nature machine intelligence

Article

https://doi.org/10.1038/s42256-023-00749-8

Topological structure of complex predictions

In the format provided by the authors and unedited

Topological structure of complex predictions
SUPPLEMENTARY MATERIALS

Meng Liu, Tamal K. Dey, David F. Gleich
Purdue University, Computer Science

In order to enhance the readability of the supplemental materials, we reproduce some of the extended data figures here.

§1 Methods: Our GTDA method for Reeb nets & prediction functions

In this paper, we developed a framework to inspect the predictions of complex models by visualizing the interactions between predictions and data. The framework has the following properties:

- it can produce a topological view of the original dataset through pictures
- the visualization can provide clues for any sample of interest to be inspected
- it is highly scalable and can process large datasets with thousands of classes
- it can provide intuitive insights and suggest places that are worth a further study for users without any prior knowledge on the model or the data

§1.1 Background: Topological Data Analysis and the Mapper Algorithm

Our method is rooted in the growing field of computational topology and topological data analysis and the framework is closely related to the *mapper* algorithm [35] for topological data analysis (TDA). Mapper builds a discrete approximation of a Reeb graph or Reeb space (see Section §1.6, Figure 3). It begins with a set of datapoints $(x_1, \ldots, x_n)$, along with a single or multi-valued function sampled at each datapoint. The set of all these values $\{f_1, \ldots, f_n\}$ samples a map $f : X \to \mathbb{R}^k$ on a topological space X. The map f is called a *filter* or *lens*. The idea is that when f is single valued, a Reeb graph shows a quotient topology of X with respect to f and mapper discretizes this Reeb graph using the sampled values of f on points $x_1, \ldots, x_n$. Algorithmically, *mapper* consists of the steps:

- Sort the values f_i and split them into overlapping bins $B_1, \ldots, B_r$ of the same size.
- For each bin of values B_j, let S_j denote the set of datapoints with that same value and *cluster* the datapoints in each S_j independently. (That is, we run a clustering algorithm on each S_j as if it was the entire dataset.)
- For each cluster found in the previous step, create a node in the Reeb graph.
- Connect nodes of the Reeb graph if the clusters they represent share a common point.

The resulting graph **is a discrete approximation of the** *Reeb graph* and represents a compressed view of the shape underlying the original dataset.

Our goal is to extract a similar type of topological description for lenses that are multi-valued, which we interpret as a collection of single-valued lenses.

§1.2 Rationale for a graph-based method

The input format for *mapper* is usually a point cloud in a high dimensional space where the point coordinates are used only in the clustering step.

In our methodology, we are interested in datasets that are even more general. Graph inputs provide this generality. Datasets not in graph format like images or DNA sequences can be easily transformed into graphs by first extracting intermediate outputs of the model as embeddings and then building a nearest neighbor graph from the embedding matrix. Then the resulting graph facilitates easy clustering: for each subset of points, we extract the subgraph induced by those points and then use a parameter-free connected components analysis to generate clusters.

Our method could also work with point cloud data and clustering directly through standard relationships between graph-based algorithms and point cloud-based algorithms. We focus on the graph-based approach for simplicity and because we found it the most helpful for these applications.

§1.3 The Reeb network construction on a prediction function using a graph (GTDA)

We take as input:

1. an n-node graph G
2. a set of m lenses based on a prediction model as an $n \times m$ matrix $\boldsymbol{P}$

The lenses we use are the prediction matrix $\boldsymbol{P}$ of a model where P_{ij} is the probability that sample i belongs to class j. Key differences from existing studies of TDA frameworks on graphs include using the connected components of each bin [4, 11] as clusters and also additional steps to improve the analysis of prediction functions by adding weak connections from a minimum spanning tree.

Problems with straightforward algorithmic adaptation. Mapper does extend to multidimensional lens functions by using a tensor product bin construction. We found issues with a straightforward adaptation of *mapper* to such multidimensional input for prediction functions. In our extensive trials, we found that most of the resulting Reeb networks end up with too many tiny components or even singletons where no prediction-specific insights were possible. This is especially so when the dataset has many classes, most multi-dimensional bins will just contain very few samples because the space grows exponentially, limiting the potential of overlap to find relationships. Simply reducing the dimension of $\boldsymbol{P}$ with PCA will lose the interpretability of the lens. Moreover, classic *mapper* uses a fixed bin size and density-based or multi-scale alternatives [8] were unsuccessful in our investigations although they solve this problem from a theoretical perspective. (We note this is a potential area for followup work to better understand why.)

Preprocessing to smooth the predictions. As a preprocessing step, we apply a few steps (usually four or five) of the smoothing iteration: $\boldsymbol{P}^{(i+1)} = (1-\alpha)\boldsymbol{P} + \alpha \boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{P}^{(i)}$. Here $\boldsymbol{P}^{(0)} = \boldsymbol{P}$, $\boldsymbol{A}$ is the adjacency matrix of the input graph, $\boldsymbol{D}$ is the diagonal degree matrix and $0 < \alpha < 1$. This helps to prevent sharp changes between adjacent nodes. This equation is a diffusion-like equation closely related to the PageRank vector that is known to smooth data over graphs and has many uses [10]. The iteration keeps all the prediction data non-negative and the smoothed $\boldsymbol{P}$ will also be min-max column normalized so that each value is between 0 and 1. As is standard, this setup can use any weights associated with the adjacency matrix, or remove them and use an unweighted graph.

Our graph-based construction for a prediction function. The following approach was used for datasets in the main paper. We call this a graph-based topological data analysis framework (GTDA). It uses a recursive splitting strategy to build the bins in the multidimensional space. For each subgroup of data, the idea is that we find the lens that has the maximum difference on those data. Then split the component by putting nodes into two approximately equal sized overlapping bins based on the values in this lens. Then if the resulting groups are big enough, we add them back as sets to consider splitting.

Detailed pseudo code can be found in Algorithm 1. An animation of the method can be found in the supplemental video. We give a quick outline here. The recursive splitting starts with the set of connected components in the input graph. This is a set of sets: $\mathbb{S}$. The key step is when the algorithm takes a set $\mathbb{S}_i$ from $\mathbb{S}$, it splits $\mathbb{S}_i$ into new (possibly) overlapping sets $\mathbb{T}_1, \ldots, \mathbb{T}_h$ based on the lens with maximum difference in value on $\mathbb{S}_i$ and also connected components. Each $\mathbb{T}_i$ is then either added $\mathbb{S}$ if it is large enough (with more than K vertices) and where there exists a lens with maximum difference larger than d. Otherwise, $\mathbb{T}_i$ goes into the set of finalized sets $\mathbb{F}$.

Once we have the final set of sets, $\mathbb{F}$, then we do have two final merging steps, along with building the Reeb net. The first is to merge sets in $\mathbb{F}$ if they are too small (Algorithm 2). The second is to add edges to the Reeb net to promote more connectivity (Algorithm 3).

In the first merging (Algorithm 2), which occurs before the Reeb net is constructed, we check and see if any set in $\mathbb{F}$ is too small (smaller or equal to s_1). If so, then we find nearby nodes based on the input graph G and based on a user-provided distance measure f and

merge the small component with the closest component (giving preference to the smallest possible set to merge into). This could be a simple graph-distance measure (e.g. shortest path), something suggested by the domain, or a weight based on the prediction values (what we use). The algorithm is closely related to Borůvka's algorithm for a minimum spanning tree.

Next, we build the Reeb net $\hat{G}$ from this set of sets $\mathbb{F}$. Each group $\mathbb{F}_i$ becomes a node, and nodes are connected if they share any vertex.

In the second merging (Algorithm 3) we seek to improve the overall connectivity of the Reeb net by connecting small disconnected pieces of the Reeb net $\hat{G}$. This step is designed to enhance the ability to work with predictions by adding weaker connections to the more strongly connected topological pieces. It uses the same distance measure f to find components and uses a similar Borůvka-like strategy. We save the set of edges added at this step to study in the error estimation procedures noted below.

Choices for the parameters. As a result, GTDA has 8 parameters as in Table 1. Tuning of the parameters is straightforward, and we often use the default choice or values from a small set. The values K, d and s_1 provide direct control about the number of nodes in the final group visualization, while r and s_2 control how connected we want the visualization to be. In practice, we could first tune K and d to determine the number of nodes, then tune r so that no component in the Reeb net is too large and finally tune s_1, s_2 to remove any tiny nodes or components. We leave the smoothing parameters fixed at $\alpha = 0.5$ and $S = 5$ or 10 (very smooth). A detailed discussion on these parameters can be found in Section §7.

Choice of distance function for merging We suggest using the ℓ^∞ norm of the difference between rows of the preprocessed $\boldsymbol{P}$ as the distance between 2 samples for the merging function. This choice greatly stabilizes the results because it roughly means how much larger the bin containing one of those 2 samples should be in order to include the other sample due to overlap. Put another way, this choice makes us less sensitive to the exact choice for the overlapping ratio r because we will add small connections that would have been included in a slightly larger bin. On the hand, this the user can choose any distance metric f in the merging step, although we are unaware of any other choice that would be consistent with the goals of TDA besides a metric on the lenses. One possibility would be to incorporate the metric structure identified by an advanced technique such as GENEOs [3].

Drawing the graph. Unless otherwise specified, all coordinates of any layout we show are computed with Kamada-Kawai algorithm [16].

Showing the Reeb network and explorations. In the Reeb net of a prediction function, we draw each node as a small pie-chart. The size of the pie-chart represents the number of nodes. The pieces of the pie show the local prediction distribution. In some cases, we find it useful to study the predicted labels directly, such as when studying mechanisms underlying the prediction. In other cases, we find it useful to study predictions and training data, such as when studying possible errors. These visualizations facilitate exploring regions of the prediction landscape based on interactions among predicted values and training data. By

parameter	description	suggested choices
K	component size threshold to stop splitting	5% of smallest class size
d	lens difference threshold to stop splitting	0 or 0.001
r	overlapping ratio	0.01
s_1	Reeb node size threshold	5
s_2	Reeb component size threshold	5
α	lens smoothing parameter	0.5 (used in all experiments)
S	lens smoothing steps	5 or 10
f	distance function in the merging step	ℓ^∞ difference of row i, j of $\boldsymbol{P}^{(S)}$

Supplemental Table 1: (Reproduction of Extended Data Table for self-contained supplementary notes.) List of parameters in GTDA.

mapping these *small* regions back to the original data, it suggests what the model is utilizing to make the predictions. Examples on this can be found in the experiments in the main text as well as in the supplemental information.

§1.4 Demonstration of GTDA

We use a 3 class Swiss roll dataset to demonstrate each step of our GTDA framework (plot (A) of figure 1). For the GTDA parameters, we set $K = 20$, $d = 0$, $r = 0.1$, $s_1 = 5$, $s_2 = 5$, $\alpha = 0.5$, and $S = 5$. In (B), we show the three prediction lenses we use in the top plot as well as the predicted labels of the model we use. We also add additional edges based on nearest neighbors from node embeddings to take node features into account. This is standard practice in graph neural network methods. Details on the dataset and the model can be found in Section §1.8. Each lens is just the prediction probability of a class after smoothing and normalization. In (C), we pick the lens with the largest min-max difference and split it into 2 bins with 10% overlap (we pick the one with smaller index to break ties). This round of splitting finds 2 components. For each component found in the first iteration, we pick the lens with the largest min-max difference and split it again. In this case, the inner component is split along lens 3 while the outer component is split along lens 2. This round of splitting further divides the graph into 7 components. We repeat the splitting until no component has more than 20 vertices of the original graph.

In the end, we find 247 unique components. As noted above, we use a pie chart to represent each Reeb node and connect Reeb nodes with black lines if they have any samples in common to get the initial Reeb net, (D). Node size is proportional to the number of samples it represents, the pie chart shows the distribution over predicted values. This initial Reeb net has many tiny components or even singletons that are a barrier to deeper insights; the merging steps address this issue. In (E), we use red dashed lines to mark how we will merge those small Reeb nodes so that all nodes will contain more than 5 samples. Similarly, we use red dashed lines to mark extra edges that will be added so that each connected component in the Reeb net will contain more than 5 Reeb nodes. The final Reeb net is shown in (F) with the original graph embedded in the background. We can see that all important structures found in (D) are also preserved in (F) such as the mixing of nodes from

Algorithm 1 `GTDA`$(G, \boldsymbol{P}, K, d, r, s_1, s_2, \alpha, S, f)$ See Table 1 for parameters description

Smooth $\boldsymbol{P}$ for S steps with $\boldsymbol{P}^{(i+1)} = (1-\alpha)\boldsymbol{P} + \alpha \boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{P}^{(i)}$ and $\boldsymbol{P}^{(0)} = \boldsymbol{P}$
Perform a min-max normalization along each column of $\boldsymbol{P}$
Find connected components in G and put all components with size larger than K and maximum lens difference larger than d in $\mathbb{S}$, otherwise in $\mathbb{F}$
while $\mathbb{S}$ is not empty **do**
  Let $\mathbb{S}^{(\text{iter})}$ be a copy of $\mathbb{S}$
  for each $\mathbb{S}_i$ in $\mathbb{S}^{(\text{iter})}$ **do**
    Remove $\mathbb{S}_i$ from $\mathbb{S}$
    Find column c_i (for a lens) such that $\boldsymbol{P}^{(S)}[\mathbb{S}_i, c_i]$ has the largest min-max difference
    Split interval $[\min(\boldsymbol{P}[\mathbb{S}_i, c_i]), \max(\boldsymbol{P}^{(S)}[\mathbb{S}_i, c_i])]$ into two halves of the same length and extend the left half by a ratio of r to give overlapping groups $\mathbb{R}_1$ and $\mathbb{R}_2$ based on which vertices had values in the left and right parts of the interval.
    Create sets $\mathbb{T}_1, \ldots, \mathbb{T}_h$ based on the connected components in $\mathbb{R}_1, \mathbb{R}_2$.
    for each $\mathbb{T}_i$ **do**
      If there are more than K vertices in $\mathbb{T}_i$ and if there is a lens with a maximum difference larger than d, then add $\mathbb{T}_i$ to $\mathbb{S}$. Otherwise, add $\mathbb{T}_i$ to $\mathbb{F}$.
    end for
  end for
end while
Run `node-merging`$(\mathbb{F}, G, s_1, f)$ to get the updated $\mathbb{F}$
Generate Reeb net $\hat{G}$ by considering each component of $\mathbb{F}$ as a Reeb net node and connecting two Reeb net nodes if their corresponding components have overlap
Run `component-merging`$(\mathbb{F}, G, \hat{G}, s_2, f)$ to get the updated $\hat{G}$ and the extra set of edges $\mathbb{E}$
Return $\hat{G}$, $\mathbb{E}$

Algorithm 2 `node-merging`$(\mathbb{F}, G, s_1, f)$

while there exists components in $\mathbb{F}$ with at most s_1 vertices **do**
  Set $\mathbb{C}$ to be empty.
  for each component $\mathbb{F}_i$ in $\mathbb{F}$ where $|\mathbb{F}_i| \leq s_1$ **do**
    for each edge (v_i, v_j) in G where $v_i \in \mathbb{F}_i$ and $v_j \in \mathbb{F}_j \neq \mathbb{F}_i$, compute $f(v_i, v_j)$
    Select the pair of nodes v_i, v_j with the smallest $f(v_i, v_j)$. Let $\mathbb{F}_j$ be the set associated with v_j and choose the smallest size F_j if v_j is in multiple such sets. Add $(\mathbb{F}_i, \mathbb{F}_j)$ to $\mathbb{C}$.
  end for
  View the choices in $\mathbb{C}$ as edges of an undirected graph H where vertices are $\mathbb{F}_i$.
  Compute connected components of H.
  for each connected component H_i of H of size larger than 1 **do**
    Let $\mathbb{F}_1, \ldots, \mathbb{F}_h$ be the underlying sets of H_i from $\mathbb{F}$. Remove each $\mathbb{F}_i$ from $\mathbb{F}$. Then add $\mathbb{F}_1 \cup \ldots \cup \mathbb{F}_h$ to $\mathbb{F}$.
  end for
end while
Return the updated $\mathbb{F}$

Algorithm 3 `component-merging`$(\mathbb{F}, G, \hat{G}, s_2, f)$

Initialize the set of extra edges $\mathbb{E}$ to be empty
Compute connected components of Reeb net $\hat{G}$
Let $\mathbb{D}$ be the set of connected components of $\hat{G}$.
while there exists any $\mathbb{D}_i \in \mathbb{D}$ where $\mathbb{D}_i$ has at most s_2 Reeb nodes **do**
  for each $\mathbb{D}_i \in \mathbb{D}$ where $\mathbb{D}_i$ has at most s_2 Reeb nodes **do**
    Let $\mathbb{C}$ be the union of vertices in G (not $\hat{G}$) for Reeb nodes in $\mathbb{D}_i$.
    For each edge $(v_i, v_j) \in G$ where $v_i \in \mathbb{C}$ and $v_j \notin \mathbb{C}$, compute $f(v_i, v_j)$.
    Select the pair of nodes v_i, v_j with the smallest $f(v_i, v_j)$. Let $\mathbb{F}_i$ and $\mathbb{F}_j$ be the Reeb nodes associated with v_i and v_j and choose the smallest size $\mathbb{F}_j$ if v_j is in multiple such sets. Pick an arbitrary $\mathbb{F}_i$ (we used smallest index in our data structure) if $\mathbb{F}_i$ in multiple such sets.
    Add $(\mathbb{F}_i, \mathbb{F}_j)$ to $\mathbb{E}$.
    Connect the Reeb nodes for $\mathbb{F}_i, \mathbb{F}_j$ in $\hat{G}$
  end for
  Recompute connected components analysis of $\hat{G}$ and update $\mathbb{D}$
end while
Return $\hat{G}$ and $\mathbb{E}$

different classes. And what merging does is to estimate how the tiny nodes and components are connected in the original graph or via the prediction lens so that we have a clear view of predictions over the entire dataset. This supports an inspection of the model's prediction on any sample we want.

As a comparison, in plot (G), we show two Reeb nets that are constructed by the original *mapper* algorithm with different number of bins along each lens. These Reeb nets are not useful to understand the prediction structure. Most samples from the green class are grouped into a few nodes because prediction probability distribution on this class is more skewed, which makes the inspection hard.

§1.5 Error estimation of GTDA

The model often highlights places where there is no reasonable basis for a prediction, e.g. where there is training data with a different label closer to a prediction. This scenario admits an estimate where the model will likely make mistakes by checking the relative locations between predictions and training data.

Using the Swiss roll example, in plot (A) of figure 2, we zoom in on two components GTDA. We then look at the induced subgraph of this region in a projection of the Reeb network. The Reeb network projection expands each Reeb node into the original set of input vertices with duplicated nodes merged and also adds in edges that we put into study predictions (the extra set $\mathbb{E}$ in the algorithms). A detailed projection procedure can be found in Algorithm 5.

Put formally: Given a set of Reeb network nodes in $\hat{G}$, find the union of all vertices in G these nodes represent and call that T. We look at the induced subgraph of T in the projection of the $\hat{G}$ from Algorithm 5.

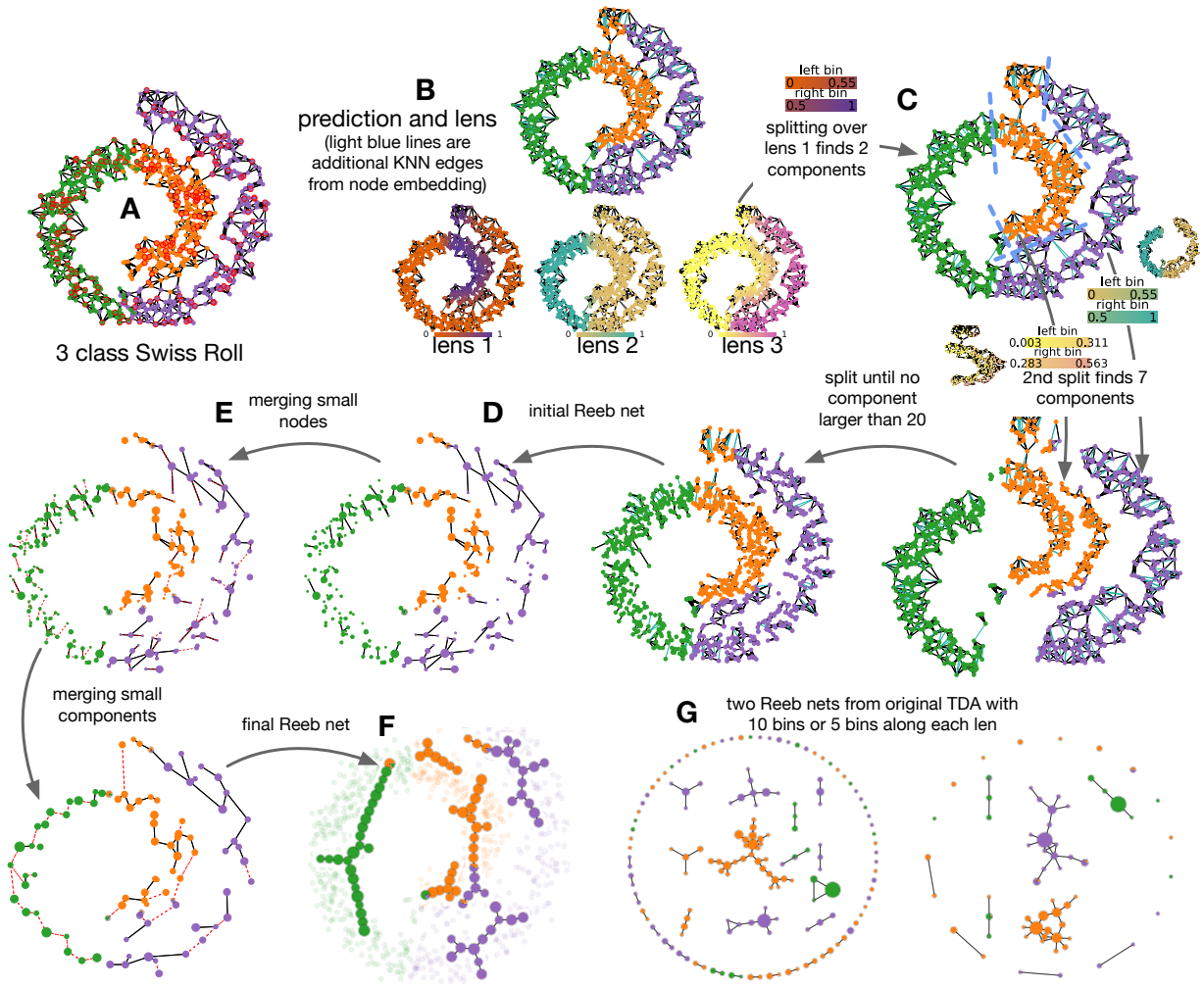
To show these induced subgraphs, we can either use Kamada Kawai layout or, as an alternative to Kamada Kawai, we can also compute coordinates for each projected Reeb node and then combine different layouts using their relative coordinates in Reeb net.

Then we use red circles to mark training and validation data and color them with the true labels. Unknown data points are still colored with predicted labels.

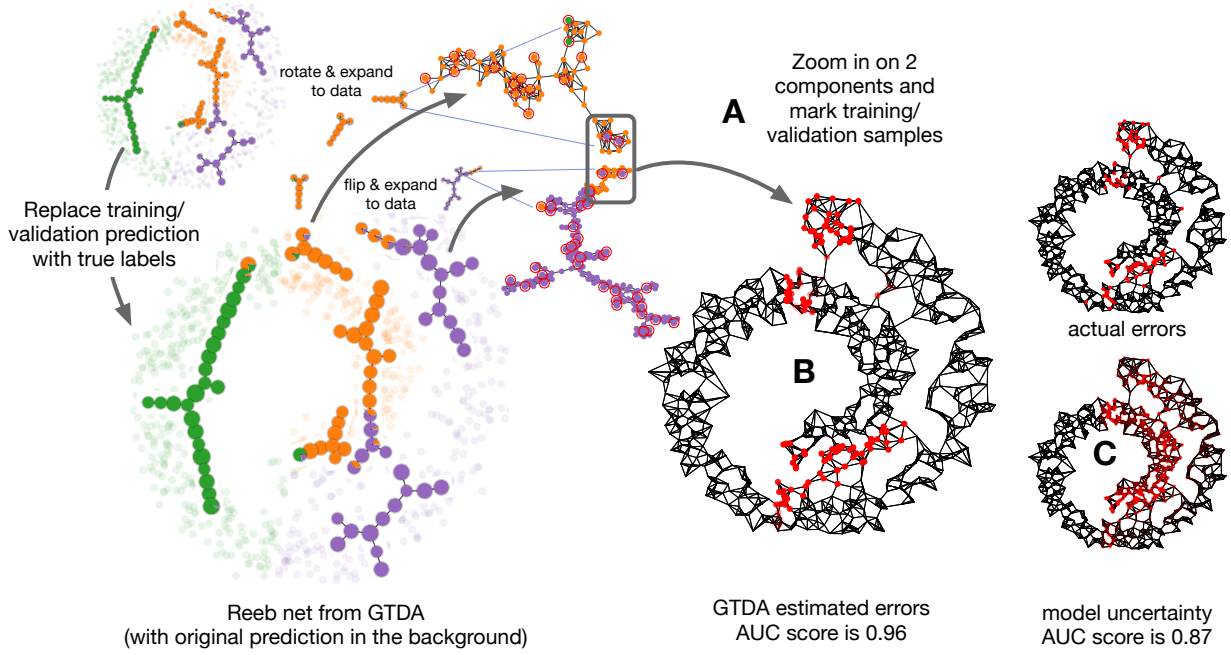
One can immediately notice the problem: *There are some orange predictions in the grey box, but there is no orange training or validation data nearby to support them.* Thus, either the model or the dataset itself have issues with these prediction and merit a second look. In this case, it is just the model that cannot classify some parts of the graph correctly due to noisy links.

We developed an intuitive algorithm to automatically highlights which parts of the visualization will likely contain prediction errors, Algorithm 4. The core part of this algorithm is to perform a few steps of random walk starting from nodes with known labels. Predictions that are close to training/validation data with the same labels in the Reeb net will have higher scores in the column of predicted labels and hence have smaller error estimates.

Applying this algorithm can successfully find other places where mistakes will happen (see plot (B) of figure 2). As a simple comparison, we also include another plot where we directly use model uncertainty (i.e. 1 minus model prediction probability) to estimate errors (see plot (C) of figure 2). This metric has been previously used to estimate uncertainty of dataset labels [25]. Clearly, GTDA is able to localize model errors much better and has



Supplemental Figure 1: A detailed illustration of applying GTDA to build a Reeb net on a 3-class Swiss roll dataset. The original data graph and "ground truth" values are in (A). We show the model prediction for a simple GCN and the three prediction lenses (after smoothing) in (B). See also the supplementary video illustrating the process. The first splitting iteration over lens 1 finds 2 components, (C). At the second split, for each component, we choose the lens with the largest difference, which means the outer ring is split over lens 2 and the inner ring is split over lens 3. The second splitting finds 7 components in total. We continue to split until no more components larger than 20 and get the initial Reeb net, (D). Then small nodes are merged to neighbors iteratively as shown by the red dashed lines in (E). Similarly, small components in the Reeb net are iteratively connected to get the final Reeb net in (F). As a comparison, two Reeb nets from the original *mapper* using 10 lens or 5 lens have many isolated nodes or components and are not suitable for the subsequent inspection. The figure (F) uses predicted classes for training and validation points instead of the actual training and validation classes as in Main Figure 2(D).



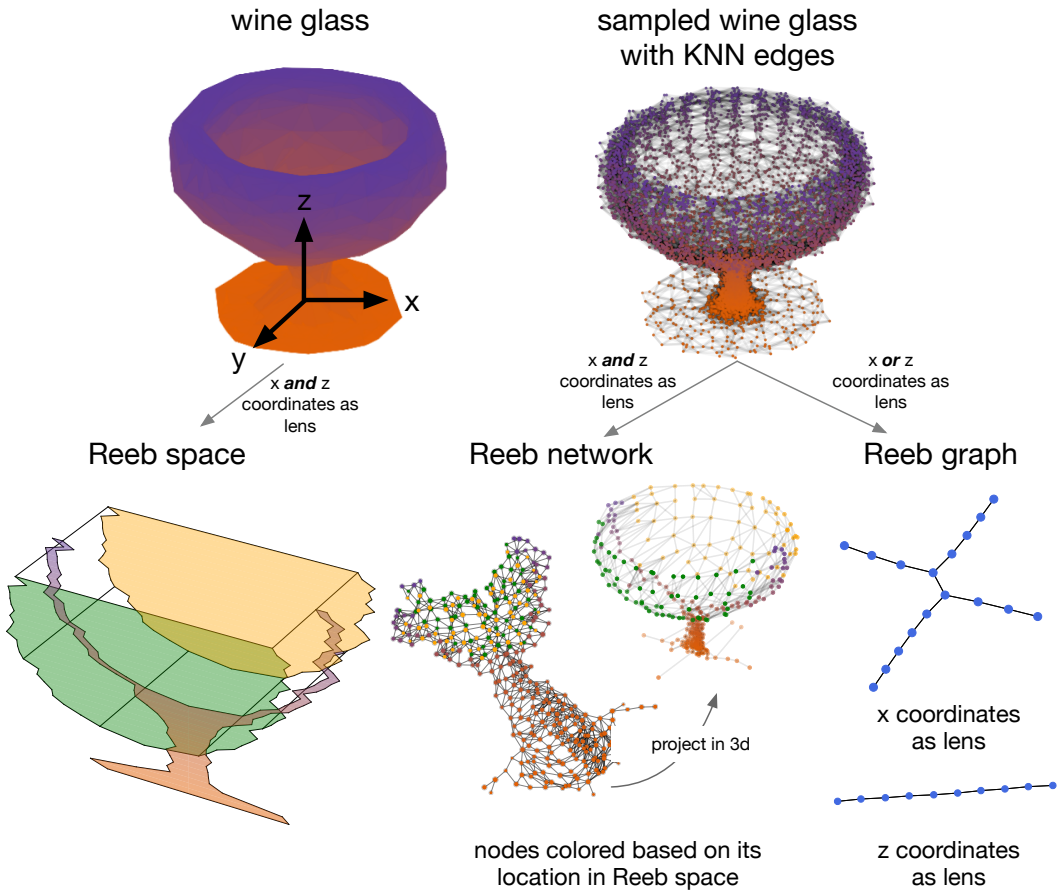
Supplemental Figure 2: This figure demonstrates the procedure of estimating errors from the Reeb net produced by GTDA. In comparison with Figure 1, we show the training data labels in the pie charts instead of the predicted values. If we zoom in on two components and mark training and validation samples (red circles) with true labels, we see many orange predictions without any training or validation data nearby to support them (inset box nearby) (A), which suggests potential errors – note that the model may be using additional features to predict these values, but these examples do merit closer inspection. We develop an error estimation procedure in Algorithm 4 to automate this inspection. Overall, GTDA estimated errors have a AUC score of 0.95 with true errors (B), while using model uncertainty (one minus prediction probability) only has a AUC score of 0.87 (C).

a higher AUC score (0.95 vs 0.87). There always exists other methods [15] that can also give similar error estimations or even correct predicted labels. But explaining why those methods should work or be trusted to a user without background knowledge is a challenge, while our method offers a map-like justification that can give a rough rationale. Moreover, any results from Algorithm 4 can always be validated and supported through pictures similar to plot (A) of figure 2. Also, other than finding possible errors, as shown in the following experiments sections, we can often get many other insights about the model and the dataset by checking abnormal areas of GTDA visualization, ranging from labeling issues to strong correlation between model predictions and a particular dataset property. These are explored in subsequent case studies.

§1.6 Reeb graph vs. Reeb space vs. Reeb network

The main difference between a Reeb graph and Reeb network is the number of lenses used because the Reeb net involves a multivalued map which can be thought of as a collection of single valued maps. A demonstration to show this difference can be found in Figure 3.

Formally, let $f : X \to \mathbb{R}^k$ map a topological space X to a k-dimensional real space. Two



Supplemental Figure 3: (Reproduction of Extended Data Figure for self-contained supplementary notes.) This illustrates the difference between a Reeb graph and a Reeb network on a topologically interesting object. The lenses we use here are the x and z coordinates. The inspiration for the object is [37].

Algorithm 4 `error_estimation`$(\hat{G}, \mathbb{E}, \ell, n, \alpha)$ where $\hat{G}$ and $\mathbb{E}$ is the Reeb net and extra set of edges from algorithm 1, ℓ are the original predicted labels, S is an integer for the number of steps (10, or 20 were used), and $0 < \alpha < 1$ (we use $\alpha = 0.5$ in all experiments).

Compute $G^{(R)}$, the projection of the Reeb net back to a graph from Algorithm 5.
Let $\boldsymbol{A}^{(R)}$ be the adjacency matrix of $G^{(R)}$
Compute a diagonal matrix $\boldsymbol{D}^{(R)}$ where $\boldsymbol{D}^{(R)}_{ii}$ is the degree of node i in $G^{(R)}$ and 0 elsewhere.
Initialize matrix $\hat{P}^{(0)}$ where $\hat{P}^{(0)}_{ij} = 1$ iff node i is a training/validation node with label j, otherwise $\hat{P}^{(0)}_{ij} = 0$.
for $i = 1...S$ **do**
    $\hat{\boldsymbol{P}}^{(i)} = (1-\alpha)\hat{\boldsymbol{P}}^{(0)} + \alpha \boldsymbol{D}^{(R)^{-1}} \boldsymbol{A}^{(R)} \hat{\boldsymbol{P}}^{(i-1)}$
end for
Row normalize $\hat{\boldsymbol{P}}^{(S)}$ so that each row sums to 1.
Compute estimated prediction error for node i to be $e_i = 1 - \hat{\boldsymbol{P}}^{(S)}[i, \ell_i]$
Return estimated errors **e**.

Algorithm 5 `Reeb-graph-projection`$(\mathbb{F}, \mathbb{E}, G)$ where $\mathbb{F}$, $\mathbb{E}$ is the final set of components and extra set of edges from Algorithm 1 and G is the original graph

Initialize $G^{(R)}$ with the same dimension of G and no edges
for Each $\mathbb{F}_i$ of $\mathbb{F}$ **do**
    Add the set of edges of $\mathbb{F}_i$ from G to $G^{(R)}$
end for
Add edges in $\mathbb{E}$ to $G^{(R)}$
Return $G^{(R)}$

points $x, y \in X$ are called equivalent if (i) $f(x) = f(y)$ and (ii) they belong to the same connected component of the level set $f^{-1}(f(x))$. Denoting this equivalence relation with $\sim$, we obtain the quotient space $R^f_X = X/\sim$. When the range of f is $\mathbb{R}$, R^f_X is a one-dimensional space called the Reeb graph of f. When f is multi-valued, that is, $k > 1$, R^f_X becomes a space called Reeb space. By choosing the bins in $\mathbb{R}^k$, we discretize this Reeb space with a graph which we call the *Reeb net* here. We choose the term Reeb net to distinguish it from discretized Reeb graph because both are graphs but one discretizes a one-dimensional space (a graph) and the other discretizes a quotient space that need not be one-dimensional.

§1.7 Opportunities and extensions of the method

We presented the GTDA framework for the main methods we used. In the following case studies and demonstrations, we show there are multiple variations that would be easy to adapt. For instance, we could easily combine multiple graphs from different sources to reveal potential errors that might hidden in a single source.

Areas for future improvement. Our current GTDA framework does rely on some tuning of parameters and manually finding any interesting local structures in the visualization,

especially the component size threshold, which behaves similarly to bin size in the original TDA algorithm. While we designed the algorithm to be as robust as possible, it remains an open question on whether we can automatically select a good set of parameters and identify structures worth looking at. Existing work selects parameters for the original TDA framework based on statistical analysis [5]. But it is not clear how to extend such technique to our GTDA framework.

Areas for additional topology. Another direction is to study the outputs of GTDA under perturbations or filtrations over parameters. Alternatively, there are opportunities to utilize additional topological insights to improve the graph drawing. Consider that a study of persistence of structures in the graph should suggest their placement, i.e. two components that will be connected more easily by perturbing parameters should be put closer. This can then lead to a better overall view of the entire dataset.

Ideas for error improvement. There are a number of opportunities to study the error estimation procedure (Section §1.5) in concert with correcting predictions. In cases of binary classification such as the harmful gene prediction problems, the evidence from the error estimation is often sufficient to allow us to predict the opposite class. We have not pursued this idea in the multiclass scenario and that may be worth doing. We do limited additional study and comparison between the error estimation procedure in Section §10. These experiments show that the error estimation from GTDA is more precise than using the original graph created when building the GTDA data, although the overall AUC values are slightly lower. We believe the reduction in AUC occurs because the GTDA graphs are much sparser and this limits a few predictions in the tail. Consequently, there are a wide variety of explorations that could be done to sharpen and refine these estimates.

§1.8 Other details

Swiss Roll dataset construction We use *scikit-learn* package to build the Swiss Roll dataset. We use 1000 samples in total and the noise parameter is set to be 1.2. The initial Swiss Roll dataset is a 1000-by-3 matrix X and a vector y which represents the position of each sample in the main dimension of the manifold. We only keep the first and the third columns of X and use them as node features. And we sort samples based on y and consider the first 33% samples as the first class, the second 33% samples as the second class and all the other samples as the third class. The graph is a nearest neighbor graph with each node connecting to its 5 closest neighbors using Euclidean metric on X. We use a random set of use 10% samples as training, another 10% samples as validation and all the other points as testing.

Model and parameters We use a standard 2-layer GCN model to predict labels of testing samples. The dimension of the hidden layer is 64, learning rate is 0.01 and weight decay is 10^{-5}. Once the model is trained, we use outputs of the first layer as node embeddings. The embedding matrix is reduced to 16 dimension using PCA with whitening and then each row is ℓ_2 normalized. We build another 2-NN graph using the preprocessed embedding matrix

and cosine similarity to encode any information from node features. This graph is combined with the original graph. GTDA framework is then applied on the combined graph. For GTDA parameters, we set $K = 20$, $d = 0$, $r = 0.1$, $s_1 = 5$, $s_2 = 5$, $\alpha = 0.5$ and $S = 5$. We use 10 steps of iterations for GTDA error estimation.

Alternative neural networks. We note that we saw similar results using the GNN methods from [6]. We include discussions and images with this alternative method for the Amazon dataset (next section) to evaluate our statement from the main text about the taxonomy.

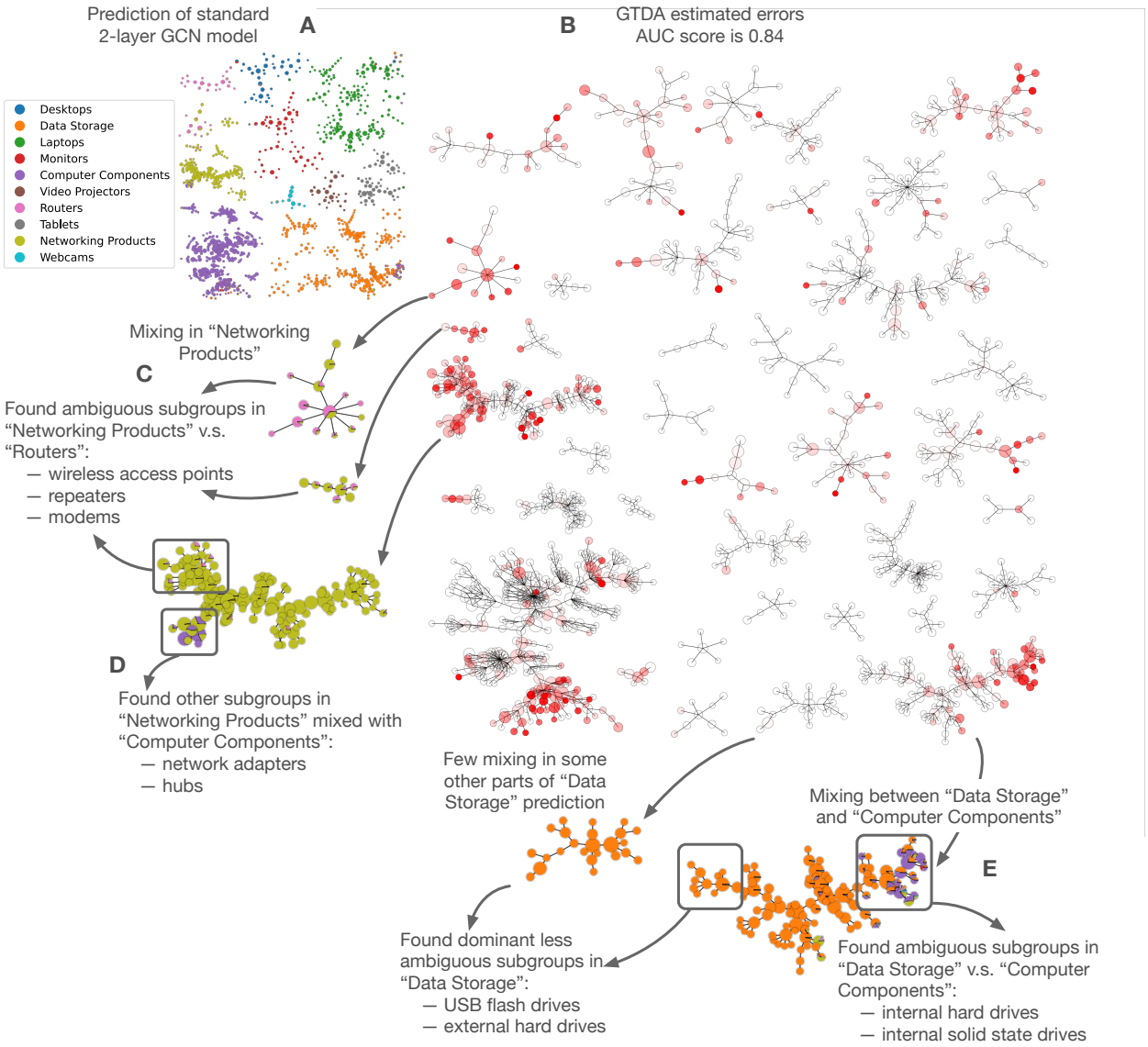
§2 Demonstration in graph based prediction

We apply the Reeb net framework to a graph neural network that predicts the type of product on Amazon based on reviews. This framework identifies a key ambiguity in product categories that limits prediction accuracy (Figure 4). Specifically, "Networking Products" and "Routers" overlap (a Router is a specific type of Network Product) and show high levels of confusion as do "Data Storage" and "Computer Components" (an internal data storage drive is a computer component). These results suggest that large improvements are unlikely with better algorithms and would require label improvements to differentiate categories or other divisions in a hierarchy [42]. This was verified by checking another graph neural network [6] with similar behavior see Supplemental Material Section §2.3.

In the following part of this section, we provide more details for the application of our GTDA framework on the Amazon co-purchase graph [32] constructed from Amazon reviews data [21]. Each node in this graph is a product, edges connect products that are purchased together and node features are bag-of-words from product reviews. The goal is to predict product category. The original dataset [32] that has been used in several GNN papers does not have information for each specific product. To better understand the visualization from GTDA, we build a similar dataset directly from the Amazon reviews data [21]. We use the 2014 year version of reviews data and extract products with the same set of labels as in the original [32]. In the remainder of this section, we will work through how the dataset is constructed and the GCN model parameters that are used. We will also provide GTDA results on another more recent GNN model, GPRGNN [6], that is based on spectral theory. We will inspect this model's prediction on both the customized dataset and the original dataset. We will see later in this section that the same conclusions as Figure 4 still hold even after switching to the new model.

§2.1 Dataset and GNN model

Our own version of the Amazon co-purchase graph has the same set of the labels as the original one [32]. We download all products and reviews in the category of "Electronics" by following the link provided in [21]. We use the 2014 version as we can find the exact same set of labels in this version. In the Amazon reviews dataset, each product is associated with a list of categories. To assign labels, for each product, we check from the most general category (i.e. Electronics) to the most specific one (i.e. Routers). And if we find a match to the set of labels we choose, we directly assign the matched label to that product and ignore the other categories in the list. Two products will be connected if they are marked as "also



Supplemental Figure 4: (Reproduction of Extended Data Figure for self-contained supplementary notes.) Reeb network of a standard 2-layer graph convolutional network model trained and validated on 10% labels of an Amazon co-purchase dataset (A) and estimated errors shown in red (B). The map highlights ambiguity between “Networking Products” and “Routers”. Checking these products shows wireless access points, repeaters or modems as likely ambiguities (C). Additional label ambiguities involve “Networking Products” and “Computer Components” regarding network adapters (D); likewise “Data Storage” and “Computer Components” are ambiguous for internal hard drives (E). These findings suggest that the prediction quality is limited by arbitrary subgroups in the data, which Reeb networks helped locate quickly.

category	number
Desktops	1,757
Data Storage	7,297
Laptops	4,590
Monitors	1,710
Computer Components	15,167
Video Projectors	804
Routers	1,086
Tablets	1,919
Networking Products	4,869
Webcams	548

Supplemental Table 2: Number of products for each category in our own version of Amazon Computers dataset.

bought", "bought together" or "buy after viewing". After we get the initial graph, we first make the graph undirected and then filter out components that are smaller than 100. We use bag-of-words node features with TF-IDF term weighting constructed from each product's review text. The final graph we get has 39,747 products and 798,820 edges. The number of products for each category is listed in table 2.

To get the prediction results used in Figure 2, we use the same 2-layer GCN model as the Swiss Roll experiment to predict product categories (Section §1.8). The dimension of the hidden layer is 64, learning rate is 0.01 and weight decay is 10^{-5}. We randomly use 10% samples as training, another 10% samples as validation and all the other samples as testing. We extract the output of the first layer as node embeddings and we also build a 2-NN graph using cosine similarity to combine with the original graph. This will let GTDA show the impact of the feature similarity on the GNN. For GTDA parameters, we set $K = 100$, $d = 0$, $r = 0.01$, $s_1 = 5$, $s_2 = 5$, $\alpha = 0.5$ and $S = 5$. We use 20 steps of iterations for GTDA error estimation. For the more advanced GPRGNN model used below, we use the same set of parameters as suggested by its authors [6] and node embeddings are extracted from the first layer output as well. We also use the same GTDA parameters as GCN.

§2.2 Inspecting model predictions with GTDA

In Figure 4, we found ambiguous subgroups inside "Data Storage" and "Networking Products" with the help of GTDA visualization. Similar ambiguities persist after switching to the more advanced GPRGNN model as shown in Figure 5. Here, we also notice many estimated errors in "Routers" and "Data Storage" as before. We show a detailed breakdown of products true categories for some components. For each component highlighted, we list top 2 most common categories. The other categories are put in "Others". For "Networking Products" and "Data Storage", we also list the top 3 most common subcategories. For the two "Routers" components in (A), we see many "Modems" or "Wireless Access Points" from "Networking Products". These should be frequently bought together, and "Routers" should be considered as another subcategory of "Networking Products". As a comparison, for the other "Networking Products" component that is less mixed (B), the most common

subcategories are "Network Adapters" and "Hubs", which are more precise than the more ambiguous "Routers". Similarly, for the two "Data Storage" components in (C), the mixed one has many "Internal Drives" such as solid state drives (SSDs). These are essential parts of a PC and should be considered as a part of "Computer Components" as well. There are also a small portion of "Network Attached Storage", which may be confused with "Networking Products". On the contrary, the less mixed one mostly contains "External Drives" like USB drives, which are common additions to an already built PC. These results suggest that for this dataset, no matter which model we choose, the performance on some portion of the dataset will always be limited by the same type of underlying labeling issues. GTDA helps capture those issues in both cases.

§2.3 GTDA with a more advanced method

As a final check on our results, in Figure 6, we apply GTDA to inspect GPRGNN's prediction on the original Amazon dataset built by [32] with the same setting. We can observe similar behavior to Figure 5, that is "Routers" is mixed with "Networking Products" and components of "Data Storage" are mixed with "Computer Components".

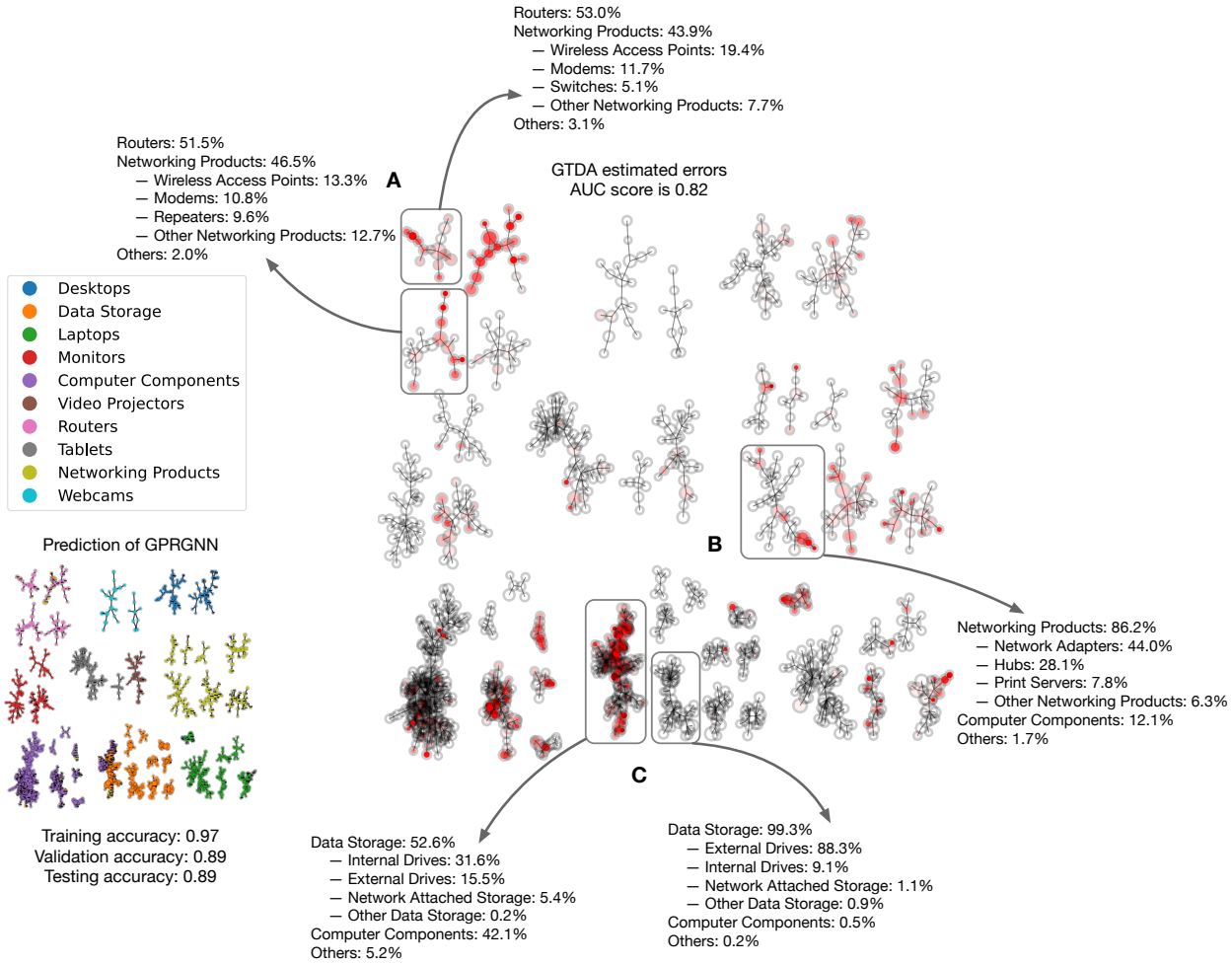
§3 Understanding image predictions

Summary When the GTDA framework is applied to a pretrained ResNet50 model [12] on the Imagnette dataset [13], then it produces a visual taxonomy of images suggesting *what* ResNet50 is using to categorize the images (Figure 7). This example also highlights a region where the ground truth labels of the datapoints are incorrect and cars are erroneously predicted as "cassette player". This arises because of mislabled images in the original ImageNet data.

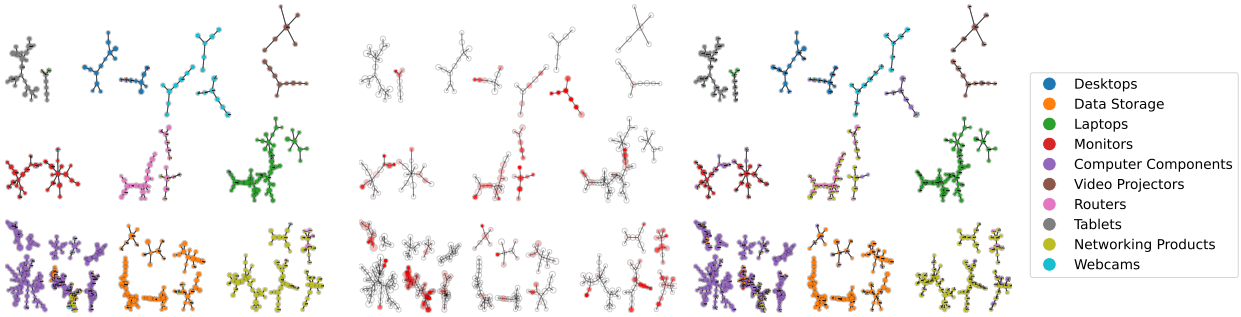
Motivation One of the most successful applications for complex neural network models is detecting objects in images. Image classifiers based on convolutional neural networks (CNN) can achieve extremely high accuracy, sometimes even higher than humans. What remains not entirely understood is how to explain a model's prediction and whether it will generalize well beyond the training scenario.

Summary of GTDA results In Figure 7, we have shown how we can use the GTDA visualization to study predictions made by a pretrained ResNet50 classifier on a subset of ImageNet called Imagenette. The Reeb net from GTDA helps identify "cassette players" that are really pictures of cars. This subgroup arises inside of subgroup of "gas pump" images. These pictures of cars are predicted and labeled as cassette players because the training data itself contains this mislabeled images. We also tested the network with an image of another car taken at a gas pump and found that the model labeled it as cassette player instead of gas pump. Note that this arises from only a few samples where "cassette player" is erroneously applied to a picture of a car.

In the following, we will provide more details on the dataset and the CNN model. Then we will use a random experiments to show that GTDA is stable in detecting this cassette player and car behavior and the criteria we use to find it cannot be easily satisfied in a



Supplemental Figure 5: We provide GTDA results on inspecting the prediction on the GPRGNN method instead of the GCN used in Figure 4. We list a detailed breakdown of categories and subcategories for a few components. For the two "Routers" components in (A), there are many estimated errors because of ambiguous subgroups of "Networking Products" like "Wireless Access Points", "Modems" or "Repeaters". The estimated errors are much less in (B) because "Networking Products" has dominant less ambiguous subgroups. Similarly, for two "Data Storage" components in (C), the one with more estimated errors has dominant ambiguous subgraphs like "Internal Drives" or "Network Attached Storage" which is confusing with "Computer Components" or "Networking Products".



(1) predicted labels (2) GTDA estimated errors (3) true labels

Supplemental Figure 6: GTDA visualization of GPRGNN's prediction on the original Amazon Computers dataset [32]. Similar to Figure 5, "Routers" is mixed with "Networking Products" and some components of "Data Storage" are mixed with "Computer Components".

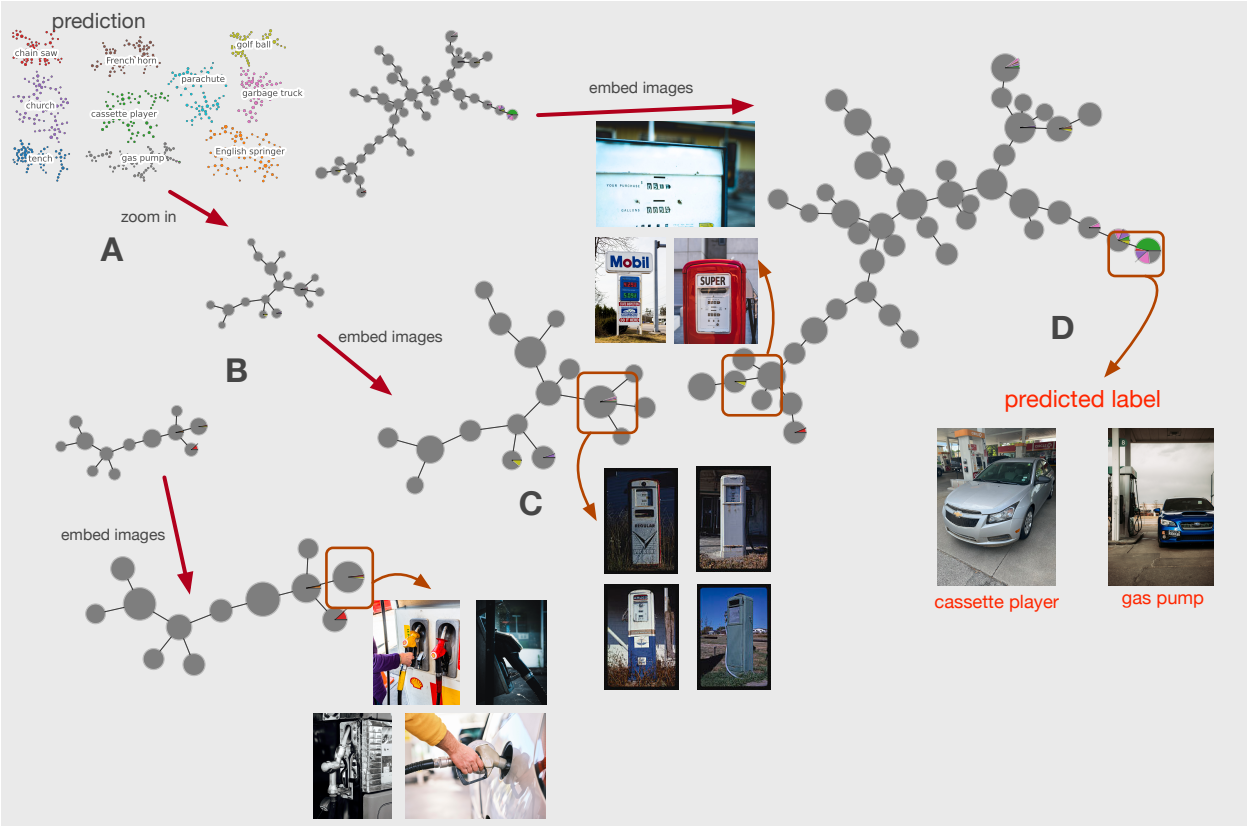
random set of images. We also compare to influence functions to Finally, we compare with results of the original *mapper*.

Displaying Reeb networks for images. Because each image can be displayed, we customize a display of a Reeb net (which is simply a graph) to show the results of a Reeb net analysis by placing images directly on the layout. This involves a few relevant details that may assist others in the future so we detail our methodology here. It was inspired by Tufte's work on image quilts and small multiples [38].

Prior work on understanding image predictions. Existing research seeks to explain model predictions by computing activation maps or saliency maps [36, 43, 31, 34]. In these maps, areas that contribute to the final prediction will be highlighted and the user can justify model predictions by checking whether the areas highlighted make sense. Some other studies take a different approach by training a simple and explainable model (i.e. a linear classifier) to mimic the prediction functions of the original model [29]. However, all these approaches can only explain the model's prediction on a single sample each time instead of model's prediction ability in the entire dataset. The training and testing datasets can contain hundreds of thousands of images. So examining the explanation for all images is not straightforward. Finding representative samples is another alternative [29], but checking explanation on each selected sample is still required. We note that our GTDA analysis could assist such efforts by studying the topology of the saliency maps, along with the predictions, although we have not pursued this direction.

§3.1 Dataset and CNN model

The dataset we use is Imagenette [13], which is a subset of the entire ImageNet containing 10 easily classified classes, "tench" (a type of fish), "English springer" (a type of dog), "cassette player", "chain saw", "church", "French horn", "garbage truck", "gas pump", "golf ball" and "parachute". This dataset can be directly downloaded from a Github repository [13]. The author uses a different training and testing split from the original ImageNet dataset so we first restore the original split before model training. This choice is because



Supplemental Figure 7: (Reproduction of Main Figure for self-contained supplementary notes.) We take a pretrained ResNet50 model and retrain the last layer to predict 10 classes in Imagnette (A). In (B), we zoom into the Reeb network group of "gas pump" predictions and display images at different local regions (C). This shows g as pump images with distinct visual features. Examining these subgroups can provide a general idea on how the model will behave when predicting future images with similar features as well as help us quickly identify potential labeling issues in the dat aset. For instance, we find a group of images in (D) whose true labels are "cassette player" even though they are really images of "cars".

label	training	testing
tench	1,300	50
English springer	1,300	50
cassette player	1,300	50
chain saw	1,194	50
church	1,300	50
French horn	1,300	50
garbage truck	1,300	50
gas pump	1,300	50
golf ball	1,300	50
parachute	1,300	50

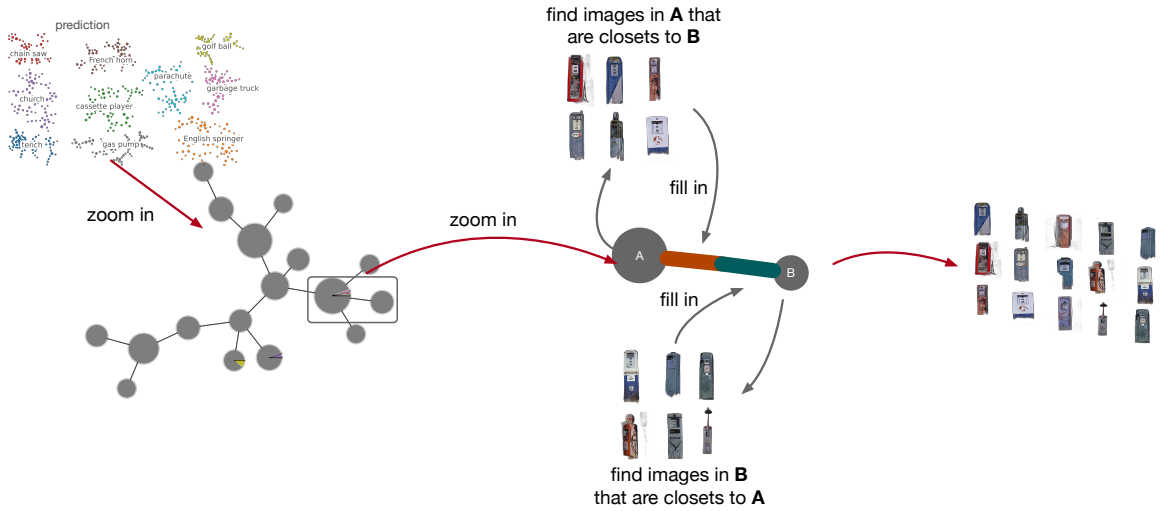
Supplemental Table 3: Number of training and testing images for each label.

the pretrained model from the full ImageNet dataset may have had access to images in the Imagenette test set. The number of training and testing images for each class is shown in table 3.

We use a pretrained ResNet50 model that is included in the PyTorch package and retrain the last fully connected layer to make predictions on these 10 classes only. We use a batch size of 128, learning rate of 0.01 and run for 5 epochs. We also use the common image transform during training and testing. That is, each training image will be randomly cropped into 224-by-224, randomly horizontally flipped and normalized by the mean and standard deviation computed over the entire ImageNet dataset, while each testing image will be resized to 256 along the shorter edge, center cropped to 224-by-224 and then normalized. We modify the pooling of the last convolutional layer from average pooling to maximum pooling and extract its output as node embeddings. Similar techniques are used in the context of image retrivial [28]. Initially, the embedding dimension is 2048. We first PCA reduce the dimension to 128 with PCA whitening. Then each row is ℓ_2 noramlized. A 5-NN graph is constructed on the preprocessed embedding matrix with cosine similarity. For GTDA parameters, we set $K = 25$, $d = 0.001$, $r = 0.01$, $s_1 = 5$, $s_2 = 5$, $\alpha = 0.5$ and $S = 10$. We use 10 steps of iterations for GTDA error estimation.

§3.2 Details on selecting images to embed

We developed a procedure to embed images directly onto a Reeb net in order to make browsing easier. Illustrations of this can be found in the arXiv version of this manuscript [20]. For each pair of adjacent Reeb net nodes, for each image in one end, we measure its smallest distance in the projected Reeb net to some node in the other end. Note some images can be duplicated in two ends, in such case, we consider the distance to be zero. If two images have the same distance, we include the one with larger degree in the projected Reeb net. Then we fill in the closest images to one half of the edge and vice versa. A simple demo can be found in Figure 8. We also apply a background removal algorithm [26] for each image we embed. After embedding selected images, we can then easily browse around different regions of the component to understand the model's behaviour of predicting "gas pumps". Then we can



Supplemental Figure 8: (Reproduction of Extended Data Figure for self-contained supplementary notes.) This figure demonstrates the procedure of embedding images on a Reeb net component. For each pair of adjacent nodes, we select images from one end that are closest to the other end and fill in those images in half of the edge and vice versa. Browsing around embedded images at different regions can help us quickly identify 7 ambiguous "cassette player" images that are really just "cars".

simply select a few Reeb net nodes at different places and check them in detail by listing all images it contains to look for the most common patterns. This can help understand and diagnose problematic regions in the prediction landscape such as finding those "cassette player" images that are really just "cars".

§3.3 Statistical validation

There are 7 cassette player images in the original data that are mistaken as cars. We verify that GTDA is stable in detecting those 7 confusing "cassette player" images. We randomly train 100 models in the same way as described before and check the visualization using each of these 100 models. On average, only 1.3 of these 7 images are predicted wrong, which means simply iterating through all the prediction errors is not enough. We define that this labeling issue can be detected in a visualization if the following criteria can be met:

- All or most of these 7 images are in the same component
- Some neighbors of these images are from a different class
- These images are well localized in the component with small pairwise path length

In our results, we find the visualization from all 100 models can meet these criteria. More specifically, for 74 models, all 7 images can meet these 3 criteria. In the other 26 models, for 22 of them, 6 images can meet all 3 criteria, for 2 models, 5 images can meet and for the rest 2 models, 4 images can meet. Also the maximum pairwise path length for images meeting the criteria is 4 (for most models, this maximum length is 2). Secondly, we verify that a random group of 7 images will be very unlikely to satisfy these criteria. We pick one of the 100 models and randomly sample 7 images from each Reeb net component. We cannot

find any randomly sampled group in 10000 Monte Carlo experiments that can satisfy these criteria simultaneously.

§3.4 Comparing to influence functions

Influence functions [17] is a framework recently proposed to extract the most influential training samples on any specific testing sample. It can also be used to find adversarial or mislabeled training data. We used an existing implementation of influence functions from `https://github.com/nimarb/pytorch_influence_functions` to find ambiguous training samples of Imagenette. The biggest issue of influence functions is scalability. Computing influence for all 12,894 images will take almost 4 hours while our GTDA framework only takes about 1 minute to process the entire dataset. Table 4 compares the top 30 most confusing training images of "cassette player" from influence functions or GTDA. For GTDA, we directly take top 30 images with the largest estimated errors using Algorithm 4. Both methods find training images that indeed look confusing. However, another advantage of GTDA is we get more insights by grouping these ambiguous training images based on their locations in the visualization and checking nearby images in the visualization.

§3.5 Comparing to a Reeb net from original TDA framework

Since the original format of the image representations is an embedding matrix, we can get another Reeb net from the original TDA framework (i.e. *mapper*) without transforming the embedding matrix into a KNN graph. The embedding matrix is still PCA reduced to 128, whitened and ℓ_2 normalized. We also use the prediction lens without softmax as the softmax function will make lens highly skewed, i.e. most lens will be close to 0 or 1. We split each lens into 10 bins with 10% overlap. Then we apply density based spatial clustering [9] for samples in each bin so that we don't need to select the number of clusters. This clustering scheme will consider some samples as noise and not clustering them. We set the maximum distance between two points to be in the same cluster as 3. The Reeb net is shown in Figure 9, which doesn't show any obvious subgroups other than 10 major components representing 10 classes or any labeling issues previously discovered by GTDA. We also find that no information can be extracted at all for around 28% images as they are either in some very small Reeb net components or simply considered as noise by the clustering scheme.

§4 Comparing models on ImageNet-1k predictions

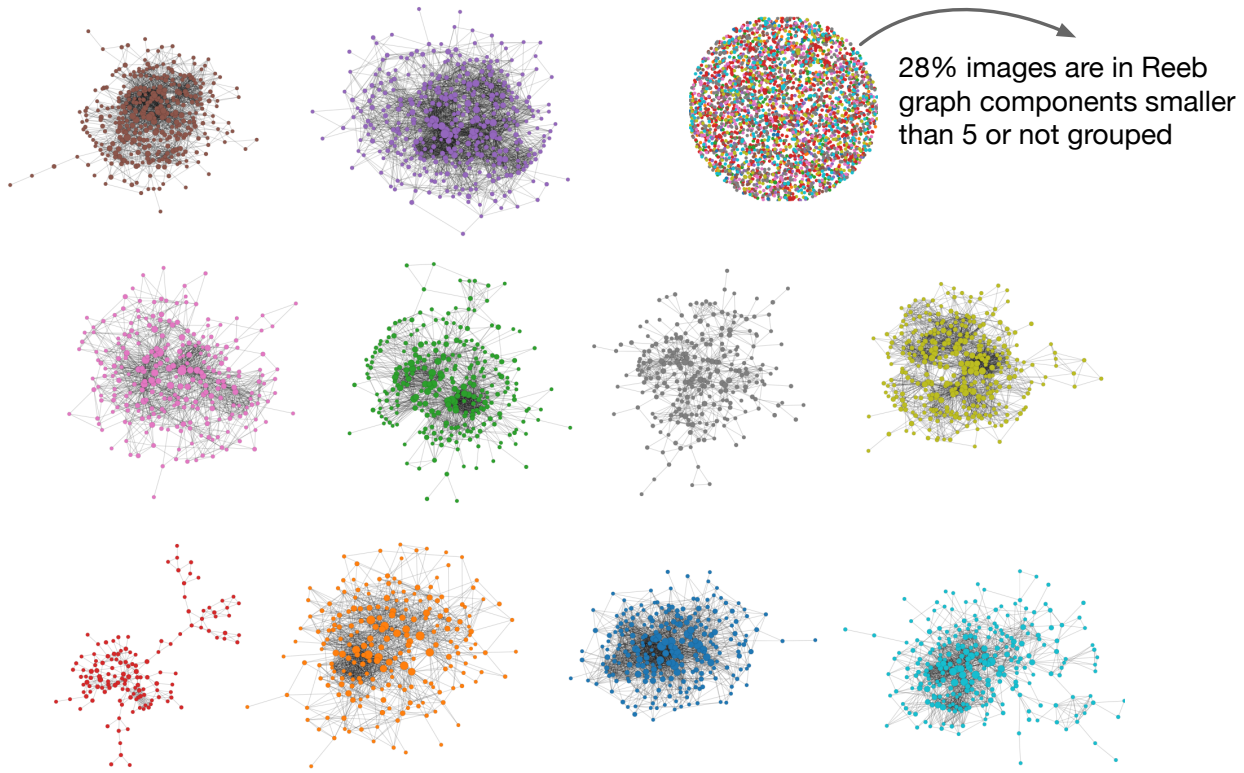
In this section, we apply GTDA framework on the entire ImageNet dataset with 1000 classes from 2012 [30] to compare performance between state of the art CNN models and historical models in any individual class. The results in the later sections show that GTDA can highlight which subgroups inside a class the more advanced models can have improved performance. It also shows how models predict when the image itself has confusing labels.

§4.1 Dataset and CNN models

We use the training and validation images of entire ImageNet dataset with 1000 classes that was released in 2012 [30]. We use 3 different CNN models for comparison, AlexNet,

Influence Functions		GTDA	
Top Images	Description	Top Images	Description
n02979186_15972	Upcycled transparent cassette player	n02979186_6056	Green car
n02979186_15931	Small cassette recorder	n02979186_13851	Woman changing cassette in walkman
n02979186_10523	Person with Cassette Player	n02979186_21644	Boombox
n02979186_11999	Portable cassette player	n02979186_3484	Unknown box
n02979186_4780	Video recorder by Sony	n02979186_3665	SUV-like car with sales sticker
n02979186_3665	SUV-like car with sales sticker	n02979186_6008	Minivan with sales sticker
n02979186_13346	Person holding boombox	n02979186_15096	Couple holding boombox
n02979186_9394	Interior of record and CD store	n02979186_15568	Cassettte tape
n02979186_2785	Cameras with boombox in back	n02979186_11299	Boombox
n02979186_13265	Car interior with cassettee player	n02979186_10211	Keyboard and Cassette Player
n02979186_13510	Woman with boombox	n02979186_3321	Commodore Cassette player
n02979186_4266	Cassette deck with clock	n02979186_809	Woman listening to cassette player
n02979186_16253	Small cassette player	n02979186_13701	Car cassette player
n02979186_16277	Broken cassette deck	n02979186_14158	Man changing cassette in walkman
n02979186_465	Car with date stamp	n02979186_465	Car with date stamp
n02979186_2839	A picture of back of a truck	n02979186_13346	Person holding boombox
n02979186_10289	Broken portable cassette player	n02979186_13086	Cassette player interior
n02979186_351	Small car	n02979186_2839	A picture of back of a truck
n02979186_14158	Man changing cassette in walkman	n02979186_351	Small car
n02979186_6176	Portable cassette player back	n02979186_6176	Portable cassette player back
n02979186_2883	Car interior	n02979186_15335	Doll with cassette player
n02979186_15335	Doll with cassette player	n02979186_13510	Woman with boombox
n02979186_13166	Old cassette player	n02979186_2883	Car interior
n02979186_24591	Zoom in of Walkman	n02979186_13740	Woman with boombox
n02979186_809	Woman listening to cassette player	n02979186_603	Minivan with sales sticker
n02979186_13740	Woman with boombox	n02979186_13289	Cassette tape unraveled
n02979186_603	Minivan with sales sticker	n02979186_24591	Zoom in of Walkman
n02979186_13289	Cassette tape unraveled	n02979186_21014	Car cassette player
n02979186_1823	Car with cassette player removed	n02979186_11063	cassette tape stereo systems
n02979186_11957	My first sony cassette player	n02979186_174	Cassette deck cleaner

Supplemental Table 4: This figure compares the top 30 most confusing training images of "cassette player" from influence functions [17] or GTDA. Both method can find some common training images that are indeed ambiguous. However, it will take influence functions almost 4 hours to compute influence for all 12,894 training images while GTDA only takes about 1 minute to process the entire dataset.



Supplemental Figure 9: (Reproduction of Extended Data Figure for self-contained supplementary notes.) Reeb net on the 10 easy classes of ImageNet created by the original TDA framework. TDA is directly applied to the ResNet image embedding matrix without transforming into KNN graph. Unlike GTDA visualization, we cannot find any obvious subgroups other than 10 major components representing 10 classes or the labeling issues of some "cassette player" images. Moreover, no information can be extracted at all for around 28% images as they are either in some very small Reeb net components or simply considered as noise by the clustering scheme.

ResNet-50 and VOLO. AlexNet is one of the historical CNN models, with around 60% top-1 testing accuracy. ResNet is one of the most widely used CNN models nowadays with a better performance of about 75% top-1 testing accuracy. Finally, VOLO is one of the state of art CNN models that achieves about 87% top-1 testing accuracy without using any additional training data. Then, for each CNN model, we extract the prediction matrix and the image embeddings. For AlexNet and ResNet, the image embeddings are the outputs of the layer before final prediction layer. Similar to the previous section, we replace the average pooling by max pooling in the last convolutional layer. For VOLO, we directly used the dedicated feature forwarding function to get image embeddings. Similar to previous sections, all image embeddings are PCA reduced to 128 with whitening and normalization. For GTDA parameters, we set $K = 25$, $d = 0.001$, $r = 0.01$, $s_1 = 5$, $s_2 = 5$, $\alpha = 0.5$ and $S = 10$. We use 10 steps of iterations for GTDA error estimation.

	AlexNet v.s. ResNet	ResNet v.s. VOLO
original graph nodes	1,331,167	1,331,167
original graph edges	5,954,900	5,805,714
Reeb nodes	63,239	68,354
Reeb edges	59,881	64,360
Reeb components	3,395	4,046
max Reeb component size	169	79
max Reeb node size	330	643
average Reeb components for each class	3.5	4.0

Supplemental Table 5: Statistics on Reeb nets. Reeb node size is the number of samples represented in a Reeb net node. Average Reeb components for each class is the average number of Reeb net components where the most frequent predicted label (by one of the two models) is that class. The maximum Reeb component just has a few hundred of nodes, which guarantees that any component of the Reeb net can be easily visualized and analyzed.

§4.2 Building graphs and initial results of GTDA

We first compare AlexNet and ResNet. To do so, we build a 5-NN graph using the image embeddings of ResNet only. Then we concatenate the prediction matrix of AlexNet and ResNet to get 2,000 lens. GTDA framework is then applied using the same set of parameters as Section §3. Similarly, to compare ResNet and VOLO, we build a 5-NN graph using the image embeddings of VOLO and concatenate the prediction matrix of ResNet and VOLO. In Table 5, we provide some initial statistics on the final Reeb nets. We can see that despite the Reeb net has tens of thousands of nodes, the maximum Reeb component size is just a few hundred of nodes, which guarantees that we can easily visualize any component of the Reeb net.

§4.3 Utility

The utility of GTDA for this scenario is that it can be used to analyze difference in how the methods might be working or what errors might be expected. For instance, we could study compare the the predictions between ResNet and VOLO on "desktop computer". Both models have very similar training or validation accuracy on this class. But they make mistakes in different places. Using our visual taxonomy, we could identify possible scenarios for these mistakes.

§5 Understanding Malignant Gene Mutation Predictions

In this section, we apply our method to inspect model predictions of gene sequence variants effects. A gene sequence variant means that some part of the DNA sequence for this gene is mutated compared with the reference. Modifications include single nucleotide variation, deletion, duplication, etc. We study a model proposed to predict whether such variant is harmful or not [1]. In the following section, we will provide details on the model and the dataset we use. Then we will show that the model's prediction is highly correlated with both gene variants coordinates as well as mutation types. We also discover abnormal places that could imply unreliable labels.

§5.1 Dataset and model

The model we use is recently proposed to predict gene expression from DNA sequence by integrating long-range interactions [1]. In this model, a consecutive DNA sequence of 196,608bp is used to predict 5,313 human genome tracks. For each gene variant, we follow the same steps as proposed by [1] to compute its embedding. First, we extract the reference and alternate DNA sequences from homo sapiens (human) genome assembly, either hg19 or hg38 as specified by the gene variant record. This gives a 393,216bp long DNA sequence with the centered on the VCF position (Variant Call Format). Note that for the alternate sequence, the gene variant is applied first before extracting the modified sequence. Then, we directly use the pretrained model from [1] to make predictions on the reference and alternate sequences. This model will aggregate the center 114,688bp into 128-bp bins of length 896. The prediction for each 128bp bin is a 5,313 vector, where each element represents the predicted gene expression in one of the 5,313 genome tracks for the human genome (including 2,131 transcription factor chromatin immunoprecipitation and sequencing tracks, 1,860 histone modification tracks, 684 DNase-seq or ATAC-seq tracks and 638 CAGE tracks). The prediction vector of the 4 128bp bins located in the center is then summed together to get a prediction vector for the reference or alternate sequence. After that, the elements in each prediction vector corresponding to the CAGE tracks is $\log(1 + x)$ transformed. Finally, we compute the difference of preprocessed prediction vectors between reference and alternate sequences as the final embedding for the gene variant. In total, we get a 23,376-by-5,313 embedding matrix for 23,376 gene variant records. Then, a linear classifier will be trained on this 5,313 difference vector to predict variants effects. The original paper uses the training and testing datasets from CAGI5 competition [33], where a Lasso regression is trained to predict a label of -1 (significant downregulating effect), 0 (very little to no effect on expression) or +1 (significant upregulating effect). We were not able to download the dataset from the official CAGI5 competition website. Therefore, we use similar procedure to predict harmful (label 1) vs non-harmful (label 0) gene mutations from ClinVar. We download gene variants experiments from the official ClinVar website [19]. We choose all experiments that are targeting **BRCA1** as it is one of the genes with the most number of experiments and part of the protein it encodes has known 3D structures (i.e. **1JNX**). Gene variants without a valid VCF (variant call format) position are removed. As for the labels, we directly use the "ClinSigSimple" field as the label of each gene variant record. An integer 1 means at least one current record indicates "Likely pathogenic" or "Pathogenic",

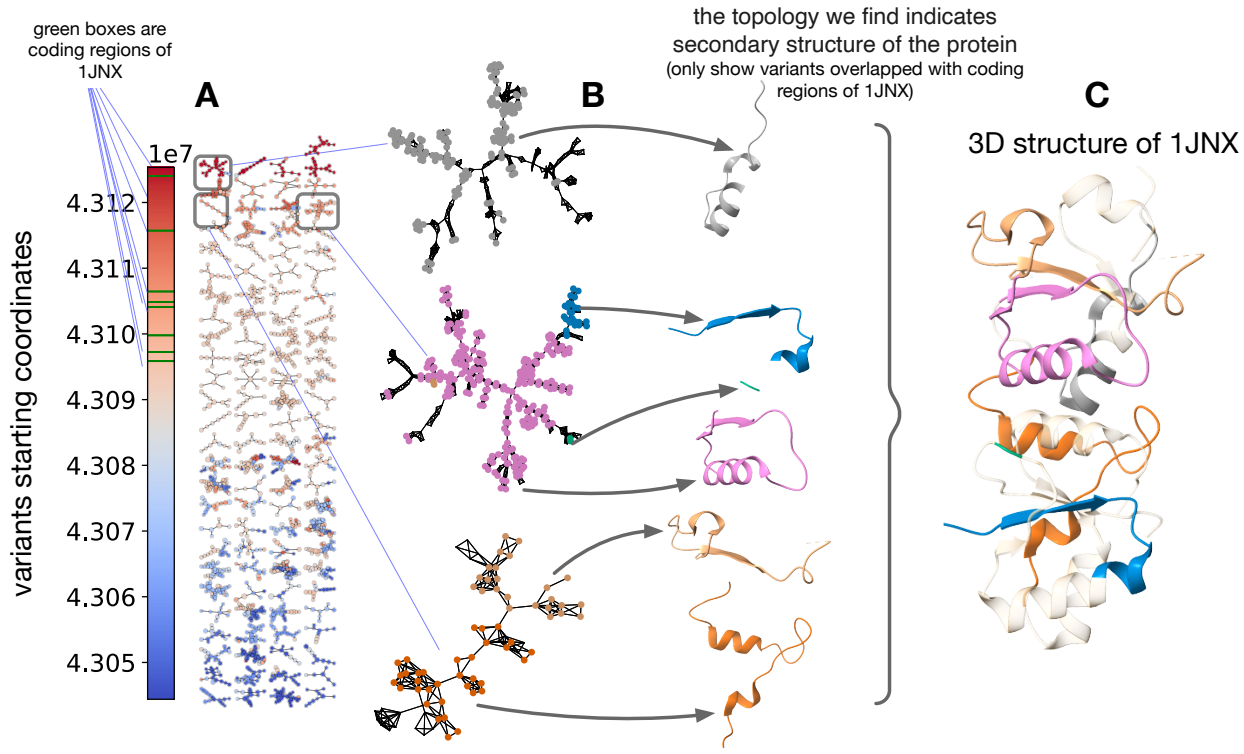
but doesn't necessarily mean this record includes assertion criteria or evidence. An integer 0 means there are no current records of "Likely pathogenic" or "Pathogenic". An integer -1 means no clinic significance and is replaced by label 0 in our experiments. And we use a logistic regression with L1 penalty since this is a binary prediction. We include 23,376 gene variants where 50% of them are used as training, and the other 50% are used as testing. To build the graph for GTDA, the embedding matrix is PCA reduced to 128 dimensions with PCA whitening and then each row is ℓ_2 normalized. A 5-NN graph is constructed on the preprocessed embedding matrix with cosine similarity. This 5-NN graph has some small components smaller than the threshold set by s_1 and s_2. As a result, 338 out of 23,376 gene variants ($\sim 1.4\%$) are not included in the final Reeb net; this is not expected to impact the results. We use 2 prediction lens and the first 2 PCA lens of the embedding matrix for GTDA analysis. For GTDA parameters, we set $K = 30$, $d = 0$, $r = 0.05$, $s_1 = 5$, $s_2 = 5$, $\alpha = 0.5$ and $S = 10$. We use 20 iterations for GTDA error estimation.

§5.2 Validating the GTDA visualization

The visualization we get from this dataset is shown in Figure 10. The first finding is that different components in this visualization are strongly related to different regions of the DNA sequence. Such a result is not surprising because this model aims to predict gene expressions from a long range of DNA sequence while most gene variants will only change one or two base pairs. Therefore, it is expected that gene variants close to each other in coordinates will also get similar embeddings. To further validate whether this visualization can capture finer 3D protein structures, we check the crystal structure of the BRCT repeat region (PDB id is **1JNX**), also shown in plot (C) of Figure 10. In total, **BRCA1** encodes a protein with 1863 amino acids. And **1JNX** covers amino acids from 1646 to 1849. In the color bar of Figure 10, we mark the protein coding regions (exons) of **1JNX** in green. In (B) of Figure 10, we check a few components in detail that contains gene mutation locations overlapped with the green area. Each green area represents an exon. Different node colors are assigned based on which exon they overlap with. [1] We can find that different local structures of this crystal are also very well localized in our visualization. All these findings suggest that the model's embedding space has a strong correlation with VCF (variant call format) positions of gene variants and GTDA can capture such property successfully.

Statistical validation We conduct 10000 random experiments to see if such strong location sensitivity can be found in a random graph. In each experiment, we shuffle the embeddings and rebuild the KNN graph. The PCA lenses and prediction lenses are kept the same. Then we run GTDA on each of the 10000 random graphs. We consider a random graph to shows location sensitivity if in the results of GTDA, one component has more than 40 mutation samples that overlap with exons and more than half of them are overlapped with the same exon. We were not able to find any random graph that can pass this criteria in these experiments.

[1] 4 out of 2756 mutation samples overlap with more than 1 exons of 1JNX, we color those based on the first exon they overlap with.



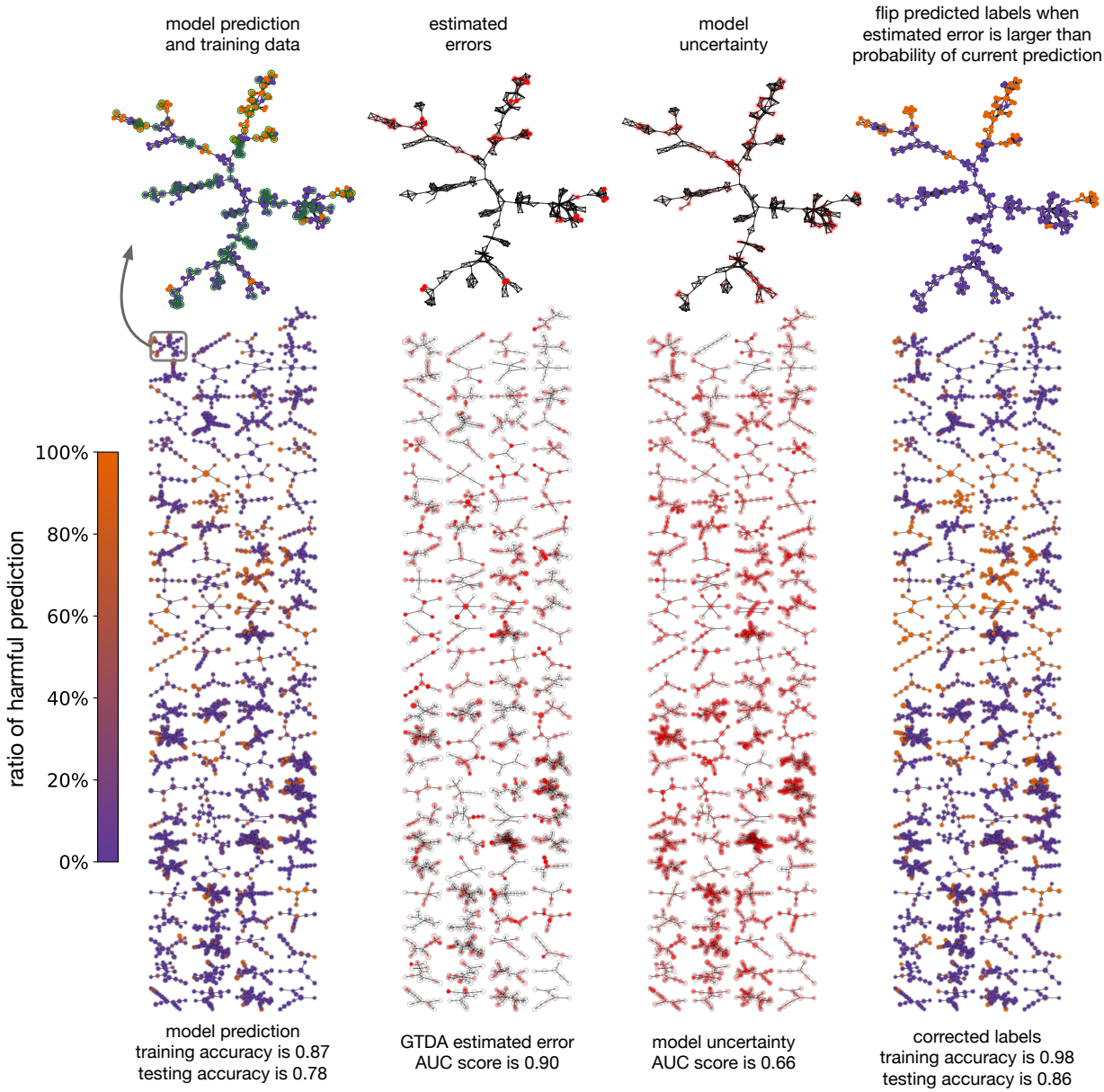
Supplemental Figure 10: (A) shows components found by GTDA, where each node is colored by median hg38 coordinates of mutation starting positions. Different components are ordered by the averaged median coordinates in a zig-zag fashion from lower right to upper left. We zoom in a few components where the gene variants have the highest overlap ratio with the coding regions of **1JNX** (B). Different node colors are assigned based on which consecutive protein coding region they overlap with. Nodes for gene variants not in the coding regions of 1JNX are not plotted. We can find that different secondary structures of the crystal of **1JNX** (C) are also well separated in the GTDA visualization.

§5.3 Estimating and correcting prediction errors

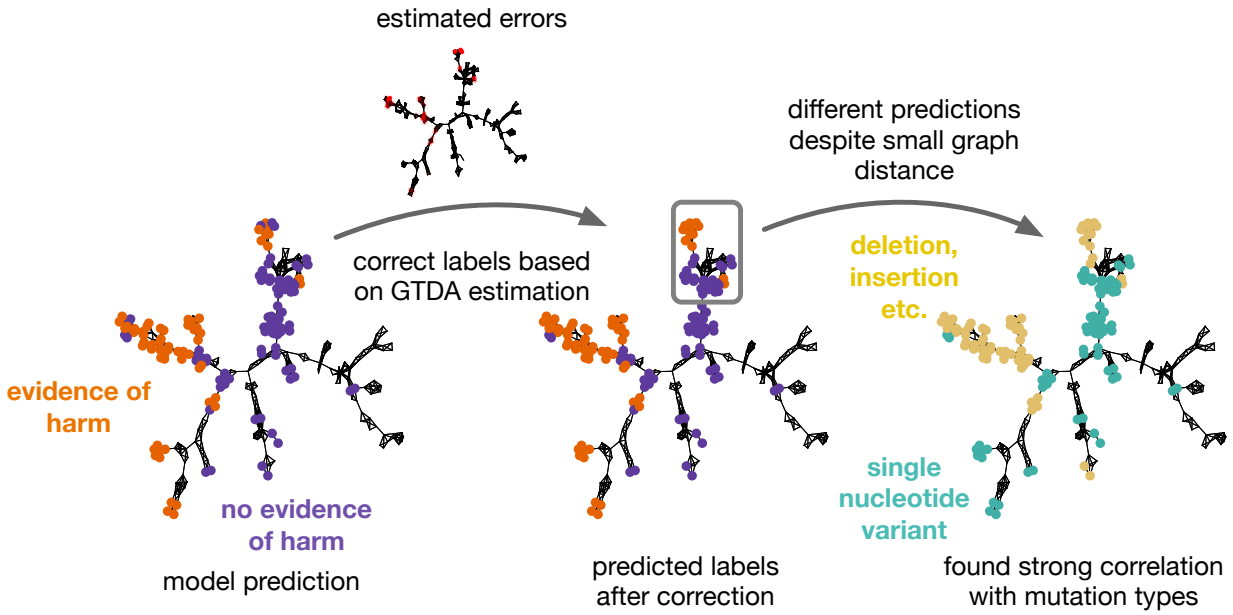
We apply Algorithm 4 to estimate errors of model prediction Figure 11. Overall, GTDA estimated errors (after normalizing to 0 to 1) achieve an AUC score of 0.90. In comparison, using model uncertainty gives an AUC score of 0.66. Since this is a binary classification, we can also flip predicted labels if they are more likely to be errors. Instead of setting a single threshold, we flip predicted labels when the estimated errors are larger than the probability of the current prediction. The corrected labels can improve training accuracy from 0.87 to 0.98 and testing accuracy from 0.78 to 0.86.

§5.4 Extracting insights about mutation types and single nucleotide variants

As we explore model predictions for gene mutations happening inside protein encoding regions, i.e., green boxes in Figure 10, we find different predicted labels for mutations that target a small area of the protein structure. One such example is Figure 12, where records in the grey box happen in a small region of the protein structure with around 17 amino acids. So there should be other aspects that help the model make different predictions.



Supplemental Figure 11: (Reproduction of Main Figure for self-contained supplementary notes.) In the top part, we zoom in a component and mark training data using green circles. Then we show GTDA estimated errors and model uncertainty on this component. We flip predicted labels if the estimated error is larger than the prediction probability. In the lower part, we can see GTDA error estimation has much better overall AUC score and the corrected labels have higher training and testing accuracy.



Supplemental Figure 12: We zoom in one component GTDA finds and only show mutation records that happen in the protein coding regions (non-coding regions are not shown as colored dots, but do impact the Reeb net structure). After correcting prediction based on GTDA error estimation, we still see records that happen in a small region of the protein but still get different predictions. By checking these records, such difference can be well explained by different mutation types.

By checking the actual mutation record, we find the non-harmful mutations are all single nucleotide variant (SNV), while harmful mutations are other types of mutations including deletion, insertion or duplication. This makes sense as the latter types will not only affect the current amino acid, but also the subsequent amino acids and hence cause more substantial changes to the final protein structure.

Overall, we find for gene mutations that are predicted harmful (after GTDA correction) and target gene encoding regions, 70% of them are mutations like deletion, insertion or duplication. For gene mutations that are predicted as non-harmful and target gene encoding regions, only 6% are mutations like deletion, insertion or duplication. When including gene mutations outside protein encoding regions as well, 72% of harmful predictions are mutations like deletion, insertion or duplication, while that number drops to 5% for non-harmful predictions.

We assess the statistical significance of the relationship between single nucleotide variants (SNV) and harmful predictions for each component GTDA identifies in Table 6. The associated Chi-square p-values highlight a few components where this association is missing, such as component 100 with 34 non-harmful non-SNV results throughout the entire BRCA1 structure (coding and non-coding regions), with a p value of 0.22. This suggests a difference in behavior for this component in comparison with the remainder of the components. Other large p-values include the nearby components 99 and 101, along with component 3, 26. Overall, this highlights another way these GTDA results could be used.

	Prediction and Type (coding regions of 1JNX)				Chi-square p-value	Prediction and Type (all)				Chi-square p-value
	Harmful	Harmful	non-Harm	non-Harm		Harmful	Harmful	non-Harm	non-Harm	
Component	SNV	non-SNV	SNV	non-SNV		SNV	non-SNV	SNV	non-SNV	
0	11	6	83	4	5.1e-04	13	6	167	8	1.2e-04
2	0	0	0	0	-	17	264	49	16	1.2e-36
3	1	3	99	5	1.8e-05	12	3	230	9	2.5e-02
4	0	0	0	0	-	13	14	181	4	1.2e-16
6	0	10	10	2	-	24	38	298	20	2.7e-27
7	0	0	0	0	-	6	42	152	24	1.5e-22
8	0	4	0	0	-	16	82	96	22	6.2e-21
9	0	0	0	0	-	6	23	44	11	4.9e-07
10	0	0	0	0	-	17	102	129	22	1.0e-30
14	0	2	2	0	-	13	31	297	19	7.6e-30
15	6	2	40	0	-	25	146	437	24	8.7e-90
16	0	0	0	0	-	6	155	136	13	3.9e-53
17	5	14	49	4	6.5e-08	55	93	485	23	2.7e-59
18	0	0	2	0	-	9	7	115	3	7.5e-08
19	0	8	0	2	-	36	70	422	32	1.0e-44
21	0	6	64	2	-	20	31	376	17	4.7e-33
22	12	16	102	2	4.2e-13	32	20	188	6	1.1e-12
25	0	0	0	0	-	15	17	19	3	7.7e-03
26	0	0	0	0	-	34	4	256	12	2.4e-01
28	0	1	0	1	-	29	42	63	12	1.7e-07
29	3	6	193	18	8.1e-07	9	9	339	31	1.3e-07
30	0	0	0	0	-	19	57	93	9	6.4e-19
31	0	0	0	0	-	16	18	68	8	4.3e-06
32	0	4	2	0	-	5	23	51	5	1.0e-10
33	10	55	64	11	5.5e-16	16	55	204	23	1.1e-28
34	16	18	32	0	-	32	70	66	12	3.5e-12
36	2	4	250	6	1.8e-12	8	11	314	23	3.2e-12
37	0	0	0	0	-	40	76	26	14	1.5e-03
38	0	0	0	0	-	21	37	137	19	8.6e-14
39	0	0	0	0	-	14	198	24	14	3.4e-18
40	0	2	6	0	-	50	81	488	13	1.5e-63
41	0	0	2	0	-	16	14	158	6	1.1e-11
44	0	0	0	0	-	27	29	423	17	4.2e-30
46	0	16	30	2	-	19	36	51	10	2.0e-07
48	0	4	20	4	-	30	14	220	16	3.1e-06
50	0	0	62	0	-	4	36	262	10	2.2e-45
52	0	4	18	2	-	60	67	830	79	5.7e-40
53	0	0	0	0	-	15	27	303	25	2.7e-22
54	0	0	10	0	-	19	27	365	13	2.5e-32
55	0	0	0	0	-	21	19	109	3	2.8e-11
56	0	0	0	0	-	5	34	29	4	9.4e-10
57	0	0	0	0	-	40	18	78	6	4.5e-04
58	2	2	0	0	-	25	30	157	10	2.3e-15
59	0	0	0	0	-	17	44	163	6	6.6e-28
60	6	0	36	2	-	6	7	130	13	2.8e-05
62	2	33	12	7	1.9e-05	12	69	218	35	8.1e-33
64	2	25	114	7	5.0e-22	6	25	192	17	4.4e-22
66	0	4	24	0	-	27	23	165	9	1.9e-12
67	0	6	0	0	-	9	30	111	4	1.0e-20
68	21	13	87	9	3.3e-04	76	108	570	92	3.8e-36
69	2	3	78	7	4.4e-03	4	11	314	35	8.6e-12
70	0	0	0	0	-	40	48	6	6	1.0e+00
71	0	0	0	0	-	17	8	269	18	4.4e-05
72	0	0	12	0	-	6	5	318	21	1.7e-05
73	0	0	0	0	-	12	9	142	3	3.1e-10
74	0	0	0	0	-	19	11	119	15	1.5e-03
77	6	20	14	4	1.1e-03	13	33	213	11	6.0e-28
78	0	0	2	0	-	3	8	181	6	4.1e-16
79	10	4	52	0	-	33	10	203	6	3.3e-06
81	0	0	0	0	-	9	57	95	15	9.5e-21
82	0	0	0	0	-	8	4	212	6	1.5e-05
85	0	1	38	1	-	14	61	496	29	5.3e-65
88	0	0	0	0	-	3	34	269	16	6.8e-41
90	2	4	76	2	4.4e-07	10	4	162	4	1.0e-04
99	0	0	8	0	-	6	3	100	7	3.3e-02
100	0	0	2	0	-	4	6	64	34	2.2e-01
101	0	2	2	0	-	6	4	60	6	2.8e-02
102	0	0	0	0	-	7	9	109	5	5.3e-09
Overall	148	344	2114	122	2.9e-259	1506	3986	16208	1338	0.0e+00

Supplemental Table 6: For each component in the Reeb networks, 2 contingency tables are computed, where the left table only considers variants in the coding regions of 1JNX and the right table considers all variants. Only components where each cell of the right table has a count 3 or higher are included. Chi-square p-values are computed for tables where each cell has a count larger than 0.

§5.5 Incorrect GTDA error estimation implies unreliable labels

When we compared the GTDA error estimation with true errors, we found a few places where the GTDA estimate is entirely wrong.

To understand this abnormality, in Figure 13 we zoom in a few components and use green circles to mark training and validation data. We show the GTDA estimated errors as well as the false estimations when comparing to the true errors. We can see a few false error estimates in each of these components. And on checking those false estimations, we find they are either testing experiments with insignificant or conflicting results or affected by nearby insignificant training experiments.

To understand this effect across all components found by GTDA, we use the difference between the true presence of an error and our estimate. For instance, if GTDA estimation on whether a prediction is wrong is 0.3 and the prediction is indeed wrong based on its true label, such difference will be 1 minus 0.3. In total, we can find 2,031 GTDA error estimations where such difference is larger than 0.5. These are spread over 771 Reeb nodes. Since an error estimation being wrong can be due to either its own label being unreliable or training samples nearby have unreliable labels, we study how many of those 771 Reeb nodes have at least 1 insignificant or conflicting samples (either training or testing sample). We find 662 of them (81%) have at least one problematic label. Consequently, the intuition from Figure 24 would hold across much of the dataset.

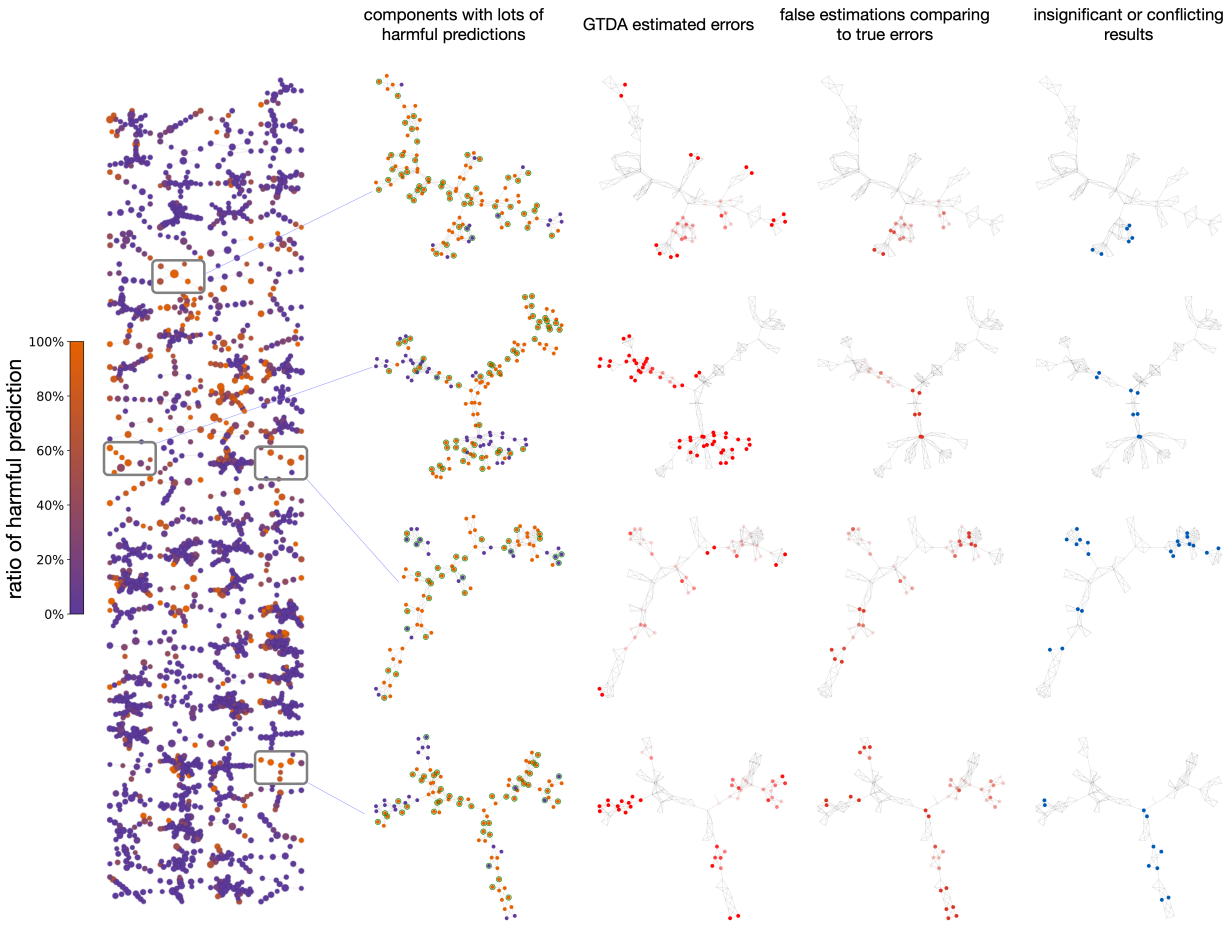
§5.6 Comparison with other methods

In Figure 4of the main text, we have visually compared the visualization of GTDA with other methods including Mapper, UMAP and tSNE. We have shown that the visualization of GTDA clearly shows the location sensitivity of mutation samples in the DNA sequence that is not quite obvious in the visualizations of other methods. In this section, we quantify this visual advantage of GTDA. We first convert UMAP and tSNE visualizations into graphs by building a 5-NN graph on top of the 2 dimensional embedding. For GTDA and Mapper, we project each Reeb net node using Algorithm 5 to get the corresponding graphs. We also add the original KNN-graph that has been used as input to GTDA and 100 random graphs by shuffling edges for comparison. Then we design the following metrics:

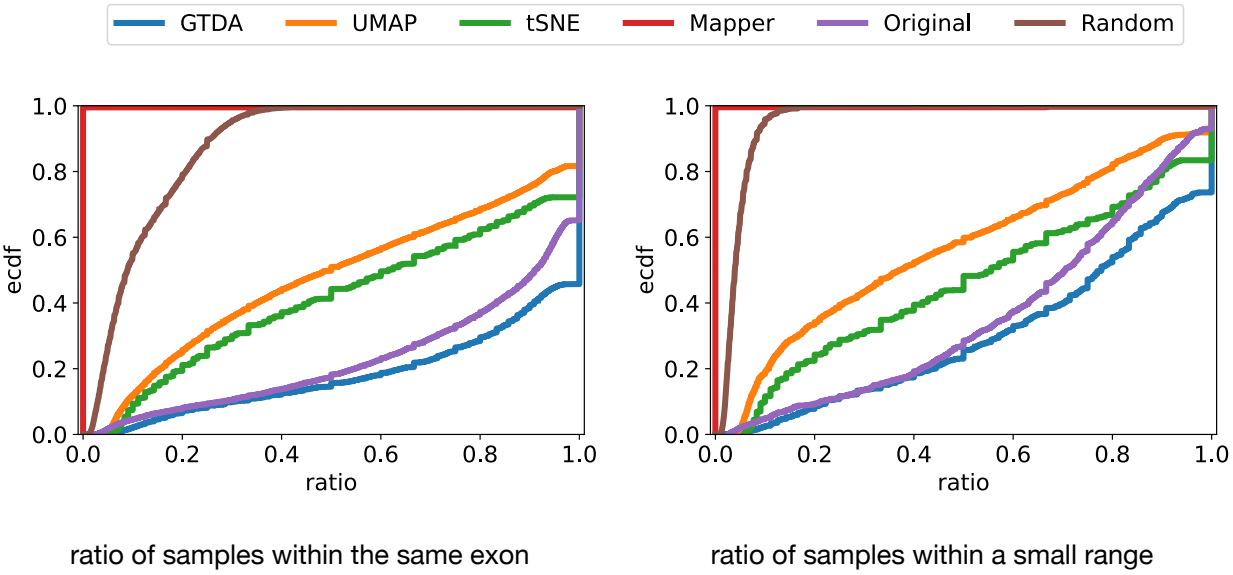
- ratio of samples within the same exon: in this metric, for each mutation sample that overlaps with an exon, we search the neighbors within 3 hops on each graph and compute the ratio of mutation samples that overlap with the same exon. Note that we only consider exons that encodes 1JNX.

- ratio of samples within a small range: in this metric, for each mutation sample, we search the neighbors within 3 hops on each graph and compute the ratio of mutation samples whose mutation starting coordinates are within 1000 base pairs of the starting coordinate of the selected mutation sample.

We also consider the corresponding ratio to be zero if the number of neighbors within 3 hops is smaller than 5. This is because the visualization of Mapper has too many single nodes or tiny components which could result in better metrics despite the visualization itself is much worse. In Figure 14, we compare the empirical cumulative distribution function of the



Supplemental Figure 13: (Reproduction of Main Figure for self-contained supplementary notes.) Checking false error estimations of GTDA in some components reveals that they are likely to be caused by variants experiments with insignificant or conflicting results.



Supplemental Figure 14: Overall GTDA performs the best on both metrics, while the other methods are not clearly better or even worse than the original graph. This suggests (1) the strong location sensitivity of mutation samples indeed exist in the original graph (2) GTDA can not only preserve and enhance such location sensitivity, but also visualize such property easily.

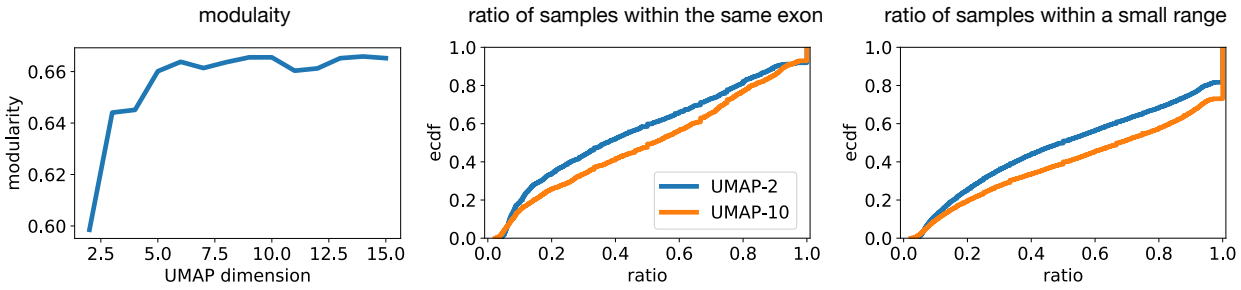
	GTDA vs tSNE		GTDA vs UMAP		GTDA vs Mapper	
ratio within the same exon	0.23	$p < 10^{-10}$	0.35	$p < 10^{-10}$	0.99	$p < 10^{-10}$
ratio within a small range	0.33	$p < 10^{-10}$	0.40	$p < 10^{-10}$	0.97	$p < 10^{-10}$

Supplemental Table 7: The ks statistics and p-value of the one tailed Kolmogorov–Smirnov test. The null assumption is that the ecdf of GTDA is larger than the ecdf of other methods at all locations. The p-values were extremely small or numerically 0 in floating point, which we report as less than 10^{-10}

ratio distributions. For each of the 100 random graphs, we compute the ecdf and report the average of the 100 ecdf curves. We can first notice that comparing to random graphs, the ratio in both metrics is much higher in the original graph, which means mutation samples are indeed significantly localized in the original graph. Also, GTDA performs the best on both metrics., which can be verified by the Kolmogorov–Smirnov test in Table 7.

One key advantage of topological based methods like GTDA is that it visualize the embedding space by directly simplifying it without reducing the dimensions too much. In the case of mutation dataset, the KNN graph that will be used as the input of GTDA is constructed on a 128 PCA reduced embedding space. In comparison, tSNE or UMAP try to project the original embedding space on only 2 dimensions, which could cause huge information loss. We find that using more dimensions in UMAP can better preserve the location sensitivity. To show this, we increase the embedding dimensions on UMAP and rebuild the KNN graph using only mutation samples that overlap with exons. To study how mutation samples from different exons will be localized, we consider samples within each exon as a community and compute the graph modularity [24]. A higher modularity score means these communities are better localized in the graph. As we can see in Figure 15, the modularity

score increases as we use more dimensions until after 10 dimensions. UMAP embedding in higher dimension also performs better on the two metrics we designed. However, using more than 2 dimensions on these types of methods will make the subsequent visualization difficult or impossible. We were not able to replicate the same experiment on tSNE as the running time of tSNE becomes extremely long when setting the dimension larger than 2.



Supplemental Figure 15: In the first plot, we compute the modularity by considering mutation samples that overlap with the same exon as a community. This plot shows that the modularity increases as we use more dimensions in the output of UMAP, which suggests these communities are less mixed in the corresponding KNN graph of UMAP embeddings in higher dimensions. In the second and third plots, we compare the two metrics we designed between UMAP embedding in 2 dimensions and UMAP embedding in 10 dimensions. Higher dimension also performs better on both metrics. More specifically, in one tailed Kolmogorov–Smirnov test, the ks statistics and p-value are 0.11 and 3.0^{-16} for ratio within the same exon and such numbers become 0.11 and 9.8^{-132} for ratio within a small range, showing that UMAP-10 is better localized than UMAP-2.

§6 Inspecting chest X-ray images

In this section, we apply our GTDA framework to inspect the prediction of disease on 112,120 images of chest X-rays [41]. Each X-ray image might be either normal or indicating one or more diseases. Our results show that GTDA is very useful to help radiologists detect images with incorrect normal and abnormal labels.

§6.1 Dataset and model

The NIH ChestX-ray14 dataset we use comprises 112,120 de-identified frontal-view X-ray images of 30,805 unique patients [41]. Among these images, 86,524 images are used as training or validation and the others are used as testing. Images are split at the patient level, which means images belonging to the same patient will be put in the same group. Among the 86,524 images, we randomly choose 20% patients and use their associated images as validation data while images for the other patients are used as training data. In the original dataset, a text mining approach is used on the associated radiological reports to find the existence of 14 possible diseases and one image can have multiple disease labels. As a result, it is expected that many of the labels assigned are incorrect. In some other studies of these data, expert labels are solicited for 810 selected testing images from multiple experienced radiologists [23].

The model we use GTDA to study is called CheXNet [27] which is a 121-layer Dense Convolutional Network (DenseNet) [14]. When applying our GTDA framework, we first reduce the 14 disease predictions to a simple normal (label 0) vs abnormal (label 1) prediction. To do so, we first take a row wise maximum to reduce the prediction matrix for 14 disease into a vector v with values ranging 0 to 1. Then we consider each individual value as a threshold and generate predicted labels by treating values larger than this threshold as 1 or 0 otherwise. Then we compute the F1 score using the union of training and validation data. The threshold that gives the largest F1 score will be kept, denoted as t. Similar procedures have been used in other papers that predict ontological annotations [7, 18]. Finally, we transform each value of v using $v_i = \min(1, 0.5v_i/t)$. The transformed v also ranges from 0 to 1 and is considered as the probability of being abnormal. As a result, the row wise maximum column index of the new prediction matrix $\boldsymbol{P} = [1 - v, v]$ will give the same largest F1 score. Other than the abnormal vs normal lens, we also include the original disease prediction matrix as the extra lenses. This process gives 16 lenses in total. For GTDA parameters, we set $K = 50$, $d = 0$, $r = 0.005$, $s_1 = 5$, $s_2 = 5$, $\alpha = 0.5$ and $S = 10$. We use 10 iterations for GTDA error estimation.

§6.2 GTDA finds incorrect normal vs abnormal labels

Out of the 810 images in the test set with expert labels, 222 images have incorrect normal vs abnormal labels. Our goal is to use the GTDA visualization to find images in this set (i.e. those that are more likely to an incorrect label). The procedure of finding those images is similar to find insignificant or conflicting gene mutation experiments from the previous section.

We first use GTDA to estimate prediction errors. The estimation is normalized to a

Supplemental Figure 16: (Reproduction of Main Figure for self-contained supplementary notes.) We give a demonstration on how to use GTDA results to find which testing labels are likely to be problematic. We first zoom in a component found by GTDA and use green circles to mark testing images where we have expert labels (A). Then we use GTDA to estimate prediction errors on circled images (B). Comparing GTDA estimation with original testing labels can identify a few places with false estimations (C). We consider these false estimations are due to problematic testing labels and do a simple thresholding of 0.5, which flags 17 problematic testing labels in this component (D). Comparing to expert labels can find 14 true positives with a precision of 0.82 and a recall of 0.78 (E).

number between 0 and 1. Then we use the original testing labels (i.e. without the correction from experts) to find which of these error estimates are wrong. We can then sort the test samples in the order of descending absolute difference between estimated error and true error.

For simplicity, images in the test set where such differences are larger than 0.5 are considered to have incorrect labels. A demonstration on this process can be found in Figure 16. Overall, out of the 810 testing images with expert labels, GTDA highlights 265 images are likely to have incorrect normal vs abnormal labels and 138 of them are confirmed by the expert labels, which gives a precision of 0.52 and a recall of 0.62. As a comparison, randomly sampling 265 images for experts to check can only find around 73 images with incorrect labels in average. More detailed results on each component are shown in Table 8. By testing multiple thresholds instead of 0.5, we get an AUC score of 0.75. As a comparison, using self confidence [25] gives an overall AUC score of 0.60.

Type	Expert Labels in Component	Incorrect by Experts	Flagged as Problematic	Precision	Recall
Single Component	53	18	17	0.82	0.78
Single Component	10	5	5	1.0	1.0
Single Component	9	5	4	0.25	0.2
Single Component	19	4	7	0.57	1.0
Single Component	9	4	5	0.8	1.0
Single Component	10	4	3	0.33	0.25
Single Component	7	4	2	1.0	0.5
Single Component	8	4	5	0.6	0.75
Single Component	14	4	4	1.0	1.0
Single Component	4	4	2	1.0	0.5
Single Component	7	4	3	0.33	0.25
Single Component	10	3	2	0.0	0.0
Single Component	6	3	1	0.0	0.0
Single Component	4	3	2	0.5	0.33
Single Component	6	3	3	0.33	0.33
Single Component	3	3	2	1.0	0.67
Single Component	5	3	3	1.0	1.0
Single Component	5	3	2	0.5	0.33
Single Component	8	3	5	0.4	0.67
Single Component	7	3	4	0.5	0.67
Single Component	19	3	8	0.25	0.67
Single Component	9	3	8	0.38	1.0
Single Component	8	3	3	0.33	0.33
Single Component	8	3	4	0.5	0.67
Components with 2 incorrect labels	135	56	50	0.74	0.66
Components with 1 incorrect label	219	67	78	0.5	0.58
Components with 0 incorrect label	208	0	33	0.0	NaN
Overall	810	222	265	0.52	0.62

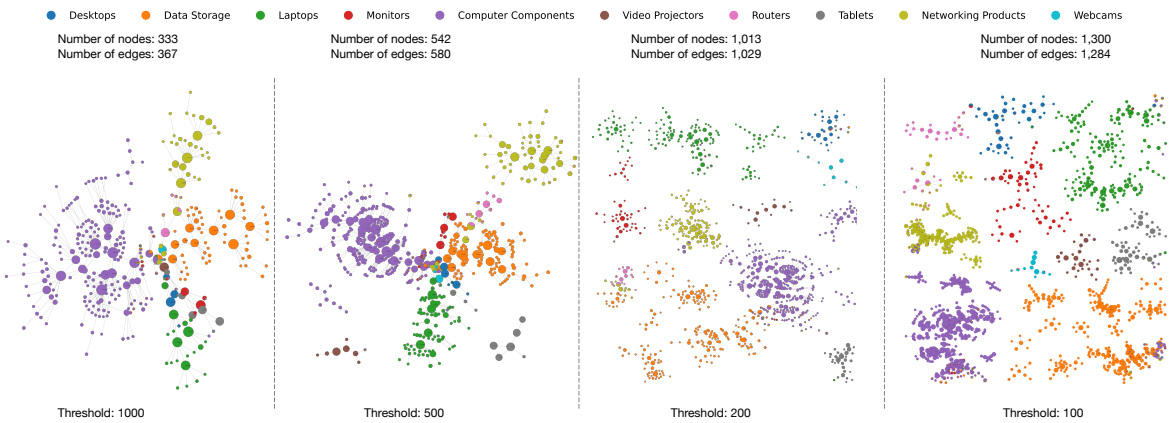
Supplemental Table 8: Detailed precision and recall on different components when using GTDA to find likely incorrect testing labels of ChestX-ray14 dataset. Components are ordered by decreasing number of incorrect labels identified by experts they contain. Results for components with less than 3 incorrect labels are reported together.

§7 Parameter selection of GTDA

In this section, we will discuss how to select parameters for our GTDA framework, especially the component size threshold and overlapping ratio in Algorithm 1. We have designed our default parameters to be consistent with existing ideas and theories of TDA. Currently, we manually focus the Reeb net's structure by varying these parameters. It remains an open question on how one might automatically select parameters for our GTDA framework as proposed for other TDA frameworks [5]. Although GTDA has 8 parameters (Table 1), the two most important are the component size threshold and the overlapping ratio.

§7.1 Selecting component size threshold

Recall that the component size threshold is the smallest component where we stop splitting. Choosing a good component size threshold depends on the dataset we want to analyze.



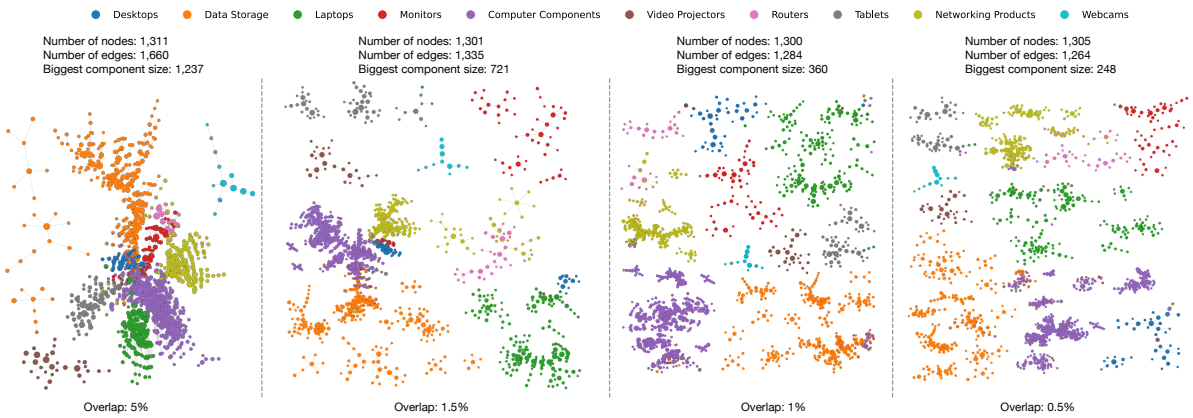
Supplemental Figure 17: We show different GTDA visualizations as we vary the component size threshold. The overlapping ratio is fixed as 1%. Using a large threshold will cause different classes to be mixed together and the structure of small class like "Routers" or "Webcams" will be over simplified. As we gradually reduce the thresholds, the number of nodes and edges in the visualization will increase as well and different classes will be separated into several components. The results look similar between 100 and 200, which suggests GTDA structure are stable with respect to small change in parameters.

If the threshold is too small, we will end up with too many nodes to make the subsequent visualization and analysis difficult. On the other hand, if the threshold is too large, the topological structure of some small classes might be over simplified and components from different classes can be mixed. Figure 17 shows how the Reeb net will change as we vary component size threshold. Topological theory would suggest making this as large as possible, so we start from a larger component size threshold and then check the Reeb net we get, especially the size of the largest Reeb net component as well as whether different classes are mixed. If we have a component that is too large to be easily visualized or different classes are clearly mixed, we reduce this value. If the class sizes are highly skewed, we usually choose the threshold based on the smallest class. In this case, the lower bound on the absolute difference of the lens parameter becomes useful. This is to avoid oversplitting class with large size, i.e., if the difference is smaller than the lower bound, we stop splitting as well.

We find the results are stable to the choices and in particular, for uses to find clues of possible predicting errors or labeling issues from the visualization. As we can see in Figure 17, choosing a threshold between 100 and 200 or choosing an overlapping ratio between 0.5% and 1.5% can all show the ambiguity in "Networking Products" v.s. "Routers" and some part of "Data Storage" v.s. "Computer Components" allowing human insight into the predictions.

§7.2 Select overlapping ratio

Consistent with the mapper theory, we want a goldilocks overlapping ratio: not too large to connect everything and not too small to prohibit connections. The selection of overlapping ratio is similar to selecting the component size threshold, we can start from a larger ratio like 10% and then check the Reeb net to see if there is any component that is too large or too mixed. If so, we need to gradually reduce the ratio until every component can be properly visualized by a simple layout algorithm like spring layout [39] or Kamada Kawai algorithm [16]. Figure 18 shows different Reeb nets as we vary overlapping ratio.



Supplemental Figure 18: We show different GTDA visualizations as we vary the overlapping ratio. The component size threshold is fixed as 100. Using a large overlapping ratio will cause different classes to be mixed together and some components too large to be properly visualized. As we gradually reduce the overlapping ratio, different classes will be separated into several components with each one easier to be plotted. Similar ambiguity in "Networking Products" v.s. "Routers" and some part of "Data Storage" v.s. "Computer Components" can be observed for overlapping ratio between 0.5% and 1.5%.

§7.3 Merging Reeb nodes and components

Although the choice of merging function f has the potential for a large impact, we regard the choice of the l^∞ distance between the prediction lenses as natural. The value of lenses is used to determine overlap. So finding the closest nodes in terms of the l^∞ distance records links that were split but would not have been at higher levels of overlap. This is the most consistent choice with Reeb graph construction perspective. Alternatives here would include domain specific measures of similarity or other measures that might include representation invariant comparison among data [3].

§7.4 Notes on other parameters

Other than component size threshold and overlapping ratio, Algorithm 1 has several other parameters. Two important ones are the smallest node size and the smallest component size. In our experiments, we can get consistently good visualizations by requiring the size of any Reeb net node or Reeb net component to be larger than 5.

§8 Performance and scaling

Our GTDA framework scales to predictions with thousands of classes and millions of datapoints. We only split along the lens with the maximum difference at each iteration, which can be easily recomputed in linear time in the data or even more efficiently updated. After each split, we immediately check all the connected components we have found, which can be done in $O(N + M)$ where N is the number of nodes and M is the number of edges.

It is difficult to estimate how many splitting iterations are needed. Assuming we have L lenses, initially the min-max difference across all lenses is 1 and the overlapping ratio is 0, then we will need at most L iterations before the largest min-max difference across all components

is reduced to 0.5, which means at most $O(tL)$ iterations are needed to reduce such difference below 2^{-t}. If after a sufficient number of iterations, we still see large components with size bigger than K, it means new lenses are needed to further distinguish those nodes or a lower bound on the difference is needed to stop the splitting early.

Another step is to find out which pairs of components have overlap. This can be easily done in the original *mapper* algorithm by checking the adjacent bins of each bin. In our GTDA framework, we first build a bipartite graph with all component indices on one side, all samples on the other side and connecting each component index to all samples it includes. Then identifying the overlapping components is equivalent to find 2-hop neighbors of each component index, which can also be done in $O(N+M)$. Finally, for the merging step, since the size of each super node or the size of Reeb net component will be at least doubled, it needs at most $O(M(\log(s_1 s_2)))$ time. Also note that, many steps of our GTDA algorithm can be easily parallelized. In our code, we mainly parallelize the merging steps using 10 cores, which has already given reasonable running time on graphs with millions of nodes and edges.

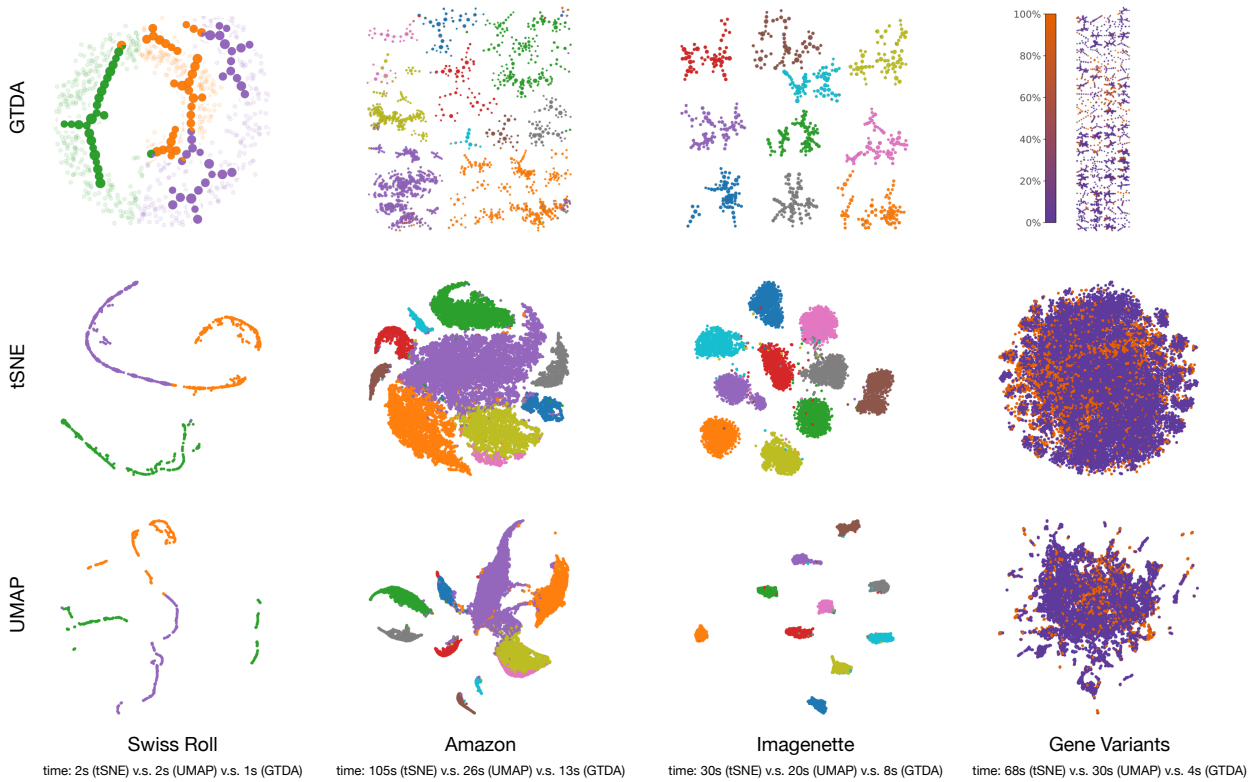
Detailed running time for all datasets we have tested can be found in the table 9. All running time are reported on a server with 2 AMD EPYC 7532 processors (128 cores in total), 512 GB memory and one A100 GPU.

dataset	nodes	edges	classes	lens	predicting & embedding (s)	preprocessing (s)	GTDA time (s)
Swiss Roll	1,000	3,501	3	3	0.003	0.3	1
Amazon Computers	39,747	399,410	10	10	0.17	7	10
Subset of ImageNet	13,394	51,520	10	10	27	5	7
ImageNet-1k (ResNet vs AlexNet)	1,331,167	5,954,900	1,000	2,000	2,379	717	26,036
ImageNet-1k (VOLO vs ResNet)	1,331,167	5,805,714	1,000	2,000	13,426	617	18,894
BRCA1 Gene Variants	23,376	83,096	2	4	18,583	21	3
Chest X-rays	112,120	431,893	2	16	821	35	26

Supplemental Table 9: (Reproduction of Extended Data Table for self-contained supplementary notes.) Statistics on datasets and running time in seconds. Predicting and embedding represents the time used to generate prediction and extract embedding for all samples from a trained model. Preprocessing time includes PCA, normalization as well as building a KNN graph if the original dataset is not in graph format. GTDA time is the time to compute Reeb network given the input graph and the lens.

§9 Comparing to tSNE and UMAP

The goals of the Reeb net analysis from GTDA are distinct from the goals of dimension reduction techniques such as tSNE and UMAP. We seek the topological information identified by the Reeb net without reducing dimensions. The Reeb net is both useful for generating pictures or maps of the data as well as the algorithmic error estimate. We use the Kamada-Kawai [16] method to compute a visualization of the Reeb net, which does have many similarities with summary pictures from tSNE and UMAP. We compare here GTDA



Supplemental Figure 19: (Reproduction of Main Figure for self-contained supplementary notes.) Comparing the results of the dimension reduction techniques tSNE and UMAP on 4 datasets to the topological Reeb net structure from GTDA shows similarities and differences among summary pictures generated by these methods. The graph created by GTDA permits many types of analysis not clearly possible with tSNE and UMAP output. For running time comparison, since we also need to extract model embeddings and predictions just like GTDA, we exclude such time and only report the time of the actual execution of tSNE or UMAP or GTDA (including Kamada-Kawai).

results with visualization from tSNE [40] and UMAP [2, 22] on all 4 datasets of the main text. For tSNE, we directly use the implementation from *scikit-learn*. For UMAP, we use the implementation from `https://umap-learn.readthedocs.io`. The inputs to tSNE and UMAP are the concatenation of neural model embedding and prediction probability. We keep all parameters as default except setting the number of final dimension as 2. The results are shown in Figure 19.

These pictures support different uses and purposes. Reeb nets from GTDA offer a number of compelling advantages as described throughout the main text and supplement. Among others, note that GTDA is faster than tSNE (2 to 15 times faster) and UMAP (2 to 8 times faster) in all 4 datasets. It also scales easily to datasets with millions of datapoints.

§10 Comparing error estimation with and without GTDA

The error estimate returned by Algorithm 4 needs as input a graph and a set of labeled nodes. The input we provide arises from the Reeb graph projection, Algorithm 5. In principle, however, it can utilize any graph. In this section, we investigate what changes when we use the original graph that was input to GTDA instead of the projected Reeb graph.

As an additional point of comparison we also use a randomly modified graph with the same number of edges as the projected Reeb graph.

We evaluate the error using two different measures. The first is *precision* at the number of true errors. Recall that we are predicting errors. Let T be the number of true errors among the predictions. Let L be the list of predictions of errors. Then the "Pr" or precision measure, is the precision of the first T entries of L. That is, the number of accurate predictions among the top T entries. This is designed to evaluate roughly what a human might inspect where they keep going as long as they keep finding errors. The second measure is the standard AUC.

The precision results are in Table 10 and the AUC results are in Table 11. Note that the precision values are better for the projected Reeb graph (the GTDA entry in the table) for all studies except Gene mutation. This is likely because the Gene mutation experiment has unreliable labels (see Section §5.5). The differences are most pronounced when GTDA removes a large number of edges, as in the Amazon and Imagenette cases.

In terms of AUC, the results for the GTDA graph are worse (except barely different from Imagenette) – and can even be worse than a random edge reduction of the same level. We believe these results can be explained by the removal of edges. Removing these edges limits the method's ability to identify certain classes of extremely hard to predict errors (basically those right at the boundary that could have either prediction). Because the original graph contains these extra edges, the AUC score will be slightly higher due to the less focused predictions that result. The random edge removal is unlikely to systematically remove these edges, and so scores slightly higher. That said, the precision results are more in line with our experience in using these methods. We illustrate these ideas in Figure 20.

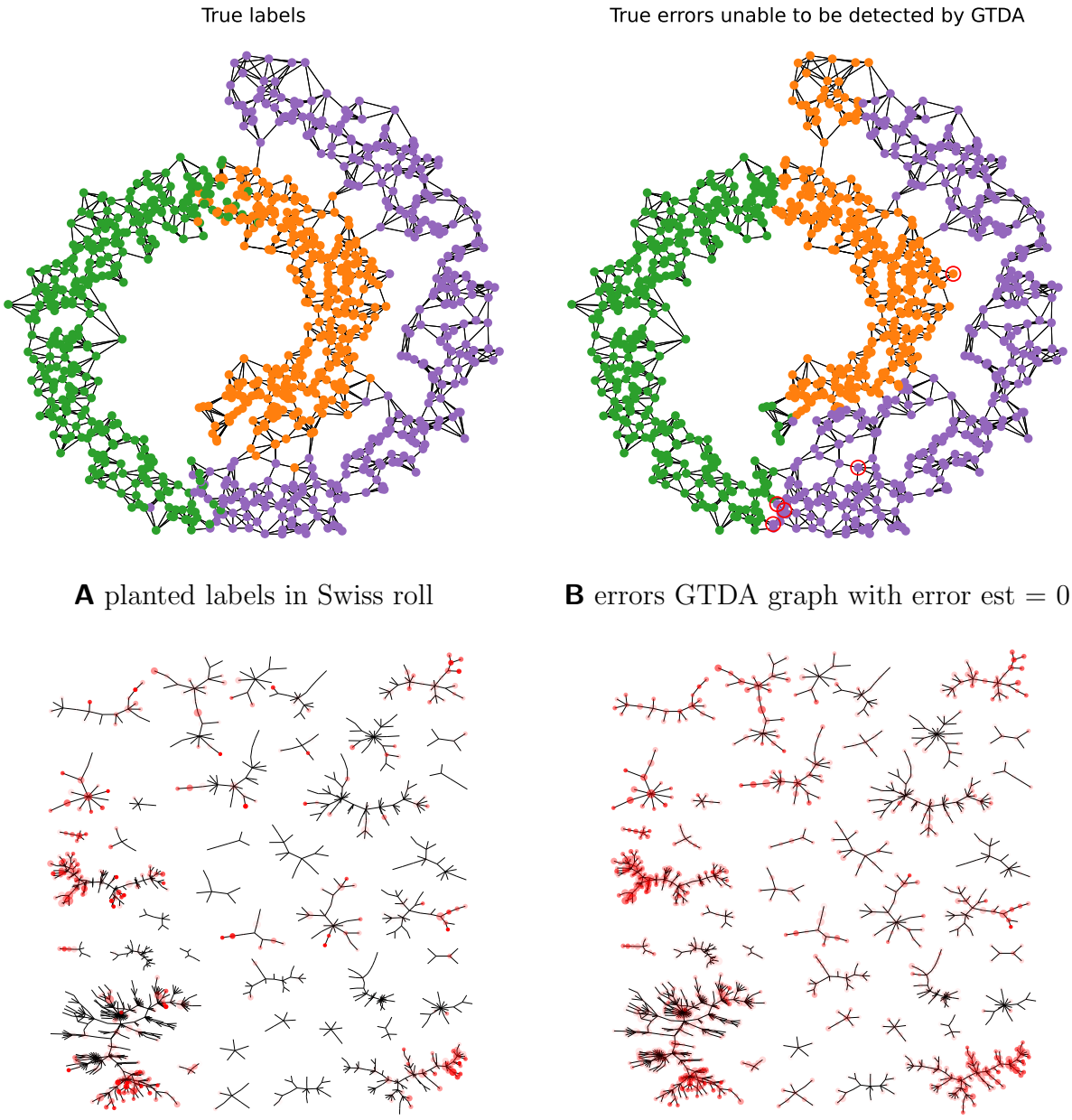
More broadly speaking, we note that the GTDA layout and visualization can present any error measure in a fairly intuitive fashion. There are also more sophisticated label propagation routines that might be able to further improve these results [15]. Finally, there are a number of ideas that would involve using these error prediction methods discussed in Section §1.7.

References

[1] Žiga Avsec, Vikram Agarwal, Daniel Visentin, Joseph R Ledsam, Agnieszka Grabska-Barwinska, Kyle R Taylor, Yannis Assael, John Jumper, Pushmeet Kohli, and David R Kelley. Effective gene expression prediction from sequence by integrating long-range interactions. *Nature methods*, 18(10):1196–1203, 2021. Cited on page 27.

[2] Etienne Becht, Leland McInnes, John Healy, Charles-Antoine Dutertre, Immanuel W H Kwok, Lai Guan Ng, Florent Ginhoux, and Evan W Newell. Dimensionality reduction for visualizing single-cell data using UMAP. *Nature Biotechnology*, 37(1):38–44, December 2018. Cited on page 43.

[3] Mattia G. Bergomi, Patrizio Frosini, Daniela Giorgi, and Nicola Quercioli. Towards a topological–geometrical theory of group equivariant non-expansive operators for data analysis and machine learning. *Nature Machine Intelligence*, 1(9):423–433, September 2019. Cited on pages 4 and 41.



A planted labels in Swiss roll

B errors GTDA graph with error est = 0

C predicted errors with the GTDA graph in Amazon are focused

D predicted errors with the original graph in Amazon are spread throughout

Supplemental Figure 20: We illustrate how sparsity in the GTDA graph causes predictions to be more focused, but also miss a few challenging predictions, resulting in a lower AUC score but higher precision score. In (A) we show the intended planted labels for the Swiss role dataset. In (B) we show the predictions from the GCN method and we circle mistakes where the GTDA error prediction probability is numerically 0. The value of 0 arises because the GTDA graph in this region has been disconnected from other relevant predictions. These mistakes cannot be inferred with our technique, giving a lower AUC score. These mistakes are at challenging interfaces among classes, although there are others that are equally challenging. In (C) and (D) we show the distribution of error predictions between the GTDA-based graph and the original graph on the Amazon graph. This illustrates how using the GTDA graph focuses the predictions on a much smaller region.

	Problem							
	Swiss Roll		Amazon		Imagenette		Gene mutation	
	Pr	Edges	Pr	Edges	Pr	Edges	Pr	Edges
original graph	0.821	100%	0.482	100%	0.789	100%	0.769	100%
GTDA graph	0.830	73.7%	0.530	15.8%	0.868	36.6%	0.750	75.6%
random removal	0.808	73.7%	0.352	15.8%	0.666	36.6%	0.760	75.6%

Supplemental Table 10: We evaluate a precision metric for our estimated error metric that is derived from Algorithm 4 as we vary the graph used as input. The value "Pr" is the precision of the prediction list limited to the length of the number of true errors. Put another way, this is about how precise the measure might appear to someone investigating the list. The value of edges indicates how many edges are in the graph. These results show that the pruning done with GTDA focuses on the most likely errors. This is largely successful, except for the Gene mutation dataset, which suffers from a label quality issue, see Section §5.5.

	Problem							
	Swiss Roll		Amazon		Imagenette		Gene mutation	
	AUC	Edges	AUC	Edges	AUC	Edges	AUC	Edges
original graph	0.978	100%	0.874	100%	0.9996	100%	0.932	100%
GTDA graph	0.961	73.7%	0.841	15.8%	0.9997	36.6%	0.905	75.6%
random removal	0.976	73.7%	0.685	15.8%	0.9913	36.6%	0.928	75.6%

Supplemental Table 11: We evaluate the AUC for our estimated error metric that is derived from Algorithm 4 as we vary the graph used as input. This shows that the AUC measure is slightly lower for the GTDA graph and also possibly sometimes lower than random removal. This occurs because the GTDA predictions focus more on the most likely errors (see Table 10) and show higher precision for the level of the true number of errors. The other graphs cause the method to make more diffuse predictions and which leads to slightly higher AUC scores integrated over the complete set of predictions.

[4] Cristian Bodnar, Cătălina Cangea, and Pietro Liò. Deep graph mapper: Seeing graphs through the neural lens. *Frontiers in big Data*, 4, 2021. Cited on page 2.

[5] Mathieu Carriere, Bertrand Michel, and Steve Oudot. Statistical analysis and parameter selection for mapper. *The Journal of Machine Learning Research*, 19(1):478–516, 2018. Cited on pages 13 and 39.

[6] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations*, 2021. Cited on pages 14 and 16.

[7] Wyatt T Clark and Predrag Radivojac. Information-theoretic evaluation of predicted ontological annotations. *Bioinformatics*, 29(13):i53–i61, 2013. Cited on page 37.

[8] Tamal K Dey, Facundo Mémoli, and Yusu Wang. Multiscale mapper: Topological summarization via codomain covers. In *Proceedings of the twenty-seventh annual acm-siam symposium on discrete algorithms*, pages 997–1013. SIAM, 2016. Cited on page 3.

[9] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, volume 96, pages 226–231, 1996. Cited on page 23.

[10] David F Gleich. Pagerank beyond the web. *SIAM Review*, 57(3):321–363, 2015. Cited on page 3.

[11] Mustafa Hajij, Paul Rosen, and Bei Wang. Mapper on graphs for network visualization. *arXiv preprint arXiv:1804.11242*, 2018. Cited on page 2.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2016. Cited on page 17.

[13] Jeremy Howard. Imagenette dataset. `https://github.com/fastai/imagenette`, 2021. Cited on pages 17 and 19.

[14] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, pages 4700–4708, 2017. Cited on page 37.

[15] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin Benson. Combining label propagation and simple models out-performs graph neural networks. In *International Conference on Learning Representations*, 2021. Cited on pages 10 and 44.

[16] Tomihisa Kamada, Satoru Kawai, et al. An algorithm for drawing general undirected graphs. *Information processing letters*, 31(1):7–15, 1989. Cited on pages 4, 40, and 42.

[17] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. PMLR, 2017. Cited on pages 23 and 24.

[18] Maxat Kulmanov and Robert Hoehndorf. Deepgoplus: improved protein function prediction from sequence. *Bioinformatics*, 36(2):422–429, 2020. Cited on page 37.

[19] Melissa J Landrum, Jennifer M Lee, Mark Benson, Garth R Brown, Chen Chao, Shanmuga Chitipiralla, Baoshan Gu, Jennifer Hart, Douglas Hoffman, Wonhee Jang, et al. Clinvar: improving access to variant interpretations and supporting evidence. *Nucleic acids research*, 46(D1):D1062–D1067, 2018. Cited on page 27.

[20] Meng Liu, Tamal K. Dey, and David F. Gleich. Topological structure of complex predictions. *arXiv*, cs.LG:2207.14358, 2022. Cited on page 21.

[21] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pages 43–52, 2015. Cited on page 14.

[22] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv*, stat.ML:1802.03426, 2018. Cited on page 43.

[23] Zaid Nabulsi, Andrew Sellergren, Shahar Jamshy, Charles Lau, Edward Santos, Atilla P Kiraly, Wenxing Ye, Jie Yang, Rory Pilgrim, Sahar Kazemzadeh, et al. Deep learning for distinguishing normal versus abnormal chest radiographs and generalization to two unseen diseases tuberculosis and covid-19. *Scientific reports*, 11(1):1–15, 2021. Cited on page 37.

[24] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113, Feb 2004. Cited on page 35.

[25] Curtis G. Northcutt, Lu Jiang, and Isaac L. Chuang. Confident learning: Estimating uncertainty in dataset labels. *Journal of Artificial Intelligence Research (JAIR)*, 70:1373–1411, 2021. Cited on pages 8 and 38.

[26] Xuebin Qin, Zichen Zhang, Chenyang Huang, Masood Dehghan, Osmar R Zaiane, and Martin Jagersand. U2-net: Going deeper with nested u-structure for salient object detection. *Pattern Recognition*, 106:107404, 2020. Cited on page 21.

[27] Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Ding, Aarti Bagul, Curtis Langlotz, Katie Shpanskaya, et al. Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. *arXiv preprint arXiv:1711.05225*, 2017. Cited on page 37.

[28] Ali S Razavian, Josephine Sullivan, Stefan Carlsson, and Atsuto Maki. Visual instance retrieval with deep convolutional networks. *ITE Transactions on Media Technology and Applications*, 4(3):251–258, 2016. Cited on page 21.

[29] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016. Cited on page 19.

[30] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. Cited on page 23.

[31] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 618–626, 2017. Cited on page 19.

[32] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018. Cited on pages 14, 17, and 19.

[33] Dustin Shigaki, Orit Adato, Aashish N Adhikari, Shengcheng Dong, Alex Hawkins-Hooker, Fumitaka Inoue, Tamar Juven-Gershon, Henry Kenlay, Beth Martin, Ayoti Patra, et al. Integration of multiple epigenomic marks improves prediction of variant impact in saturation mutagenesis reporter assay. *Human mutation*, 40(9):1280–1291, 2019. Cited on page 27.

[34] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In Yoshua Bengio and Yann LeCun, editors, *ICLR (workshop track)*, 2014. Cited on page 19.

[35] Gurjeet Singh, Facundo Mémoli, and Gunnar E Carlsson. Topological methods for the analysis of high dimensional data sets and 3d object recognition. *SPBG*, 91:100, 2007. Cited on page 1.

[36] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR (workshop track)*, 2015. Cited on page 19.

[37] B. Strodthoff and B. Jüttler. Layered reeb graphs for three-dimensional manifolds in boundary representation. *Computers & Graphics*, 46:186–197, 2015. Shape Modeling International 2014. Cited on page 11.

[38] Edward Tufte. *Seeing with fresh eyes: Meaning, Space, Data, Truth.* Graphics Press, 2020. Cited on page 19.

[39] William Thomas Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, 3(1):743–767, 1963. Cited on page 40.

[40] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. Cited on page 43.

[41] Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald M Summers. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2097–2106, 2017. Cited on page 37.

[42] Bin Zhao, Fei Li, and Eric Xing. Large-scale category structure aware image categorization. In *Advances in Neural Information Processing Systems*, volume 24, 2011. Cited on page 14.

[43] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016. Cited on page 19.