ReVACNN: Real-Time Visual Analytics for Interactively Steering Convolutional Neural Network

Blinded for Review
Blinded for Review
blinded@for.review

Abstract

Convolutional neural networks (CNNs), a particular type of deep neural networks, have been successfully applied to solve longstanding real-world problems in computer vision and natural language understanding. Despite its outstanding capability, its complicated inner workings make it difficult to interact with CNNs in a user-driven manner; training a CNN model properly is time-consuming and sensitive to initialization and parameter settings. To tackle these issues, we present a real-time visual analytics system for CNNs called ReVACNN, which aims at enhancing the interpretation of CNNs and allowing users to steer its training process in real time in their own manner. In detail, ReVACNN visualizes the overall training process by (1) showing the amount of forward-propagated activations and back-propagated gradients of each filter/layer during training iterations and by (2) presenting the 2D embeddings of trained filter coefficients and activation maps to show the relationships among different filters and data items. Additionally, ReVACNN allows users to perform novel interactions in real time: (1) skipping the gradient descent update on particular layers of a CNN model to reduce the subsequent training time and (2) selecting those misclassified data or those with low confidence scores from the 2D embedding view and retraining the model using them to further improve classification performances. We present several use cases demonstrating the benefits of such interactions in ReVACNN.

1 Introduction

Deep learning recently made major breakthroughs in numerous machine learning problems such as computer vision [Krizhevsky *et al.*, 2012], speech recognition [Hinton *et al.*, 2012], and natural language processing [Collobert and Weston, 2008; Bahdanau *et al.*, 2014]. Beyond the traditional fully-connected model, the deep learning structure has evolved in various forms, including convolutional neural network (CNN) [LeCun *et al.*, 1998; Krizhevsky *et al.*, 2012], a type of neural network suited for real-world image classification and other tasks, as well as recurrent neural network [Elman, 1990] and long short-term memory (LSTM) network [Hochreiter and Schmidhuber, 1997], another type of neural network utilizing dependencies in sequential and temporal data.

While significant achievements have been made, the understanding of underlying processes behind complicated deep learning models received less examination, and the need for tools and techniques for exploring and understanding the inner workings of these various models ensued. This made it difficult to properly train a deep learning model by avoiding underfitting/overfitting, which has been a time-consuming task that requires repetitive model selection and hyper-parameter tuning. For example, in order to train the model, a user has to constantly switch between different combinations of layers and filters, the step size, and so on, but there have existed no intuitive or straightforward guidelines on how to properly perform these processes.

In response, we present a **re**al-time **v**isual **a**nalytics system for **CNN**s called ReVACNN, which allows users to understand the behavior of a CNN model and steer the training process in real time in their own manner. In detail, ReVACNN visualizes the overall training process by (1) showing the amount of forward-propagated activations of data items and back-propagated gradients of each filter during the iterative training processes and by (2) presenting the 2D embeddings of trained filter coefficients and the activation maps of data items to show the relationships among different filters and data items. ReVACNN further allows users to perform novel interactions in real time: (1) skipping the gradient descent update on the sub-part of a CNN model to reduce the subsequent training time and (2) selecting those misclassified data or ambiguous data located near decision boundary from the 2D embedding view and retraining the model with them to improve classification performances. We present several use cases demonstrating the benefits of the proposed interactions in ReVACNN.

The main contributions of this paper are summarized as follows:

- Real-time visualization of how each filter/layer in a CNN model is being trained, e.g., the stability of filters and the relationships between them and

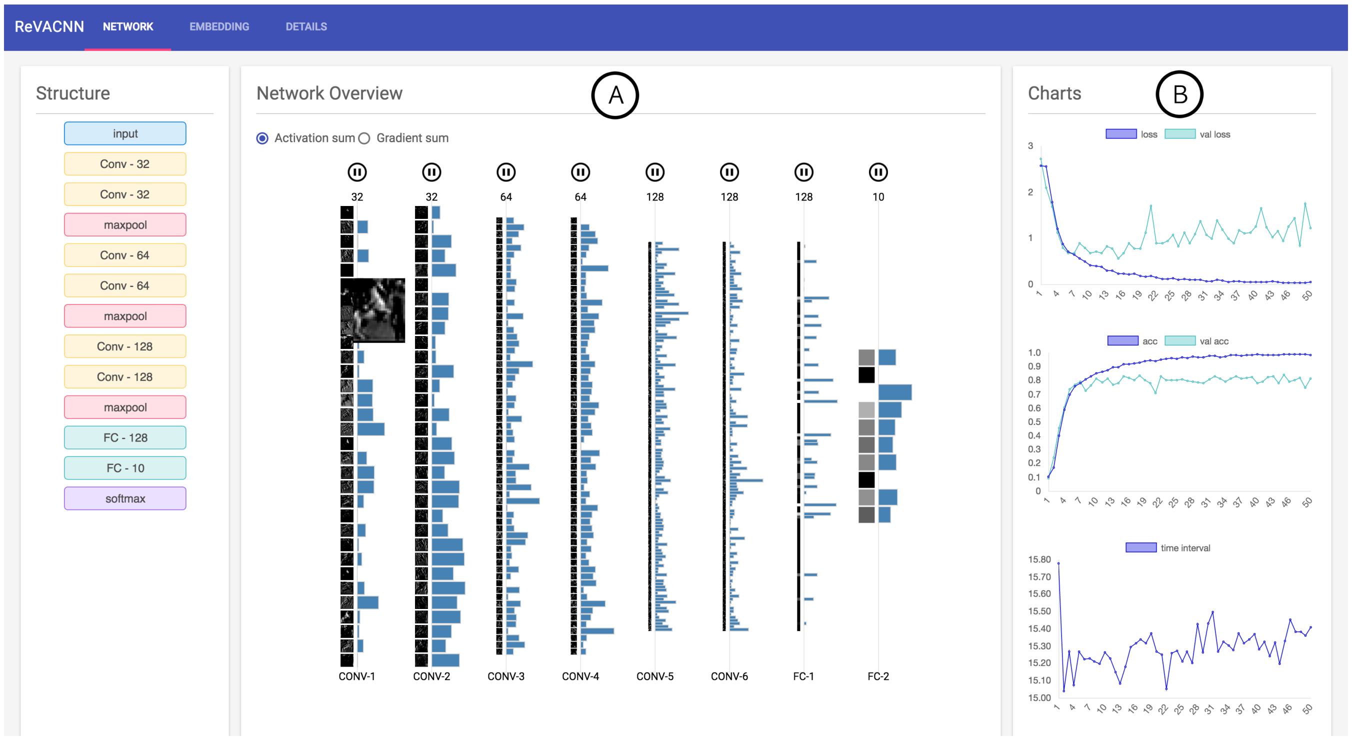


Figure 1: Overview of ReVACNN: (A) the network view (B) the training statistics view

- User-driven steering of a CNN model by interactively skipping the gradient decent update of particular filters and layers to reduce training time and retraining/fine-tuning the model with user-selected data that are misclassified or with a low confidence score.

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 neural networks for its deep understanding and user interactivity.

Bruckner et al. [Bruckner *et al.*, 2014] developed the system called deepViz, an interactive visualization based on the time slider interface that shows the layer-wise transitions of heatmap representations of filters in each layer, the confusion matrix, and the clustered images at different check-points for understanding and diagnosing the network. Zeiler and Fergus [Zeiler and Fergus, 2014] showed the practical application of a visualization system for the diagnostic purpose by utilizing a feature inversion technique called deconvolution to refine the model. With the system that visualizes the activation maps of video streams in real time and features at each layer, Yosinski et al. [Yosinski *et al.*, 2015] proposed new methods that reconstruct the trained filters in CNNs as interpretable images. Rauber et al. [Rauber *et al.*, 2017] explored deep neural network by using dimensionality reduction techniques to visualize filters and activation maps. It included a variety of examples of two-dimensional embedding views with various datasets such as MNIST, CIFAR-10 and SVHN. This study also analyzed the effect of training processes by showing the differences in terms of the relationships between filters and activations.

On the interactive visualization side, a web-based implementation, such as ConvNetJS [1], made it possible to train a CNN model in a browser environment using a Javascript library. Bolei et al. [Zhou *et al.*, 2014] developed another web interface where a user can select the activation of a single data item at a particular layer and check the highly activated nodes across different layers.[2] Harley et al. [Harley, 2015] visualized a CNN model in a three-dimensional space where the network structure and the activation maps for each node are simultaneously visualized. Google's TensorFlow library provides a graphical user interface called TensorBoard,[3] which visualizes a 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 [Smilkov *et al.*, 2016][4] publicly available so that users can understand various effects of different parameter settings by running with a relatively simple neural network model on several toy data sets. On the other hand, NVIDIA developed its own deep learning

[1]http://cs.stanford.edu/people/karpathy/convnetjs/
[2]http://people.csail.mit.edu/torralba/research/drawCNN/drawNet.html
[3]https://www.tensorflow.org/get_started/graph_viz
[4]http://playground.tensorflow.org/

library and a web-based monitoring system called DIGITS.[5]

In visual analytics communities, those systems with advanced analytic as well as visualization components have also been proposed. Liu et al. presented a system called CNNVis [Liu *et al.*, 2016], which applies bi-clustering and edge bundling techniques to effectively visualize the learned filters by aggregating the low-level features into high-level ones. Through these approaches, CNNVis selectively visualized representative filters and images from clustering results to avoid the information overload and visual clutter. LSTMVis [Strobelt *et al.*, 2016] is another visual analytic system that utilizes a parallel coordinates plot to visualize the activation patterns of particular hidden nodes given input text streams.

Even with various efforts mentioned above, a significant amount of room is still available to improve the interactive visualization aspects of deep learning models. In particular, in most of these systems, the real-time monitoring and interaction capabilities during the training phase of real-world deep learning models has not been fully addressed. In this sense, our model can be viewed as one of the first systems where users can check the status of the training process of CNNs in real time and perform nontrivial interactions to steer the model in a user-driven manner.

3 ReVACNN: Real-Time Visual Analytics for Steering Convolutional Neural Network[6]

The main goal of ReVACNN is to provide real-time monitoring and steering capabilities on a CNN model using the visual analytics approach in an easy-to-use manner. As shown in Fig. 1, the main visualization modules of ReVACNN are composed of (1) the network view and (2) the training statistics view.

3.1 Network view

This module provides users with an overview of the activation of filters in layers. As shown in Fig. 1(a), the view shows the activation of each node as well as a bar chart showing the total amount of (forward-propagated) activations or (back-propagated) gradients of individual filters during the training process.

It gives users the ability to monitor how the network is being trained in real time. For example, by checking those nodes with a large amount of activations, one can see which part or pattern of a given image is mainly captured by the model. On the other hand, the distribution of the gradient amount across different nodes/layers indicates which of them become relatively converged or stable compared to the others and which are going through major changes.

In this visualization module, users can hover through nodes to view the magnified 2D activation images. It helps users understand how the input image is transformed into a class score as it goes through the deep neural layers. Furthermore, users can selectively "freeze" particular layers so that they can be skipped in the subsequent training processes. This capability can drastically reduce the training time while having little effect on and preserving the accuracy. Once these interactions are performed, the view gets dynamically updated so that users can visually identify the effect of their interactions.

3.2 Training statistics visualization

During training, the loss function serves as a clue for identifying whether the network is properly being trained. Thus, our module, shown in Fig. 1(b) displays the training loss as a line chart. Users can keep track of the temporal progress of the loss function. 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. Also, training time for each epoch is displayed. This is especially useful when we use the "freezing" function. We can directly compare the training time before and after freezing layers.

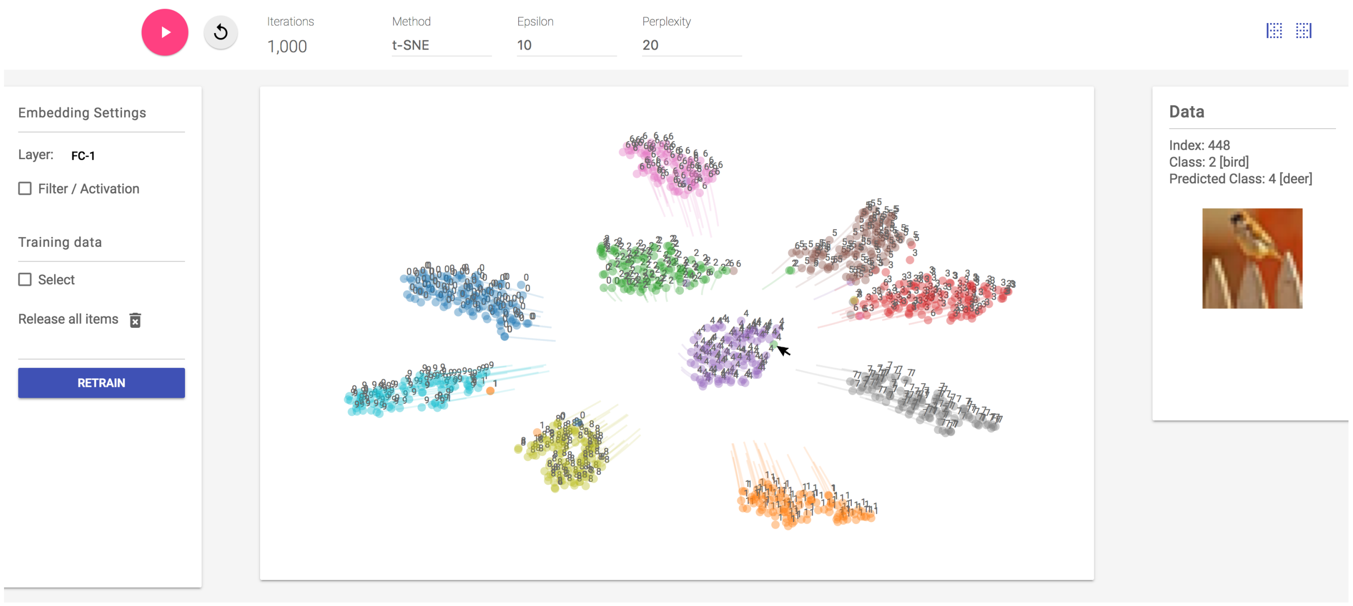
3.3 2D embedding view

As discussed above, the filter coefficients and the activation maps have frequently been the main subject of visualization when analyzing CNNs. In our system, we explore them using the 2D embedding view computed by t-distributed stochastic neighbor embedding (t-SNE). As shown in Fig. 2, this view shows the relationships of individual filters/activations of a user-selected layer. Filters are presented using the vector representation of their filter coefficients in the same layer. Activation maps are displayed using images of training/test data. The user can change the parameter settings of t-SNE from the upper and the left panels. The right panel shows the image of a data item that the user hovers over with the mouse. It also shows the class label and its predicted label. Using the embedding view, users can explore various aspects of filters/data being trained. In this view, each filter/activation is represented as an image or circle showing the details of the chosen input image, along with the trailing lines indicating the traces of its coordinates over the five t-SNE updates. We can track each representation as it changes during the training or the t-SNE iteration process.

By visualizing filters via this view, one can obtain an idea about which filters are closely related, i.e., which filters capture redundant information and which part of the information is not being captured. In general, the first layer filters directly look at the raw pixel data of an input image, and thus their images are often the most interpretable among all the layers. 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 the trained filters have smooth transitions among them so that they can capture as diverse patterns as possible. The t-SNE visualization of activations shows clustered pattern of activations as they pass through different layers. As input data go through deeper layers, we get more clustered and separate patterns than in previous layers. In this view, we can analyze which data items are correctly classified and which are not.

[5]`https://developer.nvidia.com/digits`

[6]The system is currently available at `http://121.134.151.241:5000/`. Later, we will make our system publicly available with a proper domain name.


Iterations 1,000
Method t-SNE
Epsilon 10
Perplexity 20

Embedding Settings
Layer: FC-1
☐ Filter / Activation

Training data
☐ Select
Release all items

Data
Index: 448
Class: 2 [bird]
Predicted Class: 4 [deer]

Figure 2: 2D embedding view of ReVACNN. Each node represents an input data item as its activation map at a user-selected layer, color-coded with its label. The trailing line shows the traces of the five previous iterations so that a user can track the training process in real time.

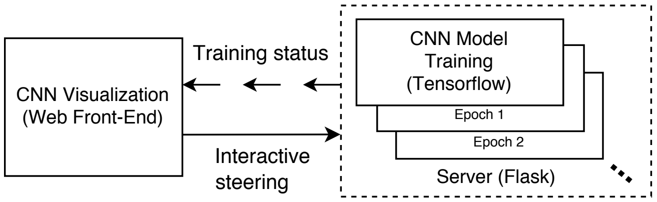


Figure 3: System architecture of ReVACNN

Streamed t-SNE

Standard t-SNE has to restart with random initial coordinates once new input data become available. Because of this, we always obtain different results even though we have almost identical data with slight changes. This makes it difficult to keep track of changes in filters/activations in the embedding view. To overcome this and show filters/activations during training in a seamless manner, we developed the streamed t-SNE. While maintaining the previous t-SNE view with a minimal change, the streamed t-SNE internally updates the pair-wise distance matrix using new data and continue the coordinate updating step instead of restarting it from scratch.

3.4 System Architecture

Fig. 3 shows the architecture of ReVACNN. The system consists mainly of the front-end web interface and the back-end server. The server, which uses a Python web framework called Flask,[7] runs a deep learning library called TensorFlow, and it transmits the model information to the web front-end during the training process in real time. Our front-end web-based system is implemented using HTML, CSS, and Material design lite. We also utilized Keras-Hualos[8] for connecting the back-end server and the front-end web interface. The server side computations are performed with Python in the Jupyter Notebook environment.[9] To enable real-time interactive visualization, we used web workers, a method to run a JavaScript code in a thread programming environment. In this manner, we maintain the high efficiency of the user interfaces against the computationally intensive task of t-SNE.

4 Usage Scenarios

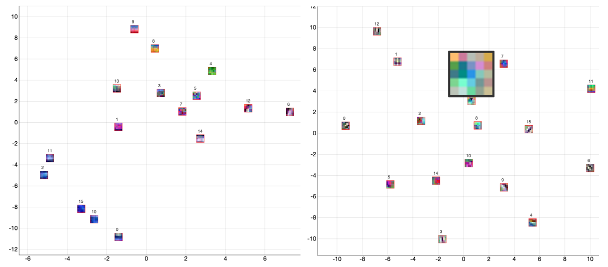
In this section, we present several use cases demonstrating the advantage of ReVACNN in monitoring and steering a CNN model during the training phase.

4.1 Experimental settings

CNN model. We start with a VGG-style neural network to train and test our system, as shown in the 'structure' view in Fig. 1. It has an input layer taking $32 \times 32 \times 3$ input images, followed by six convolutional layers which have 32, 32, 64, 64, 128 and 128 filters, respectively, with each filter size as 3×3, the stride size as 1, and the padding value as 1. Additionally, each convolutional layer has a batch normalization layer and a ReLU layer, and after its second, fourth, and sixth layers, the max pooling layer follows. Afterwards, the two fully-connected layers followed by a softmax layer with ten classes are added at the end of our network.
Dataset. For our experiment, we used CIFAR-10 [Krizhevsky and Hinton, 2009] dataset, which consists of 60,000 32×32 color images in ten classes, with 6,000 images per class. The ten classes represent different objects such as airplane, automobile, bird, cat, deer, dog, frog, horse,

[7]http://flask.pocoo.org/

[8]https://github.com/fchollet/hualos
[9]https://ipython.org/notebook.html



(a) Improperly trained case (10% training accuracy) (b) Properly trained case (80% training accuracy)

Figure 4: Cluster patterns and the reconstructed images of the first-layer filters.

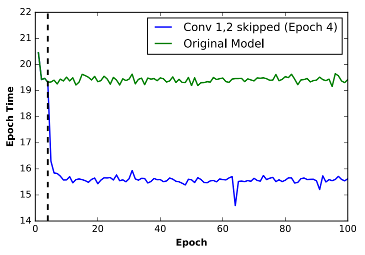


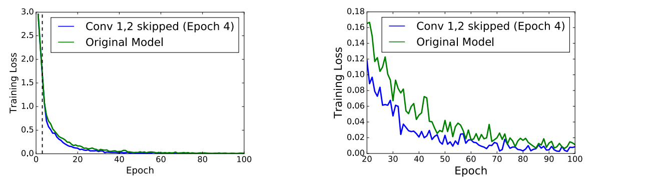
Figure 5: Computing times per epoch for the original model vs. the model with Conv1,2 layers skipped starting from epoch 4 (a vertical dotted line) on CIFAR-10 dataset.

ship, and truck. We randomly split the entire data into 50,000 training and 10,000 test images.

4.2 Real-time monitoring

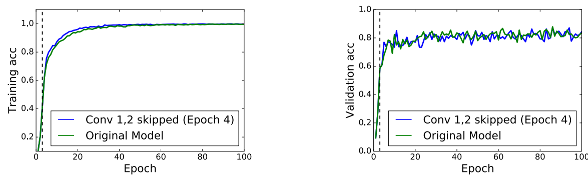
In general, neural network and deep learning models are sensitive to initialization, hyper-parameters, and other settings. Thus it is often difficult to properly train the model so that it performs reasonably well even for the training data. Our 2D embedding view can visualize each filter as a data item using their trained coefficients as their feature vectors, which provides important insights about the characteristics of a properly trained model. As shown in Fig. 4, when the training accuracy stays low even after training, the 2D embedding view of filter coefficients shows a clustered pattern at the lower-left part in Fig. 4(a), which means multiple filters capture the information in a redundant manner. It indicates that this model is not properly trained so that it can extract diverse patterns from data. On the other hand, where the accuracy reaches high, the 2D embedding view of filter coefficients exhibits somewhat evenly distributed filters with no clear cluster patterns (Fig. 4(b)). This example reveals an important characteristics of a well-trained model that the diversity of trained filters is generally desirable in achieving a satisfactory classification accuracy.

Furthermore, the reconstructed images of the clustered filters shown in the lower-left part in Fig. 4(a) mostly show the blue-colored patterns. This indicates that these filters consider only a single color channel from an input image instead of all the three color channels. On the contrary, when the model shows a relatively good performance, the reconstructed images of filters is shown to involve all the different colors in a balanced manner with no distinct color patterns. Based on such diverse color patterns, our observation reveals that those filters that combine all the channels of the previous activation maps can be generally desirable in achieving a good generalization capability of the trained model.



(a) Batch loss per epoch (b) Zoomed-in view after epoch 20

Figure 6: Training loss comparison between the original model and the layer-freezed model



(a) Training accuracy (b) Validation accuracy

Figure 7: Training and Validation accuracy per epoch for the original model and the Conv1,2 Skipped (Epoch at 4) model on CIFAR-10 dataset. The vertical dotted line displays a particular epoch at which the layer was freezed.

4.3 Dynamic model steering

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.

Freezing layers

A main obstacle facing the use of deep learning is the time-consuming nature of its training process. One novel interaction supported by ReVACNN is called 'layer freezing,' which skips the gradient descent update of particular layers and/or nodes to reduce the subsequent training time. We observed that this interaction performed in the middle of the training process maintains the comparable training loss values to or even better values than the original case without this interaction involved.

In the example shown in Fig. 5, the model where Conv 1 and 2 layers are skipped for its gradient descent update after epoch 4 shows significant decrease in the training time, e.g., at least 4 seconds per epoch, compared to the original model. In this example, we decided to perform this interaction at epoch 4 since the t-SNE view of the filters in the corresponding layers showed stable patterns by their short to no trailing lines.

Additionally, the validation loss and the validation accuracy values by the model with our interaction were shown to be sometimes slightly better than the original model, as

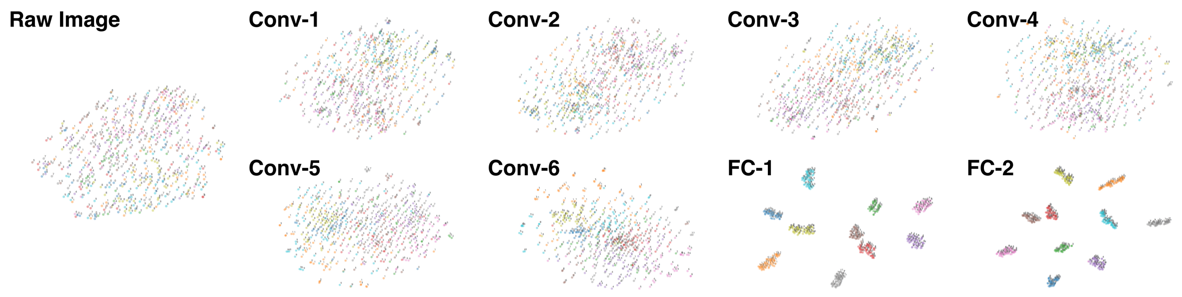
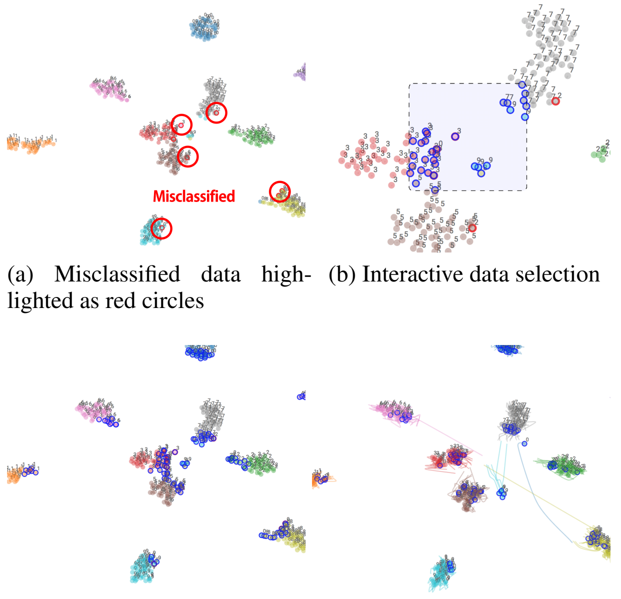


Figure 8: t-SNE view of activation maps across layers



(a) Misclassified data highlighted as red circles (b) Interactive data selection

(c) Before retraining (d) After retraining

Figure 9: Model retraining using user-selected data

shown in Fig. 6. In other words, the model with our interaction actually converged faster and achieved a higher training accuracy than the original model. At last, as shown in Fig. 7, one can see no noticeable differences in the training accuracy and validation accuracy between the model with our interaction and the original model.

Retraining the model with user-selected data

Our main model steering capability provided in the 2D embedding view is for users to interactively select the misclassified or ambiguous data located near decision boundary and retrain the model based on them. First, we browse t-SNE view through across different layers, as shown in Fig. 8 and select the FC-1 layer to initiate our interaction from. As shown in Fig. 9(a), data points in the middle are shown as misclassified. Afterwards, we select the data around them via mouse dragging interaction, as shown in Fig. 9(b).

At this point, the fully trained CNN model shows the training accuracy of 98% and the test accuracy of 80%. We now retrain the model using these selected data with additional randomly selected to match the batch size. After retraining the data with 700 iterations we found all the selected misclassified ones now correctly classified. As shown in Fig. 9(c), before retraining, we can see two class cluster are close to each other. But after retraining with the user-selected data (Fig 9(d)) two clusters become clearly separated. Furthermore, their enhanced separations are also visualized as the opposite direction of data trails after this interaction is performed. Although not reported, we verified that the validation accuracy still remained the same, not impacting the overall model performance.

This scenario shows that one can interactively steer the training process of a model to focus more on particularly confusing classes and data items that users choose.

5 Conclusion and Future Work

In this paper, we proposed ReVACNN, a real-time visual analytics system for convolutional neural networks. ReVACNN supports the task of exploring and steering CNNs in real time via various visualization modules including a 2D embedding view by t-SNE as well as novel interaction capabilities to speed up the training processes and train the model based on user-selected data for performance improvement.

Our work opens up various challenges and opportunities towards interactive, user-driven deep neural networks. One such example would be interactive network structure building, which may involve dynamic addition/removal of nodes and layers, and it poses several interesting research questions: (1) how to initialize the newly added filters and nodes so that they can capture complementary information of data to the existing nodes/layers and (2) how to provide informative measures as to which nodes/layers to remove while having a minimal impact to the overall performance.

The visual patterns and associated interaction capabilities presented in this paper, such as a redundant, clustered filter patterns, may give some hints towards this research directions, and many more characteristics and treatments can be discovered by other visual analytic approaches.

Acknowledgments

Blinded for review.

References

[Bahdanau *et al.*, 2014] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

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

[Collobert and Weston, 2008] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Prof. the 25th International Conference on Machine Learning (ICML)*, pages 160–167, 2008.

[Elman, 1990] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179 – 211, 1990.

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

[Hinton *et al.*, 2012] 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. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[Hochreiter and Schmidhuber, 1997] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, Nov 1997.

[Krizhevsky and Hinton, 2009] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[Krizhevsky *et al.*, 2012] 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.

[LeCun *et al.*, 1998] 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.

[Liu *et al.*, 2016] Mengchen Liu, Jiaxin Shi, Zhen Li, Chongxuan Li, Jun Zhu, and Shixia Liu. Towards better analysis of deep convolutional neural networks. *arXiv preprint arXiv:1604.07043*, 2016.

[Rauber *et al.*, 2017] Paulo E Rauber, Samuel G Fadel, Alexandre X Falcao, and Alexandru C Telea. Visualizing the hidden activity of artificial neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):101–110, 2017.

[Smilkov *et al.*, 2016] Daniel Smilkov, Shan Carter, D. Sculley, Fernanda B. Viegas, and Martin Wattenberg. Direct-manipulation visualization of deep networks. In *ICML Workshop on Visualization for Deep Learning*, 2016.

[Strobelt *et al.*, 2016] Hendrik Strobelt, Sebastian Gehrmann, Bernd Huber, Hanspeter Pfister, and Alexander M Rush. Visual analysis of hidden state dynamics in recurrent neural networks. *arXiv preprint arXiv:1606.07461*, 2016.

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

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

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