Visualizing the Hidden Activity of Artificial Neural Networks

Paulo E. Rauber, Samuel G. Fadel, Alexandre X. Falcão, and Alexandru C. Telea

Abstract—In machine learning, pattern classification assigns high-dimensional vectors (observations) to classes based on generalization from examples. Artificial neural networks currently achieve state-of-the-art results in this task. Although such networks are typically used as black-boxes, they are also widely believed to learn (high-dimensional) higher-level representations of the original observations. In this paper, we propose using dimensionality reduction for two tasks: visualizing the relationships between learned representations of observations, and visualizing the relationships between artificial neurons. Through experiments conducted in three traditional image classification benchmark datasets, we show how visualization can provide highly valuable feedback for network designers. For instance, our discoveries in one of these datasets (SVHN) include the presence of interpretable clusters of learned representations, and the partitioning of artificial neurons into groups with apparently related discriminative roles.

Index Terms—Artificial neural networks, dimensionality reduction, algorithm understanding

1 Introduction

In machine learning, advances in computing power and techniques for building and training (deep) artificial neural networks (ANNs) have allowed these models to achieve state-of-the-art results in many applications related to pattern recognition [39]. However, successfully training ANNs is generally time-consuming, and requires significant expertise [5].

In data visualization, dimensionality reduction has been successfully used to compute *projections*: representations of high-dimensional data in lower-dimensional (usually 2D) spaces that try to preserve the data *structure*. This structure is related to the data distribution, relationships between points, and presence of clusters [26, 24]. When depicted by scatterplots, projections can be used to reason about the original data. Compared to other high-dimensional data visualizations, such as scatterplot matrices, parallel coordinates, star coordinates, and their variants, projections are considerably more scalable with respect to the number of dimensions [25, 44]. They are also scalable with respect to the number of observations, although visual clutter eventually becomes problematic. Concerning computational cost, recent dimensionality reduction techniques are able to deal with hundreds of thousands of observations at interactive rates [33, 19].

In this paper, we demonstrate the potential of dimensionality reduction techniques to provide insightful visual feedback about ANNs. Although we focus on multilayer perceptrons and convolutional neural networks, our approach is extensible to other types of networks (*e.g.*, LSTM or Elman networks [14]). Specifically, our proposed visualizations address the following two tasks:

- **T1**: Exploring the relationships between alternative representations of observations *learned* by ANNs.
- **T2**: Exploring the relationships between artificial neurons.

The projection-based visualization approach that we propose for **T1** is (sparsely) found in the machine learning literature [7, 29, 41, 36, 16]. However, such projections are typically used for illustrative purposes (Sec. 3). In contrast, we show how projections can aid existing approaches for understanding and improving ANNs (Sec. 5). Specifically, using three widely studied benchmark image classification datasets, we show how our visualization approach is able to confirm facts that are already known about ANNs, and reveal previously unseen relationships between learned representations. In this context, we also propose a novel visualization of the *evolution* of such representations (Sec. 5.4).

Our approach towards **T2** is completely new in the context of ANNs, although it is related to techniques developed for feature-space exploration (Sec. 3). Similarly to our approach for **T1**, we use projections to represent *similarities* between artificial neurons (given a particular set of input observations). We also propose a novel visualization of the relationships between artificial neurons and classes (Sec. 6.2). Although presented separately, our visualization approaches for **T1** and **T2** should be seen as complementary for understanding ANNs, as we exemplify in Sec. 6.2.

This paper is organized as follows. Section 2 briefly introduces pattern classification, together with the notation and definitions used in the text. Section 3 relates this paper to previous work in machine learning, information visualization, and visual analytics. Section 4 details the protocol followed by our experiments, which, although not crucial to our claims, allows for reproducibility. Section 5 presents the results of our projection-based visualizations of the relationships between learned representations for different datasets (**T1**), highlighting valuable insights gained from visualization. Section 6 presents our projection-based visualizations of relationships between artificial neurons (**T2**). Section 7 discusses the limitations of our work. Section 8 summarizes our findings and suggests future work.

2 Preliminaries

A dataset $\mathcal{D}$ is composed of pairs $(\mathbf{x}, \mathbf{y})$, where $\mathbf{x} \in \mathbb{R}^m$ is an *observation*, and $\mathbf{y} \in \{0,1\}^d$ is a *target class assignment*. If $y_c = 1$, observation $\mathbf{x}$ belongs to class c. In this paper, each observation corresponds to a 2D image and belongs to a single class, although these are not limitations of our proposal.

We consider two kinds of ANNs: *multilayer perceptrons* (MLPs) and *convolutional neural networks* (CNNs). Such a network represents a parameterized function $f : \mathbb{R}^m \to (0,1)^d$, which usually attempts to generalize class assignments from examples in $\mathcal{D}$. Computation in these networks is performed by *artificial neurons*, which are typically organized into *layers*, as outlined next.

Multilayer perceptrons: In this model, the weighted input to neuron j in layer l is defined as $z_j^{(l)} = b_j^{(l)} + \sum_k w_{j,k}^{(l)} a_k^{(l-1)}$, where $b_j^{(l)}, w_{j,k}^{(l)} \in \mathbb{R}$ are free parameters, and $a_k^{(l-1)}$ is the activation (output) of neuron k in layer $l-1$ (Fig. 1). In other words, each neuron computes a linear combination, plus a bias, of neuron activations from the previous

- *Paulo E. Rauber is with the University of Groningen and with the University of Campinas. E-mail: p.e.rauber@rug.nl.*
- *Samuel G. Fadel is with the University of São Paulo. E-mail: samuel.fadel@usp.br.*
- *Alexandre X. Falcão is with the University of Campinas. E-mail: afalcao@ic.unicamp.br.*
- *Alexandru C. Telea is with the University of Groningen. E-mail: a.c.telea@rug.nl.*

Manuscript received 31 Mar. 2016; accepted 1 Aug. 2016. Date of publication 15 Aug. 2016; date of current version 23 Oct. 2016.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TVCG.2016.2598838

layer. The activation $a_j^{(l)}$ depends on the *activation function* chosen for layer l [5]. In a sigmoid layer, $a_j^{(l)} = 1/(1+\exp(-z_j^{(l)}))$; in a rectified linear layer, $a_j^{(l)} = \max(0, z_j^{(l)})$; in a softmax layer, $a_j^{(l)} = \exp(z_j^{(l)}) / \sum_k \exp(z_k^{(l)})$.

The *activation of layer* l is defined as $\mathbf{a}^{(l)} = \left(a_1^{(l)}, \ldots, a_{N^{(l)}}^{(l)}\right)$, where $N^{(l)}$ is the number of neurons in layer l. Thus, if we let f denote the function computed by the network and L denote its number of layers (including the input), we have $f(\mathbf{x}) = \mathbf{a}^{(L)}$ when $\mathbf{a}^{(1)} = \mathbf{x}$. Any layer between the first and the last is called a *hidden* layer. A network with more than one hidden layer is called a *deep* neural network [4].

Crucial to our approach is the fact that the activation of layer $l > 1$ can be seen as an alternative (learned) representation of the input observation, since the activation of layer l depends only on *learned* parameters and the activation of layer $l-1$ (see Fig. 1). The activations of a given layer for a set of observations (network inputs) is the focus of our visualization.

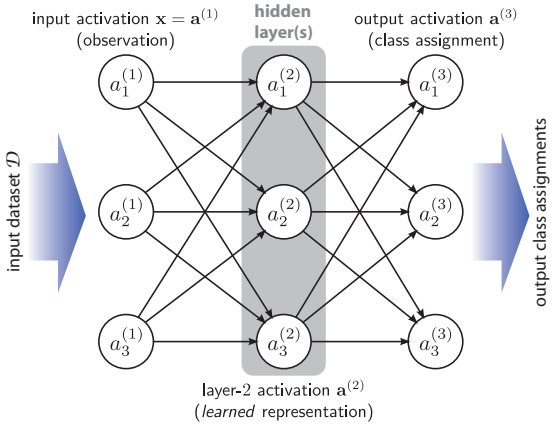

Fig. 1. Schema of MLP with three layers and three neurons per layer.

Convolutional neural networks: We consider three distinct layer types for this model: convolutional, max-pooling, and fully connected. A *convolutional* layer receives as input a $w \times h$ image with c color channels, and connects each of its neurons to a small window (all channels included) of the input. Neurons compute their weighted inputs and activations as usual. However, each neuron is replicated (by *parameter sharing*) for many input windows, given a pre-defined stride. When the output of all corresponding replicas are organized into a single-channel image, the operation is analogous to multichannel image convolution [21]. The output of a convolutional layer is obtained by stacking the outputs of sets of replicated neurons into a single multichannel image. The number of output channels can be seen as the number of convolutional filters. A *max-pooling* layer reduces the spatial dimensions of a multi-channel image by keeping the highest-value activation in a neighborhood (independently for each channel, for a pre-defined stride), and also outputs a multichannel image. A *fully connected* layer is analogous to an MLP layer, and is only followed by other fully connected layers. The activation of such a layer can also be seen as a learned representation of the input.

Network training: The weights and biases of an ANN, which we collectively denote by a vector $\boldsymbol{\theta}$, are adapted to minimize a cost function C that penalizes prediction errors on the dataset $\mathcal{D}$. For example, a softmax output layer is typically combined with a negative log-likelihood cost function

$$C = -\sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \sum_{k=1}^{d} y_k \ln a_k^{(L)},$$

where $a_k^{(L)}$ is the activation of neuron k on the last layer L when the network receives $\mathbf{x}$ as input [4]. Note that $-y_k \ln a_k^{(L)} \to \infty$ when $y_k = 1$ and $a_k^{(L)} \to 0$, which characterizes a prediction error.

The process of minimizing C with respect to $\boldsymbol{\theta}$ is called *training*. As C is differentiable with respect to every network parameter[1], minimization can be attempted by gradient descent. This technique iteratively updates $\boldsymbol{\theta}$ by the rule $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} C$, where η is the *learning rate*. In our work, we use (momentum-based) mini-batch stochastic gradient descent [5]. This technique partitions the dataset $\mathcal{D}$ into batches, and approximates $\nabla_{\boldsymbol{\theta}} C$ by using a single batch for each parameter update. After each batch is considered, training has completed one *epoch*. The *hyperparameters* (batch size, learning rate, number of layers, number of neurons per layer, etc) are not affected by training, and are usually chosen using previous experience and cross-validation.

Dimensionality reduction is performed by a function $p : \mathbb{R}^k \to \mathbb{R}^2$. The *projection* $\mathcal{A}_p \subset \mathbb{R}^2$ of a set $\mathcal{A} \subset \mathbb{R}^k$ is defined as $\mathcal{A}_p = \{p(\mathbf{a}) \mid \mathbf{a} \in \mathcal{A}\}$, and attempts to preserve the high-dimensional structure of $\mathcal{A}$, as already mentioned in Sec. 1.

3 Related Work

Machine learning experts have developed many strategies to design and improve ANNs, since the success of these models is highly impacted by the choice of preprocessing steps and several (interacting) hyperparameters. During training, a common approach is comparing model accuracy on a validation set with accuracy on a training set [35]. This helps diagnosing overfitting (low validation accuracy when compared to training accuracy) and underfitting (low accuracy in both cases). Manual choice of hyperparameters requires significant expertise and effort, and comprehensive guides have been written on the subject [32, 5]. The high dimensionality of the data and large number of parameters make ANNs hard to interpret, and make improving models a challenging task. Although automatic hyperparameter search is possible [40], it is generally (computationally) expensive.

Visual analytics and information visualization systems have been developed to inspect ANNs, since visual feedback is considered highly valuable by practitioners. For instance, Zeiler and Fergus [50] show how insight gained from visualizing ANNs has enabled them to outperform the state-of-the-art (at the time) on an major image classification benchmark. Their work aims to reconstruct an input image (observation) given a particular output channel of a convolutional layer (*feature map*). Reconstruction from activations is also investigated by Mahendran and Vedaldi [27]. Erhan *et al.* [9] search, through optimization, inputs that cause high activations in particular neurons, with the goal of understanding their roles. Yosinski *et al.* [48] visualize feature maps from CNNs trained for image recognition as they receive an input video stream, which enables the visual search for filters that detect a particular object. They also extend the work of Erhan *et al.* [9], showing how regularization techniques can be used to generate more interpretable images that cause high activations in a neuron. As will become clear, our approach is *complementary* to these visualizations.

Dimensionality reduction techniques can be divided into linear (*e.g.*, PCA, LDA, MDS) and non-linear (*e.g.*, Isomap, LLE, t-SNE) [24, 44]. Highly scalable techniques have been proposed (*e.g.,* LSP [33], LAMP [19]), which are able to deal with hundreds of thousands of observations (or more) at interactive rates. Dimensionality reduction has been previously applied to ANN visualization, due to its scalability in number of dimensions and observations. For instance, Erhan *et al.* [8] use projections of *learning trajectories* to study the effects of unsupervised pre-training. Each point in such a trajectory corresponds to the concatenation of output layer activations for a whole dataset at a given training stage. Closer to our work, projections of hidden layer activations, the subject of Sec. 5, have been used as high-level evidence of model efficacy [7, 29, 41, 36, 16]. Aubry and Russell [1] visualize hidden layer activations using PCA, aiming to understand their invariance with respect to several factors present in real and synthetic images.

In contrast to these works, our work is the first (to our knowledge) to present a detailed analysis of the insights on classification systems

[1]The rectified linear activation function is not differentiable at 0, but that is usually irrelevant in practice.

obtainable by projections of hidden layer activations. Separately, in Sec. 6, we use projections to explore relationships between neurons in a hidden layer. This visualization is completely new in the context of ANNs, but related to previous work on feature space exploration [49, 42]. However, in contrast to such previous work, which explores relationships between *input* features (dimensions) to a pattern classification technique, we visualize relationships between features (neuron activations) *learned* by such a technique. The importance of visualizing *black-box* techniques is discussed more generally by Mühlbacher *et al.* [30].

4 Experimental protocol

Our visualization approach is based on hidden layer activations extracted from a network trained for a given dataset, and can be divided into two parts: creating projections from these activations (**T1**), and depicting the relationships between the neurons that originate these activations (**T2**). This section details the protocol followed by the experiments presented in Secs. 5 and 6, which simultaneously detail and evaluate our visualization approach.

Datasets include three well-known image classification benchmarks: MNIST [22], SVHN [31] and CIFAR-10 [20]. MNIST has 50K training images, 10K validation images, and 10K test images (28×28 grayscale images of handwritten digits). SVHN has 63.2K training images, 10K validation images, and 26K test images (32×32 color images of house number digits). CIFAR-10 has 30K training images, 10K validation images and 10K test images (32×32 color photographs in ten object classes). Although the images in SVHN and CIFAR-10 are quite small, which allows fast experimentation, these are not *toy* datasets, and are widely used to evaluate state-of-the-art ANNs [47, 23].

Neural networks of two types are considered:

1. Multilayer perceptron (MLP): 3072 (784, for MNIST) input neurons, followed by four rectified linear hidden layers of 1000 neurons each. The output layer is softmax with 10 neurons. Dropout [41] is applied from the first hidden layer, growing from 0.2 to 0.5 in steps of 0.1 per layer.

2. Convolutional neural network (CNN): $32 \times 32 \times 3$ input image ($28 \times 28 \times 1$, for MNIST), followed by a convolutional layer with 32 $3 \times 3 \times 3$ (or $3 \times 3 \times 1$) filters, a convolutional layer with 32 $3 \times 3 \times 32$ filters, a 2×2 max-pooling layer (dropout 0.25), a convolutional layer with 64 $3 \times 3 \times 32$ filters, a convolutional layer with 64 $3 \times 3 \times 64$ filters, a 2×2 max-pooling layer (dropout 0.25), a fully connected layer with 4096 (or 3136) neurons (dropout 0.5), a fully connected layer with 512 neurons, and a softmax output layer with 10 neurons. All convolutional and fully connected layers (except the output) are rectified linear.

While larger models (in number of layers and parameters) are used for certain difficult classification tasks, the architectures sketched above are fully realistic, typical for image classification tasks, and sufficiently complex to warrant exploration.

Training is performed by momentum-based mini-batch stochastic gradient descent [5]. For MLPs, the batch size is 16, learning rate is 0.01, momentum coefficient is 0.9, and learning decay is 10^{-9}. For CNNs, the batch size is 32, learning rate is 0.01, momentum coefficient is 0.9, and learning decay is 10^{-6}. Initial weights for a neuron in layer l are sampled from a uniform distribution on $[-s, s]$, where $s = [6/(N^{(l-1)} + N^{(l+1)})]^{1/2}$, and biases start at 0. We manually chose these hyperparameters, together with the aforementioned architectures, based on cross-validation using the pre-defined validation sets. After the hyperparameters were chosen, we trained the models again using all data except the pre-defined test sets.

Table 1 summarizes the test set accuracy (AC, fraction of correctly classified observations) of our networks, and compares it to state-of-the-art networks, some of which also use preprocessing and data augmentation. Clearly, our networks achieve good accuracy on benchmark datasets. As such, they should be seen as realistic from an application perspective.

Table 1. Test Set Accuracies for our Two Architectures

Dataset \ Model	MLP	CNN	State-of-the-art
MNIST	98.52%	99.62%	99.79% [47]
SVHN	77.38%	93.76%	98.08% [23]
CIFAR-10	52.91%	79.19%	91.78% [23]

Activations for a given layer, the subject of our analysis, are extracted for a random subset of 2000 observations from the test sets, strictly to facilitate visual presentation. This subset is always the same for a given dataset. In two cases, we also extract activations from a random subset of a training set (Sec. 5.2). For CNNs, we only extract activations from fully connected layers.

Projections are created using a fast (approximate) implementation of t-distributed Stochastic Neighbor Embedding (t-SNE) [43], using the default parameters. We chose this technique based on its widespread popularity, proven ability to preserve neighborhoods and clusters in projections [44], and our previous experience in similar contexts.

We visualize projections as scatterplots, with points colored to show class assignment. We measure projection quality by the neighborhood hit (NH), which indicates how well classes are visually separated [33]. For a given k (in our work, $k = 6$), the NH for a point $\mathbf{a}_p$ is the ratio of its k-nearest neighbors that belong to the same class as $\mathbf{a}_p$. The NH for a whole projection is the average NH over its points. When displaying classification results for a test set in a projection, we use triangle glyphs to show misclassified observations, colored by their (incorrect) classifications (*e.g.*, inset in Fig. 3b).

Implementation of all our work is based on Python, Keras, Theano [3], NumPy [45], scikit-learn [34], and our extension of the feature-space exploration tool developed by Rauber *et al.* [37], which we used to conduct most of our visual exploration[2].

5 T1: Relationships between learned representations

This section presents the results of the experiments conducted to evaluate our proposed visualization of relationships between learned representations of observations (**T1**). For brevity, we focus on the most distinctive results and insights obtained for each dataset (Secs. 5.1 - 5.3). In Sec. 5.4, we present a novel visualization of the *evolution* of learned representations.

5.1 MNIST dataset: Exploring effects of training

The MNIST dataset is known as a relatively easy classification benchmark. This is confirmed by the clear visual separation between classes in the (raw data) projection of a subset of 2000 test observations (784-dimensional vectors), shown in Fig. 2. Points are colored according to their classes.

What untrained ANNs know: As already mentioned, we aim to understand the relationships between the alternative representations *learned* by ANNs trained for pattern classification. Firstly, consider an untrained MLP, whose parameters are randomly initialized according to Sec. 2. It is reasonable to hypothesize that a projection of the hidden layer activations of this MLP would have a significantly poorer visual separation between classes than the one shown in the projection in Fig. 2. To assess this, we show in Fig. 3a the projection of the last MLP hidden layer activations *before* training, for the same test subset used in Fig. 2. We see that our hypothesis was contradicted, since both projections in Figs. 2 and 3a show similar visual separation between classes. The good separation in Fig. 3a cannot be due to class information, since class labels are not used by the dimensionality reduction technique. Thus, there must be a clear structure in the hidden layer representations *before training*, which leads to the reasonably good NH of 83.78%. We are unaware of any previous visualizations

[2]Available at `http://www.cs.rug.nl/svcg/People/PauloEduardoRauber-featured`.

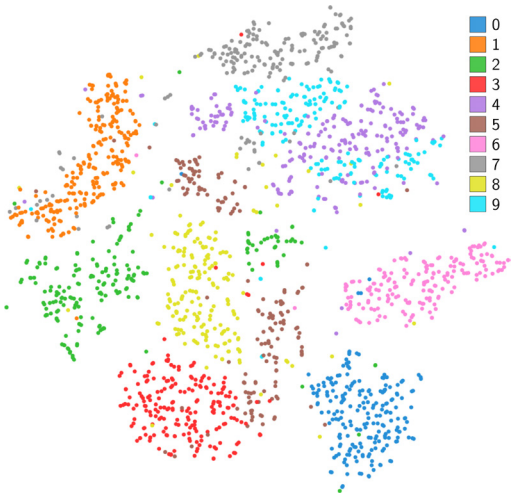

Fig. 2. Projection of observations, *MNIST* test subset (NH: 89.12%).

showing this qualitative insight, which could be used to compare different ANN initialization strategies. Alternatively, while it would be possible to train and evaluate a separate learning algorithm using the hidden layer activations before training, such approach would be more time-consuming, and would not convey the *structure* of the data in an easily interpretable manner. For instance, see the relationship between the visual clusters that correspond to classes 4 and 9 in Fig. 3a.

Training effects: A second natural hypothesis is that visual separation between classes would become better *after* training. This is related to the commonly-held view that ANNs learn to detect higher-level features that are useful for class discrimination [4]. To study this hypothesis, consider the projection of the last MLP hidden layer activations *after* 100 training epochs (Fig. 3b). Compared to Figs. 2 and 3a, we see a dramatic improvement in the visual separation between classes. Hence, the learning process definitely arrived at an alternative representation of the data that captures class structure, which is reflected by the projection.

Understanding misclassifications: Figure 3b shows several *visual outliers*, *i.e.*, points whose neighbors belong to a different class. Assuming the projection preserves the high-dimensional data structure, we could suspect that such outliers would be misclassified by the ANN. To check this, we color all points by their classes, and mark misclassifications by triangle glyphs. The inset in Fig. 3b shows several such misclassifications. When inspected, the visual outliers often correspond to observations that even humans would recognize as hard cases. For instance, Fig. 3b shows how an image of the digit 3 is (understandably) mistaken for the digit 5, and placed near the visual cluster corresponding to digit 5. This example shows that, despite the fact that projections may sometimes not fully preserve the data structure, as we discuss in Sec. 7, they are often predictive about class assignment. In other words, the similarities between hidden layer activations (shown by the projection) are a good predictor of the final class assignment by the ANN. This type of feedback is particularly useful when projections are combined with background knowledge and manual inspection of ANN inputs, as we continue to show in the next section.

5.2 SVHN dataset: Interpreting visual clusters

This section presents a compelling case for the visualization of learned representations in a second dataset (SVHN). Visualization provides a particular qualitative insight that is not easily available by other means.

The SVHN dataset is much more challenging for classification than MNIST. This is reflected, before training, in the visual separation between classes in the projection of the last hidden layer activations of an MLP, which is considerably poorer for SVHN (Fig. 4) than for MNIST (Fig. 3a) This is also confirmed by the corresponding NHs. Just as for

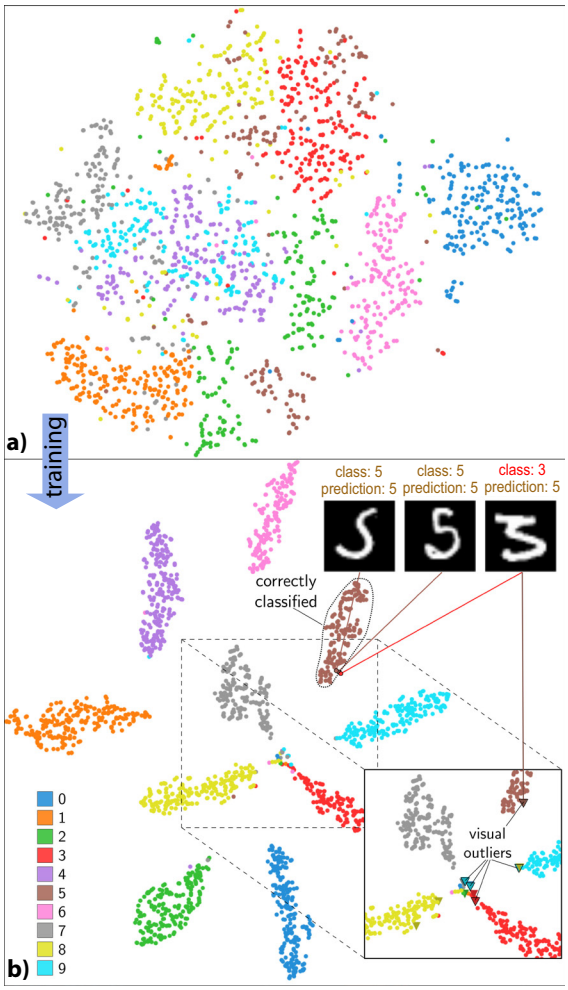

Fig. 3. Projection of the last MLP hidden layer activations, MNIST test subset. a) Before training (NH: 83.78%). b) After training (NH: 98.36%, AC: 99.15%). Inset shows classification of visual outliers.

MNIST, the visual separation is significantly improved after training, as shown by Fig. 5b.

Comparing different layers: All projections presented so far showed activations of last MLP hidden layers. However, our MLP architecture has four hidden layers. It is often hypothesized (but not usually supported by evidence) that a properly trained deep ANN has activations at later layers that correspond to discriminative higher-level features of the original observations [4]. We can verify this by inspecting activations of earlier layers which, in our case, have the same number of neurons. Consider the projection of the activations of the *first* MLP hidden layer after training (Fig. 5a). Visual separation between classes is clearly inferior to the one shown in the last hidden layer (Fig. 5b). We saw this phenomenon for most of the ANNs trained in our study. This is a new finding, which is not documented in the existing literature on ANNs. Note that there are no easy alternatives to obtain such an insight. For instance, confusion matrices could be used to diagnose the confusion between classes for a learning algorithm trained on the hidden layer activations. However, a confusion matrix would not convey nearly as much information about the structure of the data as a projection. Furthermore, a confusion matrix for a 10-class problem (our case) has 45 independent scalar values (after considering symmetries), which makes it quite difficult to inspect.

In comparison to the results obtained with the MLP (AC: 77.38%), the CNN obtains considerably better classification results on the test set (AC: 93.76%). Figure 6 shows the projection of the last CNN hid-

Fig. 4. Projection of the last MLP hidden layer activations before training, *SVHN* test subset (NH: 20.94%). Poor class separation is visible.

den layer activations after training. Clearly, visual separation and classification results improved together (cf. Fig. 5b), which is another example of the predictive power of projections.

Improvement based on visual feedback: We now present a particularly salient example of the value of the insight provided by projections. In Fig. 5b and (most notably) in Fig. 6, we notice a very distinctive pattern: each class (color) seems to be split into two visual clusters. Upon further inspection (brushing points), we found that one of the same-colored visual clusters corresponds to dark digits on light backgrounds, and the other to light digits on dark backgrounds (see examples in Fig. 6). We are unaware of previous work that documents such a remarkable pattern (visual or otherwise) in learned representations for the SVHN dataset, even though this dataset has been extensively used to evaluate ANNs in hundreds of publications. This is an example of how qualitative feedback can be hard to obtain by the typical quantitative approaches used to evaluate ANNs.

Since the projection in Fig. 6 suggests that, for each class, there are two kinds of images that have dissimilar internal representations, we naturally suppose that removing such (apparently unnecessary) variability in the input images would improve classification efficacy. To evaluate this, we preprocessed the images in SVHN in a simple manner: we apply the Sobel operator, after a small Gaussian blur, to approximate the gradient magnitude of the grayscale counterpart to each image. This yields grayscale images that are bright on the edges between background and foreground, and avoids the task of detecting if a digit is light or dark, which is not trivial given the high variability of the images (see supplemental Fig. 3). Next, we use the experimental protocol in Sec. 4 to classify the preprocessed test set. We obtain an increase in accuracy of 3.96% (1030 test observations) with the MLP, and 0.65% (169 test observations) with the CNN. While the CNN gain is small, we stress that it was obtained by *exactly* the same network architecture that was used for the original images. In contrast, the MLP gain is quite significant. The difference in gains for the two architectures can be explained by two facts. Firstly, it is easier to obtain gains when a model is further away from ideal performance, as in the MLP case. Secondly, our preprocessing is highly related to the operation of convolutional layers. This can be sufficient to enable good generalization in the CNN case, given the large amount of labeled data available for training. Finally, we also note that, in contrast to Figs. 5b and 6, the corresponding projections of the preprocessed test subset (supplemental Figs. 1 and 2) do not show two distinct visual clusters for each class. The increase in neighborhood hit (MLP 9.06%, CNN 1.43%) for those projections also mimics the increase in accuracy. Overall, these facts corroborate our hypothesis about the *semantics* of the internal ANN representations.

We note that it is already known in the literature that preprocessing this dataset (by local contrast normalization) sometimes leads to better performance [12]. However, this procedure was never justified by the foreground-background insight that we discovered. In the general case, a practitioner might be unaware of important preprocessing steps for a particular domain or dataset. In such situations, we claim that projections can provide highly valuable qualitative information about the representations learned by ANNs. As we already argued,

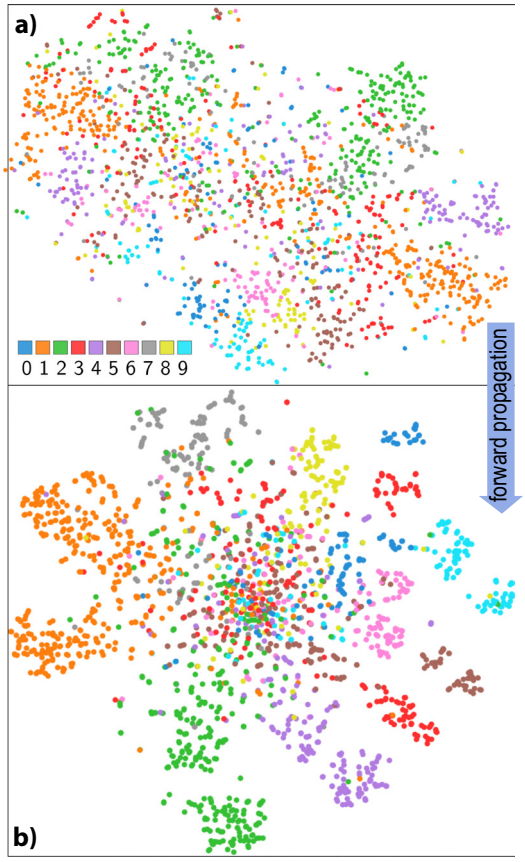

Fig. 5. Projection of the MLP hidden layer activations after training, SVHN test subset. a) First hidden layer (NH: 52.78%). b) Last hidden layer (NH: 67%).

such feedback is very hard to obtain by other (non-visual) means.

Explaining misclassifications: Consider the cyan point outlined in Fig. 6, which corresponds to digit 9, but is placed near the green cluster corresponding to dark digits 2 on light backgrounds. Notice how the dark border to the right of the digit 9 could be interpreted as a malformed digit 2. Knowing the semantics behind the green cluster, we can explain the misclassification more easily. While an experienced practitioner may have guessed why the misclassification occurred without the visual feedback, the *semantics* assigned to the visual clusters (found through visualization) provides extra evidence about the misclassification. In the general case, misclassification causes may be less obvious, and the visualization of learned representations becomes even more useful. Understanding the causes of misclassifications is useful both for improving models and for deciding when a model has achieved satisfactory performance.

Assessing training: All the projections presented so far were created from activations from a subset of a *test* set. However, insight can also be gained by inspecting projections of a subset of a *training* set. For training data, it is natural to expect that the visual separation between classes will be even better than in test data. To verify this, we compare the projection of a subset of the SVHN training set activations in the last MLP hidden layer (Fig. 7) with that of the corresponding test set (Fig. 5b). Indeed, we see a better visual class separation in the former, which is also reflected by a better NH (71.43% *vs* 67%). Comparing the two visual separations (training *vs* test data) supports several assessments. Firstly, a badly separated training set projection may indicate a poorly trained network, which has low chances of performing well on test data. A very well separated training set projection and a poorly separated test set projection may indicate poor generalization (caused, for example, by overfitting). Such assessments can also be

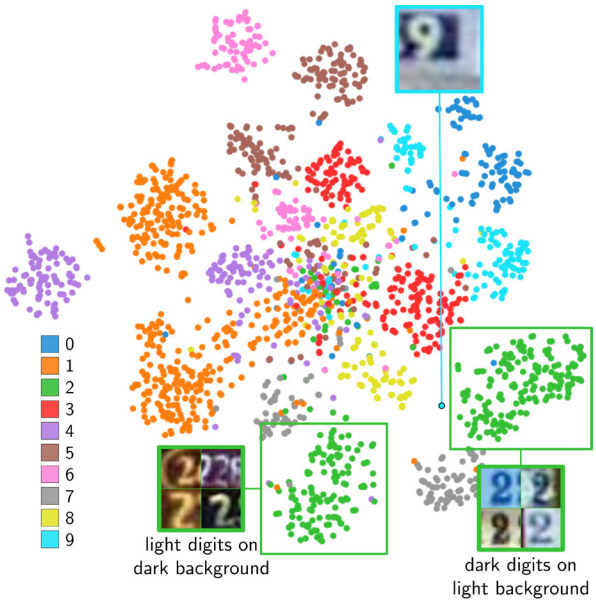

Fig. 6. Projection of the last CNN hidden layer activations after training, *SVHN* test subset (NH: 85.02%). Insets show example observations (images) from the visual clusters.

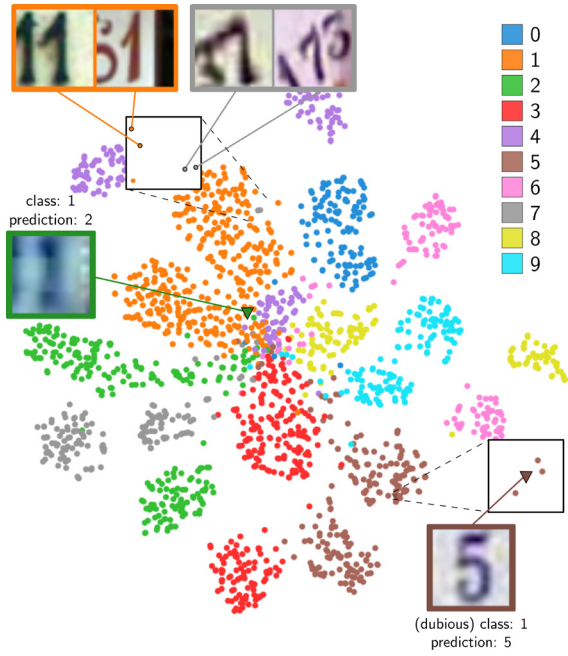

Fig. 8. Projection of last CNN hidden layer activations after training, *SVHN* training subset (NH: 93.83%, AC: 99.9%).

restricted to *parts* of a projection. In Fig. 7, for example, we see bad visual separation in the center. This is also the area where most classification errors (marked by triangles) are found.

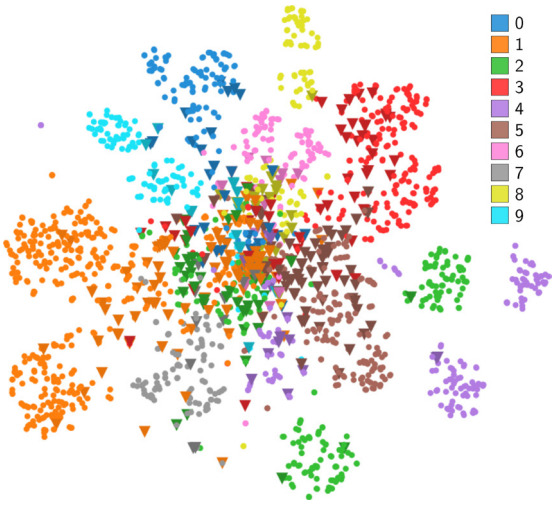

Fig. 7. Projection of last MLP hidden layer activations after training, *SVHN* training subset (NH: 71.43%, AC: 81.65%).

In the CNN case, similar results are obtained by comparing Fig. 8 (projection of last CNN hidden layer activations after training, for a training subset) with Fig. 6 (corresponding projection for the test subset). Comparing projections from different architectures is also insightful: in our case, we see that the CNN performs considerably better on the training set than the MLP, which matches the perceived visual separation and NH for Figs. 7 and 8. In fact, the CNN yields only two training set misclassifications. By brushing them in the projection (Fig. 8), we discover that one of them is incorrectly labeled (bottom right inset in Fig. 8).

Discovering potential overfitting: Figure 8 provides a final interesting insight. Consider the two gray points placed near the orange visual cluster (top left). Although the CNN assigns the correct class to these points (digit 7), the representation of the last hidden layer places them near orange points (digit 1). This appears to be due to the fact that the two images of the digit 7 actually resemble images of the digit 1. The placement of these two points in the projection can be a sign that the last layer learned to *work around* (overfit) the internal representation (recall that we are looking at training data). This could mean that the network will not work well in similar cases, classifying digits 7 as digits 1. Naturally, care must be taken when drawing conclusions from the placement of small sets of points, as we discuss in Sec. 7.

5.3 CIFAR-10 dataset: Interpreting confusion zones

The CIFAR-10 dataset is considerably more challenging for classification than the previous two (Tab. 1). This dataset provides another example where poor visual separation between classes predicts low classification accuracy. The projection of the last CNN hidden layer activations after training shows significant overlap between visual clusters (Fig. 9), matching the somewhat low classification accuracy (78.7%). Similarly to Fig. 7, the area with poorest separation between classes, or *confusion zone*, predicts well where most misclassifications occur (triangles in the center of Fig. 9).

As in Sec. 5.2, inspecting the visual outliers is also interesting. Consider the outlier in the middle of the large cyan visual cluster in Fig. 9. Since the class in cyan corresponds to truck images, and the outlier observation (automobile) looks very similar to members of that class, it is not surprising that the corresponding point becomes a visual outlier given the learned representation. Many other examples like this can be found in the projection.

5.4 Evolution of learned representations

The previous sections presented projections of learned representations (activations) for combinations of datasets, training stages (epochs), and layers. However, a single projection does not show how these representations *evolve*. The goal of this section is to introduce a compact visualization that summarizes two dimensions of evolution: inter-epoch and inter-layer. Given an observation and a layer, inter-epoch evolution refers to changes to the activations of that layer that are a consequence of learning (parameter changes as epochs progress). Separately, given an observation and an epoch, inter-layer evolution refers

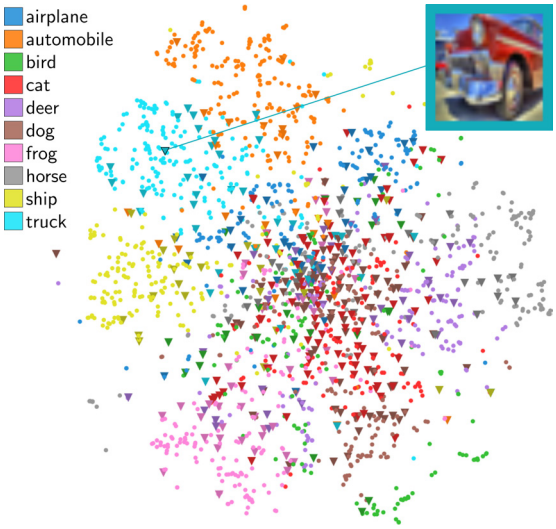

Fig. 9. Projection of last CNN hidden layer activations after training, *CIFAR*-10 test subset (NH: 53.43%, AC: 78.7%).

to the changes in internal representation as the observation "flows" through the layers of the network.

Let $\mathcal{A}[1], \ldots, \mathcal{A}[T]$ be a sequence of sets of (high-dimensional) activations, where each activation $\mathbf{a}[t] \in \mathcal{A}[t]$ originates from the same observation as a single activation $\mathbf{a}[t+1] \in \mathcal{A}[t+1]$. One way to visualize the evolution in such sets of activations is dimensionality reduction, applied in such a way that changes in the resulting projections will reflect changes in the corresponding high-dimensional data (as in the work of Jäckle *et al.* [18], for instance). This can be done by creating a projection $\mathcal{A}_p[t]$ for each activation set $\mathcal{A}[t]$. When doing this, it is essential to eliminate variability between projections that does not reflect changes in the high-dimensional data. The (arguably) simplest way to do this is to compute $\mathcal{A}_p[t]$ independently for each t, and use point-cloud registration [13] to align the resulting projections. However, dimensionality reduction techniques, including t-SNE, often yield large changes in *global* visual cluster placement for *small* data changes, which registration cannot eliminate [10]. For iterative techniques such as t-SNE, another intuitive solution is to initialize the positioning in $\mathcal{A}_p[t+1]$ with the previously computed $\mathcal{A}_p[t]$. We verified that this is a poor alternative, as it significantly biases the sequence of projections to show the evolution due to initialization in a (likely) *better* state with respect to the optimization goal. Instead, we employed a simple strategy: computing a (randomly initialized) projection $\mathcal{A}_p$ of $\mathcal{A}[1]$ using t-SNE, and using $\mathcal{A}_p$ to initialize each $\mathcal{A}_p[t]$.

The resulting sequence of projections can be visualized in several ways. We discard animation, since it is hard to track the motion of a large number of colored points over many frames [38]. An alternative is to create a 2D trail for each sequence $\mathbf{a}_p[1], \ldots, \mathbf{a}_p[T]$ of corresponding points. However, directly drawing these trails causes a large amount of clutter and occlusion. We address this by using trail bundling. This is analogous to earlier applications of bundling to visualize vehicle and eye-tracking trails [17]. We employ a recent high-performance GPU-based bundling algorithm [46], which allows a high degree of control in real time. The following examples show how the resulting bundled images can be explored.

Inter-layer evolution: Figure 10 shows the inter-layer evolution of a MNIST test subset (after training). The bundled image summarizes a sequence of four projections, one per hidden layer, shown as thumbnails. Trail hues encode classes, and edge brightness encodes layer number (depth). Thus, the brightness gradient shows how activation data "flow" through the four network layers. The bundle shapes show that the visual clusters are quite stable over all layers. Hence, the network arrives at a reasonably good separation between classes already at early layers. The gradients also show that some visual clusters become more compact in later layers (*e.g.*, tight bright area in the green cluster, whose evolution is indicated by the gray arrows in the figure), and that some clusters distance themselves from the others (*e.g.*, brightness pattern in the purple cluster). Thus, later network layers *strengthen* the class coherence and separation achieved by the first layer. We also see that there are only a few visual outliers (stray trails) that connect distinct visual clusters. Therefore, only few observations change clusters as the activation data flow through the network. In summary, we infer that the network layers after layer 1 mainly *refine* cluster coherence.

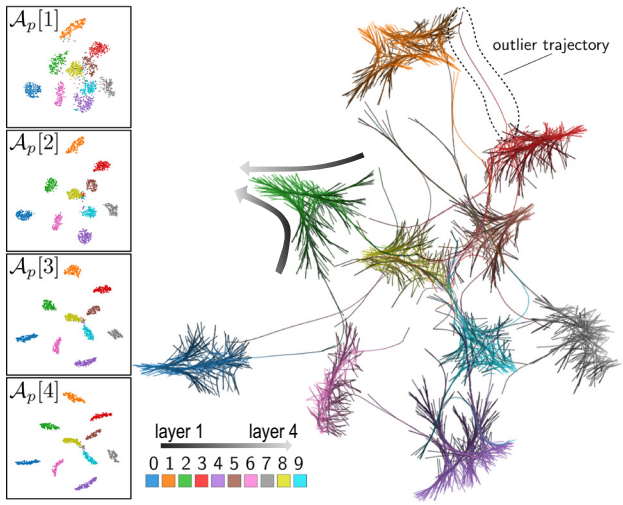

Fig. 10. Inter-layer evolution, four MLP hidden layers after training, *MNIST* test subset. Brighter trail parts show later layers.

Inter-epoch evolution: We could employ the same ideas to visualize inter-epoch evolution. However, our results in this case show that the images are significantly harder to interpret (*e.g.*, supplemental Fig. 4). This is due to a combination of large changes in the very first epoch, high intra-visual-cluster variance between epochs, and a much larger number of *frames* (typically hundreds) to be summarized. For this reason, we employed a different strategy to visualize inter-epoch evolution. Consider again the sequence $\mathcal{A}[1], \ldots, \mathcal{A}[T]$ of activation sets. For inter-epoch evolution, $\mathcal{A}[t] \subset \mathbb{R}^k$ for a fixed k, for all t. Hence, we can create a projection for the set $\bigcup_t \mathcal{A}[t]$, which contains activations for all epochs. As we compute a single projection, there is no spurious inter-frame variation. Figure 11 shows the inter-epoch evolution for the last CNN hidden layer activations using this strategy, from epochs 0 to 100, in steps of 20 (12K points in total). Hues indicate class, and brighter edge fragments correspond to later epochs. The thumbnails in Fig. 11 show points from three selected epochs. It is interesting to note how the dimensionality reduction technique placed the points corresponding to earlier epochs (darker) in the center of the projection, considering that it does not explicitly receive this information. This phenomenon also happens for SVHN and CIFAR-10.

Finally, we note that our choice of bundling algorithm provides a high degree of control over the level of *trail simplification* [46], which leads to a visualization that can also be explored in different levels of abstraction.

6 T2: Relationships between Neurons

The projections shown in Sec. 5 help understanding the relationships between the learned representations of *observations*. However, they do not represent relationships between the *neurons* in a given layer, or how neurons interact to fulfill their discriminative tasks. For this, we complement the *activation projections* shown so far by *neuron projections*. In a neuron projection, a point depicts a neuron. Points are

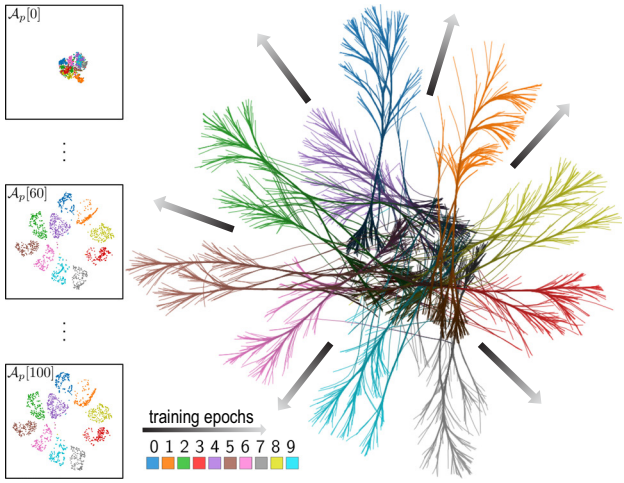

Fig. 11. Inter-epoch evolution, last CNN hidden layer, epochs 0-100, in steps of 20, *MNIST* test subset. Brighter trail parts show later epochs.

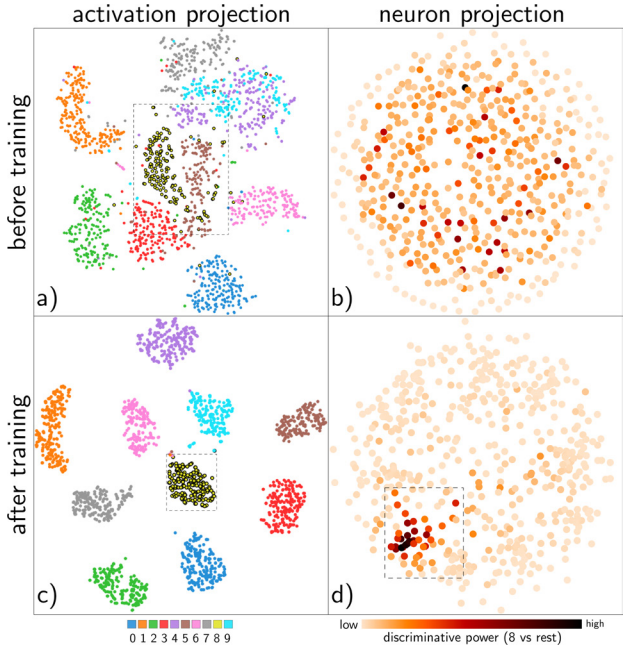

Fig. 12. Activation and neuron projections of last CNN hidden layer activations before and after training, *MNIST* test subset. Neuron projection colors show the neurons' power to discriminate class 8 *vs* rest.

placed in 2D based on the *similarity* between neurons. To our knowledge, this is the first time artificial neurons are visualized this way.

We define the dissimilarity $d_{i,j}$ between neurons i and j as $d_{i,j} = 1-|r_{i,j}|$, where $r_{i,j}$ is the empirical (Pearson's) correlation coefficient between neurons i and j on a dataset composed of layer-l activations (recall that each element of an activation vector is a neuron output). This metric captures both positive and negative linear correlations between pairs of neurons. From the matrix of pairwise dissimilarities, we compute a projection using (absolute metric) multidimensional scaling (MDS) [6]. While t-SNE is particularly concerned with preserving neighborhood relationships [44], absolute metric MDS attempts to preserve *global* pairwise dissimilarities as well as possible [6], which is more appropriate in this scenario. As we confirmed through preliminary experiments, MDS presents more coherent relationships between neurons that are discriminative for a particular class, which is important for a neuron projection (see Secs. 6.1 - 6.2).

6.1 MNIST dataset

We use the MNIST dataset to introduce neuron projections. Figure 12c shows the activation projection and Figure 12d the corresponding neuron projection for the last CNN hidden layer activations, after training. Ignoring the colors for a moment, we see no clear pattern in the neuron projection (Fig. 12d), except for some ill-defined visual clusters. We next color each point (neuron) based on its ability to discriminate between class 8 (marked yellow in Fig. 12c) and all other classes, computed by a standard feature selection technique, based on extremely randomized trees [11]. A very clear pattern emerges: all discriminative neurons for class 8 are placed near each other in the neuron projection. In contrast, consider the corresponding activation/neuron projections for the same hidden layer *before* training (Figs. 12a,b): the discriminative neurons for class 8 are *scattered* over the neuron projection. This shows that training creates sets of highly related neurons, which work together in the classification task. An analogous phenomenon can be observed for all other classes (see supplemental Fig. 5).

We can use feedback about the relationships between neurons and classes to diagnose the absence of specialization for a particular class in a given layer. This can help pinpoint causes of poor classification efficacy. As a related example, consider *dropout*, a widespread heuristic for training deep ANNs [41]. Dropout is often justified by its hypothesized capacity to inhibit co-adaptation of artificial neurons [41]. We believe our approach could be applied to *qualitatively* investigate and compare this and similar heuristics (*e.g.*, DropConnect [47]), which are still poorly understood [47].

6.2 SVHN dataset

As mentioned in Sec. 6.1, some feature selection methods provide a score that measures the importance of a given neuron (feature) to discriminate between a given class and other classes. We show next how this information can be used to depict how each neuron contributes to class discrimination.

For a given feature scoring technique (extremely randomized trees, in the next example), each neuron j receives a normalized score $s_{c,j} \in [0, 1]$, which measures the power of neuron j, relative to other neurons, to discriminate between class c and all other classes. We associate neuron j to the class $c^*_j = \arg\max_{c'} s_{c',j}$. We depict this by coloring point j with the hue associated to class c^*_j, and with a saturation given by $s_{c^*_j,j}$. We call this depiction a *discriminative neuron map*. Notice that the score $s_{c,j}$ is normalized over neurons for each class, so a highly saturated point in the visualization may have a low *absolute* discriminative power.

Figure 13 shows the discriminative neuron map for the SVHN test subset, last hidden layer activations, after training. The presence of compact visual clusters shows how the entire set of neurons can be (almost) partitioned into groups with related discriminative roles (specializations), even though the neuron projection is created without any class information.

The activation and neuron projections can be combined to elucidate the role of particular neurons. Consider neuron 460, which is highly associated to class 3 according to Fig. 13. The activation of this neuron is encoded using colors in Fig. 14, for all inputs in the test subset. According to that image, neuron 460 is responsible for finding one of the two red visual clusters in the projection (see bottom left inset in Fig. 14), which corresponds to images of the digit 3 on a light background (as we discovered through visualization in Sec. 5.2). It is also interesting to note that an observation that has a high activation for that neuron, and belongs to another visual cluster, resembles a digit 3 upon closer inspection (digit 5 on a light background, top left inset in Fig. 14). Obtaining these informations by typical approaches employed in machine learning would be significantly more difficult and time-consuming, which is key to the importance of our visual approach. Finally, as already mentioned in Sec. 2, understanding the role

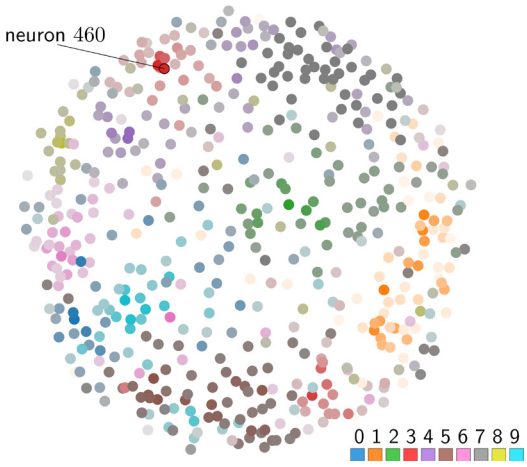

Fig. 13. Discriminative neuron map of last CNN hidden layer activations after training, *SVHN* test subset.

of particular neurons in ANNs is considered a very important problem, for which the discriminative neuron map is a novel approach.

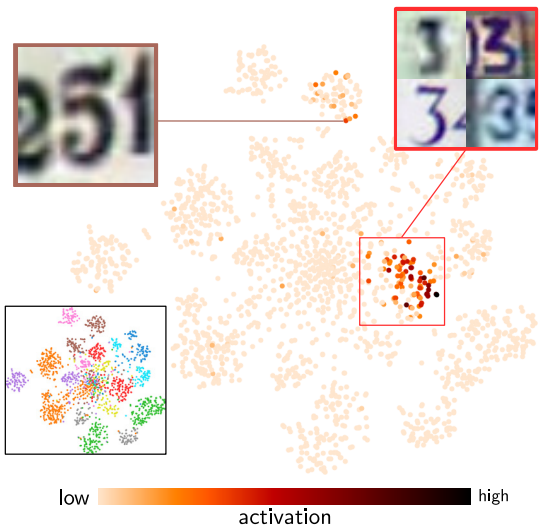

Fig. 14. Activation projection of the last CNN hidden layer after training, *SVHN* test subset. Color shows the activation of neuron 460, highly associated to class 3 (see also Fig. 13).

7 Discussion

In this section, we discuss several important aspects of our proposed visualizations and the experimental analysis conducted to evaluate them.

Scalability: Although dimensionality reduction is among the most scalable methods for high-dimensional data visualization, it still has some issues. Firstly, visual clutter occurs when visualizing a large number of activations or neurons. Secondly, considering the activation projections, although our particular choice of technique (Barnes-Hut t-SNE) is computationally scalable, it still requires approx. 10 minutes to compute a projection containing 70K 50-dimensional observations [43]. Fortunately, preliminary experiments with dimensionality reduction techniques that are able to deal with hundreds of thousands of observations at interactive paces [33, 19] were also promising. Thirdly, we use categorical color-coding to show class information. This creates well-known clutter and color-distinguishing challenges in scatterplot visualization when the number of classes is large.

Techniques: Our choice of (Barnes-Hut) t-SNE is justified by its widespread popularity and well-known ability to preserve clusters and neighborhoods in projections [44]. Although the latter property is very important to understand relationships between learned representations, our proposal is not highly coupled to t-SNE. Similarly, neuron projections are not coupled to absolute metric MDS. In both cases, other dimensionality reduction techniques can be used, provided that they preserve neighborhoods and distances well, respectively.

Coverage: As an experimental study that involves many free parameters, our conclusions are limited to the datasets and networks that we presented. However, our findings for all datasets and networks were consistent. In particular, there were no cases where projections would not provide any useful feedback. Additionally, the extent of our validation (*i.e.*, experimental protocol, number of datasets) is in line with comparable works in visual analytics and machine learning.

Validity: We employed good practices to train artificial neural networks in well-known benchmark datasets, and carefully detailed our experimental protocol to maximize reproducibility.

It is extremely important to address a specific threat to the validity of our approach: the fact that dimensionality reduction techniques provide few *quality* guarantees, and may introduce misleading visual artifacts [28]. For instance, different initializations of t-SNE may or may not yield the visual outliers presented in Sec. 5. To solve this issue, users should primarily evaluate the quality of a given projection using existing metrics [28, 2]. Such metrics support both global assessment (overall quality of an entire projection) and local assessment (*i.e.*, whether a subset of points is placed well).

If a projection (or some of its parts) has poor quality, it should be discarded from further use. Conversely, if a projection (or some of its parts) has high quality, the patterns it shows are actually present in the data, an can be relied upon. As a side note, it should be clear that most interesting phenomena observed in the projections in Secs. 5 and 6 would be extremely unlikely artifacts (*e.g.*, visual cluster separation, partition of visual clusters between light/dark digits on dark/light backgrounds, and partitioning of neurons into specialties). Finally, we note that the feedback given by (activation) projections for classification problems is, in a sense, asymmetric: clear visual separation between classes surely implies an easy classification task, whereas unclear separation does not necessarily imply a difficult task.

8 Conclusion

In this paper, we have shown how dimensionality reduction can be used to visualize the relationships between learned representations (**T1**) and between neurons (**T2**) in artificial neural networks. Concerning the first task, our visualizations support the identification of confusion zones, outliers, and clusters in the internal representations computed by such networks. Separately, we also show how to visually track inter-layer and inter-epoch evolution of learned representations. Concerning the second task, we enable the inspection of relationships between neurons and classes (specialization), and similarity between neurons. In experiments on traditional benchmark datasets, we have shown that both our contributions can provide valuable visual feedback for network designers. This feedback may confirm the known, reveal the unknown, and prompt improvements along the classification pipeline, as we have shown through concrete examples.

There are several possibilities for future work. They include visualizing representations learned by recurrent networks, which currently achieve state-of-the-art results in many sequence-related tasks [14, 15]. The sequential nature of these networks introduces yet another challenge for visualization. Our approach for evolution visualization would also benefit from dimensionality reduction techniques designed specifically for time-dependent datasets.

Acknowledgments

We would like to thank CAPES, FAPESP (2012 / 24121-9, 2015 / 19851-6) and CNPq (302970 / 2014-2, 479070 / 2013-0) for the financial support.

REFERENCES

[1] M. Aubry and B. Russell. Understanding deep features with computer-generated imagery. In *International Conference on Computer Vision (ICCV)*, 2015.

[2] M. Aupetit. Visualizing distortions and recovering topology in continuous projection techniques. *Neurocomputing*, 70(7):1304–1330, 2007.

[3] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*, 2012.

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

[5] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, pages 437–478. Springer, 2012.

[6] I. Borg and P. J. Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005.

[7] 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. *arXiv preprint arXiv:1310.1531*, 2013.

[8] 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*, 11:625–660, 2010.

[9] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. *Dept. IRO, Université de Montréal, Tech. Rep. 4323*, 2009.

[10] F. J. García Fernández, M. Verleysen, J. A. Lee, and I. Díaz Blanco. Stability comparison of dimensionality reduction techniques attending to data and parameter variations. In *Eurographics Conference on Visualization*. The Eurographics Association, 2013.

[11] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[12] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[13] J. C. Gower and G. B. Dijksterhuis. *Procrustes Problems*. Oxford Statistical Science Series 30, 2004.

[14] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385 of *Studies in Computational Intelligence*. Springer, 2012.

[15] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[16] P. Hamel and D. Eck. Learning features from music audio with deep belief networks. In *Proc. International Society for Music Information Retrieval*, pages 339–344, 2010.

[17] C. Hurtr, O. Ersoy, S. Fabrikant, T. Klein, and A. Telea. Bundled visualization of dynamic graph and trail data. *IEEE Transactions on Visualization and Computer Graphics*, 20(8):1141–1154, 2014.

[18] D. Jäckle, F. Fischer, T. Schreck, and D. A. Keim. Temporal MDS plots for analysis of multivariate data. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):141–150, 2016.

[19] P. Joia, F. Paulovich, D. Coimbra, J. Cuminato, and L. Nonato. Local affine multidimensional projection. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2563–2571, 2011.

[20] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images, 2009. `www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. Neural Information Processing Systems*, pages 1097–1105. 2012.

[22] Y. LeCun, C. Cortes, and C. J. Burges. The MNIST database of handwritten digits, 1998. `http://yann.lecun.com/exdb/mnist/`.

[23] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. *arXiv preprint arXiv:1409.5185*, 2014.

[24] J. A. Lee and M. Verleysen. Nonlinear dimensionality reduction of data manifolds with essential loops. *Neurocomputing*, 67:29–53, 2005.

[25] S. Liu, D. Maljovec, B. Wang, P.-T. Bremer, and V. Pascucci. Visualizing high-dimensional data: Advances in the past decade. In *Proc. EuroVis – STARs*, 2015.

[26] S. Liu, B. Wang, P.-T. Bremer, and V. Pascucci. Distortion-guided structure-driven interactive exploration of high-dimensional data. *Computer Graphics Forum*, 33(3):101–110, 2014.

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

[28] 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.

[29] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

[30] 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, Dec 2014.

[31] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *Proc. Neural Information Processing Systems*, volume 2011, page 5, 2011.

[32] G. B. Orr and K.-R. Müller. *Neural networks: tricks of the trade*. Springer, 2003.

[33] F. V. Paulovich, L. G. Nonato, R. Minghim, and H. Levkowitz. Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. *IEEE Transactions on Visualization and Computer Graphics*, 14(3):564–575, 2008.

[34] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[35] L. Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

[36] A. r. Mohamed, G. Hinton, and G. Penn. Understanding how deep belief networks perform acoustic modelling. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4273–4276, March 2012.

[37] P. E. Rauber, R. R. O. Silva, S. Feringa, M. E. Celebi, A. X. Falcão, and A. C. Telea. Interactive Image Feature Selection Aided by Dimensionality Reduction. In *EuroVis Workshop on Visual Analytics (EuroVA)*, 2015.

[38] G. Robertson, R. Fernandez, D. Fisher, B. Lee, and J. Stasko. Effectiveness of animation in trend visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1325–1332, Nov 2008.

[39] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[40] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[41] N. Srivastava, G. 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.

[42] C. Turkay, P. Filzmoser, and H. Hauser. Brushing dimensions - a dual visual analysis model for high-dimensional data. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2591–2599, 2011.

[43] L. Van Der Maaten. Accelerating t-SNE using tree-based algorithms. *Journal of Machine Learning Research*, 15(1):3221–3245, 2014.

[44] L. Van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

[45] S. Van Der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[46] M. van der Zwan, V. Codreanu, and A. Telea. CUBu: Universal real-time bundling for large graphs. *IEEE Transactions on Visualization and Computer Graphics*, 2016. doi:10.1109/TVCG.2016.2515611.

[47] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *Proc. International Conference on Machine Learning*, pages 1058–1066, 2013.

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

[49] X. Yuan, D. Ren, Z. Wang, and C. Guo. Dimension projection matrix/tree: Interactive subspace visual exploration and analysis of high dimensional data. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2625–2633, 2013.

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