Towards Better Analysis of Deep Convolutional Neural Networks

Mengchen Liu, Jiaxin Shi, Zhen Li, Chongxuan Li, Jun Zhu, Shixia Liu

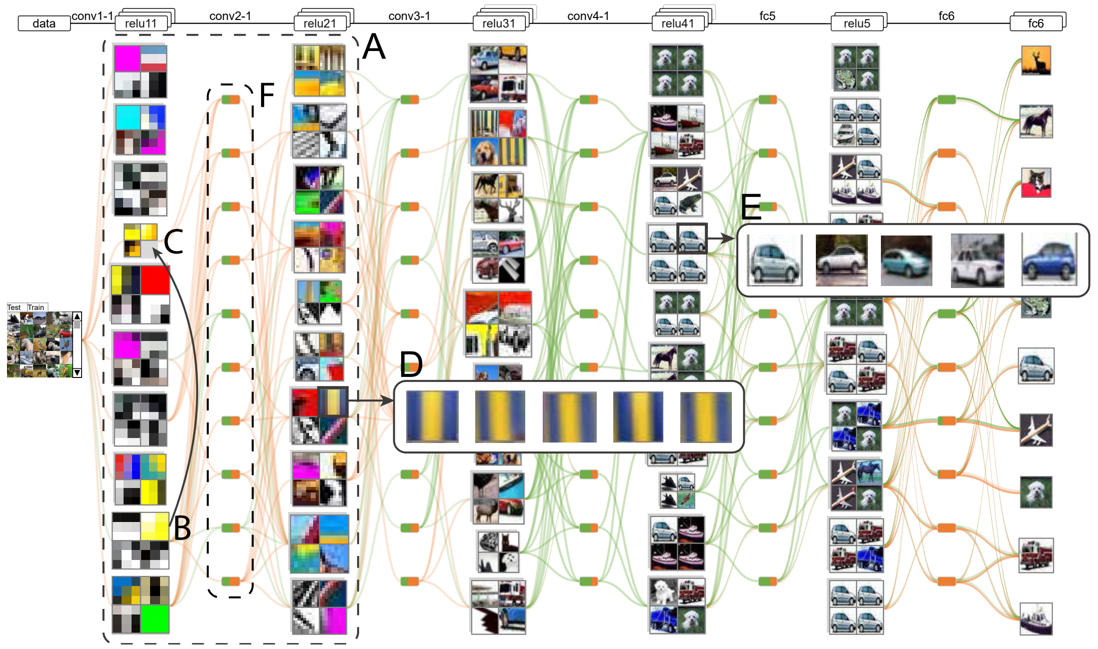

Fig. 1. CNNVis, a visual analytics system that helps experts understand, diagnose, and refine deep CNNs.

Abstract— Deep convolutional neural networks (CNNs) have achieved breakthrough performance in many pattern recognition tasks such as image classification. However, the development of high-quality deep models typically relies on a substantial amount of trial-and-error, as there is still no clear understanding of when and why a deep model works. In this paper, we present a visual analytics approach for better understanding, diagnosing, and refining deep CNNs. We formulate a deep CNN as a directed acyclic graph. Based on this formulation, a hybrid visualization is developed to disclose the multiple facets of each neuron and the interactions between them. In particular, we introduce a hierarchical rectangle packing algorithm and a matrix reordering algorithm to show the derived features of a neuron cluster. We also propose a biclustering-based edge bundling method to reduce visual clutter caused by a large number of connections between neurons. We evaluated our method on a set of CNNs and the results are generally favorable.

Index Terms—Deep convolutional neural networks, rectangle packing, matrix reordering, edge bundling, biclustering

1 Introduction

Deep convolutional neural networks (CNNs) have demonstrated significant improvements over traditional approaches in many pattern recognition tasks [34], such as image classification [22, 33] and video classification [30, 59]. More recently, deep CNNs have been employed as function approximators in deep reinforcement learning to extract robust representations and help make decisions, which has led to human-level performance in intelligent tasks such as Atari games [43] and the game of Go [48]. However, a deep CNN is often treated as a "black box" model because of its incomprehensible functions and unclear working mechanism [5]. It is generally difficult for machine learning experts to understand the role of each component (neuron, connection) due to the large number of interacting, non-linear parts in a CNN. Without a clear understanding of how and why these networks work, the development of high-performance models typically relies on a time-consuming trial-and-error procedure [4, 5, 58]. For example, training a single deep CNN on a large dataset may take several days or even weeks.

There are two technical challenges to understand and analyze deep CNNs. First, a CNN may consist of tens or hundreds of layers (depth), with thousands of neurons (width) in each layer, as well as millions of connections between neurons. Such large CNNs are hard to study due to the sizes involved. Second, CNNs consist of many functional components whose values and roles are not well understood either as individuals or as a whole [5]. In addition, how the non-linear components interact with each other and with other linear components in a CNN is not well understood by experts. In most cases, it is hard to summarize reusable knowledge from a failed or successful training case and transfer it to the development of other relevant deep learning models.

To tackle these challenges, we have developed an interactive, visual analytics system called CNNVis, which aims to help machine learning experts better understand, diagnose, and refine CNNs. In CNNVis, diagnosis helps experts analyze a training process that has failed to converge; while refinement aims to find a potential direction to improve model accuracy. Based on the characteristics of a deep CNN, we formulate it as

• M. Liu, Z. Li, and S. Liu are with the School of Software and TNList, Tsinghua University. Email: {liumc13,zhen-li11}@mails.tsinghua.edu.cn; shixia@tsinghua.edu.cn. S. Liu is the corresponding author.
• J. Shi, C. Li, and J. Zhu are with Dept. of Comp. Sci. & Tech., State Key Lab of Intell. Tech. & Sys., TNList Lab, CBICR Center. E-mail: {shijx15,licx14}@mails.tsinghua.edu.cn; dcszj@tsinghua.edu.cn.

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

a directed acyclic graph (DAG), in which each node represents a neuron and each edge represents the connection between neurons. In order to visualize a large CNN, we first cluster the layers in the network and select a representative one from each layer cluster. We then cluster neurons in each representative layer and select several representative neurons from each neuron cluster. On the basis of the DAG representation, we develop a hybrid visualization to disclose the interactions between neurons and the multiple facets of each neuron by indicating its role for different types of images. In particular, we have developed a hierarchical rectangle packing algorithm to show the derived features of a neuron cluster. We have also designed a matrix reordering algorithm based on the Held-Karp algorithm [23] to demonstrate the cluster patterns in the activations of each neuron cluster. Here, the activation is the output value of a neuron, which is determined by the activation function that transforms the input value to the output value of the neuron. Moreover, we have proposed a biclustering-based edge bundling method to reduce visual clutter caused by the large number of connections between neurons.

To support our research, we used image classification as an example and conducted three case studies with experts. The case studies have shown that CNNVis allows experts to better explore and understand a deep CNN, including the role of each neuron and the connections between neurons. For example, the neurons in the lower layers are learned to detect simple patterns such as corners and stripes. Furthermore, experts can diagnose the potential issues of a model and refine a CNN, which enables more rapid iteration and faster convergence in model construction.

The key technical contributions of this work are:

- **A visual analytics system** that helps experts understand, diagnose, and refine deep CNNs.
- **A hybrid visualization** that combines a DAG with rectangle packing, matrix visualization, and a biclustering-based edge bundling method.

2 Related Work

To help experts gain a better understanding of a deep CNN, researchers in the field of computer vision have strived to illustrate the learned features of each neuron, which is represented by part of a real image or a synthesized image. Existing methods can be classified into two categories, namely, code inversion [15, 41, 59] and activation maximization [17, 49, 58].

The code inversion method synthesizes an image from the activation vector of a specific layer, which is produced by a real image. For example, Zeiler et al. [59] utilized a multi-layered Deconvolutional Network [60] to project the activations onto the input pixel space. However, a simple projection without any consideration for priors will produce images that do not resemble natural images. To solve this problem, Mahendran et al. [41] proposed incorporating several natural image priors like α-norm and total variation to make the reconstructed images more realistic. Recently, Dosovitskiy et al. [15] trained a CNN to reconstruct the images from the activations. They argued that a CNN can learn more powerful priors and have better performance than manually defined priors.

The activation maximization method aims to find an image that maximally activates a given neuron. It can be modeled as an optimization problem over the image space. Similar to the code inversion method, natural image priors are used as regularization during the optimization to obtain realistic images. As a result, most activation maximization methods focus on defining the regularization term using natural image priors [17, 58]. For example, Erhan et al. [17] constrained the L2-norm of the image to be a constant. Yosinski et al. [58] defined several more powerful priors, including Gaussian blur, clipping pixels with a small norm, and clipping pixels with a small contribution.

The aforementioned methods employ a grid-based representation to display the neuron features. Although they can show the reconstructed intermediate states of each layer, they fail to disclose the inner working mechanisms of CNNs, especially the role of each neuron for different types of images and the interactions between neurons. Unlike these methods, we formulate a deep CNN as a DAG. Based on the DAG representation, we have developed a hybrid visualization that consists of rectangle packing, matrix ordering, and biclustering-based edge bundling. Empowered by the hybrid visualization, our visual analytics approach well discloses the multiple facets of each neuron and the interactions between them, which is very useful for understanding the inner working mechanism of a deep CNN.

In the field of visual analytics, researchers have developed interactive image classification systems [3, 47, 51]. These systems provide users with visual interfaces to guide them to choose new training examples and evaluate the model quality. They usually employ line charts or scatter plots to demonstrate the model quality. As a result, none of these systems can be directly applied to analyze the inner mechanisms of deep CNNs because they fail to illustrate the neurons and the connections between them.

More relevant to our work is to visualize neural networks as node-link diagrams [21, 56]. For example, Tzeng et al. [56] employed a DAG to represent a neural network. Although this visualization method can illustrate the interactions between neurons, it suffers from serious visual clutter when handling large neural networks. To address this issue, we improved existing DAG layout methods [18, 40, 52, 55, 57]. In particular, we first cluster the layers in the network and select a representative from each layer cluster. Then we cluster neurons in each representative layer and select several representative neurons from each neuron cluster. Based on BiSet [53], we have proposed a biclustering-based edge bundling method to reduce visual clutter caused by a large number of connections between neurons.

3 Background

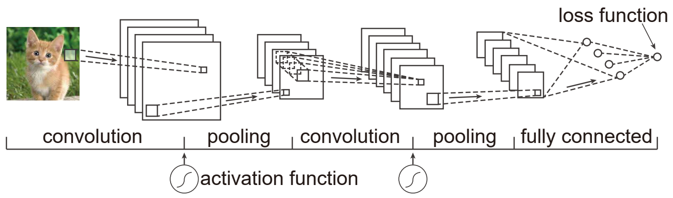

Fig. 2. The typical architecture of a CNN.

In this section, we briefly introduce the architecture of CNNs and several basic concepts, which will be useful for the subsequent discussion.

CNNs are a special kind of neural network for processing data that has a known, grid-like topology [34]. A typical CNN is structured as a series of stages (Fig. 2).

In the first few stages, there are two kinds of layers: convolutional layers and pooling layers. In a convolutional layer, each neuron is connected to local patches of the previous layer through a set of weights. The result of this local weighted sum is then input to an activation function. An activation function is a non-linear transformation that can prevent CNNs from learning trivial linear combinations of the inputs. While the convolutional layer aims to detect local combinations of features from the previous layer, the pooling layer aims to merge semantically similar features into one. A pooling operation computes a summary statistic (e.g., maximum) of a local patch of the input. Adopting pooling in a CNN has two benefits. First, pooling allows the output to vary very little when the input vary in position and appearance. Second, pooling can significantly reduce computational cost when the network contains many layers.

Several stages of convolution, activation function, and pooling are stacked, followed by one or several fully connected layers. Then, at the output of the model, a loss function is adopted as a means of measuring performance, namely, the difference between the output of a CNN and a true image label (i.e., the loss). The goal of training a CNN is to minimize the loss function. This is usually achieved with stochastic gradient descent [7], an optimization method that first calculates the gradient of the loss function with respect to the weight of each edge in the network and then updates the weight according to the computed gradient.

4 CNNVis

CNNVis was designed in consultation with a team of deep learning experts (six researchers) over the course of twelve months. For simplicity's sake, we denote these experts as E_i ($i = 1, 2, \cdots, 6$). We held

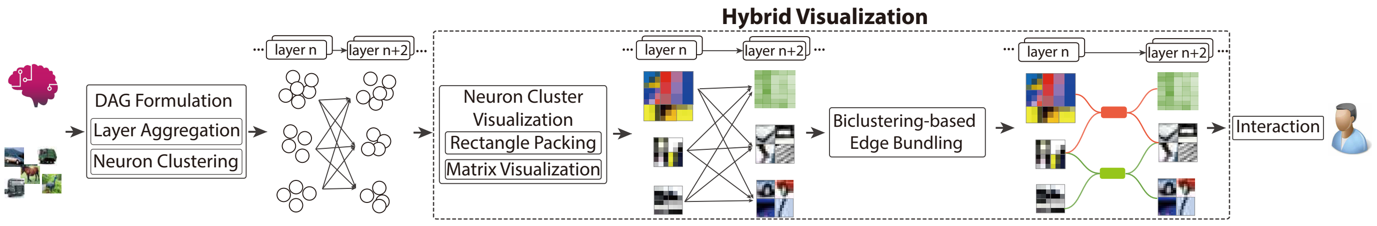

Fig. 3. CNNVis pipeline.

discussions every two weeks. Three co-authors of this paper are also members of the team. The development of CNNVis was triggered by their need to make sense of the inner mechanisms of deep CNNs and their dissatisfaction with the state-of-the-art tools.

Common deep learning frameworks include Caffe [28], Theano [6], Torch [10], and TensorFlow [1]. Researchers can use these frameworks to train, debug, and deploy CNNs. Although the deep learning frameworks output high-level statistical information, such as training loss, as well as debugging information, such as the learned features of neurons and the gradients of weights, it fails to disclose the role of each neuron for different categories of images and how the neurons work together. Accordingly, if a training process fails, it is difficult for experts to figure out what is wrong with the current model design. The experts commented that the development of high-quality CNN models is usually a trial-and-error procedure. To solve this problem, they expressed the need for a tool that supports the following functions:

- Understanding: study the influence of the network architecture;
- Diagnosis: diagnose a training process that failed to converge;
- Refinement: find a potential direction to improve the model.

4.1 Requirement Analysis

We identified the following high-level requirements based on our discussions with the experts and previous research.

R1 - Providing an overview of the learned features of neurons. All the experts commented that an overview of the learned features of neurons is necessary to begin their analysis (e.g., diagnosis or refinement of the model). They usually examine the quality of each learned feature layer by layer to discover potential problems. However, such an examination can be very difficult for a deep CNN with tens or hundreds of layers and thousands of neurons in a layer. As a result, they stated the need to cluster neurons into clusters so they can gain a quick overview of the learned features of each cluster.

R2 - Interactively modifying the neuron clustering results. Since the clustering algorithm may be imperfect and different users may have different needs, experts need to interactively modify the clustering results based on their knowledge. Expert E_2 commented that when examining the training results of a CNN, he found a neuron for detecting a color patch in a cluster that mainly consists of neurons for detecting black and white local patterns (Fig. 1B). To increase the clustering accuracy and better compare these clusters, he moved the neuron to a cluster that mainly consisted of neurons for detecting color patches.

R3 - Exploring multiple facets of neurons. Previous work mainly focused on visualizing the learned features of neurons. In addition to this feature, the experts also requested the ability to view other facets of neurons. For example, expert E_1 said, "In addition to the learned features, other numerical features such as activation (of a neuron) can also help me better understand its role in a classification task." During the discussion, we gradually identified that the major facets of interest are the learned features (all the experts), activations (E_1, E_3, E_4, E_5, E_6), and contributions to the final result (all the experts). Visually illustrating them can help experts gain a more comprehensive understanding of the roles of neurons.

R4 - Revealing how low-level features are aggregated into high-level features. In a CNN, neurons in lower layers learn to detect simple features such as stripes or corners, neurons in middle layers learn to detect a part of an object, and neurons in higher layers learn to detect a concept (e.g., a cat). This is achieved with a local connectivity pattern between neurons of adjacent layers, which means the inputs of neurons in layer m are from a subset of neurons in layer $m-1$. As a result, the experts wanted to learn how neurons in adjacent layers interact with each other and aggregate the low-level features into high-level features. Previous research has also shown that analyzing such connections can help experts understand how a large number of non-linear parts interact with each other [56]. A large CNN may contain millions of connections between neurons. If we display all of them, it is difficult to discern an individual connection due to visual clutter caused by excessive edges and edge crossings. Thus, the experts requested the ability to examine the major trends among these connections.

R5 - Examining the debugging information. In the discussions, the experts expressed the need to examine the debugging information of the deep model. Expert E_3 said, "I often examine the debugging information such as the gradients, to diagnose a training process that failed to converge." In addition to gradients, showing other derived values such as the relative changes of weights, was also requested by the experts. The amount of debugging information is usually huge. For example, there are millions of gradients. It is very hard to examine them one by one and develop a full understanding. As a result, the experts also requested having an overview of such debugging information. This need is consistent with the findings of previous research [5, 20].

4.2 System Overview

The list of requirements has motivated us to develop a visual analytics system, CNNVis, consisting of the following components:

- A DAG formulation module that converts a CNN to a DAG and aggregates neurons and layers for an overview (**R1**,**R4**);
- A neuron cluster visualization module that discloses multiple facets of each neuron (**R3**);
- A biclustering-based edge bundling that reduces visual clutter caused by a large number of connections (**R4**);
- An interaction module that provides a set of interactions such as interactive clustering result modification (**R2**) and shows debugging information on demand (**R5**).

As shown in Fig. 3, CNNVis takes a trained CNN and the corresponding training dataset as the input. The input CNN is formulated as a DAG with each node representing a neuron and each edge representing the connection between neurons. To effectively present a large CNN, the DAG formulation module clusters the neurons in each layer. The clustered DAG is then passed to the neuron cluster visualization module. This module employs a rectangle packing algorithm to show the learned features of each neuron in a cluster and a matrix visualization to depict the activations of neurons. Next, a biclustering-based edge bundling clusters the edges to reduce visual clutter. Users can also interact with the generated visualization for further analysis.

The visualization is shown in Fig. 1. The large rectangle with packed images represents a neuron cluster. An "in-between" layer between the input and output neuron clusters is used to represent the biclustering-based edge bundling result (Fig. 1F). With this visualization, the expert can get an overview of the model structure as well as the learned feature of each layer. For example, the expert found that the neurons in the lower layers learned to detect low-level features such as corners, color patches, and stripes (Fig. 1A). He identified a neuron for detecting a color patch in a cluster that mainly consisted of neurons for detecting black and white local patterns (Fig. 1B). To better compare the neurons that detect color patches, he dragged the neuron to a cluster that mainly consisted of neurons for detecting color patches (Fig. 1C).

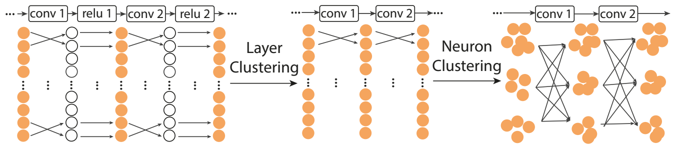

Fig. 4. Illustration of the DAG formulation.

5 DAG Formulation

A CNN can be formulated as a DAG, where each node represents a neuron and each edge represents the connection between neurons. To effectively present a large CNN with tens or hundreds of layers and thousands of neurons in each layer, we first aggregate adjacent layers into groups. There are several ways to do this aggregation. For example, we can classify layers by merging two adjacent convolutional layers that have a small difference between their activation variance. We can also divide layers into groups at each pooling layer. In our current implementation, we employ the second method. In addition, the experts are interested in the output of an activation layer instead of that of a convolutional layer. As the outputs of these two layers have a one-to-one mapping relationship, we merge these two layers and simply show the output of the activation layer (Fig. 4).

We then cluster the neurons in each layer with the aim of grouping neurons with similar roles together. We assume that neurons with similar activations have similar roles. Directly using these activations to cluster the neurons is very time-consuming as there can be millions of images in the training set. Thus, we aggregate the activations into an average activation vector over the set of classes in the training set. In particular, suppose the training samples can be categorized into m classes: $c_1, c_2, ..., c_m$. The training samples of class c_i is represented by: $S_i = \{s_1^{(i)}, s_2^{(i)}, \cdots, s_{N_i}^{(i)}\}$, where N_i is the number of training samples in class c_i. We first process each training sample $s_j^{(i)}$ through the network and obtain the activation of neuron n: $a_n(s_j^{(i)})$. Then we calculate the average activation $a_n(c_i)$ of neuron n on class c_i by:

$$\frac{1}{N_i}\sum_{j=1}^{N_i} a_n(s_j^{(i)}). \quad (1)$$

Next, we combine each average activation into an m-dimension real-valued activation vector $\vec{a}_n = [a_n(c_1), a_n(c_2), ..., a_n(c_m)]$.

Finally, we cluster the neurons based on the derived activation vectors. In CNNVis, we employ two widely used clustering methods, K-Means [42] (parametric clustering) and MeanShift [11] (nonparametric clustering). The second method does not require prior knowledge of the number of clusters. Thus, it is applicable when experts do not know the number of clusters. To better present each neuron cluster, we select several representative neurons that are closer to the cluster centroid.

6 Visualization

6.1 Overview

Based on the aforementioned requirements (Sec. 4) and the DAG formulation, we have designed a hybrid visualization (Fig. 5) that illustrates neuron clusters (nodes) and the connections between neurons (edges).

Each neuron cluster is represented by a large rectangle (Fig. 5A), which can be analyzed from multiple facets, such as the learned features, activations, and contributions to the final result (**R3**). Specifically, we have adopted a rectangle packing algorithm to place the learned features of neurons in a neuron cluster, where each learned feature is encoded by a smaller rectangle (Fig. 5B1). Neuron activations are visualized as a matrix visualization (Fig. 5B2). Users can switch between the rectangle packing representation and the matrix visualization to explore different facets of the neurons.

To reduce visual clutter caused by dense edges and their crossings, we have developed a biclustering-based edge bundling algorithm (**R4**). For each layer, we first generate the biclusters between the input neuron clusters and output neuron clusters. Inspired by BiSet [53], we have also added an “in-between” layer between the input neuron clusters

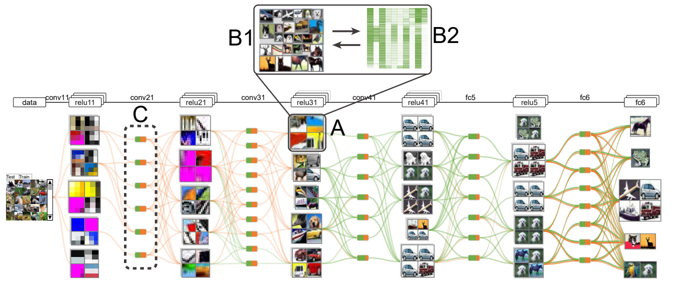

Fig. 5. Visualization overview: A is a neuron cluster, B1 and B2 are two facets of a neuron cluster, and C is an in-between layer to represent the biclusters of edges.

and output neuron clusters (Fig. 5C). In this layer, each bicluster is treated as a node in the DAG and is represented by a small rectangle.

In CNNVis, we employ the layout algorithm in TextFlow [12, 13] to calculate the position of each node (e.g., neuron cluster or bicluster) (**R1**). We also provide a set of interactions to facilitate the exploration of a deep CNN (**R2**, **R5**).

Next, we will introduce the neuron cluster visualization and biclustering-based edge bundling in detail.

6.2 Neuron Cluster Visualization

6.2.1 Learned Features as Rectangle Packing

Computing learned features of neurons. We employ the method used in [19] to compute the learned feature of a neuron because it is fast and the results are easy to understand. We also compute the activations of each neuron on a large set of image patches (e.g., sampled from the training set) and sort the patches in decreasing order according to their activations. To help experts better understand the role of each neuron, we select the top-5 patches with the highest activation scores to represent the learned feature of that neuron. By default, we show the top patch for a neuron and allow users to switch among these five patches. Other methods for computing the learned feature [41, 59] can easily also be integrated into CNNVis.

Layout. A straightforward way to visualize the learned features (image patches) is to employ a grid-based layout where each image patch is represented by a rectangle of the same size [58, 59]. However, this method fails to emphasize the important neurons.

To tackle this issue, we formulate the layout of image patches as a rectangle packing problem, aiming to pack the given rectangles into an enclosing rectangle of a minimum area. We use the size of an image patch to encode the importance of the corresponding neuron because size is among the most effective visual channels [44]. In CNNVis, we provide several options to define the importance of a neuron, including its average or maximal activation on a set of classes and its contribution to the final result [37].

Existing rectangle packing algorithms [26, 31] can handle a small number of rectangles well (e.g., 15 rectangles in less than 0.1s [26]). However, the computing time grows exponentially as the number of packed rectangles increases (e.g., 25 rectangles in more than one hour [26]). Since a neuron cluster may consist of hundreds or even thousands of neurons, existing rectangle packing algorithms cannot directly be applied to our visualization.

To solve this problem, we have developed a hierarchical rectangle packing algorithm. The basic idea of the algorithm is to divide the

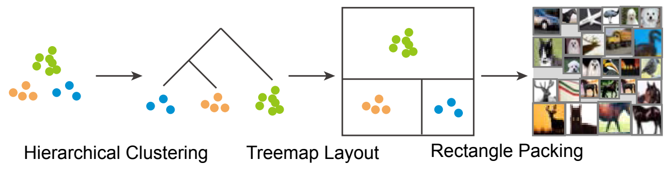

Fig. 6. Illustration of hierarchical rectangle packing.

problem into several smaller sub-problems. Each sub-problem can be efficiently solved by the state-of-the-art rectangle packing algorithm [26]. Specifically, our algorithm contains the following steps (Fig. 6).
Step 1: Hierarchical clustering. In this step, we perform a hierarchical clustering to divide the image patches into several groups. Specifically, we start with the cluster containing all of the neurons. Then we repeatedly split a cluster until the number of neurons in it is smaller than a threshold. This cluster splitting is done with a widely used graph clustering method [46].
Step 2: Computing the layout area for each cluster. Based on the hierarchical clustering results, we compute the layout area for each sub-cluster using a Treemap layout algorithm [29].
Step 3: Rectangle packing of each cluster. In this step, we compute the position and size for each image patch using the state-of-the-art rectangle packing algorithm [26].

6.2.2 Activations as Matrix Visualization

In our first prototype, we simply encode the activation of a neuron according to its size. However, the experts we consulted were not satisfied with such a design because it failed to help them compare the roles of the neurons for different classes of images. To allow experts to compare different neurons, we stack the average activation vectors of neurons into an activation matrix, where each row is an average activation vector of a neuron. Accordingly, a matrix visualization is employed to visually illustrate the activation of the neurons. In particular, the color of a cell in the i-th row and j-th column represents the average activation of the i-th neuron n_i in class c_j.

This design was then presented to our experts for evaluation. Overall, they liked the matrix visualization that provides a global overview of the activations for different classes. Their major concern was that the current visualization cannot reveal the cluster patterns in the activations of a neuron cluster. To solve this problem, we developed a matrix reordering algorithm that can visually reveal cluster patterns within the data.
Matrix Reordering. The order of columns (classes) should be consistent in different neuron clusters. Otherwise, experts are unable to directly compare the roles of neurons in two neuron clusters because of the different order of classes (columns). As a result, we only reorder the rows (neurons) in the matrix.

The basic idea of our algorithm is to maximize the sum of the similarities between adjacent neurons in the matrix. The aim is to place neurons with similar activations close to each other, thus revealing the cluster pattern in the neuron cluster. Given neuron cluster $C = \{n_1, n_2, \cdots, n_{N_C}\}$, the goal of the reordering is to find a row index $\pi(i)$ for each neuron n_i, to better reveal the cluster pattern in a neuron cluster. For row r in the matrix, we denote its corresponding neuron as $n_{\pi^{-1}(r)}$. To achieve this goal, we try to maximize the sum of the similarities between adjacent neurons in the matrix:

$$max \sum_{r=1}^{N_C-1} sim(n_{\pi^{-1}(r)}, n_{\pi^{-1}(r+1)}), \tag{2}$$

where $sim()$ is the similarity function between two neurons. In CNNVis, we adopt the widely used cosine similarity.

This combinational optimization problem can be solved by the Held-Karp algorithm [23] with a time complexity of $O(2^{N_C} \cdot N_C^2)$, where N_C is the number of neurons. The problem of directly applying it in our system is that we may have hundreds of neurons in a neuron cluster and the running time of the algorithm is very long. Thus, we developed a divide-and-conquer method to accelerate the algorithm, which consists of the following steps.
Divide. If the number of neurons in a cluster is too large to be efficiently solved via directly running the Held-Karp algorithm, the cluster is divided into several sub-clusters by a widely used graph clustering method developed by Newman [46].
Conquer. Computing the ordering of sub-clusters by running the Held-Karp algorithm.
Combine. Merging the ordering of sub-clusters into a global ordering.

Fig. 7 shows one result generated using our reordering method. With this method, several clusters can easily be detected.

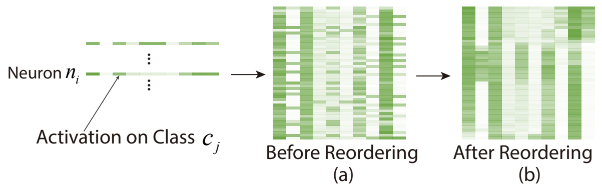

Fig. 7. Matrix reordering: (a) before reordering; (b) after reordering.

6.2.3 Interaction

To better facilitate understanding of the multiple facets of each neuron cluster, CNNVis provides a set of user interactions.
Interactive clustering result modification. Since the clustering algorithm is not perfect and experts may have different needs, we allow experts to interactively modify the clustering results based on their knowledge (**R2**). Inspired by NodeTrix [24], we allow experts to drag a neuron out of a neuron cluster or to another neuron cluster.
Selecting particular neurons to view. There are thousands of neurons in a CNN. Thus, it is necessary to allow experts to view only some of the neurons. We allow users to select a set of classes and show the neurons that are strongly activated by the images in these classes. Any irrelevant neurons are deemphasized by setting them to be translucent.
Switching between Facets. Exploring the multiple facets of neurons can help experts better understand the roles of neurons. One way to visualize multiple facets is to overlay one facet on top of another [8, 9]. However, in our scenario, each facet already contains too much information. Overlaying multiple facets can easily lead to users becoming overwhelmed. This problem is confirmed by our experts. They said that they were more interested in analyzing each single facet. Thus, we decided to allow users to switch between these facets [38] instead of overlaying multiple facets (**R3**). For example, users can view the learned features or the activation matrix.

6.3 Biclustering-based Edge Bundling

Initially, we visualized each edge as a curve. The major concern of the experts is visual clutter caused by millions of edges between nodes.

In order to reduce visual clutter, we tried the geometry-based edge bundling method [14, 25, 39] to cluster the edges between two layers. After interacting with CNNVis, the experts commented that this bundling method reduces visual clutter to some extent. However, the clusters revealed by the geometry-based bundling methods did not help their analysis because the edges with similar weights were not clustered together. The experts are more interested in edges with larger absolute weights, because this indicates that the corresponding inputs have a larger impact on the output.

To fulfill this requirement, we developed a biclustering-based edge bundling method to bundle edges with both similar and large absolute weights. For a given layer, a bicluster is a subset of input neuron clusters and a subset of output neuron clusters. This method can logically aggregate multiple individual connections and thus provides an opportunity to visually bundle edges between neuron clusters. Our method was inspired by the biclustering method in BiSet [53]. In BiSet, each edge is unweighted. While in a CNN, each edge has a weight and we need to bundle the edges with similar weights. If we simply employ the biclustering method in BiSet, we may lose some important biclusters with larger weights and fewer edge numbers. To solve this problem, we have developed a weighted biclustering method, which consists of the following steps (Fig. 8).
Step 1: Aggregating Connections between Neurons. We first calculate the strength w_{ij} of the connection e_{ij} between two neuron clusters, C_i and C_j. We denote $E = \{e_{ij}\}$ as the edge set. An intuitive approach is to use the average of all the weights of the edges that connect neurons in C_i and C_j. The problem with this method is that it aggregates positive edges (edges with positive weights) and negative edges (edges with negative weights) and may result in an aggregated edge with a small weight. This may lead to a misunderstanding. Therefore, we calculate the strength of the connection between two neuron clusters as a two-dimensional vector $\vec{w}_{ij} = [w_{ij}^{pos}, w_{ij}^{neg}]$, where w_{ij}^{pos} is the average of the positive edge weights and w_{ij}^{neg} is the average of the negative edge weights.

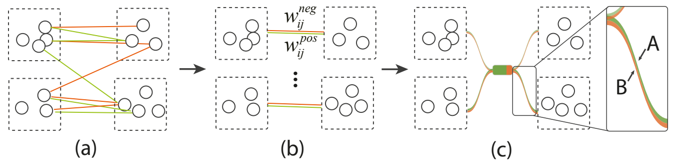

Fig. 8. Illustration of biclustering-based edge bundling.

Step 2: Biclustering. Based on the aggregation results, we then detect biclusters between the input neuron clusters and the output neuron clusters. Because experts are interested in both larger positive edges and smaller negative edges, we cannot simply convert it to an unweighted graph and perform biclustering. Thus, we first seek the maximum value w_{max} in $W = \{w_{ij}^{pos}\} \cup \{|w_{ij}^{neg}|\}$. If $w_{max} \in \{w_{ij}^{pos}\}$, then we select the edges satisfying: $|w_{ij}^{pos} - w_{max}| < \tau$, where τ is a user-defined parameter denoting the tolerance of similarity. If $w_{max} \in \{|w_{ij}^{neg}|\}$, we then perform a similar extraction. For these edges, we then mine the closed item sets as biclusters, where each input neuron cluster is connected to each output neuron cluster. To mine the closed item sets, we adopt the widely used Apriori, an algorithm for frequent item set mining [2]. After that, we remove the edges in the extracted biclusters from E and then repeat the process until w_{max} is under a user-defined threshold.
Step 3: Edge Bundling. In this step, we bundled the edges in the same bicluster to reduce visual clutter. Inspired by BiSet [53], we also add an "in-between" layer between the input and output neuron clusters (Fig. 8 (c)). In this layer, each bicluster is visualized as a rectangle. In a bicluster, we use two colored regions (green and red) to indicate the proportion between the number of positive edges and negative edges. An edge between two neuron clusters consists of two aggregated curves (A, B in Fig. 8), where green and red visually encode positive and negative weights, respectively. Since experts are less interested in analyzing edges with smaller absolute weights, they are not displayed by default. These edges can be shown by user request.
Interaction. The debugging information can help experts diagnose a failed training process. In CNNVis, we allow experts to analyze the debugging information at different granularities (**R5**). For example, they can change the color coding of edges to analyze the gradient of each weight. Experts also have the option to view the average gradient at each layer as a line chart to get an overview of the debugging information.

7 Application

In this section, we present the case studies to demonstrate how CNNVis help experts understand, diagnose and refine a CNN.

7.1 Overview

We have worked closely with the team of experts to select the base CNN model and to design the case studies.

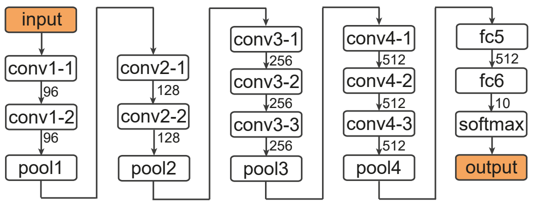

Fig. 9. The architecture of BaseCNN. It contains four groups of convolutional layers and two fully connected layers. The number below a layer is the number of neurons in that layer.

Base CNN. The base CNN was contributed by E_3 of the expert team. For brevity's sake, we refer to the base CNN as BaseCNN. BaseCNN was designed based on a widely used deep CNN introduced in [50], which is often used in image classification. Recently, the expert team that we collaborate with has been redesigning this CNN and testing the performance of the variants. BaseCNN consists of 10 convolutional layers and two fully connected layers. The convolutional layers are organized into four groups, containing 2, 2, 3, and 3 convolutional layers, respectively. Each group ends with a max-pooling layer, which outputs the maximum pixel value of the input region. When designing BaseCNN, the expert employed a commonly used activation function, rectified linear unit (ReLU) [45]. ReLU is a piecewise linear function that prunes the negative part of the input to zero and retains the positive part. In addition, he chose cross-entropy to measure the difference between the output of a CNN and a true image label (i.e., the loss). The architecture of BaseCNN is depicted in Fig. 9.

BaseCNN was trained and tested on a benchmark image dataset, CIFAR10 [32], which consists of 60,000 labeled color images of size 32×32 in 10 different classes (e.g., airplane, bird, and truck), with 6,000 images per class. The dataset was split into a training set containing 50,000 images and a test set containing 10,000 images. Training and testing of BaseCNN were performed under a widely used deep learning framework, Caffe [28]. The BaseCNN model achieved an 11.32% error rate on the test set.
Design of Case Studies. We have worked closely with the expert team to design three case studies from their current research on CNNs.

First, based on BaseCNN, the expert team constructed several variants and aimed to study the influence of the network architecture on performance. The experts said that such an analysis would help to better understand why CNNs with different architectures perform differently (Sec. 7.2).

Second, the expert team required the diagnosis of a training process that failed to converge. For example, in one training trial, E_3 changed the output activation function and the loss function of BaseCNN. However, the training failed. The expert team wanted to diagnose the training process and find potential issues. This scenario triggered the second case study (Sec. 7.3).

Finally, the expert team wanted to further improve the model accuracy of the BaseCNN model. To this end, the expert team decided to examine the output of each layer from a global overview to local details and detected a potential direction to improve the model. This requirement is addressed in the third case study.

Due to the page limit, we have focused on the first two case studies. Interested readers may refer to the attached video and supplemental material for the study on model refinement (third case study).

7.2 Case Study: Influence of Network Architecture

This case study was a collaboration with expert E_2. E_2 is focused on combining CNNs with deep generative models. In this case study, E_2 evaluated the effectiveness of CNNVis on a set of variants of BaseCNN (with different depths and widths) qualitatively based on his experience. He also checked the possibility of selecting a CNN with a suitable architecture under the guide of CNNVis. Though a number of high-performance models can be referred to in benchmark datasets, it usually takes a long time to transfer the experience to other scenarios (e.g., choose a suitable CNN on a new dataset). Therefore, E_2 emphasized that a systematic study of the network architecture and its influence on performance will help them decide on an appropriate network architecture in their research.
Overview of BaseCNN. We first provided expert E_2 with an overview of BaseCNN (Fig. 1) to evaluate the quality of CNNVis.

From the overview, he identified that the neurons in the lower layers learned to detect low-level features such as corners, color patches, and stripes (Fig. 1A). A similar observation was reported in previous work [33]. Switching between the top-5 image patches that highly activate a given neuron in lower layers (Fig. 1D), he noticed that the retrieved patches did not show much difference in appearance. Then he turned to higher layers. After exploring the top-5 image patches for a given neuron in higher layers (Fig. 1E), he noticed that these neurons could learn to detect high-level features (e.g., an automobile). He concluded that, "The ability to detect more abstract features in the higher layers is a nice property of well-trained deep CNNs and CNNVis indeed shows this pattern well."

To evaluate the ability of CNNVis to illustrate the finer details of CNNs, E_2 selected two similar classes (automobiles and trucks) and then examined the activation patterns of the relevant neurons. From the learned features in the lower layers, he found some commonalities between trucks and automobiles, such as wheels (A1, A2 in Fig. 10(a)).

He indicated that these features are not sufficient to distinguish the two classes. Thus, he expanded the 4-th group of convolutional layers for further examination (Fig. 10(b)). Expert E_2 noticed that the number of "impure" neuron clusters (B1-B3 in Fig. 10(b)) gradually decreased as he moved to the higher layers. Here, an "impure" neuron cluster means that the image patches that maximally activate the neurons in the cluster are from different classes. Examining the "purity" means that we check the ability of a CNN to distinguish different images from different classes. In a pure cluster, the image patches that have the same class label are gathered together in the activation space generated by the outputs of the layer. In the lower layers, we prefer "impure" clusters because we want the neurons to detect as many different kinds of features as possible. While in higher layers, we prefer "pure" clusters because we want the model to separate different classes by a large margin, so that the image patches from different classes seldom exist in the same cluster. The expert commented that this criterion would apply to other CNNs as well. We illustrate this criterion in Fig. 11. For example, in the top convolutional layer of BaseCNN, all clusters look "pure," which indicates that the output activations of the images given by BaseCNN match well with their corresponding classes.

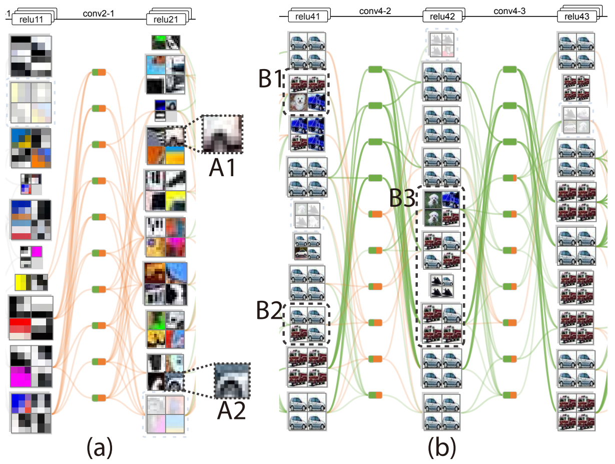

Fig. 10. Learned features of BaseCNN: (a) low-level feature; (b) high-level features.

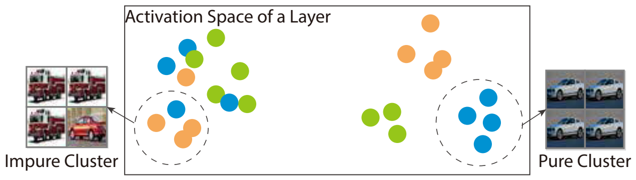

Fig. 11. Illustration of an "impure" cluster and a "pure" cluster.

Network Depth. E_2 further investigated how the depth of the network affects the features detected by the neurons. He compared BaseCNN with two variant models, including ShallowCNN, which cuts off the 4-th group of convolutional layers, and DeepCNN, which doubles the number of convolutional layers. The architectures and accuracies are summarized in Table 1.

Table 1. Performance comparison between CNNs with different depth. "#ConvLayers" is the number of convolutional layers and "#Layers" is the number of layers that can be visualized.

	Error	#ConvLayers	#Layers
ShallowCNN	11.94%	7	30
BaseCNN	11.33%	10	40
DeepCNN	14.77%	20	70

He also selected the truck and automobile classes, and expanded the last group of convolutional layers (Fig. 12(a)). In ShallowCNN, he identified that there were indeed a lot more "impure" clusters in the top convolutional layers compared to those in BaseCNN, which

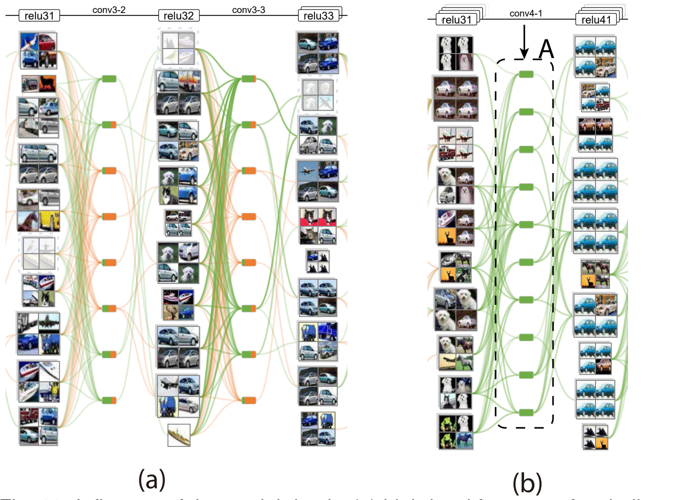

Fig. 12. Influence of the model depth: (a) high-level features of a shallow CNN; (b) A layer whose weights are almost positive in DeepCNN.

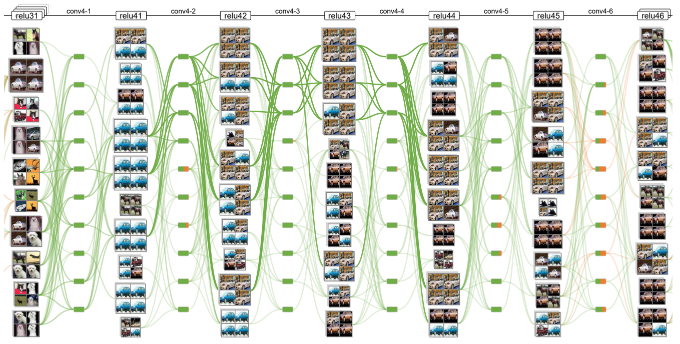

Fig. 13. Consecutive convolutional layers whose weights are almost positive in DeepCNN.

indicates that a model without a sufficiently large depth is often incapable of distinguishing images from similar classes, which can lead to a decrease in performance. In DeepCNN, expert E_2 noticed that almost all the edges in the first convolutional layer in the 4-th group were green (A in Fig. 12(b)). This indicated that almost all the weights in that layer were positive. The expert commented that since the inputs of that layer were non-negative, the outputs were mostly positive. The outputs were then fed into ReLU. Because ReLU retains the positive part of the input, the ReLU, together with its corresponding convolutional layer, can be viewed as a close-to-linear function. By further expanding the 4-th group of convolutional layers, expert E_2 identified several consecutive layers that had similar patterns (Fig. 13). Because the composition of linear functions is still linear, he concluded that this phenomenon indicates redundancy in the layers. He also commented that such redundancy may hurt overall performance and make the learning process computationally expensive and statistically ineffective. These findings are consistent with previous research [54]. E_2 then concluded that CNNVis could be used to check the abstractness of the features extracted by CNNs.

Network Width. Another important factor that influences performance is the width of a CNN. To have a comprehensive understanding of its influence, E_2 evaluated several variants of BaseCNN with different widths, named by BaseCNN$\times w$, where w denotes the ratio of the number of neurons in a layer compared to that of BaseCNN. For example, BaseCNN$\times$4 contains four times the neurons of BaseCNN. In the case study, w is selected from $\{4, 2, 0.5, 0.25\}$. The architecture and performance of these variants as well as BaseCNN are listed in Table 2.

Compared to BaseCNN, a wider network (BaseCNN$\times$4) has a much lower training loss than testing loss. The expert commented that this phenomenon is known as overfitting in the field of machine learning. It means that the network tries to model every minor variation in the input,

Table 2. Performance comparison between CNNs with different widths. #Params is the number of parameters in the model, which is measured in millions.

	Error	#Params	Training loss	Testing loss
BaseCNN×4	12.33%	4.22M	0.04	0.51
BaseCNN×2	11.47%	2.11M	0.07	0.43
BaseCNN	11.33%	1.05M	0.16	0.40
BaseCNN×0.5	12.61%	0.53M	0.34	0.40
BaseCNN×0.25	17.39%	0.26M	0.65	0.53

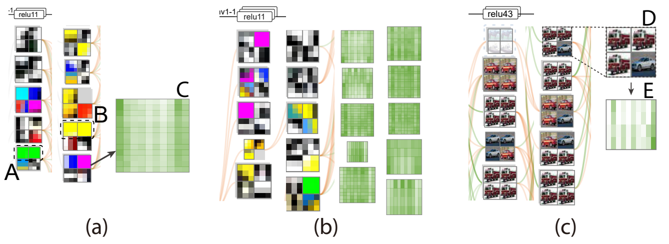

Fig. 14. Comparison between models with different widths: (a) low-level features of BaseCNN×4; (b) low-level features of BaseCNN; (c) high-level features of BaseCNN×0.25

which is more likely to be noise. It often occurs when we have too many parameters relative to the number of training samples. When a model overfits, its performance on the testing set will be much worse than that on the training set. E_2 wanted to examine the influence of overfitting on CNNs. He visualized BaseCNN×4 with our visual analytics system.

After examining high-level features, the expert did not find much difference compared to BaseCNN. Then he switched to examine low-level features. He instantly found that the same image was shown in multiple neurons (A, B in Fig. 14(a). It indicated that these neurons learn to detect almost the same features. The expert inferred that there may be redundant neurons in an overfitting CNN. For further verification, he decided to examine the activations of the neurons in this cluster. Compared to the activations in lower layers of BaseCNN (Fig. 14(b)), he found that many neurons have very similar activations (C in Fig. 14(a)). This observation verified that there are redundant neurons in the lower layers of a CNN that is too wide.

E_2 commented, "We often use a quantitative criterion (e.g., accuracy) to evaluate the quality of a model. However, a quantitative criterion itself cannot provide sufficient intuition and clear guidelines. Even when I know a CNN overfits, it is hard to decide which layer to narrow down or remove. CNNVis can guide me in locating the candidate layers, which is very useful in my research."

E_2 then compared the performance of BaseCNN with narrower networks (BaseCNN×0.5 and BaseCNN×0.25). Although the training loss and testing loss of these narrower networks are comparable, which indicate that these narrow networks generalize well, their performance was worse than BaseCNN (Table 2). The expert explained that this phenomenon is known as underfitting. It happens when the task is complex but we are trying to use a simple model to perform the task. In image classification, one of the major disadvantage of underfitting is that the model is too simple to distinguish images from similar classes (e.g., automobiles and trucks). In addition to the decrease in accuracy, he wanted to know the influence that underfitting brought to the model.

The expert visualized BaseCNN×0.25 for further exploration. He selected two similar classes, automobile and truck, to examine the patterns of relevant neurons. After analyzing low-level features, he did not find much difference compared to BaseCNN. Thus, he switched his attention to high-level features. When examining the features of the last convolutional layer, he found that there were several "impure" neuron clusters. For example, cluster D in Fig. 14(c) is represented by three trucks and an automobile (outlier). He switched to explore the activations in this cluster (E in Fig. 14(c)). The expert found the outlier has similar activations on the two classes (i.e., truck and automobile), which means that this neuron does not distinguish automobiles from trucks. As a result, the ability of the model to correctly classify images from similar classes is hindered, which is reflected in the decrease in accuracy.

Expert E_2 commented, "It is really hard for me to choose the architec-

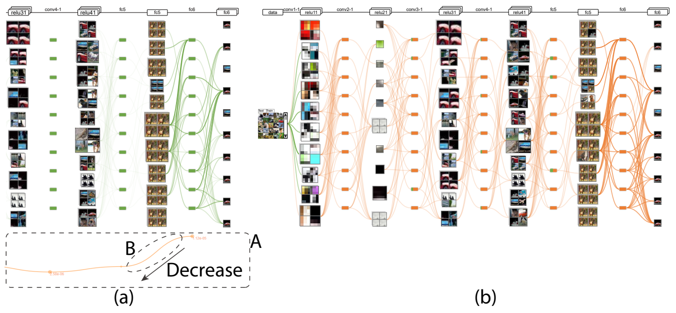

Fig. 15. Exploring the connections between neurons: (a) edge colors encode the relative changes of weights and the line chart represents their average relative changes; (b) edges colors encode the weights.

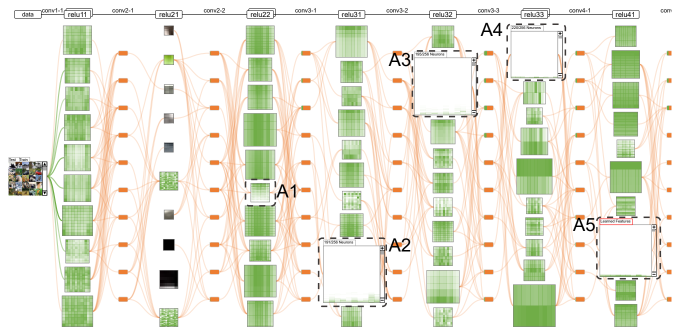

Fig. 16. Exploring the neuron clusters.

ture, including the depth and width of the network on a new dataset, as there are not many high-quality deep models to refer to. I usually need to try a series of parameters to achieve a satisfactory performance. CNNVis can intuitively show the quality of the model in various ways, such as the purity of clusters, and help transfer knowledge from previous experience and enable me find the suitable architecture more quickly."

7.3 Case Study: Training Diagnosis

This case study demonstrates how CNNVis helps an expert (E_3) diagnose a failed training process. E_3 is a deep learning researcher who focuses on integrating concept learning with deep learning. Recently, during the research triggered by [36], E_3 tried to construct a variant of BaseCNN. Specifically, he replaced the loss function with hinge loss, which measures the difference between the score of the correct class and the score of the predicted class. However, the training of this model failed. The problem was that the training process got stuck when the loss decreased to around 2.0, where the model was far from achieving good accuracy.

To help the expert diagnose the failed training process, we provided him with the visualization of a snapshot after the training process got stuck. Because he often used the relative changes of weights to diagnose a training process in his previous research, he set the initial color coding of edges as the relative changes of weights.

Based on the overview, expert E_3 observed that the edges were difficult to recognize after the top-2 layers (Fig. 15(a)). This indicated that the relative changes of weights were very small. He then used the line chart (A in Fig. 15(a)) to check the average relative changes in weights. He found that the average relative changes decreased a great deal after the top-2 layers (B in Fig. 15(a)), which caused the training process to become stuck. E_3 was curious about what led to such small relative changes in weights, so he used the color of edges to represent the weights. He immediately identified that an overwhelming majority of edges were negative (Fig. 15(b)).

He wanted to find what influence the negative weights had on the model. As the learned features could not reveal much information due to the failed training process, expert E_3 switched to examining the activation matrices. He found some activation matrices were essentially blank. This indicated that some neurons had zero activations on all classes. To further study this phenomenon, he sequentially expanded the second, third, and fourth groups of convolutional layers. He found that the ratio of neurons with zero activations became larger and larger from the lower layers to the higher layers (A1-A5 in Fig. 16). The activation functions of these neurons were ReLUs. He continued to zoom in and further examine the inputs fed into the ReLUs, which he found were always negative. Thus, if the input of a ReLU is less than zero, it generates a zero activation.

Expert E_3 explained that because the input of each convolutional layer was the output of the ReLUs in the previous layer, it must be nonnegative. As the weights of the linear transformation in this layer were mostly negative, the values fed into ReLUs were mostly negative. Consequently, the outputs of ReLUs were mostly zeros. In the training method that we used (i.e., stochastic gradient descent [7]), zero outputs of a neuron mean zero updates to its weights.

Having learned why the training process got stuck, expert E_3 proposed a method to force the network away from that situation. He added a batch-normalization layer [27] after each convolutional and fully connected layer, and before the ReLU activation function. A batch-normalization layer can normalize the output of its corresponding convolutional or fully connected layer through linear operations. With batch-normalization, the input fed into the ReLUs should no longer be mostly negative. This means that the model can still be trained even when most weights are negative.

The improved model achieved an error rate of 9.43% on the CIFAR-10 dataset, with which expert E_3 was very satisfied. He commented, "I have investigated this problem for a long time and inserted all kinds of code fragments to print the debugging information during training. However, after many unsuccessful attempts and a great deal of effort spent reading the debugging information, I eventually gave up. It is awesome to have a tool like CNNVis, which intuitively illustrates the training statistics and allows me to explore the training process from multiple perspectives." The expert also expressed that he would try batch-normalization when using hinge loss in his future research. This is one important lesson he has learned.

8 Discussion

Lessons Learned. We have learned two lessons in the process of collaborating with the machine learning experts.

First, sometimes experts may not clearly know what they exactly want to investigate before they use the prototype. For example, in one discussion, the experts required that the edges need to be clustered to reduce visual clutter (Sec. 6.3), but had no idea how the edges should be clustered. In order to explore the experts' requirements, we first tried traditional geometry-based edge bundling methods. After seeing the new prototype, they instantly realized that this was not what they wanted and expressed a need for clustering edges with similar weights together. Accordingly, this requires us to quickly develop new prototypes based on the expert feedback.

Second, using the data that experts are interested in is crucial for gaining insight. For example, in our development process, we first adopted relatively simple data to accelerate the implementation. In particular, we used a handwritten digit dataset (MNIST [35]) and a traditional network (LeNet [35]), which only contains two convolutional layers. Using simple data can help us more easily verify the correctness of the system and more effectively introduce the basic idea to the experts. However, after sticking to this data for a long time, we gained little useful insight for diagnosis and model refinement because this data was too simple for these tasks. As a result, we decided to switch to a more complex dataset (CIFAR10) and a real network the experts were working on (BaseCNN), which indeed raised several issues in their research and development process. The switch helped us to gain much more useful insights (Secs. 7.2 and 7.3).

Limitations. Our case studies demonstrate the effectiveness of CNNVis. Nevertheless, there are still several limitations of CNNVis.

First, CNNVis cannot visualize deep models that cannot be formulated as DAGs because we adopted a DAG layout to calculate the positions of neurons. Recurrent neural networks (RNNs) [34] are one example. In a RNN, connections between neurons form a directed cycle, and thus a RNN cannot be formulated as a DAG. This problem can be potentially solved by unfolding a RNN into a very deep DAG [34]. However, the depth may cause additional computational costs.

Second, the scalability of the activation matrix is limited. When the number of classes is large (e.g., >100), an activation matrix will have too many columns. It will be difficult to examine the activation of a neuron on each class. Since experts may use a very large dataset with hundreds or thousands of classes, improving the scalability of the activation matrix is desirable. For example, the ImageNet dataset [33] contains more than 1,000 classes. This problem can be solved by clustering similar classes and aggregating the activations in the same class cluster.

Third, there is a learning curve associated with the system. It took the experts about one or two hours to become fully familiar with the visual encodings and interactions of CNNVis. The visual design of the neuron cluster seems to be the most confusing part for some of them. Initially, two experts thought a large rectangle (e.g., D in Fig. 14) was a neuron instead of a neuron cluster. After communicating with them, we found that this misunderstanding was caused by aggregating the edges connecting two neuron clusters into one edge. It made them mistake a neuron cluster for a neuron, and further mistake the representative neurons shown in a cluster for the learned features of a single neuron. The experts suggested that we provide a more intuitive visual design for the neuron cluster. This will shorten the time of getting familiar with the system. Furthermore, it will enable them to adopt CNNVis as an educational tool to illustrate the working mechanism of a CNN for average users or researchers from other fields. Possible solutions are to employ the range traversal technique [16] or provide additional visual hint(s).

9 Conclusion

In this paper, we have presented a novel visual analytics system to help machine learning experts better understand, diagnose, and refine CNNs. Powered by a hybrid visualization consisting of rectangle packing, matrix ordering, and biclustering-based edge bundling, the system allows experts to explore and understand a deep CNN from different perspectives. In addition, it enables experts to diagnose and refine the CNN architecture to further improve performance. Three case studies were conducted to demonstrate the effectiveness and usefulness of the system for comprehensive analysis of CNNs.

There are several directions for future work to further improve our system. Currently, CNNVis focuses on analyzing a snapshot of a CNN model in the training process, which is useful for conducting offline analysis. All the experts expressed the need to integrate CNNVis with the online training process and continuously get an update of the training status. A key issue is the difficulty of selecting representative snapshots and comparing them effectively.

Another interesting venue for future work is to apply CNNVis to other types of deep models that cannot be formulated as a DAG, such as a recurrent neural network (RNN). The major bottleneck is to design an effective visualization to facilitate experts in understanding the data flow through different types of deep models. For example, in addition to the conventional multi-layer neural network, RNN has a feedback loop from an output to an input. Better understanding of the working principle of the feedback loop helps experts design more effective models.

Acknowledgments

M. Liu, Z. Li, and S. Liu are supported by National NSF of China and a Microsoft Research Fund (No. FY15-RES-OPP-112). J. Shi, C. Li, and J. Zhu are supported by the National Basic Research Program (973 Program) of China (No.2013CB329403), National NSF of China (Nos. 61322308, 61332007), and the Youth Top-notch Talent Support Program. The authors would like to thank Liu Jiang and Xizhou Zhu for insightful discussions and help in case studies.

References

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

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, pages 487–499, 1994.

[3] S. Amershi, J. Fogarty, A. Kapoor, and D. Tan. Examining multiple potential models in end-user interactive concept learning. In *CHI*, pages 1357–1360. ACM, 2010.

[4] Y. Bengio. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[5] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE PAMI*, 35(8):1798–1828, 2013.

[6] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *SciPy*, 2010.

[7] L. Bottou. Stochastic gradient learning in neural networks. *Neuro-Nımes*, 91(8), 1991.

[8] M. Chen and H. Jaenicke. An information-theoretic framework for visualization. *IEEE TVCG*, 16(6):1206–1215, 2010.

[9] M. Chen, S. Walton, K. Berger, J. Thiyagalingam, B. Duffy, H. Fang, C. Holloway, and A. E. Trefethen. Visual multiplexing. *CGF*, 33(3):241–250, 2014.

[10] R. Collobert, S. Bengio, and J. Mariéthoz. Torch: a modular machine learning software library. Technical report, IDIAP, 2002.

[11] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE PAMI*, 24(5):603–619, 2002.

[12] W. Cui, S. Liu, L. Tan, C. Shi, Y. Song, Z. J. Gao, H. Qu, and X. Tong. Textflow: Towards better understanding of evolving topics in text. *IEEE TVCG*, 17(12):2412–2421, 2011.

[13] W. Cui, S. Liu, Z. Wu, and H. Wei. How hierarchical topics evolve in large text corpora. *IEEE TVCG*, 20(12):2281–2290, 2014.

[14] W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *IEEE TVCG*, 14(6):1277–1284, 2008.

[15] A. Dosovitskiy and T. Brox. Inverting visual representations with convolutional networks. *arXiv preprint arXiv:1506.02753*, 2015.

[16] N. Elmqvist and J.-D. Fekete. Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines. *IEEE TVCG*, 16(3):439–454, 2010.

[17] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. Technical report, University of Montreal, 2009.

[18] S. Gad, W. Javed, S. Ghani, N. Elmqvist, T. Ewing, K. N. Hampton, and N. Ramakrishnan. Themedelta: Dynamic segmentations over temporal topic models. *IEEE TVCG*, 21(5):672–685, 2015.

[19] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, pages 580–587, 2014.

[20] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, pages 249–256, 2010.

[21] A. W. Harley. An interactive node-link visualization of convolutional neural networks. In *ISVC*, pages 867–877, 2015.

[22] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. To appear.

[23] M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *SIAM*, 10(1):196–210, 1962.

[24] N. Henry, J.-D. Fekete, and M. J. McGuffin. Nodetrix: a hybrid visualization of social networks. *IEEE TVCG*, 13(6):1302–1309, 2007.

[25] D. Holten and J. J. Van Wijk. Force-directed edge bundling for graph visualization. *CGF*, 28(3):983–990, 2009.

[26] E. Huang and R. E. Korf. Optimal rectangle packing: An absolute placement approach. *JAIR*, 46:47–87, 2012.

[27] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015.

[28] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[29] B. Johnson and B. Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Visualization*, pages 284–291, 1991.

[30] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, pages 1725–1732, 2014.

[31] R. E. Korf, M. D. Moffitt, and M. E. Pollack. Optimal rectangle packing. *Annals of Operations Research*, 179(1):261–295, 2010.

[32] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Montreal, 2009.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[34] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[35] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *IEEE*, 86(11):2278–2324, 1998.

[36] C. Li, J. Zhu, T. Shi, and B. Zhang. Max-margin deep generative models. In *NIPS*, pages 1828–1836, 2015.

[37] H. Li, T. Jiang, and K. Zhang. Efficient and robust feature extraction by maximum margin criterion. *Neural Networks*, 17(1):157–165, 2006.

[38] M. Liu, S. Liu, X. Zhu, Q. Liao, F. Wei, and S. Pan. An uncertainty-aware approach for exploratory microblog retrieval. *IEEE TVCG*, 22(1):250–259, 2016.

[39] S. Liu, W. Cui, Y. Wu, and M. Liu. A survey on information visualization: recent advances and challenges. *The Visual Computer*, 30(12):1373–1393, 2014.

[40] S. Liu, J. Yin, X. Wang, W. Cui, K. Cao, and J. Pei. Online visual analytics of text streams. *To appear in IEEE TVCG*, 2015.

[41] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, pages 5188–5196, 2015.

[42] S. Marsland. *Machine learning: an algorithmic perspective*. CRC press, 2015.

[43] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[44] T. Munzner. *Visualization Analysis and Design*. CRC Press, 2014.

[45] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

[46] M. E. Newman. Fast algorithm for detecting community structure in networks. *Physical review E*, 69(6):066133, 2004.

[47] J. G. S. Paiva, W. R. Schwartz, H. Pedrini, and R. Minghim. An approach to supporting incremental visual data classification. *IEEE TVCG*, 21(1):4–17, 2015.

[48] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[49] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR Workshop*, 2013.

[50] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[51] G. Sun, Y. Wu, R. Liang, and S. Liu. A survey of visual analytics techniques and applications: State-of-the-art research and future challenges. *JCST*, 28(5):852–867, 2013.

[52] G. Sun, Y. Wu, S. Liu, T. Q. Peng, J. J. H. Zhu, and R. Liang. Evoriver: Visual analysis of topic coopetition on social media. *IEEE TVCG*, 20(12):1753–1762, 2014.

[53] M. Sun, P. Mi, C. North, and N. Ramakrishnan. Biset: Semantic edge bundling with biclusters for sensemaking. *IEEE TVCG*, 22(1):310–319, 2016.

[54] S. Sun, W. Chen, L. Wang, X. Liu, and T.-Y. Liu. On the depth of deep neural networks: A theoretical view. In *AAAI*, pages 2066–2072, 2016.

[55] Y. Tanahashi, C. H. Hsueh, and K. L. Ma. An efficient framework for generating storyline visualizations from streaming data. *IEEE TVCG*, 21(6):730–742, 2015.

[56] F. Y. Tzeng and K. L. Ma. Opening the black box - data driven visualization of neural networks. In *IEEE VIS*, pages 383–390, 2005.

[57] Y. Wu, N. Pitipornvivat, J. Zhao, S. Yang, G. Huang, and H. Qu. egoslider: Visual analysis of egocentric network evolution. *IEEE TVCG*, 22(1):260–269, 2016.

[58] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. In *ICML Workshop on Deep Learning*, 2015.

[59] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*, pages 818–833, 2014.

[60] M. D. Zeiler, G. W. Taylor, and R. Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *ICCV*, pages 2018–2025, 2011.