Volume 0 (1981), Number 0 pp. 1–4 COMPUTER GRAPHICS *forum*

TopoAct: Visually Exploring the Shape of Activations in Deep Learning: Supplementary Material

Archit Rathore[1], Nithin Chalapathi[1], Sourabh Palande[1], Bei Wang[1]

[1] School of Computing, Scientific Computing and Imaging (SCI) Institute, University of Utah, USA

1. TopoAct User Interface and System Design

We provide details regarding the user interface and system design of **TopoAct**. Figure 3 in the main paper illustrates the user interface under single-layer exploration mode.

The control panel includes information regarding the layer of choice (e.g., *3a*, *3b*, *4a*), the dataset (across various mapper parameters) under exploration (e.g., *overlap-30-epsilon-fixed*, *overlap-50-epsilon-adaptive*), and a class search box that supports filtering by a set of classes. It enables projections of the activation vectors using t-SNE and UMAP. The control panel also contains a check box that superimposes averaged activation images over the graph nodes to provide an alternative overview of the topological summary (see feature visualization panel for details). It also supports the filtering of graph edges based on the Jaccard index.

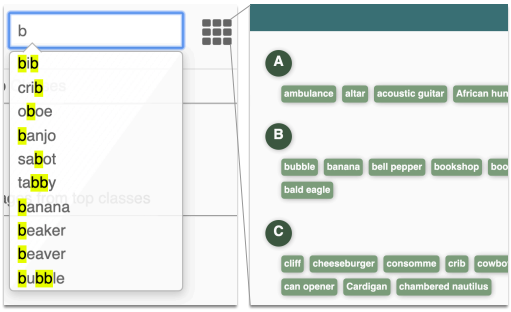

Figure 1: *Class search box used to specify a set of classes to be filtered by the mapper graph.*

Class search box with a shopping directory view. As illustrated in Figure 1, users can type a class name in the search box, which is used to filter the mapper graph. The search bar uses partial matching to locate a list of possible class names. Alternatively, users can select a subset of classes from the "shopping directory" view in which top classes within the current layer are listed in alphabetical order. The mapper graph will highlight the clusters that contain any of the user-specified classes among their top three classes.

When the projection view is enabled, class search will also highlight all activations for that class in the t-SNE/UMAP projection. As an example, in Figure 3, we look at t-SNE projection

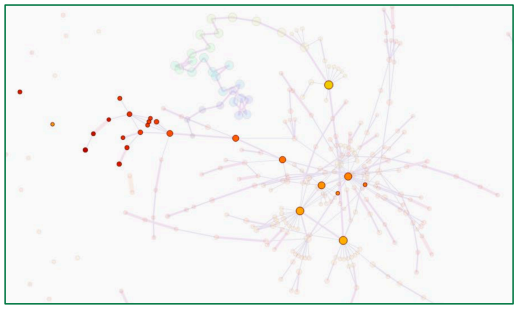

Figure 2: *A mapper graph highlighting nodes that include classes of large motor vehicles.*

of activations from layer *5a* of the ImageNet dataset (*overlap-30-epsilon-adaptive*). Using the shopping directory view, we select several classes of large motor vehicles, for example, **school bus**, **tow truck**, **fire engine**, **minibus**, **minivan**, etc. Each node highlighted in the mapper graph of Figure 2 contains at least one of the selected classes among its top three classes.

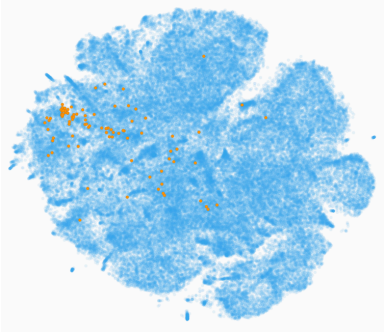

Figure 3: *Class search in the projection view for the **lifeboat** class. Users can search one or more classes and visualize them in the t-SNE or UMAP projection.*

1.1. Single Layer Exploration Mode

For single layer exploration, the interface is composed of three panels: the mapper graph panel, the data example panel, and the feature

visualization panel (see Figure 3 in the main paper for an illustration).

Mapper graph panel. For ImageNet dataset, **TopoAct** uses the mapper construction to construct a topological summary from the activation vectors of $300K$ images across $1K$ classes. Different from dimensionality reduction approaches such as t-SNE [MH08] and UMAP [MHM18], **TopoAct** computes and captures the shape of the activation space in the original high-dimensional space in the form of a mapper graph and preserves the structural information as much as possible when the mapper graph is drawn on the 2-dimensional plane.

As shown in Figure 3(a) in the main paper, we use a force-directed layout by Dwyer [Dwy09] to visualize the mapper graph. Each node represents a cluster of "similar" activation vectors (in terms of their proximities in Euclidean distance), and each edge encodes the relations between clusters of activation vectors. Given two clusters of activation vectors C_u and C_v, an edge uv connects them if $|C_u \cap C_v| \neq \emptyset$. Given C_u and C_v connected by an edge uv, the edge weight of uv is their Jaccard Index, that is, $J(C_u, C_v) := |C_u \cap C_v| / |C_u \cup C_v|$. Each edge is then visualized by visual encodings (i.e., thickness and colormap) that scale proportionally with respect to their weights. Weights on the edges highlight the strength of relations between clusters.

To explore the mapper graph, users can zoom and pan within the panel. Hovering over a node in the mapper graph will display simple statistics of the cluster: the number of activation vectors in the cluster and the averaged lens function value. Clicking on a node will give information on the top three classes (with a membership percentage) within the selected cluster; it will also update the selection for the data example panel and the feature visualization panel, as described below.

Data example panel. To make each cluster more interpretable, we combine the original data examples with feature visualization. For a selected node (cluster) in the mapper graph, we give a textual description of the top three classes in the cluster as well as five data examples from each of the three top classes. For example, as illustrated in Figure 4a, a selected cluster in the mapper graph view for layer *5a* (of *overlap-30-epsilon-adaptive*) contains the three top classes of images: **fire engine**, **tow truck**, and **electric locomotive**. Its corresponding data example view contains five images sampled from each class to give a concrete depiction of the input images that trigger the activations.

Feature visualization panel. After a user selects a node (cluster) in the mapper graph panel, we display activation images pregenerated for each input image from the data example panel. These individual activation images are generated by applying feature visualization to individual activation vectors from the $300K$ input images. The feature visualization displays up to 15 of such individual activation images, up to 5 for each of the top classes; see Figure 4(b). Furthermore, we also average the activation vectors that fall within the cluster and run feature inversion on the averaged activation, producing an *averaged activation image* per cluster, as shown in Figure 4(c). Moving across clusters following edges of the mapper graph will help us understand how the averaged activation images vary across clusters. We obtain a global understanding of not only what the network "sees" via these idealized images but also how these idealized images are related to each other in the space of activations.

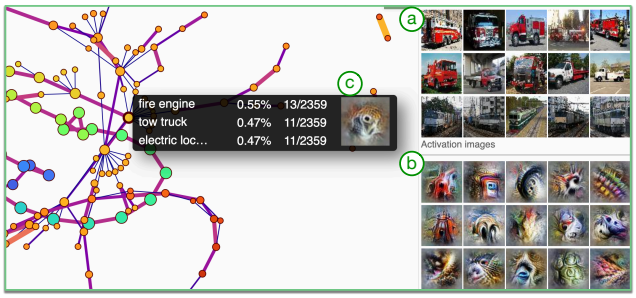

Figure 4: *A data example panel (a) and a feature visualization panel (b) for layer 5a, where (c) contains an averaged activation image for the chosen cluster.*

In addition to the graph view, we can replace each node in the mapper graph by an averaged activation image as a glyph. This can be perceived as an alternative to the *activation atlas* [CAS*19] with one crucial difference: the mapper graph captures clusters of activation vectors in their original high-dimensional space and preserves relations between these clusters. Such a global view provides valuable insights during in-depth explorations.

t-SNE and UMAP projections. For comparative purpose, we perform dimensionality reduction on the activation vectors for each layer using t-SNE and UMAP. The projection is done using all 300K activation vectors onto a 2-dimensional space. For t-SNE, we use the Multicore-TSNE [Uly16] Python library and set perplexity to be 50 following the parameter choice used in the activation atlas [CAS*19]. The UMAP projection is performed using its official Python implementation [MHSG18] with 20 nearest neighbors and a minimum distance of 0.01. t-SNE and UMAP projections are precomputed due to the large number (300K) of activation vectors. We also provide a linked view between the mapper graph and the t-SNE/UMAP projection. Selecting a node in the mapper graph will highlight its corresponding activation vectors in the t-SNE/UMAP projections. We provide subsampled versions of these projections (5K, 10k, 50K, 100K, and 300K) to deal with the issue of visual clutter and to accommodate various browser rendering capabilities on a number of devices.

1.2. Multilayer Exploration Mode

In the multilayer exploration mode, three adjacent layers are explored side by side; see Figure 6(top). After choosing a particular class or a set of classes using the class search box, **TopoAct** highlights nodes (clusters) across all three layers that contain the chosen set of classes among its top three classes. Other visualization features are inherited from the single layer exploration. Multilayer exploration helps capture the evolution of classes as images are run through the network and supports structural comparisons of summaries across layers. Such exploration can be particularly useful when used in conjunction with the class search tool. As an example of a class search in multilayer mode, we look at layers *4e*, *5a* and *5b*

of the *overlap-30-epsilon-adaptive* dataset. We use the same selection of classes of large motor vehicles used in the earlier example of a class search in single layer mode (Figure 2). Figure 5 shows the class search results, now in the multilayer exploration mode.

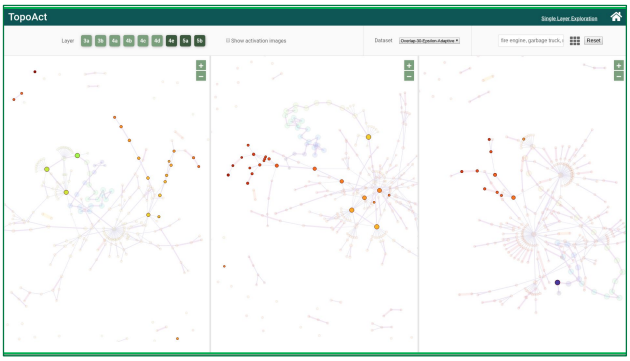

Figure 5: *Class search highlights nodes that include classes of large motor vehicles across multiple layers.*

Under the multilayer exploration mode, we can compare the shape of activation spaces across multiple layers. As illustrated in Figure 6, we show a side-by-side comparison of all layers for the ImageNet dataset (*overlap-30-epsilon-adaptive*). A further investigation into structural comparisons across layers, such as tracking the evolution of a particular branching node, is nontrivial and left for future work.

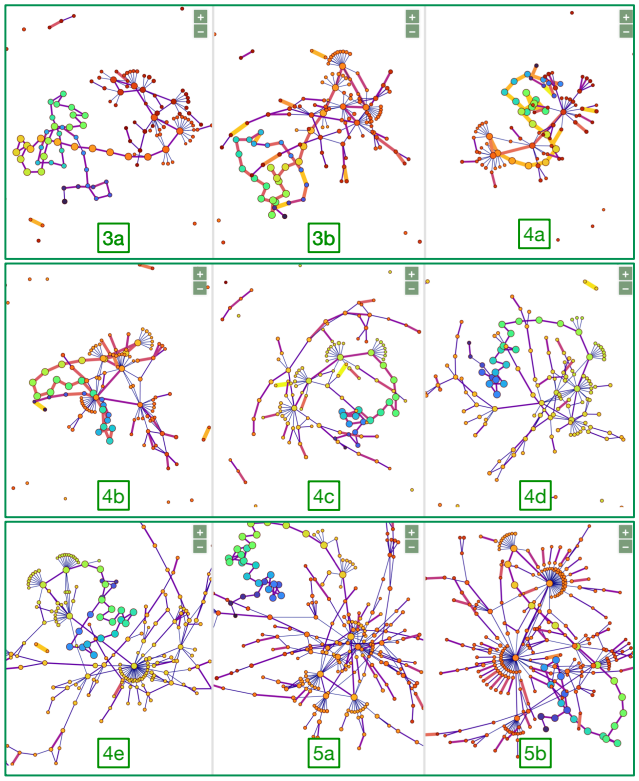

Figure 6: *Comparing nine mapper graphs for the ImageNet dataset using multilayer exploration. Configuration:* 70 *intervals,* 30% *overlap, adaptive* ε *for DBSCAN.*

1.3. System Design

TopoAct is open-source via GitHub: https://github.com/tdavislab/TopoAct/, and web-based with a public demo: https://tdavislab.github.io/TopoAct/. It is tested for Google Chrome and Mozilla Firefox. It is developed using Javascript, HTML, and CSS, along with D3.js and Chart.js. The 300*K* dataset examples were sampled from ImageNet dataset. For our mapper graph construction, we used a modified version [ZCR*20] of the open-source KeplerMapper library [vVS19] that we optimized to handle the large number of data points that we encountered in our use case. The construction of mapper graphs across layers was performed on high-performance server machines with 128, 160, and 256 CPU cores, and RAM ranging from 504 GB to 1024 GB. The construction took around 15 minutes for layers with lower dimensional activation vectors (i.e., layer *3a* produces 256-dimensional activation vectors) and 25-30 minutes for higher dimensional activation vectors (e.g., layer *5b* produces 1024-dimensional activation vectors). For our choice of ε for the DBSCAN algorithm, we ran PyNNDescent [McI] on a commodity workstation with a 4 core intel i7 (4750HQ) and 8GB of RAM. Computing ε took on average 5 minutes per layer. Finally, we used Google Colab [Bis19, Goo] to run our feature visualization with GPUs, either from an Nvidia P100, Nvidia K80, or Nvidia T4 GPU. Feature visualization of all 300K input images was done via the Lucid library [Ten], which took on average 8 hours. Feature visualization of average activation vectors took between 2.5 (i.e., *3a*) and 6 hours (i.e., *5b*) per mapper graph.

2. L_2 Norm and Adaptive Cover

In the demo, we used a uniform cover, which caused large variations in cluster sizes. Although some clusters were composed of only a handful of activation vectors, several clusters had thousands of activation vectors, and large intersections between neighboring clusters. Finding meaningful relationships across such large clusters is difficult in these cases since the top three classes may not be good representatives of the cluster as a whole.

The branches and loops explored in our examples contain relatively small clusters for which the averaged activation images are more meaningful. The best way to remedy the large variation in cluster sizes is to use an adaptive cover, in which interval lengths are modified in such a way that each interval contains approximately the same number of points. Creating adaptive cover elements may be achieved by looking at the distribution of lens function values using histograms. We now discuss this in more detail.

In general, vectors with a dimension as high as the ones from a neural network (maximum of 1024 dimensions in our case) tend to suffer from the curse of dimensionality, which implies that in very high dimensions, the Euclidean metric or the L_2 norm does not exhibit variation - all distances and norms look the same.

Figure 7 shows the distribution of L_2 norms for all layers in the Inception architecture. Notice that the distribution is bell-shaped with long tails, and the variance of the distribution is reasonably large. The severity of the curse of dimensionality may be reduced by using an adaptive cover that has more intervals in the denser regions of lens function. The resulting mapper graphs with such an

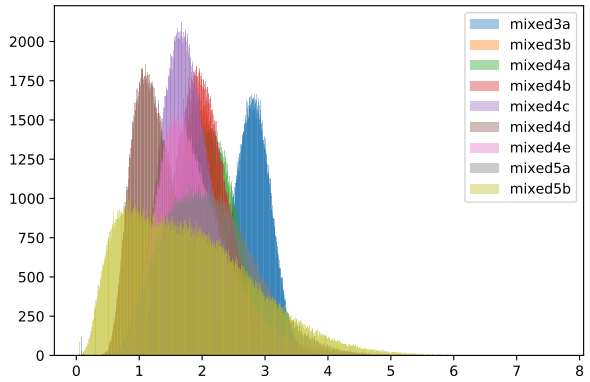

Figure 7: *L_2 norms of activation vectors across all layers.*

adaptive cover will contain nodes of comparable sizes. This is left for future work.

References

[Bis19] BISONG E.: Google colaboratory. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Apress, Berkeley, CA, 2019. 3

[CAS*19] CARTER S., ARMSTRONG Z., SCHUBERT L., JOHNSON I., OLAH C.: Activation Atlas. *Distill 4*, 3 (2019), e15. 2

[Dwy09] DWYER T.: Scalable, versatile and simple constrained graph layout. *Computer Graphics Forum 28*, 3 (2009), 991–998. 2

[Goo] GOOGLE RESEARCH: Google colab. https://colab.research.google.com. 3

[McI] MCINNES L.: PyNNDescent. https://github.com/lmcinnes/pynndescent. 3

[MH08] MAATEN L. V. D., HINTON G.: Visualizing data using t-SNE. *Journal of Machine Learning Research 9* (2008), 2579–2605. 2

[MHM18] MCINNES L., HEALY J., MELVILLE J.: UMAP: Uniform manifold approximation and projection for dimension reduction. arXiv:1802.03426, 2018. 2

[MHSG18] MCINNES L., HEALY J., SAUL N., GROSSBERGER L.: UMAP: Uniform manifold approximation and projection. *Journal of Open Source Software 3*, 29 (2018), 861. 2

[Ten] TENSORFLOW: Lucid library. https://github.com/tensorflow/lucid. 3

[Uly16] ULYANOV D.: Multicore-TSNE. https://github.com/DmitryUlyanov/Multicore-TSNE, 2016. 2

[vVS19] VAN VEEN H. J., SAUL N.: KeplerMapper. http://doi.org/10.5281/zenodo.1054444, Jan 2019. 3

[ZCR*20] ZHOU Y., CHALAPATHI N., RATHORE A., ZHAO Y., WANG B.: Mapper Interactive: A scalable, extendable, and interactive toolbox for the visual exploration of high-dimensional data. arXiv:2011.03209, 2020. 3