Adaptive Hierarchical Down-Sampling for Point Cloud Classification

Ehsan Nezhadarya, Ehsan Taghavi, Ryan Razani, Bingbing Liu and Jun Luo
Noah's Ark Lab, Huawei Technologies Inc.
Toronto, Canada
{ehsan.nezhadarya, ehsan.taghavi, ryan.razani, liu.bingbing, jun.luo1}@huawei.com

Abstract

Deterministic down-sampling of an unordered point cloud in a deep neural network has not been rigorously studied so far. Existing methods down-sample the points regardless of their importance to the network output. As a result, some important points in the point cloud may be removed, while less valuable points may be passed to next layers. In contrast, the proposed adaptive down-sampling method samples the points by taking into account the importance of each point, which varies according to application, task and training data. In this paper, we propose a novel deterministic, adaptive, permutation-invariant down-sampling layer, called Critical Points Layer (CPL), which learns to reduce the number of points in an unordered point cloud while retaining the important (critical) ones. Unlike most graph-based point cloud down-sampling methods that use k-NN to find the neighboring points, CPL is a global down-sampling method, rendering it computationally very efficient. The proposed layer can be used along with a graph-based point cloud convolution layer to form a convolutional neural network, dubbed CP-Net in this paper. We introduce a CP-Net for 3D object classification task that achieves high accuracy for the ModelNet40 dataset among point cloud based methods, which validates the effectiveness of the CPL.

1. Introduction

In most robotic applications, laser point cloud data plays a key role in perception of the surrounding environment. Autonomous mobile robotic systems in particular use point cloud data to train deep models to solve different problems, such as dynamic object detection, Simultaneous Localization and Mapping (SLAM), path planning, etc. With the introduction of many new methods, such as PointNet [19], PointNet++ [20], DGCNN [26], PointCNN [11], and SO-Net [10], extracting features from unordered point cloud with deep neural networks has become a highly active field of research. These methods are shown to be quite successful in point cloud classification benchmarks, such as ModelNet40 [27].

In practical scenarios, the number of points in the point cloud associated with an object may be quite large, especially as a result of using high density sensors such as Velodyne-64 [16]. One possible way to reduce computation is to down-sample the points in the point cloud as it gets passed through the network. A class of methods are proposed in which k-NN search [18] is used to find the neighbourhood for each point and down-sample according to these neighbourhoods. Such methods, however, trade one kind of expensive computation (neighbourhood search) for another one (processing large point cloud).

In another family of works such as [19] and [20], the 3D point cloud is processed directly, while other works transform point cloud to regular voxels such as methods in [27, 14, 6]. Transforming to regular voxels however, leads to loss of goemetric information and high computational complexity. Recently, the method introduced in RS-CNN [12] attempts to learn irregular CNN-like filters to capture local point cloud features which achieves the state-of-the-art accuracy in classification. In addition to these papers, a deep learning network for point cloud sampling is presented in [5]. This method produces a down-sampled point cloud from raw and unordered input point cloud which is not guaranteed to be a subset of the original one. Therefore, a post-processing matching step is required, leading to a more complex system.

In order to fully leverage a down-sampling method, what is highly needed is a deterministic content-sensitive but fast way of down-sampling an unordered point cloud that can be integrated into a deep neural network – a technique similarly effective and efficient as max pooling in conventional CNN. In this paper, we introduce the *Critical Points Layer* (CPL), which meets these requirements.

Unlike previous down-sampling methods that generate a set of points different from the input points, CPL not only *selects* the output points from the input, but also down-samples the points within the network in a way that the *critical* ones are not lost in this process. Unlike random

sampling type of layers which generate randomly different set of points at inference time, CPL is a *deterministic* layer, producing the same set of points after each run. It is *invariant* to permutation of input points, i.e. order-agnostic. It is *adaptive* in that it learns to down-sample the points during training. Last but not least, it is a *global* method not limited to neighbourhood search, which makes it quite efficient.

2. Related Work

2.1. Deep Learning on Point Clouds

To cope with sparsity of point cloud, deep learning methods tend to voxelize space and then apply 3D CNNs to the voxels [14, 28]. A problem with this approach is that network size and computational complexity grow quickly with spatial resolution. On the flip side, lower spatial resolution means larger voxels and higher quantization error. One way to alleviate this problem is to use octree [25] or kd-tree [8] instead of voxels. In [8], for example, a kd-tree of point cloud is built and then traversed by a hierarchical feature extractor, exploiting invariance of the point cloud at different spatial scale. However, such methods still rely on subdividing a bounding volume and fail to exploit local geometric structure of the points themselves. In contrast, point-based neural networks does not require converting point clouds to another format. Resolution loss is thus avoided [29].

2.2. CNNs on Point Cloud as Graphs

A promising way to exploit local geometric information is to model point cloud as graph of unordered (non-euclidean) points and then apply CNNs to it. This important research direction [23] has two main variations.

Spectral methods redefine spatial graph convolution as a multiplication in the spectral domain [23, 22]. The first proposed methods along this line lack spatial locality of filters. Parameterization of convolution filters as Chebyshev polynomials of eigenvalues and approximately evaluating them lead to a computationally efficient way to create localized spatial filters [4]. Unfortunately, these filters are learnt in the spectral domain [1] and thus have to be the same for all graphs in the dataset. This means that where graph structure varies in the dataset, such as point clouds, a graph learnt on one shape cannot generalize to others.

Local spatial filtering [23, 17, 9, 24, 30, 3, 13, 15], in contrast, employs spatial filters. A notion of local patch on graphs is employed to allow an operator similar to convolution to be applied to each local patch. Depending on the specific correspondence between filter weights and nodes in each local patch, we have variants such as MoNet [15], GCNN [7] and DCNN [2]. Although much work has been done to apply spatial filtering for deep learning on general graphs, only a few methods, such as KCNet [21], FoldingNet [29], ECC [23] and DGCNN [26] use deep learning on *point cloud* graphs.

2.3. Point Cloud Down-Sampling in Deep Networks

While graph convolution on point clouds has recently received great attention, point cloud down-sampling is not properly explored in the literature. Such down-sampling is highly desirable for a few reasons:

- Most graph convolution methods on point cloud use k-NN search to find the neighbourhood of each point. Thus, down-sampling the points cuts the computational cost for subsequent convolution layers.
- Reducing the number of points in the network results in lower runtime memory usage.
- Down-sampling can boost robustness to certain perturbations in the input data.

In typical point-based neural networks, such as PointNet and DGCNN, the number of points in the point cloud is fixed throughout the network. PointNet++ [20] does down-sample the point cloud using farthest point sampling (FPS). However, since it generates overlapping partitions by finding the k-NN points around each sample point, it needs much more computational power than PointNet, due to the required search in the high-dimensional feature space.

KCNet [21] and FoldingNet [29] down-sample the graph using a graph-based max-pooling that takes maximum features over the neighbourhood of each node using a pre-built k-NN graph (KNNG). However, these methods provide no guarantee that the most important points, which we call *critical points*, will be passed to downstream. A point with less relevant features may be selected or generated, while an important one may be removed or devalued.

Moreover, the down-sampling used in some of these networks are static, where the sampling is only based on spatial locations of points in the input point cloud, but not on their corresponding learnt features. On the other hand, methods that do use feature space distance between points, such as PointNet++ [20], are computationally prohibitive as explained earlier. In another prominent method, [5] introduces a deep learning approach for optimized point cloud sampling in which a NN model is trained to generate a down-sampled point cloud from the original dataset. Finally, all these methods *generate* a set of new points, instead of *selecting* a subset of input points. This makes it difficult to track the contribution of each input point to the output.

In this paper, we introduce a computationally efficient *Critical Points Layer* (CPL), which down-samples the points *adaptively* based on the learnt features. CPL *globally* filters out unimportant points while keeping the important ones, according to a point's level of contribution to the global max-pooling (max-reduced feature vector). The CPL is computationally very efficient because it does not need

local nearest neighbour search. Moreover, since the feature vectors obtained from a graph convolution layer already contain the local neighbourhood information of each point that is important, the CPL yields a smaller subset without losing relevant information. In the next two sections, we explain the CPL in detail and report our experimental results.

3. Proposed Solution

In this section, we propose two new adaptive down-sampling methods that can be used in deep neural networks. The focus of the proposed methods is to introduce a systemic solution to down-sample the points (or the feature vectors associated to them) in an arbitrary neural network architecture. This differs with methods such as [5], in which a novel method is proposed to down-sample a specific dataset. Our proposed layers, named *Critical Points Layer* (CPL) and *Weighted Critical Points Layer* (WCPL), can efficiently down-sample the features related to an unordered point cloud, while being permutation invariant. In this section, CPL, WCPL and a systemic approach to use these two layers in deep neural networks and more specifically classification networks will be explained in details.

3.1. Critical Points Layer (CPL)

Lets assume the input to the CPL is an unordered point cloud with n points, each represented as a feature vector $\mathbf{x} \in \mathbb{R}^d$, where $\mathbb{R}$ is the set of real numbers and d is the dimension of the feature vector. The goal of CPL is to generate a subset of input points, called *Critical Points* (CP), with $m \leq n$ points, each represented as a feature vector $\mathbf{y} \in \mathbb{R}^l$, where l is the dimension of the new feature vector. The critical points of a point cloud are the points with maximum information that are needed to be preserved in a down-sampling (or pooling) process. These points may be changed based on the task and application.

The block diagram of the proposed Critical Points Layer (CPL) is illustrated in Figure 1a. In order to elaborate more on the functionality of CPL, its pseudo-code is also provided in Algorithm 1. The steps of the algorithm are explained in more details as follows:

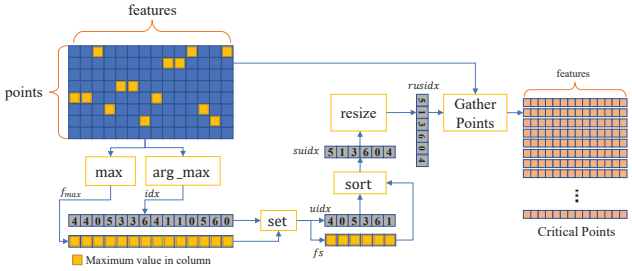
1. The input point cloud $\mathbf{F_S}$ is a matrix with n rows (corresponding to n input points) and d columns (corresponding to d-dimensional feature vectors).

2. In the first step (Operation 3), the maximum feature value is obtained for each column of the matrix $\mathbf{F}_S$. This is the same as the max-pooling operation in PointNet [19]. The resulting d-dimensional feature vector, denoted by $\mathbf{f_{max}}$, has the same dimension as input feature vectors and can be independently used for classification and segmentation tasks. However, we are interested in down-sampling the input points rather than generating a single feature vector out of them. To this aim, the index of each row with a maximum feature value is also saved in the index vector $\mathbf{idx}$. Vector $\mathbf{idx}$ contains the indices of all the points that have contributed to the feature vector $\mathbf{f_{max}}$. By definition, we call these points, the *Critical Points* (CP). These are the important points that should be preserved in the down-sampling process.

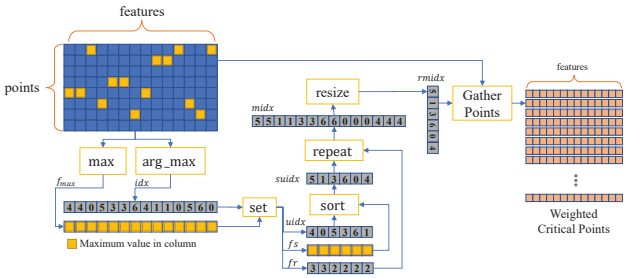
3. Index vector $\mathbf{idx}$ may contain multiple instances of the same point. To avoid these repetitions, unique indices are extracted from $\mathbf{idx}$, using the "set (unique)" function (Operation 6). Output set which has the unique indices is called the *Critical Set* (CS) and is denoted by $\mathbf{uidx}$. Beside finding the unique vector, we also add-up the feature values from $\mathbf{f_{max}}$ that correspond to the same point or index (Operation 7). Resulting feature vector $\mathbf{f_S}$ will be later used to sort the input points.

4. Next, feature vector $\mathbf{f_S}$ is sorted (in an ascending order). Corresponding indices in $\mathbf{uidx}$ are also rearranged based on the sorting output (Operation 12), resulting in an index vector which is denoted by $\mathbf{suidx}$. This step is necessary for the following sampling (resizing) operation. It also makes CPL invariant to the order of input points.

5. Number of elements in $\mathbf{suidx}$ may differ for different point clouds in the input batch. For batch processing



(a) Critical Points Layer (CPL)



(b) Weighted Critical Points Layer (WCPL)

Figure 1: Illustration of the proposed CPL and WCPL.

Algorithm 1 (Weighted) Critical Points Layer (CPL/WCPL)

function $(\mathbf{F_O}, \mathbf{f_O}, \mathbf{suidx})$ = CPL($\mathbf{F_S}$, $\mathbf{F_I}$, k)
  for $i = 0$ **to** ncols($\mathbf{F_S}$) $-$ **1 do** ▷ max pooling
    $\mathbf{f_{max}}[i]$ = max($\mathbf{F_S}[:, i]$)
    $\mathbf{idx}[i]$ = argmax($\mathbf{F_S}[:, i]$)
  end for
  $\mathbf{uidx}$ = unique($\mathbf{idx}$) ▷ set operation
  $\mathbf{f_S}[\mathbf{j}] = \sum_{\mathbf{idx[i]=uidx[j]}} \mathbf{f_{max}}[\mathbf{i}]$
  if WCPL **then**
    $\mathbf{fr}[\mathbf{j}] = |\{i \mid \mathbf{idx}[i] = \mathbf{uidx}[j]\}|$ ▷ frequency of $\mathbf{uidx}[j]$ in $\mathbf{idx}$
  end if
  $-, \mathbf{l}$ = sort($\mathbf{f_S}$) ▷ sorting
  $\mathbf{suidx} = \mathbf{uidx}[\mathbf{l}]$
  if WCPL **then**
    $\mathbf{midx}$ = repeat($\mathbf{suidx}, \mathbf{fr}$)
    $\mathbf{rmidx}$= resize($\mathbf{midx}$, k) ▷ nearest-neighbor resizing
    $\mathbf{F_O} = \mathbf{F_I}[\mathbf{rmidx}, :]$ ▷ point collection
  else
    $\mathbf{rsuidx}$= resize($\mathbf{suidx}$, k) ▷ nearest-neighbor resizing
    $\mathbf{F_O} = \mathbf{F_i}[\mathbf{rsuidx}, :]$ ▷ point collection
  end if
  for $i = 0$ **to** ncols($\mathbf{F_O}$) $-$ 1 **do** ▷ max pooling
    $\mathbf{f_O}[i]$ = max($\mathbf{F_O}[:, i]$)
  end for

  if WCPL **then**
    return $(\mathbf{F_O}, \mathbf{f_O}, \mathbf{rmidx})$
  else
    return $(\mathbf{F_O}, \mathbf{f_O}, \mathbf{rsuidx})$
  end if
end function

however, these numbers need to be the same. To address this, for each point cloud in the input batch, the index vector $\mathbf{suidx}$ is up-sampled to a fixed size vector $\mathbf{rsuidx}$ using an up-sampling method for integer arrays, such as *nearest neighbor* resizing (Operation 18).

6. As the final step, the up-sampled index vector $\mathbf{rsuidx}$, which contains the indices of all the critical points, is used to gather points and their corresponding feature vectors. Since different feature vectors may correspond to a single point, and because of the information being filtered in hidden NN layers, we may want to gather the features from other layers (denoted by $\mathbf{F_I}$) than those used for selecting the points (denoted by $\mathbf{F_S}$). However, critical points are defined based on the contribution of each point in the maximum feature vector obtained from $\mathbf{F_S}$, thus here we use $\mathbf{F_I} = \mathbf{F_S}$.

One of the main requirements of any layer designed for point cloud processing is its invariance to point cloud permutation. The proposed CPL, fulfills this requirement via following properties:

- Sorting the feature vector $\mathbf{f_S}$ in step 4 is order independent, because sorting is based on feature values and not based on indices of the points.
- Nearest-neighbor resizing in step 5 is invariant to swapping the index of the input points, i.e.

$$resize(sort(swap(\mathbf{uidx}))) = swap(resize(sort(\mathbf{uidx}))) \quad (1)$$

where $sort$ is applied based on feature values, and $swap$ is applied on index only.

3.2. Weighted Critical Points Layer (WCPL)

In CPL, a point in the point cloud is counted as a critical point if any of its features contributes to the output maximum feature vector $\mathbf{f_{max}}$, regardless of the number of its contributing features. For example, if a point contributes with two of its features, while another point has ten contributing features, both are treated the same in CPL. In other words, in CPL the "*importance*" of a point has a binary value: a given point is either important (critical) or unimportant (uncritical). In this section, we introduce a modified version of CPL, called Weighted Critical Points Layer (WCPL). The proposed WCPL (Figure 1b) assigns weights to points based on their level of contribution to $\mathbf{f_{max}}$.

In this context, to increase the weight of a point by a factor of C, we repeat the point index C times. By increasing the repetition frequency, the probability of selecting the point in the down-sampling process will also increase. From another point of view, in WCPL, the probability of missing a critical point in the output is lower than that in CPL. The pseudo-code of WCPL is given in Algorithm 1 using the **if** statements.

3.3. Critical Points Net (CP-Net)

In this section, we propose a hierarchical architecture to apply deep convolutional neural networks to point clouds, by systematically reducing the number of points using the proposed CPL/WCPL. In the proposed network model, named *Critical Points Net* (CP-Net), any graph convolution method, such as DCNN [2], GCNN [13], MoNet [15] or EdgeConv (from DGCNN [26]) can be used in convolution layers. The block diagram of CP-Net using EdgeConv as an example is shown in Figure 2.

The input in Figure 2 is an unordered point cloud of size n. In the first step, the point cloud is passed into a convo-

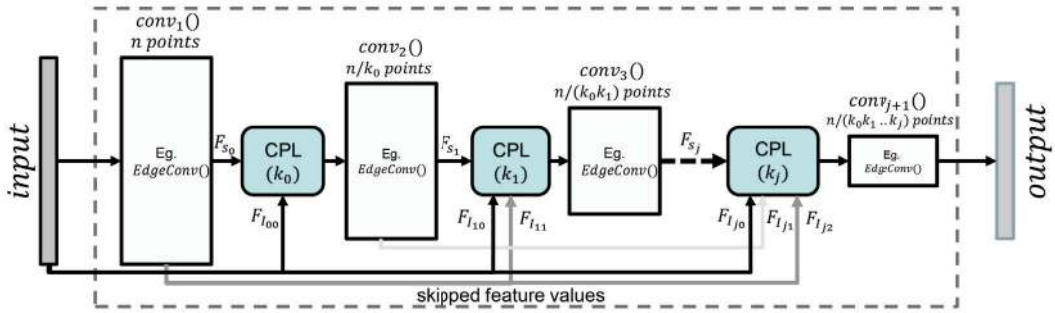


Figure 2: General block diagram of the proposed CP-Net.

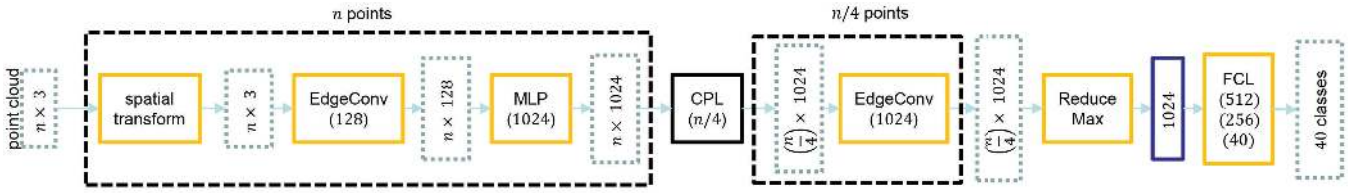


Figure 3: The proposed point cloud classification network using CP-Net.

lution layer of choice to filter the input into a richer set of features. The filtered output point cloud F_{S_0} is then used as an input to the first CPL/WCPL. Using a CPL/WCPL with down-sampling factor k_0, the number of points in F_{S_0} is reduced to n/k_0 points. These steps are repeated for as many times as necessary to achieve a desired size of the point cloud (both in terms of number of points and feature vector size). Note that at j-th CPL/WCPL block, one can also benefit from using or concatenating features from all or some of the previous layers, i.e., $\{F_{I_{j0}}, F_{I_{j1}}, \ldots, F_{I_{jj}}\}$, as long as they correspond to the same points. As a result, the number of output points will be $\frac{n}{k_0 k_1 \ldots k_{j+1}}$.

3.4. CP-Net for 3D Object Classification

Here we give an example of CP-Net application in the 3D object classification problem. Block diagram of the proposed network is illustrated in Figure 3. The network is composed of three subnets: 1) n-point feature extraction subnet, 2) $(n/4)$-point subnet and 3) classification subnet. The detailed steps of the proposed network are as follows:

- Network input is an unordered point cloud of size $n \times 3$, where each point is a 3D vector.

- The input data goes through a spatial transformer network as explained in [19], to make it robust against any rigid transformation, including rotation and translation. It is worth noting that instead of using the original input, a modified version of EdgeConv [26] edge feature is used for spatial transformation, as explained in the next step [1].

- The output of the spatial transform goes into a filtering CNN, here EdgeConv [26], to produce richer features. Unlike the original EdgeConv [26] operator which uses two kernels in the edge feature function, we use the triple-kernel version $h_\Theta(x_i, x_j - x_i, (x_j - x_i)^2)$, where $(x_j - x_i)^2$ is element-wise square operation between each point x_i and its neighbouring point x_j. In the proposed network, applying the EdgeConv with 128 filters to the input point cloud of size $n \times 3$, results in a point cloud of size $n \times 128$.

- A multi-layer perceptron (MLP) layer expands the feature dimension from 128 to 1024 features, resulting in a point cloud of size $n \times 1024$.

- Next, CPL/WCPL is applied to find the critical points and to reduce the number of input points. As shown in Section 4, this step reduces the computational complexity without any loss in the classification accuracy. A down-sampling factor of $1/4$ is chosen to reduce the number of points from n to $n/4$.

- Another EdgeConv layer is used to filter the point cloud, this time by preserving the depth and size to further process the received point cloud. Note that reducing the number of points in the previous layer highly reduces the computational complexity of the this layer.

[1] The proposed layer can be efficiently used along with other complex graph convolution layers such as EdgeConv [26] or MoNet [15]. In this paper, we use a modified version of EdgeConv [26], for graph convolution.

- A reduce-max layer is used to generate a vector of size 1024, out of the point cloud of size $n \times 1024$.
- Finally, fully connected layers of size $512, 256$ and 40 are applied to transform the feature vector of size 1024 to the number of classes in the ModelNet40 dataset [27], which is 40.

In the proposed 3D classification method, standard softmax cross entropy is used as the loss function. In addition, all layers include a ReLU activation function and batch normalization.

4. Experiments

4.1. Data preprocessing

We evaluate our model on ModelNet40 3D object classification dataset [27]. The dataset contains $12,311$ meshed CAD models from 40 different object categories out of which $9,843$ models are used for training and $2,468$ models for testing. From each model mesh surface, 1024 points are uniformly sampled and normalized to the unit sphere. For data augmentation, we randomly scale, rotate and shift each object point cloud in the 3D space.

4.2. Training Details

To train the model, we use Adam optimizer with an initial learning rate 0.001 and exponentially decay it with a rate of 0.5 every $200,000$ steps. The decay rate of batch normalization starts from 0.5 and is increased to 0.99. The Dropout with probability 0.5 is used in the last two fully-connected layers. Training the network with TensorFlow on an Nvidia P100 GPU with batch size 32, takes $9-10$ hours for 400 epochs.

4.3. Statistical Results

To evaluate the performance of a 3D point cloud classification method, we use both *overall accuracy* and *per-class average accuracy*, calculated over all the test samples.

The classification accuracy results for our proposed CP-Net/WCP-Net are shown in Table 1 with comparisons against the previously proposed methods. As illustrated, our CP-Net/WCP-Net methods rank as the runner-up to [12] and surpass the accuracy of all other methods in Table 1 and of ModelNet40 benchmark leader board.

4.4. Qualitative Results

Figure 4 shows how the proposed CPL learns to down-sample different point clouds. The original point clouds of size 1024 for the object classes *lamp*, *airplane*, *flower-pot*, *laptop* and *car* are shown in Figure 4(a). Figures 4(b-e) correspond to the outputs obtained using the down-sampling ratio of 0.25, at epochs 1, 100, 200 and 300, respectively. As seen in Figure 4(b), at the beginning of the training, some important parts of the objects, such as lamp column, flower leaves and laptop screen are partially lost as a result of down-sampling. After 300 epochs of training however, CPL learns to down-sample the point cloud such that the *critical points* of the object are mostly retained. In the context of point cloud classification, by *important points* of an object we mean those points that contain the necessary information to discriminate between different objects in the dataset.

Figure 4(f) shows the corresponding point clouds down-sampled by the ratio of $\frac{1}{16}$, after 300 training epochs. As seen, the important points of each object for our classification task are still preserved even in such small 64-point point clouds. The lamp column, airplane wings and six corners of the laptop are some examples of the preserved important object parts.

Algorithm	Overall Accuracy (%)	Mean Class Accuracy (%)
Vox-Net [14]	83.00	85.9
ECC [23]	83.2	-
SO-Net [10]	89.16	-
Pointnet [19]	89.20	86.0
Pointnet++ [20]	90.70	-
KCNet [21]	91.0	-
KD-Net [8]	91.8	-
DGCNN[30] (1 vote)	91.84	89.40
RS-CNN [12]	93.6	-
Ours (CP-Net)	92.33	89.90
Ours (WCP-Net)	92.41	90.53

Table 1: Classification accuracy results on ModelNet40 dataset [27], for input size 1024×3.

Method	Double Kernel	Triple Kernel
DGCNN	91.84 (135ms)	89.26 (141ms)
CP-Net	**91.88** (115ms)	92.33 (119ms)
WCP-Net	91.76 (116ms)	**92.41** (120ms)

Table 2: Effect of edge feature kernels on overall classification accuracy (%) and execution time (in ms).

4.5. Ablation studies

EdgeConv Kernels The effect of using two and three kernels (in EdgeConv operator used in DGCNN [26]) on the overall classification accuracy and execution time is shown in Table 2. For double-kernel version, we use the one used in [26]. The triple-kernel version is defined in section 3.4-c. As seen, the triple kernel version is computationally more complex than the double kernel version. In both cases, not only the proposed CP-Net/WCP-Net outperforms the DGCNN in classification accuracy, it is computationally less complex, due to CPL/WCPL point cloud down-sampling.

Effect of Bottleneck Dimension The effect of bottleneck layer size (number of features in the output feature vector) on classification accuracy is shown in Table 3.

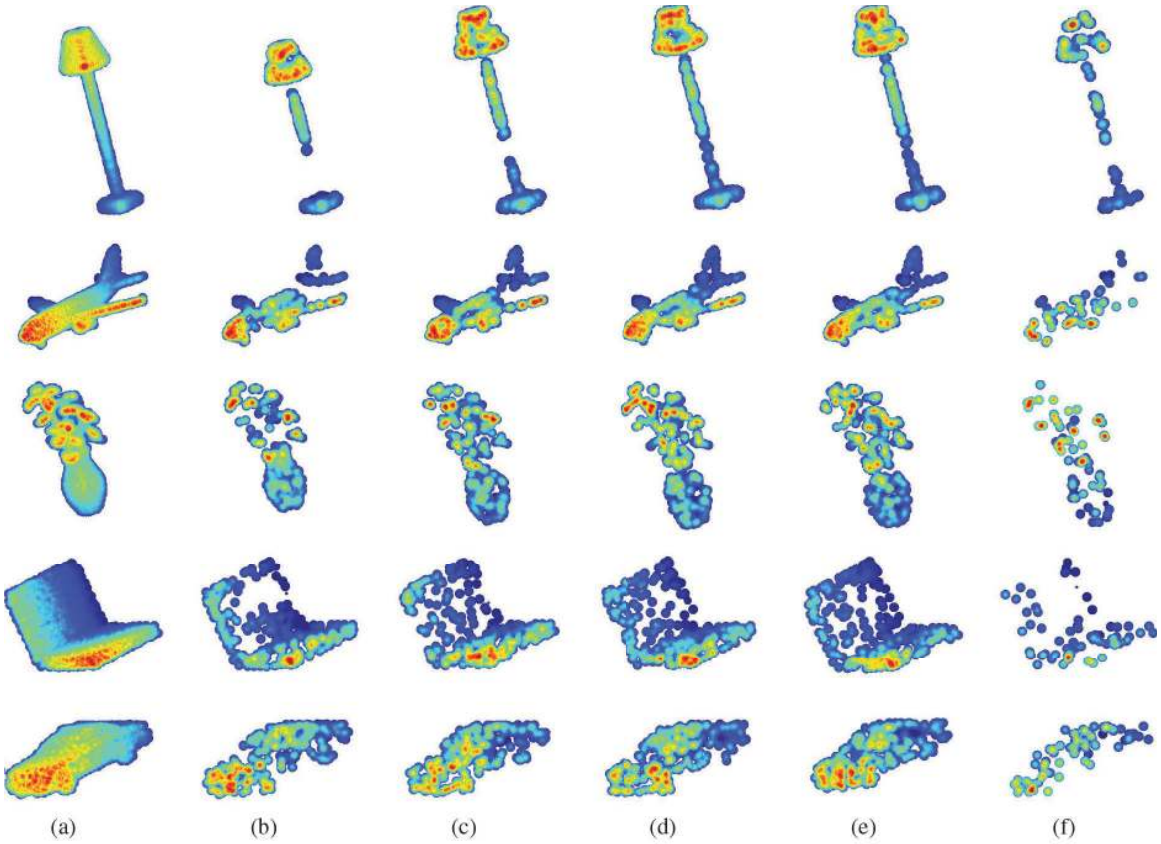


Figure 4: (From top to bottom and left to right) The original point clouds for the laptop and car object categories in ModelNet40 dataset [27], and their down-sampled versions (with ratio $\frac{1}{4}$) obtained after training the classification CP-Net, shown in Figure 3, for 1, 100, 200 and 300 epochs. (f) The result of training for 300 epochs with down–sampling ratio $\frac{1}{16}$. The images are color coded to reflect the depth information.

Clearly, increasing the bottleneck layer size improves the accuracy, however it almost saturates at around 1024 features. Note that even with the bottleneck size of 64, accuracy of the proposed CP-Net (%89.35) is more than that of the PointNet with bottleneck size of 1024 (%89.20).

	64	128	256	512	1024
CP-Net	89.35	89.83	90.85	91.94	92.33
WCP-Net	89.16	89.71	90.73	91.54	92.41

Table 3: Effect of bottleneck dimension on accuracy (%).

Effect of Down-Sampling Ratio Table 4 shows the effect of down-sampling ratio on classification accuracy. As expected, the more the point cloud is shrunk, the lower the accuracy is. This is because some of the important information about the object is lost as a result of down-sampling. The proposed CPL and WCLP layers however, preserve the important object points as much as possible. This can be verified from Table 4, where the difference between the accuracy values at down-sampling ratios 1 and $1/16$ (corresponding to point clouds of size 1024 and 64 points) is only %0.73. This means that in the down-sampling process, CPL preserves the most important information of each object, so that with such small number of points, objects are still classified with high accuracy.

	1	1/2	1/4	1/8	1/16
CP-Net	92.25	92.24	92.33	92.29	91.52
WCP-Net	92.09	92.15	92.41	92.03	91.81

Table 4: Effect of down-sampling ratio on accuracy (%).

Taking the effect of down-sampling for a ratio of 4, i.e., 256 points, it is worth comparing the accuracy of CPL with a random down-sampler. If we use random down-sampling of ratio 4 instead of CPL/WCPL, the accuracy of the CP-Net classification network drops to 91.47 from 92.33 and 92.41 for CPL and WCPL, respectively, which shows the effectiveness of CPL in obtaining the critical points and features in the architecture. It is worth noting that although using random down-sampling is viable in training, it is not math-

Method	Model Size (MB)	Inference Time (ms)	Accuracy (%)
PointNet	40	36.1	89.20
PointNet++	12	232.2	90.70
KCNet	**11**	**26.4**	91.0
DGCNN	21	134.5	91.84
CP-Net/WCP-Net	57	118.9	**92.33/92.41**

Table 5: Model size (in MB), inference mode execution time on P100 GPU (in ms) and classification accuracy (%).

ematically and statistically sound at inference. The reason is that every time the inference is run, the random sampler produces a new set of indices for down-sampling, resulting in a non-deterministic output/classification for the same input. Besides, our experiments showed that unlike CPL, random down-sampling is unable to find the critical points of the input point cloud.

Time and Space Complexity

Table 5 compares the execution time of running the proposed CP-Net/WCP-Net with other state-of-the-art object classification methods, on an Ubuntu machine with a single P100 GPU and an Intel 1.2 GHz CPU. Batch size of 32 1024-point point clouds is used in all the experiments.

In terms of model size, the smallest models are generated by KCNet. CP-Net and WCP-Net generate the largest models due to their larger number of network parameters. The model size can be reduced by decreasing the bottleneck dimension.

In terms of computational complexity, CP-Net and WCP-Net run faster than both DGCNN and PointNet++. However, they are slower than PointNet and KCNet. The lower computational complexity of CP-Net in comparison with DGCNN is due to the employment of CP layer in its network. Similarly, by a proper network design, the proposed CPL/WCPL is expected to accelerate other classification deep networks, such as KCNet and PointNet++.

5. Conclusion

In this paper we propose a new deterministic adaptive down-sampling method that can be trained to pass on the most important points (critical points), to the next layers in a neural network. Two down-sampling layers, the critical points layer (CPL) and its weighted version, weighted CPL (WCPL) are proposed. As a systemic approach of using CPL/WCPL in deep neural networks, CP-Net, a hierarchical implementation of the these layers, is also introduced. Finally, a deep classification network is designed based on the proposed CP-Net for 3D object classification. Experimental results using a common dataset show the competitiveness of the proposed method in terms of classification accuracy vs computational complexity trade-off, in comparison with previous state-of-the-art methods.

Last but not least, the effectiveness of CPL to dynamically down-sample the unordered data, inspires the design of a new type of auto-encoders that can handle unordered data such as point clouds. This approach is already under investigation and results will be shown in future works.

References

[1] William N Anderson Jr and Thomas D Morley. Eigenvalues of the laplacian of a graph. *Linear and multilinear algebra*, 18(2):141–145, 1985.

[2] James Atwood and Don Towsley. Search-convolutional neural networks. *CoRR*, 2015.

[3] Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 3189–3197, 2016.

[4] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[5] Oren Dovrat, Itai Lang, and Shai Avidan. Learning to sample. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2760–2769, 2019.

[6] Matheus Gadelha, Rui Wang, and Subhransu Maji. Multiresolution tree networks for 3d point cloud processing. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 103–118, 2018.

[7] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[8] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 863–872. IEEE, 2017.

[9] Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *arXiv preprint arXiv:1705.07664*, 2017.

[10] Jiaxin Li, Ben M Chen, and Gim Hee Lee. So-net: Self-organizing network for point cloud analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9397–9406, 2018.

[11] Yangyan Li, Rui Bu, Mingchao Sun, and Baoquan Chen. Pointcnn. *arXiv preprint arXiv:1801.07791*, 2018.

[12] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8895–8904, 2019.

[13] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 37–45, 2015.

[14] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 922–928. IEEE, 2015.

[15] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proc. CVPR*, number 2, page 3, 2017.

[16] Frank Moosmann and Christoph Stiller. Velodyne slam. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 393–398. IEEE, 2011.

[17] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023, 2016.

[18] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[19] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4, 2017.

[20] Charles Ruizhongtai 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*, pages 5099–5108, 2017.

[21] Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. Mining point cloud local structures by kernel correlation and graph pooling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 4, 2018.

[22] David I Shuman, Benjamin Ricaud, and Pierre Vandergheynst. A windowed graph fourier transform. In *Statistical Signal Processing Workshop (SSP), 2012 IEEE*, pages 133–136. Ieee, 2012.

[23] Martin Simonovsky and Nikos Komodakis. Dynamic edge–conditioned filters in convolutional neural networks on graphs. In *Proc. CVPR*, 2017.

[24] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 1(2), 2017.

[25] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (TOG)*, 36(4):72, 2017.

[26] 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.

[27] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[28] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Computer Vision and Pattern Recognition*, 2015.

[29] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, volume 3, 2018.

[30] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of AAAI Conference on Artificial Inteligence*, 2018.