ReVACNN: Real-Time Visual Analytics for Convolutional Neural Network

Sunghyo Chung
Korea University
sunghyo.chung@gmail.com

Sangho Suh
Korea University
sh31659@gmail.com

Cheonbok Park
Korea University
chunbok94@gmail.com

Kyeongpil Kang
Korea University
rudvlf0413@korea.ac.kr

Jaegul Choo
Korea University
jchoo@korea.ac.kr

Bum Chul Kwon
IBM T.J. Watson Research
bumchul.kwon@us.ibm.com

ABSTRACT

Recently, deep learning has gained exceptional popularity due to its outstanding performances in many machine learning and artificial intelligence applications. Among various deep learning models, convolutional neural network (CNN) is one of the representative models that solved various complex tasks in computer vision since AlexNet, a widely-used CNN model, has won the ImageNet challenge[1] in 2012. Even with such a remarkable success, the issue of how it handles the underlying complexity of data so well has not been thoroughly investigated, while much effort was concentrated on pushing its performance to a new limit. Therefore, the current status of its increasing popularity and attention for various applications from both academia and industries is demanding a clearer and more detailed exposition of their inner workings. To this end, we introduce ReVACNN, an interactive visualization system that makes two major contributions: 1) a network visualization module for monitoring the underlying process of a convolutional neural network using a filter-level 2D embedding view and 2) an interactive module that enables real-time steering of a model. We present several use cases demonstrating benefits users can gain from our approach.

Categories and Subject Descriptors

H.2.8 [**Data Visualization**]: Convolutional Neural Network; I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Design, Performance

Keywords

Deep Learning; Visualization; Convolutional Neural Network

[1] http://www.image-net.org/challenges/LSVRC/

KDD 2016 Workshop on Interactive Data Exploration and Analytics (IDEA'16) August 14th, 2016, San Francisco, CA, USA.

1. INTRODUCTION

Recently, deep learning has made major breakthroughs in many machine learning problems such as computer vision [6] and speech recognition [4]. A traditional neural network model is basically composed of multiple layers, each of which contains multiple nodes where each node is computed as a linear combination of nodes in the previous layer, followed by a nonlinear transformation such as a sigmoid, a tanh, a softmax function. However, neural network has not been widely used until recently since it was difficult to train due to the significant computing time, its sensitivity to initialization and hyper-parameters, and other issues. Various treatments have been proposed including dropout [9], batch normalization [5], and alternative nonlinear functions such as a rectified linear unit [8], which successfully handled most of the existing issues.

Beyond the traditional model, the neural network structure has evolved in various forms, leading to tremendous success in important applications. Largely responsible for this recent success is convolutional neural network (CNN), a type of neural network suited for real-world image classification tasks. Although convolutional neural networks have been originally proposed by LeCun et al. [7] back in the early 1990s, demonstrating an outstanding performance in hand-written digit recognition, it was not widely used until 2012 when Krizhevsky et al. [6] achieved a superior performance on image classification tasks in ImageNet challenge, using a deep architecture model of convolutional neural network. This propelled major research movement towards creating variants in architectures and improving algorithms for even higher performance. In just few years, much progress has been made to the point of approaching or even surpassing human abilities in various challenging tasks.

While making significant achievements, the understanding of underlying processes in these models received less examination, and the need for tools and techniques for exploring and understanding the inner workings of these various models ensued. However, complicated deep learning structures are difficult to understand. Different types of layers such as convolution, pooling, and fully-connected layers interact with each other, handling different parts of data characteristics. Furthermore, each layer has different sets of hyper-parameters to determine before training the model. Thus, such a model selection process including setting the number of layers and nodes, and hyper-parameter values has not been intuitive nor straightforward, leaving users with no idea about how to properly perform this process.

In addition, the significant amount of time required to train a deep learning model has made the training process largely detached

from dynamic user intervention. For example, a recently proposed model called ResNet [3], which is considered one of the state-of-the-art models, takes days to weeks to train using ImageNet datasets with the fastest graphics processing unit available.

In response, we propose a proof-of-concept visual analytic system for allowing users to understand and steer a deep learning model in real time during the training process. To the best of our knowledge, our system is one of the first systems that visualize detailed information about the model during the training process and support dynamic user interactions with the model in real time.

In particular, the main contributions of this paper are summarized as follows:

- Real-time visualization of how each node/filter in a deep learning model is trained, e.g., the stability of nodes/filters and the relationships between them
- Real-time model steering by dynamically adding/removing nodes and layers during the training process

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 presents detailed description of our system and its visual components. Section 4 presents usage scenarios. Finally, Section 5 concludes our discussion with plans for future work.

2. RELATED WORK

In this section, we discuss recent efforts towards interactive visualization of deep learning for its in-depth understanding and user control.

Bruckner et al. [1] developed the system called deepViz, an interactive visualization based on the timeline framework that shows the heatmap representations of filters in each layer, the confusion matrix, and the clustered images at different checkpoints for understanding and diagnosing the network. Zeiler and Fergus [12] showed the practical application of a visualization system for the diagnostic purpose by utilizing a feature inversion technique called deconvolution to refine the model and further improve performances. With the system that visualizes live activations in real time and features at each layer, Yosinski et al. [11] made contributions to the visualization of convolutional neural network by providing several new regularization methods that produce qualitatively clearer visualization of images.

More on the interactive visualization side, a web-based implementation, such as ConvNetJS [2], made training convolutional neural network possible in a browser using the Javascript library. In addition, Bolei et al. [13] developed another web-based interface where a user can select an activation of a particular data item at a particular layer and check the highly activated nodes together in the other layers.[3] Harley et al. [2] visualized a convolutional neural network in a three-dimensional space where the network structure and the used images are shown simultaneously. Additionally, Google's TensorFlow library provides a graphical user interface called TensorBoard[4], which visualizes neural network as a computational graph where users can check the status of the trained model and change the detailed configurations. More recently, Google also made a web interface called TensorFlow Playground[5] publicly available so that users can play with neural network models using several toy data sets. On the other hand, NVIDIA developed its own deep learning library and a web-based monitoring system called DIGITS.[6]

Even with these various efforts, there exist significant room to improve the interactive visualization aspects of deep learning models along with the recent advancement in this area. Among them, real-time monitoring and steering of deep learning has not been properly addressed, which is the focus of our system proposed in this paper.

3. REVACNN: REAL-TIME VISUAL ANALYTICS FOR CONVOLUTIONAL NEURAL NETWORK

To empower users to dynamically monitor and interact with a convolutional neural network in real time during its training stage, we propose ReVACNN. In this section, we present (1) the system overview, (2) the visualization modules for real-time monitoring and steering of the model, and (3) the implementation details of the proposed system. The front-end of our web-based system is implemented using HTML, CSS, and Bootstrap. D3.js[7] is used for animating filters ('jittering') in the diagram. All the computations are currently performed with Javascript in the browser on the client side.

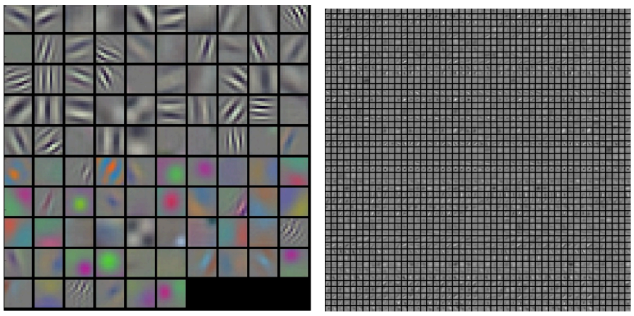
3.1 System Overview

The main goal of our system is to provide real-time steering capabilities in an easy-to-use manner. To this end, we decided to build our proof-of-concept system based on Javascript-based deep learning library called ConvNetJS.[8] In contrast to other major deep learning libraries such as Theano, Tensorflow, Torch, and Caffe, ConvNetJS runs completely in an easily accessible web browser on the client side, which is appropriate in visualizing dynamic changes of deep learning models and responding immediately to user interactions in real time. Note, however, that in exchange of such ease of use and real-time interactivity, ConvNetJS that our system uses lacks the GPU-based acceleration of computations that most of the other major libraries offer. In the scope of the current paper, we mainly investigate the real-time visualization capabilities for viewing and steering the deep learning process, using ConvNetJS as an example. We leave the topic of integrating other libraries with our real-time visual analytics system as future work.

In fact, our visual interface is built upon the implementation of CIFAR-10 demo using ConvNetJS,[9] as shown in Fig. 4, which uses the CIFAR-10 dataset[10] for object recognition. In addition, we developed additional capabilities of monitoring and steering the CNN model in real time. In summary, the main functionalities of ReVACNN we added are as follows:

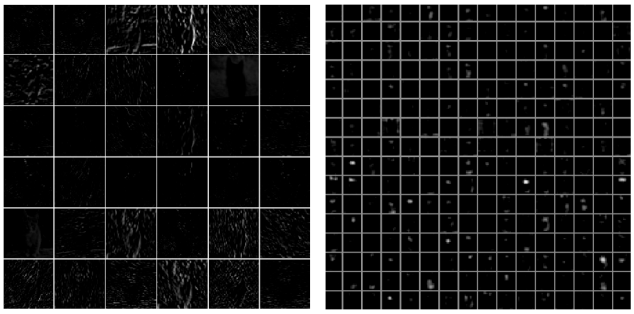
- **Network visualization and configuration view**. This module provides users with a visual overview of the network and more importantly, the ability to monitor the dynamic training process in real time. Moreover, users can modify the network dynamically and incrementally, adding or removing nodes and layers with simple "point-and-click" interactions.

[2] http://cs.stanford.edu/people/karpathy/convnetjs/
[3] http://people.csail.mit.edu/torralba/research/drawCNN/drawNet.html
[4] https://www.tensorflow.org/versions/r0.8/how_tos/graph_viz/index.html
[5] http://playground.tensorflow.org/
[6] https://developer.nvidia.com/digits
[7] https://d3js.org/.
[8] https://github.com/karpathy/convnetjs
[9] http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html
[10] https://www.cs.toronto.edu/~kriz/cifar.html



(a) Filters in conv1 layer (b) Filters in conv2 layer

Figure 1: Filter coefficients in AlexNet



(a) 1st convolution layer (b) 5th convolution layer

Figure 2: Activation maps in AlexNet [6]

- **Filter-level 2D embedding view**. This view shows the relationships of individual filters/nodes at a particular layer as a 2D embedding view. To generate such a view, we utilize a t-distributed stochastic neighbor embedding (t-SNE) [10]. Using the vector representation of each filter, we allow users to flexibly choose one among its filter coefficients, filter gradients, its activation map of a particular image, and its activation gradient maps so that users can explore various aspects of filters that are being trained.

The details of the above functionalities our system provides will be further discussed in Section 3.2.

3.2 Network Visualization and Configuration

To facilitate the understanding of how each node/layer has been trained, it is important that we directly interpret the filter coefficients, which correspond to the linear combination of coefficients or weights assigned to different positions of pixels in an image. In addition, given an image, an activation map corresponding to a particular filter gives another insight from the perspective of how much the filter gets activated depending on the image. For example, Fig. 1 shows filter coefficients found in AlexNet. Since the first-layer weights, as shown in Fig. 1a(a), are the filters directly looking at the raw pixel data of an input image, their images are often the most interpretable among the filters from all the other layers. However, as layers deepen, their meaningful interpretation becomes increasingly challenging. Fig. 1b(b) shows that the filters found in the second convolution layer are too complex and vague to be informative. Normally, an analysis of the first-layer weights can help users recognize whether the network has been successfully trained. Users can assess the success of training based on whether trained filters have smooth transitions among them so that they can capture as diverse patterns as possible. However, since such analysis relies on subjective judgment, it is clear that users need additional evidences to decide whether such argument is reasonable.

Additionally, Fig. 2 shows the activation maps found in the first and fifth convolution layers. Clearly, it is difficult to make sense of the activation maps in deeper layers because they are representing a composite mixture of already complex patterns. Also, these activation maps are shown to be relatively sparse, which means that the majority of pixels in these activation maps are mostly zero given an input image. This indicates that they hardly get activated, which may not be helpful in generating useful information. Nevertheless, identifying those filters bearing these characteristics by just looking at these activation maps is a difficult task. Our visualizations approach helps users handle the task more easily.

3.2.1 Network visualization

As shown in Fig. 3, our network visualization module provides users with a quick overview of the model. In addition, users can gain insight from the dynamic evolution of the network during the training process. In particular, among its various parameters representing dynamic evolution of the network, our system highlights how stable or converged each node is during the algorithm iterations in the form of jittering animation of nodes. That is, those nodes with a large amount of movements in their jittering animations indicate that they are being actively trained at a given moment. The quantitative value to determine this amount is computed as the magnitude of an average gradient back-propagated per each filter coefficient in the corresponding node.

In addition, the path connecting two layers shows how input images are being forward-propagated through the network layers. That is, the thickness of a path corresponds to the sum of pixel values on a particular filter in the corresponding layer. Note that only those whose values belong to the top 50% are configured to be visible in order to avoid a visual clutter in the visualization module. This path visualization has additional benefit of helping users identify how convolution layers are outputting filtered images and most importantly, which path influences the softmax layer responsible for classifying an input image. From this visualization module, users can also easily add or delete filters in the hidden layer with simple "point-and-click" interactions, and the change in the model is reflected in real time. The interactive feature helps to steer the training process of the model.

3.2.2 Training statistics visualization

During training, the loss function serves as a clue for identifying whether the network is properly trained. Thus, our module, shown in Fig. 4, displays the training loss as a line chart. Users can keep track of the temporal progress of the loss function. Since the batch size is set as four by default, the loss is plotted each time as the average training loss of a batch of four input images. If the batch size is modified by users, the loss chart also changes accordingly. In addition, other statistics, such as training accuracy and validation accuracy, are updated for each input image and shown to users for an in-depth analysis. Other hyper-parameters such as the learning rate, the momentum, the batch size, and the weight decay, can be modified. To facilitate the understanding of how each node/layer has been trained, it is important that we directly enables users to observe changes in the training accuracy immediately. Using the capability, users can properly adjust learning rates, batch sizes, and momentum values when the network is stuck in an undesirable local minimum.

3.2.3 Filter-level 2D embedding visualization

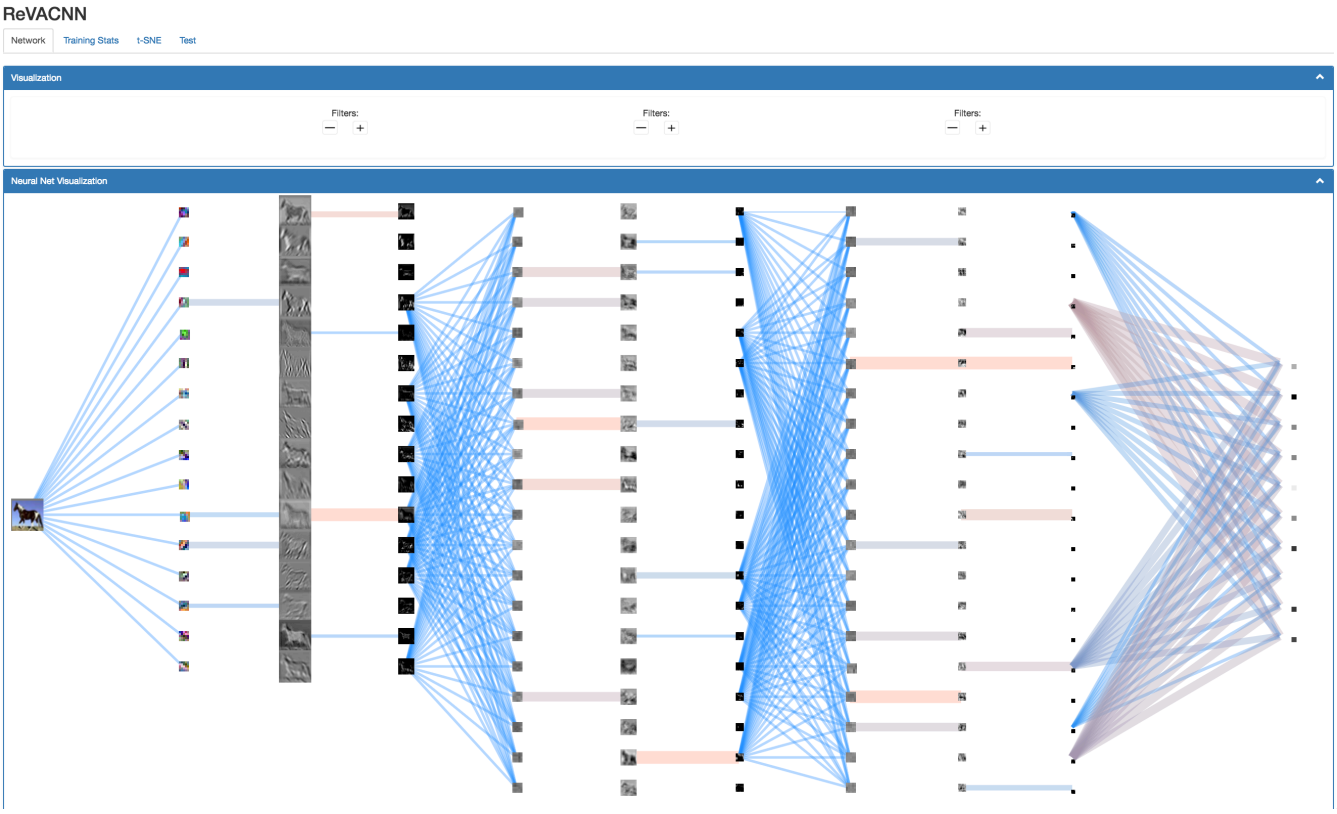


Figure 3: Network visualization of ReVACNN

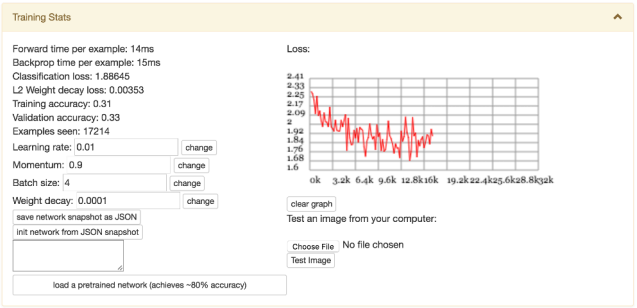


Figure 4: Training statistics view of ReVACNN

As described above, the filter coefficients and the activation maps have frequently been the main subject of visualization when analyzing a convolutional neural network. In our system, we explore them using their 2D embedding view computed by t-SNE. As shown in the left side of Fig. 5, users can open up each layer panel and observe the 2D embedding view of filter coefficients, filter gradients, its activation maps, and the activation gradients at the corresponding layer by clicking the radio button in the left pane. When users change the network architecture and initiate training, the left pane changes accordingly to reflect the model's layer configuration. This t-SNE view of the system provides users with the capabilities of node-level as well as layer-level exploration. In the case of node-level exploration, users can view the similarity between filter coefficients and activation maps or even activation gradient maps in the selected layer. In the case of layer-level exploration, the comparison of clusters of filter coefficients, the activation maps, the activation gradient maps between different layers can reveal insights for understanding of the model and further diagnosis. The usage scenarios demonstrating the usefulness of this view will be discussed in detail in Section 4.

4. RESULTS

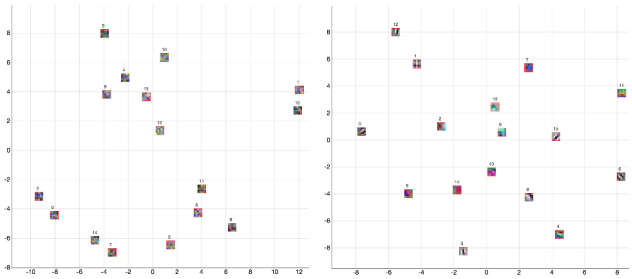
In this section, we present two use cases demonstrating the advantage of our system to monitor and steer the deep learning model in real time.

4.1 Real-Time Dynamic Model Configuration

In this section, we present four use cases where we can reveal various insights from our filter-level 2D embedding view in which a user can extract valuable insights about the model. In these use cases, we used a CNN model, which has an input layer that takes in $32 \times 32 \times 3$ input images and three convolutional layers with a filter size of 5×5 with the stride size of 1 and padding of 2 where the three convolutional layers–each of which has 16, 20, and 20 filters, respectively–have both ReLU and pooling layers behind each convolutional layer, finally followed by a softmax layer with ten classes as the last layer of our network.

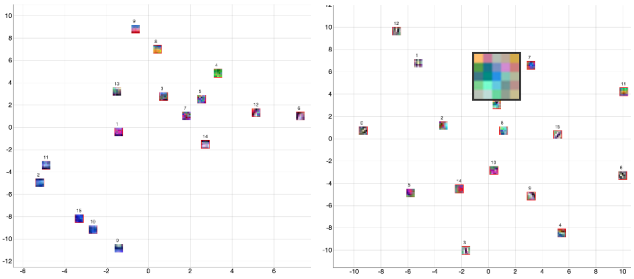
Cluster patterns. In general, neural network and deep learning models are sensitive to initialization, hyper-parameters, and other settings. Thus it is difficult to properly train the model so that it performs reasonably well even for the training data. Our filter-level 2D embedding view provides important insights about the characteristics of a properly trained model. While training the above-specified model, we checked the 2D embedding of filter coefficients at 30

Figure 5: 2D embedding view of ReVACNN



(a) Clear cluster pattern (b) No cluster patterns

Figure 6: Comparison of cluster patterns of filter coefficients



(a) Improperly trained filters (b) Properly trained filters

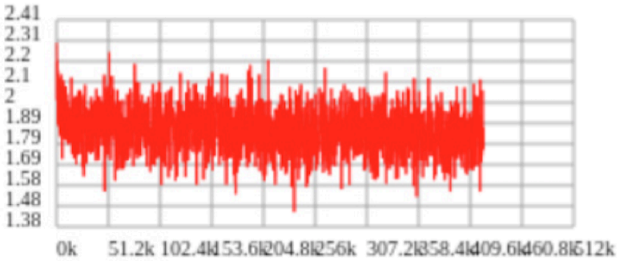
Figure 7: RGB patterns of the first-layer filters

epochs where the accuracy was quite low, e.g., 0.32. In this case, the 2D embedding view of filter coefficients shows a clustered pattern among these filters, as shown in Fig. 6a(a). This indicates that those filters belonging to a particular cluster capture somewhat redundant patterns from input data. In other words, they are not trained well enough to extract diverse patterns from the training data. On the other hand, at 120 epochs where the accuracy reaches 0.78, the 2D embedding view of filter coefficients exhibits somewhat evenly distributed filters with no clear cluster patterns, as shown in Fig. 6b(b). This example indicates an important characteristics of a well-trained model that the diversity of trained filters is generally desirable in achieving a greater classification accuracy.

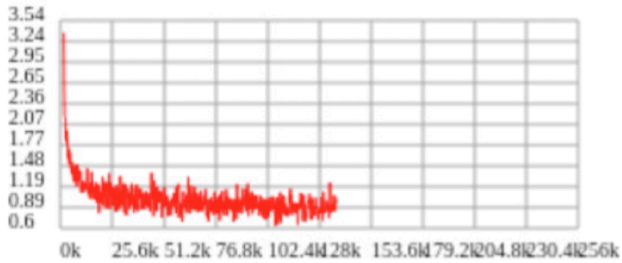
RGB patterns. The first convolutional layer takes an input image that has the depth of three RGB channels and generates each filter that linearly combines all the three channels. Similar to the previous case, when the model is not properly trained, i.e., showing a low accuracy value, we found that a trained filter often reflects only a single color channel as opposed to a combination of all the three color channels, as seen from all-blue colored filters in Fig. 7a(a). However, as seen in Fig. 7b(b), when the model shows a relatively good performance, each filter usually combines the information from all the color channels. Based on such different RGB patterns, one can infer that those filters that combine all the channels of the previous activation maps contribute to improving the generalization ability of the trained model.

4.2 Steering

In this use case, we set up our model as follows: three convolutional layers followed by ReLU and pooling layers after each convolution layer; 20 filters in each convolution layer (based on this property, we call this model a '20-20-20' model), with the filter size of 5×5; a fully-connected layer as the last layer. We trained a '20-20-21' model and found that this model does not train well, and its loss function value does not go below 1.48, as illustrated in Fig. 8a(a). On the other hand, we initially trained a '20-20-20' model, which converged relatively fast and showed a much better loss function value well below 1.19. Utilizing our interaction



(a) 20-20-21 model trained from scratch



(b) 20-20-21 model generated after filter addition

Figure 8: Effects of dynamic filter addition

capability of dynamically adding a node during training in the network visualization module, we added a filter after 15 epochs as the loss function graph reached a plateau. Accordingly, the model was dynamically changed from a '20-20-20' model into a '20-20-21' model while maintaining the currently trained model except for the added filter. Finally, the resulting model still maintained a relatively good loss function value, even experiencing a slight increase in accuracy at times. Without this dynamic network configuration process, the '20-20-21' model, which is our target model, would be more difficult to train from scratch. This shows the importance and the value of dynamic network configuration in real time during the training process.

5. CONCLUSION AND FUTURE WORK

In this paper, we proposed ReVACNN, a real-time visual analytics system for a convolutional neural network. It supports exploring and steering the network by visualizing its layers and nodes. Additionally, we provided a filter-level 2D embedding view by applying t-SNE to various filter information, such as filter coefficients, filter gradients, the activation maps, and the activation gradients. Through these capabilities offered by our system, one can obtain in-depth information such as whether the network is trained properly or not as well as other insights about the trained filters. By using such information, one can flexibly steer the model and achieve better performances.

As our future work, we plan to improve our work as follows:

Real-time monitoring between GPU and CPU. Currently, in our proof-of-concept system, we relied on ConvNetJS, a Javascript-based library for deep learning. However, other more scalable libraries run most of the intensive computations in GPU, which has its own memory space separate from that of CPU. However, the front-end monitoring system usually works on the CPU side, so in order to truly achieve the real-time monitoring of the training process, memory copy operations from GPU to CPU should be frequently performed, which can degrade the computational efficiency of the training process. To handle this issue, some partial information could be selectively transferred based on a particular criterion, e.g., only when the nontrivial amount of changes of a parameter occur. Otherwise, multi-threaded syncing between the memory spaces of GPU and CPU, which performs memory copy operations only when the computing resource of GPU is available, could also be another option.

Advanced dynamic steering capabilities. So far, we provided the capabilities of dynamic node/layer addition/removal in our system. However, many other advanced dynamic steering capabilities could be developed. For instance, skipping some nodes/layers that are already trained sufficiently can accelerate the subsequent optimization steps. When nodes/layers are added/removed, their initialization could be carefully performed so that the newly added nodes/layers can capture complementary information of data to the existing nodes/layers. When removing nodes/layers, we could recommend those that have minimal impact to the overall performance, e.g., a redundant node from clustered nodes. We may be able to define the criteria to determine such minimal effects in various ways, e.g., the variable importance score of each node/layer.

Acknowledgments

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No. R2215-15-1021, Visual Analytics for Real-time User-driven Deep Learning).

References

[1] D. Bruckner, J. Rosen, and E. R. Sparks. deepviz: Visualizing convolutional neural networks for image classification. 2014.

[2] Adam W Harley. An interactive node-link visualization of convolutional neural networks. In *Advances in Visual Computing*, pages 867–877. Springer, 2015.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[4] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

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

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

[8] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way

to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[10] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

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

[12] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer vision–ECCV 2014*, pages 818–833. Springer, 2014.

[13] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*, 2014.