NeuralVis: Visualizing and Interpreting Deep Learning Models

Xufan Zhang, Ziyue Yin, Yang Feng, Qingkai Shi, Jia Liu, Zhenyu Chen
State Key Laboratory for Novel Software Technology, Nanjing University, China
corresponding author: liujia@nju.edu.cn

***Abstract*—Deep Neural Network(DNN) techniques have been prevalent in software engineering. They are employed to facilitate various software engineering tasks and embedded into many software applications. However, because DNNs are built upon a rich data-driven programming paradigm that employs plenty of labeled data to train a set of neurons to construct the internal system logic, analyzing and understanding their behaviors becomes a difficult task for software engineers. In this paper, we present an instance-based visualization tool for DNN, namely NeuralVis, to support software engineers in visualizing and interpreting deep learning models. NeuralVis is designed for: 1). visualizing the structure of DNN models, i.e., neurons, layers, as well as connections; 2). visualizing the data transformation process; 3). integrating existing adversarial attack algorithms for test input generation; 4). comparing intermediate layers' outputs of different inputs. To demonstrate the effectiveness of NeuralVis, we design a task-based user study involving ten participants on two classic DNN models, i.e., LeNet and VGG-12. The result shows NeuralVis can assist engineers in identifying critical features that determine the prediction results.**

Video: https://youtu.be/solkJri4Z44.

***Index Terms*—visualization, neural network, comparative research**

I. Introduction

Deep Neural Networks have been employed to develop many DNN-based applications because of its high accuracy and superior performance in handling some well-defined tasks, including natural language processing(NLP), image classification, and face recognition. However, because DNN-based software applications are constructed based on the design and programming diagrams that are different from the conventional software systems, it becomes difficult for engineers to analyze and further understand their behaviors and execution in the development. On the other hand, understanding the behavior of DNN is critical for both developers and end-users to build trust with the DNN models, which is an important problem when the model is used for decision making. For example, medical diagnosis cannot be entirely acted upon on DNN-based applications, as the consequences may be catastrophic. To alleviate these problems, researchers have proposed many software testing and debugging methods to improve model quality. However, practices in deep learning testing are in the early stage [1] , many approaches focus on finding adversarial examples based on the structure coverage [2] but fail to provide interactive ways to support engineers in analyzing and understanding the incorrect behaviors of DNNs. Especially, for safe-critical applications, understanding reasons why the DNN behaves incorrectly is a challenging but critical task.

By transforming abstract data into graphics, visualization techniques are considered to be an extremely useful aid to study in fields like hydrodynamics or chaos theory. To assist in analyzing DNNs' behaviors, software engineering researchers have designed some visualization tools. For example, Google has implemented TensorBoad to visualize scalar values, such as loss and accuracy, during the training stage [3]. However, exploration in the training stage focuses on metrics to monitor performance other than concrete behaviors of DNN models. Engineers can do little interaction with model components during the training phase since hyper-parameters in the model keep tuning.

In this paper, we present an interactive visualization tool for trained neural networks, namely NeuralVis, to assist software engineers in analyzing and understanding both the static structure and the dynamic behaviors of neural network models.

NeuralVis has the following features:

1) NeuralVis is designed for bringing software engineers intuitive understanding of the trained DNNs. It extracts and visualizes information of both static structures and runtime behaviors of a model.
2) It implements an algorithm to mutate models and thus help developers to compare intermediate outputs of the model and corresponding mutated ones. Based on the differential analysis, software engineers can identify critical features connections, and layers of a model.
3) NeuralVis provides GUI to enable engineers to interact with the visualized neural network. These interfaces facilitate engineers in manipulating the model structure and thus analyzing their behaviors.

II. Approach

For trained network models, researchers have been working on visualization methods to make it understandable. Regardless of the high score it achieves over the whole dataset, how it predicts a single input is still hard to understand. To simplify the task, instance-based visualization is adopted.

Several visualization tools are proposed to visualize deep neural networks, especially convolutional neural networks. Both 2D and 3D visualization are adopted in our tool. 3D visualization brings users an overview of the structural architecture for a trained model and 2D enables testers to concentrate on the detailed activation status on a specified

978-1-7281-2508-4/19/$31.00 ©2019 IEEE
DOI 10.1109/ASE.2019.00113

IEEE computer society

layer. To make it more interactive, comparative research and differential analysis are introduced in our visualization tool. Engineers can compare the intermediate output of different inputs to check the similarities and differences, which adds to their comprehension of how adversarial samples deceive the neural network model.

NeuralVis is built on top of TensorSpace, thus we will not elaborate on how model components are visualized in detail. Instead, we introduce how we bring more intuitions with our innovative approach. To bring engineers more intuition about the trained model, we propose a novel approach to implement visualization by integrating the concept of comparative research. Comparative research is the act of comparing multiple things to seek out any findings. In NeuralVis, two features to achieve comparative research in trained models are introduced.

Feature 1: Output comparison. For a given trained model M, let x denote the input, y denote the output. A common way to describe neural network is: $y = M(x)$.

NeuralVis is capable of visualizing complicated models. Model M consists of multiple layers, if we use f_i, w_i and b_i to describe the activation function, weight matrix, and bias of i^{th} layer, it can be expanded as:

$$y = f_n(w_n(f_{n-1}(w_{n-1}(..., f_1(w_1 x, b_1), ...)), b_{n-1}), b_n) \quad (1)$$

The intermediate output of layer i can be calculated and visualized. At the moment, it is still difficult for engineers to tell how the network model works even with these intermediate outputs. For example, engineers can hardly tell why the model fails at a specific input. Rather than telling the causes of the prediction, by adding another input for comparison, engineers are expected to tell the differences between two inputs in intermediate layers.

Common tasks like adversarial sample generation are also included in this tool since it is a useful aid to create test inputs. To learn more details about the differences among different inputs, especially the differences between the original input and the adversarial sample, engineers can look into a specific layer to check the activation status.

Feature 2: Model mutation. The model mutation is achieved by adding perturbations to equation 1. Filters in CNNs are considered to extract features from the original input. NeuralVis grants engineers the ability to prevent a target group of filters from working while the others are unchanged.

Filters in different layers can be frozen at the same time without improving complexity. The **mutate_output** algorithm is shown in Algorithm 1. It takes as input the trained neural network $model$, the sample $input$, and the frozen filter configurations in JSON format $config$. Static structural information of layers is extracted from the model. The output of each layer is calculated in the loop, if no $config$ is present, the result will be passed to the next layer as input. Otherwise, function $prepare\_input$ is responsible for applying zero vectors on marked filters in that layer. The output of each layer is stored in an array that is returned once it reaches the last layer.

Algorithm 1 mutate_output($model$, $input$, $config$)

```
structure = extract_layers(model)
for i = 0 to len(structure) do
    output = inner_output(model, input, i)
    if config is None then
        input = output
    else
        input = prepare_input(output, config, structure, i)
    end if
    result[i] = output
end for
return result
```

III. The Design of NeuralVis

We present the architecture of NeuralVis in Figure 1. Model visualization and output visualization are fundamental functions. For a given trained model and input, static structural information is extracted, how the input transforms inside the network is calculated. With this data, the model and the output are visualized. To make the process more interactive, NeuralVis reacts to engineers' operations in model mutation and input generation.

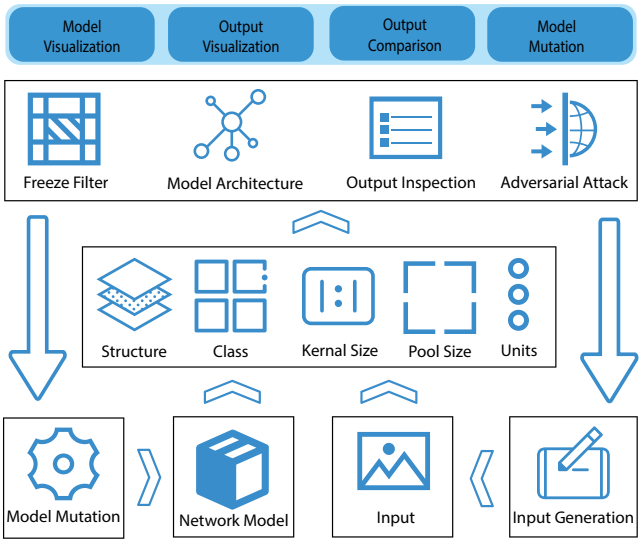

Fig. 1. Components and features of NeuralVis

NeuralVis consists of three main sections as depicted in Figure 2 where 1) the operation panel on the left to perform primary tasks, 2) the detailed panel to visualize details of the model structure and the data transformation throughout the model, 3) the comparison panel to intercept intermediate output of multiple inputs on a specified layer when the comparison task is activated.

A. The operation panel

The operation panel shows primary tasks engineers can perform, including 1) model upload, 2) model selection, 3)

input upload, 4) input selection, and 5) synthetic samples generation, 6) single input visualization, and 7) a pair of inputs comparison. As shown in Figure 1, the network model and the input data are two primary components engineers work with. NeuralVis support Keras models at the moment. Uploaded network models appear in the model list. NeuralVis calls the API in Keras library to load the model. Static structural information is extracted from the model, transformed into a 3D graph. However, only general configurations of the models are provided for trained network models. NeuralVis handles tasks related to image classification, for simple images which can be created by hand easily, engineers can create stick figures consisting of lines and points with the provided sketchpad.

Besides, engineers can leverage the sample generation function to create adversarial samples with particular attack algorithms, which is located at the bottom of this section. Available algorithms are provided in an option list. If an adversarial sample is generated successfully, it will be added to the sample list. Engineers waste much time in code implementation for adversarial samples. They are free from the coding implementation in NeuralVis. All they need to do is operate on the panel to tell NeuralVis the original input together with the selected algorithm.

B. The detailed panel

The detailed panel works as a response to users' actions in the operation panel. Detailed information about models and intermediate activation status will be presented to realize model visualization and output visualization. For model structure and component visualization, we leverage the implementation of TensorSpace. This visualization technique helps users work with the model graph in more interactively compared to node-link graphs proposed in TensorBoard. Intermediate outputs in each layer are rendered together with model components. Engineers can click on the model to expand the detailed components in each layer, drag and spin the model to check it from different angles, zoom in to look into a single layer, or zoom out to have a global view.

To make *feature 2* more interactive, inspired by mutation analysis proposed in [4], engineers can click on filters in an expanded layer to tell NeuralVis which filters should be frozen to implement model mutation in Figure 1. Filters are marked as grey by altering the opacity to indicate that it is frozen. As illustrated in *Feature 2*, no activation status will be passed to the next layer by these frozen filters. Any changes brought to the following layers are visualized as a response to the user's action.

C. The comparison panel

Inspired by comparative research, software engineers can compare the output of multiple inputs with NeuralVis as stated in *Practice 1*. By selecting a pair of inputs from the sample list, section C will pop out at the bottom of section B. Layers to intercept need to be selected. Activation status of that specified layer will be visualized. In this way, engineers can compare the intermediate output between the original input and a corresponding adversarial sample, similarities and differences observed will bring them intuition about hidden layers.

Since feature maps are of different sizes, it can be difficult for engineers to compare intermediate output in a relatively small panel. To make it more friendly, engineers can expand the comparison panel to full size.

IV. Evaluation

To evaluate the usefulness and effectiveness of NeuralVis, we train LeNet [5] on MNIST [6] and VGG-12 [7] on CIFAR-10 [8] to conducted a task-based user study with ten participants. We used MNIST since it is easier for participants to draw handwriting images online. CIFAR-10 was used to illustrate the intuition brought to engineers in more complex situations. In this study, we design the following tasks:

1) describing the structure, including the layers, specifications and their relationships based on both trained DNN models,
2) creating some inputs with the sketchpad and using the model to predict their label based on the trained LeNet;
3) describing the differences between the intermediate outputs of two inputs based on the trained LeNet;
4) highlighting critical features in the original input that determine the prediction outputs based on the trained VGG-12;

We recruit ten master students of software engineering as participants. All participants are required to perform the designed tasks on the browser of their laptops within one hour.

The study result shows that all candidates can perform tasks in time with NeuralVis. As is shown in Figure 2, with the model file selected, structural information will be displayed. All participants can describe the structure of used models.

All participants can perform the tasks in time with NeuralVis. For task 1, as is shown in Figure 2, with the model file selected, structural information will be displayed. All participants were able to describe the structure of used models. For task 2, participants manually write some digits on the sketchpad, and all of them successfully finish this task within several seconds. For task 3, participants report that "they can tell the feature that influences the final output". Note that if a layer is frozen by users, no activation information is passed to the next layer. In this task, intermediate outputs are inspected, which offered the participant a chance to view the differences in hidden layers via comparative research. We analyze the result submitted by all participants and find that there were some filters participants considered to be important and filters that are useless according to the filters they highlighted, which proves that NeuralVis brings them intuition in understanding importance of neurons inside a neural network.

After participants finish tasks, we also interview them to collect feedback on NeuralVis. The feedback shows the comparison of intermediate output is critical for the analysis of the behavior of DNN models.

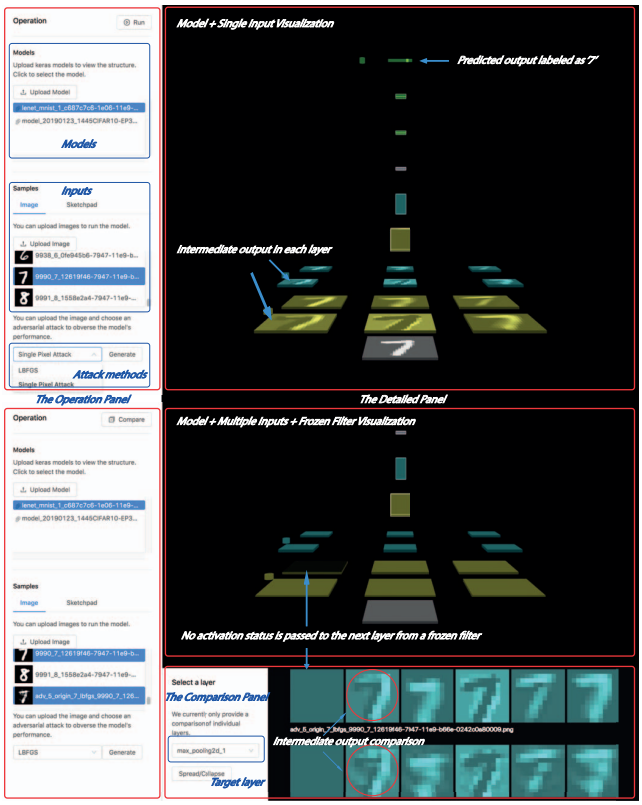

Fig. 2. Visualization tasks performed in the study

V. Related Work

Kahng et al. [9] implement ACTIVIS that can offer an overview of the computation graph and activation status for any given input. Developers can inspect on layers they are interested in when dealing in large and complex models. Harley et al. [10] design a technique to analyze the architecture of DNN models by visualizing it into a node-link diagram. However, the graph is not suitable for complex models as there will be numerous links if there is a large number of neurons in the network. 3D visualization techniques solve the problem by distributing neurons in each layer in space. Zhang et al. [11] visualize the feature map in the image classification task. In this technique, they treat the output feature map as a 2D array which can be transformed into an image by nature. Bolei et al. [12] design a technique to compute the image-resolution receptive field of neural activation by inverting the feature map back to the input space.

On the other hand, researchers develop many techniques to visualize the behavior of DNN models to assist their analysis. Girija et al. [3] present TensorBoard that can display the computational graph of the model and produces scalar values during the computation. Gong et al. [13] develop NeuronBlocks that empowers engineers to check model configuration and model architecture.

Different from these works, our research enables developers to interact with DNN models and compare the output of intermediate layers for given inputs. It provides developers with an overview of both static structures and the dynamic behaviors of neural network models.

VI. Conclusions

In this paper, we present NeuralVis, a web-based tool for visualizing trained neural networks to enhance engineers' understanding of both model structure as well as data transformation. Also, NeuralVis provides the functionality of comparing intermediate output between a pair of inputs. Based on this functionality, engineers can identify critical features that determine the prediction results. Because NeuralVis is instance-based when visualizing behaviors, input selection counts a lot concerning efficiency and efficacy. Similar to prioritize a test case during the test, inspired by [14], we plan to guide engineers in choosing an input by labeling them with different priorities.

Acknowledgment

The work is partly supported by the National Natural Science Foundation of China (61932012, 61832009).

References

[1] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 1–18.

[2] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*. ACM, 2018, pp. 303–314.

[3] S. S. Girija, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *Software available from tensorflow. org*, 2016.

[4] W. Shen, J. Wan, and Z. Chen, "Munn: Mutation analysis of neural networks," in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 2018, pp. 108–115.

[5] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, vol. 20, 2015.

[6] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[8] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, vol. 55, 2014.

[9] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. P. Chau, "A cti v is: Visual exploration of industry-scale deep neural network models," *IEEE transactions on visualization and computer graphics*, vol. 24, no. 1, pp. 88–97, 2018.

[10] A. W. Harley, "An interactive node-link visualization of convolutional neural networks," in *International Symposium on Visual Computing*. Springer, 2015, pp. 867–877.

[11] Q.-s. Zhang and S.-C. Zhu, "Visual interpretability for deep learning: a survey," *Frontiers of Information Technology & Electronic Engineering*, vol. 19, no. 1, pp. 27–39, 2018.

[12] Z. Bolei, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Object detectors emerge in deep scene cnns," 2015.

[13] M. Gong, L. Shou, W. Lin, Z. Sang, Q. Yan, Z. Yang, and D. Jiang, "Neuronblocks–building your nlp dnn models like playing lego," *arXiv preprint arXiv:1904.09535*, 2019.

[14] Q. Shi, J. Wan, Y. Feng, C. Fang, and Z. Chen, "Deepgini: Prioritizing massive tests to reduce labeling cost," *arXiv preprint arXiv:1903.00661*, 2019.