In situ TensorView: In situ Visualization of Convolutional Neural Networks

Xinyu Chen[◇], Qiang Guan[‡], Li-Ta Lo[†], Simon Su[*],Zhengyong Ren[‡], James Paul Ahrens[†], Trilce Estrada[◇]
[◇]University of New Mexico, [‡]Kent State University,
[†]Los Alamos National Laboratory, [*]US Army Research Laboratory
{xychen, estrada}@cs.unm.edu, {qguan, zren2}@kent.edu, {ollie, ahrens}@lanl.gov, simon.m.su@mail.mil

***Abstract*—Convolutional Neural Networks(CNNs) are complex systems trained to recognize images, texts and more. However, once trained, they are regarded as black-boxes that are not easy to analyze and understand. Visualizing the dynamics within such deep artificial neural networks can provide a better understanding of how they are learning and making predictions. In the field of scientific simulations, visualization tools like Paraview have long been utilized to provide insights. We present in situ TensorView to visualize the training and functioning of CNNs as if they are systems of scientific simulations. In situ TensorView is a loosely coupled in situ visualization open framework that provides multiple viewers with the ability to visualize and understand their networks. It leverages the capability of co-processing from Paraview to provide real-time visualization during training and predicting phases, and avoids heavy I/O overhead. Tensorview is easily coupled with Tensorflow, as it only requires the insertion of a few lines of code into a TensorFlow framework. In this work, we showcase visualizing LeNet-5 and VGG16 using in situ TensorView. With the insight provided by Tensorview, users can adjust network architectures, or compress pre-trained networks guided by visualization results.**

***Keywords*-Deep Neural Networks; Neural Network Compression; Online Pruning**

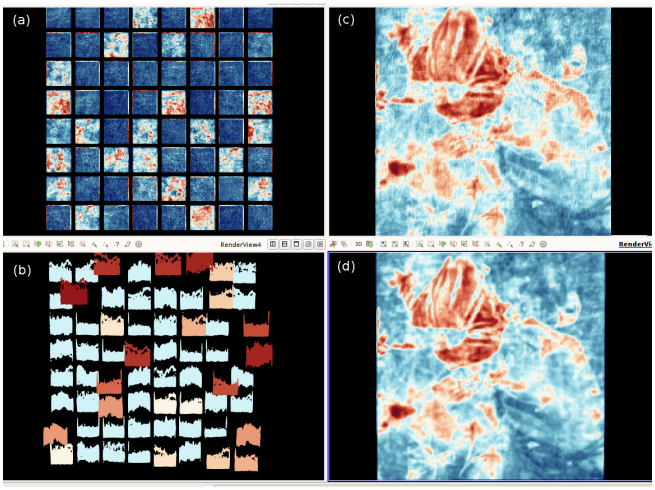


Figure 1: The first convolutional layer in the VGG16 network. (a) The activation images of one batch of samples. (b)The activation distributions of the same batch. Colors indicate similar groups. (c),(d) Two activation images with high similarity.

I. Introduction

Deep Convolutional Neural Networks (CNNs) have improved the state-of-art in vision and speech recognition[1]. Large CNNs[2], [3], [4] contain dozens to hundreds of layers. Training a deep convolutional neural network is slow [5] because the number of trainable parameters is extremely large (431k in LeNet-5, 61M in AlexNet) [6]. Tuning CNNs is very time-consuming, as researchers often need to try out different architectures and hyper-parameters such as learning rates, optimizers, batch sizes, etc. Sometimes, deep neural networks cannot learn effectively, or even stop learning altogether, because of inappropriate hyper-parameters.

To steer away from trial-and-error designs, we hope to answer the following questions through visualization means: Are there any redundant neurons or connections in large networks? Can we remove some neurons to make networks simpler? Can we quickly find gradient vanishing or explosion problems during training process? In order to answer these questions, the visualization tools themselves need to be scalable in order to process and visualize huge amounts of neurons and their connections.

Scientific visualization frameworks like Paraview [7] and VisIt [8] can visualize large simulations and enable scientists to interactively explore the data [9]. They greatly help scientists to get insights and accelerate data-driven researches and discoveries. Thus, it is promising to use Paraview to visualize the training, predicting and fine-tuning of deep convolutional neural networks. In this paper, we extend our previous work[10] to overcome the large I/O overhead in visualizing deep convolutional neural networks. The new in situ TensorView is a loosely coupled in situ visualization framework[11]. It embeds light-weighted TensorView Catalyst adaptor in CNNs built on TensorFlow framework and provides in situ visualizations of the extracted data in a Paraview client through the Catalyst co-processing pipeline.

In the feed forward pass, we visualize the activations a.k.a feature maps(we would use the term *activation* and *feature map* interchangeably for the rest of this paper) to identify similar responses among neurons and analyze if there exist redundant neurons. In the back propagation

[1]The publication has been assigned the LANL identifier LA-UR-17-27402.

pass, we visualize the changes of weights to quickly detect gradient vanishing or explosion problems. The goal of in situ TensorView is to help researchers to interactively build, use and understand their deep neural networks.

The Contribution of this paper can be summarized as follows:

- We extend our previous work which applied Paraview and visualization toolkit (VTK)[12] techniques to visualize deep convolutional neural networks. The improved method aims to help researchers to interactively visualize their neural networks in training and predicting phases. To the best of our knowledge, we are the first to use the Catalyst co-processing library to provide in situ visualization of deep convolutional neural networks. This reduces the I/O overhead and enables us to visualize large networks.
- We provide multiple in situ viewers to show both statistical information about similarities between neurons and intuitive images of activations and weights of convolutional filters. Previously, users have to wait until the training have finished. This improvement enables real time visualization.
- We carry out case studies to demonstrate the scalability and generality of our framework. We visualize both a small LeNet-5 network being trained from scratch and a large VGG16 network with pre-trained weight parameters.

The rest of the paper is organized as follows. In section II we discuss the related work.I n section III, we explain the methodology of our framework. In section IV, we present the results of two case studies. We conclude the paper in section V.

II. Related Work

Researchers have been working hard to understand deep convolutional neural networks. A recent survey by Homan et al[13] summarized state-of-art visualization work from 6 aspects: Why do people want to visualize deep learning? Who needs such visualizations? What can be visualized in deep learning? How to visualize these content? When should these visualizations be carried out? Where is the deep learning visualization being used? As we are interested in designing an interactive and scalable visualization framework, we summarized some previous visualization works from the aspect of answering two non-mutually exclusive questions: What is the relevant content to be visualized in deep learning? When can these visualizations be carried out?

A. Visualize learned features

The first type of visualization content shows what features are most helpful in distinguishing different classes. Deconvnet[14] and Guided propagation[15] reverse activations back to features in the original input spaces to demonstrate the learned features in higher layers. Deep visualization tool[16] and ConvNets[17] use back-propagated gradients to generate expected features that have the most class discriminant power. Methods in this category visualize activations to provide concrete illustration of class discriminant features. Most of them visualize saved activation files after training.

B. Visualize computation graphs

Another type of visualization efforts helps to design and examine the network architectures. TensorFlow Graph Visualizer[18] provides the visualization of network structures as mental map of data flows and low-level operations. These methods visualize network structures from programming code before training.

C. Visualize weight parameters

The third category visualizes the weight values to depict searching paths through high dimensional weight parameter spaces. The approximate linear paths and the following trajectory of stochastic gradient descent(SGD) is studied by[19] to answer questions such as *Do neural networks enter and escape a series of local minima?* Lorch[20] visualizes the learning trajectory with the principle components(PCA) of weight parameters. These methods visualize the learning trajectories with saved weight files after training.

D. Comprehensive visualization frameworks

Many comprehensive frameworks visualize all the above contents. Glorot and Bengio[21] visualize statistical distributions of weights, activations, gradients and loss functions. CNNVis[22] applies clustering algorithm to analyze the roles of neurons according to their averaged activations. TensorBoard[23] uses histograms to visualize statistical distributions of loss function, weight parameters and other variables defined in TensorFlow[24] framework. It provides graph tools to visualize computation graphs and operations. Both TensorBoard Embedding Projector and Rauber et.al[25] apply projection techniques like t-SNE[26] or PCA to visualize the data of interests. Although the above methods provide users with interactive visualizations, they need to save summary data files during training and the visualizations are processed after training. The amount of summary data is huge for large networks and long training process. To addresses this challenge, ActiVis[27] uses sampling(around 1000 samples) and CNNVis applies clustering algorithm to visualize only part of the neurons and their connections.

Our previous work[10] provides a case study of visualizing the dynamics of a small convolutional neural network using Paraview to process the saved data. However, it takes long response time to visualize a pre-trained VGG16 network due to the heavy I/O overhead(For example, we need to render $batchsize \times 224 \times 224 \times 64 = batchsize \times 3$ million points per time step to visualize the feature maps produced

by the first convolutional layer of VGG16 network). Compared with our previous work and other above comprehensive visualization tools, our improved method uses loosely coupled in situ visualization pipeline and VTK polygon data structure to improve the scalability of our previous TensorView open framework. Thus, we can visualize the training of large neural networks in real time.

III. In situ TensorView Open Framework

We describe the details of our open framework in this section. This includes the choice of visualization and deep learning platforms, the target data to visualize and the output and rendering of visualization results.

A. The platforms

We choose the Paraview Catalyst[7] co-processing pipeline and TensorFlow as the building blocks of our open framework. Figure2 shows the architecture of the in situ TensorView open framework. In situ Tensorview can directly render the visualization results from data coming through the Catalyst co-processing pipeline. So it scales when visualizing large neural networks. Users also can save data of interest in ascii (.csv) or binary (.vtp) format for later use.

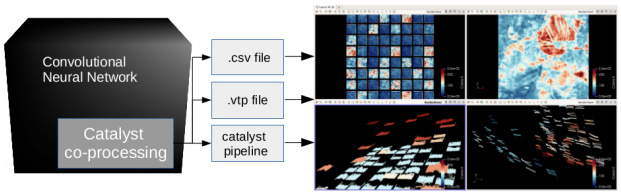


Figure 2: The architecture of our open framework. Then users can choose to output .csv or .vtp file or use the Catalyst co-processing pipeline to perform in situ rendering on a Paraview client.

B. The visualization content and their representations

In this section, we summarize the contents and their representation in our previous work as the background. The in situ TensorView framework utilizes Paraview's color maps and 3-dimensional models to visualize four types of content. They are weight windows in grid layout, trajectories of weight changes, distribution view of activations, and image view of activations.

1) Grid of weight windows: We directly display the changes of weight values in convolutional filters in the layout of a grid of small windows. Each convolutional filter has a window of size $w \times w$. Each convolutional layer has $c \times f$ number of such windows. c and f are the number of color channels and the number of filters respectively. We arrange all $c \times f$ filter-windows into a grid layout. This is the *weight-grid*. Figure3 shows this grid of weight windows

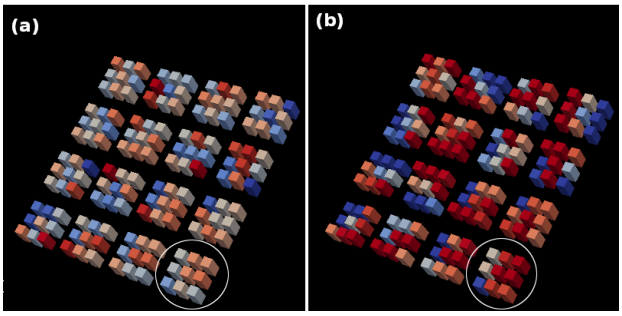


Figure 3: The weight-grid of $conv_1$ in simplified LeNet-5. (a)The weight-grid at initial state. (b)The weight-grid after 8 epochs of training.

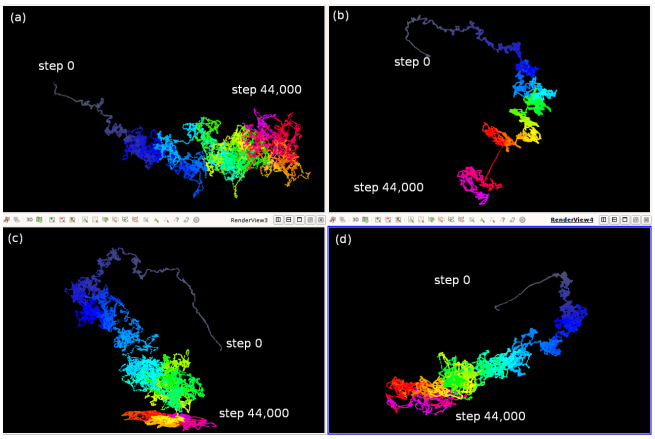


Figure 4: 4 trajectories of different subspaces of weight parameters in $conv_1$ of simplified LeNet-5. $Step_0$ is gray, $Step_{44,000}$ is pink.

of a convolutional layer. For this particular example, the 16 filter-windows are arranged into a 4×4 grid. Each filter-window has only one grey color channel and is represented by $3 \times 3 = 9$ colored blocks. Blue blocks represent negative weights and red blocks represent positive weights. Red blocks are higher than blue blocks along the z-axis in this 3-D representation. This helps users to see the changes of weights more clearly.

2) Trajectories of weight changes: We depict the changes of weight parameters like a trajectory through the high dimensional spaces. Figure4 shows 4 trajectories. We use 3 arbitrary dimensions of the weight parameters as our x,y and z coordinates. Then we use gray to pink colors to represent time steps.

3) Distribution view of activations: We flatten each feature map and display them in shapes of probability distributions. We compute Pearson's correlation coefficient(PCC) between feature maps and use colors to represent similar convolutional filters. Figure5(a) shows 16 feature maps of one training batch from the first convolutional layer of simplified LeNet-5. Each feature map is flattened and the

activation values are normalized to between $[0, 1]$. The blue group contains 3 feature maps have high Pearson's correlation scores. The green group contains 2 feature maps that are also similar to each other.

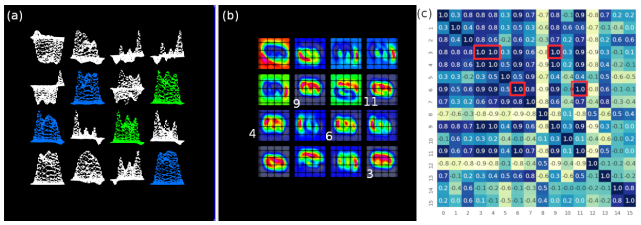


Figure 5: Visualizing feature maps from the first convolutional layer of simplified LeNet-5. (a)Distribution-view. (b)Image-view. (c)Heatmap of Pearson's Correlation Coefficients(PCC). Darker blue represents higher correlation values.

4) Image view of activations: We directly display activations as images. Combining the statistics view and the image view, users can find out redundant neurons in their networks. Figure5(b) shows the same 16 feature maps as the above example. Their sizes are 28×28. Users can find they resemble some input images.

C. Output format and rendering

The in situ TensorView open framework provides flexible output methods that work for multiple data format supported by the Paraview. The most convenient method is to use ascii files because Paraview can directly import *.csv* files. Users can easily build the output files and use them in other visualization tools like Matlab or Python Matplotlib. However, the I/O overhead makes this format not scalable for visualizing large networks. For example, it takes 85 Megabytes storage and 60 seconds to load feature maps from the first convolutional layer of VGG16 into Paraview client. The VTK polygon data format is more efficient to load. The feature maps mentioned above uses 35 Megabytes disk storage in .vtp format. It only takes 2-3 seconds to load them into Paraview client. It is a good choice for large and complex neural networks and if users want to save the data for later analysis.

The Catalyst co-processing pipeline is great for in situ analysis. The data directly goes through the socket connection to the Paraview client. It essentially uses the same VTK polygon data structure but entirely avoids the overhead of file I/O. Thus users are able to see the visualization results in real time.

IV. Case Studies

We use two case studies to illustrate how users can use our open framework to visualize their neural networks. First, we visualize the training process of a simplified LeNet-5. Second, we visualize the predicting process of a pre-trained VGG16 network. In both cases, our visualization open framework can help users to find out redundant convolutional filters. We only visualize the weights and activations from the first convolutional layers. This is because the feature maps produced in this layer are larger than feature maps produced by later layers. Also, because this layer is at the end of the back-propagation path, we are more likely to observe here the problems of gradient vanishing or explosion.

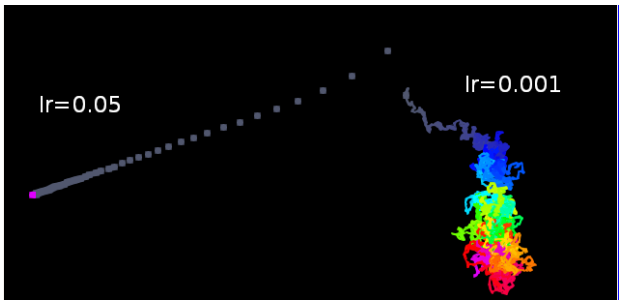


Figure 6: Left: Gradient vanished due to inappropriate learning rate. Right: A normal learning process.

We build the in situ TensorView open framework with Catalyst co-processing libraries on top of the Tensorflow framework. Then we use use Paraview client 5.2.0 to render and visualize the data coming through the co-processing pipeline. Experiments were carried out on servers with Intel Xeon CPUs of 2.40GHz and 128G RAM, accelerated by Nvidia P100 GPU cards.

A. Simplified LeNet-5

We simplify the LeNet-5 to half of its original size. It has 16 and 32 filters in convolutional layers. The filter size is reduced to 3×3. The number of neurons in the fully connected layer is also reduced to 512. With the visualization results from in situ TensorView, we find out activations of some filters are similar. When the learning rate is too high, both the grid of weight windows and the trajectory of weight changes stop to change.

1) Visualization of weights: Fig. 3 shows the grid of weight windows of convolutional layer$_1$. Users can see the colors of red and blue blocks grow darker as the training continues. The colors stop to change within a short time if the learning rate is too high. Figure 4 presents 4 trajectories of weight parameters from the first convolutional layer. Along the z-axis, the paths go down several steps. This indicates the neural network is searching around one local minimum then finds a path to jump to another local minimum. Figure6 shows two trajectories. The right side trajectory is the same as in the Figure 4(a). The left trajectory is very straight. This is because the learning rate is too high and the network stops learning due to gradient vanishing.

2) Visualizing activations: Figure 5(a) shows the distribution-view and image-view of activations from the

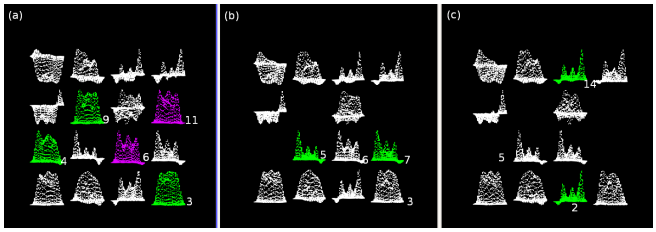


Figure 7: (a)At step 1080, filters {3,4,9} and filters {6,11} are similar. (b)At step 1680, filters {4,9} and {11} are removed. A new group contains filter {5,7} are marked with green color. (c)At step 2220, filter {7} is removed and {5} is kept. A new group contains filters {2,14} is green.

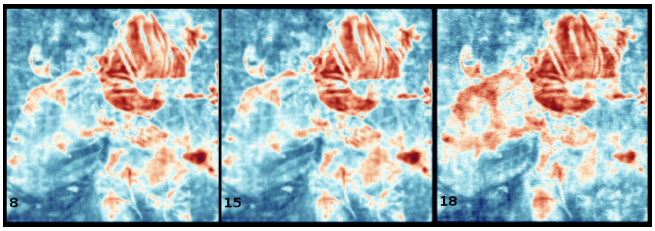
first convolutional layer in the simplified LeNet-5. According to Pearson's Correlation Coefficient values between filters, we put filter 3,4 and 6 into a blue group and filter 6 and 11 into a green group. With this visualization results, users can identify redundant convolutional filters and prune similar neurons during the training process. A detailed pruning method is described in our previous work [10]. Figure7 shows the distribution-view in real time. After 5 epochs, we reduced 16 filters to 12. The network stays stable after that. After 40 epochs of training, the pruned network keeps 99.17% accuracy.

B. Pre-trained VGG16 network

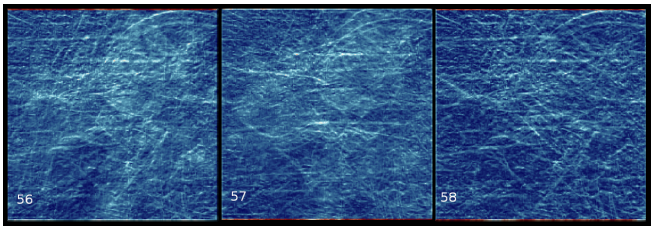
As transfer learning becomes prevalent, it has become standard practice to reuse pre-trained convolutional layers and only fine-tune the fully connected layers. Thus, our second case study visualizes the feature maps from a pre-trained VGG16 network. Figure1 shows the image-view,distribution-view and two feature maps. The visualization results indicate that some convolutional filters are redundant.

1) Image-view of feature maps: The 64 feature maps contains $64 \times 224 \times 224 = 3$ million pixels. So we convert the feature maps into the VTK polygon data structure to render the image-view quickly. In Paraview client, users can zoom in and check the images in detail. Figure8a shows filters {8,15,18} produce similar feature maps from one batch of training data. Users can easily find two of these three filters are redundant. However, Figure8b shows some texture feature maps. It is difficult for users to check whether the corresponding filters {52,53,54} are similar or not. In this case, the distribution-view provides better comparison of feature maps.

2) Distribution-view of feature maps: Figure9 display the distribution-view of the feature maps from the first training batch. The two groups {8,15,18} and {32,61} marked in figure9 can be confirmed in the image-grid visualization(previous Figure8a). We can check the distribution-views of filters {56,57,58}(The 3 filters at the upper left corner of Figure9). They have obvious differences and they are not in one group.



(a) Zoom in feature maps of filter {8,15,18}



(b) Zoom in feature maps of filter {52,53,54}

Figure 8: (a)Filter8, 15, 18 produce similar feature maps. (b)Texture feature maps are difficult to compare.

Combining the image-view and distribution-view, users can find out the similarities and decide if they want to compress and retrain the network. We did not experiment on merging similar filters and retraining this pre-trained VGG16 network. Because the activations of similar filters approximate to each other, we expect the retraining of the pruned network can converge quickly in a few epochs.

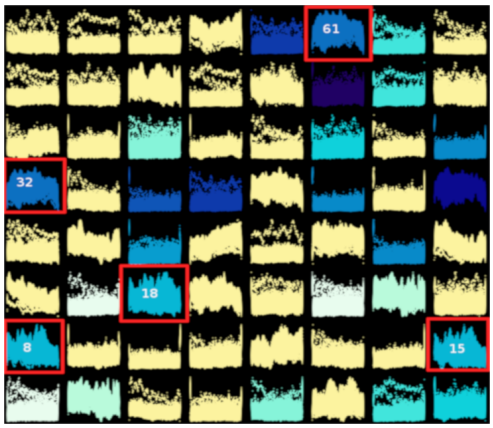


Figure 9: The distribution-grid shows accumulated feature maps of the first training batch at VGG16's convolutional layer_1. Distribution-window of {8,15,18} are quite similar. {32,61} are similar.

V. Conclusion

In this paper, we present the in situ TensorView open framework. This open framework supports multiple output formats to visualize neural networks of different complexities. In situ TensorView leverages the scalability of Paraview Catalyst co-processing library to visualize and analyze

the training and predicting of large Covolutional Nueral Networks. It provides flexible views that visualize weight parameters and the activations of convolutional filters. These views include the grid of weight windows, the trajectories of weight changes, distribution-view of activations and image-view of activations. With these views, users can observe the dynamics of their networks during training and predicting phases. The Pearson's Correlation Coefficient encoded by group colors in the distribution-view can help users to decide whether some of the convolutional filters are redundant. The visualization results can guide users to prune or compress their networks.

We present the visualizations in two case studies. They are a simplified LeNet-5 network and a pretrained VGG16 network. The experiments show that in situ TensorView can visualize both small and large convolutional neural networks. It can help users to visualize the networks in cases of training a network from scratch or use some pretrained networks to make predictions.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[3] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR*, vol. abs/1409.1, 2014.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[5] K. He and J. Sun, "Convolutional neural networks at constrained time cost," in *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE, 2015.

[6] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143.

[7] J. Ahrens, B. Geveci, and C. Law, "36-paraview: An end-user tool for large-data visualization," *The visualization handbook*, vol. 717.

[8] H. Childs, E. Brugger, B. Whitlock, J. Meredith, S. Ahern, D. Pugmire, K. Biagas, M. Miller, G. H. Weber, H. Krishnan *et al.*, "Visit: An end-user tool for visualizing and analyzing very large data," *High performance visualization–enabling extreme-scale scientific insight*, 2012.

[9] J. Ahrens, S. Jourdain, P. O'Leary, J. Patchett, D. H. Rogers, and M. Petersen, "An image-based approach to extreme scale in situ visualization and analysis," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014.

[10] X. Chen, Q. Guan, X. Liang, L.-T. Lo, S. Su, T. Estrada, and J. Ahrens, "Tensorview: Visualizing the Training of Convolutional Neural Network Using Paraview," in *Proceedings of the 1st Workshop on Distributed Infrastructures for Deep Learning*, ser. DIDL '17. New York, NY, USA: ACM, 2017, pp. 11–16. [Online]. Available: http://doi.acm.org/10.1145/3154842.3154846

[11] J. Kress, S. Klasky, N. Podhorszki, J. Choi, H. Childs, and D. Pugmire, "Loosely Coupled In Situ Visualization: A Perspective on Why It's Here to Stay," in *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, ser. ISAV2015. New York, NY, USA: ACM, 2015, pp. 1–6.

[12] W. J. Schroeder, B. Lorensen, and K. Martin, *The visualization toolkit: an object-oriented approach to 3D graphics*. Kitware, 2004.

[13] F. M. Hohman, M. Kahng, R. Pienta, and D. H. Chau, "Visual Analytics in Deep Learning: An Interrogative Survey for the Next Frontiers," *IEEE Transactions on Visualization and Computer Graphics*, 2018.

[14] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[15] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806*, 2014.

[16] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," *arXiv preprint arXiv:1506.06579*, 2015.

[17] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[18] K. Wongsuphasawat, D. Smilkov, J. Wexler, J. Wilson, D. Mané, D. Fritz, D. Krishnan, F. B. Viégas, and M. Wattenberg, "Visualizing Dataflow Graphs of Deep Learning Models in TensorFlow," *IEEE transactions on visualization and computer graphics*, vol. 24, no. 1, pp. 1–12, 2018.

[19] I. J. Goodfellow, O. Vinyals, and A. M. Saxe, "Qualitatively characterizing neural network optimization problems," *arXiv preprint arXiv:1412.6544*, 2014.

[20] E. Lorch, "Visualizing Deep Network Training Trajectories with PCA," in *The 33rd International Conference on Machine Learning,JMLR volume 48*, 2016.

[21] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[22] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu, "Towards better analysis of deep convolutional neural networks," *IEEE transactions on visualization and computer graphics*, vol. 23, no. 1, pp. 91–100, 2017.

[23] Google, "TensorBoard: Visualizing Learning," https://www.tensorflow.org/get_started, 2015.

[24] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, and Others, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[25] P. E. Rauber, S. G. Fadel, A. X. Falcao, and A. C. Telea, "Visualizing the hidden activity of artificial neural networks," *IEEE transactions on visualization and computer graphics*, vol. 23, no. 1, pp. 101–110, 2017.

[26] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of machine learning research*, vol. 9, no. Nov, 2008.

[27] 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.