

DeepTracker: Visualizing the Training Process of Convolutional Neural Networks

DONGYU LIU, Hong Kong University of Science and Technology

WEIWEI CUI, KAI JIN, and YUXIAO GUO, Microsoft Research Asia

HUAMIN QU, Hong Kong University of Science and Technology

Deep Convolutional Neural Networks (CNNs) have achieved remarkable success in various fields. However, training an excellent CNN is practically a trial-and-error process that consumes a tremendous amount of time and computer resources. To accelerate the training process and reduce the number of trials, experts need to understand what has occurred in the training process and why the resulting CNN behaves as it does. However, current popular training platforms, such as TensorFlow, only provide very little and general information, such as training/validation errors, which is far from enough to serve this purpose. To bridge this gap and help domain experts with their training tasks in a practical environment, we propose a visual analytics system, DeepTracker, to facilitate the exploration of the rich dynamics of CNN training processes and to identify the unusual patterns that are hidden behind the huge amount of information in training log. Specifically, we combine a hierarchical index mechanism and a set of hierarchical small multiples to help experts explore the entire training log from different levels of detail. We also introduce a novel cube-style visualization to reveal the complex correlations among multiple types of heterogeneous training data, including neuron weights, validation images, and training iterations. Three case studies are conducted to demonstrate how DeepTracker provides its users with valuable knowledge in an industry-level CNN training process; namely, in our case, training ResNet-50 on the ImageNet dataset. We show that our method can be easily applied to other state-of-the-art “very deep” CNN models.

CCS Concepts: • **Human-centered computing** → **Visualization**; **Visualization application domains**; **Visual analytics**;

Additional Key Words and Phrases: Deep learning, training process, multiple time series, visual analytics, correlation analysis

ACM Reference format:

Dongyu Liu, Weiwei Cui, Kai Jin, Yuxiao Guo, and Huamin Qu. 2018. DeepTracker: Visualizing the Training Process of Convolutional Neural Networks. *ACM Trans. Intell. Syst. Technol.* 10, 1, Article 6 (November 2018), 25 pages.

<https://doi.org/10.1145/3200489>

The work is supported by the National Basic Research Program of China (973 program) under Grant No. 2014CB340304 and ITC Grant with No. UIT/138.

Authors' addresses: D. Liu, CYT 3007, HKUST, Clear Water Bay, Kowloon, Hong Kong; email: dliuae@cse.ust.hk; W. Cui, K. Jin, and Y. Guo, Building 2, No. 5 Dan Ling Street, Haidian District, Beijing, 100080, China; emails: weiwei.cui@microsoft.com, jinkai@cnu.edu.cn, yuxiao.guo@outlook.com; H. Qu, Rm 2529, Academic Building, HKUST, Clear Water Bay, Kowloon, Hong Kong; email: huamin@cse.ust.hk.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 Association for Computing Machinery.

2157-6904/2018/11-ART6 \$15.00

<https://doi.org/10.1145/3200489>

1 INTRODUCTION

Deep Convolutional Neural Networks (CNNs) have achieved huge success in solving problems related to computer vision, such as image classification [23, 35], object detection [13], and semantic segmentation [28]. However, in practice, training a high-quality CNN is often a complicated, confusing, and tedious trial-and-error procedure [6]. For a large CNN, one complete training trial may take a couple of weeks. However, domain experts often have to repeat this process several times with slightly different settings to obtain a satisfying network, which may take several weeks or even months. To accelerate this process, experts have to understand the training processes further to check whether they are on the right track, find latent mistakes, and make proper adjustments in the next trial. Visualizing the concealed rich training dynamics (e.g., the changes of loss/accuracy and weights/gradients/activations over time) is of vital importance to understanding the CNN training process. Unfortunately, CNNs usually contain a large number of interacting and nonlinear parts [7] and recently have become wider and deeper [17, 35, 37, 38]. Both of these issues bring considerable difficulties for experts in reasoning about CNN training behaviors.

Many previous studies investigate what features have been learned by a CNN in one (e.g., usually the last one) or several representative snapshots during the training process [4, 11, 12, 21, 27, 30–32, 36, 41, 43]. However, little research focuses on visualizing the overall training dynamics. One recent work [26] visualizes the training process of deep neural networks, but it is not tailored for CNNs and not scalable enough to analyze those CNNs that are not only wide but also deep. In addition, tools like TensorBoard, Nvidia Digits, and DeepLearning4j¹ are able to show some high-level training dynamics, such as loss and the mean of weights in a layer. However, these tools can neither handle industry-level training (i.e., training a large CNN on a very large dataset) nor answer complex questions. For example: How does the model's performance on each class of images changes over time? How do the changes of parameters impact the classification results for each class? Given so many layers or image classes, which of them are worth paying more attention to, and what is the best manner to support comparative analysis? With these concerns, we are in urgent need of a scalable visualization solution to conduct more advanced analytical tasks.

To this end, we must deal with two major challenges. First, the system needs to handle the large-scale training log data. Typically, millions of CNN parameters and tens of thousands of validation images are involved in a training process. In addition, training is an iteration-based process that usually requires a million iterations to complete, which makes things worse because the parameters and classification results need to be recorded for every few iterations. In our experiments (Section 6), a sampled training log may easily exceed a couple of terabytes per training. To allow an interactive exploration of the data at such scale requires not only an effective data storage and index mechanism but also a scalable visualization technique. Second, the log information is heterogeneous. The full log contains structural (e.g., neural network), numeric (e.g., neuron weights), image (e.g., validation dataset), and nominal data (e.g., classification results). Given that significant insights are often hidden underneath the complex relationships among these data, our system also needs to present all these types of data intuitively and assist experts in their analysis tasks.

To address these challenges, we use a downsampling method to store the raw data, and then we preprocess and organize these data in a hierarchical manner. We also design several efficient index mechanisms to support real-time interactions. To help experts identify the iterations of interest quickly, we propose an application-specific anomaly detection method. We also integrate many filtering and aggregation approaches to reduce the amount of presenting information and ensure

¹TensorBoard: https://www.tensorflow.org/get_started/summaries_and_tensorboard; Nvidia Digits: <https://developer.nvidia.com/digits>; DeepLearning4j: <https://deeplearning4j.org/visualization>.

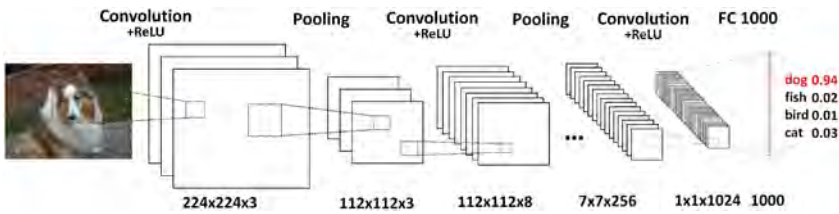


Fig. 1. Illustration of a CNN architecture that contains three types of layers (i.e., CONV layer, POOL layer, and FC layer) and transforms an image volume into a class score vector.

the preservation of noteworthy information. For visualization, we design a set of hierarchical small multiples that is combined with a network structure layout to facilitate an effective exploration of the entire training log from different levels of detail. To reveal the complex correlations among neuron weights, validation images, and training iterations, we further introduce a cube-style visualization. The cube integrates the small multiples and a matrix-based correlation view, thereby allowing experts to effectively slice and dice the data from different perspectives.

The main contributions of this article are summarized as follows:

- A systematic characterization of the problem of visualizing the rich dynamics in CNN training processes and a thorough discussion and summary of the design requirements and space.
- A visual analytics system that integrates a tailored large data storage and index mechanism, an anomaly iteration detection algorithm, and a set of well-designed visualization techniques.
- A couple of new visualization and interaction techniques, including hierarchical small multiples, grid-based correlation view, and cube-style visualization.

2 BACKGROUND

A typical CNN can be viewed as a sequence of layers (Figure 1) that transforms an image volume (e.g., a 224×224 image with three color channels of R, G, and B) into an output volume (e.g., a vector of size 1,000 indicating the probability for an input image to belong to 1,000 predefined classes) [24]. There are three main types of layers used to build a CNN architecture: a *convolutional layer* (CONV layer), *pooling layer* (POOL layer), and *fully connected layer* (FC layer).

A CONV layer comprises numerous neurons that are connected to a local region in the previous layer's output volume through weighted edges, many of which share the same weights through a parameter sharing scheme. The weights in each neuron compose a *filter*, the basic unit for detecting *visual features* in the input image, such as a blotch of color or the shape of an area. The output of each neuron is computed via a dot product operation between the weights and inputs from the previous layer, and is then optionally applied via an elementwise activation function (e.g., ReLU, $\max(0, x)$). A POOL layer is usually inserted between successive CONV layers to reduce the volume of input through a downsampling operation, thereby reducing the amount of parameters and computation in the network. The only difference between the FC layer and CONV layer is that, in contrast to the neurons in CONV layer that are locally connected and have shared parameters, the neurons in FC layers have full connections to the previous layer's output volume. In addition, the output of the last FC layer is fed to a classifier (e.g., Softmax) that computes the scores for all predefined classes, where each score represents the probability that an input image belongs to the corresponding class.

To obtain an effective CNN, the weight parameters in the CONV and FC layers need to be trained using gradient descent methods [9] to ensure consistency between the predicted class labels and the predefined class for each training image. Specifically, a training process involves two separate datasets: the training set D_t and the validation set D_v . To start the training, the parameters of weighted layers are usually initialized via Gaussian distribution [14], and D_t is partitioned into non-overlapping batches. For each iteration, one batch of images in D_t is fed to the network. Afterward, the output classification results are compared with the ground truth (i.e., the known labels of the fed images) to compute the *gradients* with respect to all neurons. These gradients are then used to update the weights of each neuron. When all batches in D_t are completed (i.e., finish one *epoch*), D_t is reshuffled and partitioned into new non-overlapping batches. After several epochs, the initially randomized neural network will be gradually shaped into a specified network targeting a specific task. Meanwhile, for every given number of iterations, the network is evaluated via D_v . Similarly, D_v is fed into the network, and then the output classification results are collected and compared with the ground truth. However, the results are only used to validate whether a training process goes well and never used to update neuron weights.

3 RELATED WORK

3.1 CNN Visualization

CNNs have recently received considerable attention in the field of visualization [34]. Existing approaches can be classified into two categories: feature-oriented and evolution-oriented.

Feature-oriented approaches aim to visualize and interpret how a specific CNN behaves on the input images to disclose what features it has learned. Most of the existing studies belong to this category. Some studies [41, 42] modify part of the input and measure the resulting variation in the output or intermediate hidden layers of a CNN. By visualizing the resulting variations (e.g., using a heatmap), users can identify which parts of the input image contribute the most to the classification results. By contrast, other studies [11, 30, 36, 41] attempt to synthesize an image that is most relevant to the activation (i.e., the output of a layer after an activation function) of interest to help experts determine which features of a specified image are important for the relevant neurons. For example, Mahendran and Vedaldi [30] reconstruct the input images that can either activate the neurons of interest or produce the same activations as another input image. In addition, some methods focus on retrieving a set of images that can maximally activate a specific neuron [12, 13]. In this manner, users can discover which types of features or images are captured by a specific neuron. Built on this method, Liu et al. [27] develop a visual analytics system that integrates a set of visualizations to explore the features learned by neurons and reveal the relationships among them. In addition, some work [10, 21, 31, 32] utilizes a dimension reduction technique to project the high-dimension activation vectors of FC or intermediate hidden layers onto a 2D space to facilitate revealing the relationships among outputs.

In contrast to those studies that investigate how a specified network works, only a few studies concentrate on visualizing network training processes. One typical method is to pick several snapshots of a CNN over the training process and then leverage feature-oriented approaches to compare how a CNN behaves differently on a given set of inputs at various iterations [4, 10, 41, 43]. For example, Zeiler and Fergus [41] use a deconvnet approach to visualize a series of synthesized images that are most relevant to one activation of interest at a few manually picked snapshots and then observe the differences between them. Zeng et al. [43] present a matrix visualization to show the weight differences of filters of one layer as well as this layer's input and output in two model snapshots. Their system also supports side-by-side comparison on the learned features of neurons (the computation method is similar to other work [13]) in different model snapshots. One

limitation of these methods is that, when there are myriad filters, images, and iterations, it is challenging to select the proper ones to observe and compare. By contrast, we attempt to reveal the rich dynamics of the network parameters and the classification results at a larger scale to help experts find the notable filters, images, and iterations. In this point, Alsallakh et al. [4] analyze the same data facets (i.e., input images, network parameters, and classification results). However, their work focuses more on identifying the hierarchical similarity structures between classes and still belongs to the category of multiple snapshot comparison, thereby suffering from the same limitation.

To analyze the evolution of CNN parameters, Eliana [29] treats the parameters of the entire network at one iteration as a high-dimension vector, uses PCA to map the vectors at all iterations onto a 3D space, and creates trajectories for these points. However, this visualization is too abstract to extract useful insights for understanding and debugging the training process. To analyze the classification results, Rauber et al. [32] create a 2D trails graph to present an overview of how the CNN classification results evolve by leveraging the projection techniques. However, this method suffers from scalability and visual clutter problems when applied to large-scale datasets (e.g., ImageNet). In addition, this method only provides a very high-level overview and cannot answer those questions that involve individual classes or images. The work most similar to ours is that by Liu et al. [26]; however, that work mainly targets deep generative models and would have serious scalability issue if applied in industry-level CNN training. In addition, compared with that work, we specifically provide a series of hierarchical methods that are tailored for CNNs to visualize the training dynamics, including classification results and network parameters. Furthermore, we design a novel correlation matrix and 2.5D cube-style visualization to help experts examine the complex relationships existing among network parameters, classification results, and iterations.

3.2 Multiple Time Series Visualization

Time series data have been extensively studied due to their ubiquity. Numerous approaches [2, 3, 5] to time series visualization have been proposed. Our system also falls into this field, since training logs are essentially a type of time-based information. Specifically, our system is most related to existing work that visualizes multiple time series.

To visualize multiple time series, one method is to place multiple charts (e.g., line charts) in the same graph space to produce overlapping curves. However, this method reduces the legibility of individual time-series [20]. In contrast to overlaying multiple series, one popular alternative is to use small multiples [39] that split the space into individual graphs, each showing one time series. Using small multiples also allows for an effective comparison across charts. Several factors may also affect the performance of small multiples [19, 20], such as the types of charts used (e.g., line charts and horizon graphs), the number of series, and the vertical space allocated to each series. The improper use of time series charts, the increased series, and the small vertical space of each series will result in a serious visual clutter problem.

In this work, the evolution of the image classification results of each class and the weight parameters of each layer can be viewed as two types of multiple time series data. Given the large number of classes and layers in a practical CNN training, we identify several space-efficient charts that can characterize training dynamics. In addition, we propose a similarity-based layout and a hierarchical exploration method to support the exploration of relationships among multiple time series to address the visual clutter problem. We also present a novel cube-based visualization targeting the exploration of complex relationships among various types of heterogeneous time series data (e.g., image classification results, neuron weights, and training iterations).

4 REQUIREMENT ANALYSIS

DeepTracker was developed in approximately nine months, during which time we collaborated closely with three experts (denoted by E_a , E_b , and E_c) who have considerable experience in CNNs. We held regular discussions with these experts once a week.

From our regular discussions, we learned that the training process should be carefully monitored. The experts have to tune and fix a number of *hyper-parameters* (learning rate, batch size, layer number, filter number per layer, weight decay, momentum, etc.) before starting a training trial. These hyper-parameters, which strongly influence how a CNN is trained, are usually selected based on experiences learned from previous trials. During a training, there are several useful quantities that our experts wanted to monitor, such as loss function (the residual error between the prediction results and the ground truth), train/validation error rate (the percentage of mislabeled images), weight update ratio (the ratio of the update magnitudes to the value magnitudes), and weight/gradient/activation distributions per layer. Basically, both loss and error rate should decrease over time: The consistent increase or a violent fluctuation of loss on D_t may indicate a problem; a big gap between the error rates of D_t and D_v may suggest that the model is over-fitting, while the absence of any gap may indicate that the model has a limited learning capability; the update ratio² is expected to be around $1e-3$; the weight distributions per layer in the beginning should overall follow Gaussian distributions with different standard deviation settings³ and may become diverse after training; and exploding or vanishing gradient values are a bad sign for the training.

These rules of thumb are all based on the analysis of high-level statistical information. However, our experts still very much wanted to examine more details underlying the statistics so that they could gain more knowledge and suit the remedy to the case when problems occur. For example, experience tells our expert that it is better to continue training models (e.g., ResNet [17]) with the same learning rate rather than turn the rate down immediately when the overall error rate stops decreasing. Our experts were very curious about what happens to the model and its performance on each class of images during this period. Also, we may often see a fluctuation of loss or an occasional explosion of gradient values: What brings about this? Are there any layers or filters that behave abnormally? These details have never been uncovered before. Thus, our experts wanted a tool to enable them to explore the hidden dynamics behind a CNN training. After several rounds of structured interviews with our experts, we finally summarized and compiled the following list of requirements:

- R.1 **Exploring multiple facets of neuron weight information.** A single iteration may have millions of weight updates, but individual values are generally meaningless to our experts. Instead, they are more interested in the statistical information of these weights at different levels of detail (e.g., CONV layer level and filter level). All three experts emphasized the importance of intuitively and effectively examining general types of statistics, such as sum and variance, between these levels. In addition, E_a and E_b strongly desired seeing the weight change degree for filters over iterations, to identify which filters change dramatically at which iteration or how a filter changes across iterations.
- R.2 **Comparing multiple layers.** All three experts like to compare layer-level statistical information. For example, they wanted to know whether a specified measure of different

²In most cases, if the update ratio is lower than $1e-3$, the learning rate might be too low; if it is higher than $1e-3$, the learning rate is probably too high.

³Xavier initialization [14] is applied in our model. Basically, the deeper (close to loss layer) the layer is, the smaller the sd.

layers shows a similar trend or distribution (e.g., whether the sum is positive or negative). Accordingly, our visualization should help these experts perform such comparisons.

- R.3 Tracking the classification results of validation classes.** Validation is a critical step in the training process that “test drives” the trained CNN and tracks its performance change across iterations [6]. Previous tools only measure the global validation loss/error, thereby concealing the rich dynamics of how the image labels of each class change over time. When the error rates do not reduce as expected, our experts find that such highly aggregated information is useless and cannot help them understand why or when training runs into a bottleneck. Therefore, our experts wanted to know how the images of different classes behave differently and identify those classes that are easy or difficult to train.
- R.4 Detecting important iterations.** One complete training process usually contains millions of iterations, but it is obvious that not all iterations are equally important. For example, some image classes may suddenly change their error rates after certain weight updates. Our experts were interested in these patterns, which may help reveal in-depth relationships between filters and image features. However, the overall error rate trend does not help much since it generally decreases slowly and steadily. Thus, our experts hoped that our system could help them identify abnormal iterations and corresponding classes.
- R.5 Examining individual validation classes.** Our initial prototype shows that different classes clearly have different error rate evolution patterns. Thus, experts E_b and E_c were curious about those classes with very poor or excellent performance and wanted to further explore the image label information for these classes. For example, they wanted to see whether and why some images are always misclassified.
- R.6 Enabling correlation exploration.** Apart from analyzing the weight and validation data separately, our experts were also curious about their relational patterns. They were specifically interested in uncovering the relationships between the layers or filters and the validation images, such as how changes in network parameters respond to the image labeling results for each class. By connecting these two pieces of information together, they hoped to gain fundamental insights into the behaviors of CNNs and improve training results.

5 SYSTEM OVERVIEW

DeepTracker is a web-based application developed under the full-stack framework, MEAN.ts (i.e., MongoDB, Express, AngularJs, Node, and Typescript). The back-end part of our application is deployed in a server with 3.10GHz Intel Xeon E5-2687W CPU and 32GB memory.

The architecture of our system (Figure 2) begins with the data processing part, where the entire training log is hierarchically organized and several application-specific indexes are built to support real-time interactions. On top of this efficient data storage, we build three coordinated views—Validation View, Layer View, and Correlation View—to support an interactive analysis from different levels and perspectives. The Validation View aims at providing a visual summary of CNN performance on different validation classes. By combining our anomaly detection algorithm and small multiples, our experts could easily identify different image class behavior patterns and critical iterations for each image class (**R3**, **R4**). Our experts could also drill down to the class of interest to explore further image label information (**R5**). The Layer View aligns the weight information with the CNN structure to help our experts explore various statistical information in the network hierarchy. They can further drill up or down in the network hierarchy to compare these measures at different levels of detail (**R1**, **R2**). The Correlation View presents a novel grid-based visualization method to provide an overview of the correlation between the image classification

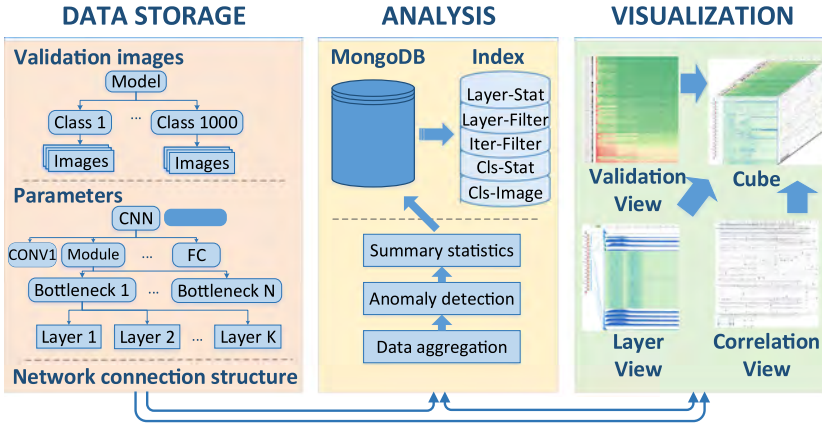


Fig. 2. Three components of DeepTracker. The raw data are preprocessed into a hierarchical structure and then stored into five application-specific data indexes to enable real-time interactions. On top of the efficient data storage, several visualizations are combined to help experts with analysis tasks from different levels and perspectives.

results and the neuron weights of each filter (**R6**). The three views compose a cube, with which our experts could simultaneously explore the changes of class-level performances, the variations of filter-level weights, and the correlations between them.

6 DATA ACQUISITION AND CONSTRUCTION

The primary motivation of this work is to monitor industry-level CNN training processes. Therefore, we conducted our experiments with ResNet-50 [17] and ImageNet Dataset [33]. ResNet-50, containing 50 weighted layers (i.e., CONV and FC layers), is among the most popular CNNs that have been recently used in practice. ImageNet 2012 is also among the largest and most challenging publicly available datasets. The dataset includes 1,000 classes of images, with each class containing 1,300 training images and 50 validation images. Training such a model needs around 120 epoches (nearly 1.2 millions iterations when batch size is 128) to achieve convergence. Simply dumping all the information for every iteration can easily have the size of dumped data exceed several petabytes and take about 4 weeks. Throughout our discussion, we agreed that 1,600 is a reasonable interval to capture meaningful changes (about 7 times per epoch). This reduces the log to a manageable size (about 1TB).

For each dump, we recorded two pieces of information: neuron weights/gradients of the CONV layer and FC layer and image classification results. The parameters on BN layers were not recorded as they can be completely recovered given the weights on CONV and FC layers and always need to be updated when applied in a new dataset. In addition, we did not record the activations of each layer/filter for every validation image as doing so is technically impracticable considering the extremely large models and datasets and the limited disk storage. Furthermore, activation evolution visualization is beyond the research scope of this article. Section 11 discusses activation visualization as a perfect complementary technique to our work.

We organized the weight/gradients information according to the natural hierarchical structure of ResNet-50. It consists of four *CONV modules* (plus the first CONV layer and the final FC layer; there are 50 layers in total). Each module contains several *bottleneck blocks* [17] that comprise three to four basic CONV layers (data storage in Figure 2). Thus, we grouped all neuron weights to align with such hierarchy. In a similar manner, we organized the classification results hierarchically

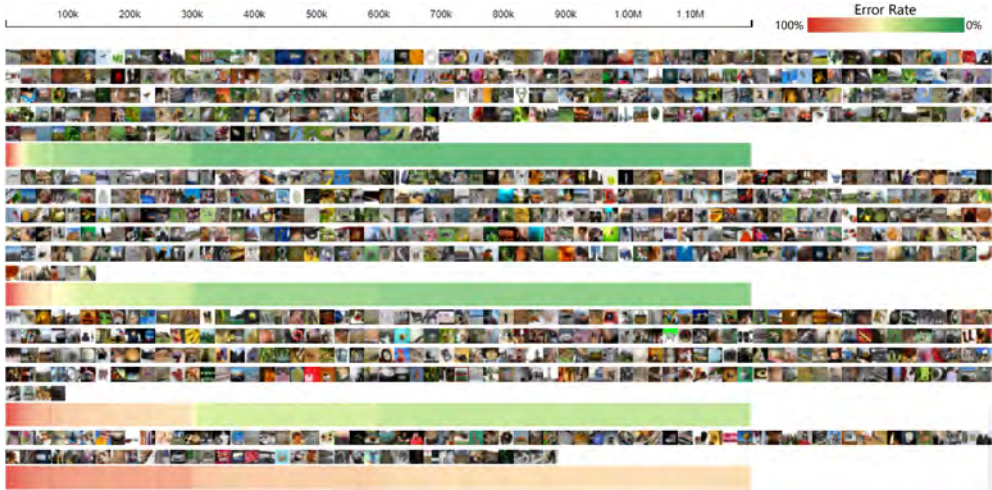


Fig. 3. From top (a) to bottom (b), image classes are increasingly difficult to train.

from individual level, to class level, to model level. We stored all data in MongoDB⁴. In particular, we precomputed all relevant aggregation values, such as weight means and error rates, for each filter, layer, image, and class. Nevertheless, the distilled data are still too large to load into memories (dozens of gigabytes per training). Therefore, we analyzed the frequent needs of our experts and built several indexes to enable real-time interactions, including the Layer-Stat index I_{ls} , Layer-Filter index I_{lf} , Iter-Filter index I_{if} , Cls-Stat index I_{cs} , and Cls-Image index I_{ci} .

- I_{ls} retrieves the statistical values (e.g., mean and sd) at every iteration for any given layer;
- I_{lf} lists all the filter-level information (e.g., changing degree of each filter) at every iteration for any given layer;
- I_{if} searches the top changing filters from all layers at any given iteration;
- I_{cs} extracts class-level information (e.g., class performances, the different types of abnormal images Section 7.1.2) over all iterations for any given class;
- I_{ci} fetches the meta-data of images for any given classes.

7 VISUALIZATION

In this section, we describe our three coordinate views, the Validation View, the Layer View, and the Correlation View, that help experts accomplish the aforementioned analytical tasks.

7.1 Validation View

Several of our experts (**R3**, **R4**, **R5**) needed to examine how the evolving CNN acts differently on the validation images of each class rather than how the overall validation error rate differs over training. Thus, we designed the Validation View (Figures 3 and 6) to present all image classes in D_v .

7.1.1 Visual Encoding. By default, the view starts with a visualization of **cluster-level** performance (**R3**). The classes with similar evolving trends form a cluster, and then their error rates at every iteration are averaged. The averaged error rates are then depicted as a colored strip, where the x-axis encodes the iterations and the error rates are encoded by colors (Figure 3). We choose k-means as the clustering algorithm, and k can be adjusted according to demands (Figure 3 shows

⁴MongoDB is a free and open-source cross-platform document-oriented (NoSql) database. www.mongodb.com.

the case when $k = 4$). Our experts could open up one cluster to further examine the performance at **class-level (R3)**. The design is based on the following considerations. First, our experts were more interested in the overall classes than in individual iterations. Thus, the small multiples technique (juxtaposed techniques) was chosen for its superior performance in high-level comparison tasks (e.g., global trends for every series) [20]. Second, we cannot present all the classes (1,000 in ImageNet) at one time; we have to consider a hierarchical and highly space-efficient visualization. However, many traditional charts, such as line charts and horizon graphs, require a larger vertical space [19] than do 1D heatmaps. Compared with traditional charts, heatmaps are also easier to place for side-by-side comparison of their symmetrical space (i.e., no irregular white spaces). As a result, all our experts preferred the heatmap-based small multiples.

Image-level performances, R5. The class-level color strips can be further unfolded to explore the image-level evolution patterns. Unfolding a heatmap reveals a pixel chart (Figure 6(d)) with each row (1px height) representing an image and each column (1px width) representing a dumped iteration (consistent with the class heatmap). We use red and green to indicate the incorrect and correct classifications, respectively. Meanwhile, our experts could zoom/pan the pixel chart for a closer inspection. Clicking on a row shows the original corresponding image.

Anomaly iterations, R4. Because our experts were concerned about those iterations with abnormal behaviors, we particularly propose an algorithm to detect these anomaly iterations (refer to Section 7.1.2). Our experts could choose to only show those classes with anomaly iterations (Figure 6). At this point, for each class-level color strip, we use triangular glyphs to highlight these detected anomaly iterations. The upside-down triangles (∇) and upright triangles (Δ) indicate those anomaly iterations that are detected by the left-rule and right-rule, respectively. The widths of triangles encode the anomaly scores. Users can set a threshold value to filter those triangles with low anomaly scores.

7.1.2 Anomaly Detection. In our scenario, the classification results for an image can be represented by a 0/1 sequence ($[a_1, \dots, a_n]$), where each element represents a correct or incorrect result at the corresponding validation iteration. Our experts were curious about those iterations when a significant amount of 1/0 flips (i.e., 0 to 1 or 1 to 0) occur for a class. In general, this problem can be modeled and solved using Markovian-based anomaly detection algorithms [1]. Despite the popularity of using Markovian methods to detect outliers in discrete sequence, we decided to employ rule-based models [1] for two reasons. First, Markovian methods are a black box, and the resulting outlier values are sometimes difficult to comprehend. Second, our experts explicitly described two types of iterations that they are very interested in; namely, those iterations when many images with values that remain stable for many previous iterations suddenly flip (denoted by the *left-rule*) and those iterations when many images flip and keep their values stable after many iterations (denoted by the *right-rule*). Fortunately, these anomalies can be easily modeled using rules. The rule-based models primarily estimate the value $P(a_i | a_{i-k}, \dots, a_{i-1})$, which can be expressed in the following rule form: $a_{i-k}, \dots, a_{i-1} \Rightarrow a_i$. In our scenario, if an image has the same value (either 0 or 1) in the previous consecutive k iterations ($i - k, \dots, i - 1$), then its value must be the same at iteration i (the left-rule). Otherwise, iteration i is considered an outlier for the specified image. Based on these considerations, we developed an application-specific algorithm to detect anomaly iterations in the validation history. The algorithm includes the following steps:

- (1) **Rule-Judgment:** The algorithm computes a vector $[l_{i1}, \dots, l_{in}]$ for every image i , where $l_{ij} = 1$ if the left-rule is satisfied, otherwise, $l_{ij} = 0$;
- (2) **Aggregation:** For each class that contains m images, the algorithm aggregates all the computed vectors for each image into one $[L_1, \dots, L_n]$, where $L_j = \sum_{i=1}^m l_{ij}$, denoting the left anomaly score at iteration j for this class.

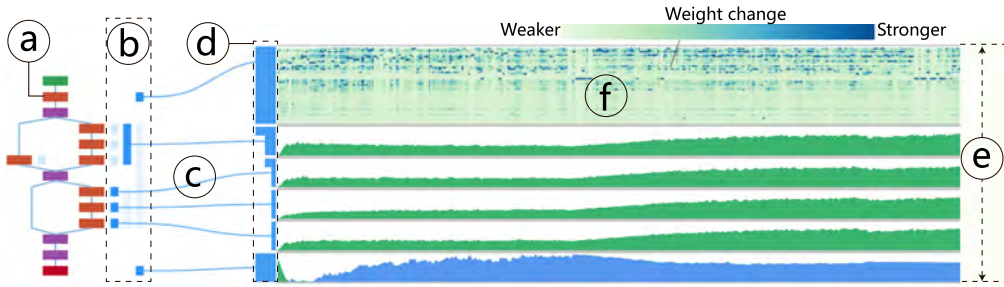


Fig. 4. Visual encodings in the Layer View: (a) a CONV layer, (b,d) hierarchical bars, (c) links between the CNN structure and (e) the hierarchical small multiple charts, (f) a pixel chart for one layer.

The approach is a window-based method, and our experts could adjust the window size k to control the sensitivity of the anomalies. In a similar manner, we detected the anomalies from the opposite direction for the right-rule.

7.2 Layer View

The Layer View focuses on weight-related tasks (**R1**, **R2**). The view consists of two connected parts; namely, the CNN structure and the hierarchical small multiples (Figure 4), so that our experts could hierarchically explore and compare various types of statistic in the context of the network structure.

CNN structure. Our experts hoped that our tool could help them explore the statistical information of each layer and also know their relative positions in the entire network (**R1**). Thus, we adopt Netscope⁵, a popular neural network visualizer, in our system. The green rectangle is the data input layer, the red rectangles are the CONV layers, and the purple rectangles represent the pooling layers. The links between these rectangles show the network structure. We further add blue level bars (Figure 4b) to encode latent hierarchy (from CONV modules, bottlenecks to basic CONV layers, Section 6). The right-most level bars represent CONV modules (Section 5), which are recursively divided into smaller level bars until reaching elementary CONV layers (i.e., red rectangles).

Hierarchical small multiples. To assist our experts in exploring and comparing layers of a deep CNN (**R1**, **R2**), a space-efficient visualization technique is necessary. Thus, we leverage hierarchical small multiples to show layers of interest (Figure 4(e)). By default, users are presented with information about CONV modules and then can drill down to see more information about low-level CONV layers with interactions with the network graph (i.e., click on the corresponding level bars). The width of outcropping rectangles (Figure 4(d)) encodes the aggregation level of current layer charts. For example, the top second-layer chart in Figure 4(e) shows the bottleneck-level aggregation information, and the following three layer charts show the basic CONV layer level information. In addition, the links (Figure 4(c)) mark the real positions of the layer charts in the network structure.

The small multiples support multiple types of charts, including line charts, horizon graphs [18], and box plots, to emphasize the different aspects of the statistical data. Our experts used box plots to see the rough distribution of statistical values and basic line charts to examine individual values. In addition, they preferred to use a horizon graph when performing tasks in regard to trend tracking and comparison (**R2**) because of its effectiveness in visualizing divergent weight values [20].

⁵Netscope is a web-based tool for visualizing neural network architectures. <http://ethereon.github.io/netscope/>.

ALGORITHM 1: Minimum Set Partition

Input: Target set S_{target} .
Output: Minimum partition for S_{target} .

```

1  $S_{result} = \emptyset$ ;
2 for each target set  $s_t$  in  $S_{target}$  do
3    $S_{new} = \emptyset$ ;
4   for each mini set  $s_r$  in  $S_{result}$  do
5      $intersection = s_t \cap s_r$ ;
6      $S_{new} = S_{new} \cup intersection \cup (s_r - s_t)$ ;
7      $s_t = s_t - s_r$ ;
8   end
9   if  $s_t$  is not empty then
10     $S_{new} \cup s_t$ ;
11  end
12  remove all empty set in  $S_{new}$ ;
13   $S_{result} = S_{new}$ ;
14 end
15 return  $S_{result}$ 

```

Similar to unfolding the class heatmap to a pixel chart, they were also able to open the layers of interest as a pixel chart (Figure 4(f)) that presents the filter-level information (R1). Each row (1px height) in the pixel chart represents one filter, and each column (1px width) indicates one iteration. We use sequential colors to encode pixel value (e.g., the cosine similarity between two subsequent dumped iterations).

7.3 Correlation View

This view helped our experts establish connections between filters and images. In particular, they wanted to understand further how changes in network parameters are related to class performance (R6). For example, several anomaly iterations may be detected for a single class. For each detected anomaly iteration, we can identify a set of *anomaly filters* (i.e., the top k filters with largest changes at that iteration). Since different classes may share anomaly iterations, and different anomaly iterations may share anomaly filters, are there any filters that are commonly seen in these iterations? Do any of the anomaly classes or filters strongly co-occur? We designed the Correlation View to answer these questions.

Filter set partition. We first introduce the *mini-set* concept to organize anomaly filters that are shared by multiple anomaly iterations and different classes. For each class $C_i \in \{C_i | 1 \leq i \leq n\}$, we denote its anomaly iterations by $T_i = \{t_{i,k} | 1 \leq k \leq n_i\}$. Thus, all anomaly iterations are $\cup_{1 \leq i \leq n} T_i$, denoted by T . For each anomaly iteration $t \in T$, we denote its anomaly filters at CNN layer $L_j \in \{L_j | 1 \leq j \leq m\}$ by $s_{j,t}$. Thus, for each layer L_j , we can collect all anomaly filter sets $\{s_{j,t} | t \in T\}$ (denoted by S_j) and all anomaly filters $\cup_{t \in T} s_{j,t}$ (denoted by s_j^*). Thus, mini-set aims to find a minimum number of partitions of s_j (denoted by s_j^*) that each $s_{j,t}$ can be assembled from some elements (i.e., mini-sets) in s_j^* . We specifically propose a Set Partition Algorithm (Algorithm 1) to find s_j^* . The algorithm accepts a target set as input (i.e., S_j). S_{result} is initially empty, and a new anomaly filter set is used at each time to partition the mini-sets contained in S_{result} (cf. Lines 4 to 7). If the new anomaly filter set is not empty after partitioning, then it is added as a new mini-set (cf. Line 8). Finally, the partitions contained in S_{result} will be returned.

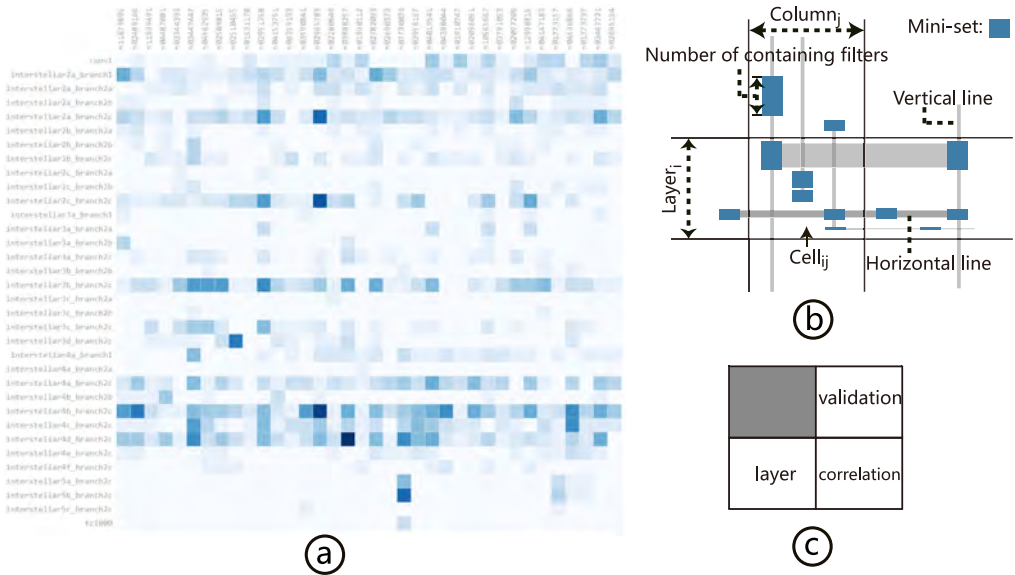


Fig. 5. (a) The abstract version of Correlation View, where rows and columns represent layers and image classes, respectively. A sequential color scheme is used to encode the number of anomaly filters. (b) The complex version of the Correlation View, where the detailed information of individual anomaly filters is shown. (c) A layout solution for coordinated analysis without using skewed axes.

Visual encoding. To intuitively represent these relationships, we introduce a grid-style visualization (Figure 5), where rows and columns represent layers and image classes, respectively. The number of rows and columns is equal to the number of layers with anomaly filters and classes with anomaly iterations, respectively. We start from an abstract version. In this version (Figure 5(a)), for Cell_{*i,j*}, a sequential color scheme is used to encode the number of anomaly filters ($\cup_{t \in T_j} s_{i,t}$). The darker the color, the more anomaly filters appear in layer L_i that are related to class C_j . From this visualization, we can easily observe the correlations between layers and classes, although it also hides much detailed information. We cannot answer questions like whether the filters in Cell_{*i,j*} are the same with Cell_{*i,k*} (this kind of information shows how many classes this filter can impact and helps examine the relationships among classes), or whether there are filters in Cell_{*i,j*} appearing in more than one anomaly iteration (this shows the importance of these filters for class C_j).

To solve these problems, we provide an advanced version (Figure 5(b)). For Cell_{*i,j*}, the width and height encode the number of anomaly iterations and the number of anomaly filters of the corresponding class and layer (i.e., $|T_j|$ and $|s_i|$), respectively. Based on these numbers, the columns and rows are further divided with vertical/horizontal lines. For a class (e.g., Column_{*j*}), $|T_j|$ vertical lines are drawn to represent all related anomaly iterations (i.e., $[t_{j,1}, t_{j,2}, \dots, t_{j,n_j}]$). For each row of layer L_i , there are $|s_i^*|$ horizontal lines representing all mini-sets in that layer. The intersections between these horizontal and vertical lines are highlighted with blue rectangles if the corresponding mini-set is part of the anomaly filters of the corresponding anomaly iteration. The height of the rectangles represents the number of filters of the corresponding mini-set. Obviously, the introduction of mini-sets dramatically cuts down the number of horizontal lines and blue rectangles; otherwise, each anomaly filter requires one horizontal line and one rectangle, which may cause serious visual clutter. In fact, mini-sets can be viewed as a partial aggregation version instead of

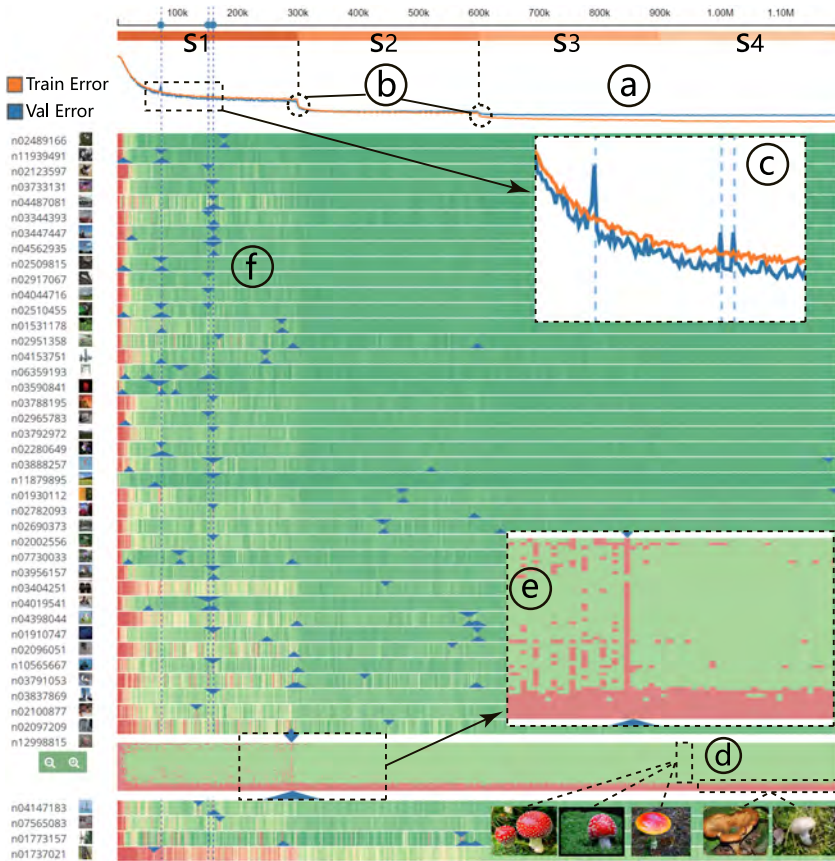


Fig. 6. Overview of validation classes. (a) Two curves show the overall training/validation error rate. (b) Two turning points align well with the boundary of stage s2. (c) Three peaks appear in stage s1 and align well with (f) the detected anomaly iterations. (d) Two types of mushroom images have different behaviors in the class. (e) Most images in the class flip at the anomaly iteration.

representing all the anomaly filters as horizontal lines and rectangles. Users can set the minimum appearing number of mini-sets to filter out those sets with lower importance.

7.4 Cube Visualization

The log data contain three main aspects of information: iterations, validation information, and weight information. The three views are designed to show all possible 2-combinations of these three types of information, respectively. Although these views can be used individually, they need to be combined together to form a complete picture. Thus, we propose a novel and intuitive visualization technique that naturally and seamlessly stitches the three views together based on their shared axis (inspired from Binx [8]) into a “cube” shape (Figure 9). When users find or highlight a pattern of interest, they can easily track the pattern over the edges to find the related information in the other two views.

The use of skewed axes may bring about a possible perspective distortion problem. Nevertheless, the advantages of the cube-style design far outweigh its disadvantages. Given the limited pixels in a computer screen and each view requiring a large display space, our experts all agreed that

the cube-style design is the most space-efficient and intuitive manner to show all the information. Furthermore, such a design prevents users from switching from multiple views, thus reducing the memory load. This allows users to conduct correlation analysis more effectively. We also provide a compromise solution to handle the distortion problem; we lay out the three views as shown in Figure 5(c) so users can first examine the layer view (horizontally) or validation view (vertically) together with the correlation view and then switch to cube mode to explore the three views together.

Note the different settings for several views in the cube. For the layer view (front), only the layers with anomaly filters are activated (see the activated blue bars in the front view of Figure 9), and the weight variation of each anomaly filter is represented as a horizontal color strip. For the validation view (top), only classes with anomaly iterations are preserved. The following lists several common exploration pipelines:

- P1: From the layer view (front), we can quickly check the distribution of activated layers in the overall network and pick some anomaly filters of interests. Then, by tracking along the horizon axis to the correlation view (right), we can examine which classes these filters impact and how important these filters are to the classes. Finally, we can observe the evolving patterns of these classes and the corresponding anomaly iterations in validation view (top).
- P2: From the validation view (top), we can first mark several anomaly iterations of some classes. Then, we can check the corresponding columns in the correlation view, finding those rows that contain anomaly filters and exploring the importance of these filters to these classes and how these filters impact the other classes. Finally, by highlighting these corresponding rows, we can observe them in the layer view to see how these filters behave around the chosen anomaly iterations.
- P3: From the correlation view (right), we can search the horizontal lines across many rectangles (showing filters that impact many classes at the same time) or the rectangles that appear more than once in the same cell (these filters are judged as repeated anomalies for a class and may have great impact on that class). With these selected horizontal lines or rectangles, we can simultaneously track their corresponding weight variation information in the layer view (front) and class performance information in the validation view (top).

8 USE EXAMPLES

We derived these examples with the assistance of our collaborating experts, who were familiar with our designs and data. The following results are derived from an experiment using an 8 times larger batch size and learning rate setting than the basic setting introduced in Section 6.

8.1 Exploring Validation Results

The first scenario demonstrates how our experts used DeepTracker to explore image classification results (R3, R4, and R5).

Performance evolution patterns. Figure 6(a) shows a typical visualization of training/validation errors that may appear on any popular training platform. The timeline at the top shows a total of 1.2 million iterations. Beneath the timeline, four line segments represent four stages (s_1 , s_2 , s_3 , and s_4 in Figure 6) in the training process, where the later stage has one-tenth of the *learning rate* of the previous stage. We can observe that two sudden drops in the curves match well with the boundaries of the training stages (Figure 6(b)). However, this is a well-known pattern to our experts. On the other hand, although the overall error rate continues to decrease, the class-level error rates show a more complicated story that is new to our experts. By quickly scanning the small multiples at the cluster level, they identified that there are generally four types

of evolving class patterns (Figure 3). From top to bottom, the four types are increasingly difficult to train. For example, for the type at the top, these classes are recognized correctly after a few iterations. By contrast, the classes at the bottom always have high error rates throughout the entire training process, which means that the resulting network fails to recognize the related images. From this, our experts learned that the model spent most of its time improving its performance on the classes of middle-level classification, since it has already performed well on the easily trained classes at a very early stage and is always performing miserably on the hard-trained classes over the entire training. From these patterns, our experts considered it promising to accelerate the training process and improve the overall performance by treating classes differently during the training process. That is, stop feeding the easy-trained classes at an appropriately early stage, put more effort into training the classes of middle difficulty for classification, and figure out why some classes always have extremely high error rates. One similar attempt has been made in a recent work [25].

Anomaly iterations. Our experts were curious about the three sudden peaks in stage s_1 , and then marked these three iterations with dotted lines (Figure 6(c)), which look like anomaly iterations (R4). However, the colors in the small multiples do not have clear patterns related to these iterations. Our experts turned on anomaly detection and immediately found that many triangles are aligned well with the dotted lines (Figure 6(f)), thereby confirming their suspicions. Our experts could click on the corresponding image icons to see the detailed images that contribute to the three peaks. In addition, there are more anomaly iterations in stage s_1 than in the later stages. This interesting pattern can be explained by the reduction of learning rate and the convergence of the model in the later stages. At the same time, it also implies that the learning rate in stage s_1 is slightly too high, leading to instability in the model (the case in Section 8.2 indicates the same finding for the discovery of potential “dead” filters).

Details in classes. To further examine what happens at the anomaly iterations for a class, our experts can further check the image-level information of the class (R5). For example, they are curious about the abnormally large anomaly iterations in the class of “mushroom” (Figure 6(d)) that are captured by both the left-rule and right-rule. They click and expand the color strip to see the pixel chart of images. First, they confirm that this iteration is indeed special for this class because nearly all images flip during that particular dumped interval (Figure 6(e)). They may further investigate to find the layers or filters that cause such flips based on the filter updates around that iteration. In particular, our experts commented that, after the iteration, it seemed that the CNN model jumped to a better local optimal for the class because the green color is more stable after the iteration. This may result from the reduction of the learning rate (from s_1 to s_2). This kind of pattern appears frequently in many classes during the whole training process, with many of them not occurring at the learning rate transition point. Our experts wondered if the model was continuously trying to jump from one local optimal to another better local for these classes to gradually reduce the overall error rate. This insight had never been obtained before because our experts initially thought that the error rate for one class should decrease steadily. In addition, our experts also found that, at the bottom of the pixel chart, several images are mislabeled during the entire training process, although the class is easy to train overall (Figure 6(d)). To understand why, they clicked on these images to find that the contents in the mislabeled images have a clear color pattern difference from the rest of the mushroom images. The correctly labeled mushrooms are all red, while the mislabeled ones are white or orange. This finding indicates that color is a critical feature that the CNN has learned to classify this class of images.

8.2 Exploring Weight-Relevant Information

This scenario shows how to discover patterns in neuron weights via the Layer View (R1, R2).

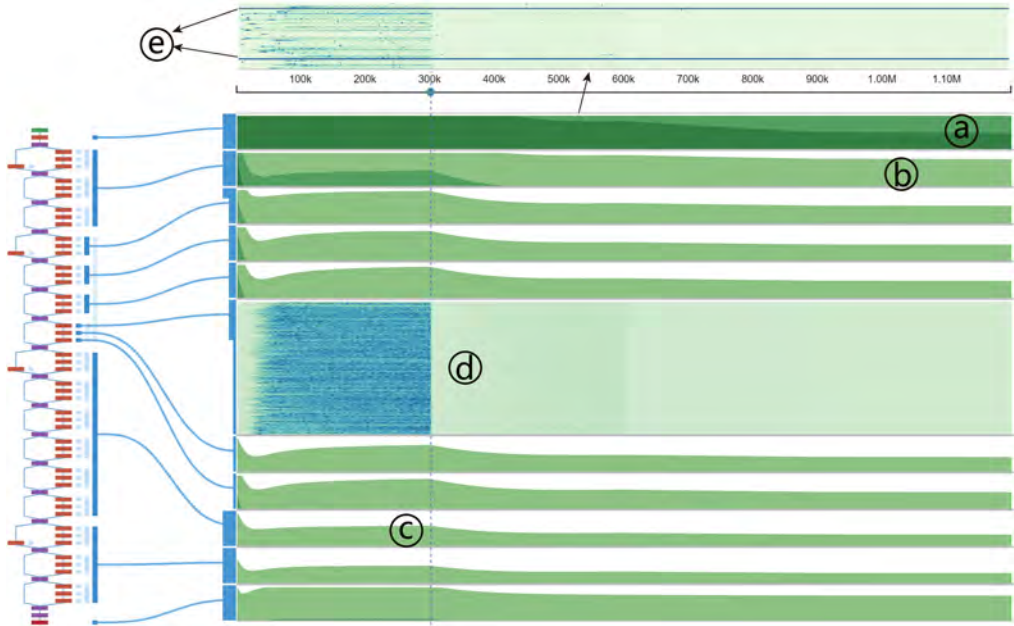


Fig. 7. (a, b, c) The sd values of each layer decrease slowly and have different scales in different CONV modules. (d) The weight changes in filters are large at the beginning. (e) One outlier filter is detected whose weights never change during the entire training process.

First, our experts chose to show the standard deviation (sd) of the weights at the layer-level using horizontal graphs (Figure 7). As they expected, all the trends show a similar pattern of slow decrease, indicating that the weights in the entire model are converging over iterations. In addition, our experts also found that deeper layers (closer to the loss layer) tend to have smaller sd values. In particular, by tuning the band number (finally to 3) of the horizon graphs, they found that the sd values of a CONV module are usually twice as large as those of the one below it (a, b, and c in Figure 7). Given that we apply Xavier initialization³ and, for ResNet-50, the input sizes of layers in a CONV module are twice as large as the ones in the layers of its previous CONV module, the observed result is not beyond our experts' expectations. This suggests that no problem exists on the initialization approach.

Analogously, our experts found that the weight means of each layer become negative quickly (from green to blue instantly, Figure 8(a)) except for the FC layer (Figure 8(b)). At first, the pattern looked strange to our experts, until they realized that it is reasonable to have more negative weights than positive ones since negative values are often used to filter out trivial features owing to the use of ReLU activations. The increase of negative weights suggests that the network is trained to extract useful information from the original image layer by layer and then finally retain the most relevant features. As for the FC layer, it plays a function in shaping the extracted useful features into feature vectors of given dimension for further classification. One strange phenomenon intrigued our experts: The FC layer weight means are always positive in repeated training of ResNet-50 (with different batch sizes and learning rates) on the ImageNet Dataset, whereas they become negative when training ResNet-164 on CIFAR Dataset [22]. This finding is worth further investigation.

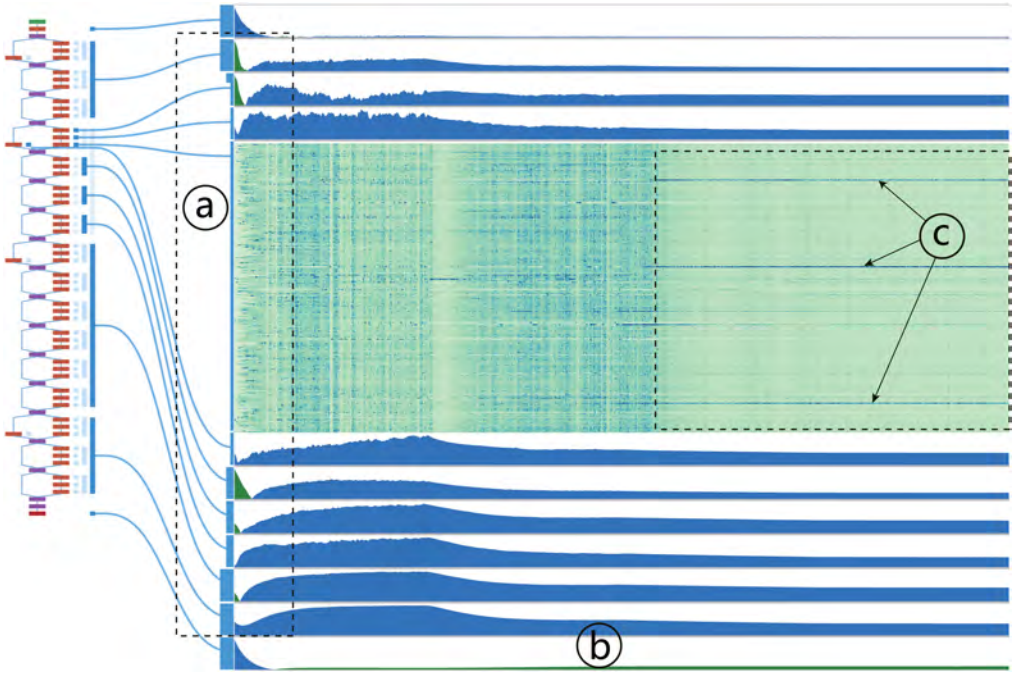


Fig. 8. (a, b) The means of weights in each CONV layer become negative quickly (from green to blue) except for the FC layer. (c) Three filters are always more actively changed than the other filters in the later part of training progress.

Apart from layer-level values, our experts also explored the filter-level information (**R1**). In our system, two different methods (i.e., filter-based or iteration-based) are used to normalize weight changes at the filter-level. For filter-based normalization, changes are grouped and normalized by filters, which aims to help experts see the change distribution over iterations for individual filters. Similarly, iteration-based normalization allows users to examine the distribution over filters for individual iterations. For example, Figure 7(d) visualizes the filter changes in one of the CONV layer belonging to the second CONV module using filter-based normalization. Our experts found that the changes are drastic in stage s_1 and become relatively small in the later stages because of the decrease in learning rate and the convergence of the model. However, they also identified two strange filters among 64 filters in the first CONV layer that have a constant deep blue color (Figure 7(e)). By further checking, they found that the weights of these two filters never change during the entire training process.

This was a surprise: Excluding programming bugs, the most likely reason should be due to the dying-ReLU problem; namely, these two filters are inactive for essentially all inputs, and no gradients flow backward through the neurons of the two filters. Our experts suspected that the dying-ReLU problem results from the high learning rate at the beginning of training. In fact, experts usually follow a rule of thumb to set the hyper-parameter learning rate: Multiply the learning rate by k if the batch size is multiplied by k . This rule is currently formally introduced in a recent work [15]. In this experiment, we use 32x batch size for the use of 32 GPUs to train the model with the corresponding 32x learning rate. However, dying-ReLU problem still occurs. This alerted our experts that the rule may not be accurate for extremely large batch sizes. The problem can be solved by carrying out a warmup strategy (i.e., using a lower learning rate at the start of

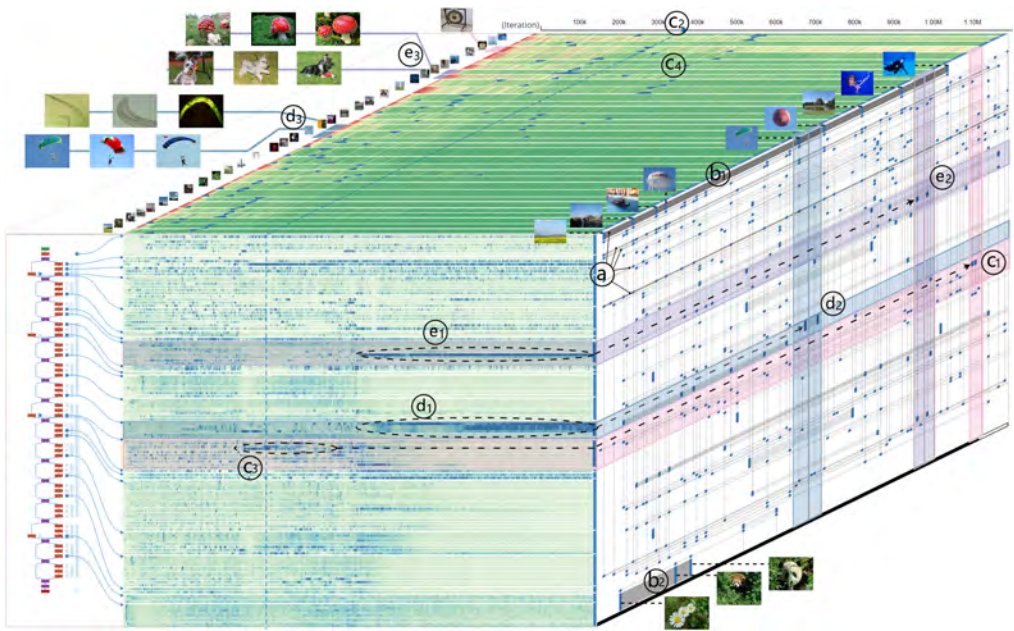


Fig. 9. A cube-style visualization that fuses three coordinated views together to reveal the rich dynamics in a CNN training process: (top) the Validation View shows the error rate changes of validation classes; (front) the Layer View shows the weight changes in CNN filters; (right) the Correlation View shows the potential relationships between filters and validation classes.

training [17]), which our experts had not done in previous trainings. One further interesting finding is that by inactivating these two “dead” filters (i.e., set their weights at 0 so that they are inactive for any inputs), our experts found the overall performance not affected at all, whereas if we inactivated other randomly selected filters in the first CONV layer of the model, the number of mislabeled images in D_v would increase by a few thousands. Thus, our experts finally modified the network configuration and eliminated these two filters so that the model runs faster while using less memory.

Figure 8(c) visualizes the weight changes in one middle layer using iteration-based normalization. Our experts found that a small number of filters were always more actively changed than the other filters (long deep blue lines in Figure 8(c)) in the later part of iterations. This pattern implies that the updates inside a layer may be highly divergent. In the later part of the training, where the learning rate is decreasing and the model is converging, only a couple of filters are still continually actively updated for every iteration. We tried to inactivate these identified filters, but the results show that overall performance is not affected. This was not beyond our experts’ expectations due to ResNet’s ensemble-like behavior [40] (several entire layers can be removed without impacting performance). In the end, our experts still could not fully explain the behavior of these continually updating filters. One possible reason could be that these special filters are not well-trained (do not converge) to extract some specific features, thus reacting violently at every iteration even in later stages of the training.

8.3 Exploring Filter-Image Correlations

In this scenario, we demonstrate how our experts used the Correlation View to explore correlations between images and filters.

Shallow-layer filters vs. deep-layer filters. At first, our experts chose to only show the top k (100 in this case) changing filters in the layer view. By checking the network structure visualization, they found that the activated shallow layers (the layers close to data input layer) are more numerous than the activated deep layers, and most activated layers are in the last basic CONV layers of bottlenecks for deep CONV modules. In addition, Figure 5(a) shows that deep CONV modules tend to contain more anomaly filters (especially for CONV modules 4). Our experts considered this kind of knowledge of great importance for network compression [16]. They then switched to the complicated version to examine more detailed correlation information. They filtered the rarely appearing mini-sets, finding that anomaly filters in shallow layers are generally shared by more anomaly classes (columns) and iterations (vertical lines in one column) than those in deep layers (Figure 9(a)). Our experts believed that this pattern may relate to the fact [40] that shallow-layer filters are more likely to capture basic visual features than deep ones, and therefore thereby the huge changes in these filters affect more classes of images (e.g., the long and opaque lines marked by b in Figure 9). By contrast, a deep filter tends to learn higher level features, thus only relating to specific classes of images.

To further explore the correlations, our experts selected two mini-sets (b1 and b2 in Figure 9), for comparison. Both horizontal lines of b1 and b2 are opaque and thick. By tracking them in Layer View and Validation View, our experts could see that b1 is in the first CONV layer and related to many classes. They opened these classes and discovered that many of their images have a common feature (i.e., a large background of blue sky or ocean; b1 in Figure 9). This discovery suggests that these filters in E1 may target the basic pattern to help identify images that contain large blue areas. By contrast, b2 is located at the fifth CONV module and related to only three classes. Interestingly, the images in the three classes also share a more concrete feature (i.e., objects in a bush; b2 in Figure 9). Thus, this case confirms that we are on the right track to reveal how model weight changes relate to classification result changes.

Important filters for a class. To find stronger correlations between filters and classes, our experts focused on anomaly filters that appear more than once in a cell for a specific class. For example, they found two appearances of the same mini-set (containing two anomaly filters) for the class of “gong” (c1 in Figure 9). Tracking horizontally (along with the pink-highlighted area), our experts found that the mini-set does not appear in other anomaly iterations, which also implies a strong correlation between filters in the mini-set and the class. Our experts clicked on these two rectangular glyphs to highlight the corresponding iterations on the timeline (c2 in Figure 9) and the filter locations in the Layer View (c3 in Figure 9). It is clear that the gong class is not a well-trained class as it has a very large yellow area (indicating a relatively high error rate) in the Validation View. However, our experts also found a period in the middle when this class shows relatively good performance (c4 in Figure 9), which happens to contain the highlighted anomaly iterations. Meanwhile, the Layer View shows that the highlighted filters are also updated dramatically during the period of good performance (c3 in Figure 9). Considering these patterns, our experts speculated that filters in the mini-set have a strong impact on the classification of gong images. As expected, we conducted experiments to inactivate these two filters and found that overall performance and performance on “gong” class are not impacted (see the reason in the last paragraph in Section 8.2). Nevertheless, it provided our experts with a new way to investigate the functions of filters co-working together to classify one class of images. That is, increase the threshold to find as many anomaly filters as possible, find the mini-sets containing many filters to some classes from multiple layers, and then inactivate them all to validate corresponding impacts.

Abnormal anomaly filters. Our experts were also attracted by two mini-sets (d1 and e1 in Figure 9), because of their abnormal color patterns. The filters in these two mini-sets exhibit large persistent changes in the latter part of the training, which is very different from other anomaly

filters. Our experts further checked their correlated classes in the Correlation View (right). Interestingly, each abnormal mini-set only appears with two classes (d2 and e2 in Figure 9), and each pair of classes has very similar performance displayed in the Validation View (d3 and e3 in Figure 9). By checking the detailed images of these classes, our experts discovered some common patterns. For example, for mini-set e1, the corresponding classes are about mushrooms growing on grass and dogs playing on grass (e3 in Figure 9). For mini-set d1, the corresponding classes are related to curved shapes, such as parachutes and round textures (d3 in Figure 9). Although our experts were still unclear about why these two mini-sets exhibit such special behavior, they believed that these filters are likely to play important roles in identifying middle-level features such as grass and curved shapes. We also conducted further experiments to validate the impact of inactivating these filters, and the results were similar to the previous case (i.e., important filters for a class).

9 EXPERT FEEDBACK

Usability. DeepTracker was built with close collaboration with three domain experts who constantly underscored their requirements and provided suggestions during the implementation process. After several iterations of refinement, these experts were happy with the current version. They all praised our way of effectively exploring a extremely large-scale training log via a hierarchical manner. E_a and E_b mentioned that the well-designed validation and layer views were very intuitive and helped them greatly. For example, the layer view, by allowing users to effectively observe and compare layer-related information (e.g., weight/gradient distribution), can help them diagnose network structures. The detection of a dying-ReLU problem in the early stage of a training is useful for tuning hyper-parameters (e.g., learning rate). This kind of knowledge can also be leveraged to conduct model compression [16] to improve the model in respect to computing speed and memory cost. Although our experts still cannot figure out exactly why some filters are always more actively updated in the later training stages, they believe the insight that would be obtained from future investigations will be helpful in diagnosing and improving network structures. In addition, the divergent evolving patterns of classes and the numerous anomaly iterations found in validation view provide our experts with a new promising direction to train a better model. Both E_a and E_c were particularly fond of the cube-style visualization as a new perspective by which to observe the training of CNNs. They both found and explained many interesting patterns using the cube visualization. However, our experts also failed to explain some other patterns, notwithstanding several testing experiments. Nevertheless, they were still convinced that our system could help them identify potential subjects for further study.

Generality. During implementation, we were concerned about the generality of DeepTracker; that is, whether the design was biased to the specific requirements of these three experts. Therefore, to check how our system would be accepted by broader expert communities, we presented our system in a workshop involving about 20 experts in the machine learning and visualization fields. In addition, we also interviewed another group of 12 experts who worked on a large project about using CNNs to improve image search quality. We presented the latest version of DeepTracker to these experts, encouraged them to experiment with the system, and collected their feedback in the process. Exceeding our expectation, DeepTracker was well accepted by these experts. Although they proposed several new requirements, the experts shared many major interests with our three collaborators, such as tracking class-level performance, filter-level information, and the relationships between them. After introducing our system, they immediately understood the purposes of each view, and all appreciated this novel, intuitive, and expressive way to watch training processes. Although the demo was performed on our experimental datasets, these experts saw its potential in their project and immediately asked to collaborate with us so that they could plug in and analyze their own datasets.

Improvement. Apart from this positive feedback, the experts also made several interesting suggestions to further improve DeepTracker. For example, two experts suggested that our current system only differentiates correct or incorrect classifications for validation images (i.e., 1 and 0). However, the exact incorrect labels should also be presented because such information can help identify confusing or similar classes. One expert mentioned a strong interest in what happens before the anomaly iteration and suggested a dump of data for every iteration at that abnormal interval for fine-grained analysis. Another expert suggested that our system should be integrated with online dashboards, as visualizing the training log on the fly could allow early termination of training and save time if the visualization shows undesirable results.

10 DISCUSSION

DeepTracker is our first step to open the “black box” of CNN training. Although our consulted experts have high expectations for this tool, we all agree to start with two fundamental pieces of information: neuron weights and validation results. Considering our target users and the large scale of datasets, we try to avoid using sophisticated visual encodings to ensure a fluent exploration experience. Unsurprisingly, our bare-to-metal visualizations are preferred by the experts, and they use it to find many patterns easily, either expected or unexpected. However, we still have several limitations.

First and foremost, although our system can effectively help experts identify strange or interesting patterns, there is still a gap between finding patterns and accelerating CNN training. The experts still have to reason about and understand what these patterns mean or how to use them to accelerate model training in future. We think it is not a problem faced just by our system, but by all CNN visualizations in general. Like previous work, DeepTracker may only poke a hole in the box and reveal limited information. But we hope that, by poking enough holes, all these strange patterns will start to connect and make sense by themselves, thus providing a clear picture of CNNs.

Second, we have adopted many space-efficient visualizations and interaction techniques (e.g., hierarchy, filtering, and aggregation) to address the scalability issue. Our current design can well support showing dozens of layers and classes at the same time. The correlation view shares all the filter strategies with the other two views, and vice versa. Thus, our system can perform well in most cases. Nevertheless, the worst-case scenario still requires the display of hundreds or thousands of small multiples at the same time. A possible solution is to employ task-specific aggregation or filtering methods to show data of interest.

Third, we propose a rule-based anomaly detection method that requires experts to manually pick a reasonable window size k and set the threshold for filtering. The number and patterns of anomalies are sensitive to these settings. One potential solution to this problem is to develop an automatic method to enumerate all potential parameter settings and identify those that can detect a reasonable amount of significant anomalies and provide these settings to the experts as guidance.

Finally, we only conduct experiments on ResNet-50 [17], but our method can also be applied to other state-of-the-art deep CNN models, which often have similar hierarchical structures (e.g., “inception block” in google-inception-v4 [37]). In addition, the cube visualization is a general technique that can be used to explore multiple heterogeneous time series data and their complex correlations. However, to further generalize it, a strict user study has to be conducted to find the best manner of use, such as the axis skew degree and the minimum height/width for each row/column in the three faces.

11 CONCLUSION

We proposed a novel visual analytics solution to disclose the rich dynamics of CNN training processes. Knowing such information can help machine learning experts better understand, debug,

and optimize CNNs. We developed a comprehensive system that mainly comprises the validation, layer, and correlation views to facilitate an interactive exploration of the evolution of image classification results, network parameters, and the correlations between them. We conducted experiments by training a very deep CNN (ResNet-50) on ImageNet, one of the largest labeled image datasets that is commonly used in practice, to demonstrate the applicability of our system. The positive feedback from our collaborating experts and experts from an internal workshop validates the usefulness and effectiveness of our system.

Future studies may integrate some feature-oriented visualization techniques that typically require recording the activation information for input instances. Feature visualizations can provide insights on what features a filter in a given snapshot of a CNN has learned. Our system can track critical iterations to take snapshots of a CNN over the course of training and then use feature visualization techniques to analyze the learned features' evolving patterns to detect important filters. The other urgent need is to deploy the system in a real-time environment. To this end, we have to consider some new design and interaction requirements to fill the gap between finding patterns and accelerating CNN training.

ACKNOWLEDGMENTS

The authors would like to thank Kai Yan for providing support in editing the relevant media materials.

REFERENCES

- [1] Charu C. Aggarwal. 2013. *Outlier Analysis*. Springer.
- [2] Wolfgang Aigner, Silvia Miksch, Wolfgang Müller, Heidrun Schumann, and Christian Tominski. 2008. Visual methods for analyzing time-oriented data. *IEEE TVCG* 14, 1 (2008), 47–60.
- [3] Wolfgang Aigner, Silvia Miksch, Heidrun Schumann, and Christian Tominski. 2011. *Visualization of Time-Oriented Data*. Springer.
- [4] Bilal Alsallakh, Amin Jourabloo, Mao Ye, Xiaoming Liu, and Liu Ren. 2018. Do convolutional neural networks learn class hierarchy? *IEEE TVCG* 24, 1 (2018), 152–162.
- [5] Benjamin Bach, Pierre Dragicevic, Daniel Archambault, Christophe Hurter, and Sheelagh Cpendale. 2014. A review of temporal data visualizations based on space-time cube operations. In *Proceedings of the Eurographics Conference on Visualization*. The Eurographics Association, 23–41.
- [6] Yoshua Bengio. 2012. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller (Eds.). Springer, 437–478.
- [7] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE TPAMI* 35, 8 (2013), 1798–1828.
- [8] Lior Berry and Tamara Munzner. 2004. Binx: Dynamic exploration of time series datasets across aggregation levels. In *Proceedings of the IEEE Symposium on Information Visualization*. IEEE Computer Society, Los Alamitos, CA, USA, 5–6.
- [9] Léon Bottou. 1991. Stochastic gradient learning in neural networks. *Neuro-Nimes* 91, 8 (1991).
- [10] Sunghyo Chung, Cheonbok Park, Sangho Suh, Kyeongpil Kang, Jaegul Choo, and Bum Chul Kwon. 2016. Revacnn: Steering convolutional neural network via real-time visual analytics. In *Future of Interactive Learning Machines Workshop at Proceedings of Conference on Neural Information Processing Systems (NIPS'16)*.
- [11] Alexey Dosovitskiy and Thomas Brox. 2015. Inverting convolutional networks with convolutional networks. *arXiv preprint arXiv:1506.02753* (2015).
- [12] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2009. *Visualizing Higher-Layer Features of a Deep Network*. Technical Report 1341, University of Montreal, QC, Canada.
- [13] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'14)*, 580–587.
- [14] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, Vol. 9. 249–256.

- [15] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch SGD: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677* (2017).
- [16] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).
- [17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*. 770–778.
- [18] Jeffrey Heer, Michael Bostock, and Vadim Ogievetsky. 2010. A tour through the visualization zoo. *Communications of the ACM* 53, 6 (2010), 59–67.
- [19] Jeffrey Heer, Nicholas Kong, and Maneesh Agrawala. 2009. Sizing the horizon: The effects of chart size and layering on the graphical perception of time series visualizations. In *Proceedings of ACM SIGCHI Conference on Human Factors in Computing Systems*. 1303–1312.
- [20] Waqas Javed, Bryan McDonnell, and Niklas Elmquist. 2010. Graphical perception of multiple time series. *IEEE TVCG* 16, 6 (2010), 927–934.
- [21] Minsuk Kahng, Pierre Andrews, Aditya Kalro, and Duen Horng Chau. 2018. ActiVis: Visual exploration of industry-scale deep neural network models. *IEEE TVCG* 24, 1 (2018), 88–97.
- [22] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning Multiple Layers of Features from Tiny Images*. Master's thesis. Department of Computer Science, University of Toronto.
- [23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Proceedings of Conference on Neural Information Processing Systems (NIPS'12)*. 1097–1105.
- [24] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.
- [25] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002* (2017).
- [26] Mengchen Liu, Jiabin Shi, Kelei Cao, Jun Zhu, and Shixia Liu. 2018. Analyzing the training processes of deep generative models. *IEEE TVCG* 24, 1 (2018), 77–87.
- [27] Mengchen Liu, Jiabin Shi, Zhen Li, Chongxuan Li, Jun Zhu, and Shixia Liu. 2017. Towards better analysis of deep convolutional neural networks. *IEEE TVCG* 23, 1 (2017), 91–100.
- [28] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*. 3431–3440.
- [29] Eliana Lorch. 2016. Visualizing deep network training trajectories with PCA. In *Proceedings of the ICML Workshop on Visualization for Deep Learning*.
- [30] Aravindh Mahendran and Andrea Vedaldi. 2015. Understanding deep image representations by inverting them. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*. 5188–5196.
- [31] Nicola Pezzotti, Thomas Höllt, Jan van Gemert, Boudewijn PF Lelieveldt, Elmar Eisemann, and Anna Vilanova. 2018. DeepEyes: Progressive visual analytics for designing deep neural networks. *IEEE TVCG* 24, 1 (2018), 98–108.
- [32] Paulo E. Rauber, Samuel G. Fadel, Alexandre X. Falcao, and Alexandru C. Telea. 2017. Visualizing the hidden activity of artificial neural networks. *IEEE TVCG* 23, 1 (2017), 101–110.
- [33] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* 115, 3 (2015), 211–252.
- [34] Christin Seifert, Aisha Aamir, Aparna Balagopalan, Dhruv Jain, Abhinav Sharma, Sebastian Grottel, and Stefan Gumhold. 2017. Visualizations of deep neural networks in computer vision: A survey. In *Transparent Data Mining for Big and Small Data*, Tania Cerquitelli, Daniele Quercia, and Frank Pasquale (Eds.). Springer, 123–144.
- [35] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
- [36] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. 2014. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806* (2014).
- [37] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. 2016. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261* (2016).
- [38] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*. 1–9.
- [39] Edward R Tufte. 1983. *The Visual Display of Quantitative Information*. Graphics Press.
- [40] Andreas Veit, Michael J. Wilber, and Serge Belongie. 2016. Residual networks behave like ensembles of relatively shallow networks. In *Proceedings of Conference on Neural Information Processing Systems (NIPS'16)*. 550–558.
- [41] Matthew D. Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV'14)*. 818–833.

- [42] Matthew D. Zeiler, Graham W. Taylor, and Rob Fergus. 2011. Adaptive deconvolutional networks for mid and high level feature learning. In *Proceedings of IEEE International Conference on Computer Vision (ICCV'11)*. 2018–2025.
- [43] Haipeng Zeng, Hammad Haleem, Xavier Plantaz, Nan Cao, and Huamin Qu. 2017. CNNComparator: Comparative analytics of convolutional neural networks. *arXiv preprint arXiv:1710.05285* (2017).

Received August 2017; revised December 2017; accepted February 2018