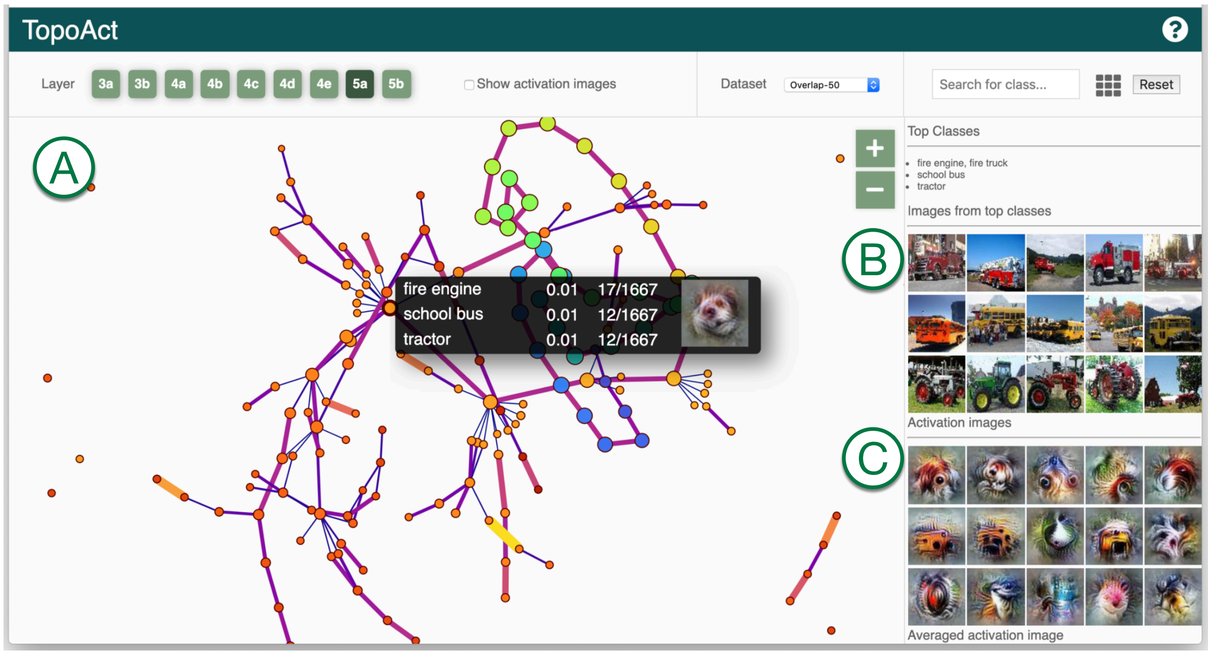
TopoAct: Visually Exploring the Shape of Activations in Deep Learning Supplementary Material

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



fire engine	0.01	17/1667
school bus	0.01	12/1667
tractor	0.01	12/1667

Top Classes

- fire engine, fire truck
- school bus
- tractor

Figure 1: *With **TopoAct**, users can interactively explore topological summaries of activations in a neural network for a single layer and across multiple layers. Users can investigate activations at a particular layer under the single layer exploration mode. (A) The mapper graph view provides a graph-based topological summary of the activation vectors from 300,000 images across 1,000 classes, where each node of the mapper graph represents a cluster of activation vectors and each edge encodes the relationships between the clusters. For a chosen cluster in the summary graph, data example view (B) gives textual description of the top three classes within the cluster together with five image examples from each top class. The feature visualization view (C) applies feature inversion to generate idealized images (called activation images) for individual activation vectors (obtained from data examples) as well as an averaged activation vector within a chosen cluster.*

1. TopoAct User Interface and System Design

We describe details regarding the user interface and system design of **TopoAct**. Figure 1 illustrates the user interface under single layer exploration mode.

The header 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. The header 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 View for details).

Class search box with a shopping directory view. As illustrated in Figure 2, users can type a class name in the search box which is used to filter the mapper graph. The search algorithm uses par-

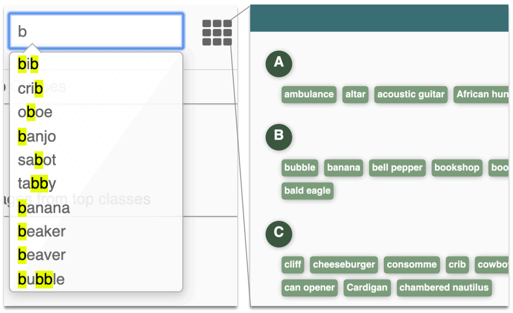


Figure 2: *Class search box used to specify a set of classes to be filtered by the mapper graph.*

tial 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. As an example, we look at layer *5a* of the *overlap-30-epsilon-adaptive* dataset. 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 of the nodes highlighted in the mapper graph of Figure 3 contains at least one of the selected classes among its top three classes.

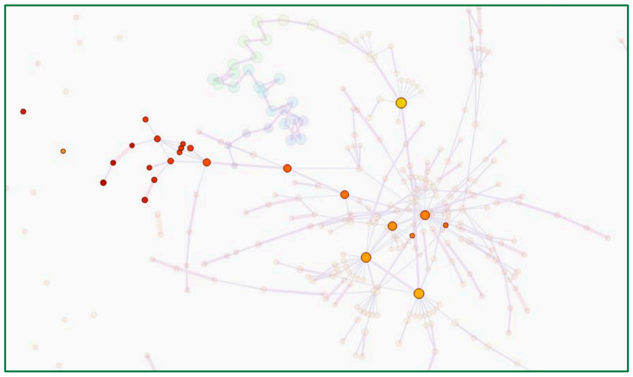


Figure 3: *A mapper graph highlighting nodes that include classes of large motor vehicles.*

1.1. Single Layer Exploration Mode

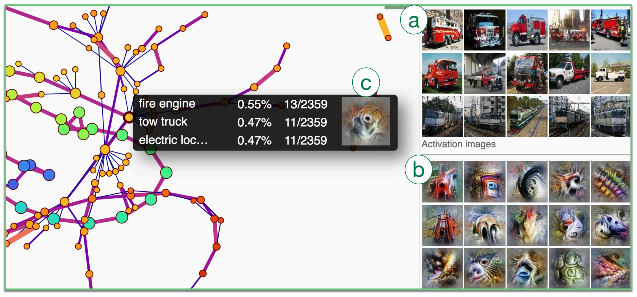
For single layer exploration, the interface is composed of three views: the mapper graph view, the data example view and the feature visualization view, see Figure 1 for an illustration.

Mapper graph view. TopoAct uses the mapper construction to construct a topological summary from the activation vectors of 300,000 images across 1,000 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 screen.

As shown in Figure 1(A), we use force-directed layout by Dwyer [Dwy09] to visualize the mapper graph. Each node represents a cluster of “similar" (in a Euclidean distance sense) activation vectors; and each edge encodes the relations between clusters of activation vectors. Given two clusters of activation vectors C_u and C_v, there is an edge uv connecting 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 view. Hovering over a node in the mapper graph displays simple statistics of the cluster: number of activation vectors in the cluster and 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 view and the feature visualization view, as described below.

Data example view. 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 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.



```
fire engine      0.55%   13/2359
tow truck        0.47%   11/2359
electric loc...  0.47%   11/2359
```

Figure 4: *A data example view (a) and a feature visualization view (b) for layer* 5a, overlap-30-epsilon-adaptive*, where (c) contains an averaged activation image for the chosen cluster.*

Feature visualization view. After a user selects a node (cluster) in the mapper graph view, we display activation images pre-generated for each input image from the data example view. These individual activation images are generated by applying feature visualization to individual activation vectors from the 300,000 input images. The feature visualization displays up to 15 of such individual activation images, up to 5 for each of the top class, see Figure 4b. 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 4c). 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.

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 2D space. For t-SNE, we set perplexity to be 50 following the parameter choice used in the Activation Atlas [CAS*19] and implemented using the Multicore-TSNE [Uly16] Python library. UMAP projection is performed using the official python implementation [MHSG18] with 20 nearest neighbors and a minimum distance of 0.01. Due to the large size (300K) of activation vectors, t-SNE and UMAP projections are precomputed. We also provide 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 browser's rendering capabilities on a number of devices.

1.2. Multilayer Exploration Mode

In 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 run through the network and supports structural comparisons of summaries across layers. This can be particularly useful when used in conjunction with the class search tool. As an example of 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 class search in single layer mode (Figure 3). Figure 5 shows the class search results, now in the multilayer exploration mode.

1.3. System Design

TopoAct is web-based with a public demo available via GitHub [†]. It is tested for Google Chrome and Mozilla Firefox. It was developed using Javascript, HTML, and CSS, together with D3.js[‡].

[†] https://architrathore.github.io/TopoAct-v2.1/
[‡] D3.js: https://d3js.org/

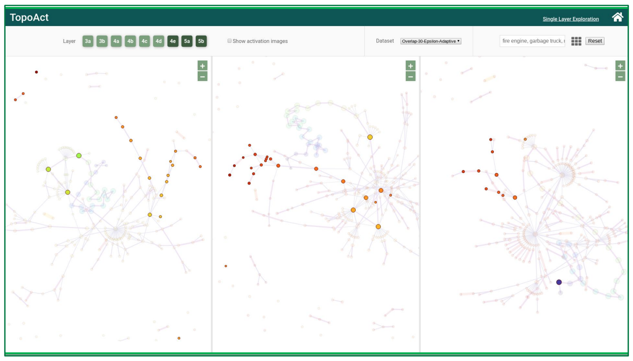


Figure 5: *Class search highlights nodes that include classes of large motor vehicles across multiple layers.*

The 300,000 dataset examples were sampled from ImageNet with reduced resolution. For our mapper graph construction, we used a modified version of the open sourced Kepler Mapper library [vVS19] that we optimized to handle large number of data points that we encounter in our use case. The construction of mapper graphs across layers were 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[§] 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 [¶] to run our feature visualization with GPUs, either from an Nvidia P100, Nvidia K80, or Nvidia T4 GPU. Feature visualization of all 300,000 input images was done via the Lucid library [‖], which took on average 8 hours. Feature visualization of average activation vectors took between 2.5 (i.e., *3a*) to 6 hours (i.e., *5b*) per mapper graph.

2. Multilayer Comparison of Mapper Graphs: InceptionV1

We can compare the shape of activation spaces across multiple layers. As illustrated in Figure 6, we show side-by-side comparison of all layers for the dataset *overlap-30-epsilon-adaptive*. There are several observable trends. First, there are more high-degree branching nodes at deeper layers (i.e. 5b), indicating more specialized differentiation among activation vectors, and consequently, among learned features. Second, there are more "islands", that is, small isolated components at deeper layers, indicating further separation among activation vectors and the corresponding learned features at those layers. 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.

[§] https://github.com/lmcinnes/pynndescent
[¶] https://colab.research.google.com
[‖] Lucid: https://github.com/tensorflow/lucid

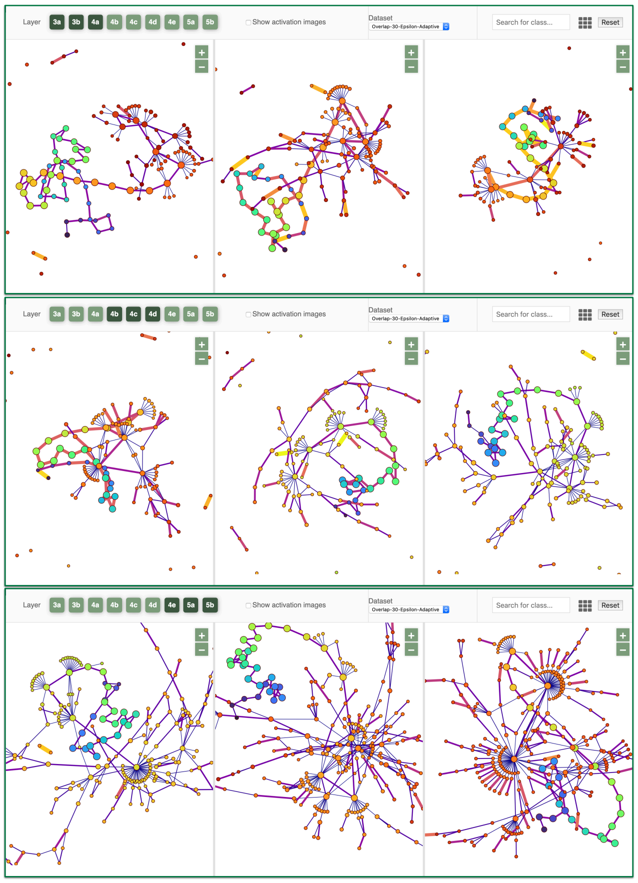


Figure 6: *Comparing 9 mapper graphs for* overlap-30-epsilon-adaptive *dataset using multilayer exploration.*

3. L_2 Norm and Adaptive Cover

In the demo, we used a uniform cover which caused large variations in cluster sizes. While some clusters were composed of only a handful of activation vectors, there were several very large clusters with thousands of activation vectors, and large intersection between neighboring clusters. Finding meaningful relationships across such large clusters is difficult in these cases since 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 a bit more details.

In general, vectors with a dimension as high as the ones from 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 activation vectors for each layer vs. random vectors of the same dimension. The mode associated with the distribution of L_2-norm of activation vectors is left-shifted in comparison with the distribution of random vectors, and the activation vectors have less sharp peaks. We hypothesize that such an observation is due to two reasons:

- The ReLU activation sets all negative values to be zero, hence a significant number of elements in a single activation vector are zero. On average 33% of elements of an individual activation vectors are zero.
- The activation vectors likely have low intrinsic dimensions (in comparison to the high-dimensional space they are embedded into).

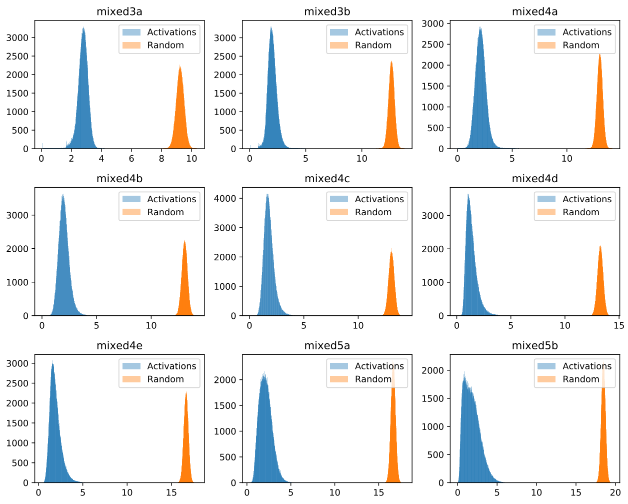


Figure 7: *Comparison of distribution of L_2-norms between activation vectors and random vectors.*

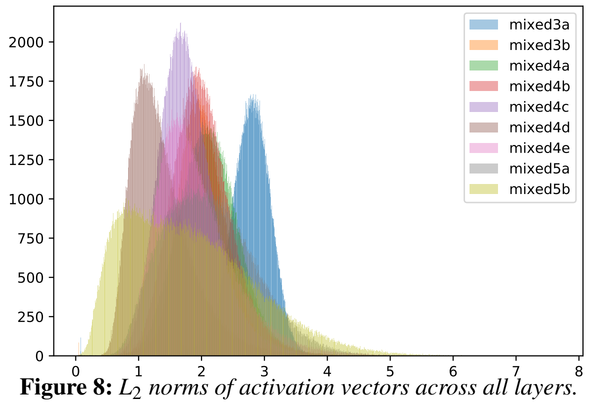


Figure 8: *L_2 norms of activation vectors across all layers.*

Figure 8 shows the distribution of L_2 norms for all layers in the Inception architecture. Notice that the distribution is close to a Gaussian and the variance of the distribution is reasonably large as opposed to very low variance of random vectors. Additionally, the severity of the curse of dimensionality can be further reduced by using an adaptive cover that has more intervals in the denser regions of lens function. The resulting mapper graphs with such an adaptive cover will contain nodes of comparable sizes.

References

[CAS*19] Carter S., Armstrong Z., Schubert L., Johnson I., Olah C.: Activation atlas. *Distill 4*, 3 (2019), e15. 3

[Dwy09] Dwyer T.: Scalable, versatile and simple constrained graph layout. *Proceedings of the 11th Eurographics/IEEE - VGTC conference on Visualization* (2009), 991–1006. 2

[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. 3

[Uly16] Ulyanov D.: Multicore-TSNE. https://github.com/DmitryUlyanov/Multicore-TSNE, 2016. 3

[vVS19] van Veen H. J., Saul N.: Keplermapper. http://doi.org/10.5281/zenodo.1054444, Jan 2019. 3