VISUALIZING POINT CLOUD CLASSIFIERS BY MORPHING POINT CLOUDS INTO POTATOES

Anonymous authors
Paper under double-blind review

ABSTRACT

Recently, various networks that operate directly on point clouds have been proposed. It is of interest to us what features are utilized in those classifiers for their predictions. In this paper, we propose a novel approach to visualize important features used in classification decisions from point cloud networks. Following ideas in visualizing 2-D convolutional networks, our approach is based on gradually smoothing parts of the point cloud to remove certain shape features, and then evaluating the resulting point cloud on the original network to see whether the performance has dropped or remained the same. From these it can be seen whether certain parts are important to the point cloud classification. A main technical contribution of the paper is to propose an algorithm for smoothing point cloud shapes based on moving least squares and curvature flow. This algorithm can smoothly transition from the original point cloud to a either a uniform sphere, or a disk if the original shape is on a plane. With this algorithm, we can obtain a saliency map by adapting the Integrated-Gradients Optimized Saliency (I-GOS) algorithm, a state-of-the-art perturbation-based visualization techniques, to 3-D shapes. Experiment results revealed insights into these classifiers.

1 INTRODUCTION

Recently, direct deep learning on unstructured 3-D point clouds has gained significant interest. Many point cloud networks have been proposed. PointNet and PointNet++ utilizes max-pooling followed by multi-layer perceptron PointConv (Wu et al., 2019) realizes a real convolution operation on point clouds. DGCNN (Wang et al., 2018) builds on PointNet++ (Qi et al., 2017) by learning features from edges instead of vertices. SPLATNet (Su et al., 2018) embeds features into a high-dimensional lattice and applies convolution on the lattice. Other works such as (Xu et al., 2018; Atzmon et al., 2018; Li et al., 2018; Fey et al., 2018; Tatarchenko et al., 2018) all have their own merits. As with 2-D image classifiers, we are curious about what indeed these models have learned. Following (Fong & Vedaldi, 2017)'s definition of *explanations* as meta-predictors, we want to *explain* those models by identifying which parts of a shape contribute most to the final score, and which parts the least.

A natural representation of this *explanation* is a saliency map, which associates each point in the point cloud with an importance score. To show that a saliency map is valid, following the *deletion* and *insertion* metric proposed by (Petsiuk et al., 2018), we should expect the predicted score to drop quickly when we "cover up" those parts with highest importance score from the network, and to rise quickly when we gradually "reveal" *only* those parts with highest importance score to the network.

Here we put "cover up" and "reveal" in quotes because they have not been defined yet on 3-D data. It is easy to "cover up" some parts of a 2-D image: simply turn those pixels into grey or black, or apply the a significant Gaussian blur to those pixels. It is not easy to extend this notion to 3-D point clouds, since however we move the points, they will always be part of the point cloud, and thus contributing to the underlying shape. Current point-based deep networks may not be robust enough to generalize to new point clouds after these operations. For example, not all point cloud networks accept inputs with varying number of points, hence unable to adapt deleting points. Prior work (Zheng et al., 2018) proposed an approximation of point deletion to simply moving those "deleted" points to the median position of the point cloud. However, their argument for the validity of this operation is true only when the network has a max-pooling layer that directly operates on point positions. Such an assumption cannot be made for a model-agnostic algorithm. Additionally, during the process of moving the points toward the median position, extra unnatural geometric structures appear, e.g., a car might suddenly have a pointy bump on its surface pointing inward. This non-smooth data is not within the training distribution of the point cloud network, hence their performance on it are undefined and will likely suffer. In practice, we often see point cloud classifiers give significantly lower predictions on point clouds with such unnatural geometric structures, which may work for the task of generating adversarial examples in (Zheng et al., 2018), but does not bring real understanding of the features those networks use to classify the point cloud.

Our goal is to perform this "cover up" process in a manner so that the resulting point cloud is still part of the training distribution. For this, we attempt to smoothly morph the 3-D shape to remove distinctive shape features. As an example, for an airplane one thought would be to smoothly eliminate the wings to some other shape. Such kind of smoothing and fairing have been well-established on 3-D meshes. However, we have not found a satisfactory approach that directly applies on point clouds, which usually have very sparse and irregular sampling distribution.

In this paper, we propose a new algorithm for smoothing point clouds. For each point in the point cloud, we fit a local plane from its neighborhood. Under some assumptions we prove that the distance from the point to its local plane can be used to approximate the local curvature. This allows us to utilize a mean-curvature-flow-based algorithm similar to (Desbrun et al., 1999) to smooth the shape. Our new algorithm does not rely on explicit edges which are not available in point clouds, and is practically capable of smoothing many different shapes to a sphere with constant mean curvature.

With the new smoothing tool, we adapt a recent 2-D heatmap algorithm called I-GOS (Qi et al., 2019) onto point clouds. We experiment our method on PointConv (Wu et al., 2019) and DGCNN (Wang et al., 2018), two state-of-the-art point cloud networks. Results on the ModelNet40 dataset reveals that, different from image-based networks that often classify based on a small distinctive feature, point-based networks usually rely heavily on the entire shape to classify (usually more than 50% points need to be inserted for the score to be close to the original classification score, a proportion higher compared to 2-D images). However, certain important parts can be found so that once distorted, the score will drop quickly. Also, symmetry is very important for the networks to recognize certain classes. We believe that these results improve our understanding of those networks and may help improving their training in the future.

2 Related Work

Classifier visualization Using saliency maps to visualize networks has attracted much research effort these years. There are two main categories of approaches: gradient-based and perturbation-based. Gradient-based approaches regard the gradients of the output score with respect to the input as the standard of measuring the contribution of the input ((Simonyan et al., 2013; Zeiler & Fergus, 2014; Springenberg et al., 2014; Bach et al., 2015; Shrikumar et al., 2016; Sundararajan et al., 2017). Perturbation-based methods, on the other hand, perturbs the input and see which part of the input has the largest influence on the output. Object detectors in CNNs (Zhou et al., 2014), Real Time Image Saliency (Dabkowski & Gal, 2017), Meaningful Perturbation (Fong & Vedaldi, 2017), RISE (Petsiuk et al., 2018) and I-GOS (Qi et al., 2019) all belong to this family.

As far as we know none of these methods have been tried on 3-D point cloud classifiers. (Zheng et al., 2018), is the only prior work we know that attempts to visualize point cloud networks. (Zheng et al., 2018) uses a gradient-based approach and calculates the gradients of the output score with respect to the straight line from median to the input points and regards those gradients as saliency. As mentioned in the introduction, their method is justified when the network has a max-pooling layer, where points shifted to the median would have no impact on the classification. Unfortunately this is not true for many point cloud networks, e.g. SpiderCNN (Xu et al., 2018) uses average pooling, while DGCNN (Wang et al., 2018) maxpools on edges instead of points. We build a model-agnostic approach hence cannot adopt their strategy.

A close relative of visualization is adversarial attack. Recent works on adversarial attack on 3-D point cloud classifiers inlcude (Xiang et al., 2019) and (Liu et al., 2019). Their approaches usually include shifting existing points negligibly, or adding to the shape a small set of points that can be hidden in the human psyche. The difference between visualization and adversarial attack is that in visualization, we aim to stay as close as possible to the training distribution in order to not mislead the classifier, which is generally quite brittle outside the training distribution. Adversarial attacks have no such constraints hence can fully exploit the brittleness of networks outside the training distribution. The *insertion* metric proposed in (Petsiuk et al., 2018) is a nice approach to evaluate whether a mask is adversarial or not, since it hinges on the ability of the classifier to successfully classify the object using only part of its features. Attacks generally create patterns that are not semantically meaningful, hence the classifier exposed only to those patterns is usually not possible to recover the correct category.

3-D shape morphology There has been active research in smoothing and fairing 3-D structures. For mesh smoothing, (Taubin, 1995) has proposed a method based on diffusion and signal processing, and proved it to serve as a low-pass filter and is anti-shrinkage. However, as (Desbrun et al., 1999) pointed out, this diffusion method is flawed due to its unrealistic assumption about meshes. (Desbrun et al., 1999) proposed a scheme based on curvature flow, where a local "curvature normal" is computed at each vertex and the diffusion is based on it. Meshes are easier to smooth than point clouds because they provide readily estimated planes that can be used to compute curvature. Some noise-removal scheme that directly operates on point clouds were proposed in (Alexa et al., 2001) and (Mederos et al., 2003). Most of these methods are based on moving least-squares (Levin, 1998) with a local plane/surface fitting. However, the goals of these approaches are mainly removing noises, rather than gradually morphing the shape to one with constant curvature as in our goal.

In terms of mathematical morphology, several work aimed to extend the well-known 2-D morphological operations such as dilation / Minkowski sum to point clouds (Calderon & Boubekeur, 2014; Lien, 2007). In (Calderon & Boubekeur, 2014), a point set surface is fitted for the point cloud to get a signed distance function (SDF) representation for the point cloud, and then a point structuring element (PSE), which is a SDF itself, is fitted for each point using mean shift. Finally, the morphological projection of the point can be computed using the PSE. (Lien, 2007) proposed a purely point-based approach for defining the Minkowski sum for point clouds, which is fast and simple. In their approach, a structuring element (SE) is a set of vectors. For each point in the point cloud, all the vectors in the SE are added to it to get a new set of points. Then a decimation step is taken to remove the points inside the boundary. However, most of them require the shape to be closed and orientable, i.e., have an "inside" and an "outside", an assumption we did not make since some of the 3-D points could form a 2-D plane with no interior.

3 METHODS

Throughout this paper we work on a point cloud with N points, denoted as $P = \{p_1, \dots, p_N\}$, where $p_i \in \mathbb{R}^3$ is a 3-tuple of x, y, z coordinates. Denote a neighborhood of p_i as $\mathcal{N}(p_i)$ and K as the size of the neighborhood.

3.1 SMOOTHING POINT CLOUDS

Our goal is to smoothly morph a point cloud into a new shape with constant mean curvature, such as a sphere, in that all the shape features such as edges and corners in the original point cloud would be eliminated. If the shape is already on a plane, then we aim to smooth it into a disk (so that its boundary has constant curvature). The total number of points should not change, and each point should be traceable from its initial position to its final position. In the following we first describe the classical Taubin smoothing Taubin (1995), then describe our algorithm.

3.1.1 TAUBIN SMOOTHING

The local Laplacian at a vertex p_i is linearly approximated using the umbrella operator:

$$L(p_i) = \frac{1}{K} \sum_{j \in \mathcal{N}(p_i)} (p_j - p_i). \tag{1}$$

This approximation assumes that the mesh unit-length edges and equal angles between two adjacent edges around a vertex (Desbrun et al., 1999), so that the discrete second derivative at p_i on any direction $\vec{u}$ can be defined as:

$$L_{\vec{u}}(p_i) = \frac{1}{2}(p_{i+1}(\vec{u}) - p_i) - \frac{1}{2}(p_i - p_{i-1}(\vec{u})), \tag{2}$$

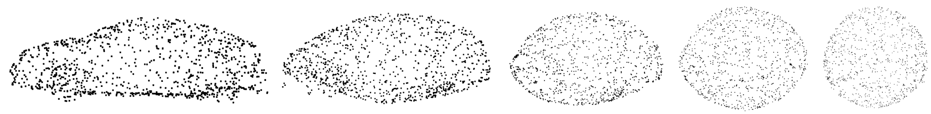
supposing p_{i-1} and p_{i+1} are the points right before and after p_i along the direction $\vec{u}$. Usually $\vec{u}$ is chosen along the line from a mesh vertex to one of its neighbors. The umbrella operator sums up the second derivatives in all different directions. Each vertex is then updated using the following scheme,

$$p_i' = p_i + \lambda L(p_i) \tag{3}$$

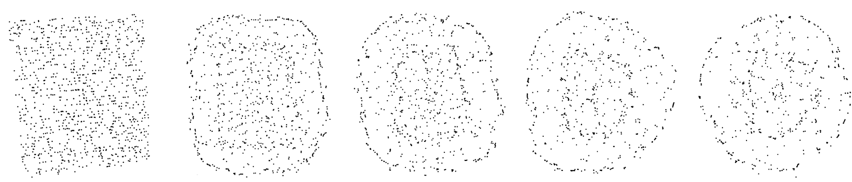
$$p_i'' = p_i' - \mu L(p_i') \tag{4}$$

where $0 < \lambda < 1$ and $\lambda < \mu$. (Taubin, 1995) proves that this iterative algorithm serves as a low-pass filter and is anti-shrinkage. The intuition is that Eq. (3) attenuates the high frequencies and Eq. (4) magnifies the remaining low frequencies, thus preventing shrinkage. However, as (Desbrun et al., 1999) pointed out, this diffusion method is flawed due to its unrealistic assumption about meshes.

3.1.2 Our algorithm



(a) Applying 3-D version of our algorithm to a car shape.



(b) Applying 2-D version of our algorithm together with the 3-D version to a curtain shape.

Figure 1: Demonstrations of the smoothing algorithm on two shapes from ModelNet40

Suppose the underlying shape of the point cloud is a closed 2-manifold, in order to accommodate unevenly distributed points in point cloud data, we use a curvature-flow-based method, inspired by (Desbrun et al., 1999) and (Alexa et al., 2001). We first fit a local plane $H = \{x : \langle x, \boldsymbol{n}\rangle + D = 0, x \in \mathbb{R}^3\}, \boldsymbol{n} \in \mathbb{R}^3, ||\boldsymbol{n}|| = 1$ for each point p_i by minimizing the least-squares error:

$$\arg\min_{\boldsymbol{n},D} \sum_{j \in \mathcal{N}(p_i)} \left(\langle p_j, \boldsymbol{n}\rangle + D\right)^2 \tag{5}$$

Under a special case we prove in Appendix A that the distance between p_i and H can represent the local curvature. More generally, this distance is an approximation of the local curvature that can be computed efficiently. It is also possible to fit a quadratic surface so that the curvature can be computed analytically, however such a fit would be both slower to compute and more prone to overfitting, as we will show in the experiments.

Let h_i denote the position of p_i after being projected onto H (i.e. $h_i = p_i - (\langle p_i, \boldsymbol{n}\rangle + D) \cdot \boldsymbol{n}$). Then $h_i - p_i$ is the vector pointing from the point p_i to the plane H. Now we can accommodate (Taubin, 1995)'s smoothing algorithm to point cloud data as follows:

$$p_i' = p_i + \lambda \left(h_i - p_i\right) \tag{6}$$

$$p_i'' = p_i' - \mu \left(h_i' - p_i'\right) \tag{7}$$

where $0 < \lambda < 1$, $\lambda < \mu$ and h_i' refers to the projection of p_i' on a new plane H' fitted for p_i'. Thus instead of moving the point toward the mean of its neighbors, we move it directly toward the locally fitted plane, which can be seen approximately as moving the point based on the local mean curvature, the approach championed by Desbrun et al. (1999). We call Eq. 6 the "erosion" round, and Eq. 7 the "dilation" round. Our algorithm has the same nice property as the one proposed by (Desbrun et al., 1999), which is that the vertices in an already flat shape (e.g., a curtain) will not be shifted by our algorithm, since p_i will be equal to h_i.

An important novel implementation detail is that the size of the neighborhood we use increases as the smoothing goes further. In practice, after every 4 rounds of erosion and dilation, we expand the neighborhood size by 20 points. The reason for this is twofold. On one hand, there might exist isolated neighborhoods in a point cloud (i.e. a set of points that is closed under the $\mathcal{N}(\cdot)$ operation). If the curvature information cannot be propagated to the entire point cloud, the final result will not be smooth. On the other hand, a larger neighborhood speeds up the smoothing process. As mentioned in (Kobbelt et al., 1998), the time step restriction $(0 < \lambda < 1)$ results in the need of hundreds of updates to cause a noticeable smoothing using the original implementation in (Taubin, 1995).

To deal with degenerate cases where the point cloud is already on a plane, we further extend the algorithm to a 2D case(Fig. 1b). Here the aim is to make the boundary smooth, transforming the plane to a disk. In this case, assuming all the neighborhood points $\mathcal{N}(p_i)$ are on the plane, we fit a line $H' = \{x : \langle x, \boldsymbol{n'}\rangle + C = 0, x \in \mathbb{R}^2\}, \boldsymbol{n'} \in \mathbb{R}^2, ||\boldsymbol{n'}|| = 1$ for $w_i = (0, 0)$ by minimizing the least-squares error:

$$\arg\min_{\boldsymbol{n'},C} \sum_{j\in\mathcal{N}(p_i)} \left(\langle w_j, \boldsymbol{n'}\rangle + C\right)^2 \tag{8}$$

Let q_i be w_i's projection on line H'. We update w_i in the same fashion as in the 3D case:

$$w_i' = w_i + \lambda\,(q_i - w_i) \tag{9}$$

$$w_i'' = w_i' - \mu\,(q_i' - w_i') \tag{10}$$

Finally, we convert $w_i = (u_i, v_i)$ back to 3-D by calculating $p_i' = p_i + u_i\vec{u} + v_i\vec{v}$. In reality, due to noises, many points are not exactly on a plane. We project them to their local planes H first, and then calculate the uv-coordinates from their projected location h_i. Note that we still shift the point from its original location p_i, not its projected location h_i. In actual implementation, the 2-D version is used together with the 3-D version and is always run first. For example, in an "erosion" round, we run Eq. (9) first, then Eq. (6); in a "dilation" round, we run Eq. (10) first, then Eq. (7). Empirically this seems to generalize well on both planar and non-planar surfaces and avoids introducing extra parameters to make a decision whether a neighborhood is on a plane.

3.2 Integrated-Gradients Optimized Saliency (I-GOS)

We summarize the I-GOS algorithm (Qi et al., 2019) which is a recent algorithm for visualizing deep networks. The goal in I-GOS is to optimize for a small and smooth mask so that when an image is masked, the prediction from the deep network drops significantly. I-GOS improves from conventional gradient descent approaches in that the optimization is solved with a mixture of conventional gradients and *integrated gradients*, where the integrated gradients point to a global optimum for the unconstrained problem of only minimizing the prediction on the image, so that the optimization can evade local optima and achieve better performance.

We seek to adapt this algorithm to point clouds. Formally, let mask $\mathcal{M}$ be of the same size as the point cloud $\mathcal{P}$, and initialized with all zeros (transparent). Let $\mathcal{P}_0$ be the fully smoothed point cloud (e.g. sphere) and let $\mathcal{M}_0$ be the baseline mask which is all ones, so that when applied to the shape, the shape becomes $\mathcal{P}_0$. Mask values are always between $[0, 1]$, where 0 means no smoothing, 1 means fully smoothing. We optimize the mask by minimizing the classification score on the masked point cloud, along with 2 regularizers:

$$L_{overall} = \lambda_{cls}L_{cls} + \lambda_{l1}L_{l1} + \lambda_{tv}L_{tv} \tag{11}$$

where L_{cls} is defined as the *integrated* classification score loss along the straight path from $\mathcal{M}_0$ to $\mathcal{M}$:

$$L_{cls} = \int_{\alpha=0}^{1} f_c(\Phi(\mathcal{P}, \mathcal{M} + \alpha(\mathcal{M}_0 - \mathcal{M})))d\alpha \tag{12}$$

where $f_c(\cdot)$ represents the classifier on the class c to be visualized (usually the class with the highest predicted confidence) and Φ represents the action of applying the mask to the point cloud. where $L_{l1} = \dfrac{1}{N}||1 - \mathcal{M}||_1$ and $L_{tv} = \left(\dfrac{1}{N}\sum_{\mathcal{M}} \dfrac{1}{K}\sum_{j\in\mathcal{N}(p_i)} |m_j - m_i|\right)^{\beta}$. the total variation (TV) regularization with β being a parameter, usually either 1 or 2. The L_1 and TV regularizations are to make the mask small and smooth, hence making the resulting point cloud more likely to stay in the same distribution as the training and less likely to be adversarial.

Note that the gradients of L_{cls} w.r.t $\mathcal{M}$ are exactly the *integrated gradients* proposed by (Sundararajan et al., 2017). In actual implementation, this integration is approximated using summation:

$$\widehat{L_{cls}} = \frac{1}{S}\sum_{s=1} f(\Phi(\mathcal{P}, \mathcal{M} + \frac{s}{S}(\mathcal{M}_0 - \mathcal{M}))) \tag{13}$$

where S is the number of intervals used.

One difficulty in extending this algorithm to 3D point clouds is to implement $\Phi(\cdot)$ as a smooth operation so that gradients can be taken w.r.t it. In 2-D images, we can simply use a weighted (by m_i) average the value of a pixel with the baseline pixel value in the baseline image. However, in point clouds, the iterative smoothing algorithm we proposed in Sec. 3.1.2 is not smooth.

In practice, we make Φ differentiable by precomputing 10 intermediate shapes with increasing level of smoothness. In practice, we approximate this process by saving 10 intermediate shapes with increasing level of smoothness. Then, a point p_i' with a mask value $m_i \in [0, 1]$ applied on it can be represented as:

$$p_i' = \frac{\sum_{l=0}^{10} \exp(-\alpha\|10 \cdot m_i - l\|^2) p_{i,l}}{\sum_{l=0}^{10} \exp(-\alpha\|10 \cdot m_i - l\|^2)} \tag{14}$$

where l refers to the l-th point cloud in our sequence of smoothed shapes ($l = 0$ refers to $\mathcal{P}_0$ and $l = 10$ refers to the original shape), $\Psi(\cdot)$ is a similarity function (a Gaussian kernel, in our case), $p_{i,l}$ refers to the position of the i-th point in the l-th point cloud.

4 EXPERIMENT RESULTS

We have conducted two types of experiments. First, since we are proposing a new smoothing algorithm, we compare against a number of baselines on the smoothing capability of those algorithms. In the second part, we utilize our extended I-GOS algorithm to visualize point cloud networks and compare with some baselines as well as performing some ablation studies on the visualization. All experiments are conducted on the test split of the ModelNet40 dataset, with the classifiers to be visualized trained on the training split. 1024 points are randomly sampled from each shape, and only xyz location information is used in all experiments. All the parameters are fixed through the entire dataset. $\lambda = 0.7, \mu = 1.0$, K grows from 20 to 60. We usually run the algorithm for 80 iterations (each iteration contains one "erosion" step and one "dilation" step).

4.1 POINT CLOUD SMOOTHING

Since there were few prior work that directly smooth point clouds, we compare against several other plausible baselines as well. We first note that directly applying Gaussian blur to the coordinates is not a valid baseline in point clouds, because Gaussian blur tends to smooth the coordinate values, they tend to push neighborhood points to all have the same coordinates, leading to a skeleton effect which is completely contrary to our goals. We mainly compare against 3 baselines:

Meshing, then smoothing One natural idea is to convert the point cloud to a mesh and then apply mesh-based smoothing techniques such as (Desbrun et al., 1999) to the result. For our goals, we need to choose an algorithm that does not change the number of points and maintain a 1-1 correspondence with the original point cloud. We utilized a greedy projection triangulation algorithm (Marton et al., 2009), but due to the noisiness and sparsity of the point cloud, the meshing result is often not ideal, as well as the smoothing results (e.g. Fig. 3).

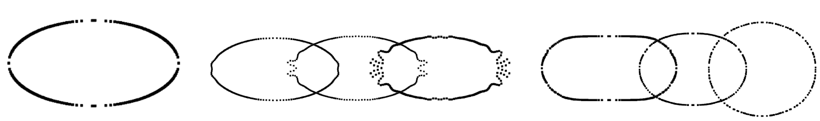


Figure 2: A comparison between Taubin smoothing and our smoothing on 2-D point cloud. Left: A 2-D ellipse point cloud with 202 unevenly distributed points. Middle: Taubin smoothing. Right: Our smoothing. In the case of Taubin smoothing, highly concentrated areas are pushing points outward, resulting in an undesired shape, while our algorithm is not influenced by point density.

Directly applying mesh smoothing techniques to points. Instead of explicit meshing, we can use neighborhood function $\mathcal{N}(\cdot)$ to construct an *implicit mesh*, i.e., assuming a point has an edge to each of the points in its neighborhood. Using this implicit mesh, mesh smoothing techniques can be directly to point clouds. However, the uneven distribution of points in a point cloud quite often distorts the result, in a method such as Taubin smoothing (Fig. 2). Though (Fujiwara, 1995) and (Desbrun et al., 1999) have proposed improvements for irregular meshes, they explicitly exploit edge information, which is not available in point cloud data (and requires explicit meshing as above).

Fitting a quadratic surface. Another natural idea is to directly fit a quadratic surface to the local neighborhood instead of a plane as in our approach. A quadratic surface allows analytic computation of the curvature hence then we can simply run algorithms based on mean curvature flow. We implemented the closed-form quadratic fitting algorithm following (Groshong et al., 1989). However, as pointed out by (Andrews & Séquin, 2014), since quadratic surfaces have a large degree of freedom compared to planes (10 parameters compared to 4), even a small bit of noise will render an undesired quadratic type or direction. As illustrated in Fig. 4, the border of the car shape ends up consisting of quadratic lines curving outward instead of inward.

Figure 3: Meshing a point cloud and then applying Laplacian smoothing as in (Taubin, 1995). Corresponding point clouds attached above.

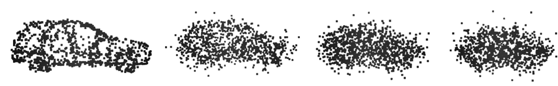


Figure 4: Fitting quadratic surfaces to local neighborhoods and running mean-curvature-flow algorithm using the mean curvature calculated using the surfaces.

For a quantitative comparison against these baselines, we propose two metrics to evaluate our smoothing algorithm based on the goals of equalizing the mean curvature of the surface. Assuming that the structure is not degenerate, the smoothing should eventually make the point cloud to be a sphere. Hence, we can evaluate the *min-max ratio* (MR), which is the ratio between the length on the long side and the short side of the point cloud. This is computed by first applying principal component analysis (PCA) to the point cloud and finding the top two principal components, say $\vec{u}$ and $\vec{v}$. Then the ratio between ranges of the values are computed on these two principal directions: $\frac{\min_{\vec{t}}(\max_i(p_{i,\vec{t}}) - \min_i(p_{i,\vec{t}}))}{\max_{\vec{t}}(\max_i(p_{i,\vec{t}}) - \min_i(p_{i,\vec{t}}))}$ where $\vec{t} \in \{\vec{u}, \vec{v}\}$ and $p_{i,\vec{t}}$ denotes the i-th point's component on $\vec{t}$. The closer this ratio is to 1, the better.

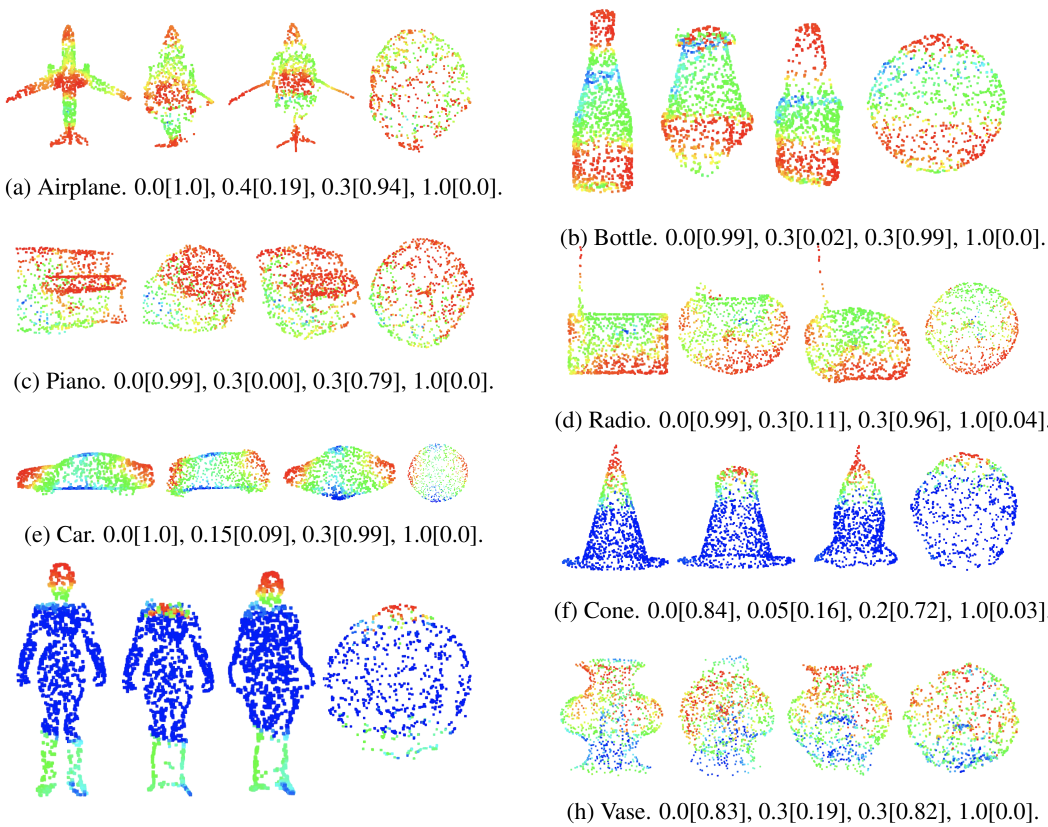
As another metric, we propose to evaluate *distance distribution similarity* (DDS) between one point cloud and its smoothed version after one iteration. This is computed by first computing the Kolmogorov-Smirnov statistic $\sup_x |D_l(x) - D_{l-1}(x)|$ where $D(x)$ denotes the empirical distribution function of the distances, and then calculating the p-value of the statistic. The larger this p-value, the more similar the distributions are. In practice, ten intermediate point clouds with increasing level of blurriness are sampled. The metrics are calculated for all of them and the results are listed in Table 1. All the algorithms are evaluated on the entire ModelNet40 testing split, and the average of all the shapes is taken. It can be seen that both implicit and explicit meshing approaches are quite unstable by having extremely low DSS for some l. Also explicit meshing does not seem to improve MR at all. The quadratic surface fitting approach morphs the shapes as smoothly as our algorithm, but fails to morph the shape into a sphere at the very end.

4.2 Classifier visualization

We experiment our adapted I-GOS algorithm on PointConv (Wu et al., 2019) and DGCNN (Wang et al., 2018), two state-of-the-art point cloud classifiers. Both networks have classification accuracy above 92% on the ModelNet 40 test set. Fig. 5 shows some example masks generated by our algorithm for PointConv and DGCNN. These pictures reveal to us some interesting insight into the patterns used in the classifiers: for airplanes, the wings and the tails are crucial; for radios, the existence of the antenna is critical; for cars, the front and trunk arer important; for vases, the curvature at the neck is more important than the curvature at the bottom.

Table 1: Comparison of Point cloud smoothing algorithms. Mesh refers to meshing and smoothing, Taubin refers to directly applying Taubin smoothing to point clouds. DDS below 0.05 are italicized, indicating extreme unsmoothness. It can be seen both Taubin and Ours converged well, but Taubin is very unsmooth in the middle. Quadratic is very smooth but does not converge in the end

Algorithm \level l		1	2	3	4	5	6	7	8	9	10
Mesh	MR	0.84	0.86	0.86	0.85	0.84	0.83	0.83	0.82	0.82	0.82
	DDS	0.12	0.12	0.05	*0.02*	*0.04*	0.28	0.57	0.63	0.63	0.63
Taubin	MR	0.84	0.82	0.82	0.78	0.84	0.89	0.92	0.94	0.95	**0.95**
	DDS	0.28	0.08	*0.03*	*0.00*	*0.00*	*0.00*	*0.00*	*0.02*	0.24	0.60
Quadratic	MR	0.80	0.79	0.80	0.80	0.80	0.80	0.81	0.81	0.81	0.81
	DDS	0.38	0.64	0.77	0.86	0.91	0.94	0.96	0.98	0.97	0.97
Ours	MR	0.85	0.87	0.88	0.89	0.91	0.92	0.94	0.94	0.95	**0.95**
	DDS	0.09	0.29	0.10	0.26	0.11	0.20	0.09	0.13	0.05	0.07



(a) Airplane. 0.0[1.0], 0.4[0.19], 0.3[0.94], 1.0[0.0].

(b) Bottle. 0.0[0.99], 0.3[0.02], 0.3[0.99], 1.0[0.0].

(c) Piano. 0.0[0.99], 0.3[0.00], 0.3[0.79], 1.0[0.0].

(d) Radio. 0.0[0.99], 0.3[0.11], 0.3[0.96], 1.0[0.04].

(e) Car. 0.0[1.0], 0.15[0.09], 0.3[0.99], 1.0[0.0].

(f) Cone. 0.0[0.84], 0.05[0.16], 0.2[0.72], 1.0[0.03].

(g) Person. 0[0.79], 0.1[0.15], 0.15[0.63], 1[0.05].

(h) Vase. 0.0[0.83], 0.3[0.19], 0.3[0.82], 1.0[0.0].

Figure 5: Example masks (best viewed in color). First four for PointConv, last four for DGCNN. Red indicates high mask value, blue low. Within each group of pictures from left to right: original shape, least amount of points smoothed to drop the prediction confidence below 0.2×original confidence, least points inserted for rising the prediction above 0.8×original prediction confidence, 100% blurred. All the numbers below the pictures are of the format: percentage blurred [prediction confidence].

We use the *deletion* and *insertion* metrics proposed by (Petsiuk et al., 2018) to evaluate the masks. For *deletion*, we gradually smooth the shape based on the mask. We then plot the curve of network prediction confidences on the different shapes and calculate area under the curve. *insertion* scores are also area under the curve, but retain points deemed as more important unsmoothed, and smooth the mosts unimportant points instead. We want the *deletion* score to be low, indicating that smoothing a small area would distract the classifier, and the *insertion* score curve to be high, indicating that the classifier can predict from a small amount of features. Experiment results averaged over all 40 classes are shown in the last row of Table 3. Individual class results are attached in the Appendix. Red are the *deletion* curves, blue (reading from right to left) are the *insertion* curves.

Table 2 shows the comparison between the two baseline methods and I-GOS: mask-only (Fong & Vedaldi, 2017) and ig-only (Sundararajan et al., 2017). mask-only learns the mask using gradients instead of integrated gradients. Each mask goes through 300 iterations under this method compared to 30 under I-GOS. ig-only directly takes the integrated gradient instead of an optimization process. From the table we can see that I-GOS performs much better than the baselines. Table 3 shows the ablation study for $l1$-loss and tv-loss (tv stands for total-variation). As we can see, both losses are useful for maximizing the performance of the algorithm.

Table 2: Baseline methods for obtaining saliency mask compared to I-GOS using the *deletion* and *insertion* metrics (averaged over 40 classes), conducted with the PointConv classifier

	deletion	*insertion*	difference
mask-only	**0.2318**	0.2474	0.0156
ig-only	0.3751	0.3099	-0.0653
I-GOS	0.2684	**0.4113**	**0.1429**

Table 3: Results on PointConv and DGCNN averaged over 40 classes, as well as ablation study for $l1$-loss and tv-loss using *deletion* and *insertion* metrics. As shown, both losses are necessary for maximizing the performance of the algorithm

	PointConv			DGCNN		
	deletion	*insertion*	difference	*deletion*	*insertion*	difference
no $l1$, no tv	0.2833	0.3889	0.1056	0.1825	0.2212	0.0387
with $l1$, no tv	0.2710	0.3989	0.1279	0.1659	0.2264	0.0605
no $l1$, with tv	**0.2677**	0.4097	0.1420	**0.1563**	0.2234	0.0670
with $l1$, with tv	0.2684	**0.4113**	**0.1429**	0.1594	**0.2315**	**0.0720**

5 Conclusions and Future Work

In this paper, we proposed a classifier visualization approach by extending the I-GOS algorithm that visualizes 2D images. In order to smooth the point clouds without abrupt changes, we proposed a novel smoothing approach that gradually smooths the point clouds and eventually converge to a shape with constant mean curvature. Experiment results show that our algorithm outperforms baselines on both point cloud smoothing and classifier visualization. As compared with 2D results in Qi et al. (2019), the 3D shapes consistently show higher deletion metric and lower insertion metrics, indicating that point cloud networks use more parts than 2D image CNNs to classify. We hope those visualization results improve our understanding on these new networks.

References

Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T Silva. Point set surfaces. In *Proceedings of the Conference on Visualization'01*, pp. 21–28. IEEE Computer Society, 2001.

James Andrews and Carlo H Séquin. Type-constrained direct fitting of quadric surfaces. *Computer-Aided Design and Applications*, 11(1):107–119, 2014.

Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *arXiv preprint arXiv:1803.10091*, 2018.

Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

Stéphane Calderon and Tamy Boubekeur. Point morphology. *ACM Trans. Graph.*, 33(4):45:1–45:13, July 2014. ISSN 0730-0301. doi: 10.1145/2601097.2601130.

Piotr Dabkowski and Yarin Gal. Real time image saliency for black box classifiers. In *Advances in Neural Information Processing Systems*, pp. 6967–6976, 2017.

Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 317–324. Citeseer, 1999.

Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 869–877, 2018.

Ruth C. Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

Koji Fujiwara. Eigenvalues of laplacians on a closed riemannian manifold and its nets. *Proceedings of the American Mathematical Society*, 123(8):2585–2594, 1995.

Bennett Groshong, Griff Bilbro, and Wesley Snyder. Fitting a quadratic surface to three dimensional data. 1989.

Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *Siggraph*, volume 98, pp. 105–114, 1998.

David Levin. The approximation power of moving least-squares. *Mathematics of computation*, 67 (224):1517–1531, 1998.

Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. In *Advances in Neural Information Processing Systems*, pp. 820–830, 2018.

Jyh-Ming Lien. Point-based minkowski sum boundary. In *15th Pacific Conference on Computer Graphics and Applications (PG'07)*, pp. 261–270. IEEE, 2007.

Daniel Liu, Ronald Yu, and Hao Su. Extending adversarial attacks and defenses to deep 3d point cloud classifiers. *arXiv preprint arXiv:1901.03006*, 2019.

Zoltan Csaba Marton, Radu Bogdan Rusu, and Michael Beetz. On Fast Surface Reconstruction Methods for Large and Noisy Datasets. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Kobe, Japan, May 12-17 2009.

Boris Mederos, Luiz Velho, and Luiz Henrique de Figueiredo. Robust smoothing of noisy point clouds. In *Proc. SIAM Conference on Geometric Design and Computing*, volume 2004, pp. 2, 2003.

Vitali Petsiuk, Abir Das, and Kate Saenko. Rise: Randomized input sampling for explanation of black-box models. *arXiv preprint arXiv:1806.07421*, 2018.

Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems 30*, pp. 5099–5108. Curran Associates, Inc., 2017.

Zhongang Qi, Saeed Khorram, and Fuxin Li. Visualizing deep networks by optimizing with integrated gradients. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.

Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje. Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*, 2016.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. Splatnet: Sparse lattice networks for point cloud processing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2530–2539, 2018.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 3319–3328. JMLR.org, 2017.

Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. Tangent convolutions for dense prediction in 3d. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3887–3896, 2018.

Gabriel Taubin. A signal processing approach to fair surface design. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pp. 351–358, New York, NY, USA, 1995. ACM. ISBN 0-89791-701-4.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018.

Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Chong Xiang, Charles R Qi, and Bo Li. Generating 3d adversarial point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9136–9144, 2019.

Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 87–102, 2018.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.

Tianhang Zheng, Changyou Chen, Junsong Yuan, Bo Li, and Kui Ren. Pointcloud saliency maps, 2018.

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*, 2014.

A CURVATURE APPROXIMATION PROOF

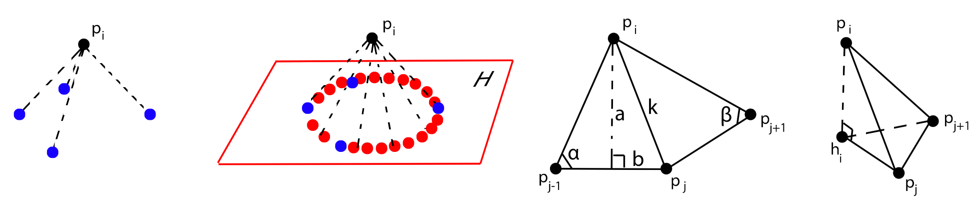


Figure 6: Auxillary graph for proof in Appendix A. From left to right: point p_i and its actual neighbors (in blue), p_i and its virtual neighbors (in red) and the fitted local plane H, enlarged graph of p_i and three of its neighbors, p_i with h_i, which is p_i's projection onto the fitted plane. h_i is also the center of the ring formed by the virtual neighbors.

Our proof will refer to Fig. 6. (Desbrun et al., 1999) has already showed that on a 3-D mesh, given a point p_i and its neighbors, the local "carvature normal" can be calculated using

$$\frac{1}{4A} \sum_{j \in \mathcal{N}(p_i)} (\cot \alpha_j + \cot \beta_j)(p_j - p_i) \tag{15}$$

where A is the sum of the areas of the triangles having p_i as common vertex and α_j, β_j are the two angles opposite to the edge e_{ij} (i.e. $p_j - p_i$). This arrangement is demonstrated Fig. 6.

Since point cloud data are usually sparse and noisy, we want to utilize some mechanism to mitigate this sparsity and irregularity. Here, we first fit a local plane to p_i's neighborhood, and then we define the notion of "virtual neighbors" as a means to fill in the gaps left by the "actual neighbors". We assume the "virtual neighbors" distribute evenly and densely on a ring surrounding p_i on the fitted plane H, each having the same distance k to p_i (k is calculated using the average distance of the actual neighbors). Let p_i's projection on H be h_i, which is at the center of the ring formed by the "virtual neighbors". Let a be the distance from p_i to each edge $e_{j,j+1}$. Let b be half of the length of $e_{j,j+1}$. Thus we can calculate A in Eq. 15 as $n \cdot ab$. Since we assumed the points are distributed evenly, we have $\cot\alpha = \cot\beta = \frac{b}{a}$. Thus we have the curvature normal to be $\frac{1}{4A}\sum_j(\cot\alpha_j + \cot\beta_j)(p_j - p_i) = \frac{1}{4nab}\cdot\frac{2b}{a}\sum_j(p_j - p_i) = \frac{1}{2na^2}\sum_j(p_j - p_i)$.

Note that the vector $p_j - p_i$ is equal to $(p_i - h_i) + (h_i - p_i)$, and it can be easily shown that $\sum_j(p_i - h_i) = \vec{0}$. Thus we can continue derive the curvature normal to be $\frac{1}{2na^2}\sum_j(p_j - p_i) = \frac{1}{2na^2}\sum_j(h_i - p_i) = \frac{n}{2na^2}(h_i - p_i) = \frac{1}{2a^2}(h_i - p_i)$. Since we assume the points are distributed densely, thus we have as $n \to \infty$, $a \to k$. Hence, the curvature normal at p_i can be approximated by the expression

$$\frac{1}{2k^2}(h_i - p_i) \tag{16}$$

where $h_i - p_i$ is just the vector pointing from p_i to the local plane H as in Eq. 6 and 7. This equation makes sense in that when the distance from p_i to H is fixed, the further away the neighbors are, the "flatter" the surface at p_i is.

In our actual experimentation however, we found that due to the extremely irregular distribution of the point cloud data, the neighborhood distance is actually misleading sometimes rather than helpful. Thus, in our final algorithm, we abandon the distance information $\frac{1}{2k^2}$ and directly use the vector pointing from p_i to plane H as our approximation for the local curvature.

B CURVE FIGURES

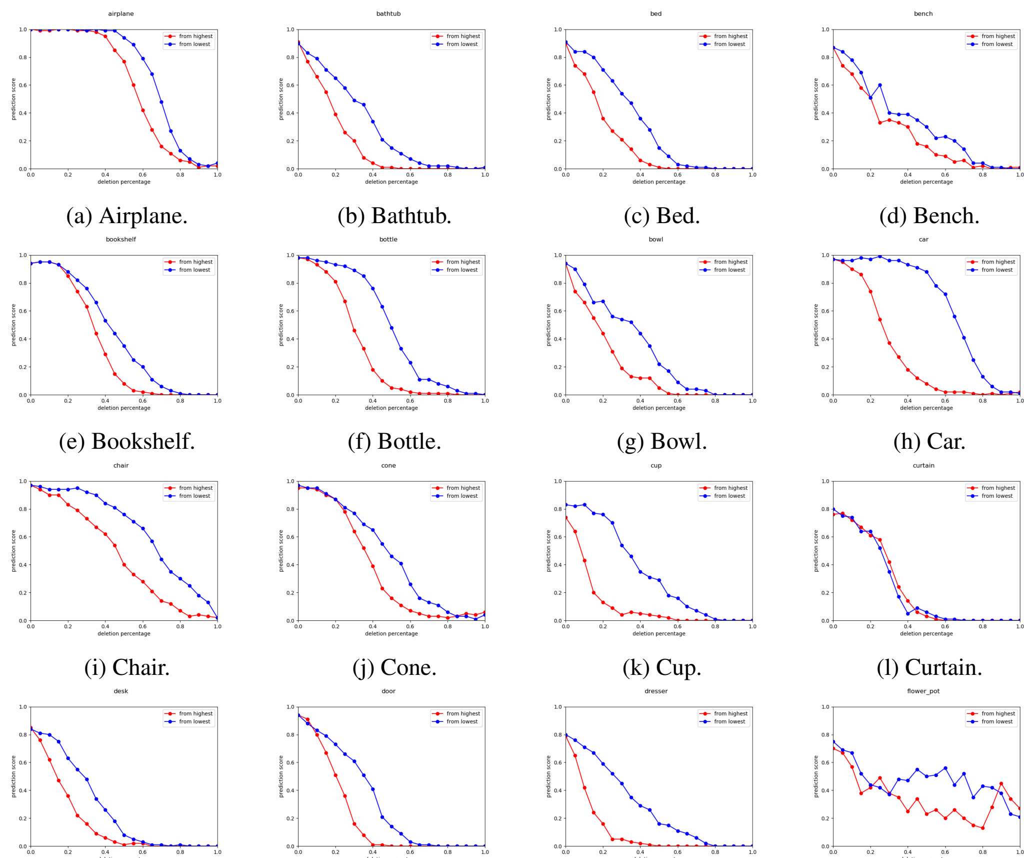
Table 4: *Deletion* score curve average and *insertion* score curve average for PointConv.

	airplane	bathtub	bed	bench	bookshelf	bottle	bowl	car	chair	cone
del.	0.5834	0.1859	0.1886	0.2557	0.3345	0.3084	0.2029	0.2917	0.4551	0.3720
ins.	0.6802	0.3052	0.3195	0.3343	0.4224	0.4907	0.3307	0.6385	0.6452	0.4672
	cup	curtain	desk	door	dresser	flowerpot	glassbox	guitar	keyboard	lamp
del.	0.1178	0.2386	0.1748	0.2112	0.1151	0.3486	0.0839	0.2331	0.2482	0.4263
ins.	0.3425	0.2315	0.2779	0.3261	0.2846	0.4730	0.1934	0.4470	0.3048	0.6227
	laptop	mantel	monitor	nightstand	person	piano	plant	radio	rangehood	sink
del.	0.2182	0.2283	0.2620	0.1429	0.1871	0.2872	0.7666	0.2601	0.2474	0.3175
ins.	0.3055	0.3728	0.4304	0.3545	0.2867	0.3822	0.8337	0.4626	0.3406	0.4408
	sofa	stairs	stool	table	tent	toilet	tv stand	vase	wardrobe	xbox
del.	0.2742	0.2521	0.1727	0.4009	0.3107	0.1933	0.1602	0.4210	0.0628	0.1446
ins.	0.3611	0.3779	0.3350	0.4656	0.7125	0.4644	0.2864	0.6709	0.1046	0.1644

Table 5: *Deletion* score curve average and *insertion* score curve average for DGCNN.

	airplane	bathtub	bed	bench	bookshelf	bottle	bowl	car	chair	cone
del.	0.3683	0.0683	0.1456	0.1473	0.2328	0.1501	0.1439	0.2661	0.3206	0.1905
ins.	0.4906	0.1158	0.1900	0.1980	0.2849	0.2972	0.1933	0.3740	0.4141	0.3338
	cup	curtain	desk	door	dresser	flowerpot	glassbox	guitar	keyboard	lamp
del.	0.0630	0.0714	0.1285	0.0674	0.0702	0.1082	0.0628	0.2503	0.1685	0.2269
ins.	0.0767	0.1511	0.1788	0.1413	0.0880	0.0983	0.0812	0.3875	0.2011	0.3464
	laptop	mantel	monitor	nightstand	person	piano	plant	radio	rangehood	sink
del.	0.0534	0.0782	0.2127	0.0860	0.1091	0.1554	0.6798	0.2013	0.1018	0.1196
ins.	0.0675	0.1077	0.2972	0.1262	0.4150	0.2031	0.7188	0.2466	0.1560	0.2267
	sofa	stairs	stool	table	tent	toilet	tv stand	vase	wardrobe	xbox
del.	0.1840	0.1677	0.1366	0.1761	0.1918	0.1362	0.0924	0.1706	0.0463	0.0584
ins.	0.2306	0.2285	0.1920	0.1932	0.2626	0.2498	0.1086	0.3781	0.0654	0.0745

C CURVE FIGURES



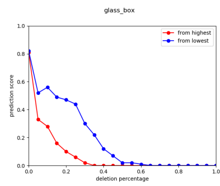
(a) Airplane. (b) Bathtub. (c) Bed. (d) Bench.

(e) Bookshelf. (f) Bottle. (g) Bowl. (h) Car.

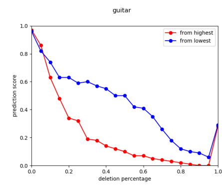
(i) Chair. (j) Cone. (k) Cup. (l) Curtain.

(m) Desk. (n) Door. (o) Dresser. (p) Flower pot.

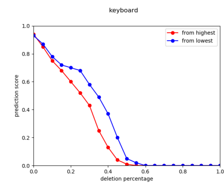
Figure 7: *Deletion* and *insertion* curves for all 40 classes in ModelNet40 for PointConv. Horizontal axis is the deletion percentage (top 5%, 10%, etc.), and vertical axis is the predicted class score. The red line is the *deletion* curve which blurs points from highest mask values, and the blue line is the *insertion* curve (if read from right to left) which blurs points from lowest mask values.



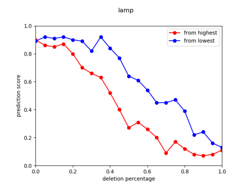
(q) Glass box.



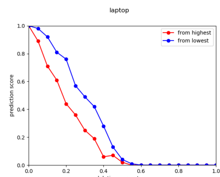
(r) Guitar.



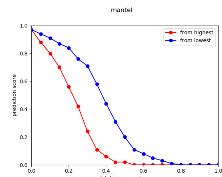
(s) Keyboard.



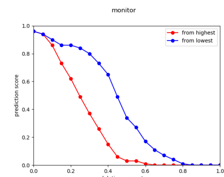
(t) Lamp.



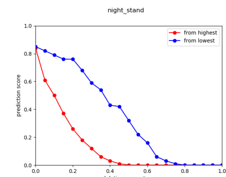
(u) Laptop.



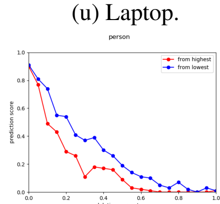
(v) Mantel.



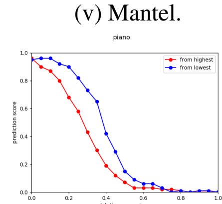
(w) Monitor.



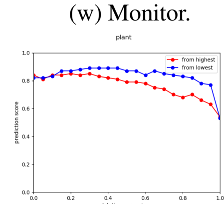
(x) Night stand.



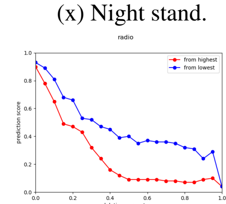
(y) Person.



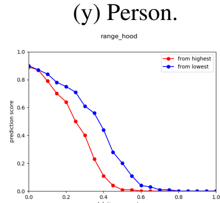
(z) Piano.



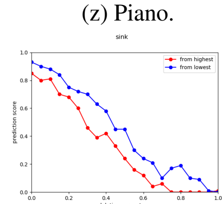
(aa) Plant.



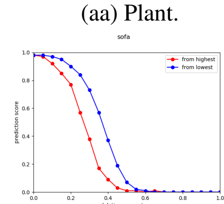
(ab) Radio.



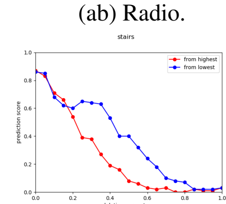
(ac) Range hood.



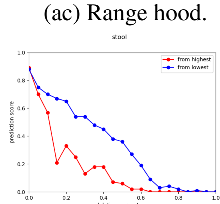
(ad) Sink.



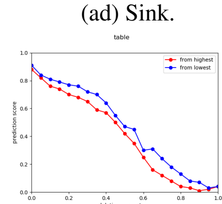
(ae) Sofa.



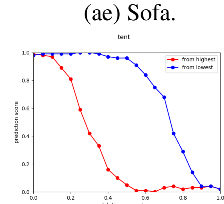
(af) Stairs.



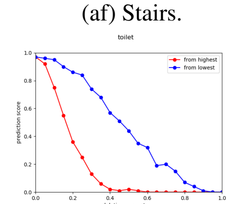
(ag) Stool.



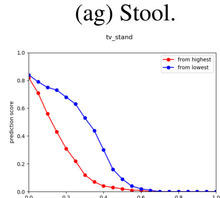
(ah) Table.



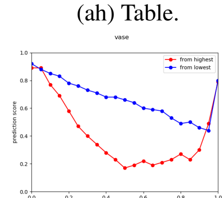
(ai) Tent.



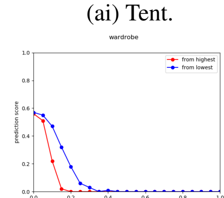
(aj) Toilet.



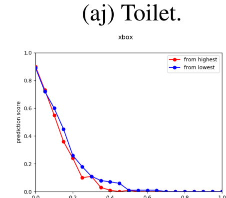
(ak) TV stand.



(al) Vases.

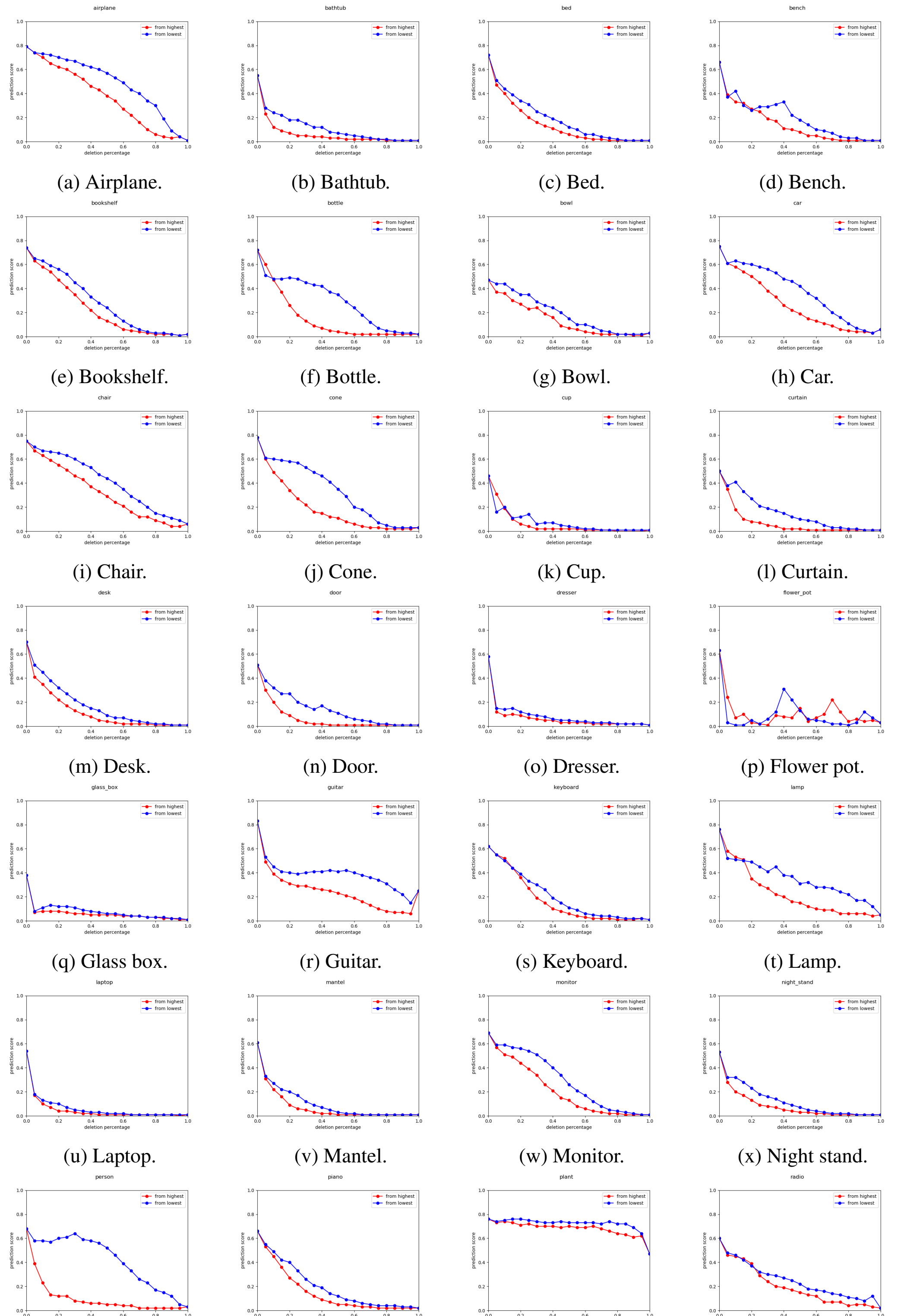


(am) Wardrobe.



(an) Xbox.

Figure 7: *Deletion* and *insertion* curves for all 40 classes in ModelNet40 for PointConv. Horizontal axis is the deletion percentage (top 5%, 10%, etc.), and vertical axis is the predicted class score. The red line is the *deletion* curve which blurs points from highest mask values, and the blue line is the *insertion* curve (if read from right to left) which blurs points from lowest mask values. (cont.)



(a) Airplane. (b) Bathtub. (c) Bed. (d) Bench.

(e) Bookshelf. (f) Bottle. (g) Bowl. (h) Car.

(i) Chair. (j) Cone. (k) Cup. (l) Curtain.

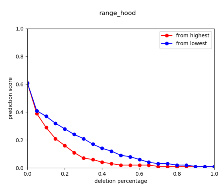
(m) Desk. (n) Door. (o) Dresser. (p) Flower pot.

(q) Glass box. (r) Guitar. (s) Keyboard. (t) Lamp.

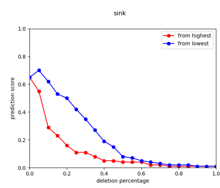
(u) Laptop. (v) Mantel. (w) Monitor. (x) Night stand.

(y) Person. (z) Piano. (aa) Plant. (ab) Radio.

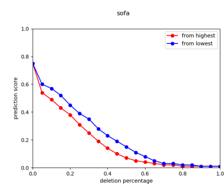
Figure 8: *Deletion* and *insertion* curves for all 40 classes in ModelNet40 for DGCNN. Horizontal axis is the deletion percentage (top 5%, 10%, etc.), and vertical axis is the predicted class score. The red line is the *deletion* curve which blurs points from highest mask values, and the blue line is the *insertion* curve (if read from right to left) which blurs points from lowest mask values.



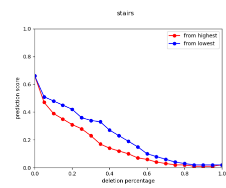
(ac) Range hood.



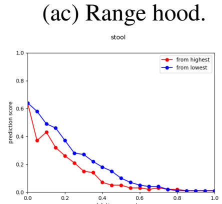
(ad) Sink.



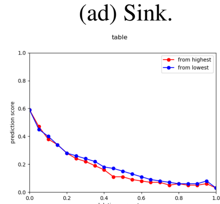
(ae) Sofa.



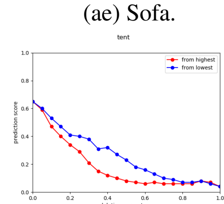
(af) Stairs.



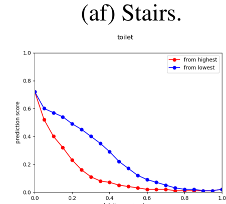
(ag) Stool.



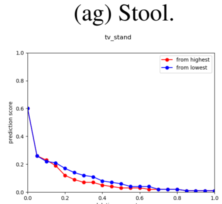
(ah) Table.



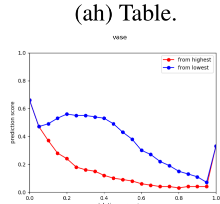
(ai) Tent.



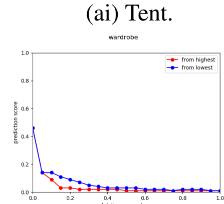
(aj) Toilet.



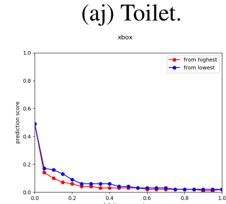
(ak) TV stand.



(al) Vases.



(am) Wardrobe.



(an) Xbox.

Figure 8: *Deletion* and *insertion* curves for all 40 classes in ModelNet40 for DGCNN. Horizontal axis is the deletion percentage (top 5%, 10%, etc.), and vertical axis is the predicted class score. The red line is the *deletion* curve which blurs points from highest mask values, and the blue line is the *insertion* curve (if read from right to left) which blurs points from lowest mask values. (cont.)