information

Article

Exploring Neural Network Hidden Layer Activity Using Vector Fields [†]

Gabriel D. Cantareira [1,*], Elham Etemad [2] and Fernando V. Paulovich [1,2,*]

[1] Instituto de Ciências Matemáticas e de Computacão, Universidade de São Paulo, São Paulo 13566-590, Brazil
[2] Faculty of Computer Science, Dalhousie University, Halifax, NS B3H 4R2, Canada; eetemad@dal.ca
[*] Correspondence: gabrieldc@icmc.usp.br (G.D.C.); paulovich@dal.ca (F.V.P.)
[†] This paper is an extended version of a paper published at IVAPP 2020.

Received: 16 July 2020; Accepted: 27 August 2020; Published: 31 August 2020

Abstract: Deep Neural Networks are known for impressive results in a wide range of applications, being responsible for many advances in technology over the past few years. However, debugging and understanding neural networks models' inner workings is a complex task, as there are several parameters and variables involved in every decision. Multidimensional projection techniques have been successfully adopted to display neural network hidden layer outputs in an explainable manner, but comparing different outputs often means overlapping projections or observing them side-by-side, presenting hurdles for users in properly conveying data flow. In this paper, we introduce a novel approach for comparing projections obtained from multiple stages in a neural network model and visualizing differences in data perception. Changes among projections are transformed into trajectories that, in turn, generate vector fields used to represent the general flow of information. This representation can then be used to create layouts that highlight new information about abstract structures identified by neural networks.

Keywords: explainable machine learning; neural network visualization; dimensionality reduction

1. Introduction

Given their ability to abstract high-level patterns and model data beyond most heuristics [1], Deep Neural Networks (DNNs) are currently among the state-of-the-art techniques for the analysis of large-scale, complex datasets. Despite the prevalence of DNNs in different domains, such as natural language processing, face and speech recognition, and generation of artificial data, its success heavily depends on the right choice of hyperparameters and architecture.

In recent years, visualization strategies have become increasingly popular in the research community to help analysts interpret the results and support the improvement of DNNs. One of the most popular visualization strategies, multidimensional projection techniques [2], has reportedly attained relative success in helping users explore and explain what happens inside DNNs [3–5]. These techniques aim to generate lower-dimensional visual representations capable of preserving multidimensional data structure, such as relationships between data instances or clusters' presence. However, the currently available techniques are somewhat limited when exploring sequential processes inside neural networks, such as the state of hidden layers during training or the shaping of high-level representations as data flows through different layers of a network. This information can provide insights to important issues, such as determining different layers' contributions in discriminating certain objects or tracking noteworthy behavior during training, to better understand unusual data.

In this paper, we present a novel approach to visualize the hidden structure of DNNs, designed to aid in understanding abstract high-level representations inside a model and how they are formed. In our approach, we project data extracted from various states of a neural network and estimate a common space between projections by computing a vector field that can show how these projections relate to one another. We show how this space can be used in visualization, as well as possibilities in identifying progress and tracking down meaningful changes in the neural network as training epochs go by or as data flows through the model. This paper is an extension of work presented at the IVAPP 2020 conference [6], refining dimensionality reduction methodology by employing a general approach to projection alignment [7], developing new visual representations for vector field and trajectory data incorporated to sequential projections, and presenting new experiments that highlight these properties.

Our main contributions are:

- A projection-based visual representation suited to represent sequential aspects of DNNs, reducing movement clutter while keeping distances meaningful;
- A visual transition space and vector field between two or more sequential projections that can be used to represent flow or evolution.

The remainder of the paper is organized as follows. In Section 2, we discuss related work regarding visualization techniques for neural networks. In Section 3, we formalize the problem and present a complete overview of our solution. In Section 4, we present an extensive set of experiments to show our solution in practice. In Section 5, we discuss our limitations, and the paper is concluded in Section 6.

2. Related Work

Artificial Neural Networks (ANNs) are structures composed of groups of simple and complex cells. Simple cells are responsible for extracting basic features, while complex ones combine local features producing abstract representations [8]. This structure hierarchically manipulates data through layers (complex cells), each using a set of processing units or neurons (simple cells) to extract local features. In classification tasks, each neuron divides input data space using a linear function (i.e., hyperplane), which is positioned to obtain the best separation as possible between labels of different classes. Thus, the connections among processing units are responsible for combining the half-spaces built up by those linear functions to produce nonlinear separability of data spaces [1]. Deep Neural Networks (DNNs) are artificial neural network models that contain a large number of layers between input and output, generating more complex representations. Such networks are called convolutional neural networks (CNNs) when convolutional filters are employed.

In the past few years, the use of visualization tools and techniques to support the understanding of neural network models has become more prolific, with many different approaches focusing on exploring and explaining different aspects of DNN training, topology, and parametrization [9]. As deep models grow more complex and sophisticated, understanding what happens to data inside these systems is quickly becoming key to improving their efficiency and designing new solutions.

When exploring layers of a DNN, a common source of data are the hidden layer *activations*: the output value of each neuron of a given layer when subjected to a data instance (input). Many DNN visualization approaches are focused on understanding the high-level abstract representations that are formed in hidden layers. This is often attained by transferring the activations of hidden layer neurons back to the feature space, as defined by the Deconvnet [10] and exemplified by applications such as the Deep Dream [11]. Commonly associated with CNNs, techniques based on this approach often try to explain and represent which feature values in a data object generate activations in certain parts of hidden layers. The Deconvnet is capable of reconstructing input data (images) at each CNN layer to show the features extracted by filters, supporting the detection of incidental problems based on user inspections.

Other techniques focus on identifying content that activates filters and hidden layers. Simonyan et al. [12] developed two visualization tools based on Deconvnet to support image segmentation, allowing feature inspection and summarization of produced features. Zintgraf et al. [13] introduced a feature-based visualization tool to assist in determining the impact of filter size on classification tasks and identifying how the decision process is conducted. Erhan et al. [14] proposed a strategy to identify features detected by filters after their activation functions, allowing the visual inspection of the impact of network initialization as well as if features are humanly understandable. Similarly, Mahendran et al. [15] presented a triple visualization analysis method to inspect images. Babiker et al. [16] also proposed a visual tool to support the identification of unnecessary features filtered in the layers. Liu et al. [17] present a system capable of showing a CNN as an acyclic graph with images describing each filter.

Other methods aim to explore the effects of different parameter configurations in training, such as regularization terms or optimization constraints [5]. These can also be connected to different results in layer activations or classification outcomes. Some techniques are designed to help evaluate the effectiveness of specific network architectures, estimating what kind of abstraction can be learned in each section, such as the approach described by Yosinki et al. [18].

The research previously described is focused on identifying and explaining *what* representations are generated. However, it is also important to understand *how* those representations are formed, regarding both the training process and the flow of information inside a network. Comprehending these aspects can lead to improvements in network architecture and the training process itself. The DeepEyes framework, developed by Pezzotti et al. [19], provides an overview of DNNs identifying when a network architecture requires more or fewer filters or layers. It employs scatterplots and heatmaps to show filter activations and allows the visual analysis of the feature space. Kahng et al. [20] introduce a method to explore the features produced by CNN's projecting activation distances and presenting a neuron activation heatmap for specific data instances. However, these techniques are not designed for projecting multiple transition states, and their projection methods require complex parametrization to show the desired information.

Multidimensional projections (or dimensionality reduction techniques) [2] are popular tools to aid the study of how abstract representations are generated inside ANNs. Specific projection techniques, such as the UMAP [21], were developed particularly with machine learning applications in mind. While dimensionality reduction techniques are generally used in ANN studies to illustrate model efficacy [4,5,22–24], Rauber et al. [3] showed their potential on providing valuable visual information on DNNs to improve models and observe the evolution of learned representations. Projections were used to reveal hidden layer activations for test data before and after training, highlighting the effects of training, the formation of clusters, confusion zones, and the neurons themselves, using individual activations as attributes. Despite offering insights on how the network behaves before and after training, the visual representation presented by the authors for the evolution of representations inside the network or the effects of training between epochs displays a great deal of clutter; when analyzing a large number of transition states, information such as the relationships between classes or variations that occur only during intermediate states may become difficult to infer. Additionally, the method used to ensure that all projections share a similar 2D space is prone to problems in alignment. In this paper, we propose a visualization scheme that employs a flow-based approach to offer a representation better suited to show transition stated and evolving data in DNNs. We also briefly address certain pitfalls encountered when visualizing neuron activation data using standard projection techniques, such as t-SNE [25], and discuss why these pitfalls are relevant to our application.

3. Visualizing the Activity Space

Our visual representation is based on gathering hidden layer activation data from a sequence of ANN outputs (steps), then projecting them onto a 2D space while sharing information to ensure semantically similar data remain in similar positions between projections. The movement of the same data instance throughout each projected step generate trajectories, which are then condensed into vector fields that reflect how data flows throughout the sequence. There are two main sequential aspects to be visualized with ANN outputs: how abstract data representations inside the network change and evolve with training and how representations are built and connected as one layer forwards information to the next. For each aspect, a different sequence of hidden layer outputs is needed: outputs from the same given layer during different epochs of training are used to visualize how training changes data representations in that layer, while outputs from different layers from the same network are used to visualize how data representations evolve as layers propagate information.

To build this representation, we first extract activation sets $A[1], A[2], \ldots, A[T]$ representing network outputs from T sequential steps of the process we want to explore. In this paper, we either (a) save a network model at different epochs of training, choose a slicing layer, feed the same set of input data to the saved models, and then save the outputs from the slicing layer, or (b) pick a given network model, slice it at different layers, feed the input data set, and save the outputs from these layers as activation sets.

Once the activation data are extracted, it is projected onto a 2D space using a multidimensional projection technique, obtaining $P[1], P[2], \ldots, P[T]$. To produce coherent projections, an alignment method is applied, ensuring that projections stay similar to each other while preserving original multidimensional distances. Then, positions of the same points in two subsequent projections form movement vectors $\phi[1], \ldots, \phi[T-1]$ that describe how data instances in one output changed to the next. This movement data are joined for all output transitions, generating trajectories (or *trails*) for each data instance across all T steps, which are then used to compute vector fields. Finally, projections, trajectories, and vector fields are combined to create visualizations. The 2D space shared by all projections is our *visual activity space*, and trajectories and vector fields describe how network outputs flow through it. Figure 1 summarizes this process.

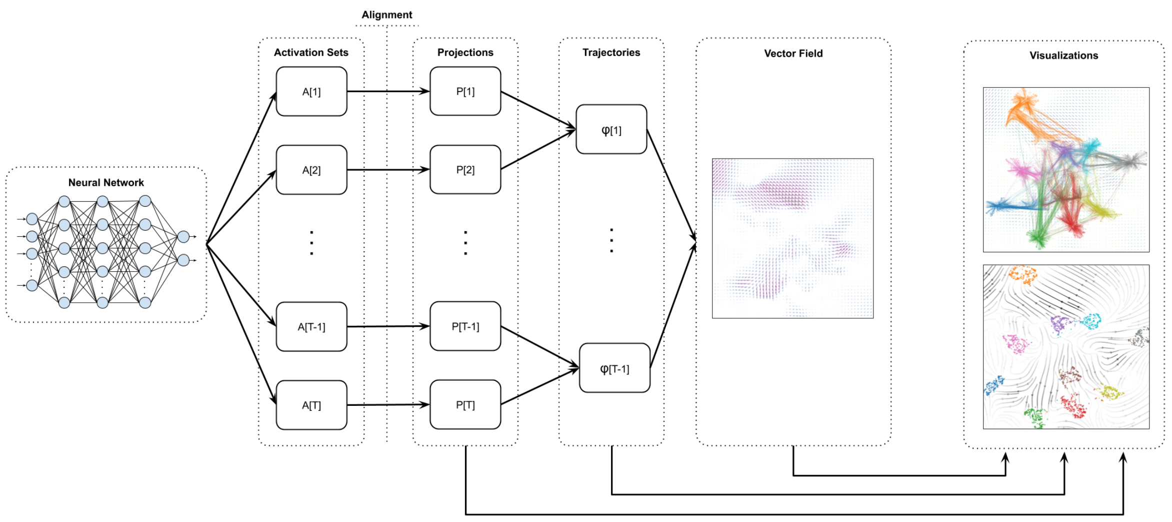

Figure 1. Overview of the learning space vector field generation. From a DNN model, we obtain sequential data in the form of hidden layer activations. We compute projections from this highly multivariate data, aligning them to ensure that obtained projections are synchronized. Differences between each pair of projections in the sequence (transition steps) are turned into trajectories and then processed to obtain vector fields representing changes in data. Finally, everything is combined to compose distinct visual representations of the neural network.

The following sections detail how data are projected, and how vector fields are generated and visualized in our model.

3.1. Projecting Data

Although we can compare any set of projections to produce vector fields, we need to eliminate changes between projections that do not reflect variations in the high-dimensional data as much as possible. Therefore, the projection technique itself must share information between all data sets to ensure the generation of a synchronized view. Although precise and popular, nonlinear projection techniques generally do not guarantee consistent visual spaces when comparing distinct projections since axes (dimensions) of the original space are not projected into straight lines on visual spaces, and thus cannot be considered equivalent in projected spaces. Some techniques offer a certain degree of control over projection layout, such as fixing control points [26] or selecting the same initialization parameters [25], but often this means a trade-off between local and global distance preservation as they are not designed with this task in mind.

In this context, projections need to be *aligned* as best as possible, i.e., distances between points in projections must be as similar as possible to those in the original data, while also keeping projections as similar as possible with each other. Currently, there are three different approaches to generating aligned projections of multiple feature sets in literature: Dynamic t-SNE (Dt-SNE) [27], Visual Feature Fusion (VFF) [28], and the generic alignment model [7].

Dt-SNE is a variation of t-SNE capable of minimizing data variability in successive projections to a certain degree to observe changes in multiple DNN states. This is achieved by adding a term to t-SNE's cost function to approach points representing the same instance in successive projections close to each other. Although attaining better results if compared to the original t-SNE in terms of aligning subsequent projections, it inherits a major problem: the misleading effect of cluster distance and shape in t-SNE projections resulted from the local nature of optimization and how hyperparameters are set up [29]—the observed distance between clusters is unreliable and sensitive to parametrization (perplexity parameter).

The VFF technique was originally developed to fuse different feature representations of data to build a new, user-driven representation. VFF calculates pairwise distances between data points in all feature sets and generates a new pairwise distance table in which the distance between points x_i, x_j is the mean distance $d_T(x_i, x_j)$ across all feature sets T. These distances are used to generate a guide projection p using a force-based model that minimizes metric stress. The technique then proceeds to generate a projection for each feature set, attracting every point towards its counterpart in p at each iteration.

The generic alignment model is the one adopted in this paper. While t-SNE's global distance preservation is unreliable and VFF's stress minimization makes it perform poorly for visualizing local neighborhoods, the generic model shows an implementation of UMAP [21], which we view as a reasonable compromise. Additionally, being designed for any gradient descent-based projection technique, this model can be useful for further experiments with other techniques.

3.1.1. Aligning Projections

According to the generic alignment model, the alignment process consists of obtaining samples $F[1], ..., F[T]$ from activation sets $A[1]...A[T]$ with the same indexes (i.e., all samples contain data related to the same points) and then calculating projections $P[1], ..., P[T] \in \mathbb{R}^2$ that preserve distance relationships in each set $F[k]$ as best as possible while aligning themselves to one another. This process implies performing an optimization task considering two restrictions. For each feature set k, the cost function to minimize is given by

$$C[k] \;=\; (1-\lambda)\cdot C_{umap}[k] \;+\; \lambda\cdot\gamma\cdot C_{alignment}[k] \tag{1}$$

where $C_{umap}[k]$ is the standard cost function for UMAP (binary cross-entropy), λ is the parameter that controls alignment interference (0.1 for all experiments in this paper), and γ is an internal scaling parameter defined automatically at each iteration. $C_{alignment}[k]$ is the penalty for inter-projection distance, given by

$$C_{alignment}[k] = \frac{1}{2N}\sum_{i=1}^{N} d(P_i[k], \bar{P}_i) \tag{2}$$

where $d(P_{i[t]}, \bar{P}_i)$ is the distance between point P_i in the projection k and the current mean position of the same point between all projections $\bar{P}_i$.

This optimization is done by gradient descent. To generate aligned projections from hidden layer outputs with widely different number of dimensions, pairwise distances need to be normalized. We performed normalization by dividing attributes by the mean instance norm from every feature set. Once aligned projections are generated, we can proceed to the generation of vector fields.

3.2. Vector Field Generation

The vector fields contained in our visualization are generated using an adapted version of the Vector Field K-means technique, proposed by Ferreira et al. [30]. This technique defines a distance measure $d(X, a)$ between a discrete grid representation of a vector field X and a trajectory a, composed by pairs of time stamp and spatial position $(t, a(t))$. Once this distance function is defined, it is minimized to find discrete vector fields that provide the best approximation to a group of trajectories and match each trajectory to a field, similar to the conventional k-means algorithm.

In this approach, a trajectory is represented by a path written as $\alpha : [t_0, t_1] \to \mathbb{R}^2$ and a vector field is defined in a domain $\Omega \subset \mathbb{R}^2$, i.e., a function $X : \Omega \to \mathbb{R}^2$. For all data points, each transition ϕ is considered a path, and time intervals between all steps are presumed equal to $1/T$. Finding the vector field that best matches a set of trajectories Φ can then be described as an optimization problem of minimizing

$$E = \min_X \ \lambda_L \left\| \Delta X \right\|^2 + \frac{(1-\lambda_L)}{N} \sum_{\alpha_i \in \Phi} \int_{t_0^i}^{t_1^i} \left\| X\left(\alpha_i(t)\right) - \left(\alpha'(t)\right) \right\|^2 dt \tag{3}$$

where $\alpha'\ (t)$ is the velocity vector of trajectory α on instant t. The first term is a regularization restriction, where Δ is the Laplace operator. This restriction ensures smoothness of the resulting vector field, with parameter λ_L controlling the weight of each term in the equation and therefore determining if the optimization should prioritize smoothness or matching vector field to trajectories.

In the Vector Field k-Means technique, vector fields are generated from trajectory groups and trajectories are then reassigned to the best matching fields at every iteration. Since our focus is to generate only one vector field that approximates a set of trajectories, we employ a simplified version of this technique that fits a single vector field to the trajectory set, with no need for reassignment.

3.3. Vector Field and Trajectory Visualization

The final step in our approach is to combine all information obtained from (1) projections, (2) trajectories, and (3) vector fields components to generate visual representations. For each of these components, we generate different visual representations that are stacked to convey distinct properties of the NN model under analysis. For data projection, we use the conventional scatterplot metaphor with points representing data instances. Since multiple projections $P[1], P[2], \ldots, P[T]$ are available, we allow users to select any projection to compose the visualization, allowing the inspection of the state of given data instances at any moment in time.

Visualization and analysis of vector fields and spatiotemporal data (either abstract or geographic) are research fields with their own bodies of literature, with several proposed methods to solve different tasks [31–35]. In this paper, we adopt two strategies to visualize vector fields: a simple glyph-based approach, showing flow directions, and streamlines, revealing motion across the whole field. With vector field visualization, users can overview data flow inside a NN model through the analysis of instance group formation and separation.

Finally, for trajectories, we employ an approach similar to the one presented by Rauber et al. [3], using bundling to group similar trajectories reducing visual clutter. However, in our approach, we define control points for bundling along the streamlines extracted from the vector field so that bundling follows the flow of data inside the network and allows analysis of the behavior of individual data instances over time. Figure 2 shows an example of these three layers of visualizations used together to represent evolution of representations in the last layer of a NN model during training. In the next section, we detail how this can be used in practice.

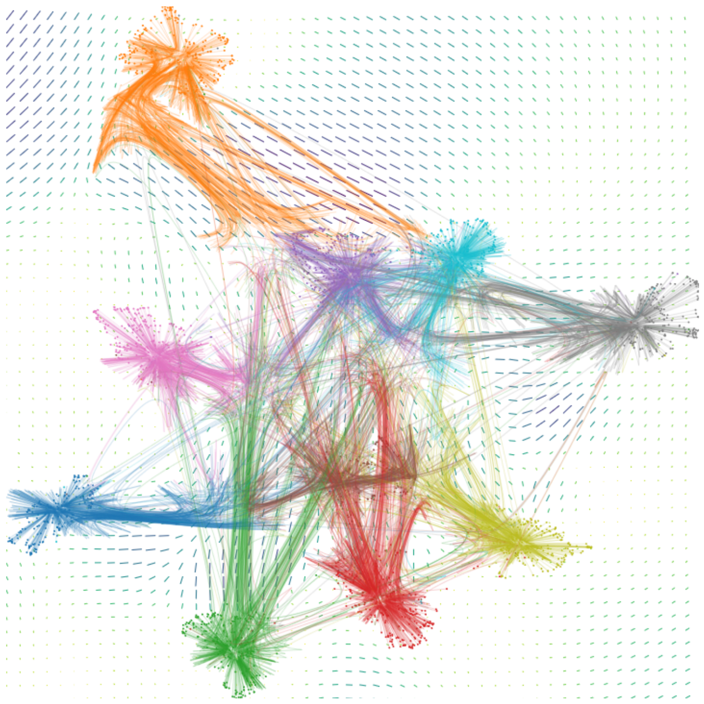

Figure 2. Example of visual representations for projection, trajectory, and vector field components stacked together. The vector field representation overviews the whole flow of the data inside a network, supporting the analysis of instance grouping over time. The scatterplot metaphor, used to represent a projection, enables the analysis of specific instances at a particular moment in time, and trajectory bundling can be used to derive insights about instances' behavior over time. Together, these representations define a powerful visual metaphor, providing abstract and detailed views of the dynamics of neural networks.

4. Results

One typical application of DNNs is data classification, which consists of inferring some model $f : \mathcal{X} \to \mathcal{Y}$ to correctly label unknown data based on a set of known labelled examples $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ [1]. This learning process is known as supervised learning, in which the model is iteratively adapted according to training examples.

Although it is well established in the literature that DNNs yield outstanding results in classification tasks for different domains, training the network and choosing appropriate parameters can still be a complex, time-consuming process. In this section, we visualize the projected activity space on a few NN models performing classification tasks with the objective of gaining insights on the training process and on how conclusions are drawn when performing classification.

It is important to clarify that these visualization methods are not restricted to a specific architecture or application. These models were chosen as to provide informative examples, and because the classification problem offers an easy way to keep track of instances by coloring points representing them using label data. Experiments were conducted using tensorflow(www.tensorflow.org) python libraries, and visualization systems were built using d3.js(d3js.org).

4.1. MNIST on the MLP model

This example shows projected data obtained from a Multilayer Perceptron (MLP) NN model, consisting of four dense layers with 256 processing units each, using ReLU activation. The output is a 10-unit layer with softmax activation to perform classification on the MNIST dataset [36]. Training is done via stochastic gradient descent (SGD), using learning rate = 0.0001 and momentum = 0.9. Output data were gathered from the last hidden layer after 0, 1, 3, and 20 training epochs. Accuracy results increased rather quickly, reaching 0.9191, 0.9533, and 0.9800 on the test set after epochs 1, 3, and 20, respectively. Figure 3 shows projected data and the obtained vector field. The projections show an expansion as the model adapts to classification, and data points move outwards to form groups, in the same way as observed by Rauber et al. [3]. This movement is reflected in the vector field.

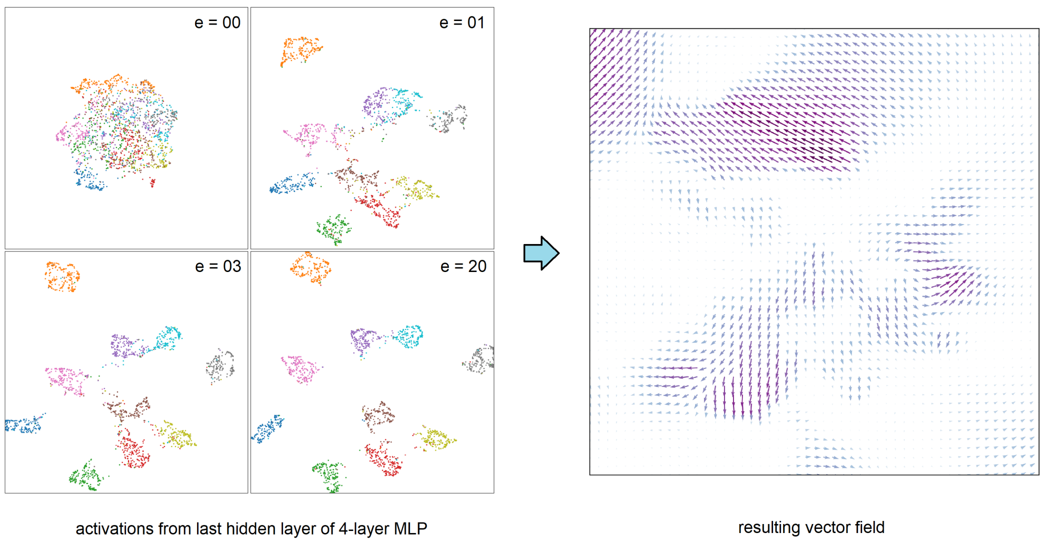

Figure 3. Outputs from the last hidden layer of a 4-layer MLP were gathered during training, after 0, 1, 3, and 20 epochs. Projected data refer to a sample of MNIST test set, with 2000 instances. Instance projected position at each epoch define trajectories that were used to approximate vector fields, as shown on the right side of the figure. A clear expansion pattern can be observed, as instances move outwards to form groups as the network is trained.

An organized flow of instances produces a more intense vector field, while a more disorganized movement implies frequent opposing directions and a muted vector field. This data can be used to display flow between projections: by displaying streamlines, previous movement information can be infused in the visualization without multiple plots or tracking points between projections. Figure 4 shows a projection of the last epoch of training, with streamlines indicating how groups moved to their positions.

While projecting data points over streamlines can offer important insight, additional information can be revealed by directly plotting trajectories. Previous work [3] used trail bundling [37] to reduce visual clutter from multiple trajectories in sequential projections. We defined control points along the generated streamlines that are used to attract nearby trajectories, as can be seen in Figure 5.

The underlying vector field also produces other exploration opportunities: after optimization, there is a resulting error value for each trajectory, which indicates how well it fits to the vector field. This value can be used to explore instances that are not well represented, and identify outliers in the visualization: instances whose movement differs highly to the field surrounding it. Figure 6 shows a highlight of a few trajectories with the highest error, moving against the general flow. Further exploration can be conducted, filtering trajectories and coloring time steps as to understand movement direction and reasoning behind the highlighted behavior.

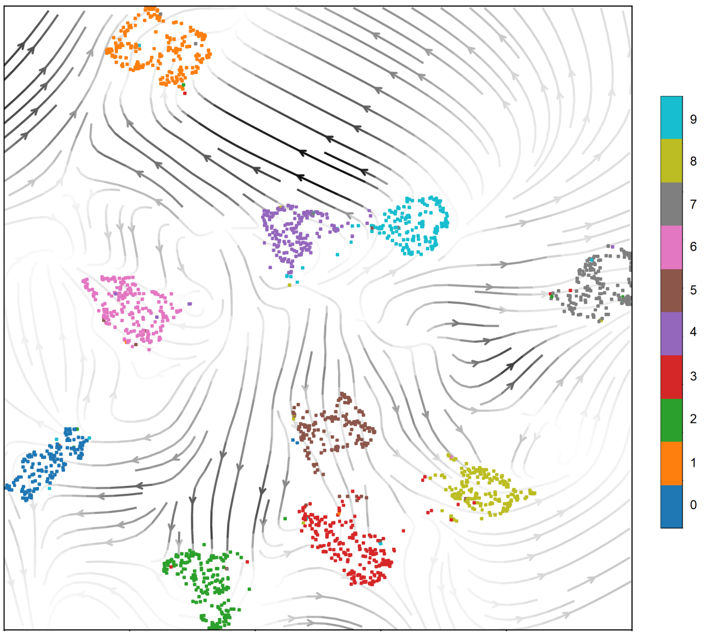

Figure 4. Outputs from the last hidden layer of a 4-layer MLP after 20 training epochs, using MNIST test data input, projected over streamlines. Streamlines are calculated based on three other outputs from the same layer during training, at epochs 0, 1, and 3. Activity between projections is turned into a vector field, which is then used to generate streamlines that offer additional insight into data flow during training without displaying points from other projections. For streamlines, color intensity represents flow speed (L2 norm of vectors) and arrows indicate direction.

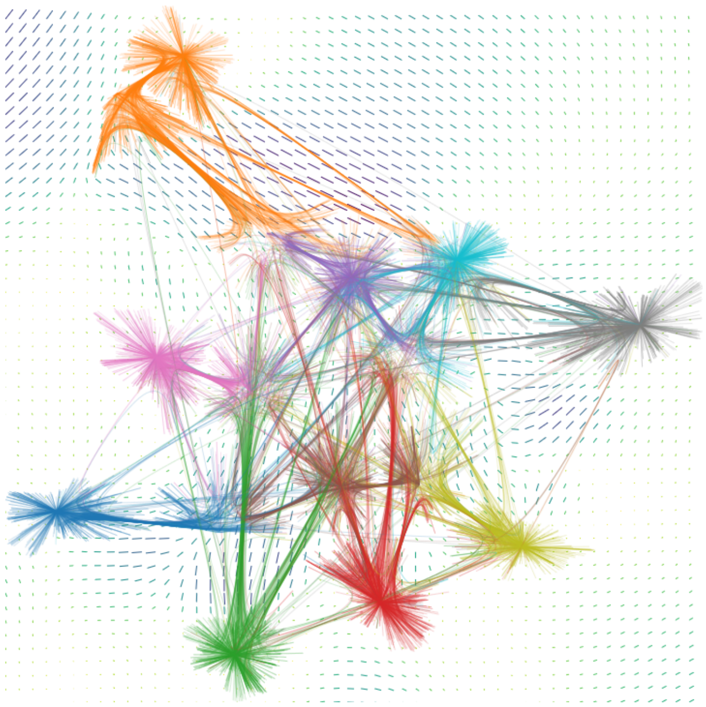

Figure 5. Trajectory data from the same outputs as the Figure 4 (last hidden layer of MLP after 20 epochs using MNIST test data as input). Lines start from instance positions at epoch 0 and end in instance positions at epoch 20. Control points were set along the streamlines defined previously, attracting lines near them into bundles.

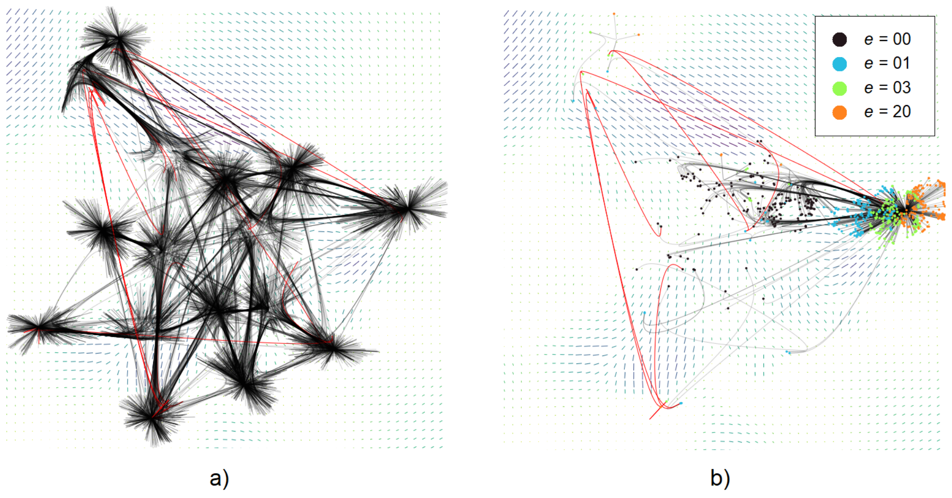

Figure 6. When visualizing trajectories, it is possible to highlight the ones that attained high error values when approximating the vector field, in search of outliers. (**a**) shows trajectories with the error above a certain threshold (0.1) highlighted in red, while all others are shown in black; (**b**) filters the projection to only instances corresponding to label "7", which seem to contain a few outliers. Data points were plotted again and colored according to which feature set they were taken from, so trajectory direction can be better understood. In this case, the highlighted trajectories move through long distances, at times in the exact opposite way of the vector field.

Figure 7 shows further analysis of a highlighted trajectory. A single instance was selected and its path observed. We can quickly notice what resulted in a high error value from vector field optimization: the instance moved rather quickly and often against the overall flow, crossing large distances and changing groups. Using previous projections as reference, we can identify large instance groups in each area this instance moved to after the first time step. This means the instance was deemed similar to these groups at one point in time, and shifted rather quickly towards a different group afterwards. This behavior is particularly interesting since misclassified examples are often projected on fringe regions of groups, indicating output uncertainty, or projected repeatedly within the same group, being resistant to changes. We can then inspect the data instance to search for clues to this behavior.

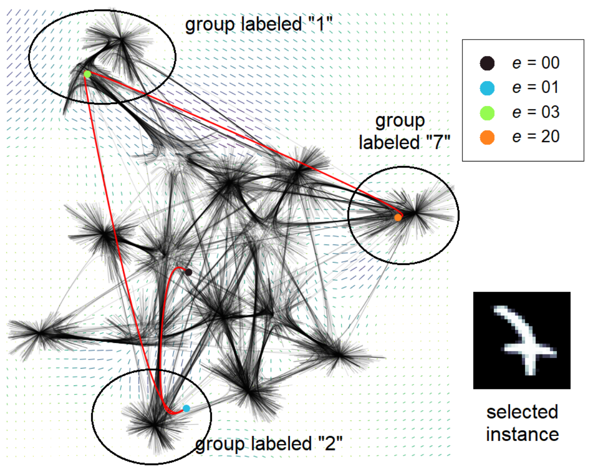

Figure 7. A single outlier trajectory is selected for examination. This data point was at the middle area at epoch 0, moved to an area containing many examples labeled as "2" at epoch 1, moved again to an area containing examples labeled "1" at epoch 3, and finally moved to another area, which contains a group of examples labeled "7", at epoch 20. We can then examine the instance to search for reasons behind this atypical behavior.

4.2. ImageNet on MobileNetV2

This experiment shows our approach applied to a more complex combination of model and data. We used the MobileNetV2 network model [38], pre-trained on Imagenet data [39]. We randomly selected 10 classes (*'clasp'*, *'delphinium'*, *'giant panda'*, *'honker'*, *'ice cream'*, *'lunch'*, *'newfoundland'*, *'outbuilding'*, *'puffin'*, *'rose'*) from the data set and fine-tuned a new top layer to classify them. Input images were resized to 224 × 224 pixels, resulting in a higher number of attributes when gathering intermediate layer outputs when compared to the previous experiment. MobileNetV2 has a depth of over 150 layers, and while it is possible to generate projections for all of them and visualize them as needed, we opted to select the last merging layer from every two network blocks (*'add'* layers), as well as the last global pooling layer, to simplify description. A sample of 1000 data instances from the 10 selected classes was used as input for intermediate layer output projections. All convolutional output arrays were flattened and data were projected using the aligned UMAP model described in Section 3.1.1. Results can be seen in Figure 8.

Our goal with this experiment was to show the forming of abstract representations and knowledge as data flows through the network. While useful information can be gathered by examining projections side by side, this visualization can quickly get out of hand, as larger numbers of projections may need to be generated to understand deeper or more complex models. Figure 9a shows a single projection of the last hidden layer, with streamlines representing previous flow in the background. Certain flow details come to light, such as movement being stronger towards the lower-left side of the image, as the right side contains classes still not fully separated (9 and 2, 5 and 6).

Finally, Figure 9b shows trajectories for this experiment. As the number of projected steps gets higher, the more cluttered the trajectory visualization becomes. This can be solved by dividing trajectories in multiple steps, such as observing how separation in the central section was achieved for the first layers and how expansion was conducted afterwards as two separate images.

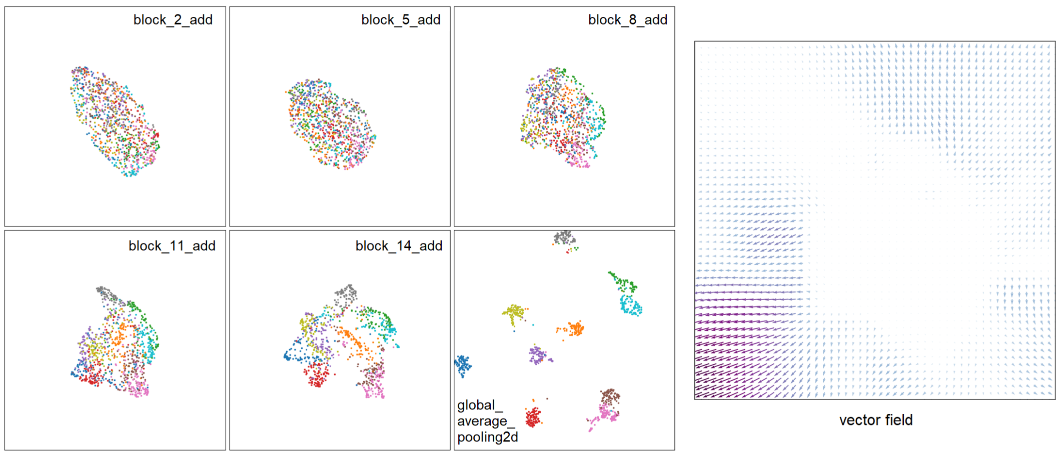

Figure 8. Projected sequential output data from six hidden layers of pre-trained MobileNetV2 using 1000 ImageNet inputs. Projections are shown in left-to-right, upper-to-lower order. It is possible to notice that, while still concentrated in the middle area in the first upper projections, points are slowly separated into classes before finally expanding in the last two projections. On the right side is the generated vector field. A high concentration of movement in all directions in the middle area makes the resulting field rather muted, but its intensity increases in outer areas of the image.

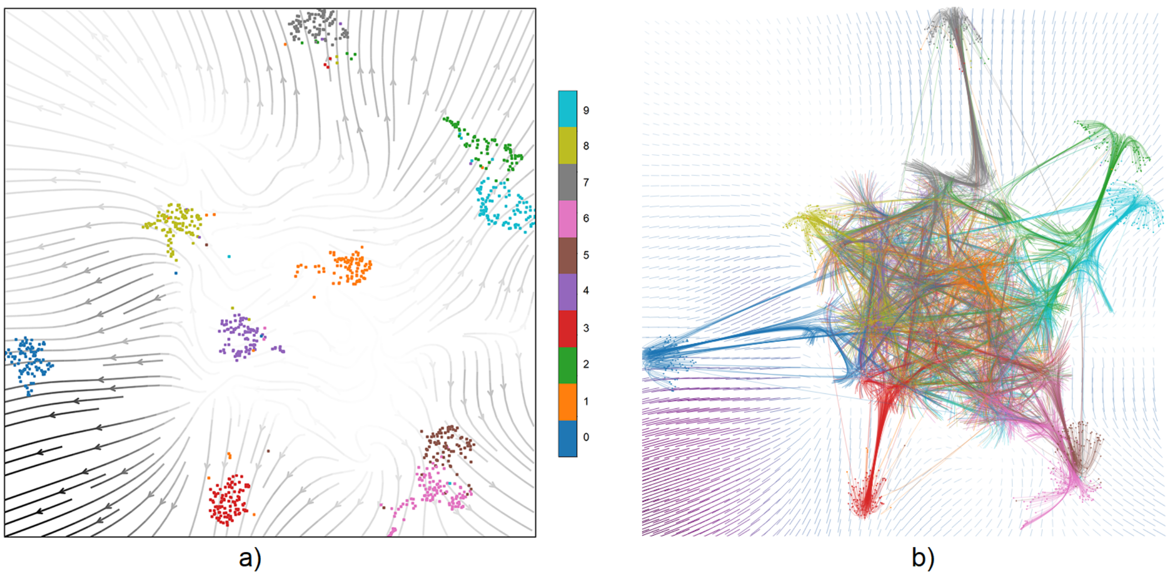

Figure 9. (**a**) projection from the last hidden layer of MobileNetV2 (global average pooling layer) over streamlines from movement trajectories of previous layers. It is possible to see paths that led from a central position to the positions occupied at the current projection, for all points; (**b**) trajectories of points over six layer outputs in the same network model. There is a high amount of clutter in the central section due to the resulting projections of the first layers, but certain movement patterns can be identified. Projected data are obtained using 1000 ImageNet data samples.

5. Limitations

The work described in this paper has certain limitations: one of them is the fact that our projections are forced to be aligned; forcing the layout to keep an aligned state can generate distortions in projected data, especially if the analyzed feature sets offer highly conflicting distance relations or form different structures. Therefore, care must be taken as to not interpret alignment restrictions as patterns in data. The trade-off between alignment and projection fidelity also has a large impact in the generation of vector fields, as an alignment parameter configuration that is too restrictive may result in vector fields that miss too much information, while not being restrictive enough will result in vector fields that show movement that is not relevant. Additionally, outputs from the employed vector field approximation method are not always visually intuitive: for instance, the most intense areas (largest vector norms) in a field will not necessarily be ones with high trajectory activity. Additionally, a single vector field may not be enough to display subtleties of some networks. There is the possibility of generating a layered visualization from multiple vector fields, in order to estimate and explore more complex activity spaces.

6. Conclusions

In this paper, we presented a new approach for projection-based ANN hidden layer visualization that uses trajectories and vector fields to provide insights on how knowledge is generated in a DNN through training and how abstract representations are formed between layers. The presented work offers a flow-based model to represent a transition space between a sequence of projections in order to remove point-based clutter, and incorporate this model into analysis to discover useful information about model and data. Observing trajectories between layers can help understand the importance of each layer in separating certain instances in the data set, and object movement can highlight the effects of a certain training period. Exploring movement throughout the sequence can highlight objects that the model finds difficult to analyze, or objects that are not being perceived in an expected manner.

This approach is not limited to ANN visualization, since any sequence of multidimensional projections can be explored in this manner, such as projecting any pipeline that processes data in multiple steps. The creation of a visual space that ties projections together may provide support to other visualization methods in the future.

We performed experiments that aim to show the usefulness of our method, and how it may lead to knowledge and model improvement. Our analysis was able to go further in certain aspects of the training process of neural networks, attempting to explain subtle aspects of how knowledge is generated in a DNN system.

Future research directions include investigating how projected vector fields relate to other aspects of neural network interpretation methods, such as filter visualization in CNNs, or recurrent models: almost any layer output from ANNs can be projected, and all differences between a sequence of outputs should have an effect on the resulting vector field. Work can also be conducted in finding correlations between movement observed in vector fields and features built by the network, such as identifying neurons (or convolutional filters) that are the most responsible for generating certain movement patterns in the vector field visualization.

Author Contributions: Conceptualization, G.D.C. and F.V.P.; data curation, G.D.C.; investigation, G.D.C.; methodology, G.D.C.; supervision, F.V.P.; validation, E.E.; visualization, G.D.C.; writing—original draft, G.D.C.; writing—review & editing, E.E. and F.V.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: We would like to thank CAPES and FAPESP (2017/08817-7, 2015/08118-6) for the financial support. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436.
2. Nonato, L.G.; Aupetit, M. Multidimensional Projection for Visual Analytics: Linking Techniques with Distortions, Tasks, and Layout Enrichment. *IEEE Trans. Vis. Comput. Graphics* **2018**, *25*, 2650–2673. doi:10.1109/TVCG.2018.2846735.
3. Rauber, P.E.; Fadel, S.G.; Falcao, A.X.; Telea, A.C. Visualizing the hidden activity of artificial neural networks. *IEEE Trans. Vis. Comput. Graphics* **2017**, *23*, 101–110.
4. Mahendran, A.; Vedaldi, A. Understanding deep image representations by inverting them. In Proceedings of the IEEE conference on computer vision and pattern recognition, Boston, MA, USA, 7–12 June 2015; pp. 5188–5196.
5. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
6. Cantareira, G.D.; Paulovich., F.V.; Etemad., E. Visualizing Learning Space in Neural Network Hidden Layers. In Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, Valetta, Malta, 27–29 Feburary 2020; pp. 110–121. doi:10.5220/0009168901100121.
7. Cantareira, G.D.; Paulovich, F.V. A Generic Model for Projection Alignment Applied to Neural Network Visualization. EuroVis Workshop on Visual Analytics (EuroVA); Turkay, C.; Vrotsou, K., Eds. The Eurographics Association: Aire-la-Ville, Switzerland, 2020. doi:10.2312/eurova.20201089.
8. Scherer, D.; Müller, A.; Behnke, S. Evaluation of pooling operations in convolutional architectures for object recognition. In *Proceedings of the International Conference on Artificial Neural Networks*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 92–101.
9. Hohman, F.M.; Kahng, M.; Pienta, R.; Chau, D.H. Visual Analytics in Deep Learning: An Interrogative Survey for the Next Frontiers. *IEEE Trans. Vis. Comput. Graphics* **2018**, *25*, 2674–2693. doi:10.1109/TVCG.2018.2843369.
10. Zeiler, M.D.; Fergus, R. Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision*; Springer: Cham, Switzerland, 2014; pp. 818–833.

11. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, Boston, MA, USA, 7–12 June 2015; pp. 1–9.
12. Simonyan, K.; Vedaldi, A.; Zisserman, A. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv* **2013**. Available online: https://arxiv.org/abs/1312.6034 (accessed on 30 August 2020).
13. Zintgraf, L.M.; Cohen, T.S.; Adel, T.; Welling, M. Visualizing deep neural network decisions: Prediction difference analysis. *arXiv* **2017**. Available online: https://arxiv.org/abs/1702.04595 (accessed on 30 August 2020).
14. Erhan, D.; Bengio, Y.; Courville, A.; Vincent, P. Visualizing higher-layer features of a deep network. *Univ. Montr.* **2009**, *1341*, 1.
15. Mahendran, A.; Vedaldi, A. Visualizing deep convolutional neural networks using natural pre-images. *Int. J. Comput. Vis.* **2016**, *120*, 233–255.
16. Babiker, H.K.B.; Goebel, R. An Introduction to Deep Visual Explanation. *arXiv* **2017**. Available online: https://arxiv.org/abs/1711.09482 (accessed on 30 August 2020).
17. Liu, M.; Shi, J.; Li, Z.; Li, C.; Zhu, J.; Liu, S. Towards better analysis of deep convolutional neural networks. *IEEE Trans. Vis. Comput. Graphics* **2017**, *23*, 91–100.
18. Yosinski, J.; Clune, J.; Nguyen, A.; Fuchs, T.; Lipson, H. Understanding neural networks through deep visualization. *arXiv* **2015**. Available online: https://arxiv.org/abs/1506.06579 (accessed on 30 August 2020).
19. Pezzotti, N.; Höllt, T.; Van Gemert, J.; Lelieveldt, B.P.; Eisemann, E.; Vilanova, A. Deepeyes: Progressive visual analytics for designing deep neural networks. *IEEE Trans. Vis. Comput. Graphics* **2018**, *24*, 98–108.
20. Kahng, M.; Andrews, P.Y.; Kalro, A.; Chau, D.H.P. A cti V is: Visual Exploration of Industry-Scale Deep Neural Network Models. *IEEE Trans. Vis. Comput. Graphics* **2018**, *24*, 88–97.
21. McInnes, L.; Healy, J.; Melville, J. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv* **2018**. Available online: https://arxiv.org/abs/1802.03426 (accessed on 30 August 2020).
22. Donahue, J.; Jia, Y.; Vinyals, O.; Hoffman, J.; Zhang, N.; Tzeng, E.; Darrell, T. Decaf: A deep convolutional activation feature for generic visual recognition. In Proceedings of the International conference on machine learning, Beijing, China, 21–26 June 2014; pp. 647–655.
23. Hamel, P.; Eck, D. Learning features from music audio with deep belief networks. *ISMIR* **2010**, *10*, 339–344.
24. Mohamed, A.r.; Hinton, G.; Penn, G. Understanding how deep belief networks perform acoustic modelling. *Neural Netw.* **2012**, 6–9. doi:10.1109/ICASSP.2012.6288863.
25. Van Der Maaten, L.; Hinton, G. Visualizing high-dimensional data using t-sne. journal of machine learning research. *J Mach Learn Res* **2008**, *9*, 26.
26. Joia, P.; Coimbra, D.; Cuminato, J.A.; Paulovich, F.V.; Nonato, L.G. Local affine multidimensional projection. *IEEE Trans. Vis. Comput. Graphics* **2011**, *17*, 2563–2571.
27. Rauber, P.E.; Falcão, A.X.; Telea, A.C. Visualizing time-dependent data using dynamic t-SNE. In *Eurographics/IEEE VGTC Conference on Visualization*; Eurographics Association: Goslar, Germany, 2016; pp. 73–77.
28. Hilasaca, G.; Paulovich, F. Visual Feature Fusion and its Application to Support Unsupervised Clustering Tasks. *arXiv* **2019**. Available online: https://arxiv.org/abs/1901.05556 (accessed on 30 August 2020).
29. Wattenberg, M.; Viégas, F.; Johnson, I. How to use t-sne effectively. *Distill* **2016**, *1*, e2.
30. Ferreira, N.; Klosowski, J.T.; Scheidegger, C.E.; Silva, C.T. Vector Field k-Means: Clustering Trajectories by Fitting Multiple Vector Fields. *Comput. Graph. Forum.* **2013**, *32*, 201–210.
31. Wang, W.; Wang, W.; Li, S. From numerics to combinatorics: a survey of topological methods for vector field visualization. *J. Vis.* **2016**, *19*, 727–752.
32. Sanna, A.; Montuschi, P.; Montuschi, P. A Survey on Visualization of Vector Fields By Texture-Based Methods. *Devel. Pattern Rec.* **2000**, *1*, 2000.
33. Peng, Z.; Laramee, R.S. Higher Dimensional Vector Field Visualization: A Survey. *TPCG* **2009**, 149–163. doi:10.2312/LocalChapterEvents/TPCG/TPCG09/149-163.
34. Bian, J.; Tian, D.; Tang, Y.; Tao, D. A survey on trajectory clustering analysis. *arXiv* **2018**. Available online: https://arxiv.org/abs/1802.06971 (accessed on 30 August 2020).
35. Kong, X.; Li, M.; Ma, K.; Tian, K.; Wang, M.; Ning, Z.; Xia, F. Big Trajectory Data: A Survey of Applications and Services. *IEEE Access* **2018**, *6*, 58295–58306.

36. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-Based Learning Applied to Document Recognition. *Proc. IEEE* **1998**, *86*, 2278–2324.
37. Hurter, C.; Ersoy, O.; Fabrikant, S.I.; Klein, T.R.; Telea, A.C. Bundled visualization of dynamicgraph and trail data. *IEEE Trans. Vis. Comput. Graphics* **2013**, *20*, 1141–1157.
38. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 4510–4520.
39. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In Proceedings of the 2009 IEEE conference on computer vision and pattern recognition, Miami, FL, USA, 20–25 June 2009; pp. 248–255.