, etc. The value of ExampleNet is perhaps best described by some of the participants themselves: "ExampleNet *gives a summary of all models for every specific machine learning task, and users can have a big picture of the neural network construction choices.*" (P3); "*The visualization of model architecture is also quite informative in showing what are some common architectures shared across different projects, while also shows how each network differs from one another.*" (P7).

Our contributions are:

- A formative study that discovers the obstacles and needs of DL programmers when designing a deep neural network
- An implementation of this interactive visualization for a set of deep learning projects on GitHub
- A within-subjects user study that demonstrates the usefulness of ExampleNet to DL programmers when designing their own neural networks

2 RELATED WORK

2.1 Learning Barriers in Deep Learning

Cai and Guo did a large survey with 645 software developers about their desire to learn ML and the learning hurdles they face [4]. They found that developers' desire to use ML frameworks extended beyond simply wanting help with APIs: "developers desired ML frameworks to teach them not only how to use the API, but also the implicit best practices and concepts that would enable them to effectively apply the framework to their particular problems." This motivates our focus on high-level design choices of building neural networks rather than low-level implementation details such as API usage. In addition, they also found out that online tutorials

and blogs often only offer a limited set of examples, falling short of helping users identify an appropriate model architecture for their own tasks. As a result, developers were left to make many design decisions at their own discretion, e.g., "how many convolution layers do I need?", "what dropout rate or optimizer should I use?".

Some other studies and surveys have also investigated programmers' practice of applying machine learning. Amershi et al. did a case study at Microsoft and found that AI is completely different from traditional software applications [1]. For example, machine learning applications have more complex data; model customization and reuse require more complex skills; and AI components are more difficult to process as separate modules. Yang et al. pointed out that most non-experts simply use pre-trained ML models as black-box tools and integrate them into their own applications, which sometimes leads to difficult or unrealistic learning goals [30]. Patel et al. identified three major obstacles of applying ML as a tool in software development, such as the difficulty of using ML in an iterative and exploratory way [17]. Dove et al. [5]and Yang et al. [29] probed the challenges that UX designers face in working with ML. Both of them found that UX designers have difficulties understanding ML, its capabilities, and its limitations.

2.2 Example-based Programming Learning

As the Internet accumulates a large volume of code and code-related artifacts, many programmers now resort to online resources while writing code [3, 20, 22, 28]. Sadowski et al. found that Google developers issued an average of 12 online code search queries per weekday [20]. Brandt et al. observed that, when writing code, programmers typically started with searching for relevant tutorials and then used the code examples in these tutorials as the scaffold for their own implementations [3]. Head et al. proposed an interactive approach that extracts runnable code examples from GitHub projects [10].

Stack Overflow (SO) is a popular site for asking and answering programming questions. Wu et al. surveyed SO users to investigate the remaining barriers of programming when finding assistance on the site. Among 453 respondents, 65% said they had to manually adapt SO examples to fit their own usage scenarios, 44% found some examples hard to understand, and 32% complained about the low quality of some examples. Besides, Zhang et al. identified the needs of API designers and discussed how community-generated API usage data can be leveraged to address those needs [32]. The results of our formative study on learning DL are highly consistent with these previous findings: 1) participants always searched for examples before building their own neural network models, and 2) participants found it difficult to identify desired information from many search results and assess their relevance for their own usage scenario. On the other hand, our participants expressed more interest in finding out high-level design choices such as the model structure, rather than learning low-level implementation details such as how to make a particular API call.

2.3 Deep Neural Network Visualization

Many neural network visualization tools have been proposed to support different activities in neural network development. TensorBoard [27] and Sony's Neural Network Console [24] provide visualizations for a single network and its layer parameters. They are primarily designed to facilitate model training, providing different features to monitor the training process. Other visualization tools, such as LSTMVis [26] and TensorFlow Playground [23], are designed to increase the interpretability of a pre-trained model, by visualizing the weight updates and hidden layers in the model.

ExampleNet differs from these visualization tools in three perspectives. First, unlike TensorBoard and Sony's Neural Network Console, which focus more on assisting users to debug and train a better model, ExampleNet targets the model design phase, in which developers can explore and discover various design choices of model structures and hyperparameter settings. Second, these visualization tools only represent a single model at a time. They do not allow users to easily compare and contrast multiple models, let alone the distribution of their design choices over an entire corpus of DNN models. Third, visualization tools such as LSTMVis and TensorFlow Playground visualize aspects of the model for interpretability purposes. However, they do not render hyperparameter settings, which are essential for beginners to design a runnable model.

2.4 Interfaces for Exploring Collections of Code and Tutorial Examples

Previous work has explored different ways of visualizing large collections of examples for D3 visualization [11], API usage examples [6], website design [13], and Photoshop tutorials [12]. Hoque et al. [11] present an approach for searching and exploring different visualization styles for a large number of D3 visualizations. Similar to Hoque et al. 's approach, the Adaptive Ideas Web design tool [14] uses an example gallery for users to choose and adapt website design ideas. Apart from these two interfaces, Delta [12] uses thumbnail images to visualize the workflows in multiple PhotoShop tutorials from different aspects. All of these interfaces visualize the examples in a stacked and grouped view, which is hard to directly compare and contrast the commonalities and variations of critical aspects of DNNs, such as the sequence of layers. In ExampleNet, we use a Sankey diagram to visualize each model side by side in a single view, with the option to align similar layers across models. In this way, users can have a bird's-eye view of the common and uncommon model architectures and how they differ.

To our best knowledge, Examplore [6] is the only work that aligns and aggregates a large collection of examples in a single view. Examplore uses a pre-defined code skeleton to visualize API usage examples, which cannot be directly reused for visualizing DNNs. It is difficult to define a particular skeleton that includes all the various architectures. In ExampleNet, instead of designing such a skeleton, we present a different approach—directly visualizing the model structures in a Sankey diagram and using a local alignment algorithm to further align them by layer types.

3 FORMATIVE STUDY

3.1 Participants

We conducted a formative study with 10 graduate students (6 females and 4 males) who have taken a deep learning class at Harvard University. Three participants have more than five years of programming experience, six have two to five years of programming

experience, and one has one-year experience. As for machine learning experience, half of them have two to five years of experience, three have only one year of experience, and two only have one semester of experience. Nine of the ten participants have used TensorFlow before, and four of the ten participants have used PyTorch. They have worked on different kinds of deep learning projects such as image recognition, object detection, and natural language processing. Participants were compensated with a $15 Amazon gift card for their time.

3.2 Methodology

We conducted a 45-min semi-structured interview with each participant. During the interview, we first asked about their learning experiences with neural networks. Specifically, we asked what kinds of neural network projects they have worked on, what kinds of challenges they faced, and what kinds of online resources they found useful. We also asked whether and how often they searched for examples when building neural networks, what information cues they intended to discover from those examples, and what kinds of difficulties they experienced.

Finally, we showed them TensorBoard [27], a popular neural network visualization tool. TensorBoard visualizes neural network layers and low-level computational operations such as additions of tensors as a dataflow graph. All ten participants said they had used TensorBoard before. We asked them what they liked and disliked about TensorBoard and whether it can surface those information cues they wished to discover from examples of neural networks.

During the interview, we encouraged participants to contextualize their answers based on their recent experience of learning and building deep learning models. Two authors coded participants' responses to each question and then categorized them following the card sorting method [15]. We synthesize our major findings in the next section.

3.3 Participants' Responses

3.3.1 Learning and Building Neural Networks. Programmers often search and adapt example neural networks on GitHub rather than building a neural network from scratch. Nine of ten participants said the first thing they would do was to search for GitHub projects that perform similar tasks on similar datasets. For example, P8 said, *"when I need to process images, I will search CNN and other keywords in GitHub, and identify similar projects to see what other people have done with images."* When asked about how they decide which GitHub project to follow or reuse, participants said they cared the most about the relevance to their own tasks and datasets. After they have decided on a GitHub project, they adapt the model structure to fit their own data. P7 mentioned, *"based on our data, we may change our (network) structure and add few more layers behind or in front of the original network."*

3.3.2 The Information Needs of Deep Learning Programmers. Table 1 lists the common information cues our participants wished to discover from GitHub examples when designing neural networks. First, eight participants wished to get a holistic view of different neural networks for similar tasks (***N1, N5***). P4 said, *"when I searched for models with the same task, I can only browse one example at a time, and I cannot compare other related examples at the same time."* In particular, five participants emphasized that they did not want to investigate all projects returned by GitHub Search but only those processing similar tasks and datasets as their own (***N7***). However, it is cumbersome to assess the relevance of a GitHub project. P7 explained, *"there is a project about some kinds of NLP tasks, but I don't know what kind of datasets they are using, or what kind of data format. I have to search in the documents to look for the datasets."* Hence, participants wished to have some tool support for distilling information such as tasks and training data from GitHub projects to help them make quick assessment.

Second, most participants expressed a desire to understand the high-level design decisions in related models in GitHub (***N2, N3, N4, N6***). Eight participants were interested in identifying the structure of neural networks (***N2***). However, it is difficult to identify model structures from GitHub projects. P4 complained, *"sometimes there are thousands of lines of source code in several different files, so you can barely have a clear overview of what the model looks like."* Nine participants wanted to understand the "tricks" used by other programmers to improve their model performance (***N3***). In addition, participants wanted to compare the hyperparameters in different models (***N4***) and identify the common choices made by other programmers (***N6***).

Participants also mentioned several information cues such as runnability and model accuracy, which are important for them to decide which model to follow (***N8, N9***). Participants put more trust in the design choices made within models with high accuracy. Yet if a highly accurate model requires many GPUs and takes a lot of time to train, they were less willing to follow and experiment with the model. Finally, several participants wanted to know what kinds of data preprocessing steps, e.g., standardization, one-hot encoding, etc., were performed in the projects (***N10***).

3.3.3 The Challenges of Identifying Desired Information. When asked about the difficulty of discovering those information cues, seven participants said they were overwhelmed by searching and navigating through related projects. P3 said, *"sometimes [GitHub] gives us too many other details that you will not use."* P4 added that *"the README files are so rough and do not describe what they are doing in their repo."* Eight participants complained about the difficulty of assessing the relevance and quality of GitHub projects in the search results. P4 said, *"even though we can sort the results in GitHub, I still need to go through each result to further identify whether it is related to what I am doing."* P8 said, *"only looking at the title or description [of a GitHub project] is not enough. I still need to check the README file or read the code directly to know what exactly they are doing."*

Four participants mentioned the difficulty of comparing and contrasting different GitHub projects. P4 said that *"after I found a suitable example, I'm still not sure what other people will do. For example, whether other people will use the same layer here, or whether other people will use the same value of this parameter."* As a result, participants found it difficult to decide which GitHub project to use. P8 said *"I don't know which model is a better match for my task, and there is no place to compare them."*

Four participants were concerned about the lack of runtime information in GitHub projects. P5 said *"I think building the environment is the most difficult. Every time after you download a GitHub repo,*

Information Needs	Participants
N1. What are different neural networks for similar tasks and datasets?	P1, P2, P3, P4, P6, P7, P8, P10
N2. I want to quickly find out the structure of a model in a project.	P1, P2, P3, P4, P6, P8, P9, P10
N3. What kinds of "tricks" (e.g., attention, dropout) have other programmers used?	P1, P2, P3, P4, P5, P6, P7, P8, P10
N4. Is my hyperparameter setting similar to those in popular projects?	P1, P2, P3, P4, P5, P6, P7, P8, P10
N5. What kinds of models are often used for specific datasets and tasks?	P1, P2, P3, P4, P6, P7, P8, P10
N6. What are the common hyperparameters set by others?	P2, P3, P4, P5, P6, P7, P8, P9, P10
N7. Do these projects use similar datasets and perform similar tasks as mine?	P1, P2, P3, P4, P5
N8. Is this model runnable? How easy? What is the running environment?	P1, P3, P7, P9, P10
N9. What is the accuracy of the model? How long does it take to train?	P1, P2, P5, P10
N10. How do others pre-process their data before feeding to a model?	P1, P2, P6, P7

Table 1: The common information cues that participants wish to discover

[you] need a lot of time to make it work. And it may take a week, or two weeks longer depending on the environment it uses."

3.3.4 What They Like or Dislike about TensorBoard. Seven of the ten participants did not like TensorBoard. They pointed out two main reasons. First, a lot of critical information they wished to know about a neural network was not displayed in TensorFlow. For example, P3, P4, and P6 all expected to see the task and dataset information to assess the relevance of an example model to their own goal. Second, the visualization in TensorBoard shows many low-level operations that participants did not care about. P3 mentioned that *"even some low-level operations such as addition and matrix multiplication are represented in the graph."* On the other hand, the other three participants liked TensorBoard, since it shows the high-level structure of a neural network, such as layers and activation functions. P9 said, *"the flow is clear, and the structure is very important to me. Compared with reading through thousand lines of codes, this is much better."* P3 also considered TensorBoard helpful since *"it distinguishes layers and functions in different colors and blocks, making it easy for people to understand."*

4 DESIGN PRINCIPLES & SYSTEM OVERVIEW

4.1 Design Principles

We summarized three design principles for a system that supports learning and designing neural networks, based on the information needs of deep learning programmers identified in the formative study and the Variation Theory [16]:

D1. Help users understand the relevance to their own tasks. From the formative study, the information needs (***N1, N5, N7***) indicate that DL programmers only care about projects that have similar tasks and datasets to their own. For example, ***N7*** represents the user's need to understand whether a neural network example is related to the task they are facing. Furthermore, ***N1*** and ***N5*** both indicate users are only willing to learn more about a neural network example when they believe that the task to which the given example belongs is highly relevant. Therefore, our system needs to provide a way to help users quickly understand whether a project is relevant.

D2. Help users distill high-level design decisions. N2, N3, N5, and ***N6*** indicate that DL programmers want to understand high-level design decisions such as model structures and hyperparameters rather than low-level implementation details. In ***N2***, users want to know the information about model structures instead of the implementation of models. And ***N3, N5, N6*** are the needs of users who want to know more about model types, hyperparameter settings, etc. respectively. Therefore, our system needs to help users easily perceive these high-level design decisions from the low-level code in deep learning projects.

D3. Help users understand the commonalities and variations of design choices. N4, N5, N6 all indicate that DL programmers want to understand the common hyperparameters and model structures used for similar same tasks or datasets. Furthermore, ***N1, N3*** indicate that users also want to find the variations in neural network design. For example, some users want to know alternative model types that handle similar tasks, and some users want to know different tricks used by different developers. Therefore, our system needs to support exploring both common and uncommon design decisions in neural network design.

4.2 System Overview

Based on the three design principles, we implemented an interactive visualization system called ExampleNet that helps programmers explore various neural network design descions in a corpus of DNN models. It contains three main features:

4.2.1 Faceted Browser. In the faceted browser view (① in Figure 1), each facet displays the names of different datasets, tasks, and model types. Through this faceted browser, users can quickly select and filter the corpus of DNN models based on their own needs. The distribution bar next to each facet shows the number of models corresponding to each facet under different selection conditions. Therefore, users can directly read the length of each bar to understand how frequent or infrequent each option is, given their prior selections. In addition, the faceted browser also renders quality-related metrics such as project stars and forks. This allows users to filter models based on these proxies for quality.

4.2.2 An Overview Diagram of Model Structures. The overview diagram of model structures (③ in Figure 1) shows the large collection of networks at scale. Since the structure of a neural network describes the order in which data flows between layers, we follow the Sankey diagram design to aggregate the structures of various models in a single view. In our Sankey diagram, each flow represents one or more models, and each node in the flow represents a layer. Models are aligned based on the type and ordering of their layers. Model layers with the same type in the same position are merged

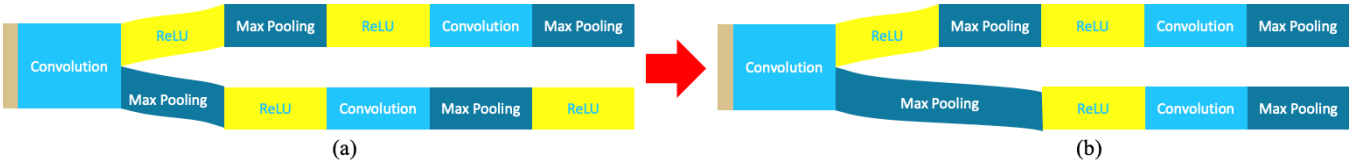


Figure 2: Align the structures of two models based on layer types

into a single flow. For example, in Figure 2(a), the first layers of two models are both convolution layers, so these two convolution layers are merged into a joint layer. In this way, users can compare and contrast the commonalities and variations of layer choices and arrangements among multiple models in a single view.

The traditional Sankey diagram design often produces a diagram with many overlaps among flows when visualizing many neural network models. The bifurcation and convergence of flows can easily cause confusion. Therefore, we decided to adapt the traditional design to only contain the bifurcation structure without no convergence. In Figure 2(a), our design renders the subsequent layers of two models as they are different. However, the traditional Sankey diagram design will maximally merge these subsequent layers, making the two flows intertwined with each other.

On the other hand, our adaption makes it difficult to identify common subsequences of layers in relatively different positions across models. In Figure 2(a), after the first joint convolution layer, the subsequent layers of the two models vary only by one layer: the second layer in the first model is ReLU, while the second layer of the second model is Max Pooling. Though the second layers are different, the following layers after the second layer are quite similar. As the subsequent layers are visualized in separate flows, it is difficult for users to mentally align them and compare their similarities. Hence, we introduced a local alignment feature to align layers with the same type in relatively similar positions. Specifically, we used the Smith-Waterman algorithm [18]. This algorithm first determines the substitution matrix and the gap penalty scheme and, from that, constructs a scoring matrix. Finally, it traces back this scoring matrix based on the source of each score recursively to generate the best local alignment. Figure 2(b) shows the aligned diagram of Figure 2(a). Based on the local alignment, the second layer (Max Pooling) of the second model is extended, so its subsequent layers are aligned with the same subsequence of layers in the first model.

4.2.3 The Summative Distribution of Layer Types and Hyperparameters. Several summative charts show the distributions of layer types (② in Figure 1) and hyperparameter values (④ in Figure 1). The summative chart of layer types renders layer types in six categories: *CNN*, *RNN*, *DNN*, *Activation Function*, *Loss Function*, and *Others*. Table 2 shows the variety of layer types that can be recognized by our system. We also used the same color scheme as the overview diagram of model structures (③ in Figure 1).

The hyperparameter charts (④ in Figure 1) show the distributions of different hyperparameter values and layer parameter values. Our system recognizes and renders 9 hyperparameters, including learning rate, batch size, epochs, optimizer, momentum, decay rate, dropout rate, number of hidden layers, and number of hidden layer units. Unlike the summative chart of layer types, we use bubble charts to render the distribution of these hyperparameters. The x-axis represents the value of hyperparameters or parameters. The y-axis and the radius of each circle indicate how many models have set this value as a hyperparameter or parameter. The reason why we used different design elements to visualize the same information is that in some hyperparameters and parameters, many values will be concentrated in a small range. For example, many models set the learning rate to less than 0.01. If we only use the size of the circle to represent the number of models that use this value, there will be many overlapping data points. When a user hovers over a circle, the chart displays the number of models that use this value.

Category	Recognized Layer Types
CNN	Convolution, Deconvolution, Max Pooling, Average Pooling
RNN	LSTM, GRU, BiRNN, RNN, CRF, Attention
DNN	Input, Dense, Dropout, Flatten
Other	Embedding, Normalization
Activation Function	Argmax, ReLU, Sigmoid, Softmax, Linear, tanh
Loss Function	Cross Entropy, Reduce Mean, CTC, L2, MSE

Table 2: The neural network layer types that are recognized and rendered by our current implementation.

5 USAGE SCENARIO

Suppose Sam is a beginner in deep learning. He wants to implement a deep learning model for text classification. Without using ExampleNet, Sam searches with the keyword "text classification" on GitHub. The GitHub search engine returns 8,899 related projects. Sam ranks those projects by their stars. Then he opens the project that received the most stars and reads its README file to understand basic information about the project.

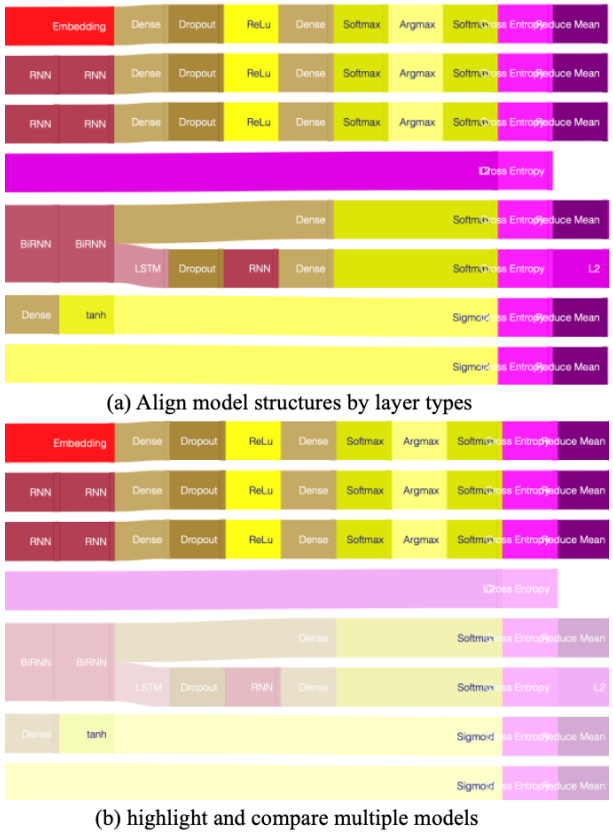
Sam wants to understand the structure of the neural network in this project, so he looks for the source code that implements it. Since he is a beginner in deep learning, he struggles to glean what he needs from the hundreds of lines of code. Even after attempting to inspect each line of code, Sam finds it hard to piece together all the details related to the neural network.

Sam also wants to go over several other projects to identify which is the most suitable project to follow for his text classification problem, or at least find out which design choices are common or atypical. However, he cannot manage to hold all the neural networks in his memory to mentally compare and contrast them. He does not even attempt to remember multiple models' lower-level details such as the parameter settings of each layer type.

With the help of ExampleNet's interactive visualizations, Sam navigates through all the relevant projects in a corpus collected from

GitHub and establishes a holistic mental model of the design choices made by other DL programmers. In the Task facet of the *faceted browser* (① in Figure 1), Sam quickly finds the text classification task and selects it. The interface updates to show him the distribution of different datasets and models used in those text-classification GitHub projects: specifically, the bars in the light color still show the number of projects in the original collection, while bars in the dark color show the *conditional distribution*—the number of projects after filtering with Sam's selections, which he may continue to make to hone in on the subset of Github projects that will collectively become his reference point for making his own design decisions. By looking at the conditional distribution in the facet of model types (Figure 1 ①), Sam finds that the length of the dark blue bar for the RNN model is the longest, which means that the majority of projects implement RNN models for text classification. Thus he also decides to design an RNN for his task.

Sam wants to filter out lower-quality projects. He believes projects' popularity, in terms of stars and forks, implies their reliability, so he drags the left border of the brush box in the *Project Stars* facet (① in Figure 1) to exclude projects with less than 3,000 stars. The histogram bars in other facets are updated accordingly. Now, Sam finds that the number of models has been reduced to eight. Sam sets a similar threshold using the *Project Forks* facet. Finally, Sam sees in the overview diagram of model structures (③ in Figure 1) that some projects have too many layers, which are challenging to train with the limited computational power of his own machine. He uses the *Number of Layers in Projects* facet to sets a threshold to a maximum of 20 layers per project. Seven projects remain.



(a) Align model structures by layer types

(b) highlight and compare multiple models

Figure 3: A user aligns models by their layer types and then compares multiple models.

5.1 Exploring the Design Space of Model Structures

The *layer type* histogram (② in Figure 1) shows the distribution of different layers in the remaining projects. Sam finds that the majority of projects (four of seven) use LSTM layers. While the majority of neural networks (four out of seven) use LSTM layers, only one model uses a GRU layer and two models use BiRNN layers. Sam realizes that GRU and BiRNN could be alternatives to LSTM layers. He also notices that all seven projects use Dropout layers, four use Embedding layers, and one uses Normalization layers. This is surprising to Sam, since he was not particularly aware and attentive to Embedding layers before.

Sam is familiar with Normalization and Dropout layers, but he is not entirely sure how and where to use them. To figure it out, Sam turns to the *overview diagram* (③ in Figure 1). He notices that five of the seven models have a Dropout layer in the middle of the network, and two place them at the end of the network. He also notices a pattern of placing a Dropout layer right after an LSTM layer, and that the Embedding layers are always placed as the first layer of a neural network.

Sam clicks the *alignment view* button to re-align the neural network structures in the overview diagram based on layer types. Figure 3(a) shows the re-aligned neural networks. In this view, Sam immediately notices the *Dense, ReLU, Dense, ReLU, ...* pattern. He also finds that four projects use the alternative *GRU/BiRNN, Dropout, GRU/BiRNN,...* pattern. Now Sam is more confident about the patterns he found earlier. When Sam looks at the end of these models, he finds all seven networks use *Cross Entropy* as their loss functions, which suggests to Sam that *Cross Entropy* might be a proper loss function choice for text classification.

5.2 Exploring the Design Space of Hyperparameters

In deep learning, hyperparameters are critical to the performance of the networks. Prior work shows that embeddings, optimizers, and dropouts have a high impact on model accuracy and training time [19]. The *hyperparameter view* allows Sam to explore what common and uncommon hyperparameters are used by other developers. In Figure 1 ④, Sam notices five hyperparameters are listed: learning rate, batch size, epochs, dropout rate, and optimizer. He did not even know about the decay rate and dropout rate before. For instance, on the learning rate chart, there are four projects that choose the learning rate as 0.001 and three projects use 0.0003 (④ in Figure 1). The distribution of various learning rates used by other programmers augments Sam's knowledge of an appropriate learning rate range. Without seeing the distribution of learning rates, Sam might have picked a learning rate that is too large or too small, costing him extra tuning iterations to find the optimal value.

6 DATASET CURATION

To fuel the visualization with real-world model data, we developed a semi-automated data curation process that extracts model characteristics from GitHub projects. Figure 4 shows the pipeline. In this work, we focused on models implemented in TensorFlow. To collect deep learning projects, we searched "deep learning" and "neural network" on GitHub and downloaded the top-starred projects in the search result. Then we wrote a script to automatically scan the source code files in those projects and discarded projects that do not import python packages from TensorFlow. Given a GitHub TensorFlow project, we took two major steps to pre-process the source code and then extract information cues mentioned in the formative study. The extracted information cues were then manually validated to ensure data consistency and quality.

6.1 Extracting Model Structures via Program Analysis

We implemented a light-weight program analyzer to identify model structures from a DL project written in TensorFlow. First, for each project, the program analyzer scanned the source code and identified imported TensorFlow APIs. Since many functions in different Tensorflow packages have the same name, the package important information was later used to resolve ambiguous function calls in the source code. Then, the analyzer conducted a whole-program call graph analysis to build the call graphs from all source code in the project. In the whole-program analysis, the analyzer parsed all source code to abstract syntax trees (ASTs) and traversed the AST of each source code file to build individual call graphs of that file. Then it connected the individual call graphs to build a bigger call graph based on function calls between files. We manually created a mapping from Tensorflow API calls to neural network layers and layer parameters. Table 2 shows all layer types that are recognized in this analysis. For example, the API method `tf.nn.conv2d` corresponds to the *convolution layer*, and `tf.keras.losses.CategoricalCrossentropy` corresponds to the *cross-entropy loss function*. Similarly, we manually created another mapping for hyperparameters. We defined a list of possible variable names for each hyperparameter, based on which the analyzer identified variables that may hold hyperparameter values via fuzz matching.

Given a call graph, the analyzer filtered out function calls that were not included in the pre-defined mapping. The remaining function calls constituted the essence of the model structure. Compared with manually reading source code and identifying model structures, this light-weight process significantly reduced the data curation effort. However, it was not precise enough and thus required manual validation. We discuss several cases that required attention from human validators.

First, some projects may use loops to repeatedly add layers to a neural network. During program analysis, if a TensorFlow API call appears in a `for` loop, the analyzer will automatically log a warning message for manual validation. The human validator then manually inspects the `loop` bound to determine how many times a layer is added to the neural network.

Second, if a TensorFlow API call appeared in an `if`-else branch, the program analyzer will also log a warning message. The human validator then manually assesses the conditional expression and decides whether to add the layer to the model structure. If taking the `if` branch and else branch could lead to two different model structures, the human validator will manually create separate models to reflect this. For example,

```
if cell_name == 'LSTM':
    cell = tf.keras.layers.LSTM(units)
else:
    cell = tf.keras.layers.GRU(units)
```

In this code, the model can have two structures—one with LSTM units and the other with GRUs. In such a case, the human validator manually one RNN with LSTMS and the other with GRUs.

Third, the fuzz matching method may fail to recognize some hyperparameter values, since GitHub developers may assign obscure variable names to their hyperparameters. Sometimes, hyperparameters may be set in a configuration file rather than being hardcoded in the source code. When a hyperparameter is missing in the analysis result, the human validator must manually go through the source code files and identify the hyperparameter values.

Finally, we discarded models that have multi-granularity architectures. For example, in *ResNet* [8], the residual layer will have two outputs. The first output is directly linked to the next layer, and the second output will skip one or more layers. Currently, our system does not support visualization of such multi-granularity architectures.

6.2 Identifying DL Tasks and Datasets from GitHub Projects

In addition to model structures and hyperparameters, we manually identified the training datasets, tasks, and model names from each project. We read through the README of each project to identify dataset names. If the dataset was not mentioned in the README, we went through the project's data preprocessing code to identify the training data. We manually classified identified datasets into six categories—image, text, video, audio, tabular, and others. Similarly, we identified the computation tasks such as image classification and sentiment analysis from the README. Regarding model names, we first attempted to look for specific names mentioned in the README, such as VGG and AlexNet. If no specific model names were found, we manually assigned a general name such as CNN or LSTM based on the model structures identified in the previous step.

In this work, we downloaded 203 GitHub projects.We ran our static program analyzer on those 203 projects and found that the analyzer failed to extract any model structures or hyperparameters from 86 projects. This is because these 86 projects were outside the scope of our program analyzer's capabilities: (a) 82 projects were not written in TensorFlow and (b) 4 projects used pre-trained models that our analyzer could not extract relevant data from. Then we manually went through the README files of the remaining 117 projects. 24 projects were discarded in this step since their README files did not contain information about their training datasets, tasks, and model names. Among the remaining 93 projects, 31 (33%) were eliminated because they included structures outside the scope of our current visualization algorithms: (a) 7 projects included residual connections (ResNets), 12 included multiple branches (Inception networks), 9 included GANs (each contains two networks, a generator, and a discriminator respectively) and 3 included other unsupported

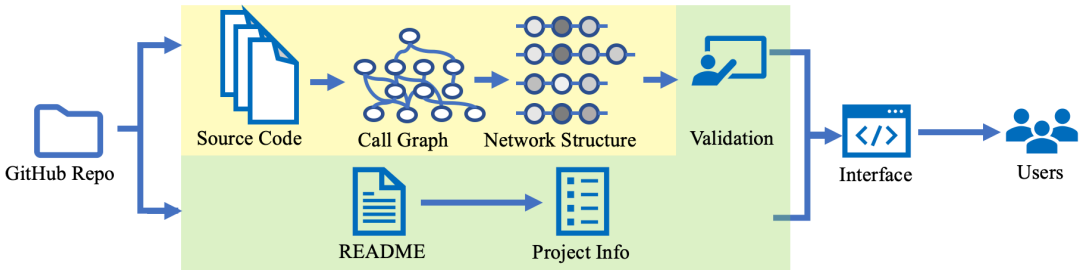


Figure 4: The semi-automated data curation pipeline of ExampleNet. Steps in light yellow are automated, while steps in light green are manual.

models such as HMM. After the analysis and manual validation steps, we extracted complete model structures, hyperparameters, and other meta-data for 62 models. These 62 models form the collection of neural network examples used in the following user study of ExampleNet. The semi-automated process took roughly 20 man-hours. The most time-consuming part is to manually go through GitHub README files to identify DNN related meta-data, which took about 6 minutes per project.

7 USER STUDY

We conducted a within-subjects study with sixteen participants to evaluate whether ExampleNet could help them more effectively develop an awareness of design choices available to them when designing a deep neural network. Online search, a common practice in neural network design as indicated by our formative study (Section 3), is used as the realistic baseline in a control condition.

7.1 Participants

We recruited sixteen master students in Computer Science and Engineering or Data Science at Harvard University. Participants received a $25 Amazon gift card as compensation for their time. Participants had diverse expertise in deep learning. Three of them had between two and five years of deep learning experience, nine participants had only one year of experience, and the remaining four had just one-semester's experience.

Participants self-reported their familiarity with the two deep learning domains, CV and NLP, on a 6-point Likert scale. For computer vision, eight participants rated themselves as beginners (0-1 on the Likert scale), seven rated themselves as only somewhat familiar (2-3 on the Likert scale), and only one participant rated themselves as familiar (4 point on the Likert scale). For NLP, the majority of participants (10 out of 16) considered themselves beginners, five rated themselves as somewhat familiar, and one rated himself as familiar.

7.2 Protocol

We selected two common deep learning tasks from CV and NLP:

- *Task 1 (Image Classification)*: Design a neural network and find reasonable hyperparameter settings to classify 10K 128× 128 animal images into dog, cat, or others.
- *Task 2 (Text Classification)*: Design a neural network and find reasonable hyperparameter settings to classify 10K English conversations into weather, animals, environment protection, or others.

For each task, participants were asked to answer two questions related to neural network architecture and hyperparameter design:

- *Q1 (Network Structure Design)*: Draw the neural network and specify the number of layers and the type of each layer.
- *Q2 (Hyperparameter Design)*: Set the values of four critical hyperparameters, including learning rate, batch size, epochs, and optimizer. In addition, provide the name and suitable value of any other hyperparameters that may help optimize your model.

Each study took about 70 minutes. Each participant was given 20 minutes to finish each task. In the control condition, participants were allowed to use any search engines they were comfortable with to find online resources, e.g., blogs, tutorials, StackOverflow posts, to answer Q1 and Q2 for the assigned task. In the experiment condition, participants were only allowed to use ExampleNet without any access to other online resources to answer Q1 and Q2 for the assigned task. Given that this was a within-subjects study, each participant experienced one task in one condition and the other task in the other condition. To mitigate any learning effects, both the order of the assigned tasks and conditions were counterbalanced across participants. Before the task with ExampleNet, participants were asked to first watch an 8-min tutorial video of ExampleNet and then spend 5 minutes familiarizing themselves with the interface After each task, the participants were asked to complete a questionnaire to record their reflections on their experience in the assigned condition. As part of the post-study survey, participants were asked to answer five NASA Task Load Index questions [7] to rate the cognitive load of the assigned task. After finishing both tasks, participants were asked to fill out another survey to directly compare the Online Search and ExampleNet conditions. We recorded each user study session with the permission of participants.

8 USER STUDY RESULTS

8.1 User Performance

Participants using ExampleNet made significantly different design choices compared with using online search. When using ExampleNet, participants designed deeper neural networks (median of 13 vs. 9.5) with more diverse layer types (median of 7 vs. 5) than using online search. Though deeper neural networks do not mean

	# of Layers			# of Layer Types			# of Hyperparameters			# of Inspected Examples			# of Design Mistakes		
	Min	Median	Max	Min	Median	Max	Min	Median	Max	Min	Median	Max	Min	Median	Max
Online Search	4	9.5	21	3	5	7	4	4.5	5	0	2	5	0	1	4
ExampleNet	7	13	30	6	7	9	4	5	10	5	6	12	0	0	1

Table 3: Statistics about number of layers, number of layer types, number of hyperparameters, number of looked examples, and number of mistakes when using online search and ExampleNet.

The Mistakes Participants Made	Participants (Online Search)	Participants (ExampleNet)
Missing Activation Function	P4, P6, P8, P9, P10, P11, P12, P14	None
Huge Epochs	P3, P4, P5, P7, P8, P15	P3, P5
Missing Loss Function	P1, P2, P3, P8, P10	None
Missing Dropout Rate	P1, P4, P8, P9	None
Missing Dense Layer	P13, P14	P3
Huge Learning Rate	P3	None
Incorrect Layer Sequencing Order	P12	None

Table 4: The mistakes participants made when using online search and ExampleNet.

more superior models, being able to see the distribution of different neural networks indeed brings programmers more awareness of various design choices such as different types of layers to leverage in their own models. For example, 8 out of 16 participants used Dropout layers in their text classification model when using ExampleNet, while only 3 of 16 participants used it when using online search. In general, adding Dropout layers can improve generalization performance on text classification tasks [25].

The differences are also clear for hyperparameter design: compared with using online search, participants using ExampleNet set more hyperparameters such as dropout rate, decay rate, and momentum. Three participants set a *decay rate* when using ExampleNet, while no participants set one when using online search. By setting a decay rate, the model will start with a large learning rate and then let it decay by the proscribed amount over the course of training. A larger initial learning rate can accelerate training and help the model escape local minimal. Decaying the learning rate can help the model converge to a minimum and avoid oscillation [31].

We manually analyzed the user study recordings and counted the number of online examples each participant inspected during the study. In the online search condition, we defined example inspection as clicking into a search result. In the ExampleNet condition, we defined example inspection when they thought out loud about a model or when they clicked into the GitHub repository page of a model from ExampleNet. As shown in Table 3, when using online search, participants only inspected two online examples on the median. Some of them even designed the neural network based on their own experience, without looking at a single example. As several participants pointed out in the post-study survey, navigating through online examples and identifying the essence from each example is time-consuming and cumbersome. By contrast, when using ExampleNet, participants inspected a median of six GitHub examples. With the faceted browser in ExampleNet, participants quickly filtered the large collection of GitHub examples and retrieved the ones that were most relevant to their task and dataset. The overview diagram provided them a bird's-eye view, enabling them to simultaneously compare and contrast multiple model structures. However, with online search, participants had to click into each search result, identified the model in it, and went back and forth to compare them. This was time-consuming and cumbersome. P14 explained this contrast between using ExampleNet and online search in the following way, *"[ExampleNet] provides you several reasonable filtering conditions and clear comparisons of structures of different well-known models. ... [When searching online,] I was easily overwhelmed when facing a massive amount of information from the internet. And I didn't know which one to start from."*

Table 3 quantifies the model design differences in terms of the number of layers, the types of layers used in a model, the number of hyperparameters, the number of inspected examples, and the number of design mistakes.

We manually assessed the models designed by each participant. Table 4 shows the distribution of different kinds of design mistakes made by participants. The most common mistake is *"Missing Activation Function"* (8/16). If the model does not contain an activation function, it will be a linear model. The complexity of the linear model is limited, the robustness is reduced, and the ability to learn the complex functional mapping from the data is weaker. Some design mistakes may lead to more severe consequences. For example, missing loss functions and incorrect layer sequencing order will cause runtime errors. As another example, missing the dense layer will cause the model to be unable to convert the dimensions of the output from the convolution and pooling layers into the output space corresponding to the classification task. Therefore, the model cannot calculate the loss and thus cannot perform backpropagation to update the weight of each parameter in the model.

We found that ExampleNet has a statistically significant impact on assisting participants to reduce mistakes (Table 4) in neural network design and hyperparameter design. On average, each participant made a median of 0.19 mistakes in both tasks. In contrast, when using online search, they made an average of 2.06 mistakes in each task. The mean difference of mistakes (1.88) is statistically significant (paired t-test: $t = 4.5281, df = 30, p - value < 0.00001$). The comparison between online search and ExampleNet indicates

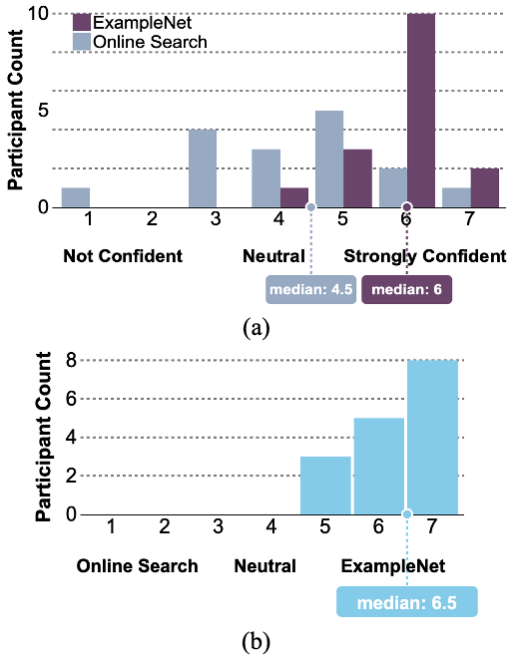


Figure 5: (a) when using ExampleNet, participants felt more confident in the networks they designed, (b) participants thought that ExampleNet is more helpful than online search when they searched and designed neural networks.

Figure 6: Participants felt less mental demand, hurry, and frustration when using ExampleNet to complete two tasks. They also believed themselves have a better performance when using ExampleNet.

that providing more examples and giving the holistic view of these examples could significantly decrease the rate of mistakes when designing deep learning models and choosing hyperparameters.

8.2 Impact of User Expertise

While not found to be statistically significant, there is little discernable difference between the ExampleNet's impact on the mistake rate of participants claiming no familiarity (1 on a 6-point Likert scale) with an assigned task (CV or NLP) and the mistake rate of participants claiming at least some familiarity (2 or higher on a 6-point Likert scale). Both the task-unfamiliar participants using ExampleNet and the task-at-least-somewhat-familiar participants made at most one mistake (min: 0, median: 0, max: 1) compared to task-unfamiliar participants using online search (min: 1, median: 1.5, max: 4) and task-at-least-somewhat-familiar participants (min: 0, median: 1, max: 3).

8.3 User Confidence and Cognitive Load

In the post-study survey, participants reported having more confidence in their neural networks when using ExampleNet. Figure 5 shows the distribution of their confidence ratings on a 7-point Likert scale. The median confidence when using ExampleNet is 1.5 points higher than when using online search. In addition, all sixteen participants found ExampleNet more helpful than online search. These results imply that rendering the commonalities and variations of a large collection of examples is more useful than overwhelming. P1 said *"when I construct the model through online resources, I usually only select the first model that makes sense as my starting point. ExampleNet lists a comparison between models, so I am more confident in the model that I selected."* P4 said, *"it [ExampleNet] helps me find the most useful information quickly and I could compare it to different models. I can clearly see the network designs and parameter settings."* With the help of a faceted browser, participants could easily filter the corpus of deep learning projects based on their own tasks or datasets. P10 said *"many times I don't know if the information I got from online search is accurate or not, or if they are tailored to my project. ExampleNet aggregates well-developed model architectures into one place. This greatly helped me make well-informed decisions when choosing model architectures."*

As shown in Figure 6, participants felt more mental demand, hurry, and frustration when using online search than using ExampleNet. This is mainly because ExampleNet directly shows a holistic summary of related models, so participants no longer need to read hundreds of lines of codes and tutorials. P3 mentioned, *"ExampleNet gives a comprehensive summary of every aspect in networks, such as hyperparameter values and depth of networks. Most online articles do not talk about the hyperparameter value choices in the model, but just the model introduction, which made me confused."*

8.4 Qualitative Feedback

Most participants (13/16) rated the overview diagram of model structures as the most useful feature in ExampleNet. They believed that this overview diagram provides a clear and comparable way for users to discover commonalities and variations through a large collection of network structures. P7 said *"the visualization of model architectures is quite informative in showing what are some common architectures shared across different projects, while also showing how each network differs from one another."* 11 out of 16 participants liked the histogram that shows the distribution of the number of models for each layer used (6 or 7 on a 7-point scale). The function of aligning different model structures received a mixed feeling from our participants—half of the participants (8/16) found it useful while the other half stayed neutral about it. Other features, such as bubble charts showing the distribution of different hyperparameters (7/16), providing the link to each GitHub repository (3/16), clear and easy-to-use interface (7/16), and the ability to sort examples by stars and forks (P7), also received a lot of positive reviews. Regarding the feature that shows the distribution of different hyperparameters,

P3 mentioned *"it gives the recommendations of hyperparameter values which helped me a lot since I didn't have experience in tuning parameters of NLP."* P7 believed that allowing users to sort models by stars and forks is an advantage of ExampleNet: *"the ability to sort projects by stars and then highlight the relevant network is also quite helpful and can get beginners up to speed in a short amount of time."*

Regarding online search, four participants pointed out that searching based on keywords only could not accurately find the desired results. P1 said, *"sometimes I don't get what I want straightforwardly, like when I want to find a good initial value for the learning rate, the website talks about the advantages and disadvantages of having it too large or too small. I also don't know what keywords I should put in this part."* P2 also complained, *"when I searched for 'optimizer', it [GitHub Search] gave me many results about 'optimization', which is completely different."* P7 said, *"the problem is, it is hard to tell which website gives the answer I am looking for and the quality of each website is not guaranteed."* Most participants (10/16) complained that the large amount of information provided by online search is time-consuming and also overwhelming. P6 said, *"there is too much information when I used online search, kind of hard to locate exactly what I want. Also, too much information could be distracting and very time-consuming."*

They also made some suggestions to improve ExampleNet. Four participants wanted ExampleNet to add more details about the dataset size and the input and output types of a model. Two participants suggested to improve the color scheme in ExampleNet. One participant suggested to add the last update time of each GitHub project, since he trusted actively maintained projects more. Two participants wished ExampleNet could automatically generate a model based on their high-level design choices.

In the last part of the post-survey, we asked how ExampleNet may fit into their programming workflow. Four out of sixteen participants wrote that they wanted to use ExampleNet when facing unfamiliar tasks, data sets, and models. P1 said, *"I will use* ExampleNet *when I am unfamiliar with the model I am gonna use. ... I am very unfamiliar with NLP and hyper-parameter selection for models, [so] I would come to* ExampleNet *for this part. It's a better place to start, especially when I am not familiar with some layers that I might use, it's good to see their positions in the model first and get a rough idea what I can expect to see in my final model."* Five participants wished to use ExampleNet when they needed to quickly find suitable and popular models.

9 DISCUSSION

The user study results suggest that ExampleNet helps deep learning programmers browse more examples, make more data-driven design decisions, and make fewer mistakes. We believe these differences are, in part, a result of ExampleNet's support on task-centered compare and contrast cognition, as well as norm-setting through showing the distribution of common and uncommon choices made by other programmers, from model structures to layer parameter values. We also believe ExampleNet answers many of the concrete questions and information needs identified in the formative study.

Specifically, the multifaceted browser provides a convenient way to filter GitHub projects based on the processed datasets, tasks, and model names. As informed by the formative study, programmers only care about those models that process similar datasets and tasks as their own (Section 3.3.2). Without the faceted browser, programmers need to dig into each project and figure out whether the project is relevant or not. The Sankey diagram and the hyperparameter charts give an overview of different layer types and hyperparameters in relevant DL models after filtering. Many user study participants started with very vague ideas such as "I want to use a CNN" and "a CNN should have a convolution layer." Yet they did not know exactly what other layers they should consider including, possible orders of layers, what optimizers to consider, etc. By looking at the Sankey diagram and the hyperparameter charts, they could quickly answer those questions. As shown in Table 4, when only looking at online tutorials without such an overview in ExampleNet, participants made more design mistakes.

ExampleNet does not remove the need for tuning model structures and hyperparameters in the training phase. Based on the training result, programmers still need to adjust some hyperparameter values and experiment with alternative hyperparameters to arrive at an optimal model. The lab study suggests that ExampleNet may give users a *better starting point* for their own iterative neural network design process and hyperparameter tuning, which may be critical to quickly getting reasonable results from a newly-designed neural network.

Since designing and tuning deep learning models requires much human expertise, the Machine Learning community has proposed a series of automated machine learning (AutoML) techniques that automatically build DL models without human assistance [9]. Though AutoML is a promising solution for reducing the manual effort and adoption barrier of deep learning, many programmers may still want to learn how to design neural network models themselves, or at least be aware of possible design choices so that they can better assess models synthesized on their behalf by tools like AutoML. Exploratory tools like ExampleNet will still be useful in such cases. In addition, the designers of AutoML techniques can also use tools like ExampleNet to discover what kinds of models, hyperparameters, and other ML tricks have been developed and used in the wild and further incorporate this variety of design options into their AutoML algorithms.

10 LIMITATIONS AND FUTURE WORK

Our current system design and implementation has several limitations, which remain to be addressed in future work. As described in Section 4, each flow in the modified Sankey diagram can only represent layers in a model sequentially, so ExampleNet is unable to support visualizing networks with branch-like structures. Two major extensions are needed to support networks with more complex structures. First, the static analyzer needs to be extended to recognize related APIs such as "tf.concat" in program analysis to handle residual connections and multiple branches. The Sankey diagram needs to be replaced with other types of visualizations such as Union Graphs [2] to aggregate models with non-sequential network structures.

Currently, our static program analyzer only extracts model structures and hyperparameters from models built by TensorFlow. It can

be extended to other DL frameworks such as PyTorch by supplementing two pre-defined mappings in JSON—one mapping between library API methods and layer types and another mapping between API method parameters and hyperparameters. Apart from the limitations on the API, supporting models written in programming languages other than Python would require swapping in a different language parser and updating the downstream AST-traversing code. Supporting additional deep learning frameworks such as PyTorch, additional layer types, and hyperparameters, will likely be a matter of engineering rather than additional novel system design.

As described in Section 6, the data curation process is only semi-automated. The major limitation is that we have to manually skim through the README file of a GitHub repository to identify DNN related meta-data, including dataset types, tasks, and model names. It took us about 6 minutes per project. The manual effort will increase linearly as the number of projects increases. This can be mitigated in two ways. First, future Github users could be encouraged to explicitly encode those DNN related meta-data in their README files, much like some medical publication sites invite authors to submit their papers with explicitly described PICO elements [21]. This can make those meta-data more readily available for search and analysis. Second, using keyword matching or some NLP methods can to some extent reduce the manual effort; manual validation is still necessary as automated methods may not always be accurate.

In our user study, though our participants are new to deep learning, they are not new to programming. All 16 participants are graduate students; 13 of them have over one year of working experience as data scientists or software engineers. 14 of them have 2 to 5 years of programming experience. Therefore, they do not represent those deep learning learners who are new to both deep learning and programming.

11 CONCLUSION

This paper presents a novel interactive visualization interface that allows users to (1) simultaneously explore design choices in a large number of deep learning projects, and (2) compare and contrast the common and uncommon design choices. We conducted a within-subjects study with sixteen deep learning programmers to evaluate ExampleNet. The study results show that when using ExampleNet, participants inspected more neural network examples than online search. After inspecting the network design and hyperparameters setting in these examples, participants using ExampleNet made more data-driven design decisions, such as picking a more reasonable learning rate as a starting point, using dropout and normalization to build more robust models. In addition, using ExampleNet, participants made significantly fewer design mistakes, e.g. missing activation functions, missing loss functions, incorrect layer orders, etc. In the end, we discussed the possibility of fully automating the data curation pipeline, supporting more complex model architectures, and surfacing more information cues such as dataset size, model accuracy, and training time.

REFERENCES

[1] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 291–300.

[2] Keith Andrews, Martin Wohlfahrt, and Gerhard Wurzinger. 2009. Visual graph comparison. In *2009 13th International Conference Information Visualisation*. IEEE, 62–67.

[3] Joel Brandt, Philip J Guo, Joel Lewenstein, Mira Dontcheva, and Scott R Klemmer. 2009. Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1589–1598.

[4] Carrie J Cai and Philip J Guo. 2019. Software Developers Learning Machine Learning: Motivations, Hurdles, and Desires. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 25–34.

[5] Graham Dove, Kim Halskov, Jodi Forlizzi, and John Zimmerman. 2017. UX design innovation: Challenges for working with machine learning as a design material. In *Proceedings of the 2017 chi conference on human factors in computing systems*. 278–288.

[6] Elena L Glassman, Tianyi Zhang, Björn Hartmann, and Miryung Kim. 2018. Visualizing API usage examples at scale. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–12.

[7] Sandra G Hart and Lowell E Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Advances in psychology*. Vol. 52. Elsevier, 139–183.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[9] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2019. AutoML: A Survey of the State-of-the-Art. *arXiv preprint arXiv:1908.00709* (2019).

[10] Andrew Head, Elena L Glassman, Björn Hartmann, and Marti A Hearst. 2018. Interactive extraction of examples from existing code. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–12.

[11] Enamul Hoque and Maneesh Agrawala. 2019. Searching the Visual Style and Structure of D3 Visualizations. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (2019), 1236–1245.

[12] Nicholas Kong, Tovi Grossman, Björn Hartmann, Maneesh Agrawala, and George Fitzmaurice. 2012. Delta: a tool for representing and comparing workflows. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1027–1036.

[13] Ranjitha Kumar, Arvind Satyanarayan, Cesar Torres, Maxine Lim, Salman Ahmad, Scott R Klemmer, and Jerry O Talton. 2013. Webzeitgeist: design mining the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 3083–3092.

[14] Brian Lee, Savil Srivastava, Ranjitha Kumar, Ronen Brafman, and Scott R Klemmer. 2010. Designing with interactive example galleries. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 2257–2266.

[15] Howard Lune and Bruce L Berg. 2016. *Qualitative research methods for the social sciences*. Pearson Higher Ed.

[16] Ference Marton. 2014. *Necessary conditions of learning*. Routledge.

[17] Kayur Patel, James Fogarty, James A Landay, and Beverly Harrison. 2008. Investigating statistical machine learning as a tool for software development. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 667–676.

[18] William R Pearson. 1991. Searching protein sequence libraries: comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms. *Genomics* 11, 3 (1991), 635–650.

[19] Nils Reimers and Iryna Gurevych. 2017. Optimal hyperparameters for deep lstm-networks for sequence labeling tasks. *arXiv preprint arXiv:1707.06799* (2017).

[20] Caitlin Sadowski, Kathryn T Stolee, and Sebastian Elbaum. 2015. How developers search for code: a case study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 191–201.

[21] Connie Schardt, Martha B Adams, Thomas Owens, Sheri Keitz, and Paul Fontelo. 2007. Utilization of the PICO framework to improve searching PubMed for clinical questions. *BMC medical informatics and decision making* 7, 1 (2007), 16.

[22] Susan Elliott Sim, Medha Umarji, Sukanya Ratanotayanon, and Cristina V Lopes. 2011. How well do search engines support code retrieval on the web? *ACM Transactions on Software Engineering and Methodology (TOSEM)* 21, 1 (2011), 4.

[23] Daniel Smilkov, Shan Carter, D Sculley, Fernanda B Viégas, and Martin Wattenberg. 2017. Direct-manipulation visualization of deep networks. *arXiv preprint arXiv:1708.03788* (2017).

[24] SONY. 2019. Neural Network Console. https://dl.sony.com.

[25] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15, 56 (2014), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html

[26] Hendrik Strobelt, Sebastian Gehrmann, Hanspeter Pfister, and Alexander M Rush. 2017. Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE transactions on visualization and computer graphics* 24, 1 (2017), 667–676.

[27] Kanit Wongsuphasawat, Daniel Smilkov, James Wexler, Jimbo Wilson, Dandelion Mane, Doug Fritz, Dilip Krishnan, Fernanda B Viégas, and Martin Wattenberg.

2017. Visualizing dataflow graphs of deep learning models in tensorflow. *IEEE transactions on visualization and computer graphics* 24, 1 (2017), 1–12.

[28] Yuhao Wu, Shaowei Wang, Cor-Paul Bezemer, and Katsuro Inoue. 2018. How Do Developers Utilize Source Code from Stack Overflow? *Empirical Software Engineering* (2018), 1–37.

[29] Qian Yang, Alex Scuito, John Zimmerman, Jodi Forlizzi, and Aaron Steinfeld. 2018. Investigating how experienced UX designers effectively work with machine learning. In *Proceedings of the 2018 Designing Interactive Systems Conference*. 585–596.

[30] Qian Yang, Jina Suh, Nan-Chen Chen, and Gonzalo Ramos. 2018. Grounding interactive machine learning tool design in how non-experts actually build models. In *Proceedings of the 2018 Designing Interactive Systems Conference*. 573–584.

[31] Kaichao You, Mingsheng Long, Jianmin Wang, and Michael I Jordan. 2019. How Does Learning Rate Decay Help Modern Neural Networks? (2019).

[32] Tianyi Zhang, Björn Hartmann, Miryung Kim, and Elena L Glassman. 2020. Enabling Data-Driven API Design with Community Usage Data: A Need-Finding Study. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.

A INTERVIEW QUESTIONS IN THE FORMATIVE STUDY

We used the following nine questions to guide the semi-structured interview in the formative study.

Section 1. Learning Neural Networks

1. What kinds of machine learning projects have you worked on?

2. What kinds of challenges do you face when learning to build neural networks?

3. What kinds of online resources do you find very useful?

Section 2. Searching Example Neural Networks

1. Do you search for examples when you learn and build neural networks? How often?

2. How do you search for such examples? Do you search on Google, Stack Overflow, GitHub? What kinds of keywords do you often use?

3. What kinds of difficulties do you have when searching for those examples?

Section 3. Visualizing Example Neural Networks

1. Suppose we have built a magic search engine that can identify many relevant deep learning projects. What kinds of information do you want to discover from this pile of projects? Or what kinds of questions do you want to answer using this data?

2. Have you used TensorBoard before? How do you like or dislike the visualization tool in TensorBoard?

3. We sketched several alternative designs for visualizing neural networks. How do you like or dislike each visualization design?