Do Convolutional Neural Networks Learn Class Hierarchy?

Bilal Alsallakh, Amin Jourabloo, Mao Ye, Xiaoming Liu, Liu Ren

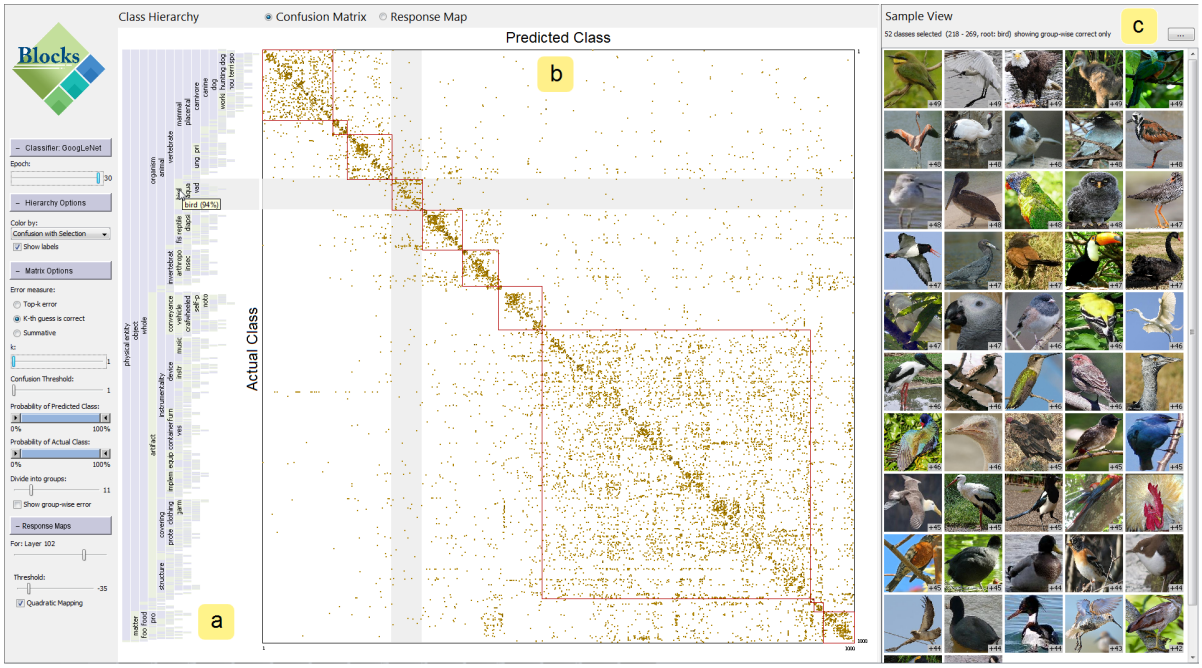

Fig. 1. The user interface of our system, showing classification results of the ImageNet ILSVRC dataset [56] using GoogLeNet [64]. (a) The class hierarchy with all classes under *bird* group selected. (b) The confusion matrix showing misclassified samples only. The bands indicate the selected classes in both dimensions. (c) The sample viewer shows selected samples grouped by actual class.

Abstract—
Convolutional Neural Networks (CNNs) currently achieve state-of-the-art accuracy in image classification. With a growing number of classes, the accuracy usually drops as the possibilities of confusion increase. Interestingly, the class confusion patterns follow a hierarchical structure over the classes. We present visual-analytics methods to reveal and analyze this hierarchy of similar classes in relation with CNN-internal data. We found that this hierarchy not only dictates the confusion patterns between the classes, it furthermore dictates the learning behavior of CNNs. In particular, the early layers in these networks develop feature detectors that can separate high-level groups of classes quite well, even after a few training epochs. In contrast, the latter layers require substantially more epochs to develop specialized feature detectors that can separate individual classes. We demonstrate how these insights are key to significant improvement in accuracy by designing hierarchy-aware CNNs that accelerate model convergence and alleviate overfitting. We further demonstrate how our methods help in identifying various quality issues in the training data.

Index Terms—Convolutional Neural Networks, deep learning, image classification, large-scale classification, confusion matrix

1 Introduction

Object recognition is a fundamental problem in computer vision that involves classifying an image into a pre-defined number of classes. Convolutional Neural Networks (CNNs) have achieved state-of-the-art results on this problem, thanks to the availability of large and labeled datasets and of powerful computation infrastructure [36]. CNNs automatically extract discriminative classification features from the training images and use them in combination to recognize complex objects. This enables CNNs to significantly outperform traditional computer vision approaches on large-scale datasets such as ImageNet [16], as the latter usually rely on heuristic features [14, 41].

To make CNNs applicable to critical domains, it is important to evaluate the reliability of the features they learn and to understand possible reasons behind classification errors [55]. A number of powerful techniques have been proposed to visualize these features in the image space. These visualizations demonstrate the power of these features and support the analogy between CNNs and natural vision systems. However, little focus has been given to visualize the classification error itself and to refine CNNs accordingly.

We repeatedly observed that classification error follows a hierarchical grouping pattern over the classes. We present a visual-analytics system, called *Blocks*, to investigate this class hierarchy and to analyze its impact on class confusion patterns and features developed at each layer in the CNN. *Blocks* integrates all three facets of classification

- *Bilal Alsallakh, Mao Ye, and Liu Ren are with Bosch Research North America, Palo Alto, CA. E-mail: bilal.alsallakh@us.bosch.com, mao.ye2@us.bosch.com, liu.ren@us.bosch.com*
- *Amin Jourabloo and Xiaoming Liu are with Michigan State University. E-mail: jourablo@msu.edu and liuxm@cse.msu.edu.*

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/

data when inspecting CNNs: input samples, internal representations, and classification results. It enables scalable inspection of these facets, at the scale of ImageNet, in order to:

- Identify various sources of classification error (**T1**).
- Exploit the hierarchical structure of the classes to improve the CNN architecture, training process, and accuracy (**T2**).
- Analyze the CNN's sensitivity to data variation and curate a balanced training data that improves its robustness (**T3**).

These tasks involve the high-level goals of visualizing machine-learning data as characterized by Liu et al. [40]: *understand*, *diagnose*, and *improve*. Section 4 illustrates how *Blocks* enables these tasks and reports quantitative results of how involving the class hierarchy reduces the top-5 error of a reference CNN by more than one third.

2 Motivation and Background

The yearly ImageNet Large Scale Visual Recognition Competition (ILSVRC) challenges participants to classify images into one thousand object categories chosen randomly from ImageNet [56]. In 2012, Krizhevsky et al. [34] trained a CNN classifier which won the competition by a large margin. This led to a paradigm shift in computer vision, with extensive research to understand how CNNs work.

We examined classification error of publically-available CNNs, pre-trained on the ILSVRC 2012 training set. For this purpose we generated confusion matrices which show how often a pair of classes are confused for each other when classifying the corresponding validation set. By re-ordering the rows and columns of these matrices by similarity, we consistently found two major blocks along the diagonal which contain more than 98% of misclassifications. One block corresponds to natural objects such as plants and animals, while the other block represents artifacts such as vehicles and devices. This means that CNNs rarely confuse natural objects for artifacts or vice versa. By reordering each block individually, we found that it in turn contains sub-blocks that capture the majority of confusions. This sparked our interest to investigate how these structures can be exploited to improve classification accuracy of CNNs.

2.1 ImageNet and the ILSVRC 2012 Dataset

Curated in 2009, ImageNet is the largest publically available labeled image dataset, encompassing more than 14 million images that belong to more than $20,000$ object categories [16]. The object categories are nouns in the WordNet database of the English language [45] .

A fundamental property of WordNet is its hierarchical organization of concepts, e.g. *birds* are *vertebrates*, *vertebrates* are *organisms*, and so on. The 1000 classes of the ILSVRC 2012 dataset are leaf nodes in this hierarchy that are randomly selected according to certain criteria that aim to reduce ambiguities. The dataset contains about 1.2 million images in the training set in addition to $50,000$ images in the validation set. By ordering these classes according to the WordNet hierarchy, we found the same class grouping structure we observed in the confusion matrices (Fig. 1). After examining the CNN classifiers, we found that they surprisingly did not make any use of the class hierarchy information in the training phase. Deng et al. [15] made a similar observation after comparing a number of classifiers on ImageNet, concluding that visual object categories are naturally hierarchical. In this work we examine how this hierarchical structure impacts CNNs.

2.2 Convolutional Neural Networks (CNNs)

CNNs are a special type of feed-forward neural networks that contain a number of *convolutional* layers. A convolutional layer consists of learnable filters that respond to certain features when convolved with a 2D input, producing a filtered 2D output. The first convolutional layer is applied to the input image, whereas subsequent layers take the output of the respective preceding layer as input (Fig. 7). Special layers are inserted between convolutional layers to reduce the dimensionality and to add necessary non-linearity [37].

After training a CNN, the convolutional filters become feature detectors in the image. Appropriate visualization techniques can reveal the features these filters respond to, as we explain next.

2.3 State of the Art in Visualizing CNNs

Visualization has played a major role in understanding and optimizing CNNs. A major focus has been made on visualizing the image features each filter learns to detect. Further techniques have addressed different aspects of the data involved in the CNN pipeline.

2.3.1 Feature Visualization

Image-based visualizations are a natural way to inspect the feature detectors developed by a CNN. Various techniques have been proposed for this purpose, based on four main approaches [24, 49, 43]:

- **Input modification / occlusion**: these techniques aim to reveal which regions in real images contribute most to a target response. This is done by occluding different regions of the input image individually and measuring the impact on the target using a forward pass [75, 77]. The result is usually a coarse 2D saliency map (also called activation map) which visualizes the importance of each region of the image to the target response.
- **Deconvolution**: these techniques also aim to find which parts in real images contribute most to a target response. In contrast to input modification, the response is traced *backwards* to the input image by reversing the operations performed at each layer using various methods [4, 58, 60, 75]. This produces a fine-grained saliency map of the input image at the pixel level, which in turn reveals the learned features and their structure (Fig. 4).
- **Code inversion**: these techniques first apply the CNN to a real image and compute the collective response, called the code, of a particular layer. An image is then synthesized that would create a similar response at this layer using various methods [42, 18, 43]. Comparing both images reveals which features of the real image are retained at the selected layer. Caricaturization can further emphasize these features [43].
- **Activation maximization**: these techniques, unlike previous ones, do not require a real image. Instead, they synthesize an artificial image that, if used as input, would maximize a target response. Early techniques often produced synthetic images that were hard to recognize [58, 74]. Accounting for the multi-faceted nature of neurons [49, 71] and imposing natural image priors [43, 47] have significantly improved the interpretability.

The above techniques were shown useful in diagnosing CNNs and in refining the architecture accordingly [75]. However, they fall short of exposing high-level patterns in collective CNN responses computed for images of all classes.

2.3.2 Projection-based Visualization

These techniques aim to provide overview of network-internal data by projecting them into a 2D space using various projection methods.

A typical use of projection is to assess class separability at different layers or at different iterations during training [17, 76, 53]. This helps in comparing classification difficulty of different datasets as well as identifying under-fitting and over-fitting models. When the projected data correspond to the input images, icons of these images can be used instead of dots [3, 49, 51]. This helps in observing visual patterns in the samples, such as possible latent subclasses among the samples of one class (e.g. red and green peppers) [3, 49]. Scatter plots have also been used to reveal similarities between neurons [13, 53] and to compare learning trajectories of multiple networks [20].

2.3.3 Network-based Visualization

Many techniques emphasize the feed-forward structure in CNNs by showing neurons as nodes in successive layers connected by links, and mapping data facets on top of them. This has been shown useful to inspect how the network classifies a selected or user-generated sample [26, 67]. *ReVACNN* [13] enables inspecting how neuron activations develop during training. *Tensorflow Playground* [59] enables novice users to construct a network and interactively observe how it learns to separate between two classes in datasets of varying difficulty.
CNNVis [39] is a powerful system designed to diagnose deep CNNs.

It employs various clustering algorithms to group neurons in representative layers based on response similarity and to group connections between these layers accordingly. A neuron cluster can be augmented with thumbnail images showing stimuli that activate these neurons most. The authors demonstrate how *CNNVis* exposes a variety of issues in network design such as redundancies in layers and neurons, as well as inappropriate learning parameters.

Finally, several frameworks offer a visualization of network architecture [57, 73]. This is useful to comprehend large networks and to compare multiple architectures.

2.3.4 Training Data and Performance Visualization

The majority of previous work focused on CNN-internal data as key to understand and optimize CNNs. Besides appropriate architecture and learning parameters, the quality of training data is also essential to learning generalizable CNNs models. Inspecting the quality of training data is nontrivial especially due to the large volume of data needed to train CNNs. In an attempt to address this issue, NVIDIA released *DIGITS*, a system that enables users to browse image datasets and inspect images of a certain class [73]. Users can apply image transformations such as cropping or resizing to match the CNN input size while preserving important parts of the image. Additional plots such as line charts and confusion matrices allow inspecting the performance. The system is limited to datasets encompassing a few dozens of classes, and does not link performance results with the input data.

Our work aims to fill the gap in available tools by offering an integrated exploration environment to analyze all three data facets involved in the CNN pipeline: input images, CNN-internal data, and classification results. Offering this integration in a scalable way is key to an advanced analysis of large-scale CNNs and to close the analysis loop by guiding model refinements that improve the accuracy.

3 *Blocks*

Being the target of classification, the class information is the most salient information along the classification pipeline. It is present both in the labeled input and in the output, and it largely determines the features learned by the CNN. Classes have varying degrees of discriminability. Some classes have unique features such as strawberries and zebras, while other classes might share similar features and are hence harder to distinguish from each other. Hinton et al. [27] noted that such similarity structures in the data are very valuable information that could potentially lead to improve classifiers. Our work offers new means to analyze these structures and their impact on CNNs.

With a growing number of classes, the similarity structures between them become complex. As we mentioned in Section 2, a key observation about these structures is their hierarchical nature: classes within the same branch of the hierarchy are increasingly more similar to each other than to other classes. We designed our visual analytics system around this idea. In particular, we focus on revealing the hierarchical similarity structure among the classes and on analyzing how it impacts both the classification results and the image features the CNN learns to recognize. We call our system *Blocks* as it extensively relies on visual block patterns in identifying similarity groups.

The main interface of *Blocks* consists of four views that show different facets of the data: the *hierarchy viewer* (Fig. 1a), the *confusion matrix* (Fig. 1b), the *response map* (Fig. 3c), and the *sample viewer* (Fig. 1c). The first three views show information aggregated at the class level and use a unified class order, dictated by the class hierarchy. The sample viewer shows image samples according to user selections in the other views. Each view contributes in certain ways to the high-level analysis tasks **T1**-**T3** listed in Section 1. At a time, the user can display either the confusion matrix or the response map as the active view. The hierarchy viewer is displayed to the left of the active view and indicates the class ordering along the vertical axis.

The class hierarchy can be either pre-defined or constructed interactively with help of the confusion matrix (Section 3.2.1). The next sections describe the above-mentioned views, illustrated on the ILSVRC 2012 dataset, classified using GoogLeNet [64]. This dataset has a predefined class hierarchy, as explained in Section 2.1.

3.1 Class Hierarchy Viewer

Blocks shows the class hierarchy using a horizontal icicle plot [35] along the vertical dimension (Fig. 1). Each rectangle in this plot represents a group of classes. The rectangle color can encode information about this group such as a group-level performance metric (Fig. 6). These metrics are computed by considering the groups to be the classification target. A sample is correctly classified with respect to a group if both its actual and predicted classes are in the group. This enables defining the following metrics:

- **Group-level precision**: this measures how many of the samples classified in a group actually belong to the group.
- **Group-level recall**: this measures how many of the samples that actually belong to a group are classified into the group.
- **Group-level F-measure**: this can be defined based on group-level precision and recall as follows:

$$F_1(g) = 2 \cdot \frac{\text{Precision}(g) \cdot \text{Recall}(g)}{\text{Precision}(g) + \text{Recall}(g)} \quad (1)$$

As we show in Section 4, inspecting group-level performance under different conditions reveals the impact of the hierarchical structure on CNN performance (**T2**) and its sensitivity to data variation (**T3**).

The child nodes of a parent node in the hierarchy can be sorted by a user-selected criterion, such as size or performance metrics. Nodes that have only one child are contracted to compress the hierarchy and reduce the visual complexity. Hovering the mouse over a rectangle shows information about the respective group including its label and performance metrics. Clicking on a rectangle selects the corresponding classes and updates the other views to focus on these classes. This enables inspecting their samples and analyzing their confusion patterns and CNN-internal responses.

3.2 Confusion Matrix

Confusion matrices have been utilized in the machine learning community for various purposes such as detailed comparison of performance and identifying frequent confusion between certain classes. We argue that these matrices can reveal further information about error structure (**T1** and **T2**) and classifier behavior (**T2**) when equipped with appropriate ordering, visual encoding, and user interactions.

3.2.1 Class ordering - constructing the class hierarchy

A confusion matrix is re-orderable [8], as long as the same class order is used along the rows and columns. This ensures that the correct classifications are encoded along the matrix diagonal. The desired ordering should reveal similarity groups among the classes. This corresponds to a *block pattern* in the matrix [6]: the majority of confusion takes places within a number of blocks along the diagonal, each of which corresponds to a similarity group of classes.

In case a pre-defined class hierarchy is available, *Blocks* displays it in the hierarchy viewer and orders the matrix accordingly. If such a hierarchy is unavailable or fails to reveal a block pattern, the user can explore if such pattern exists by interactively applying a seriation algorithm. Behrisch et al. [6] surveyed various seriation algorithms that can reveal block patterns in matrices. *Blocks* offers both fast algorithms [28, 44] and exhaustive ones such as spectral clustering [25].

The hierarchy can be refined recursively, as proposed by Griffin and Perona [23]: the user selects a high-level block and applies the algorithm on this part. At each step, the matrix is updated to allow inspecting the plausibility of the computed sub-blocks and to guide algorithmic choices. If plausible, the hierarchy viewer is updated to reflect the constructed hierarchical structure.

After the class hierarchy and the corresponding block patterns are established, it is possible to distinguish between non-diagonal matrix cells based on their location in the matrix: Cells that are within a dense block represent confusions between highly-similar classes. Cells that do not belong to a block represent unexpected confusions between classes that seem to be less related, and are hence especially interesting to explore further (Section 4.3). We call these cells *block outliers*.

3.2.2 Visual encoding

Besides an appropriate class ordering, the visual encoding of the cell values plays a major role in revealing block patterns and their outliers. In machine-learning literature, confusion matrices are often generated using the default *Jet* color map in MATLAB [22, 32, 46]. Instead, we use a sequential color scale which maps the value 1 to a light shade and the largest value to a dark shade. Cells with value 0 remain white, which facilitates identifying and selecting non-zero cells that represent actual confusions (Fig. 1b and Fig. 2).

Focusing on misclassification By default, we exclude the matrix diagonal from the visual mapping since correct classifications usually account for the majority of the value sum in the matrix. This eliminates an, otherwise, salient diagonal which interferes with fine-grained block patterns. The per-class accuracy can be displayed more appropriately on top of the class hierarchy or in the sample viewer.

Non-linear mapping Even among off-diagonal cells, there is typically a large variation in values. While the majority of non-zero cells typically have small values, a very small number of cells might have large values and indicate classes that are very frequently confused for each other. To alleviate such variation, the user can select a logarithmic mapping of values to color, which helps emphasize less frequent confusions that form the block patterns. Interactive filtering allows identifying cells that represent frequent class confusions.

Visual boosting Even though standard displays offer sufficient space to map a 1000×1000 matrix to pixels without overlaps, assigning one pixel to a cell makes it barely visible, which might leave block outliers unnoticed. The user can select to emphasize non-zero cells by enabling a halo effect [50], which extends 1-pixel cells into 3×3 pixels and assigns 30% opacity to the peripheral halo area. This effect not only emphasizes block outliers, it further improves the perception of blocks and sub-blocks within them. The halos are visual artifacts that might add shade to, otherwise, empty cells. Individual confusions can hence be examined more precisely using interaction.

3.2.3 Interaction

Blocks enables various interactions with the confusion matrix. As we illustrate in the supplementary video, these interactions are essential to identify various sources of classification errors (**T1**), especially those related to data quality issues (Section 4.3).

Selection There are two ways to select samples in the matrix:

- Drawing a box around certain cells. This updates the sample viewer to show the corresponding samples.
- Clicking on a group in the class hierarchy. This highlights false positives (FPs) and false negatives (FNs) with respect to the group classes by means of vertical and horizontal bands (Fig. 1). The intersection of these bands are confusions between classes that belong to the selected group and hence represent group-level true positives (TPs). The difference of these bands corresponds to group-level FPs and FNs respectively. The sample viewer is updated to show the highlighted samples, and allows exploring the group-level TPs, FPs, and FNs individually.

Filtering The mis-classified samples encoded in the matrix cells can be filtered according to multiple criteria. The matrix is updated to show confusion patterns among the filtered samples.

- Filtering by cell value: This retains cells representing repetitive class confusions above a selected threshold (Fig. 2). These confusions often indicate overlapping class semantics (Section 4.3).
- Filtering by top-k results: This filters out samples whose correct labels are among the top-k guesses computed by the classifier. The remaining samples represent the classifier's top-k error, a commonly-used performance measure that relaxes the requirement of correct classification by accepting multiple guesses.
- Filtering by classification probability: This retains samples for which the classifier predictions were computed with probability in a certain range. It is possible to further specify a range for the probability computed for the *actual* class.

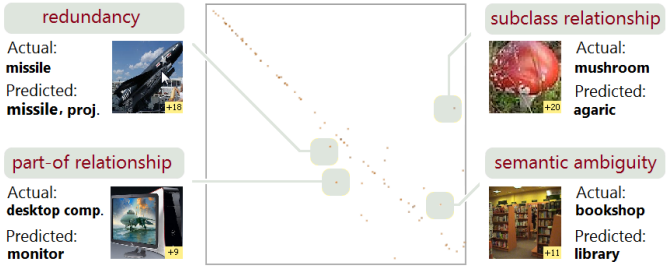

Fig. 2. Filtering out diagonal cells and cells whose values are < 10 to retain repetitive confusions. Near-diagonal cells correspond to highly similar classes while off-diagonal cells often indicate data quality issues.

Grouping *Blocks* enables emphasizing the block pattern in the matrix by drawing boxes around major blocks (Fig. 1). The user specifies the number of blocks b, which are then determined by a partitioning algorithm. The algorithm selects a partitioning which maximizes the density of its blocks. The boxes are retained during filtering, which helps keeping track of block memberships. It is possible to divide the matrix into $b \times b$ clickable regions based on the blocks, which eases the identification and selection of block outliers.

3.3 Response Map

This view provides overview of the CNN responses at a selected layer to all samples in the dataset. The aim is to identify whether classes in the same group activate a similar set of features, and which combination of features characterize a class or a group of classes. This enables understanding how well different layers in the network can discriminate between groups in different levels of the class hierarchy (**T2**) and how sensitive to data variation the developed features are (**T3**).

As illustrated in Fig. 3a, the neuron responses are averaged per class, over all of its samples. This aims to reveal variations in these responses across *classes* and *neurons*, not across samples. This further enables a compact visual encoding of responses computed from a large number of samples. Responses to individual samples of a particular class can be explored on demand in an auxiliary view (Section 3.3.4).

3.3.1 Visual encoding

We use a heatmap to encode per-class average response of each neuron in the selected layer (Fig. 3c). The rows of the heatmap represent the classes and are ordered according to the class hierarchy. The columns represent the neurons, and their order is updated according to user selection. A neuron can have multiple output channels as in the case of filters in convolutional layers and the associated pooling units and rectified linear units (ReLUs). *Blocks* visualizes these channels as vertical 1-pixel-wide lines within the neuron's column. This is done by linearizing these channels as illustrated in Fig. 3a. As a result, the 2-dimensional structure of the neuron's output is lost, in favor of emphasizing how its responses vary across multiple classes, which we denote as the *response profile* of the neuron.

Cell color represents the average response of a neuron's channel among samples of a certain class. The user can specify a threshold T on this response. Values smaller than T are mapped linearly to a color scale from black to light blue. Values equal to or larger than T are shown in yellow. This aims to emphasize cells representing high responses, in context of the other cells. Adjusting the threshold allows identifying neurons that respond specifically to certain classes and exploring subtle differences between different response profiles.

In some CNNs, the convolutional filters can be as large as 64×64, especially in early layers. To gain overview of multiple filters of this size in one view, *Blocks* allows downsampling their output e.g. to 8×8. Fig. 3a illustrates how the responses of a 12×12 filter are downsampled to 4×4 channels which fit in a 16-pixel-wide column. This allows comparing multiple response profiles side by side. Furthermore, this consolidates major variations between these profiles that would be, otherwise, scattered across numerous channels.

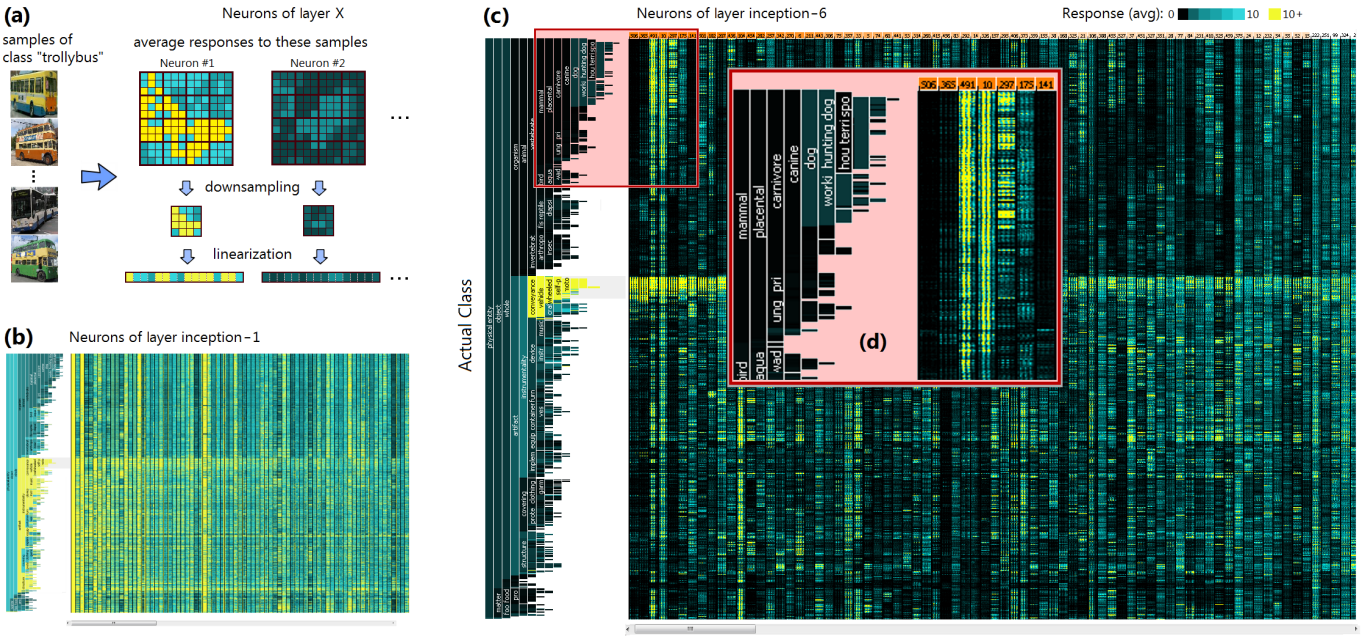

Fig. 3. The Response Map: (a) Illustrating how the row that corresponds to class *trollybus* is computed. Each column represents the average responses of a neuron in the selected layer. (b, c) The response maps of layers *inception*-1 and *inception*-6 in GoogLeNet [64]. The rows represent the classes and are ordered by the class hierarchy depicted to the left of each map. The *wheeled vehicle* group is selected, and the neurons are sorted by their relevance to it (Eq. 2). The most relevant neurons in layer inception-6 can separate the classes in this group from other classes, while inception-1 can only separate higher-level groups. (d) Pose-based detectors of *vehicles* have high responses among *mammals* as well.

3.3.2 Exploring group-level features

The unified class ordering in *Blocks* enables analyzing the relation between the response profiles of the neurons and the class hierarchy. We observe that certain profiles show high responses mainly for samples within a particular group of classes in the class hierarchy. This means that the corresponding neurons learned shared features among these classes such as shape, pose, or background. As we illustrate in the supplementary video, interaction is key to identify neurons that respond to a particular group in the class hierarchy. In Fig. 3b-c, the columns are reordered according to the ability of the corresponding neurons to distinguish *wheeled vehicles* from the other classes. For this purpose we compute a relevance measure $R_G(N)$ for each neuron N, based on its responses to group samples G and to non-group samples $\overline{G}$:

$$R_G(N) = \frac{Q_{1/4}(\{f_N(x) : x \in G\})}{Q_{3/4}(\{f_N(x) : x \in \overline{G}\})} \quad (2)$$

where $f_N(x)$ is the collective response of the neuron to a sample x, computed as the sum of all of its output channels, and $Q_{i/q}$ is the i-th q-Quantile. This measure mimics statistical significance tests and takes a high value when the response is consistently high among the group classes and consistently low among non-group classes. The column headers can communicate the computed values via color. Visual inspection enables identifying if a neuron responds to a sub-group or super-group of the selected group, or possibly to other groups as well. For example, no neuron in the early layer *inception*-1 can capture the selected group specifically (Fig. 3b), unlike the advanced layer *inception*-6 (Fig. 3c). Furthermore, certain neurons that respond to *wheeled vehicles* respond highly to *mammals* as well (Fig. 3d). These neurons detect pose features that are shared between both groups of classes.

We found that group-level features are often based on shape, pose, and background. For example, within natural objects, a combination of shape and pose features can distinguish high-level groups such as *birds*, *mammals*, and *insects*. Background features are involved in certain groups such as *fishes* and *geological formations*. On the other hand, color features as well as certain texture features are often shared across various classes that do not fall in a specific group. To facilitate analyzing such cases, the groups in the hierarchy viewer can be colored by the average response of a selected neuron (Fig. 3b-c).

For some groups, such as *devices*, no neuron exhibits significant difference in responses between group classes and non-group classes. Such lack of group-level features indicates a high variation among the group classes that develop class-specific features instead.

3.3.3 Exploring neuron properties

Besides the response profiles, *Blocks* provides additional information about a neuron either in summary or in detail forms. The header of the response map can communicate summary information about the neurons by means of color (Fig. 3c-d). Examples for this are average activation within the samples of a selected class, relevance to a selected group, or sensitivity to an image transformation (Section 4.2).

Clicking on a profile header updates the sample viewer to show samples that highly activate the corresponding neuron. This aims to help users find out common image features across these samples in order to identify the image features to which the neuron responds. The sample viewer provides several possibilities to explore the samples along with saliency maps of their image features.

Understanding the image features a neuron responds to is important to understand how each class is being detected by the CNNs and why certain samples of it are misclassified. Typically, the network characterizes each class by a set of features that are detected by different neurons. Fig. 4 illustrates image features that characterize the class *strawberry* at an intermediate layer in GoogLeNet. These features correspond to the four most relevant neurons to this class in this layer. The first neuron detects red objects, the second and third neurons detect dotted objects and objects with bumps, and the fourth neuron detects natural objects having isosceles triangular shapes. This means that strawberries are detected based on color, texture, and shape, in the respective order of importance. We found that images of unripe strawberries and strawberry images in grayscale do not activate the first neuron and are therefore often misclassified (**T1**). On the other hand, classes whose samples have varying colors such as vehicles do not rely on color. Such findings are useful to curate training data that are representative of the target classes (**T3**) as we show in Section 4.2.

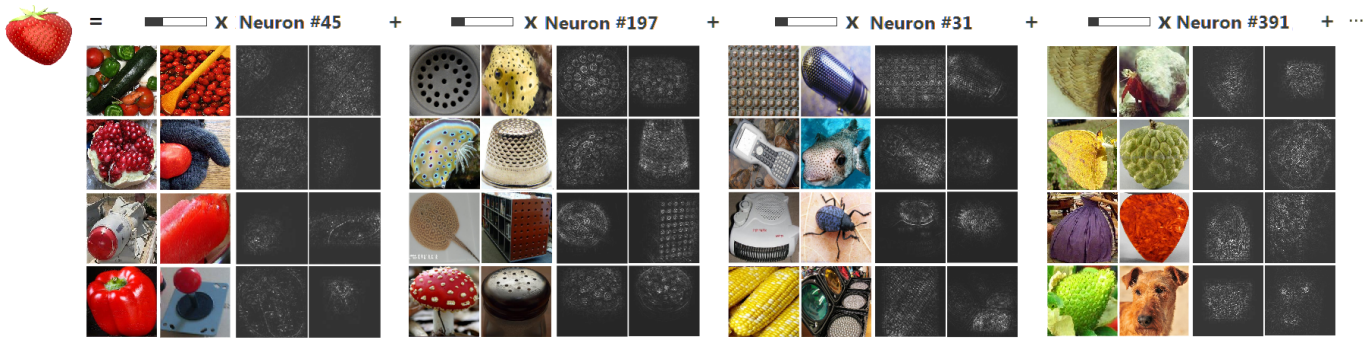

Fig. 4. Feature detectors at layer *inception-6* in GoogLeNet that show high response to samples of class *strawberry*. We depict the top-9 images in ILSVRC validation set that activate each detector most, along with the corresponding saliency maps (computed using *FeatureVis* [24]).

3.3.4 Sample-level responses and latent subclasses

The response map presented above aggregates the responses per class in order to show how they vary across different classes. In many cases, the responses vary within the same class due to latent subclasses, e.g. cut vs. full apples. *Blocks* enables exploring possible latent subclasses within the samples of a selected class in a dedicated window (Fig. 5). For this purpose, we compute the correlation matrix of network responses to these samples at a selected reference layer. We reorder the matrix using spectral clustering and visualize it along with these responses and with thumbnails of the samples. The responses are visualized using a sample-level response map which shows which neurons are active for which samples. The rows in this map represent the samples, and are assigned the same order as in the correlation matrix. The column represents the neurons of the selected reference layer. The presence of multiple blocks in the matrix indicates the presence of latent subclasses such as different types of *mushroom* (Fig. 5). Selecting a block highlights the corresponding samples and reorders the neurons according to their responses within these samples.

By inspecting the correlation matrix at successive layers, it is possible to observe how the latent subclasses emerge in the CNN. Despite activating different feature detectors in the CNN, these subclasses can still activate the same output unit. This is thanks to the final layers in CNNs being fully connected, which enables the output unit of a class to combine responses from multiple features. As noted by Nguyen et al. [49], identifying latent subclasses and analyzing their properties gives opportunities to optimize the classification process (**T2**).

3.4 Sample Viewer

The sample viewer is key to inspect classification errors (**T1**) and to analyze the impact of image transformations (**T3**). It shows thumbnail images of selected samples and offers various possibilities to manipulate and explore them (Fig. 1c). A label at the top of the view describes what the current selection represents. The samples can be grouped by their actual classes: a representative sample of each group is shown as thumbnail image along with a number indicating the count of the remaining samples. This gives an overview of all classes included in the selection and helps in inferring common features among them.

When showing individual samples, the user can obtain details about them either on top of the thumbnails or in tooltips. For example border color can indicate whether the CNN prediction for a sample is top-1 correct, top-5 correct, or otherwise. The viewer also allows exploring saliency maps of the images to analyze the role of a selected neuron in the network. These maps are computed using the *FeatureVis* library [24] and the *MatConvNet* toolbox [69]. They highlight image features the selected neuron responds to (Fig. 4).

The samples in the viewer can be filtered by various criteria such as membership of a selected class group, activation of a selected neuron, and class-level or group-level classification results. Additionally, *Blocks* allows loading multiple sets of classification results computed by different classifiers or after applying different data transformations. Users can filter the samples based on these results, e.g. to show samples correctly classified under all rotations or ones correctly classified by a selected classifier only. This enables identifying samples and classes that have certain properties such as rotation invariance and ease of discrimination, or ones that only a selected classifier excels in.

4 Applications

The components of *Blocks* offer extensive support to the analysis goals identified by Liu et al. [40], as described in Section 1. We next demonstrate how Blocks helps in *understanding* the training process, *diagnosing* the separation power of the feature detectors, and *improving* the architecture accordingly to yield significant gain in accuracy (**T2**). Additionally, we illustrate how *Blocks* helps in *improving* the curation of training datasets by *understanding* sensitivity properties of the CNN (**T3**) and *diagnosing* various quality issues in the data (**T1**).

4.1 Designing Hierarchy-Aware CNNs

Understanding the training behavior of CNNs helps in introducing targeted design improvements to large-class CNN classifiers. In particular, we show how making CNNs hierarchy-aware significantly improves the accuracy and accelerates the training convergence.

4.1.1 Understand: model convergence

The CNN classification model converges over several epochs during training phase. We inspect the model responses at each epoch and the corresponding class confusions in the respective views in *Blocks*.

Observing how the confusion matrix changes over successive epochs reveals how the final confusion patterns develop. Initially, the model is random, resulting in a uniform distribution of the values in the confusion matrix. Fig. 6a-b depicts the confusion matrix after the first two epochs while training standard AlexNet [34]. Fig. 6c depicts

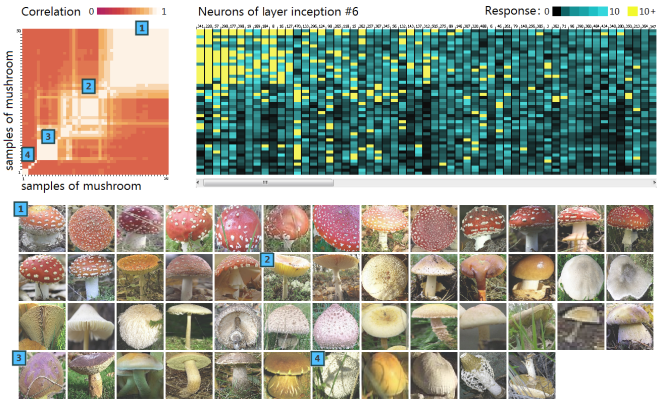

Fig. 5. The correlation matrix between the samples of class *mushroom*, along with a sample-level response map. Each block in the matrix corresponds to a sub-class of similar samples (e.g. red mushrooms).

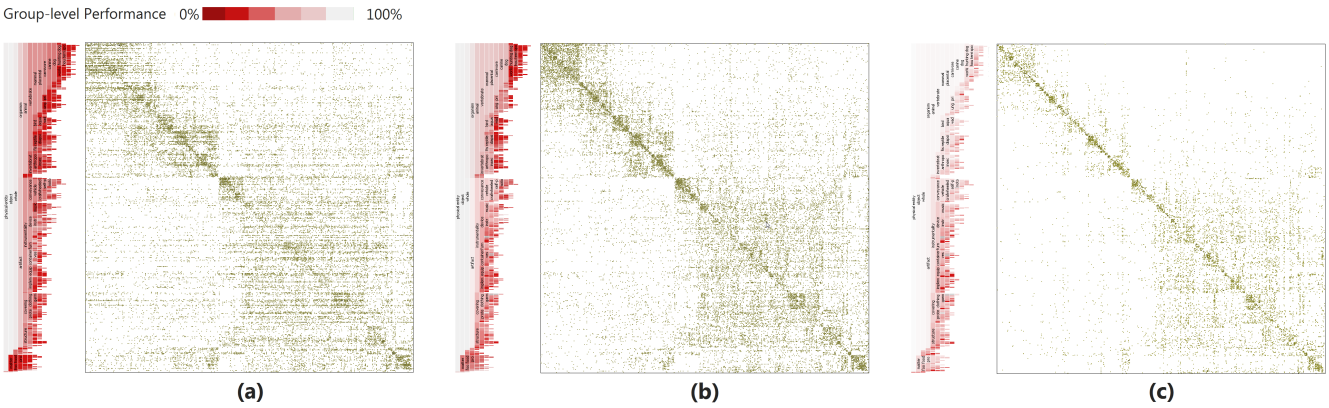

Fig. 6. The confusion matrix after the first epoch (a), the second epoch (b), and the final epoch (c) during the training of AlexNet [34]. The network starts to distinguish high-level groups already after the first epoch. The hierarchy viewers show the corresponding group-level accuracies.

the matrix after the training is terminated. It is remarkable that major blocks are already visible after only one epoch of training. This means that the network first learns to distinguish major high-level groups such as natural objects vs. artifacts. In the second epoch, the separation between these groups improves and subgroups within them emerge. In the final epoch, the CNN makes fewer overall confusions that are generally limited to narrow groups.

To further analyze this behavior, we observe how the feature detectors develop during the training. We found out that the response profiles of neurons in early layers quickly converged in the first and second epoch, with subsequent epochs leading to increasingly smaller changes. These low-level features seem to be capable of separating high-level groups, as the confusion matrices suggest. In contrast, the response profiles in deeper layers converged at later epochs, with changes in these epochs being increasingly limited to the last layers. Zeiler and Fergus reported similar findings by observing the development of feature detectors during training [75]. To confirm our observations, we next analyze the classification power of individual layers.

4.1.2 Diagnose: feature classification power

Blocks allows analyzing at which layer in the CNN the feature detectors are able to separate certain groups of classes. Each layer in the CNN abstracts the input image into a set of responses that indicate the presence of increasingly more complex features in the image. To assess the classification power of the feature detectors at a certain layer, we train a linear classifier to classify the samples based on these features only, as proposed by Rauber et. al [53]. This classifier characterizes each class by a weighted sum of the feature responses, and classifies a sample by computing corresponding class scores. To analyze the performance of this linear classifier, we create a confusion matrix of its predictions. Additionally, we color the groups in the hierarchy viewer by group-level recall. This reveals which groups the features at each layer can already separate from each other.

We are able to confirm that the features developed at early layers can separate between high level groups with group-level performance close to the output layer. Separating between fine-grained groups requires more sophisticated features that are developed at deeper layers.

We noticed that while AlexNet is able to separate dogs from other classes, it frequently confused certain types of dogs in ImageNet for each other (see topmost block in Fig. 1). Szegedy et al. [64] argued for the need of additional convolutional layers to separate highly-similar classes. Accordingly, their GoogLeNet CNN achieves higher accuracy than AlexNet on such classes. However, by comparing the performance of both CNNs, we found that GoogLeNet achieves lower accuracy for certain classes such as 'ping-pong ball' and 'horizontal bar'. The samples of these classes are composed of simple features, which suggests that they do not benefit from deep architectures. Moreover, we found that classifying these samples based on intermediate features in GoogLeNet achieves higher accuracy than the output layer. This suggests that classification decisions should be taken at different layers in deep CNNs to account for the varying complexity of the classes. Similar proposals were shown to improve classification accuracy such as variable-depth CNNs [66] and conditional networks [29].

4.1.3 Improve: exploiting the class hierarchy

Our findings about model convergence and group separability at different layers enable us to improve training speed and accuracy, by involving the hierarchy information in the design and training of CNNs.

We select AlexNet [34] as a reference architecture that is straigtforward to extend and re-train. After analyzing the classification power of convolutional layers, we extended them to be hierarchy-aware. For this purpose, we created branches from these layers that perform group-level classification and back-propagate group error (Fig. 7). We require the first layer to classify the samples into 3 broad groups only, and increased the number of groups in subsequent layers. For each layer, we selected groups that we identified as most separable using the corresponding feature detectors. These groups, along with the trained model are provided in the supplementary material.

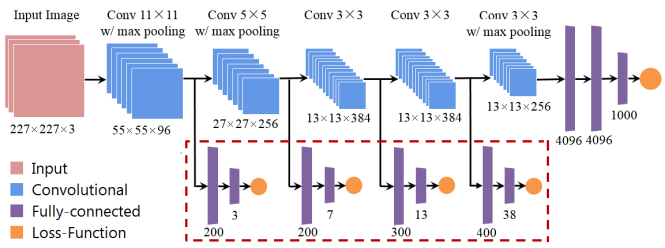

Fig. 7. The adapted AlexNet architecture. The added branches are marked with a dotted box. These branches impose the class hierarchy during the training phase and are eliminated after training completion.

We re-train the adapted network on the ILSVRC dataset for 50 epochs using Caffe [31]. Table 1 summarizes the validation error at epoch 25, compared with baseline AlexNet. The results did not improve beyond this epoch.

Architecture	Top-1 error	Top-5 error
Standard AlexNet	42.6%	19.6%
Hierarchy-Aware AlexNet	**34.33%**	**13.02%**

Table 1. Performance of baseline vs. improved architectures.

The hierarchy-aware architecture cuts the top-5 error down by more than one third. The classification results are computed from the main

branch of the network, which is identical in complexity to baseline AlexNet. The additional branches play no role after the training is completed. This means that the improved accuracy can be fully attributed to involving the hierarchy information during training.

Our results show more significant improvement on the ILSVRC dataset than *HD-CNN*, a recently proposed approach to implement hierarchical CNNs [72]. This shows the value of understanding the separation power of each layer and of introducing the hierarchy information accordingly. This is especially beneficial when the network is deep and the number of classes is large. Furthermore, the model converged quickly in our experiment, with top-5 error reaching 24.6% only after 4 epochs. This is because the additional loss functions directly update the weights of the corresponding layers to achieve group separation. This offers new solutions to the vanishing gradient problem in deep models [38]. Moreover, this aids generalizability since our trained model should satisfy multiple loss functions and is hence less likely to overfit the training data than standard CNNs.

4.2 Sensitivity to Image Transformations

The classes in ImageNet vary in their sensitivity to image transformations. In the following we analyze the impact of gray-scale color conversion and image rotation on classification accuracy. This reveals whether the corresponding features are invariant to color and rotation.

4.2.1 Color invariance

We convert the images in the ILSVRC validation dataset into grayscale and re-classify them using GoogLeNet. Figure 8 shows the impact of this transformation on the classification results. The hierarchy viewer depicts change in group-level precision for each group in the hierarchy, compared with the result of original color images. Red indicates a drop in the accuracy due to the lack of color information.

The largest absolute drop can be observed in the *food* groups such as *fruits* (−60%), *vegetables* (−43%), and *dishes* (−67%). By inspecting the confusion matrix, we found out that the CNN confuses these samples mainly for classes in other groups such as *tableware*, *cookware*, *covering*, *containers*, *fungus*, and *fishes*. In contrast, most artifact groups and classes had minimal or no change in accuracy such as *electronic equipment* (0%), *seats* (0%), *measuring instruments* (−1%), *wheeled vehicles* (−3%) and *structures* (−3%). By inspecting the training samples in these groups, we found strong variation in color. This enforces the CNN to rely on color-independent features to recognize these classes. Some exceptions were lifeboats (−84%), tennis balls (−58%), jack-o'-laterns (−48%), and lipsticks (−42%), all of

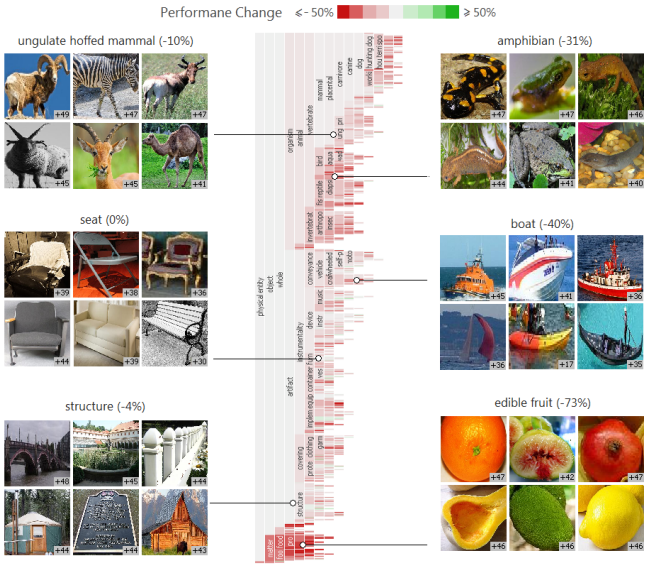

Fig. 8. Color-invariant (left) vs. color-sensitive classes (right).

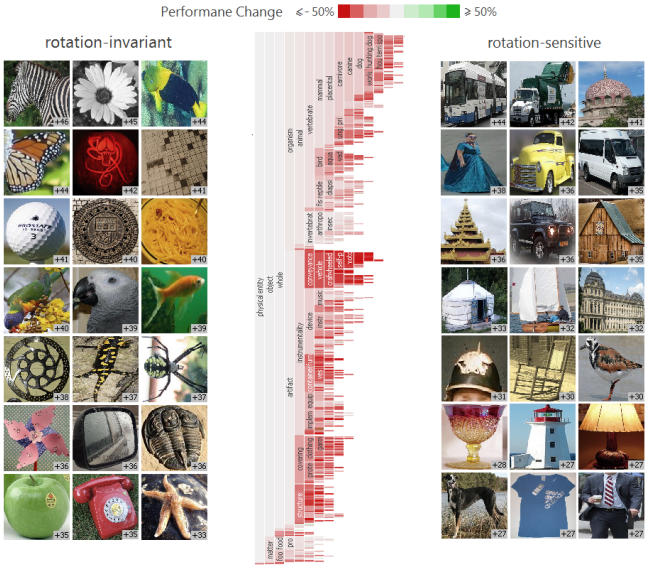

Fig. 9. Rotation-invariant (left) vs. rotation-sensitive classes (right).

which had training samples of particular color. By inspecting the corresponding features we found that the CNN relies on color-dependent features as discriminative common denominators of the corresponding samples, even if these samples have distinctive shapes.

After inspecting the changes in accuracy, the curators of training data can alleviate color dependence by including grayscale versions or additional samples of the impacted classes to balance color variation. Alternatively, the CNN architecture can be adapted to simulate rods and cones in natural vision. Color information remains necessary, however, to recognize classes with intrinsic color that are otherwise hard to distinguish from similar classes such as *green snakes*.

4.2.2 Rotation invariance

We re-classify the images in ILSVRC after rotating them by 90° and observe the change in group-level accuracy as in the previous section. By observing rotation-invariant classes (Fig. 9-left), we found that they often have circular shapes as with *ball* and *flower*, or have rotation-invariant features based on texture and color as with *zebra* and various *produce* classes. On the other hand, rotation-sensitive classes (Fig. 9-right) have non-circular shapes and mostly appear in a specific pose as with the *vehicles* and *buildings*. Accordingly the latter groups exhibit significant drop in accuracy of −55% and −50% respectively.

Among animals *invertebrates* exhibit the lowest drop of 4%, although they do not have the circular shape. By inspecting the corresponding training samples, we found that the objects exist in various rotations, which led the CNN to develop rotation-invariant features as common denominators among the samples of each class. In contrast, most samples of *aquatic birds* (−39%) and *hoofed mammals* (−45%) did have the same pose, leading to rotation-sensitive features.

4.3 Inspecting Data Quality

The classes in the ILSVRC dataset were chosen randomly from the WordNet ontology. Filtering the confusion matrix to show frequent confusions and inspecting the remaining block outliers reveals several issues with the choice of these classes (Fig. 2), such as:

- Redundancy: two classes are identical but belong to different WordNet branches such as *missile* and *projectile, missile*, *bassinet* and *cradle*, or *sunglass* and *sunglasses, dark glasses*.
- Subclass relations: one class is a special type of the other class such as *bolete* and *mushroom*, or *coffee mug* and *cup*.
- Part-of relationships: one class represents part of another class such as *wing* and *airplane*, or *monitor* and *desktop computer*.

- Semantic ambiguity: two classes have similar semantics such as *bookstore* and *library*, or *gas mask* and *oxygen mask*.
- Abstract classes: one class such as *groom* takes multiple forms that are often confused with physical classes such as *suit*.

These issues impact about 5% of the classes, and lead to a significant drop in the top-1 classification accuracy which is not caused by the classifier. Nevertheless, they apparently remained largely unnoticed due to reliance on top-5 error to compare classifiers. This error measure, however, intends to account for images that actually contain multiple objects, and is usually not used during the training phase. Ensuring non-overlapping class semantics helps in sharpening their feature detectors and improving the overall performance accordingly.

Blocks also helps in detecting mislabeled samples such as an image of a lion labeled as monkey. We found such cases by inspecting misclassified samples having very high prediction probability and very low probability assigned to the ground truth. Isolating such cases is useful to robustly compare different architectures. Finally, *Blocks* helps in restructuring the pre-defined class hierarchy of ImageNet to better reflect their visual similarity structures. For example, the groups *fruit* and *editable fruit* belong to different branches of the hierarchy root despite having high similarity, which led to frequent inter-group confusions.

5 Discussion

Blocks is the first system to enable analyzing the impact of a class hierarchy on CNNs and improving their design accordingly. Next we discuss how *Blocks* relates to previous work, summarize its limitations, and report feedback of deep-learning experts on our system.

5.1 Related Work

Previous work has utilized similar visualizations to the ones in *Blocks*, focusing, however, on different data facets or tasks.

Confusion matrices have been utilized to manipulate decision boundaries as in *ManiMatrix* [33], to combine multiple classifiers as in *EnsembleMatrix* [65], and to examine impact of model changes as in BaobabView [68]. Little focus has been given to revealing nested block patterns in these matrices, unlike matrices showing correlations [70] or distances [9] between the samples. Alternatives to confusion matrices have focused on prediction probabilities [1, 2, 12, 54] or on the ground truth [5], and hence do not involve the class hierarchy.

Heatmaps have also been used to visualize selected responses for single samples both in the input space [30] and in the class space [7]. *CNNVis* utilizes a *class* $\times$ *neuron* response map to show activation patterns within certain neuron groups [39]. Nevertheless, these maps are not designed to provide a comprehensive overview of the responses or to reveal group-level response patterns, a key focus of *Blocks*.

Sample viewers are often integrated in machine learning environments to inspect individual samples along with their attributes and models responses [1, 10, 52]. *LSTMVis* [63] features a powerful viewer for text data. It allows comparing multiple sentences to reveal linguistic properties captured by each hidden state in LSTM neural networks. In contrast, available viewers for CNN-based image classification data have focused mainly on visualizing image features for a few samples [11, 39, 74]. Unlike *Blocks*, they lack possibilities to explore a large number of samples and compare multiple result sets.

5.2 Scalability and Limitations

As we demonstrated in previous sections, *Blocks* supports analyzing complex CNN classifiers such as AlexNet [34] and GoogLeNet [64], trained to classify datasets at the scale of ILSVRC (Section 2.1).

The grouping of the classes is vital to support scalability with the number of classes. High-level groups are easy to identify in the hierarchy viewer, as their labels can be depicted. Small groups can still be identified interactively by means of tooltips. Selecting one of these groups shows thumbnails of its classes in the sample viewer, which in turn makes these classes easy to identify and select individually.

The confusion matrix view can handle a 1000×1000 matrix without need for scrolling. Multiscale aggregation [19] enables handling larger matrices, thanks to similarity-based class ordering. While this does not show confusion between individual classes, it provides overview of major block patterns and block outliers.

The response map can provide overview of neuron responses in a selected layer to a large number of samples, thanks to per-class aggregation and downsampling. A typical intermediate layer in the CNNs we examined contains about 512 4×4 filters. A standard 1920×1080 display can hence fit about $15 - 20\%$ of the corresponding response profiles, along with the auxiliary views. This is sufficient to explore the most relevant profiles for selected classes or groups, thanks to relevance-based ordering.

Besides scalability limits, *Blocks* is also limited in the data facets it shows in CNNs. Unlike *CNNVis*, *Blocks* does not provide information about layer connectivity and hence does not reveal patterns in the connection weights. Furthermore, the layer responses are visualized independently for each layer. This hinders close inspection of how the CNN develops the feature detectors, in particular how the detectors in one layer rely on the ones in previous layers. We envision that combining features from *Block* and *CNNVis* might provide such possibilities.

Finally, *Blocks* currently offers few possibilities to monitor the training process, limited to changes in the confusion matrix and response map. Further work is needed to closely examine the impact of various training parameters on the CNN features and performance, including initialization strategies such as pre-training [20, 21], learning rate, and regularization strategies such as DropOut [61].

Except for the response map, the views in *Blocks* are not restricted to CNN classifiers. Furthermore, this map can visualize internal responses of any classifier that is based on a number of feature detectors. This makes *Blocks* a potentially generic tool to analyze large-class classifiers, focusing on how an explicit or latent class hierarchy impacts the classification model and performance.

5.3 Expert Feedback

We solicited feedback on our system from an external expert in CNNs who developed various CNN visualization systems [47, 48, 49, 74]. He finds '*the visualizations are easy to follow and make sense*' and '*the idea of comparing the classes along the hierarchy is novel*'. He further comments: '*I have not seen a tool that puts all these really useful features together! Traditionally, one would have to write code to perform these analyses manually. This tool would be incredibly useful and advance science further.*' These insights we report in Section 4 demonstrate the value of the visualization, as proposed by Stasko [62]. A further study is needed to assess the usability of our system.

6 Conclusion and Future Work

We presented visual-analytics methods to inspect CNNs and to improve their design and accuracy on large-scale image classification. Our methods are based on identifying the hierarchical similarity structures between the classes as key information that impacts various properties of CNNs. These structures influence the feature detectors developed by the CNN at different layers and over different training epochs. We demonstrated how understanding these influences help in designing hierarchy-aware CNN architectures that yield significant gain in classification accuracy and in convergence speed. We further demonstrate how extracting and analyzing the class similarity structure can reveal various quality issues in the training dataset such as overlapping class semantics, labeling issues, and imbalanced distributions. This is key to improve the CNN robustness to data variation by curating a representative dataset. Our future work aims to study how class similarity structures influence other types of large-scale classifiers and how our findings can be generalized to domains other than image classification.

Acknowledgments

We thank Jitandra Malik for encouraging us to pursue our initial ideas, Anh Nguyen for feedback and Felix Grün for help on FeatureVis.

REFERENCES

[1] B. Alsallakh, A. Hanbury, H. Hauser, S. Miksch, and A. Rauber. Visual methods for analyzing probabilistic classification data. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1703–1712, 2014.

[2] S. Amershi, M. Chickering, S. M. Drucker, B. Lee, P. Simard, and J. Suh. ModelTracker: Redesigning performance analysis tools for machine learning. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 337–346. ACM, 2015.

[3] M. Aubry and B. C. Russell. Understanding deep features with computer-generated imagery. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2875–2883, 2015.

[4] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[5] E. Beauxis-Aussalet and L. Hardman. Visualization of confusion matrix for non-expert users. In *IEEE Conference on Visual Analytics Science and Technology (VAST) - Poster Proceedings*, 2014.

[6] M. Behrisch, B. Bach, N. Henry Riche, T. Schreck, and J.-D. Fekete. Matrix reordering methods for table and network visualization. *Computer Graphics Forum*, 35(3):693–716, 2016.

[7] A. Bendale and T. E. Boult. Towards open set deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1563–1572, 2016.

[8] J. Bertin. Semiology of graphics: diagrams, networks, maps. 1983.

[9] R. Brasselet, R. Johansson, and A. Arleo. Optimal context separation of spiking haptic signals by second-order somatosensory neurons. In *Advances in Neural Information Processing Systems (NIPS)*, pages 180–188, 2009.

[10] M. Brooks, S. Amershi, B. Lee, S. M. Drucker, A. Kapoor, and P. Simard. FeatureInsight: Visual support for error-driven feature ideation in text classification. In *Visual Analytics Science and Technology (VAST), 2015 IEEE Conference on*, pages 105–112. IEEE, 2015.

[11] D. Bruckner. ML-o-scope: a diagnostic visualization system for deep machine learning pipelines. Technical report, University of California at Berkeley, 2014. UCB/EECS-2014-99.

[12] N. Cao, Y.-R. Lin, and D. Gotz. Untangle map: Visual analysis of probabilistic multi-label data. *IEEE Transactions on Visualization and Computer Graphics*, 22(2):1149–1163, 2016.

[13] S. Chung, C. Park, S. Suh, K. Kang, J. Choo, and B. C. Kwon. ReVACNN: Steering convolutional neural network via real-time visual analytics. In *NIPS Workshop - The Future of Interactive Machine Learning*, pages 577–585, 2016.

[14] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893. IEEE, 2005.

[15] J. Deng, A. C. Berg, K. Li, and L. Fei-Fei. What does classifying more than 10,000 image categories tell us? In *European Conference on Computer Vision (ECCV)*, pages 71–84. Springer, 2010.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255. IEEE, 2009.

[17] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. DeCAF: A deep convolutional activation feature for generic visual recognition. In *International Conference on Machine Learning (ICML)*, volume 32, pages 647–655, 2014.

[18] A. Dosovitskiy and T. Brox. Inverting visual representations with convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4829–4837, 2016.

[19] N. Elmqvist, T.-N. Do, H. Goodell, N. Henry, and J.-D. Fekete. Zame: Interactive large-scale graph visualization. In *Visualization Symposium, 2008. PacificVIS'08. IEEE Pacific*, pages 215–222. IEEE, 2008.

[20] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[21] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 5, pages 153–160, 2009.

[22] S. Escalera, X. Baró, J. Gonzalez, M. A. Bautista, M. Madadi, M. Reyes, V. Ponce-López, H. J. Escalante, J. Shotton, and I. Guyon. Chalearn looking at people challenge 2014: Dataset and results. In *Workshop at the European Conference on Computer Vision (ICCV)*, pages 459–473. Springer, 2014.

[23] G. Griffin and P. Perona. Learning and using taxonomies for fast visual categorization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2008.

[24] F. Grün, C. Rupprecht, N. Navab, and F. Tombari. A taxonomy and library for visualizing learned features in convolutional neural networks. In *ICML Workshop on Visualization for Deep Learning*, page 8, 2016.

[25] S. Guattery and G. L. Miller. On the quality of spectral separators. *SIAM Journal on Matrix Analysis and Applications*, 19(3):701–719, 1998.

[26] A. W. Harley. *An Interactive Node-Link Visualization of Convolutional Neural Networks*, pages 867–877. Springer International Publishing, 2015.

[27] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[28] L. Hubert. Some applications of graph theory and related non-metric techniques to problems of approximate seriation: The case of symmetric proximity measures. *British Journal of Mathematical and Statistical Psychology*, 27(2):133–153, 1974.

[29] Y. Ioannou, D. Robertson, D. Zikic, P. Kontschieder, J. Shotton, M. Brown, and A. Criminisi. Decision forests, convolutional networks and the models in-between. *arXiv preprint arXiv:1603.01250*, 2016.

[30] M. Jaderberg, A. Vedaldi, and A. Zisserman. Deep features for text spotting. In *European Conference on Computer Vision (ECCV)*, pages 512–528. Springer, 2014.

[31] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[32] A. J. Joshi, F. Porikli, and N. P. Papanikolopoulos. Scalable active learning for multiclass image classification. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 34(11):2259–2273, 2012.

[33] A. Kapoor, B. Lee, D. Tan, and E. Horvitz. Interactive optimization for steering machine classification. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1343–1352. ACM, 2010.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012.

[35] J. B. Kruskal and J. M. Landwehr. Icicle plots: Better displays for hierarchical clustering. *The American Statistician*, 37(2):162–168, 1983.

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

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

[38] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[39] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu. Towards better analysis of deep convolutional neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):91–100, 2017.

[40] S. Liu, X. Wang, M. Liu, and J. Zhu. Towards better analysis of machine learning models: A visual analytics perspective. *Visual Informatics*, 2017.

[41] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the IEEE international conference on Computer vision (ICCV)*, volume 2, pages 1150–1157. IEEE, 1999.

[42] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5188–5196, 2015.

[43] A. Mahendran and A. Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision*, 120(3):233–255, 2016.

[44] E. Mäkinen and H. Siirtola. The barycenter heuristic and the reorderable matrix. *Informatica (Slovenia)*, 29(3):357–364, 2005.

[45] G. A. Miller. WordNet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[46] V. N. Murthy, V. Singh, T. Chen, R. Manmatha, and D. Comaniciu. Deep decision network for multi-class image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2240–2248, 2016.

[47] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator

networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3387–3395, 2016.

[48] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436, 2015.

[49] A. Nguyen, J. Yosinski, and J. Clune. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. In *ICML Workshop on Visualization for Deep Learning*, 2016.

[50] D. Oelke, H. Janetzko, S. Simon, K. Neuhaus, and D. A. Keim. Visual boosting in pixel-based visualizations. In *Computer Graphics Forum*, volume 30, pages 871–880. Wiley Online Library, 2011.

[51] H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese. Deep metric learning via lifted structured feature embedding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4004–4012, 2016.

[52] K. Patel, N. Bancroft, S. M. Drucker, J. Fogarty, A. J. Ko, and J. Landay. Gestalt: integrated support for implementation and analysis in machine learning. In *Proceedings of the 23nd annual ACM symposium on User Interface Software and Technology (UIST)*, pages 37–46. ACM, 2010.

[53] P. E. Rauber, S. G. Fadel, A. X. Falcao, and A. C. Telea. Visualizing the hidden activity of artificial neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):101–110, 2017.

[54] D. Ren, S. Amershi, B. Lee, J. Suh, and J. D. Williams. Squares: Supporting interactive performance analysis for multiclass classifiers. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):61–70, 2017.

[55] M. T. Ribeiro, S. Singh, and C. Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

[56] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[57] A. Saxena. Convolutional neural networks: an illustration in TensorFlow. *XRDS: Crossroads, The ACM Magazine for Students*, 22(4):56–58, 2016.

[58] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *International Conference on Learning Representations (ICLR) Workshop*, 2014.

[59] D. Smilkov, S. Carter, D. Sculley, F. B. Viegas, and M. Wattenberg. Direct manipulation visualization of deep networks. In *ICML Workshop on Visualization for Deep Learning*, 2016.

[60] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[61] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[62] J. Stasko. Value-driven evaluation of visualizations. In *Proceedings of the Fifth Workshop on Beyond Time and Errors: Novel Evaluation Methods for Visualization (BELIV)*, pages 46–53. ACM, 2014.

[63] H. Strobelt, S. Gehrmann, B. Huber, H. Pfister, and A. M. Rush. Visual analysis of hidden state dynamics in recurrent neural networks. In *Visualization in Data Science Symposium (VDS)*, 2016.

[64] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[65] J. Talbot, B. Lee, A. Kapoor, and D. S. Tan. EnsembleMatrix: interactive visualization to support machine learning with multiple classifiers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1283–1292. ACM, 2009.

[66] S. Tan and K. C. Sim. Towards implicit complexity control using variable-depth deep neural networks for automatic speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5965–5969. IEEE, 2016.

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

[68] S. Van Den Elzen and J. J. van Wijk. BaobabView: Interactive construction and analysis of decision trees. In *2011 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 151–160. IEEE, 2011.

[69] A. Vedaldi and K. Lenc. MatConvNet: Convolutional neural networks for MATLAB. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 689–692. ACM, 2015.

[70] J. Wang, B. Yu, and L. Gasser. Classification visualization with shaded similarity matrix. In *IEEE Visualization*, 2002.

[71] D. Wei, B. Zhou, A. Torrabla, and W. Freeman. Understanding intra-class knowledge inside CNN. *arXiv preprint arXiv:1507.02379*, 2015.

[72] Z. Yan, H. Zhang, R. Piramuthu, V. Jagadeesh, D. DeCoste, W. Di, and Y. Yu. HD-CNN: hierarchical deep convolutional neural networks for large scale visual recognition. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2740–2748, 2015.

[73] L. Yeager, G. Heinrich, J. Mancewicz, and M. Houston. Effective visualizations for training and evaluating deep models. In *ICML Workshop on Visualization for Deep Learning*, 2016.

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

[75] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision (ECCV)*, pages 818–833. Springer, 2014.

[76] Z. Zhang, Y. Chen, and V. Saligrama. Efficient training of very deep neural networks for supervised hashing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1487–1495, 2016.

[77] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Object detectors emerge in deep scene CNNs. In *International Conference on Learning Representations (ICLR)*, 2015.