# TensorViz: Visualizing the Training of Convolutional Neural Network Using Paraview

**Xinyu Chen**[1,3], **Qiang Guan**[3], **Xin Liang**[2,3], **Li-Ta Lo**[3], **Trilce Estrada**[1] **and James Ahrens**[3]

1. University of New Mexico, USA
2. University of California, Riverside, USA
3. Los Alamos National Laboratory, USA

LA-UR-17-26748

Correspondence: xychen@cs.unm.edu

## Problem

Deep Convolutional Networks have been very successful in visual recognition tasks recently. Lots of previous works aimed to help people to get senses of why those biology-inspired networks achieved such good performances. Deconvnet[1], Guided propagation[2] and a comprehensive visualization tool box[3] can help people to see features learned at different layers of the networks. These works in some extent provided understanding and support for the biology origin of how convolutional networks emulate visual recognition tasks. However, due to the complexity of searching in very high dimensional parameter space, the whole training remains in black-boxes. Normally a large network needs weeks of training on high-end graphic cards. Fine-tuning of hyper-parameters like the learning rate, depth and width of the network still depends on previous successful architectures or trial and error. In this poster, we study the network as a dynamic system and its learning process as the evolution of parameters. By visualization of the development and evolution of network, we aim to provide facilities to find optimal hyper-parameters.

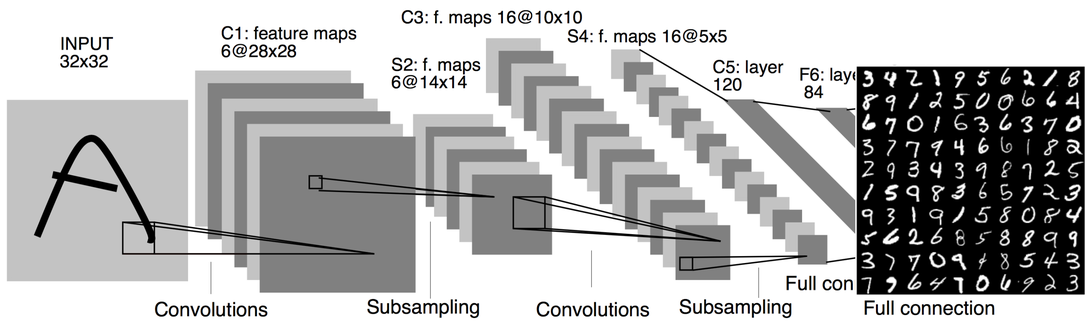


**Figure 1:** (left)We treat the Convolutional Networks (similar to LeNet-5[4]) as a dynamic system. We visualize the evolution of weights of two convolution layers during the training. The first and second convolution layer contain 16 and 32 $3 \times 3$ filters respectively. The fully connected layer contains 1024 neurons. (right)The network learns MNIST dataset which contains $55,000$ training samples, $5,000$ validation samples and $10,000$ testing samples.

## Concepts, Dataset and Tools

Some basic **concepts** are:
- Neuron: A unit that receives values from predecessors and outputs a value to successors.
- Weight: A value that defines how units are connected.
- Activation: The value computed from a non-linear function.
- Loss: The difference between the network's prediction and true value.
- Gradient: A value indicate how to adjust weights during training.
- Convolutional Networks(Fig.1): networks with convolutional and fully connected layers.

We built simple convolutional networks with different hyper-parameters. The architecture is similar to the LeNet-5. The networks were trained on the **MNIST dataset** to learn handwritten digits. They can achieve $99.2\%$ accuracy after about 16 epochs of training.

We use **Paraview** and **Python matplotlib** to visualize the evolution of weights, gradient, activation and loss.

## Visualize Weights

Fig.-2 From left to right, Paraview shows the evolution of the weights of $16$ $3 \times 3$ first layer convolution filters in one of the networks at 0, 4k and 8k steps. Each box represents a weight and each $3 \times 3$ block represents a filter. The height along z-axis represents the numeric value of weights. To make the changes more obvious, we also colored the weight. Red is more positive, blue is more negative.

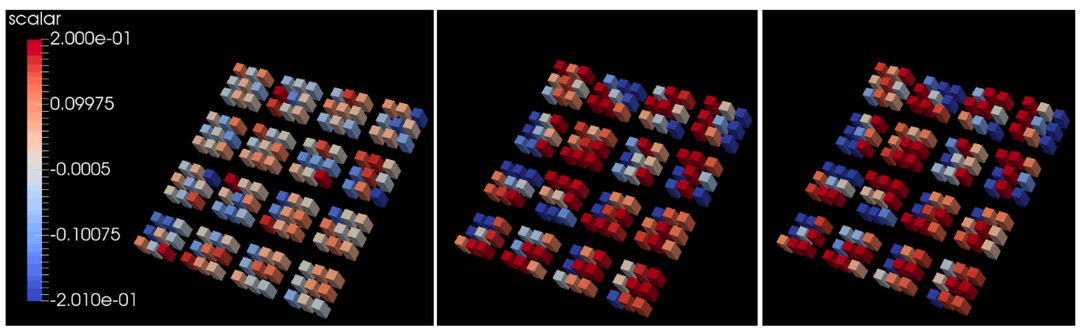


**Figure 2:** Visualization of first layer convolution filters in Paraview.

We did not map convolution filters back to image patches like Deconvnet[1] or Guided propagation [2], but display the process of their evolution. In cases when learning rate is too high, the adjustment of weights would stuck within a short period of time. By visualizing the changes of convolution filter weights, we observed that colors tends to become darker during training. Boxes seldom change from blue to red or vice versa. This leads us to the visualization of their L2norms.

## Visualize L2norms

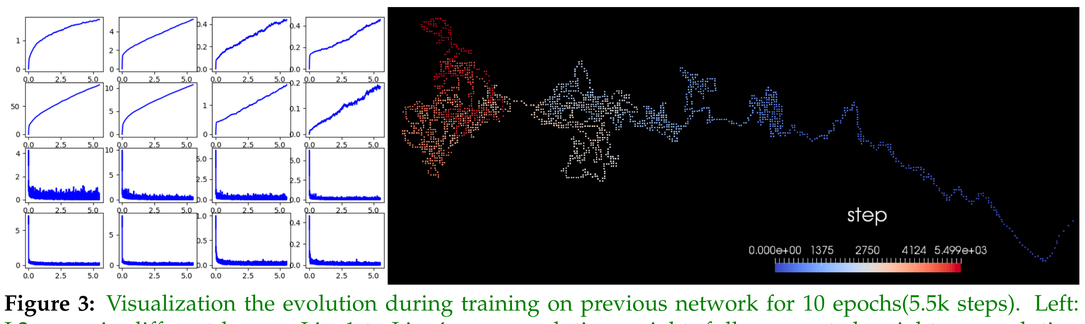


**Figure 3:** Visualization the evolution during training on previous network for 10 epochs(5.5k steps). Left: L2norms in different layers. Line1 to Line4 are convolution weights,fully connected weights, convolution gradients and fully connected gradients. Right: Visualization of first 2 dimensions of convolution weights. It looks like a random walk.

Fig.-3 shows an alternative view of weights and gradients. We concatenate weights and treat them like long *weight vectors* in each layer. In left graph, line1 and 2 show L2norms' growth roughly related with the square root of time steps. Line3 and 4 show gradients are roughly following some random distributions. The right graph roughly describes how a weight vector searches the parameter space. Although we only use the first 2 dimensions to show positions of a weight vector, it suffices to demonstrate the searching process. The color indicates time steps in training. We can examine our observation against how networks update their weights [4].

$$W(t+1) = W(t) - \eta \frac{\partial E^t}{\partial W} \tag{1}$$

## Visualize Activations

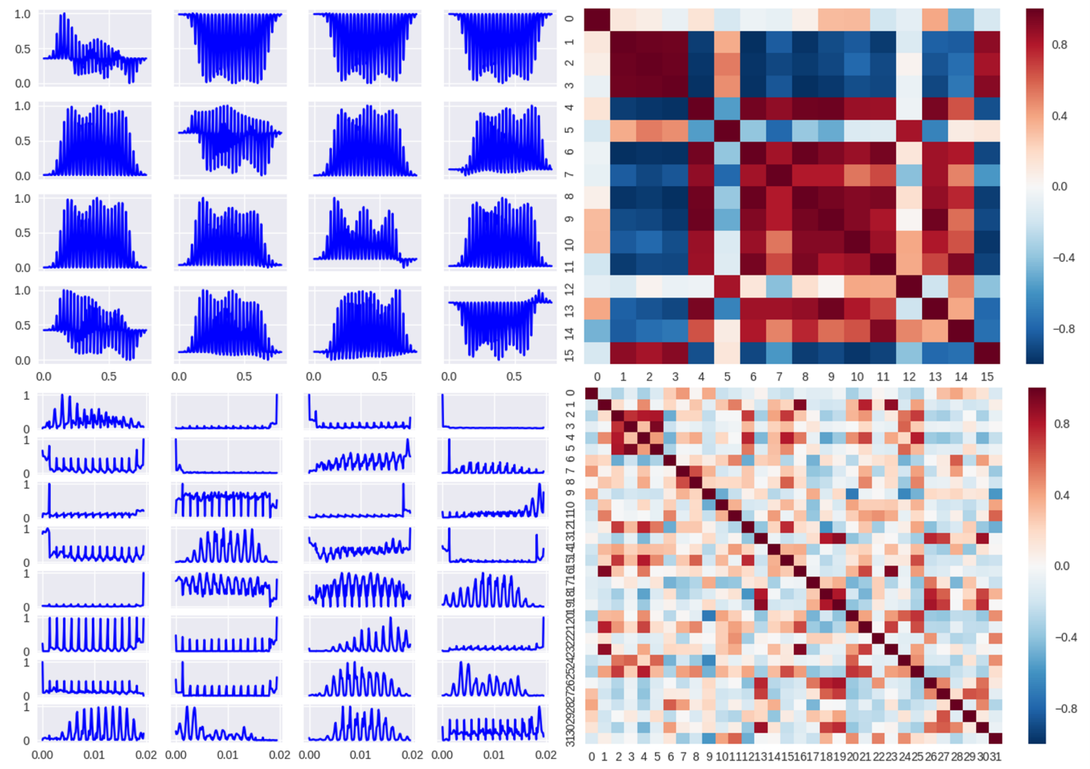

**Figure 4:** Top left: Activation of first layer convolution filters. Top right: Heatmap of first layer filters' correlation. Bottom left: Activation of second layer convolution filters. Top right: Heatmap of second layer filters' correlation.

From Fig.-4 we can see similar activations in both convolution layers. This suggests the redundancy of convolution filters. The heatmaps illustrates Pearsons' correlation between convolution filters. The heatmap can be used for reduction of such filters.

## Filter Reduction

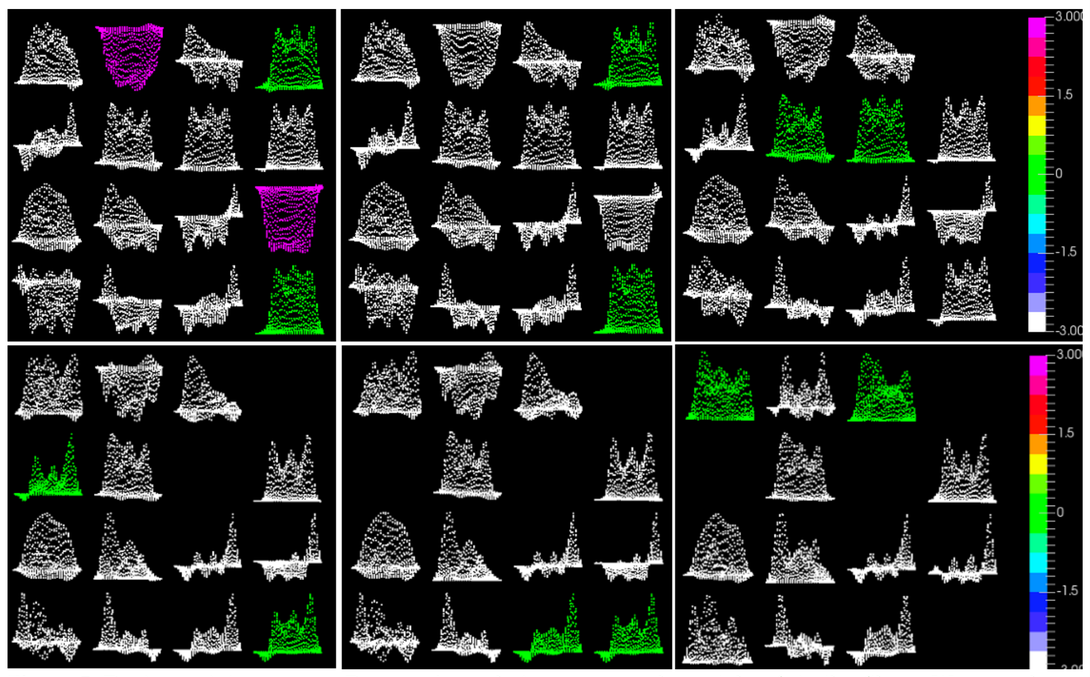

**Figure 5:** During training, we use Pearsons' correlation to merge the weight of similar filters. We use color to indicate similarities. Start from 16 filters in top left, after 20 epochs, we reduced to 12 filters in bottom right.

Fig.-5 shows reduction of similar filters. The accuracy of reduced networks is $99.12\%$, remains the same as the original network. In learning the relative simple MNIST dataset, the reduction of filters does not affect training time. However, we expect a more beneficial effect in larger networks.

## Summary

In this poster, we presented several attempts in using Paraview to visualize and analyze the training of deep convolution networks. We have found interesting evolution of weights and gradient. Finally, the visualization of activation encourages us to merge similar filters and reduce the architecture during the training. Because Paraview supports in-situ analytics, it is possible setup an interactive mechanism to facilitate the training of more complex networks in the future.

## Acknowledgments

The authors would like to thank the Data Science at Scale summer school in the Los Alamos National Laboratory.

## References

[1] Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European conference on computer vision. Springer, Cham, 2014.
[2] Springenberg, Jost Tobias, et al. "Striving for simplicity: The all convolutional net." arXiv preprint arXiv:1412.6806 (2014).
[3] Yosinski, Jason, et al. "Understanding neural networks through deep visualization." arXiv preprint arXiv:1506.06579 (2015).
[4] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.