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

Xinyu Chen
University of New Mexico
xychen@cs.unm.edu

Qiang Guan
Los Alamos National Laboratory
qguan@lanl.gov

Xin Liang
University of California, Riverside
xlian007@ucr.edu

Li-Ta Lo
Los Alamos National Laboratory
ollie@lanl.gov

Simon Su
US Army Research Laboratory
simon.m.su.civ@mail.mil

Trilce Estrada
University of New Mexico
estrada@cs.unm.edu

James Ahrens
Los Alamos National Laboratory
ahrens@lanl.gov

## ABSTRACT

Convolutional Neural Networks(CNNs) have been widely used in visual recognition tasks recently. Previous works visualize learning features at different layers to help people to understand how CNNs learn visual recognition tasks. However they only provide qualitative description and do not help to accelerate the training process. We present TensorView to enable Paraview to visualize the evolution of CNNs. TensorView provides both qualitative and quantitative visualization that help understand the learning procedure, tune the learning parameters, direct merging and pruning of neural networks.

## CCS CONCEPTS

• **Human-centered computing → Information visualization**;
• **Computing methodologies → Neural networks**;

## KEYWORDS

Visualization, Convolutional Networks, Paraview

**ACM Reference Format:**
Xinyu Chen, Qiang Guan, Xin Liang, Li-Ta Lo, Simon Su, Trilce Estrada, and James Ahrens. 2017. TensorView: Visualizing the Training of Convolutional Neural Network Using Paraview. In *DIDL'17: DIDL'17: Workshop on Distributed Infrastructures for Deep Learning, December 11–15, 2017, Las Vegas, NV, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3154842.3154846

## 1 INTRODUCTION

Since AlexNet's [9] impressive achievement in the ILSVRC2012, Deep Convolutional Neural Networks (convnets or CNNs) have been used successfully for increasingly complex visual recognition and natural language processing tasks. The state-of-art CNNs, like ResNet [6], contain hundreds of layers and achieve accuracy that outperforms human players. Despite the promising performance in achieving higher and higher accuracy, training a deep convolutional neural network is always challenging because of the complexity in searching the optimal parameters in very high dimensional spaces. The amount of trainable parameters is so large (431k in LeNet-5, 61M in AlexNet) [5] that training a real-world convolutional neural network is expensive. Normally they need weeks of training on high-end GPUs. Besides the great volume of trainable parameters, practitioners also need to carefully choose hyper-parameters such as learning rates, various types of optimizers, the depth and width of networks. Often a hyper-parameter has a critical effect on the length of training time and the accuracy of the network. Sometimes, neural networks fail to converge due to inappropriate hyper-parameters, which are known to cause the gradient vanish or gradient explosion problem.

Without a clear understanding of how and why CNNs work well in visual recognition applications, fine-tuning these hyper-parameters largely depends on previously successful architectures or trial-and-error [16]. Here are some questions we want to answer: Are all these neurons necessary? Can we be confident that there are redundancies and we should remove them? Can we know gradient vanish or explosion problems during training process?

In scientific simulations, visualization techniques have been widely used to help scientists to get better understanding of the results of computations and accelerate scientific discoveries. In this paper, we present TensorView, a visualization tool which leverages Paraview and Matplotlib to study neural networks built upon TensorFlow's framework. We study a neural network as a dynamic system and treat the learning process as the evolution of parameters. A trained neural network is made up of weights, in both convolutional and fully connected layers, activation functions, and gradients. By visualizing the evolution of weights and gradients, we gain a direct observation on how the model consumes and learns from training examples: we are able to see the neurons get updated during training. If gradient vanishes or explodes, we can quickly see the training stops. By visualizing activation functions, we are able to identify similar behaviors among neurons in the convolutional layers. This insight allows us to observe the redundancies inside deep neural network architectures. Thus we can merge and prune

[1]The publication has been assigned the LANL identifier LA-UR-17-27408.


*DIDL'17, December 11–15, 2017, Las Vegas, NV, USA*

ACM ISBN 978-1-4503-5169-0/17/12…$15.00
https://doi.org/10.1145/3154842.3154846

similar neurons while training is in progress. Our contributions are: (1) using visualization to inform researchers in real time regarding their hyper parameter selection, and (2) using quantitative information to prune unnecessary neurons, speeding up the training process.

Provide both qualitative visualization and quantitative insight to facilitate the adaptive selection of hyper-parameters and speedup the training process. Our goal is to reduce the training time and memory usage during the training and visualize the process of learning and the evolution of neural network architectures.

The rest of the paper is organized as follows. In section 2, we discuss the related work. Some background concepts about convolutional neural networks are covered in section 3. We explain the methodology in section 4. Visualization results are shown in section 5. On-line pruning is inspired by the visualization of activation. Thus the evolution of adaptive neural network architectures is presented in section 6. And we conclude the paper in section 7.

## 2 RELATED WORK

Traditionally, neural networks have been treated as black-boxes, where learning is hard to interpret and many times it is even impossible to explain. This lack of understanding results in computationally expensive and time consuming trial-and-error designs. Previous work have tried to help people to get a better understanding of why those biology-inspired networks achieve such good performance. Among them, Deconvnet [16], Guided propagation [13] and Deep Visualization tool box [15] have successfully helped people to understand the learned features at different layers of the networks. These works greatly enhance our understanding of how convolutional neural networks emulate visual recognition tasks. Their visual feature maps provide a qualitative illustration of the model. From such feature maps we can identify edges, corners and color patches, usually in the first convolutional layer of almost every CNN. Yet, it is hard to find distinguishable differences between such feature maps and Gabor filters [4] of different networks. The feature maps do not provide us with information regarding the suitability of hyper-parameters or the training process status.

Other attempts depict trajectories of training through the solution spaces [11], [3]. Yet their purpose is to provide information regarding the influence of parameters in the search task, and they do not help in accelerating the training process.

Another visualization research direction goes to providing quantitative analytics of neural networks. Histograms of weights, gradients and losses are used to visualize their distribution [2]. Tensorboard [7] can visualize such distribution for networks built upon TensorFlow. This can help to learn the effect of different hyper-parameters. But most of the visualizations and analytics in Tensorboard are post-training statistics. Tensorboard lacks on-line analysis and tuning during training.

Besides Tensorboard's post-training visualization tools, some methods that post-prune the networks have been proposed to address energy and memory concerns Denton et al. [1] [5]. To the best of our knowledge, there is no on-line tools that can provide both qualitative visualization and quantitative insight to facilitate the adaptive selection of hyper-parameters and speedup the training process.

Paraview [8] is an open-source data analysis and visualization application, which is used in large scale scientific simulations. It has the capabilities of providing interactive and in-situ analysis on extremely large datasets using distributed memory computing resources. Paraview Catalyst provides abundant interfaces to support data analysis and visualization for scientific simulation. However it does not support visualization in machine learning so far. We are the first to leverage Paraview's capabilities to visualize the training of neural networks.

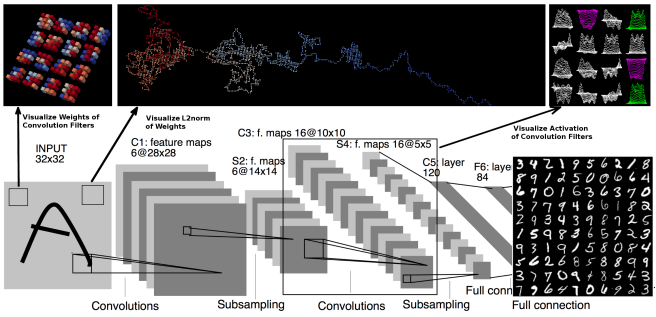



**Figure 1: Outline of TensorView. Bottom network: the testing architecture of a convolutional neural network. We built a LeNet-5 with 16 and 32 filters in the convolutional layers and 512 neurons in the fully connected layer. The CNN is trained to classify the MNIST dataset. Top left: Visualization of the filters' weights. Top middle: Visualization of L2-norms of weights. Top right: Visualization of activation functions.**

## 3 BACKGROUND

We briefly cover some background concepts on the architecture of convolutional neural networks because they are important for our design decisions.

Artificial neural networks use massive amount of connected neurons to solve complex computation problems. Artificial neurons are computing cells that are organized into layers. Each neuron performs a nonlinear *activation function* on the weighted sum of its inputs and sends the output to neurons in the next layer. Such outputs are also called activations. The connections between neurons are represented by *weight values*. The currently most widely used activation function is Rectified Linear Units (ReLU) which is $relu(x) = max(0, x)$.

Convolutional neural networks are one specific type of neural networks that are very good at recognizing local structures in vision and speech data. Besides the typical fully connected layers, they have convolutional layers . A neuron in the convolutional layer has a limited receptive field. For image recognition problems, the input field normally has a 2D layout represented as a window with size of 5×5 or 7×7 pixels. Sliding such a window over the input images, the neurons perform the convolution operations to extract local features such as edges or corners. Thus such neurons are also known to be *filters*. The outputs of the these convolution operations are feature maps which also have 2D layouts. In section 4, we describe the method to visualize the weights and activations.

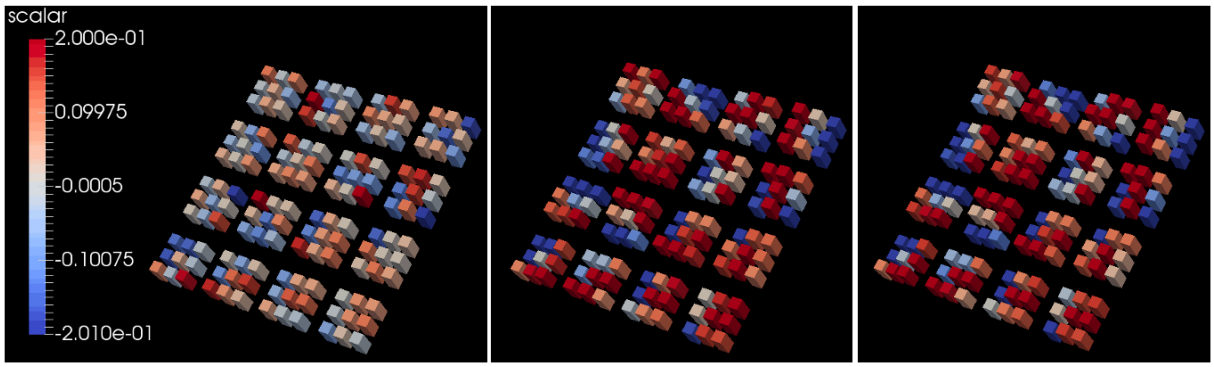



**Figure 2: Convolution filters in layer-1 at 0, 4k and 8k steps. Colored cubes represent weights. Red is more positive, blue is more negative. The elevation along z-axis represents weight values too.**

## 4 METHOD

Our method is inspired by the visualization of scientific simulations. During training, the evolving weights, activations, losses and back-propagated gradients [10] are informing us about the training status. In the forward pass, the first convolutional layer provides input for successor layers. On the other hand, the first layer is the last one that gets updated in the backward pass. If the learning is affected by gradient vanish or gradient explosion issues, we can easily observe the problems at the first convolutional layer. In the following sections, we focus on the visualization of the first convolutional layer.

For the purpose of proof of concept, we use a network similar to the LeNet-5 to learn the MNIST dataset (Fig. 1). Our CNN has 16 filters in convolutional layer$_1$ and 32 filters in convolutional layer$_2$. The filter size is $3 \times 3$. For the sake of simplicity, we reduced the number of neurons in the fully connected layer to 512. Even with the above simplifications, this CNN achieves 99.17% accuracy after 40 epochs of training. To visualize neural networks as dynamic systems, we display the evolution of weights, gradients and activations during the training process.

We use colored cubes to represent weight values in visualization of convolutional filters. In the following example, each filter has a window size of $3 \times 3$ pixels. So each of them consists of 9 cubes. The actual convolutional filters defined in TensorFlow do not have explicit indices for their weights, we need to assign *x,y* coordinates to those cubes to align them into blocks and grids for visualization. To emphasize the evolution of weights, we map the weight values to colors and *z* coordinates at the same time. First, blue color represents negative weight values and red color represent positive weight values. Second, cubes with positive weight values also looks higher than cubes with negative weight values along z-axis. This double-mapping enables us to see the color of these cubes changes and their *elevation* along z-axis moves as the weights get updated after each time-step during training.

We also use colored points to illustrate trajectories of learning process. In this case the color represent time steps. As the training goes on, the color changes from blue to red. In section 5, we present the visualization of weights, trajectories, L2-norms and activation of the convolutional filters in the first convolutional layer.

## 5 VISUALIZATION RESULTS

We use Paraview 5.2.0, Python Matplotlib for the visualization. The convolutional network is built on the TensorFlow framework. All experiments were carried out on the Darwin cluster in the Los Alamos National Laboratory. We chose Nvidia Tesla K40m Accelerator.

### 5.1 Visualizing Weights

In section 4 we described the method to assign *x,y,z* coordinates and colors to display weights in Paraview. This is the first approach that we use to visualize the training of a model qualitatively. In Fig. 2 we arrange the 16 filters in convolutional layer$_1$ into $4 \times 4$ blocks. Although we display filters separately, we do not intend to map them back to image patches like Deconvnet [16] or Guided propagation [13]. Paraview can read time-series data from files and render one image for each time step. Thus, we display the evolution of weights as animations. In cases when learning rates are too high, it is easy to see that the evolution of weights get stuck within a short period of time.

### 5.2 Visualizing learning trajectories

From the visualization in Figure 2, we notice that starting from an initial color, the cubes tend to keep their colors. That is, the blue ones tend to become darker in blue and the red ones tend to grow darker in red. They seldom change from blue to red or vice versa. This reflects that a neural network follows certain searching path during the training process. When looking at the *elevation* along z-axis, we observed the same trend. From initial positions, blue cubes are lower than red cubes in the z-direction. Then the blue cubes keep falling lower and red cubes keep rising higher during training. To better understand this evolution, we visualized the trajectories of weight values over time.

Our second qualitative visualization is based on the above observations. Figure 3 presents trajectories of 4 neural networks with different learning rates. To display these trajectories, we take the first 2 weight values from one filter as if they are x,y coordinates. Use these x,y coordinates we can draw points in a 2D plane. The color of points indicates time steps (from blue to red). We annotate learning rates beside the corresponding trajectories. The right most

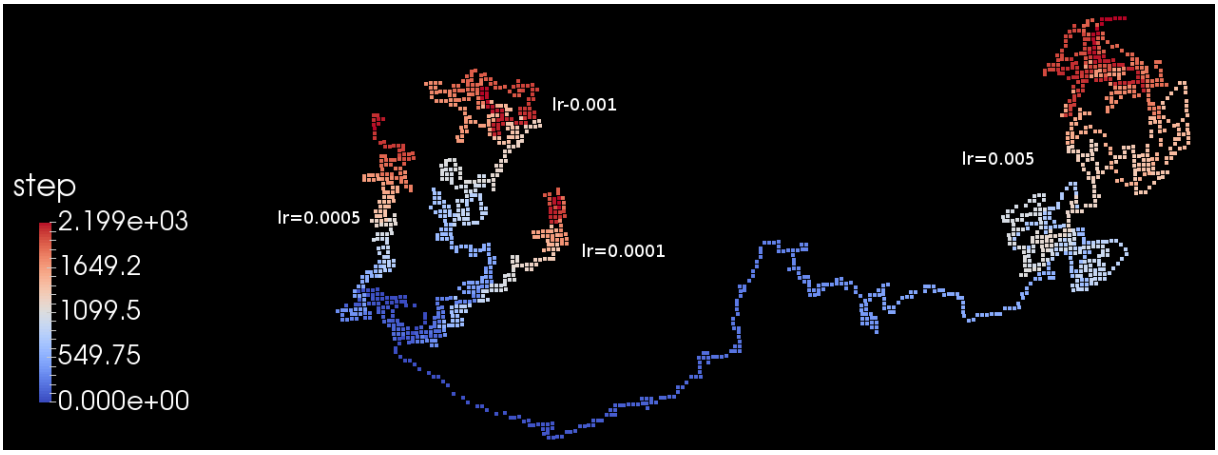



**Figure 3: Visualization of 4 trajectories using the first 2 dimensions of convolution weights. Color represents time steps, starts from 0 as blue to 2199 as red. Each trajectory corresponds to one neural network with different learning rates.**

trajectory reflects a network with the largest learning rate(learning rate=0.005). It goes longer distance than the other three trajectories with smaller learning rates. These trajectories suggest that within a short period of training, neural networks are searching along some determined directions. But after that, neural networks start to behave like random walks. In our example, all 4 neural networks with different learning rates start to behave like random walks after about 100 steps of training. The above observations are similar to a recent work by Eliana Lorch [11] using PCA to reduce the high dimensional weight vectors to 2 or 3 dimensions for visualization purposes and get training trajectories. We visualized several pairs of weights from the convolutional filters, they all show similar patterns so we only plot the trajectories of the first two weight values here.

## 5.3 Visualizing L2-norms

The equations of adjusting weights [10] in back-propagation training , also support our observation that the searching path follows a random walk model. In this equation, $W(t)$ is the weight at time step $t$. $\eta$ is the learning rate and $\frac{\partial E}{\partial W}$ is the back-propagated gradient.

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

To see how the trajectories related with random walks, we concatenate weights in each layer into long *weight vectors* and subtract their initial values. Then we compute their L2-norms for each time step to see the distance between the end point of *weight vectors* and their start positions. Similarly we computed the L2-norms for bias vectors and gradient vectors for each layer. The distances are shown in Figure 4 (a) and (b). In the first row, we visualized the L2-norms of weight vectors in the two convolutional layers and two fully connected layers(from left to right). In the second row, we visualized the L2-norms of bias vectors of the above four layers. The third row corresponds with the L2-norms of gradients to update the weight vectors in the four layers. The fourth rows corresponds to the L2-norms of gradients to update the bias vectors in the four layers.

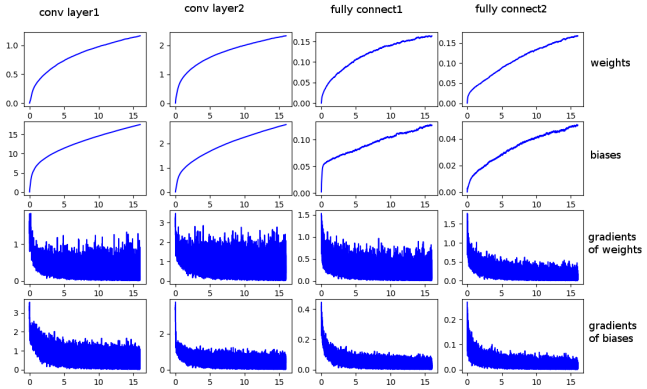
In Figure 4 we compared two neural networks with different learning rates. The graph(a) displays a neural network with a smaller learning rate. The L2-norms of weight vectors and bias vectors grow during training. They display curves that are similar to square root curves. On the other hand, the gradient vectors look more like random signals. The graph(b) display a neural network with a bigger learning rate. In this case the L2-norms of weight vectors and bias vectors saturated in a short period. We can see the gradient vectors vanish quickly. The visualization of trajectories and L2-norms can be used as a qualitative indicator of training status.

## 5.4 Visualizing Activations

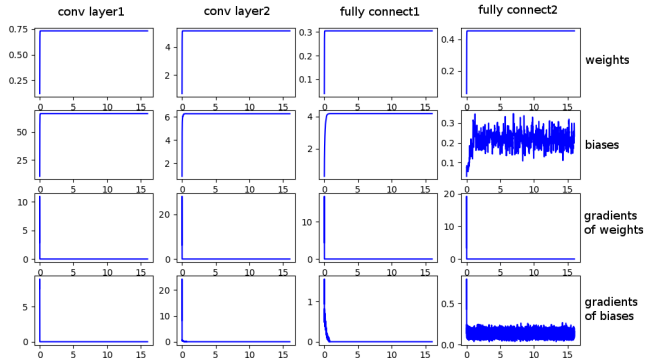
The previous three experiments provide qualitative visualizations about the models. They are useful to show how learning rates affect training. In this section, we also visualize the activation of filters in convolutional layers as a quantitative indicator to show how other hyper-parameters, such as the number of filters, affect training.

Although activations are actually 2D feature maps, we do differently from previous methods that display activations as images. We flatten the 2D feature maps instead. In our example, they are flattened to 1D vector with 784 elements. Each element corresponds to one pixel in the feature map. Because the ReLU activation function ($relu = max(x, 0)$) outputs non-negative values, we simply sum all element-wise activation values and normalize them to [0,1]. We can use the normalized activations to compare the similarities between filters because later we will merge them together instead of throwing away any of them. Then we plot those 1D normalized activation vectors to compare the accumulated activation values after the neural networks have learned all training sample images for each epoch. The accumulated activation values are simple to compute and they do well to indicate similarities between convolutional filters.

Fig.5 shows activations in both convolutional $\text{layer}_1$ (16 filters) and $\text{layer}_2$ (32 filters). Filters are numbered from 0 to 15 for the first convolutional layer and 0 to 31 for the second convolutional layer. They are labeled from top left to bottom right. Our visualization is




**(a) Training goes slowly with learning rate $\eta = 0.0001$. The weight vector and bias vector curves look like square root with regards to time steps. The gradient vector curves look like random signals.**




**(b) With learning rate $\eta = 0.01$, The weight vector and bias vector curves saturated shortly after training. The gradient vector curves show they vanish quickly. Although there are some gradient signals in the last fully connected layer, the gradient signal at the first convolutional layer drops to zero. Thus the learning stops.**

**Figure 4: Rows 1 to 4(from top to bottom): L2-norms of weight vectors and bias vectors; L2-norms of gradient vectors. Columns 1 to 4(from left to right): convolutional layer 1 and 2; fully connected layer 1 and 2. x-axis represents time steps. y-axis represents distance values.**

able to capture redundancy within the layers. The plot(a) of Fig.5 shows that convolutional filters 0, 3, 7 and 11 in layer$_1$ have a similar behavior. The heatmap(b) corroborates their correlation with a high Pearsons' correlation coefficient between convolutional filters 0, 3, 7 and 11 (in darker red color). The plot(c) shows that filters 23 and 25 in layer$_2$ are similar. So does the heatmap(d). By being able to visualize and identify redundancy, while at the same time quantifying their correlation, we are able to better inform the training process and use this insight to adaptively prune neurons during the training process. Pruning is shown in the next section.

## 6 ON-LINE NEURON PRUNING

It has been shown [14] that larger neural networks and networks that have been carelessly crafted are more likely to overfit. Then, automatically determining network topology has been a hot topic of

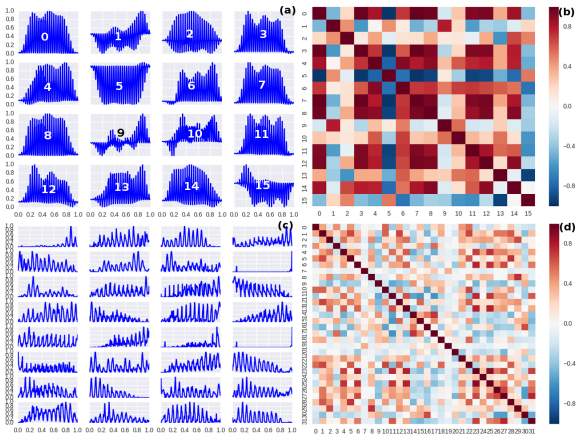



**Figure 5: (a)(b): Activation and Heatmap of activations' correlation of layer$_1$. (c)(d) Activation and correlation of layer$_2$. Color in heatmap represents correlation value. In the heatmaps, color blue is lower correlation value and red is higher correlation value.**

research in the machine learning community. As a complementary method of dropout, we propose our first approximation at dealing with this problem from an on-line pruning perspective. In Figure 5 we saw that a quantitative visualization of activations can give us information regarding redundant neurons. Intuitively, redundant neurons are not providing the network with additional discriminant capabilities and instead slow down the training process. Then, we perform on-line pruning of such neurons.

We use the pruning of convolutional layers as an example and give a simple proof by the following equations. Assume $x$ is an input batch of images. In convolutional layer$_1$, $w_1$ and $b_1$ are weight and bias vectors corresponding to convolutional filter$_1$ and $w_2$ and $b_2$ are weight and bias vectors corresponding to convolutional filter$_2$. $\sigma$ is the ReLU activation function($max(0, x)$). In the following convolutional layer$_2$, $w_1'$ and $w_2'$ are corresponding to the $1st$ and $2nd$ element of convolutional filter$_1'$ tensor. $b_1'$ is the bias of this filter$_1'$. $a_1$ and $a_2$ are activations of filter$_1$ and filter$_2$ in the first convolutional layer. We assume $a_1$ and $a_2$ are similar. $a_1'$ is the activation of filter$_1'$ in the second convolutional layer.

$$a_1 = \sigma(w_1 x + b_1) \approx a_2 = \sigma(w_2 x + b_2) \tag{2}$$

$$a_1' = \sigma(w_1' a_1 + w_2' a_2 + b_1') \tag{3}$$

$$a_1' \approx \sigma((w_1' + w_2') a_1 + b_1') \tag{4}$$

From equation (3), we have the approximate activation of $a_1'$ in equation(4). This means that for pruning, we simply add the corresponding elements of the weights tensors in the following layers and keep one of them in the current layer. Given the computed Pearson correlation coefficients, we set a similarity threshold of 0.9 to identify pruning candidates. Fig.6 shows pruning of similar filters in action(Because different random initial weight values, the shapes of activations look different from Fig.5. The results are from two individual experiments.). The model starts with 16 filters in convolutional layer$_1$. At the end of each epoch, we visualize the activations. We use color to indicate similarities. Take graph(a) as

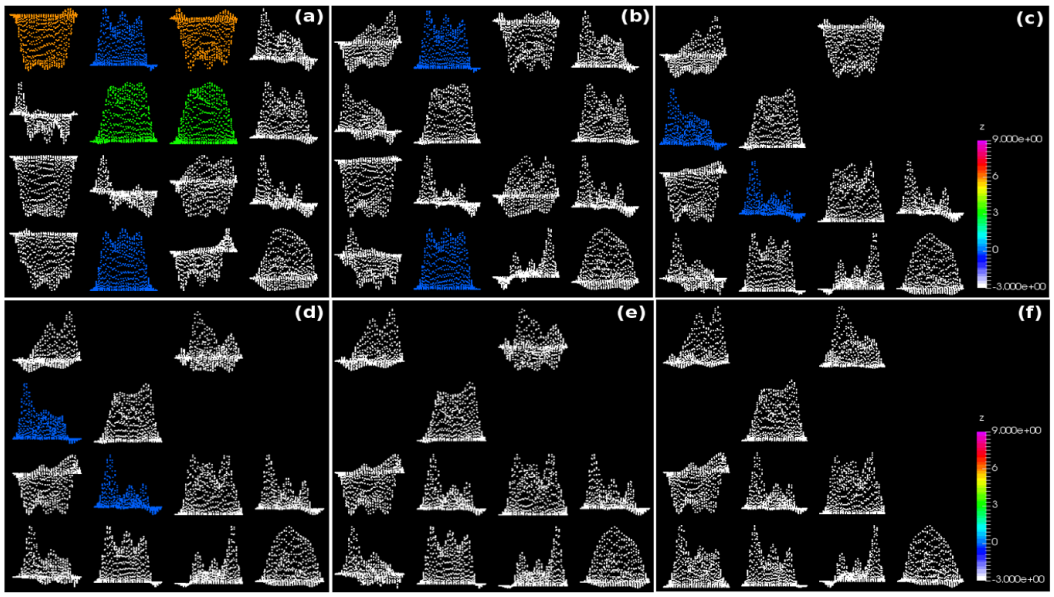



**Figure 6: Training is from (a) to (f). Colors indicate similar groups. (a)Three similar groups are found during epoch$_1$. (b)Only $f_6$ is removed after epoch$_1$ finished. (c)Filter $1, 3, 6, 7$ are removed at the end of epoch$_2$. (d)Mark $f_4$ and $f_9$ after epoch$_3$. (e)remove $f_4$ at the end of epoch$_4$. (f)after 20 epochs, 16 filters reduced to 10.**

an example, there are three pairs of filters that are similar at this time step. They are (filter$_0$, filter$_2$) in orange color; (filter$_1$, filter$_{13}$) in blue color and (filter$_5$, filter$_6$) in green color. After 20 epochs, we reduce the filters in convolutional layer$_1$ to 10. After 40 epochs, we reduced the neural network to 10 and 27 filters in the two convolutional layers. The accuracy of the pruned network remains the same as the original network (99.2%). The number of multiply-add operations (MAC) reduced from 3.2 million to 2.7 million for the original already compact neural network.

## 7 CONCLUSIONS AND FUTURE WORK

In this paper, we present TensorView, a visualization tool that can help the training of deep convolutional neural networks. TensorView uses Paraview and Matplotlib to visualize and analyze the training of CNNs. The visualization of weights, trajectories of weight vectors, L2-norms of weights and biases can provide qualitative insight of the effect of learning rate during training process. The visualization of activations provides quantitative description of redundancies of convolutional filters. These enable us to prune redundant neurons during training.

We provide a set of visualizations as a proof of concept based on the MNIST dataset. We intend to conduct more experiments with more complex datasets and larger neural networks in the future. Other hyper-parameters can also be visualized, such as how initialization affects the performance [12]. We also plan to fully exploit the in-situ analysis and visualization capabilities of Paraview, and to build an interactive mechanism to facilitate the training of more complex networks.

## REFERENCES

[1] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*. 1269–1277.

[2] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 249–256.

[3] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. 2014. Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544* (2014).

[4] Yoshihiko Hamamoto, Shunji Uchimura, Masanori Watanabe, Tetsuya Yasuda, Yoshihiro Mitani, and Shingo Tomita. 1998. A Gabor filter-based method for recognizing handwritten numerals. *Pattern Recognition* 31, 4 (1998), 395–400.

[5] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*. 1135–1143.

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

[7] Google Inc. 2015. TensorBoard: Visualizing Learning. https://www.tensorflow.org/get_started/summaries_and_tensorboard. (2015).

[8] Kitware Inc. and Los Alamos National Laboratory. 2002. Overview Paraview. https://www.paraview.org/overview/. (2002).

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[10] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[11] Eliana Lorch. 2016. Visualizing Deep Network Training Trajectories with PCA. In *The 33rd International Conference on Machine Learning,JMLR volume 48*.

[12] Dmytro Mishkin and Jiri Matas. 2015. All you need is a good init. *arXiv preprint arXiv:1511.06422* (2015).

[13] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. 2014. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806* (2014).

[14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* 15, 1 (Jan. 2014), 1929–1958.

[15] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. 2015. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579* (2015).

[16] Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *European conference on computer vision*. Springer, 818–833.