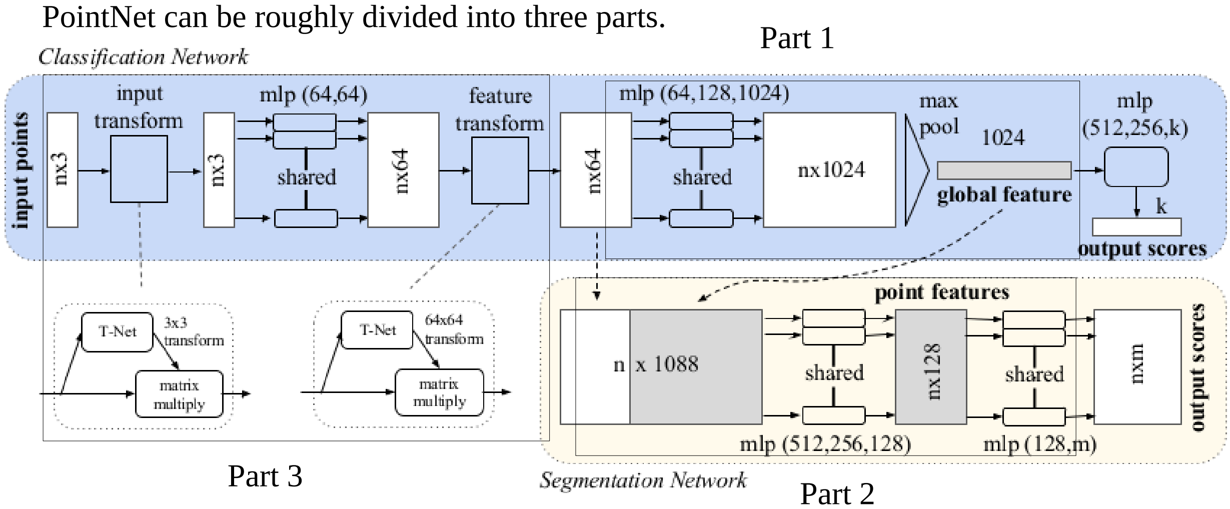
Visualization on PointNet

Zishuo Zheng 28.Dec.2017

1. Model Overview

PointNet can be roughly divided into three parts.



Part 1

Symmetry function for unordered input and aggregate information from all the points. In this part, we can get global features of all the points.

Part2

Local and global information aggregation. In this part, this model abstract new features based on the combination of global features and point features for segmentation work.

Part 3

Joint alignment network. In this part, PointNet predict an affine transformation matrix by the T-net and directly apply this transformation to the coordinates of input points. Such transformation can ensure the semantic labeling of a point cloud has to be invariant if the point cloud undergoes certain geometric transformations.

2. Problem Analyse and Solution

My target is constructing critical point sets Cs and upper-bound point sets Ns based on object classification network.

At the beginning, I considered neural network traceback and extracting expressions from neural networks to analyse. But then, I found it is difficult to find direct relationship between input and output of the PointNet especially after the transformation of T-net. Thus, I turned to the way of changing the input and comparing the output with original output.

According to the pipeline of classification network, we can simply consider that if the output of max pooling layer does not change, the result will not change. Becaues classification work is done based on global features and the max pooling layer abstracts the global features.

Here, we will discuss who to understand critical points and upper-bound points.

$$f(\{x_1, \dots, x_n\}) \approx g(h(x_1), \dots, h(x_n)), \qquad (1)$$

where $f : 2^{\mathbb{R}^N} \to \mathbb{R}$, $h : \mathbb{R}^N \to \mathbb{R}^K$ and $g : \underbrace{\mathbb{R}^K \times \cdots \times \mathbb{R}^K}_{n} \to \mathbb{R}$ is a symmetric function.

In paper 4.2, we can know that this model approximates h by a multi-layer perceptron network and g by a composition of a single variable function and a max pooling function.

Theorem 1. *Suppose* $f : \mathcal{X} \rightarrow \mathbb{R}$ *is a continuous set function w.r.t Hausdorff distance* $d_H(\cdot, \cdot)$. $\forall \epsilon > 0$, $\exists$ *a continuous function* h *and a symmetric function* $g(x_1, \ldots, x_n) = \gamma \circ MAX$, *such that for any* $S \in \mathcal{X}$,

$$\left| f(S) - \gamma\left(\underset{x_i \in S}{MAX}\{h(x_i)\}\right)\right| < \epsilon$$

where $x_1, \ldots, x_n$ *is the full list of elements in* S *ordered arbitrarily,* γ *is a continuous function, and MAX is a vector max operator that takes* n *vectors as input and returns a new vector of the element-wise maximum.*

In paper 4.3 Theorem 1, we can get the specific expressions about how does the neural network approximates the formula.

Based on two expressions above, I get the idea that the critical point is the point that determines the global features. In expression, the critical point is the point reach the max in function MAX(). As for the upper-bound point, it is the point that hidden behind the critical points in the expression of global features. In another word, critical points determines global features and upper-bound points including critical points and points that will not change the global features (In another word, smaller than global features).

3. Implement

Theorem 2. *Suppose* $\mathbf{u} : \mathcal{X} \rightarrow \mathbb{R}^K$ *such that* $\mathbf{u} = \underset{x_i \in S}{MAX}\{h(x_i)\}$ *and* $f = \gamma \circ \mathbf{u}$. *Then,*

(a) $\forall S, \exists\, \mathcal{C}_S, \mathcal{N}_S \subseteq \mathcal{X}$, $f(T) = f(S)$ *if* $\mathcal{C}_S \subseteq T \subseteq \mathcal{N}_S$*;*

(b) $|\mathcal{C}_S| \leq K$

Based on paper 5.3 Theorem2, I came up with the follow implement.

One of the difficulty of my implement is that I can not easily adjust the number of points in the input due to the model is trained and it is hard for me to change the input layer. But I find a way out.

For critical points, I rebuild the sample points one by one. First, I set all the points in the sample as the first point. Then, I put in the kth point to recover the kth position. Then I calculate the global features x of processed sample. Then comparing x with original global features to find out whether the point I take away will make global features larger. If it will, it is a critical point.

For upper-bound points, I travel through the edge-length-2 cube with step 0.2 to find points that will not make global features larger and put them into upper-bound point sets.

In a addition, according to $Cs \subseteq T \subseteq Ns$, critical point set can be got by reducing points from a sample and upper-bound point sets can be got by adding points into a sample.

4. Algorithm

Base on a sample P

To get critical points:

```
original_points = P.copy()              // save sample P
critical_points = [ ]
max_feature = []                        // record temporary max features
feature_point_index = []                // record which point results in the features' value
for points in P:
    P[ points] = P[0].copy()            // init sample with one value and then rebuild
    max_feature.append( -inf )
    feature_point_index.append( 0 )

for points in P:
    P [ points ] = original_points[ points ].copy()
    feature = get_global_features( P )          // feature is a vector with length 1024 here
    if there is a 'i' let feature[i] > max_feature[i] :
        max_feature[i] = feature[i]
        feature_point_index[i] = points

for index in feature_point_index:
    put original_points[ index ] into critical_points
```

To get upper-bound points:

```
set percent_e manually
standard_features = get_global_features( P )
upper_bound_points = P.copy()
test_point_index = 0

for points in P:
    if P[ points ] not in critical_points :      // find a not important point
        test_point_index = points
        break

for x in range(-2 , 2):
    for y in range(-2 , 2):
        for z in range(-2, 2):
            P[test_point_index] = (x,y,z)
            feature = get_global_features( P )  // feature is a vector with length 1024 here
            // for elements in feature, compare with standard_feature. If smaller, set 0;
            // else, set as feature – standard_features in this place
            distance = where(feature > standard_features, feature – standard_features, 0)
            // transform distance into relative distance
            distance = ( distance[i] /  standard_features[i] )  for every i
            if distance < percent_e :
                add (x,y,z) into upper-bound_points
```

4. Experiment

I take this experiment on the PointNet with 800 epochs' training with classification task.

Eavl mean loss	0.599767
Eval accuracy	0.884334
Eval avg class acc	0.854035

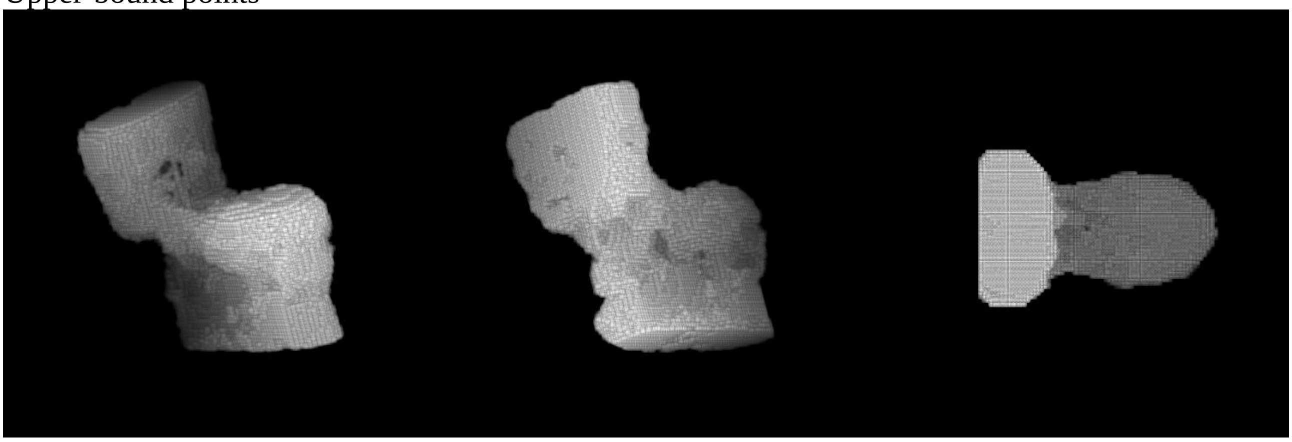
Then, I apply this model and the test set to generate critical points and upper-bound points. You can get code in file ‘visualize_points.py’ . To repeat the experiment, you just need to run ‘python visualize_points.py’.

Here are some results.

Critical Points

Orginal Points

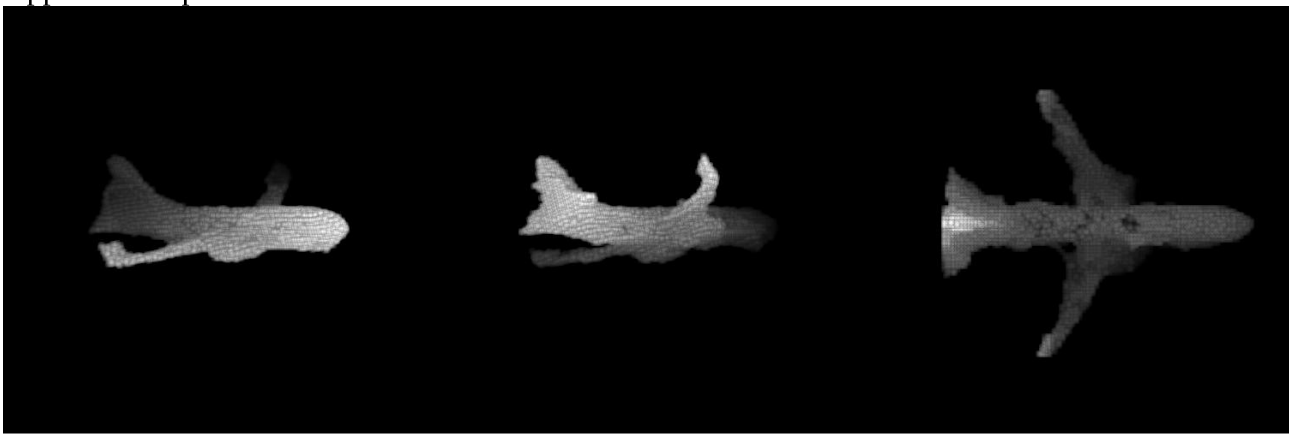
Upper-bound points



Critical points

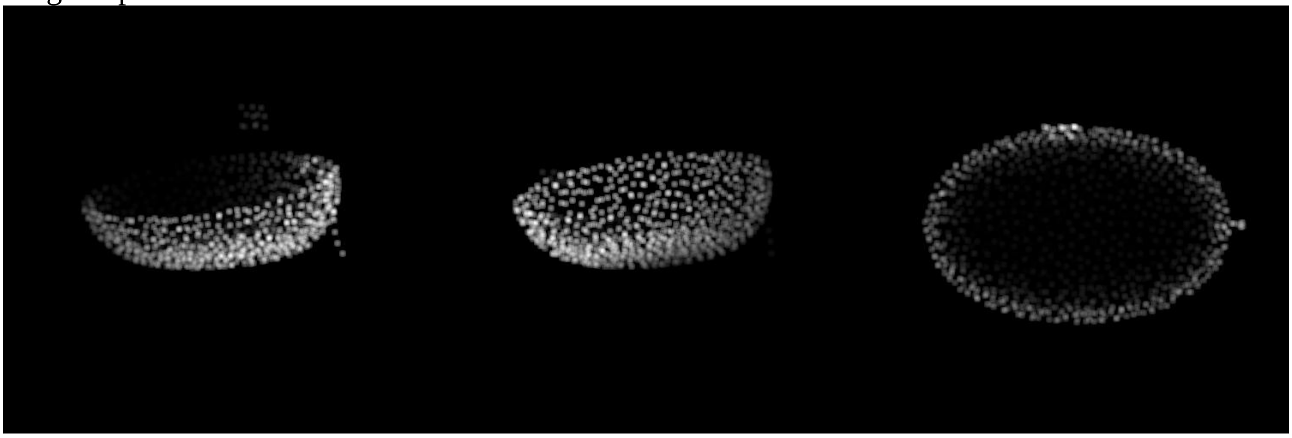
Original Points

Upper-bound points

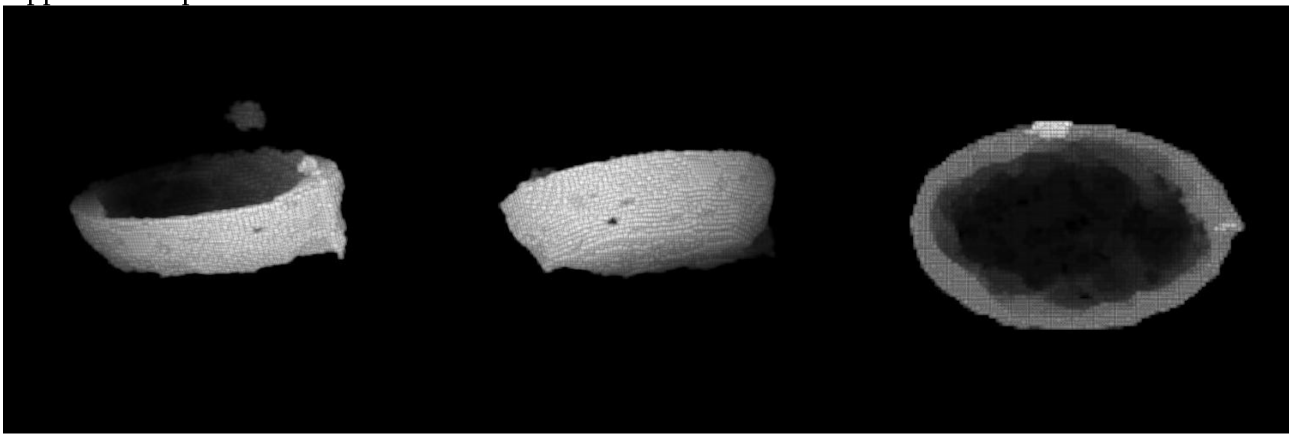


Critical points

Original points



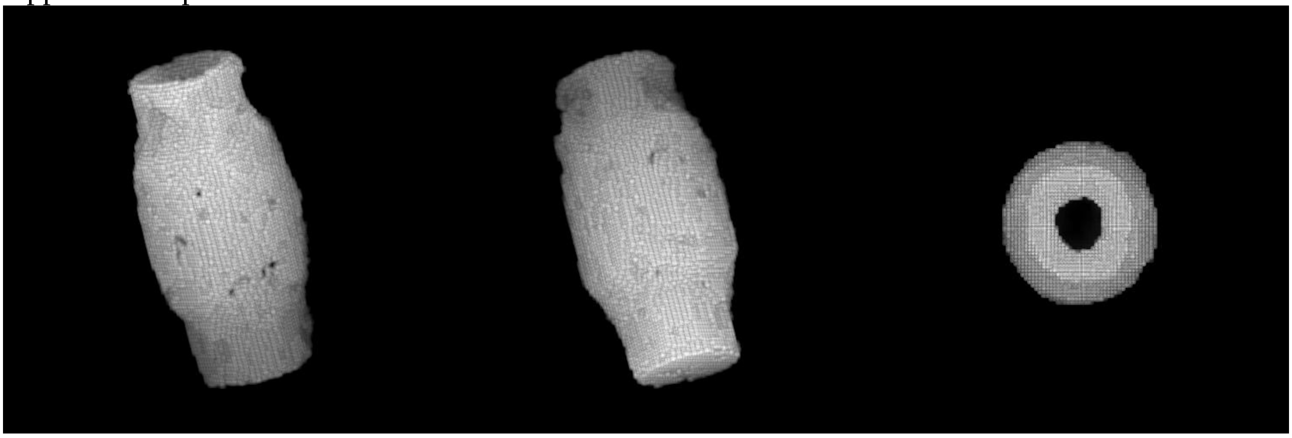
Upper-bound points



Critical points

Original points

Upper-bound points



5. Experiment Analyse

1. The visualization result is both depend on model accuracy and input sample. For some samples, if the model can not classify them correctly or have not learnt the features of them, the visualization result will be bad.
2. To get upper-bound points, traveling through the edge-length-2 cube with step 0.2 is a time costing job. Thus, intuitively, we can search just around the input point cloud to speed up this job. (done in the code)

3. Set different percent_e in part4 Algorithm for different categories will have better performance in upper-bound sets.