

DeepEyes

Progressive Visual Analytics for Designing Deep Neural Networks

Pezzotti, Nicola; Holtt, Thomas; van Gemert, Jan; Lelieveldt, Boudewijn; Eisemann, Elmar; Vilanova Bartroli, Anna

DOI

[10.1109/TVCG.2017.2744358](https://doi.org/10.1109/TVCG.2017.2744358)

Publication date

2018

Document Version

Accepted author manuscript

Published in

IEEE Transactions on Visualization and Computer Graphics

Citation (APA)

Pezzotti, N., Holtt, T., van Gemert, J., Lelieveldt, B., Eisemann, E., & Vilanova Bartroli, A. (2018). DeepEyes: Progressive Visual Analytics for Designing Deep Neural Networks. *IEEE Transactions on Visualization and Computer Graphics*, 24(1), 98-108. Article 8019872. <https://doi.org/10.1109/TVCG.2017.2744358>

Important note

To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright

Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy

Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

DeepEyes: Progressive Visual Analytics for Designing Deep Neural Networks

Nicola Pezzotti, Thomas Höllt, Jan van Gemert, Boudewijn P.F. Lelieveldt, Elmar Eiseemann, Anna Vilanova



Fig. 1. **DeepEyes** is a Progressive Visual Analytics system for the analysis of deep neural networks during training. The overview on the training is given by the commonly used loss- and accuracy-curves (a) and the Perplexity Histograms (b) a novel visualization that allows the detection of stable layers. A detailed analysis per layer is performed in three tightly linked visualizations. Degenerated filters are detected in the Activation Heatmap (c), and filter activations are visualized on the Input Map (d). Finally, in the Filter Map (e), relationships among the filters in a layer are visualized.

Abstract—Deep neural networks are now rivaling human accuracy in several pattern recognition problems. Compared to traditional classifiers, where features are handcrafted, neural networks learn increasingly complex features directly from the data. Instead of handcrafting the features, it is now the network architecture that is manually engineered. The network architecture parameters such as the number of layers or the number of filters per layer and their interconnections are essential for good performance. Even though basic design guidelines exist, designing a neural network is an iterative trial-and-error process that takes days or even weeks to perform due to the large datasets used for training. In this paper, we present DeepEyes, a Progressive Visual Analytics system that supports the design of neural networks during training. We present novel visualizations, supporting the identification of layers that learned a stable set of patterns and, therefore, are of interest for a detailed analysis. The system facilitates the identification of problems, such as superfluous filters or layers, and information that is not being captured by the network. We demonstrate the effectiveness of our system through multiple use cases, showing how a trained network can be compressed, reshaped and adapted to different problems.

Index Terms—Progressive visual analytics, deep neural networks, machine learning.

1 INTRODUCTION

In recent years, **Deep Neural Networks** (DNNs) have shown outstanding performance in various problems, like image and speech recognition [24]. DNNs consist of various interconnected **layers**. In each

layer, a number of **filters** detect increasingly complex **patterns**. For example, in networks trained to recognize objects in an image, the first layer generally contains filters that are trained to detect colors and edges. This information is aggregated by other layers to detect complex patterns, e.g., grids or stripes. By using hundreds or thousands of filters in each layer, DNNs allow for more complex patterns to be learned. Only recently the training of large DNNs was made possible by the development of fast parallel hardware, i.e., GPUs, and the creation of large training sets [22].

While the results that DNNs can achieve are impressive, they essentially remain a black box. An increasing research effort is spent on making the visualization and the analysis of these models feasible. While both, the machine learning and the visualization community, invested considerable effort in understanding how a trained network behaves [27, 37, 50], e.g., by showing the patterns learned by the fil-

- N. Pezzotti, T. Höllt, J. van Gemert, B.P.F. Lelieveldt, E. Eiseemann, and A. Vilanova are with the Intelligent Systems department, Delft University of Technology, Delft, the Netherlands.
- B.P.F. Lelieveldt is with the Division of Image Processing, Department of Radiology, Leiden University Medical Center, Leiden, the Netherlands.

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.xxx/TVCG.201x.xxxxxx

ters, little effort has been spent on the creation of tools that support design decisions given the pattern recognition problem at hand. Even though basic design guidelines exist, the process of designing a neural network is an iterative trial-and-error process [2]. For example, experts can change the number of layers or filters per layer but the effect of a modification only becomes obvious after hours, days or weeks, as the network needs to be retrained, a lengthy task given the size of the datasets involved. A visual analytics approach for the analysis of a deep network therefore seems necessary [21]. A recent paradigm, called Progressive Visual Analytics, aims at improving the interaction with complex machine learning algorithms [8, 33, 36, 40]. This interaction is achieved by providing the user with visualizations of the intermediate results while the algorithm evolves, the training of the network in this setting. However, the size of DNNs makes the application of the Progressive Visual Analytics paradigm challenging, requiring the development of visualizations that heavily rely on data aggregation at interactive rates [9, 35, 36, 43].

In this work, we present DeepEyes, a Progressive Visual Analytics system that supports the design of DNNs directly during training. After discussing with machine learning experts that collaborated in the design of DeepEyes, we came to realize that the existing work provides limited feedback on how a DNN can be improved by the designer. To overcome this limitation, we identified the following analytical tasks as critical to make informed design-decisions while the network is trained:

- (T1) **Identification of stable layers** which can be analyzed in more detail, effectively facilitating the detailed analysis while the network is trained
- (T2) **Identification of degenerated filters** that do not contribute to the solution of the problem at hand and, therefore, can be eliminated
- (T3) **Identification of patterns undetected** by the network, which may indicate that more filters or layers are needed
- (T4) **Identification of oversized layers** that contain unused filters and, therefore, can be reduced in size
- (T5) **Identification of unnecessary layers or the need of additional layers**, allowing for the identification of an efficient architecture for the problem at hand

The main contribution of this work is the DeepEyes framework itself. For the first time, DeepEyes integrates mechanisms to tackle all presented tasks to analyze DNNs during training into a single, progressive visual analytics framework. The development of DeepEyes is enabled by a set of further contributions of this paper:

- a new, data-driven analysis model, based on the sampling of sub-regions of the input space, that enables progressive analysis of the DNN during training
- Perplexity Histograms, a novel overview-visualization that allows the identification of stable layers of the DNN for further exploration
- a set of existing visualizations have been extended or adapted for our data-driven approach to allow detailed analysis: Activation Heatmap, Input Map, and Filter Map.

In the next section, we provide the reader with a primer on DNNs, with the essential components to understand our contributions and the related work, presented in Section 3. In Section 4, we present DeepEyes, describing our visualization design based on the insights and support we want to provide to the DNN designer. Furthermore we provide a first example of a DNN for the classification of handwritten digits. Two different use cases are provided in Section 5, while implementation details are given in Section 6.

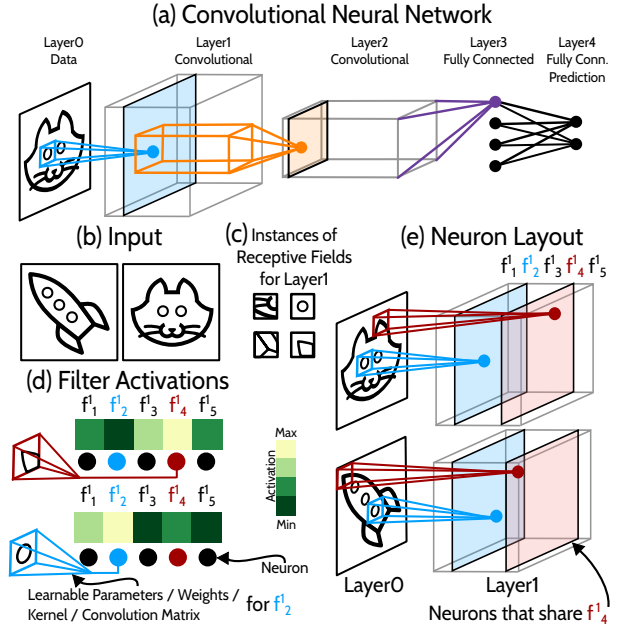


Fig. 2. **Overview of a DNN** (a). Filter functions are computed by neurons in convolutional layers by applying a kernel or convolution matrix on a subsets of the input (b), called **Receptive Field**, whose instances are image patches (c). Filter functions are trained to detect different receptive field instances (d) and they are organized in a 3D grid (e) according to the spatial relationships of the receptive fields they compute.

2 DEEP LEARNING PRIMER

Deep artificial neural networks are trained on a specific pattern recognition problem, such as image classification. The goal is to predict a class of an unseen sample. A training set consists of a set of high-dimensional inputs $\mathbf{x} \in \mathbb{R}^n$ together with an associated vector $\mathbf{y} \in \{0, 1\}^d$ with $\sum_i y_i = 1$, where d is the total number of labels. The only non-zero component indicates the associated label. The goal of a DNN is to predict the label $\tilde{\mathbf{y}} \in [0, 1]^d$ for an unseen input $\tilde{\mathbf{x}} \in \mathbb{R}^n$. The prediction is usually in the form of a discrete probability distribution over the possible labels, hence $\sum_i \tilde{y}_i = 1$.

A DNN consists of a set of layers $\mathcal{L}$. An example of a DNN that comprises five layers, more specifically one data layer, two convolutional layers and two fully-connected layers, is presented in Figure 2a. Independently from the nature of the layer, every layer $l \in \mathcal{L}$ contains a set of **neurons** that computes **filter functions** $f_i^l \in \mathcal{F}^l$, or, more concisely, **filters**. However, exactly the same filter can be computed by many neurons in the same layer. In the example in Figure 2b-e the input consist of images, where each pixel is a dimension in our input space $\mathbb{R}^n$. Filter functions in Layer1 do not take the full dimensionality $\mathbb{R}^n$ as input, but rather a subsets $\mathbb{R}^k \subset \mathbb{R}^n$, the **receptive fields** where k^l represents the size for layer l . For images, these subsets are patches and a few instance of these patches are presented in Figure 2c. Mathematically, a specific receptive field $\delta_r^l \in \Delta^l$ for layer l is a set of indices $\delta_r^l := \{i_j\}_{j=0}^{k^l} \subset \{0 \dots n\}$ that defines a corresponding projection function $\pi(\delta_r^l) : \mathbb{R}^n \rightarrow \mathbb{R}^{k^l}, (x_0, \dots, x_n) \rightarrow (x_{i_0}, \dots, x_{i_{k^l}})$. We now focus on the relationship between filters and neurons given an **instance of a receptive field**, i.e., a specific patch for a specific input image $\mathbf{x}$ identified by the projection function $\pi(\delta_r^l)(\mathbf{x})$. In Figure 2d a heatmap is shown to illustrate the output of filter functions $f_i^l(\pi(\delta_r^l)(\mathbf{x}))$, also called **filter activations**, given specific instances of receptive fields $\pi(\delta_r^l)(\mathbf{x})$. In the first layer, the filter function is usually a weighted sum of the pixel values on the receptive field. These **weights** are the **learnable parameters** that are trained to detect specific patterns in the data. Further, the weights define the filter function and are the same for all neurons computing this filter. In the example, f_4^1 detects , having high filter activation, while f_2^1 detects .

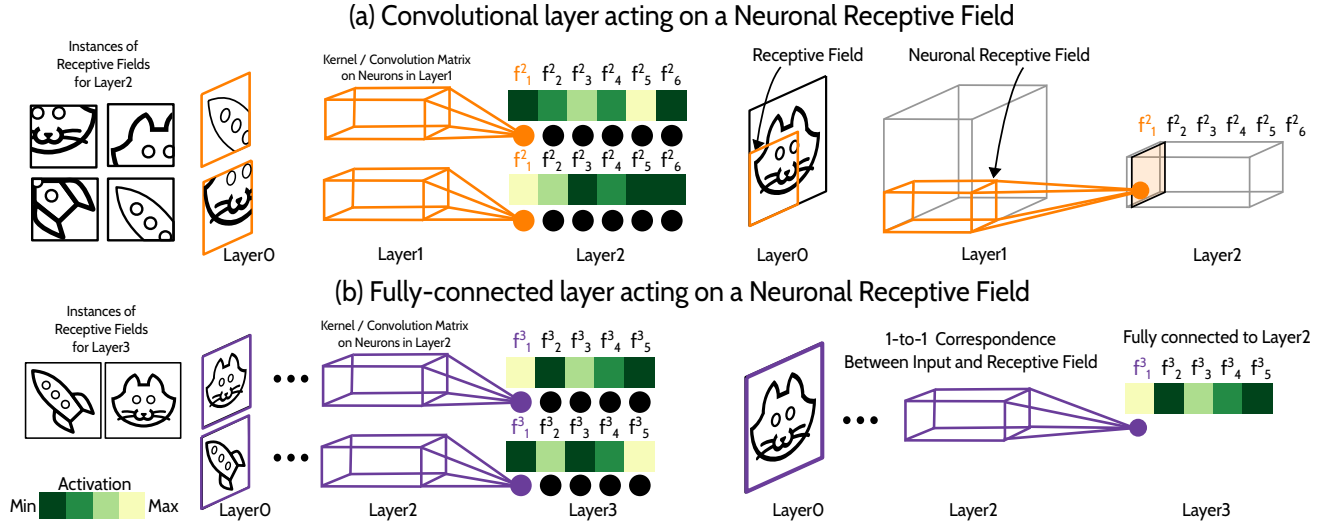


Fig. 3. In deeper layers, filter functions are trained to detect more complex patterns in larger receptive fields. In convolutional layers a subset of the neurons in the previous layer, the neuronal receptive field, is the input to the filter functions rather than the receptive field instance (a). The same description holds for a fully-connected layer, however, it differs from convolutional layers as the receptive field of a neuron corresponds to the complete input and the neuronal receptive field contains all the neurons in the previous layer (b).

Given a single instance of a receptive field, as or in Figure 2d, a **1-to-1 correspondence** exists between filters and neurons (represented as points in Figure 2). However, when the full input is considered, neurons that share the same filter function but process a different location in the image, i.e. receptive field, are organized in a grid-like layout that mimic the input shape. The layout for Layer1 is illustrated in Figure 2e, where neurons that compute the same filter function are placed on planes. By stacking these planes, the resulting layout is a 3D grid of neurons. Filter functions give better information on the detected patterns than single neurons, as they are pattern detectors learned by the layer independently of the position in the input. Note how, in Figure 2e, is detected by the filter f_1^2 which is computed by different neurons, i.e., where eyes and portholes are located.

The same description holds for any layer, as shown in Figure 3a. Here, the receptive fields are larger in deeper layers and the filter functions are trained to detect more complex patterns. For example, is detected by the filter f_1^2 as it has a high activation. The main difference from Layer1 is that, the filter functions are not a direct expression of the dimensions in the receptive field in Layer2. In this layer, the filter functions consider as input a subset of the neurons in the previous layer, whose receptive fields are fully contained in the receptive field for Layer3. We define the region in the 3D grid of neurons in the previous layer as the **neuronal receptive field** of the neuron in the considered layer. The filter activation is obtained by weighting the activation of the neurons in the neuronal receptive field. The neurons in Layer2 are also organized in a 3D grid according to the relationships between the receptive fields and the filters. In Figure 3b, the computation for the fully-connected Layer3 is presented. Similarly to Layer2, Layer3 takes the neuronal receptive field in the previous layer as input. The receptive fields of filters in fully-connected layers correspond to the complete input, hence there is no need for a 3D grid of neurons. For this reason, a 1-to-1 correspondence between filters and neurons exists, meaning that a filter function is computed by just one neuron.

In this section, we provided an overview of the relationships between relevant elements of the DNN. We only briefly introduced the learnable parameters, or weights, involved in the convolutional or fully-connected layers. These parameters are learned by optimization given the training set. In modern architectures many different layers are used to define the filter functions, e.g., max-pooling and normalization layers. The concepts introduced so far hold, as filters are defined as a composition of the operations performed by different types of layers. For the interested reader we refer to LeCun et al. [24] for a more broad overview.

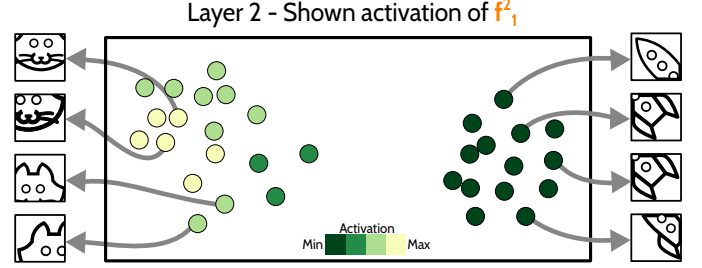


Fig. 4. **DeepEyes approach** to filter analysis. Instances of receptive fields are sampled and embedded in a 2-dimensional space based on the similarity of the activation in the **neuronal receptive-field**. Activation of filters is highlighted in the resulting scatterplot and the instances of the receptive fields are visualized in a linked view.

In this work we rely on the idea that, independently from the chosen layers, input data or receptive field instances are usually interpretable by humans, while abstract weights and relationships between neurons are not [28, 50]. Figure 4 provides an intuition of the central approach that we take in DeepEyes for analyzing what patterns a layer is trained to detect. The user creates a 2-dimensional representation of the instances of receptive fields used in the training. Instances that are perceived as similar by the layer, i.e. have similar activation in the neuronal receptive field, are close in the 2-dimensional visualization. Specific filter activation is then highlighted on demand, allowing for the understanding of the response of the filter to the input. For example, in Figure 4 we see that a separation of the receptive fields according to the input label, i.e., cat and rocket which are visualized in linked views, is available and the visualized activation of filter f_1^2 is strongly correlated with the cat label. Note that, despite the focus on the analysis of DNNs for image classification, the proposed approach is general as it focuses on filter activations and can be extended to different types of data, e.g., text or video, if appropriate linked views are used [20].

3 RELATED WORK

Existing visualization techniques for DNNs can be divided in **weight-centric**, **dataset-centric** and **filter-centric** techniques.

Weight-centric techniques aim at visualizing the relationships between filters in different layers through the visualization of the learnable parameters, or weights, introduced in Section 2. A straightforward visualization for the weights are node-link diagrams [38], similar to the one presented in Figure 2a for the connection of Layer3 to Layer4. Here

weights can be encoded in the edges, e.g., as line thickness. However, this approach does not scale to state-of-the-art networks that comprise millions of connections, limiting the application of weight-centric techniques mainly to didactic purposes [12]. To reduce the clutter generated on such networks, Liu et al. recently proposed a biclustering-based edge bundling approach [27] that aggregates neurons and bundles edges. Neurons are aggregated if they are activated by data that share the same label, while edges are bundled if they have similar and large absolute weights. However, in DNNs, neurons are trained to separate labels only in the last layers, therefore this clustering is not informative in early layers. For example, in Figure 2e the filter f_2^1 activates both on  and  an information that does not reveal the pattern  that the filter is trained to detect. Moreover, while the system allows a real-time exploration of the network, the creation of the visualizations requires hours of preprocessing, making the analysis of the network during training unfeasible. DeepEyes does not provide a weight-based visualization. After discussing with the machine learning experts involved in the development, we realized that it is more important to focus on the analysis of filters as pattern detectors, rather than on individual neurons and their connections [50].

The goal of **dataset-centric** techniques is to provide a holistic view on how the input data are processed by the network rather than providing a solution to the previously introduced tasks (T1,T2,T3,T4,T5). The training- or the test-set is processed by the DNN and the activations of neurons in the last fully-connected layer are collected as high-dimensional feature vectors. Using non-linear dimensionality-reduction techniques, the dimensionality of the feature vectors is reduced to two dimensions and visualized in a scatterplot [1, 19, 31]. Two data points that are close in the 2-dimensional space are also close in the feature space, meaning that the network perceives them as similar. Recently, Rauber et al. [37] showed the evolution of this representation during training, while Pezzotti et al. [35] showed that hierarchical information is learnt by DNNs even though this information is not encoded in the training set. While these techniques provide insight on how the network reacts as a whole, they are limited to the analysis of the last fully-connected layer of the network. The only work in the analysis of hidden layers, i.e., not the input- or last-layer, is from Rauber et al. [37] where 2D embeddings are generated for hidden and fully-connected layers. This approach suffers from a severe limitation, being restricted to the analysis of layers where a **1-to-1 correspondence** between neurons and filter functions exists, i.e., fully-connected layers. We extend their work such that it can be used for convolutional layers which are the most widely used layers in modern day architectures [14, 22, 24, 39, 41].

Filter-centric techniques aim at giving an intuition on the pattern that a filter f_i^l is trained to detect. A straightforward approach presented by Girshick et al. [11] identifies for each filter f_i^l the instance of a receptive field $\pi(\delta_r^l)(\mathbf{x})$ with the highest activation $f_i^l(\pi(\delta_r^l)(\mathbf{x}))$. The instance of a receptive field $\pi(\delta_r^l)(\mathbf{x})$ is then presented to the user, e.g., as an image patch. A more complex approach aims at inverting the filter function f_i^l by defining $(f_i^l)^{-1}$, allowing for the reconstruction of the receptive field $\pi(\delta_r^l)(\mathbf{x})$ that produces the highest activation for f_i^l [7, 28, 34, 49, 50]. However, the explicit definition of $(f_i^l)^{-1}$ is not possible and it is approximated using deconvolutional neural networks [50]. This approach generates images that can give the intuition of the patterns detected by the filters, as demonstrated by Google’s Deep Dream [32], and can be further extended for different tasks, such as style transfer [10]. However, according to the feedback provided by machine learning experts, the reconstructed receptive fields can be difficult to interpret for complex patterns, i.e., for late-layers in the network, and do not allow for a reasoning on architectural decisions (T4,T5). Moreover, the reconstruction of the receptive field is a minimization process itself that is time consuming, requires complex regularization techniques and may produce misleading results [28, 49]. Filter-centric techniques are powerful tools but are generally limited to the analysis of a single and well-behaving filter, making their application for the analysis of a neural network during training difficult. DeepEyes includes novel filter-centric techniques for the identification of badly trained filters (T2) and provides a holistic view on filter activations

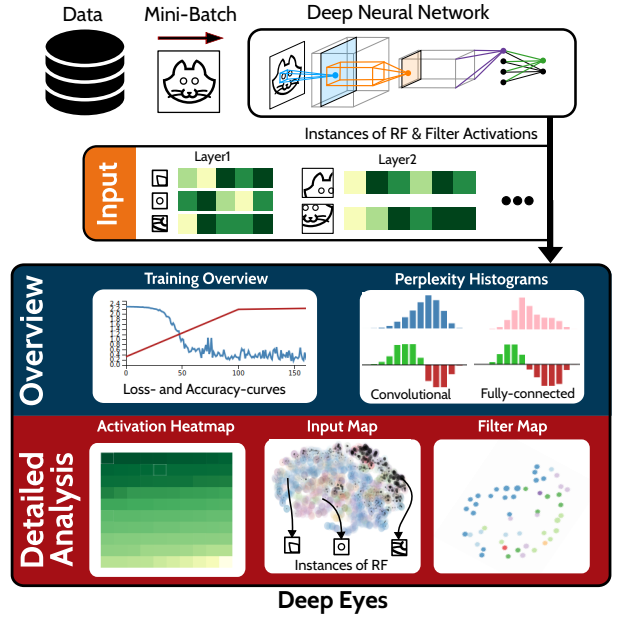


Fig. 5. **Overview** of the DeepEyes system. The network training overview provided by the loss- and accuracy-curves is integrated with the Perplexity Histograms that allow for the identification of stable layers in the network (blue background). The user focuses on stable layers that are analyzed in detail with three tightly linked visualizations, namely the Activation Heatmap, the Input Map and the Filter Map (red background).

given instances of receptive fields.

Finally, a recently proposed filter technique visualizes relationships between filters, i.e., how similarly they activate on the input and which label they are most strongly associated with [37]. Filters are represented as points and placed in a scatterplot by a multi-dimensional scaling algorithm [4]. Filter-to-label association is then highlighted by coloring every point with the color of the most correlated label. While this filter-centric technique allows for newer insights (T3), it has two limitations that we overcome with a novel approach. First it requires the analysis of the complete dataset and, second, it cannot be applied to convolutional layers.

4 DEEP EYES

In this section, we introduce DeepEyes, a Progressive Visual Analytics system for the analysis of DNNs during training that combines novel data- and filter-centric visualization techniques. We start with an overview of DeepEyes in relation to these tasks in Section 4.1. A detailed description is provided in Sections 4.2 to 4.5. As a running example throughout this section we use the MNIST dataset [25] which consists of a training set of 60K images and 10K validation images. We train with the Stochastic Gradient Descent [26] the MNIST-Network that is provided in Caffe [18], a commonly used deep learning library which provides the deep-learning framework for DeepEyes. The network comprises two convolutional layers, with 20 and 50 filters respectively, and two fully connected layers with 500 and 10 filters respectively. Note that we use the MNIST-Network as proof of concept of our implementation and, for the sake of reproducibility, we use the architecture and training parameters provided by Caffe even if they do not achieve state-of-the-art results in classification performance.

4.1 Overview

Figure 5 shows an overview of our system. A DNN is trained by computing the filter activations on subsets of the training set, called *mini-batches*. The loss function, which measures how close the prediction matches the ground truth, is computed and the error is back propagated through the network. The learnable parameters of the network are then updated in the opposite direction of the gradient of the loss function [24, 26]. DeepEyes builds on the notion that the un-

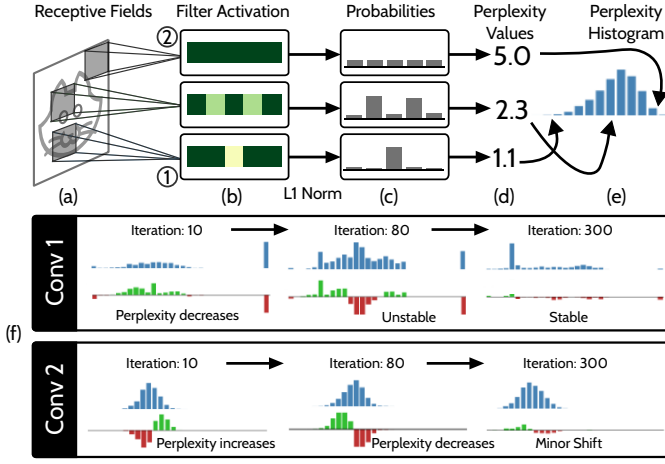


Fig. 6. **Perplexity Histograms** and their creation. Receptive fields are sampled for every input data (a). The activation of the neurons that correspond to the receptive fields are collected, i.e., the receptive field’s depth column (b). The depth columns are transformed in probability vectors (c) whose perplexity is computed (d) and used to populate the perplexity histogram (e). (f) shows the evolution of the perplexity histograms for the layer *Conv1* and *Conv2* in the MNIST-Network. Changes in the histogram over time are presented in a second histogram, highlighting the changes with red and green bars, for decreasing and increasing numbers, respectively.

understanding of the relationships between instances of receptive fields $\pi(\delta_r^l)(\mathbf{x})$, which can be visualized and understood by humans, and the activation of filter functions $f_i^l(\pi(\delta_r^l)(\mathbf{x}))$ is crucial for understanding the patterns detected by the network in every layer.

For every mini-batch that is used to train the network, we sample instances of the receptive fields for every layer and the corresponding filter activations. Unless the user specifies otherwise, we sample a number of instances that grants a coverage of at least 50% of each input. This information is used to create a continuously-updated dashboard that provides insights into which patterns are detected by the layers in the DNN. In the Training Overview, loss and accuracy over time are presented. We complement this standard visualization, with a novel visualization, the **Perplexity Histograms** (Sec. 4.2), which allows for identifying when a layer learned to detect a stable set of patterns (T1). The detailed analysis of stable layers is performed using three tightly-connected visualizations, highlighted in red in Figure 5. The **Activation Heatmap** (Sec. 4.3) allows for the identification of degenerated filters (T2), while the **Input Map** (Sec 4.4) shows the relation of filter activations on instances of receptive fields for a given layer (T3). Finally, the **Filter Map** shows how similar the filters activate on the input. Interaction with the Input- and Filter-Map support the identification of oversized and unnecessary layers (T4,T5).

4.2 Perplexity histograms as layer overview

The evolution of the loss- and accuracy-curve presented in the Training Overview, is the de-facto standard way to visualize the evolution of the network during training. However, this visualization only provides information about the global trend of the training and fails to give a per-layer visualization of the changes. Given the size of the network, it is important to guide the user [5] towards layers that can be analyzed in detail while the training progresses, i.e., layers that learned a stable set of patterns (T1). Our solution is based on the notion that every filter in a layer is trained to identify a certain pattern for a specific receptive-field size [50]. Therefore, we propose to treat every layer as a classifier designed to detect patterns, which are unknown at this moment, and we analyze its performance over time. More specifically, we want to know if the classifiers’ ability to detect patterns is stable, increasing, or decreasing during training. If it is stable, it means that the layer learned what it was able to learn. If it decreases, the knowledge that this layer

provides to the network is decreasing, and inversely when increasing.

We encode the layer stability as follows. For every input in a mini-batch, we randomly sample a number of instances of receptive fields (Figure 6a) and the corresponding filter activations (Figure 6b). We transform the activations in a probability vector $\mathbf{p} \in \mathbb{R}^{|\mathcal{F}^l|}$, where $|\mathcal{F}^l|$ is the number of filters in the layer l , by applying a L1-normalization (Figure 6c). Then, we compute for every receptive field instance the value of **perplexity** of the corresponding probability vector $\mathbf{p}$ (Figure 6d). The perplexity, a concept from information theory [23] that, in this setting, measures how well a pattern is detected by the layer under consideration. The perplexity of the distribution $\mathbf{p}$ is equal to 1 if only one filter is activated by the instance of the receptive field. An example is given by the activations marked with ① in Figure 6a. On the contrary, the perplexity of $\mathbf{p}$ is equal to the number of filters $|\mathcal{F}^l|$, if the activations of every filter are equal, as shown for the activations marked with ② in Figure 6a. The Perplexity Histogram accumulates the sampled input based on the corresponding perplexity value in the range $[1, |\mathcal{F}^l|]$ for every layer l (Figure 6e). Changes in the histograms during training are visualized in a second histogram. Here, green bars represent an increase in the corresponding bin, while red bars represent a decrease (Figure 6f). A shift to the left in the histogram, i.e., to lower values of perplexity, means that the ability to detect patterns for this layer is increasing and vice-versa. Note that, because the computed perplexity assumes continuous value, the number of bins in the histogram has no link with the number of filters in the layer. We provide a default of 30 bins, that we empirically found to be visually pleasing and does not hamper the ability to detect shifts in the histograms.

Figure 6f shows the evolution of the perplexity histograms of the convolutional layers for the MNIST-Network, i.e., *Conv1* and *Conv2*. After 10 iterations a shift to low values of perplexity in the first layer is visible. The peak in the histogram for *Conv1* corresponds to patches that are not detected by any filter (T3). While the histogram of the first layer is shifting to the left, i.e., decreasing the perplexity, the histogram of the second layer is shifting to the right. This behavior shows that the second layer is responding to a change in the filter functions computed in the first layer by becoming less specific, i.e., increasing the resulting perplexity. The histograms are updated at every iteration and the user monitors the stability of the layers. Figure 6f shows how the histograms evolved after 80 iterations. Compared to iteration 10, the first layer is still unstable and the second layer is now more specific. After 300 iterations, the first layer is stable, while the second layer shows a shift to lower values of perplexity. This shift is limited, showing that the layer is currently affected by minor changes, allowing the user to start its detailed analysis.

4.3 Activation Heatmap

Guided by the Perplexity Histograms, the user focuses on the detailed analysis of a stable layer starting from the Activation Heatmap, where every filter is visualized as a cell in a heatmap visualization (Figure 7a). The Activation Heatmap is designed for the quick identification of degenerated filters (T2). We aim at the identification of *dead* filters, i.e., filters that are not activating to any instance of a receptive field, and filters that are activating to all instances. In both cases these filters are not providing any additional information to the network. These filters are detected in a heatmap visualization that shows the maximum- and the frequency-of-activation.

For creating the heatmaps, we randomly sample instances of receptive fields and we compute the maximum activation μ_i^l for every filter f_i^l in layer l

$$\mu_i^l = \max(f_i^l(\pi(\delta_r^l)(\mathbf{x}))),$$

where $\pi(\delta_r^l)(\mathbf{x})$ is the sampled instance of the receptive field. For each filter f_i^l , the corresponding μ_i^l is visualized in the heatmap in the range $[0, \max(\mu_i^l, \forall i)]$. We use a similar approach for the identification of filters that have high activation on every input. For every filter, we keep track of how frequently they activate on the sampled data, and we display these frequencies in the heatmap. We consider a filter to be active on a given patch if its activation is greater than a percentage β of

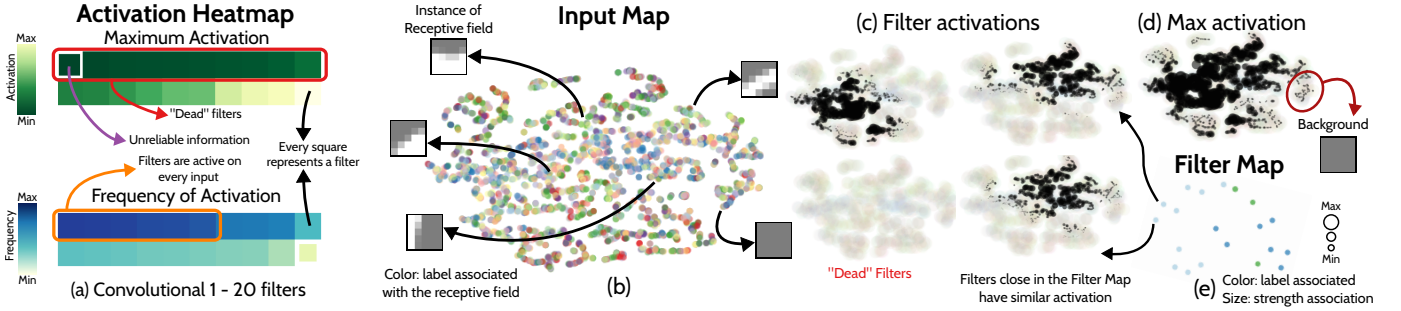


Fig. 7. **Detailed analysis** performed in DeepEyes. Degenerated filters are detected in the Activation Heatmap (a). The Input Map (b) shows a representation of the input space of a specific layer. By brushing on the Input Map receptive fields are visualized in linked views (insets in (b)). Specific filter activations (c) or the maximum activation of every filter (d) are visualized on the Input Map. The Filter Map (e) allows for the understanding of the relationships between filters that are further investigated in the Input Map. Specific filters are selected by clicking on the activation heatmap or by brushing on the Filter Map.

the maximum activation $\max(\mu_i^l, \forall i)$, where a default value of $\beta = 0.5$ is used. The user can choose if the maximum- or the frequency-of-activation is visualized in the heatmap and we distinguish between the two by using two different color scales. A green-to-yellow color scale is used for the maximum activation, while a yellow-to-blue color scale is used for the frequency of activation [13]. At this level of detail, we are interested in giving an intuition of the response of the layer as a whole, hence we provide the option to keep the filters sorted according to the currently visualized information. Because the learnable parameters are changing during training, visualizing the maximum activation for a filter may be misleading. For example, a filter that was active in the early phase of training can “die” in later steps [24]. Therefore, we compute a measure for the reliability of the information contained in the heatmap. We keep track of the last iteration where a filter f_i^l reached an activation higher than a percentage θ of its maximum activation μ_i^l , where $\theta = 0.8$ by default. We visually encode the distance between the current iteration and the last one that reached the maximum activation threshold θ as the size of the cell that we draw in the heatmap [15] and we allow the reinitialization of the computed maximum in a layer.

An example of the proposed visualization is presented in Figure 7a. The maximum activation of the filters in the first convolutional layer of the MNIST-Network after 100 iterations is presented. Ten filters, highlighted in red, out of 20 have a very low activation and do not provide additional information (T2). The smaller size of the cell in the heatmap for the filter identified by a purple arrow means that the maximum activation visualized is not reached in several iterations, leading to the conclusion that at the current iteration its activation is even lower. By visualizing the frequency of activation the user identifies several filters, here highlighted in orange, that have high activation on every input (T2). These insights lead to the conclusion the layer is oversized given the problem at hand (T4) and can be removed by the user before continuing the training, making it faster and the final network smaller. Our visual encoding is scalable in the number of visualized filters. One of the layers with most filters in state-of-the-art architectures is the last fully-connected layer in the AlexNet network [22], consisting of 4096 filters. If every filter is encoded, using a 5x5 rectangle, the heatmap results in an image of 320x320 pixels, that easily fits into our user interface.

4.4 Input Map

The Input Map is a cornerstone of DeepEyes. It provides the tools to solve several analytical tasks (T2, T3, T4, T5) and is based on the idea presented in Figure 4. The map is generated upon user’s request when a stable layer is identified. An example is given in Figure 7b where the first convolutional layer of the MNIST-Network is analyzed in detail. Instances of receptive fields are visualized as points in a scatterplot and colored according to the label of the input they are obtained from. Two instances are close in the scatterplot if they have similar activation for the neurons within the neuronal receptive field and, therefore, are similar input for the current layer (see Section 2). The layout is obtained by reducing the dimensionality of the activation of neurons in

the neuronal receptive field to 2 dimensions, while preserving neighborhood relationships [35]. By brushing on the scatterplot, the user selects instances of receptive fields of interest that are visualized in a linked view, here abstracted as arrows pointing to image patches. The mix of colors corresponding to the input labels indicates that a separation between the classes is not possible at this level (T5), also showing that a clustering of the neurons based on labels as proposed by Liu et al. [27] is not meaningful for early-layers.

The activation of a user-selected filter is visualized on top of the Input Map, as shown in Figure 7c where four filter activations are shown. We keep the Input Map in the background as a reference, drawing the data points as larger and semi-transparent circles. On top, we draw a new set of semi-transparent black circles, whose size is encoding the intensity of the filter activation on the corresponding input. The user can switch between the two visualization modes, allowing to reason on where the activations are localized in the Input Map, therefore giving a detailed understanding of which input is detected by a filter. For example, we can validate the insights previously obtained through the Activation Heatmap. By clicking on a cell in the heatmap, the corresponding filter activation is visualized in the Input Map, showing that the dead filters are not activating on any input (T2). Moreover, single filters are activating on large portions of the input. Together with the presence of many dead filters, this signals that the current layer contains more filters than needed (T4). By visualizing the maximum activation of the filters on each data point, as presented in Figure 7d, we allow for the identification of data that are scarcely or not at all detected by the network. In the example, the outer region of the Input Map contains points that do not produce a strong activation (T3). The inspection of the instances of the corresponding receptive fields reveals that they correspond to background patches and, therefore, are not informative for the problem at hand.

The Input Map is a dataset-centric technique (see Section 3), whose improvements over the state-of-the-art are twofold. First, it is built by sampling instances of receptive fields, allowing for the creation of a dataset-centric visualization even for convolutional layers. Second, differently from existing techniques that focus on the activation of the filters in the current layer, the Input Map reduces the dimensionality based on the activations of the filters in the neuronal receptive field rather than the activation of filters in the layer under analysis. This feature allows for the analysis of the relationship between input and output of a layer, an approach that was not possible before. While these two features allow for new insights, they pose computational challenges in the creation of the 2-dimensional layout in the interactive system. Tens of thousands of receptive field instances are sampled during training and ought to be placed in the Input Map. Further, the dimensionality of the feature vector considered is higher than in existing techniques as we do not just consider the activations in the current layer but the whole neuronal receptive field. We considered several dimensionality-reduction techniques for the generation of the scatterplot [45]. The t-distributed Stochastic Neighbor Embedding (tSNE) algorithm is often used [44] in dataset-centric techniques. However,

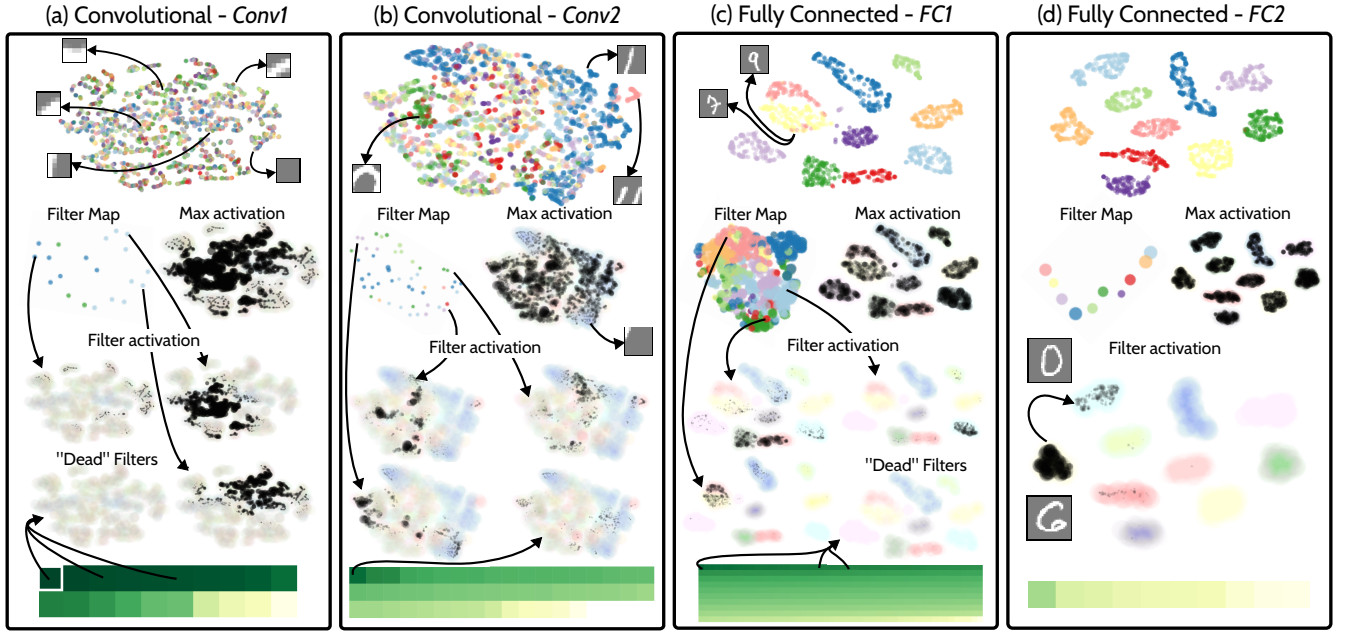


Fig. 8. **Analysis of the MNIST network.** For each layer the Input- and Filter-Maps are presented alongside their corresponding Activation Heatmaps. We highlight activations for different filters in the different layers. A detailed description of the conclusions, drawn from these visualizations is presented in Section 4.6.

as reported by Rauber et al. [37] for their proposed approach, several dozens of minutes are required for the creation of embeddings containing 70K points described by 50 dimensions, limiting its application in a Progressive Visual Analytics system like DeepEyes. Therefore we use the recently-developed Hierarchical Stochastic Neighbor Embedding (HSNE) [35], as it creates visual representations of tens of thousands of data points, described by several thousand dimensions, in seconds. HSNE enables the analysis of such large data in an interactive system by building a hierarchical representation of the data and by generating Input Maps with only a few hundreds data points sampled from the input data. The exploration of the complete dataset is then performed with a *filter* and *drill-in* paradigm. We refer to Pezzotti et al. [35] for further details.

4.5 Filter Map

The Filter Map provides a view on how similarly filters in a layer respond to the input as a whole. We visualize the filters as points in a scatterplot. Filters with a similar activation pattern are placed closer in the scatterplot (Figure 7e). If many filters activate in the same way on the input it is an indication that the layer contains too many filters (T4). Here, we are interested in visualizing the relationships between filters and labels $\mathbf{y}$. Hence, points are colored according to the training label that activates a filter the most, while the size of the point shows how strongly the filter is correlated to that label. We choose this encoding for the sake of simplicity, but different visual encodings can be used, e.g., by encoding the correlation with color brightness or saturation [6, 37]. The presence of a cluster composed by large and similarly colored points in the Filter Map is an indication that a classification can be performed at this stage (T5). To the best of our knowledge, the only existing work in this direction is from Rauber et al. [37]. In their work, the Pearson correlation between filter activations is computed and the filters are visualized using a multi-dimensional scaling algorithm. This approach requires the receptive field of the analyzed filters to cover the complete input and it cannot be used for the analysis of convolutional layers, a severe limitation if state-of-the-art architectures ought to be analyzed (see Section 2).

We propose to overcome this limitations by computing similarities in a progressive way, using instances of receptive fields instead of the complete input. The similarity between two filters is computed as a weighted Jaccard similarity [17]. This gives a measure of common amount of activation divided by the maximum activation of both filters.

If the filters activate equally for the same instances of receptive fields the value will be 1. The more they differ the smaller the similarity will be. For two filters i and j on layer l , their similarity $\phi_{i,j}$ is computed as:

$$\phi_{i,j} = \frac{\sum_{\mathbf{x}} \min(f_i^l(\pi(\delta_r^l)(\mathbf{x})), f_j^l(\pi(\delta_r^l)(\mathbf{x})))}{\sum_{\mathbf{x}} \max(f_i^l(\pi(\delta_r^l)(\mathbf{x})), f_j^l(\pi(\delta_r^l)(\mathbf{x})))}, \quad (1)$$

where $f_i^l(\pi(\delta_r^l)(\mathbf{x}))$ is the activation of the filter f_i^l , given the sampled receptive field for input $\mathbf{x}$. The similarities are updated for every training iteration and, when requested by the user, the filters are embedded in a 2D space with tSNE [44]. In Figure 7e, the Filter Map for the first layer of the MNIST-Network is presented. By brushing on the scatterplot the user selects filters whose activation is then visualized in the Input Map. In the example of Figure 7, it can be seen that two filters that are close in the Filter Map (e) also have a similar activation pattern on the input (c). We also keep track of which label is most associated with a filter. For each filter f_i^l , we compute the vector $\mathbf{t}_i^l \in \mathbb{R}^d$, where d is the number of labels in the dataset. It contains the cumulative activation f_i^l on the sampled receptive fields of instances of objects belonging to the same label:

$$\mathbf{t}_i^l(\arg\max(\mathbf{y})) = \sum_{\mathbf{x}} f_i^l(\pi(\delta_r^l)(\mathbf{x})), \quad (2)$$

where $\mathbf{x}$ is an input with associated label vector $\mathbf{y}$. For every filter f_i^l , the corresponding point in the Filter Map is drawn with the color associated with the label $\arg\max(\mathbf{t}_i^l)$. The point size in the Filter Map encodes the strength of the association with a label. This association is computed as the perplexity of the probabilities, obtained by normalizing the vector $\mathbf{t}_i^l$ with L1-norm (see Section 4.2). The points size encodes the inverse value of the perplexity, where a low value of perplexity means a strong association with the label. Filters in Figure 7e are small in size, showing a low association with the corresponding label, i.e. a large value of perplexity. Also, not all the label colors present in Figure 7b are represented in the Filter Map, showing that filters in this layer are not specialized to perform a proper classification.

4.6 From insights to network design

Here, we illustrate how insights obtained in DeepEyes support network design decisions. Figure 8 shows the analysis of the MNIST-Network

introduced in Section 4. Driven by the stability of the perplexity histograms, the user is guided to the detailed analysis of layers whose filters are stable. *Conv1* is analyzed first, then *Conv2*, *FC1* and finally *FC2*. In the Input Map of *Conv1*, a separation of the labels with respect to the input is not visible, since all label colors are mixed in the scatterplot (Figure 8a). Further, filters are active on large regions of the Input Map, see filter activations in Figure 8a for the selected filter in the filter map. Many dead filters are identified (T2) by selecting filters with low maximum activation in the Activation Heatmap (Figure 8a). The layer is oversized (T4) as overly-redundant or non-existent patterns are learnt by the filters. *Conv2* is analyzed next. Here data points in the Input Map start to cluster according to the labels (Figure 8b). Notice that the shown instances of the receptive field are larger than for *Conv1*, as *Conv2* processes a larger region of the input images. Differently from the previous layer, filter activations are localized in the Input Map, leading to the conclusion that more filters are needed in *Conv2* than in *Conv1*. Similarly as for Figure 7d, points with low maximum activation in Figure 8b correspond to background patches (T3).

In *FC1* (Figure 8c), inputs cluster in the Input Map according to the associated label. The visualization of the Maximum Activation in Figure 8c shows that every data point is activating at least one filter in the current layer, hence every input is identified by the network at this level (T3). Before we can conclude that a classification is feasible at this stage (T5), the Filter Map is analyzed. In the Filter Map, we see that the filters form visual clusters that are associated with labels. However, there is no visible red cluster, associated with the label “digit-5”. The activation of a “digit-5” associated filter is visualized on the Input Map, showing a strong activation also on points in green, i.e., “digit-3”. This insight shows that a perfect separation is not possible in this layer, and that the second fully-connected layer is needed (T5). The presence of duplicated filters and dead filters, as in *FC1*, shows that this layer is oversized and fewer filters can be used (T4).

Finally, in the last layer, which performs the prediction (Figure 8d), every filter is colored with colors of different labels, showing that a correlation between filter and label exists and the network is correctly classifying the input. By showing the activation of the filters on the Input Map, the user also gets an intuition of which labels are confused by the network, e.g., points that correspond to the “digit-0” and “digit-6”, as shown in the filter activation in Figure 8d. Based on the insights obtained from DeepEyes, we modified the network reducing the first convolutional layer from 20 to 10 filters, and the first fully-connected layer from 500 to 100. This reduction allows for a smaller network which is faster to be trained and makes predictions without any visible loss in the accuracy of the classification that is stable for both architectures at 98.2% after 2000 iterations. Note that for the sake of reproducibility we used the parameters defined by Caffe in the “lenet_train_test_prototxt” training protocol.

5 TEST CASES

In this section, we provide further examples of analysis performed with DeepEyes. In recent years a great number of different architectures have been presented. For our test cases we decided to focus on widely used architectures derived from AlexNet [22] that are often modified and adapted to solve different problems, a setting in which the insights provided by DeepEyes are greatly needed. AlexNet [22] consists of 5 convolutional layers, with 96-256-384-384-256 filters, and 3 fully-connected layers, with 4096-4096-1000 filters, leading to more than 16 million trainable parameters. Note that AlexNet is among the largest neural networks in terms of computed filter functions, where a trend in reducing the number of filters exists [14, 16]. This analysis demonstrates the scalability of our progressive system in a general setting. In the first test case, we show how DeepEyes allows for a better understanding of the fine-tuning of DNNs, while in the second test case, we show how a better architecture for the medical imaging domain is derived from insights obtained through DeepEyes.

5.1 Fine tuning of a deep neural network

Training a large DNN from scratch requires a very large training set, computational power, and time. To overcome this limitation, a common

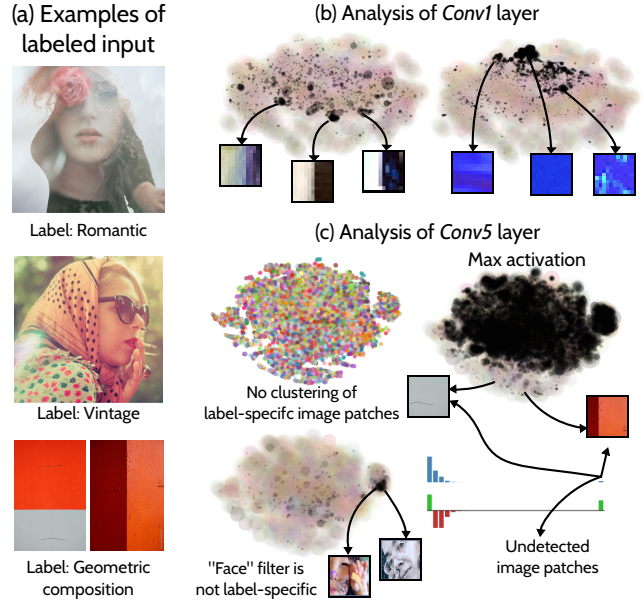


Fig. 9. **Fine tuning** of a pretrained neural network. Deep eyes allows for the identification of layers that do not need retraining, e.g. *Conv1*. Unrecognized input data are highlighted in the Perplexity Histograms and in the Maximum Activation visualization of the Input Map, here highlighting data that is labeled as *Geometric Compositions* which are not recognized by the original network. Furthermore, a filter trained to detect faces is not discriminative given the *Romantic* and *Vintage* labels.

approach is to fine-tune an already trained network for a different problem [3]. The rationale behind this approach is that low-level filters, like color- and edge-detectors, could be reused. To which degree filters can be reused is crucial but not clear a-priori [48]. In this test case, we show how DeepEyes helps in the identification of which layers contain useful filters that can be reused and filters that are not needed and must be retrained. We used the fine-tuning example provided in Caffe, where AlexNet, which was trained for image-classification, is retrained for image-style recognition [18]. In this example, the prediction layer of the network is changed from 1000 filters, used to detect 1000 objects, to 20 filters that are retrained to detect 20 styles of images, e.g. “Romantic”, “Vintage” or “Geometric Composition” (Figure 9a). The network requires 100.000 iterations and more than 7 hours to be retrained with a K40 GPU and achieves an accuracy on the test set of 34.5% [18].

The hypothesis that color and edge detectors are useful filters for the problem at hand is confirmed in the first convolutional layer, i.e., *Conv1* in Figure 9b, as they present a localized and consistent activation pattern, e.g., blue- and vertical-edge-detectors are found. While the first layer is stable, the Perplexity Histogram of the fifth convolutional layers, i.e., *Conv5*, shows that an increasingly large number of input patches are not activating any filter, hinting at a problem in the filter functions for this layer. The detailed analysis of *Conv5* shown in Figure 9c reveals that data labeled as “Geometric Composition” are in the region of the Input Map that is hardly activating any filters (max activation in Figure 9c). Images labeled as “Geometric Composition”, i.e., with large and uniform color surface, were not included in the “image-classification” training set, therefore the network has not learnt useful filters for discriminating such images. Another interesting insight is obtained by visualizing the activation of other filters on the Input Map. For example, a filter that detects human faces is found, see Figure 9c. While this filter is useful for the image-classification problem, it is not discriminative for style-recognition because human faces are associated with many different styles (Figure 9a). This insight shows that the analyzed layer needs to learn new patterns from the input. The fine-tuning of a network or, in general, the reusability of the learned filters, is an active research topic under the name of transfer learning [48]. Insights obtained from DeepEyes can help to improve the fine-tuning of networks by placing the user in the loop.

5.2 Mitotic figures detection

We present a different test case from the application of DNNs in the medical imaging domain. In this context, DNNs developed by the machine learning community are applied to different recognition problems. DeepEyes helps in filling the expertise gap, by providing insight on how the network behaves given the problem at hand. The number of nuclei separations in tumor tissue is a measurement for tumor aggressiveness. In radiotherapy treatment planning, histological images of tumor tissue are analyzed by pathologists. Nuclei separations, also known as mitotic figures, are counted. Examples of images with “mitotic figure” label are presented in Figure 10a, together with images labeled as “negative”. The counting of mitotic figures helps in deciding the dose of radiation used to treat a tumor, leading to a more personalized treatment. However, it is a tedious task and DNNs have been recently proposed to automatize the process. In this test case, we analyze the DNN developed by Veta et al. [46] that is trained on the AMIDA dataset [47] to detect mitotic figures in histological images. The network comprises 4 convolutional layers with 8,16,16 and 32 filters respectively, and 2 fully-connected layers, containing 100 and 2 filters respectively.

After a few training iterations, the first layer stabilizes and is analyzed in detail. Figure 10b shows the detailed analysis of the first convolutional layer after 40 iterations. The Input Map shows a cluster of red points, corresponding to instances of the receptive fields sampled from images labeled as mitotic figures. By visualizing the activation of the filters we see that filters are trained to detect dark regions versus bright regions, as they are an important feature at this level. Similar Input Maps are obtained in the other convolutional layers, where the patches processed by the layers are larger.

An interesting observation is made in the first fully-connected layer of the network. The Input Map and the Filter Map for this layer are presented in Figure 10c. A separation of the labeled input is visible

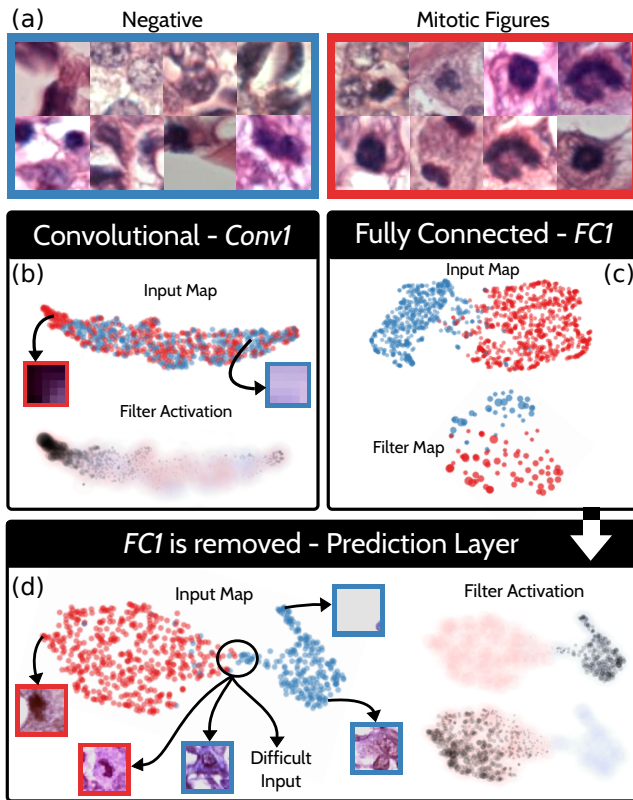


Fig. 10. **Mitotic Figures** detection. A DNN is trained to detect mitotic figures in histological images (a). Filters in the first convolutional layer *Conv1* are highly associated with mitotic figures (b). Labeled data are separated in the Input Map of the first fully-connected layer *FC1* (c). After removing *FC1* the prediction layer (d) still shows very strong separation, indicating that *FC1* is indeed not needed for classification.

in the Input Map, showing that the classification is feasible at this level. This is confirmed by the fact that filters are divided in the Filter Map according to the most strongly associated label. Thus, another layer, as is present in the network, is not needed in order to perform a prediction on the problem at hand (**T5**). Therefore, we change the design by dropping the fully-connected layer and by connecting the prediction layer directly to the last convolutional layer. The analysis of the prediction layer after retraining is provided in Figure 10d. The new network reaches an accuracy of 95.9% on the test set, which is identical to the accuracy obtained with the previous architecture, while it is much faster to compute a prediction.

We contacted Veta et al. [46], presenting DeepEyes and providing our findings. They informed us that they had come to the same conclusions after several blind modifications of their network, commenting that a system like DeepEyes is beneficial in the definition of networks for a specific medical imaging problem. Furthermore, they showed it particular interest in visualizing the instances of the receptive fields and the corresponding filter activation directly in the system. They also acknowledged that inputs which are difficult to classify are easily identified by the user in the Input Map (Figure 10d). Hence, they commented that DeepEyes also gives insights on how the training set can be modified in order to improve the classification as it shows which kind of input must be added to the training set.

6 IMPLEMENTATION

DeepEyes is developed to complement Caffe, a widely-used deep-learning library [18]. DeepEyes, requires Caffe files that describe the network and the parameters of the solver as input. DeepEyes trains the network using Caffe, but seamlessly builds the Progressive Visual Analytics system presented in this work on top of it.

For optimal performance, we implemented DeepEyes in C++. The interface is implemented with Qt. Perplexity Histograms and the Activation Heatmaps are implemented in JavaScript using D3 and are integrated in the application with QtWebKit Bridge. The Input- and Filter-Maps, are rendered with OpenGL. DeepEyes is implemented using a Model-View-Controller design pattern, allowing for the future extension to different deep-learning libraries, such as Google’s TensorFlow [1] or Theano [42].

7 CONCLUSIONS

In this work, we presented DeepEyes, a Progressive Visual Analytics systems that supports the design of DNNs by showing the link between the filters and the patterns they detect directly during training. The user detects stable layers (**T1**) that are analyzed in detail in three tightly-linked visualizations. DeepEyes is the only system we are aware of that supports DNN design decisions during training. Using DeepEyes the user detects degenerated filters (**T2**), inputs that are not activating any filter in the network (**T3**), and reasons on the size of a layer (**T4**). By visualizing the activation of filters and the separation of the input with respect to the labels, the user decides whether more layers are needed given the pattern-recognition problem at hand (**T5**). We used DeepEyes to analyze three DNNs, demonstrating how the insights obtained from our system help in making decisions about the network design.

A limitation of DeepEyes is that it relies on qualitative color palettes for the visualization of labels in the Input- and Filter-Maps. This solution does not scale when the number of labels is large, therefore we want to address this issue in future work. Further, the Input- and Filter-Map are created with dimensionality-reduction techniques, which may be affected by projection errors. Hence, adding interactive validation of the projections [29] is an interesting future work. Another interesting future work is the development of linked views that allows for the analysis of different type of data, such as text or video. We want to extend DeepEyes by integrating different deep-learning libraries, such as TensorFlow [1] or Theano [42], and to the analysis of different and more exotic network architectures, such as Recurrent Neural Networks [30] and Deep Residual Networks [14]. Finally, we want to apply DeepEyes for the analysis of DNNs in several application contexts, giving insights on their design.

REFERENCES

- [1] M. Abadi, A. Agarwal, P. Barham, and et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [2] J. M. Alvarez and M. Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pp. 2262–2270, 2016.
- [3] Y. Bengio. Deep learning of representations for unsupervised and transfer learning. In *ICML Unsupervised and Transfer Learning*, vol. 27, pp. 17–36, 2012.
- [4] I. Borg and P. J. Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005.
- [5] D. Ceneda, T. Gschwandtner, T. May, S. Miksch, H.-J. Schulz, M. Streit, and C. Tominski. Characterizing guidance in visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):111–120, 2017.
- [6] R. da Silva, P. E. Rauber, R. M. Martins, R. Minghim, and A. C. Telea. Attribute-based visual explanation of multidimensional projections. In *Proceedings of EuroVA (2015)*, pp. 134–139.
- [7] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. Technical report, University of Montreal, 2009.
- [8] J.-D. Fekete and R. Primet. Progressive analytics: A computation paradigm for exploratory data analysis. *arXiv preprint arXiv:1607.05162*, 2016.
- [9] D. Fisher, I. Popov, S. Drucker, et al. Trust me, i’m partially right: incremental visualization lets analysts explore large datasets faster. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1673–1682. ACM, 2012.
- [10] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2414–2423, 2016.
- [11] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.
- [12] A. W. Harley. An interactive node-link visualization of convolutional neural networks. In *International Symposium on Visual Computing*, pp. 867–877, 2015.
- [13] M. Harrower and C. A. Brewer. Colorbrewer.org: an online tool for selecting colour schemes for maps. *The Cartographic Journal*, 40(1):27–37, 2003.
- [14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.
- [15] T. Höllt, N. Pezzotti, V. van Unen, F. Koning, E. Eisemann, B. Lelieveldt, and A. Vilanova. Cytosplore: Interactive immune cell phenotyping for large single-cell datasets. In *Computer Graphics Forum*, vol. 35, pp. 171–180, 2016.
- [16] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.
- [17] S. Ioffe. Improved consistent sampling, weighted minhash and l1 sketching. In *2010 IEEE International Conference on Data Mining*, pp. 246–255. IEEE, 2010.
- [18] 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*, pp. 675–678. ACM, 2014.
- [19] A. Joulin, L. van der Maaten, A. Jabri, and N. Vasilache. Learning visual features from large weakly supervised data. In *European Conference on Computer Vision*, pp. 67–84. Springer, 2016.
- [20] A. Karpathy, J. Johnson, and L. Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.
- [21] D. Keim, G. Andrienko, J.-D. Fekete, C. Görg, J. Kohlhammer, and G. Melançon. Visual analytics: Definition, process, and challenges. In *Information visualization*, pp. 154–175. Springer, 2008.
- [22] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds., *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. 2012.
- [23] S. Kullback. *Information theory and statistics*. Courier Corporation, 1997.
- [24] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [25] 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.
- [26] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–48. Springer, 2012.
- [27] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu. Towards better analysis of deep convolutional neural networks. *Transactions on Visualization and Computer Graphics*, 2017.
- [28] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5188–5196, 2015.
- [29] R. M. Martins, D. B. Coimbra, R. Minghim, and A. Telea. Visual analysis of dimensionality reduction quality for parameterized projections. *Computers & Graphics*, 41:26–42, 2014.
- [30] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, vol. 2, p. 3, 2010.
- [31] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- [32] A. Mordvintsev and M. Tyka. Deepdream - a code example for visualizing neural networks, 2015.
- [33] T. Mühlbacher, H. Piringer, S. Gratzl, M. Sedlmair, and M. Streit. Opening the black box: Strategies for increased user involvement in existing algorithm implementations. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1643–1652, 2014.
- [34] 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*, pp. 3387–3395, 2016.
- [35] N. Pezzotti, T. Höllt, B. Lelieveldt, E. Eisemann, and A. Vilanova. Hierarchical stochastic neighbor embedding. In *Computer Graphics Forum*, vol. 35, pp. 21–30. Wiley Online Library, 2016.
- [36] N. Pezzotti, B. Lelieveldt, L. van der Maaten, T. Holtt, E. Eisemann, and A. Vilanova. Approximated and user steerable tsne for progressive visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, PP(99):1–1, 2016.
- [37] P. E. Rauber, S. Fadel, A. Falcao, and A. Telea. Visualizing the hidden activity of artificial neural networks. *IEEE Transactions on Visualization and Computer Graphics*, PP(99):1–1, 2017.
- [38] E. M. Reingold and J. S. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, 2:223–228, 1981.
- [39] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [40] C. Stolper, A. Perer, and D. Gotz. Progressive visual analytics: User-driven visual exploration of in-progress analytics. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1653–1662, 2014.
- [41] 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*, pp. 1–9, 2015.
- [42] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.
- [43] C. Turkay, E. Kaya, S. Balcisoy, and H. Hauser. Designing progressive and interactive analytics processes for high-dimensional data analysis. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):131–140, 2017.
- [44] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.
- [45] L. J. van der Maaten, E. O. Postma, and H. J. van den Herik. Dimensionality reduction: A comparative review. *Journal of Machine Learning Research*, 10(1-41):66–71, 2009.
- [46] M. Veta, P. J. van Diest, M. Jiwa, S. Al-Janabi, and J. P. Pluim. Mitosis counting in breast cancer: Object-level interobserver agreement and comparison to an automatic method. *PloS one*, 11(8):e0161286, 2016.
- [47] M. Veta, P. J. Van Diest, S. M. Willems, H. Wang, A. Madabhushi, A. Cruz-Roa, F. Gonzalez, A. B. Larsen, J. S. Vestergaard, A. B. Dahl, et al. Assessment of algorithms for mitosis detection in breast cancer histopathology images. *Medical image analysis*, 20(1):237–248, 2015.
- [48] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are

- features in deep neural networks? In *Advances in neural information processing systems*, pp. 3320–3328, 2014.
- [49] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.
- [50] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pp. 818–833. Springer, 2014.