An Interactive Node-Link Visualization of Convolutional Neural Networks

Adam W. Harley(✉)

Department of Computer Science, Ryerson University,
Toronto, ON M5B 2K3, Canada
aharley@scs.ryerson.ca

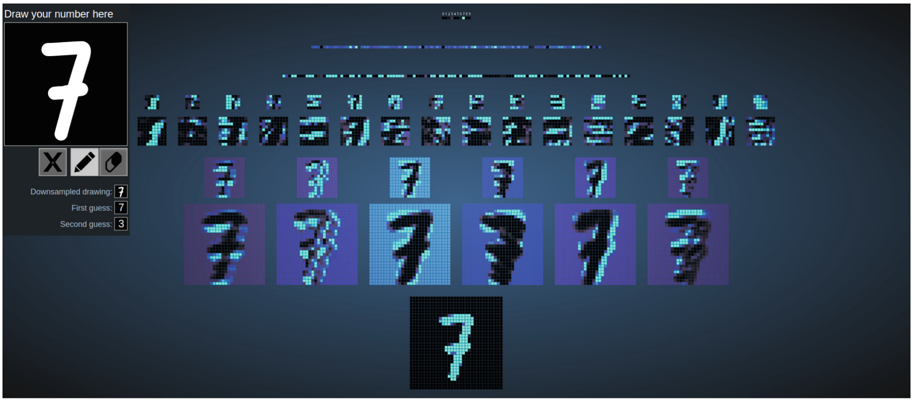
Abstract. Convolutional neural networks are at the core of state-of-the-art approaches to a variety of computer vision tasks. Visualizations of neural networks typically take the form of static diagrams, or interactive toy-sized networks, which fail to illustrate the networks' scale and complexity, and furthermore do not enable meaningful experimentation. Motivated by this observation, this paper presents a new interactive visualization of neural networks trained on handwritten digit recognition, with the intent of showing the actual behavior of the network given user-provided input. The user can interact with the network through a drawing pad, and watch the activation patterns of the network respond in real-time. The visualization is available at http://scs.ryerson.ca/~aharley/vis/.

1 Introduction

Convolutional neural networks (CNNs) are at the core of state-of-the-art approaches to a variety of computer vision tasks, including image classification [1] and object detection [2]. Despite this prevalence, interactive neural network visualization is still a relatively unexplored topic. Interactive simulations of toy networks have long existed [3], and visualizations of individual learned filters and features have emerged [4], but few visualizations illustrate how large-scale CNNs abstract from input to output. Motivated by this observation, this paper presents a new interactive visualization of a CNN trained on a specific task, with the intent of showing not only what it has learned, but how it behaves given new user-provided input.

Interactively visualizing the behavior of a neural network has a number of practical applications. First, having a detailed understanding of how neural networks behave is important for making practical use of them, so it is useful to have a visualization to support this understanding. Second, an interactive visualization of a neural network gives users the power to easily experiment with the input, and observe the effects of their experimentation immediately. This type of experimentation facilitates the process of exploring the strengths and weaknesses of a network, which is a critical part of designing better networks [4]. A third benefit of visualizing the behavior of a network is that it allows users to

G. Bebis et al. (Eds.): ISVC 2015, Part I, LNCS 9474, pp. 867–877, 2015.
DOI: 10.1007/978-3-319-27857-5_77



Downsampled drawing: 7
First guess: 7
Second guess: 3

Fig. 1. The proposed visualization: an interactive node-link diagram of a convolutional neural network trained to recognize handwritten digits. On the left is a drawing pad, where the user can draw numbers for the network to classify. The activation level of each node is encoded in hue and brightness. Studying the architecture of the network, and experimenting with its input-output process, can enlighten students of machine learning as to how the network performs its abstraction from images to digits.

explore the layer-by-layer output of a network, and build intuitions about how neural networks perform hierarchical abstraction from input to output [5].

This visualization is targeted towards students of machine learning, who are learning how to design, code, and train new neural networks. To make the visualization useful to that audience, the visualization is based on the well-established node-link diagram representation of fully-connected neural networks. The visualization is supported by an actual neural network, designed and trained to recognize handwritten digits with high (99 %) accuracy. In the neural network literature, handwritten digit recognition is a well-known solved problem, and often serves as an example of an appropriate application of neural networks [6]. Users are able to interact with this network through a "drawing pad", on which they can write new numbers for the network to recognize. A screenshot of the visualization is shown in Fig. 1.

This paper begins by reviewing prior work on visualizing neural networks, with a special emphasis on identifying why the classic node-link diagram representation has endured the test of time. The paper then proceeds to describe the approach to developing the current visualization, considering the challenges of revealing inner detail, the use of color, and the elements of interaction. Finally, the effectiveness of the visualization is discussed, and future work is proposed.

Contributions: The proposed visualization is the first to accurately and interactively illustrate the structure, scale, and low-level inner workings of a CNN applied to a practical computer vision problem. Prior work on this topic was limited to simpler architectures, smaller problems, or static visualizations. The new visualization can be explored at http://scs.ryerson.ca/~aharley/vis/.

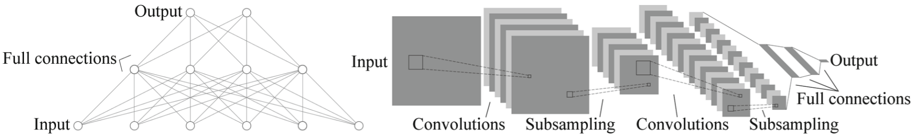


Fig. 2. Typical illustrations of fully-connected (left) and convolutional (right) neural networks (adapted from [12]).

2 Background and Related Work

This section establishes the context of the current work, by: (i) defining neural networks, and exploring why node-link diagrams are typically used to represent them, (ii) examining the challenges of complexity and scale that arise when using node-link diagrams for large neural networks, and (iii) considering the effective use of interaction.

2.1 Neurons as Nodes in a Graph

Neural networks compose many small functions in a network-like architecture, creating a larger function capable of pattern recognition [7]. In biological neural networks, the unitary function is a neuron, and neurons are connected together in extremely complex arrangements [8]. Artificial neural networks can be arbitrarily simple. One of the simplest arrangements is as a feed-forward graph with stacked “layers” of nodes, where every pair of neighboring layers is fully connected [9,10] (see Fig. 2, left). This arrangement is called a *fully-connected neural network*. A node-link visualization of this type of network is a straightforward outcome of (i) treating all neurons as identical processor units, and (ii) choosing an arrangement of neurons defined by a simple graph architecture. For these reasons, node-link diagrams of fully-connected networks are a mainstay in the inventory of visualizations for machine learning educators and researchers (e.g., see [6,11]).

Modern implementations of neural networks often use more complex architectures, but the variations typically appear underneath a traditional fully-connected network. That is, these implementations process the raw input (such as an image) with an alternative network, and then use the output of that network as input to a fully-connected network. Convolutional networks interact with fully-connected networks in this way. Figure 2 (right) shows a typical diagram for illustrating a CNN. The algorithm for CNNs relies heavily on the convolution operation, which is used to sequentially apply a series of (learned) filters to the input. A CNN can also be interpreted as a graph, which although is different in appearance from the fully-connected network graph, uses all the same mathematics for learning [12]. Figure 3 illustrates how convolutions can be interpreted as graph-like connections. For students learning about CNNs for the first time, it can be difficult to mentally assimilate the relationship between node-link diagrams and convolutional nets. One of the goals of the current visualization is to make this relationship easier to understand.

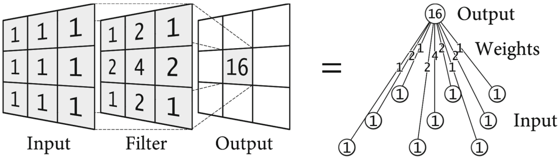


Fig. 3. A 3 × 3 convolution filter is equivalent to a node with nine weighted connections, where the filter values correspond to the weights. A convolution layer applies this filter to every location in the input image, producing a new (filtered) image.

2.2 Challenges of Complexity and Scale

The visualizations of neural networks in machine learning literature have changed only slightly over the years. The most significant trend is that the node-link diagrams have become smaller and less detailed over time. This is consistent with fully-connected networks' declining novelty, and reflects an intent of emphasizing things other than the structure of the networks. The most common simplified representation replaces each layer of nodes with a solid block, and replaces the dense connections between layers with either a single arrow pointing from one layer to the next [1], or with edges connecting the outskirts of the layers [13]. Simplifying the node-link diagram into a block diagram also addresses a problem of scale. To solve problems of practical value, the required network is often enormously large. A node-link diagram of such a network would have enough edges that the space in between layers would be opaque with lines. A block diagram is therefore an effective way of overcoming this problem of scale. The fully-connected portion of the CNN in Fig. 2 shows a simplification along these lines.

The main issue with grouping the nodes together by layer is that it eliminates the possibility for interaction and detailed analysis. In other domains, where analysis is often the primary focus, other solutions have been introduced. For example, it is sometimes possible to reveal a great deal of information about a set of connected nodes by bundling related edges together [14]. An alternative is to display only a representative subset of the nodes, or perhaps allow users to hover over a node to see edges or stubs leading into it [15,16]. A closely related strategy to these is the use of multi-scale navigation to "search, show context, expand on demand" [17]. In all, visualization research indicates that focus-plus-context techniques [18] could serve as reasonable alternatives to simply not showing the edges. The current work makes use of these ideas.

2.3 Prior Interactive Visualizations

Many interactive neural network visualizations exist. An early and popular visualization is the Stuttgart Neural Network Simulator (SNNS) [19], which shows neural nets as 2D and 3D node-link diagrams, in which the nodes and edges are colored in a scheme that maps negative and positive values to different ends of a palette. However, SNNS is targeted toward researchers, and accordingly its interface and basic usage demand some expertise. A similar tool is N^{2}VIS [20], which additionally makes use of a compact "matrix-like" visualization of a neuron's weights, in which each cell of the matrix represents a weight, and the cell is coloured according to the weight's magnitude. This is very similar to visualizing a CNN's parameters as filters, although the networks and visualizations in N^{2}VIS do not actually scale to convolutional networks and vision tasks. Interaction in these and similar research-targeted applications (*e.g.*, [21,22]) typically centers around designing new networks, and making minute adjustments to trained networks, to see how these changes affect performance.

Visualizations designed for a tutorial context are different. For example, *Neural Java* [3] provides an extensive set of web-based exercises and demos, allowing students to experiment with and learn about a variety of neural network designs. In one application, the user can make design choices on a network tasked with solving a toy version of the handwritten digit recognition problem. Once the network is trained according to the user's settings, the user can use a cursor to draw new numbers for the network to classify, and view the network's classification output. Despite there being no depiction of the actual network being trained, this type of application enables students to empirically determine reasonable answers to a variety of challenging questions, concerning (for example) convergence, the optimal number of nodes and layers, translation invariance, and more. These are the types of benefits the current visualization aims to deliver.

Most interactive visualizations only depict fully-connected networks, and furthermore only visualize networks that are too small to be effective at computer vision problems. A recent exception to this is *ConvNetJS* [23], which is a JavaScript library for training neural networks. The website for the project features a set of visualizations, which have interactive examples of neural networks. As in *Neural Java*, one example features a neural network solving a handwritten digit recognition task, although in this case using a standard dataset (MNIST [12]). The visualization shows the network's layer-by-layer activation patterns in response to example inputs, which gives the user an in-depth look at how the network arrives at its final classification. A weakness of the visualization is that the activation patterns are not organized in a way that shows the architecture of the network (e.g., in a node-link diagram); instead these are simply shown in order, from the zeroth layer to the final layer. Also, unlike the example in *Neural Java*, this visualization does not allow users to interactively create new inputs for the network to classify. Nonetheless, *ConvNetJS* sets a high benchmark for scale and practical realism, which the current visualization aims to match.

3 Technical Approach

This section describes the technical approach to creating the visualization framed in the previous section. The discussion begins with an analysis of the activities that users of the visualization are expected to be interested in, then proceeds to describe the major visual elements employed to meet the requirements of those tasks. The section concludes with a discussion of the various interactive elements implemented.

3.1 Task Analysis

The visualization should meet the following goals, which summarize the unique properties of convolutional networks in computer vision, and reflect lessons learned from prior work. First, the visualization should handle networks large enough to solve practical vision tasks, such as handwritten digit recognition. Second, the visualization should depict the entire network architecture with a node-link diagram. Third, the visualization should allow users to easily experiment with the input-output process of the network, allowing them to judge the network's robustness to translational variance, rotational variance, and ambiguous input. Fourth, the visualization should allow users to view details on individual nodes, such as the activation level, the calculation being performed, the learned parameters (*i.e.*, the node's weights), and the numerical inputs and outputs.

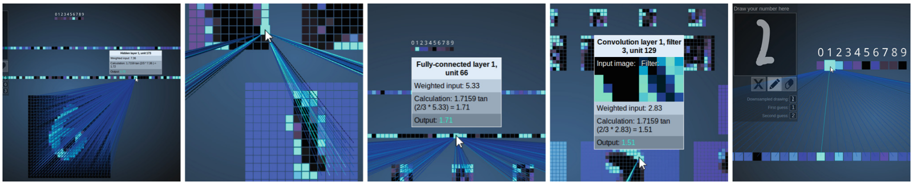
3.2 Visual Elements

This section describes the major visual elements of the visualization, emphasizing their justification through the points established in the task analysis, as well as through theoretical principles of visualization. Screenshots are shown in Fig. 4.

Node-link Diagram: The only issue with a straightforward implementation of a node-link diagram is that large networks yield a dense mass of edges between layers. This was addressed by only showing edges for one node at a time, and only on request. This strategy is uniquely possible in domains with simple network architectures: unlike networks describing natural phenomena (*e.g.*, [15,24]), the edges of CNNs can be implied by their regular pattern. This strategy achieves a compromise between the block diagram's simplicity, and the node-link diagram's potential for detail.

Camera: Users can zoom in and out by scrolling, and translate the network by dragging the right mouse button. While exploring the visualization through these controls, the users can focus their attention on particular layers or features, gain familiarity with the architecture of the network, and also build an appreciation for the network's scale.

Node Activations and Edge Strengths: Displaying the activation levels of individual nodes, as well as the strength of edges between nodes, is crucial for creating an informative visualization of a neural network's behavior. At the zeroth layer of the network, the individual nodes correspond to pixels in the input image, so the activation level of each node simply corresponds to the brightness of the respective pixel. At every other layer of the network, the activation level corresponds to how closely the output from the layer below matches with the node's learned "ideal input". The image of the ideal input is represented in the strengths of the node's edges. In this visualization, edge strengths and activation



Hidden layer 1, unit 173
Weighted input: 7.36
Calculation: 1.7159 tan (2/3 * 7.36) = 1.72
Output: 1.72

Fully-connected layer 1, unit 66
Weighted input: 5.33
Calculation: 1.7159 tan (2/3 * 5.33) = 1.71
Output: 1.71

Convolution layer 1, filter 3, unit 129
Weighted input: 2.83
Calculation: 1.7159 tan (2/3 * 2.83) = 1.51
Output: 1.51

Downsampled drawing:
First guess: 1
Second guess: 2

Fig. 4. Screenshots from the visualization. From left to right, the images show: edges revealed by hovering on a fully-connected node, edges revealed by hovering on a convolutional node, details revealed by clicking on a fully-connected node, details revealed by clicking on a convolutional node, and finally the top layer activations of a network given ambiguous input. This figure is best viewed in a zoomable PDF.

levels were encoded in a (shared) colormap, which mapped low and high values to different ends of a palette. The colormap is a variation on the rainbow palette, making use of a wide range of hues and saturations to ensure that the colors "pop out" from the background; it also uses a linear value curve, to effectively differentiate low and high levels of activation or edge strength. The colormap is consistent with colour-encoding theory, which encourages a mapping across value for ordered elements [25], and discourages the use of a full rainbow pallette [26].

Background: Preliminary user testing revealed that if the background has a solid color, and that color is drawn from the same palette as the activation patterns, then many of the nodes will inevitably blend in with the background. If the background has a color that none of the nodes have, the color schemes may clash, and attention will be drawn away from the nodes and toward the background. The solution was to give the background a large blurred ellipse, which faded in brightness toward the edges, creating the illusion of a soft light emanating from (or being projected into) the center of the visualization. Since the ellipse has a smooth color gradient, the solid-colored nodes cannot blend into it. The solution thus makes it unnecessary to alter the color palette, and also adds an aesthetically pleasing soft lighting effect.

Drawing Pad: The drawing pad is positioned at the top-left corner of the visualization, and includes three buttons just below it: a pen, an eraser, and a "clear" button. The eraser encourages the user to make small adjustments to their input. One of the greatest learning experiences to be derived from interacting with the network is in seeing how the network responds to slight changes in input. If the user gradually morphs a "1" into a "2", the network will respond in its top layer by gradually losing confidence (*i.e.*, activation strength) in the "1" node and gaining confidence in the "2" node. It is worth noting that the drawing pad has more pixels than the actual input layer of the network. The drawn input is thus downsampled before it is used by the network. An alternative strategy (pursued in *Neural Java* [3]) is to give the drawing pad "large pixels". However, the high-resolution strategy is more in line with a current issue of neural networks in computer vision: high-resolution input is available, but a high-resolution network would be computationally intractable. Showing this downsampling stage in the visualization presents another learning opportunity for the user.

Node Information on Focus: Two major focus-plus-context tools were implemented in the visualization. First, by hovering on a node, the user is able to see all of the edges leading toward that node. Second, by clicking on a node, the user can access detailed information about that node, including node's position and role in the network, the numerical input and output of the node, as well as the calculation being performed. For convolutional nodes, since each node's action can be interpreted as applying a filter to a small location in the image, clicking such a node reveals the corresponding image and filter, in addition to the numerical data.

4 Implementation Details

To support the visualization, two networks were trained on an augmented version of the MNIST dataset [12] to classify 28×28 pixel images of handwritten digits. The dataset was augmented with translated, rotated, and skewed versions of the original training data, to improve the network's robustness to these types of variations.

First, a convolutional network was trained, with 1024 nodes on the bottom layer (corresponding to pixels), 6 5×5 (stride 1) convolutional filters in the first hidden layer, followed by 16 5×5 (stride 1) convolutional filters in the second hidden layer, then three fully-connected layers, with 120 nodes in the first, 100 nodes in the second, and 10 nodes in the third (corresponding to the 10 digits). The convolutional layers are each followed by downsampling layer that does 2×2 max pooling (with stride 2). This network achieved a 99 % classification accuracy on the same dataset, which is consistent with the related work [12].

Second, a fully-connected network was trained, with 784 nodes on the bottom layer (corresponding to pixels), 300 nodes in the first hidden layer, 100 nodes in the second hidden layer, and 10 nodes in the output layer (corresponding to the 10 digits). In earlier work, this network architecture was shown to be capable of achieving approximately 97 % classification accuracy on the MNIST test set [12]. The newly trained network achieved approximately 96 % classification accuracy on the same dataset, but showed slightly better invariance to translation, rotation, and skewing, as expected.

This visualization was implemented as a WebGL application, written in JavaScript. As a WebGL application, the visualization is compatible with any device that has an HTML5-ready browser and a GPU. This includes mobile devices like the iPad 2 and higher, iPhone 4 and higher, and more. The drawing pad works especially well with touch-screen devices, since the user can draw with his or her finger, rather than with a mouse. The open source libraries *jquery*, *three.js*, and *math.js* were instrumental in achieving the final result. MATLAB was used to train the neural network, and also to export parameter matrices, colormaps, and 3D network layouts for JavaScript.

5 Empirical Evaluation

A thorough evaluation of the visualization's efficacy as an educational tool would ideally involve a two-arm randomized controlled trial, with machine learning students as participants. Future work may develop an evaluation along these lines. The present evaluation explores instead an important factor of usability known as responsiveness. This factor represents the ability of the system to provide quick and meaningful interaction, and it is tested by measuring the system's response time to user inputs [27]. Studies on human perception have identified three approximate limits for a computer application's response time [28,29]: 0.1 seconds is the limit for providing an impression of continuous feedback (*e.g.*, the manipulation of objects in the interface should meet this criteria); 1.0 seconds is the limit for providing an impression of an immediate response (*e.g.*, a command given to the application should be carried out within this limit); 10 seconds is the limit for keeping the user engaged (*e.g.*, a delay exceeding this limit may cause the user to switch away from the task).

The current visualization has two main types of interaction: manipulation of the camera, which is carried out on the GPU, and drawing pad interaction which is carried out on the CPU. On a PC with a mid-range graphics card (ATI Radeon HD 4850), the visualization runs at a consistent 60 frames per second while the camera is being manipulated, which is well within the 0.1-second criterion. With a modern processor (Intel Core i7-4770), drawing pad interaction is processed at an average of 36 milliseconds, which is again within the "continuous feedback" 0.1-second criterion. On mobile devices with GPUs (such as the iPad 2), the frame rate of the visualization stays at 60, while the drawing pad interaction slows to approximately 250ms, which places the interaction in the "immediate response" limit. These results suggest that the application is responsive enough to support engaging user interaction on a variety of devices.

In preliminary user testing, observing usage of the drawing pad led to some changes in its functionality: in an early iteration of the visualization, the drawing pad had a "Go" button, which caused the drawing to be input into the bottom layer of the network. This two-stage process allowed the user to prepare a detailed drawing before seeing the network's interpretation of it. However, users did not immediately know what to do with the button, and were confused as to why the network did not respond automatically once a number was drawn. Based on these observations, the button was removed, and the drawing pad was configured to send its data through the network after every stroke.

6 Conclusion

This paper presented a new interactive visualization of convolutional neural networks. Users can interact with the visualization by drawing new digits for the network to classify. The visualization shows the nodes of the network arranged in a node-link diagram, which is novel for visualizations of convolutional networks. Given an input image, the visualization shows the activation level of every node in the network, by setting the brightness and color of each node according to the magnitude of the activation. A variety of details-on-demand techniques were employed to incorporate additional information into the visualization, including node labels, weight images, edges, and a summary of the mathematics being computed at every node. The visualization also enables responsive experimentation with the network's input-output process, allowing users to learn about the network's strengths and weaknesses through interaction. Whereas prior visualizations have only depicted fully-connected, non-interactive, or toy-sized neural networks, the current work demonstrates that practical vision-trained convolutional nets can be effectively depicted in an interactive node-link diagram. The visualization, and its source code, are available at http://scs.ryerson.ca/~aharley/vis/.

Acknowledgements. The author gratefully thanks Tim McInerney and Kosta Derpanis for insightful discussions, and for helping improve the manuscript.

References

1. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS, pp. 1106–1114 (2012)
2. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: CVPR (2014)
3. Corbett, F.D., Card, H.C.: Neural Java: Neural networks tutorial with Java applets (2000). http://lcn.epfl.ch/tutorial/english/. Accessed on 31 Nov 2014
4. Zeiler, Matthew D., Fergus, Rob: Visualizing and understanding convolutional networks. In: Fleet, David, Pajdla, Tomas, Schiele, Bernt, Tuytelaars, Tinne (eds.) ECCV 2014, Part I. LNCS, vol. 8689, pp. 818–833. Springer, Heidelberg (2014)
5. Craven, M.W., Shavlik, J.W.: Visualizing learning and computation in artificial neural networks. Int. J. Artif. Intell. Tools **1**, 399–425 (1992)
6. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer-Verlag New York Inc, Secaucus, NJ, USA (2006)
7. Rosenblatt, F.: The perceptron: a probabilistic model for information storage and organization in the brain. Psychol. Rev. **65**, 386–408 (1958)
8. Lodish, H., Berk, A., Zipursky, S.L., Matsudaira, P., Baltimore, D., Darnell, J.: Molecular Cell Biology, 4th edn. W.H. Freeman, New York (2001)
9. Werbos, P.J.: Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Ph.D thesis, Harvard University (1974)
10. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by backpropagating errors. Nature **323**, 533–536 (1986)
11. Jarrett, K., Kavukcuoglu, K., Ranzato, M., LeCun, Y.: What is the best multistage architecture for object recognition? In: ICCV (2009)
12. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**, 2278–2324 (1998)
13. LeCun, Y., Kavukvuoglu, K., Farabet, C.: Convolutional networks and applications in vision. In: ISCAS, pp. 253–256 (2010)
14. Holten, D.: Hierarchical edge bundles: visualization of adjacency relations in hierarchical data. TVCG **12**, 741–748 (2006)

15. Al-Awami, A., Beyer, J., Strobelt, H., Kasthuri, N., Lichtman, J., Pfister, H., Hadwiger, M.: NeuroLines: a subway map metaphor for visualizing nanoscale neuronal connectivity. TVCG **20**, 2369–2378 (2014)
16. Lex, A., Partl, C., Kalkofen, D., Streit, M., Gratzl, S., Wassermann, A.M., Schmalstieg, D., Pfister, H.: Entourage: visualizing relationships between biological pathways using contextual subsets. TVCG **19**, 2536–2545 (2013)
17. Van Ham, F., Perer, A.: Search, show context, expand on demand: supporting large graph exploration with degree-of-interest. TVCG **15**, 953–960 (2009)
18. Shneiderman, B.: The eyes have it: a task by data type taxonomy for information visualizations. In: Visual Languages, pp. 336–343 (1996)
19. Zell, A., Mache, N., Hbner, R., Mamier, G., Vogt, M., Schmalzl, M., Herrmann, K.U.: SNNS (Stuttgart Neural Network Simulator). In: Skrzypek, J. (ed.) Neural Network Simulation Environments: The Kluwer International Series in Engineering and Computer Science, vol. 254, pp. 165–186. Springer, US (1994)
20. Streeter, M.J., Ward, M.O., Alvarez, S.A.: N2Vis: an interactive visualization tool for neural networks. In: Visual Data Exploration and Analysis VII, pp. 234–241 (2001)
21. Tzeng, F.Y., Ma, K.L.: Opening the black box - Data driven visualization of neural networks. In: Visualization, pp. 383–390 (2005)
22. Srp, J., Stehlík, L., Suda, M., Šašek, P., Zvánovcová, K.: Interactive neural network simulator (2007). http://sourceforge.net/projects/isns/. Accessed on 31 Nov 2014
23. Karpathy, A.: ConvNetJS: Deep learning in your browser (2014). http://cs.stanford.edu/people/karpathy/convnetjs/. Accessed on 31 Nov 2014
24. Henry, N., Fekete, J.D., McGuffin, M.J.: NodeTrix: a hybrid visualization of social networks. TVCG **13**, 1302–1309 (2007)
25. Ware, C.: Information Visualization: Perception for Design. Elsevier, Amsterdam (2012)
26. Rogowitz, B.E., Treinish, L.A.: How not to lie with visualization. Comput. Phys. **10**, 268–273 (1996)
27. Nielsen, J.: Usability Engineering. Elsevier, Boston (1993)
28. Miller, R.B.: Response time in man-computer conversational transactions. In: Proceedings of AFIPS Fall Joint Computer Conference, vol. 33, pp. 267–277 (1968)
29. Card, S.K., Robertson, G.G., Mackinlay, J.D.: The information visualizer: an information workspace. In: ACM CHI, pp. 181–188 (1991)