Going Deeper with Point Networks

Eric-Tuan Le
University College London
eric-tuan.le.18@ucl.ac.uk

Iasonas Kokkinos
University College London
i.kokkinos@cs.ucl.ac.uk

Niloy J. Mitra
University College London
n.mitra@cs.ucl.ac.uk

Abstract

In this work we introduce three generic point cloud processing blocks that improve both accuracy and memory consumption of state-of-the-art networks thus allowing to design deeper and more accurate networks. The novel processing blocks are: a multi-resolution point cloud processing block; a convolution-type operation for point sets that blends neighborhood information in a memory-efficient manner; and a crosslink block that efficiently shares information across low- and high-resolution processing branches. Combining these blocks allows us to design significantly wider and deeper architectures. We extensively evaluate the proposed architectures on multiple point segmentation benchmarks (ShapeNet-Part, ScanNet, PartNet) and report systematic improvements in terms of both accuracy and memory consumption by using our generic modules in conjunction with multiple recent architectures (PointNet++, DGCNN, SpiderCNN, PointCNN). We report a 3.4% increase in IoU on the -most complex- PartNet dataset while decreasing memory footprint by 57%.

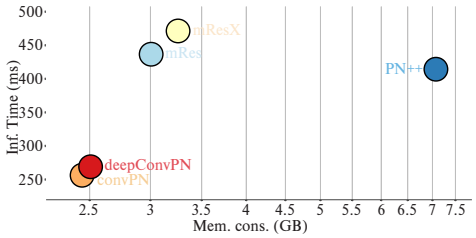

Figure 1: Memory footprint and inference speed of network variations: our multi-resolution (mRes), and crosslink (X) network blocks decrease the memory footprint, while our convolution-type network (conv) decreases both memory consumption (-67%) and inference time (-41%) compared to the PointNet (PN) baseline.

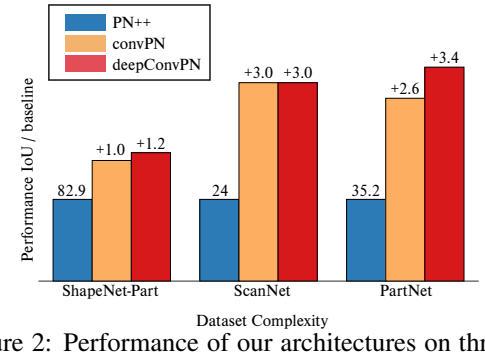

Figure 2: Performance of our architectures on three segmentation benchmarks of increasing complexity. As the data complexity grows, the spread in performance increases between the different networks. Our deep network clearly outperforms shallow architectures on the –most complex– PartNet dataset (+3.4% over PointNet++).

1 Introduction

Recent works in geometry processing have shown that the successes of deep learning in computer vision can carry over to graphics and 3D shape analysis [26, 37, 5, 2]. Still, the multi-faceted and often unstructured 3D data dictates re-inventing for geometry processing the functionality of simple image processing blocks, such as multi-resolution processing or convolution operations. These blocks gather information around a given position (pixel in images, point in a point clouds or meshes, or voxel in

The project page can be found on the following link: https://github.com/erictuanle/GoingDeeperwPointNetworks

volumetric grids), process it, and pass on to subsequent processing stages an informative description of the position's local and global context.

In the case of a polygonal mesh or a voxel-based shape representation, the explicit local structure of the signal can be exploited to construct close counterparts to image-based convolutions, e.g., in terms of mesh convolutions [27, 6, 11] or 3D volumetric convolutions [5, 12, 29]. In contrast, when operating with unstructured point clouds, one has to resort to more elementary local pooling operations that group information within a neighborhood based on Euclidean distance.

An exemplar of such methods is the recently introduced PointNet architecture [25] and its extension, PointNet++ [26]. The methods are simple yet extremely effective as demonstrated by the high accuracy across multiple benchmark tasks (e.g., classification, segmentation, etc.). Despite these substantial improvements over conventional architectures [32, 35, 36] and desirable properties such as permutation-invariance or quantization-free representation, point cloud processing networks use a limited repertoire of building blocks, both in terms of diversity, but also in terms of network width and depth. When compared to the improvements that have been achieved by hand-engineering [13, 14, 28] or learning [23, 22] the architecture of deep image processing networks, point network design is still in its infancy.

Our paper opens up the possibility for research in this direction by adding new design choices to the armament of point network design, introducing building blocks with lower memory footprint, faster inference time, and better optimization behaviour. In particular, one of the main bottlenecks for advancing in this direction is the memory-intensive nature of point processing networks. As detailed in Sec. 3.1, the PointNet++ architecture and its variants replicate point neighborhood information, letting every node carry in its feature vector information about all of its neighborhood; this results in significant memory overhead, and limits the number of layers, features and feature compositions one can compute. As has been witnessed repeatedly in the image domain, e.g., [13, 14, 43], the depth and breadth of a network directly correlates with accuracy. Furthermore, current network design often ignores computation time, which can also be reduced by careful network design. This affects both training speed and, more crucially, inference time.

In this work, we enhance deep point set processing networks by introducing three techniques that improve accuracy and memory footprint, without compromising on inference speed. Building on these techniques, we show that going deep on complex dataset increases prediction accuracy with a very low impact on network efficiency, measured in terms of inference time and memory footprint.

Firstly, in Sec. 3.2, we introduce a *multi-resolution* variant for multi-scale networks which results in a 58% decrease of the memory footprint, while sustaining the original network accuracy. In particular, PointNet++ captures multi-scale context information by progressively performing grouping with increasingly large radii on the original shape. Instead, we perform grouping at multiple radii and allow mixing of information early on in the network, which still delivers the multi-scale context, but at a reduced memory and computational cost.

Secondly, in Sec. 3.3, we replace the grouping operation used in point cloud processing networks with a low-memory alternative that is the point cloud processing counterpart of efficient image processing implementations of convolution. The resulting *'point convolution block'* is 67% more memory-efficient and 41% faster than its PointNet++ counterpart, and also has better behavior during training time due to more effective mixing of information across neighborhoods.

Thirdly, in Sec. 3.4, we improve the information flow across layers and scales within the network. Across layers we use the standard residual connections used in image processing, and across scales we introduce a new *cross-link block* that broadcasts multi-scale information across the network branches.

In the experimental result section we carefully ablate the impact of the individual blocks on network design and incrementally construct leaner and more efficient networks. The reduction in memory consumption allows us to experiment with deeper and wider network architectures, as well as with larger batchsize options. We experiment on the ShapeNet-Part, ScanNet and PartNet segmentation benchmarks, reporting systematic improvements over the PointNet++ baseline. As shown in Fig. 1 and Fig. 2, when combined these contributions deliver multifold reductions in memory consumption while improving performance, allowing us in a second stage to train increasingly wide and deep networks. On the most

complex dataset, our deep architecture achieves a +3.4% increase in IoU while decreasing both the memory footprint and the inference time by respectively -57% and -47%. We then experiment our generic blocks on three other architectures, DGCNN [37], SpiderCNN [39] and PointCNN [21] and observe similar improvements on memory (up to -63%) and IoU (up to +2.1%).

Supported by these promising results, we anticipate that our proposed design choices will become indispensable with the advent of larger datasets [24, 17] while also opening the way to experiment with deeper architecture search [23, 22] for point cloud processing.

2 Related Work

2.1 Learning in Point Clouds

Learning-based approaches have recently attracted significant attention in the context of Geometric Data Analysis, with several methods proposed specifically to handle point cloud data, including PointNet [25] and several extensions such as PointNet++ [26] and Dynamic Graph CNNs [37] for shape segmentation and classification, PCPNet [10] for normal and curvature estimation, P2P-Net [40] and PU-Net [42] for cross-domain point cloud transformation, etc.

Although many alternatives to PointNet have been proposed [31, 21, 20, 44] to achieve higher performance, the simplicity and effectiveness of PointNet and its extension PointNet++ make it popular for many other tasks [41].

Taking PointNet++ as our starting point, our aspiration has been to facilitate the transfer of network design techniques developed in computer vision to point cloud processing. In particular, compelling accuracy improvements have been obtained in vision with respect to the original AlexNet network [19] by engineering the scale of the filtering operations [45, 30], the structure of the computational blocks [33, 38], or the network's width and depth [13, 43]. As we discuss below, a catalyst for experimenting with a larger space of network architecture is the reduction of memory consumption - which has motivated us to design lean alternatives to point processing networks.

2.2 Memory-Efficient Networks

One of the main bottlenecks in training deep networks is memory. The memory complexity of the standard back-propagation implementation grows linearly in the network's depth: backprop requires retaining in memory all of the intermediate activations computed during the forward pass, since they are required for the gradient computation in the backward pass.

Several methods have been recently proposed to bypass this problem by trading off speed with memory. Checkpointing techniques [4, 9] use anchor points to free up intermediate computation results, and re-compute them in the backward pass. This is 1.5 times slower during training, since one performs effectively two forward passes rather than just one. More importantly, applying this technique is easy for chain-structured graphs, e.g. recursive networks [9] but is not as easy for general Directed Acyclic Graphs, such as U-Nets, or multi-resolution networks like PointNet++. One needs to manually identify the graph components, making it cumbersome to experiment with diverse architectures.

Reversible Residual Networks (RevNets) [8] limit the computational block to come in a particular, invertible form of residual network. This is also 1.5 times slower during training, but alleviates the need for anchor points altogether. Still, it is unclear what is the point cloud counterpart of invertible blocks.

The methods we propose here to reduce the memory footprint are in line with current practice in deep learning for computer vision, and are inspired from multi-resolution processing, and efficient implementations of the convolution operation. As such they can be used as drop-in replacements in generic point processing architectures, without any additional effort from the network designer's side.

3 Method

We start with a brief introduction of the PointNet++ network, and then present our contributions to decreasing its memory footprint and improving its information flow. The resulting architecture is displayed in Fig. 3.

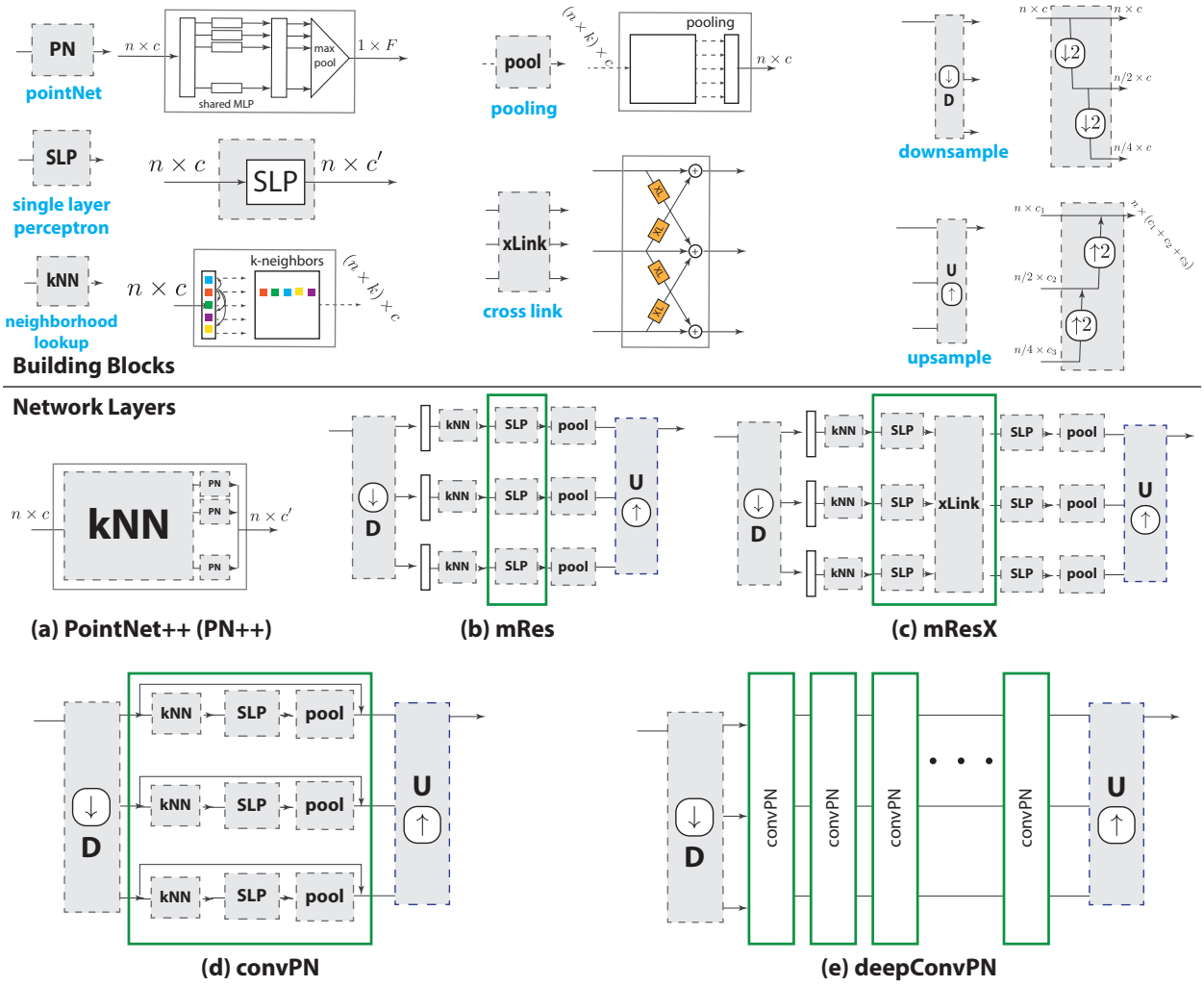

Figure 3: Top panel: elementary building blocks for point processing and network layers obtained from their composition. Apart from standard pooling and SLP layers, we introduce cross-link layers across scales, and propose multi-resolution up/down sampling blocks for point processing. Bottom panel: network layers constructed by composing the top-panel elements. The standard PN++ layer in (a) amounts to the composition of a neighborhood-based lookup and a PointNet element. In (b) we propose to combine parallel PointNet++ blocks in a multi-resolution architecture, and in (c) allow information to flow across branches of different resolutions through a cross-link element. In (d) we propose to turn the lookup-SLP-pooling cascade into a low-memory counterpart by removing the kNN elements from memory once computed; we also introduce residual links, improving the gradient flow. In (e) we combine the block in (d) with cross-links achieving better information flow through both residual and across-resolution links. Each of these tweaks to the original architecture allows for systematic gains in memory and computational efficiency. The green box indicates that the block can be grown in depth by stacking those green units.

3.1 PointNet and PointNet++ Architectures

Given a point set $P := \{\mathbf{p}_i\}$, PointNet [25] computes a global feature vector as $h(g(\mathbf{p}_1), g(\mathbf{p}_2), \dots)$, where $g()$ denotes an MLP and $h()$ is a symmetric function applied component-wise along $g(\mathbf{p}_i)$. Note that in PointNet pooling (via $h()$) happens only at the very end of the network.

PointNet++ [26] builds on top of PointNet as follows. First, each point $\mathbf{p}_i$ looks up its k-nearest neighbors and stacks them to get a point set, say $P^i_{N_k}$. Then, PointNet is applied to each such point set $P^i_{N_k}$ and the resultant feature vector assigned back to the corresponding point $\mathbf{p}_i$. While demonstrated to be extremely effective, PointNet++ has two main shortcomings: first, being reliant on PointNet, it also delays transmission of global information until the end; and second, because of explictly carrying around k-nearest neighbor information for each point, the network layers are memory intensive.

3.2 Multi-Resolution vs Multi-Scale Processing

Shape features can benefit from both local, fine-grained information and global, semantic-level context; their fusion can easily boost the discriminative power of the resulting features. As presented in Sec. 3.1, PointNet++ associates neighborhood of fixed radius and only in the later blocks of the network the pointsets are sampled and bigger radius ball searches executed. Hence, at early stages of the network, the points only has access to very local information.

We observe that this mode allows only very slow exchange of information among low-, mid- and coarse- scale information. Coarse-scale information is conveyed not necessarily by all points that are contained within a larger radius, but by obtaining potentially sparse measurements from a larger area. This underlies also the common log-polar sampling mechanism in computer and biological vision [1, 34, 18] where an constant number of measurements is obtained in concentric disks of geometrically increasing radii.

We therefore propose to extract neighborhoods of fixed size in downsampled versions of the original mesh. As shown in Fig. 4, in the coordinates of the original point cloud this amounts to increasing the effective grouping area, but now comes with a much smaller memory budget.

Figure 4: Comparison of multiscale processing (top) with multiresolution processing (down): multi-resolution processing allows us to process larger-scale areas while not increasing memory use, making it easier to elicit global context information.

We experience a 58% decrease in memory footprint on average on the three tested datasets.

3.3 Neighborhood Convolution

Having described our multi-resolution counterpart to multi-scale grouping, we now turn to improving the memory consumption in the grouping operation itself.

As shown in Figure 3, the existing PointNet++ grouping operation exposes the neighborhood of any mesh variable i, by concatenating all of its K neighboring D-dimensional vectors, $\mathbf{v}_{[i,k]}$ to form a tensor T:

$$T = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_{[1,1]} & \cdots & \mathbf{v}_{[1,K]} \\ \mathbf{v}_2 & \mathbf{v}_{[2,1]} & \cdots & \mathbf{v}_{[2,K]} \\ & \vdots & & \\ \mathbf{v}_N & \mathbf{v}_{[N,1]} & \cdots & \mathbf{v}_{[N,K]} \end{bmatrix} \quad (1)$$

of size $N \times D \times (K+1)$. Every vector of this matrix is processed separately by a Multi-Layer-Perceptron that implements a function $\text{MLP} : R^D \rightarrow R^{D'}$, while at a later point a max-pooling operation over the K neighbors of every vertex delivers a slim, $N \times D'$ matrix.

When training a network every layer constructs and retains such a matrix in memory, so that it can be used in the

backward pass to update the MLP parameters, and send gradients to earlier layers.

Algorithm 1: Low-memory grouping - Forward pass

Data: Input features tensor $\mathcal{T}_f$ $(N \times R^D)$, input spatial tensor $\mathcal{T}_s$ $(N \times R^3)$ and indices of each point's neighborhood for lookup operation $\mathcal{L}$ $(N \times K)$

Result: Output feature tensor $\mathcal{T}_f^o$ $(N \times R^{D'})$

begin

  /* Lifting each point/feature to $R^{D'}$ */

  $\mathcal{T}_{f'} \longleftarrow$ **SLP**$_f(\mathcal{T}_f)$

  $\mathcal{T}_{s'} \longleftarrow$ **SLP**$_s(\mathcal{T}_s)$

  /* Neighbourhood features $(N \times R^{D'} \rightarrow N \times R^{D'} \times (K+1))$ */

  $\mathcal{T}_{f'}^K \longleftarrow$ **IndexLookup**$(\mathcal{T}_{f'}, \mathcal{T}_{s'}, \mathcal{L})$

  /* Neighbourhood pooling $(N \times R^{D'} \times (K+1) \rightarrow N \times R^{D'})$ */

  $\mathcal{T}_{f'}^o \longleftarrow$ **MaxPooling**$(\mathcal{T}_{f'}^K)$

  FreeMemory$(\mathcal{T}_{s'}, \mathcal{T}_{f'}, \mathcal{T}_{f'}^K)$

  return $\mathcal{T}_{f'}^o$

end

The counterpart for a standard 2D image convolution amounts to forming a K^2 tensor in memory when performing $K \times K$ filtering and then implementing a convolution as matrix multiplication. This amounts to the `im2col` operation used for example in the `caffe` library to implement convolutions with General Matrix-Matrix Multiplication (GEMM) [15]. In point clouds the nearest neighbor information provides us with the counterpart to the $K \times K$ neighborhood. Based on this observation we propose to use the same strategy as the one used in memory-efficient implementations of image convolutions for deep learning.

Rather than maintaining the matrix in memory throughout, we free the memory as soon as the forward pass computes its output. As shown in Algorithm 2, in the backward pass we reconstruct the matrix *on the fly* from the outputs of the previous layer. We perform the required gradient computations and then return the GPU memory resources; we do not free the GPU memory, but rather detach the related vector from it. This retains the allocated GPU memory, but re-uses it for subsequent layers.

Using the on-the-fly re-computation of the tensor $\mathcal{T}$ has a positive impact on the backward pass. As shown

Algorithm 2: Low-memory grouping - Backward pass

Data: Input features tensor $\mathcal{T}_f$ $(N \times R^D)$, input spatial tensor $\mathcal{T}_s$ $(N \times R^3)$, gradient of the output $\mathcal{G}_{out}$ and indices of each point's neighborhood for lookup operation $\mathcal{L}$ $(N \times K)$

Result: Gradient of the input $\mathcal{G}_{in}$ and gradient of the weights $\mathcal{G}_w$

begin

  /* Gradient Max Pooling $(N \times R^{D'} \rightarrow N \times R^{D'} \times (K+1))$ */

  $\mathcal{G}_{out}^{mp} \longleftarrow$ **BackwardMaxPooling**$(\mathcal{G}_{out})$

  /* Flattening features $(N \times RD' \times (K+1) \rightarrow N \times R^{D'})$ */

  $\mathcal{G}_{out}^{fl} \longleftarrow$ **InverseIndexLookup**$(\mathcal{G}_{out}^{mp}, \mathcal{L})$

  /* Gradient wrt. input/weight */

  $\mathcal{G}_w, \mathcal{G}_{in} \longleftarrow$ **BackwardSLP**$(\mathcal{T}_f, \mathcal{T}_s, \mathcal{G}_{out}^{fl})$

  FreeMemory$(\mathcal{T}_f, \mathcal{G}_{out}, \mathcal{G}_{out}^{mp}, \mathcal{G}_{out}^{fl})$

  return $(\mathcal{G}_{in}, \mathcal{G}_w)$

end

in Algorithm 2, the backward pass through the SLP can be made more efficient by flattening the tensors after the max-pooling layer. In our unoptimized code, our convolution-type architecture shortens the time spent for backward pass by 68% on average.

In particular for a network with L layers, the memory consumption of the baseline PointNet++ layer grows as $L \times (N \times D \times K)$, while in our case memory consumption grows as $L \times (N \times D) + (N \times D \times K)$, resulting in a K-fold drop as L grows larger. This opens up the possibility of learning much deeper networks, since memory demands grow much more slowly in depth. The memory footprint of our convolution type architecture is on average 67% lower than PointNet++ baseline. Doubling the number of layers comes with a very small memory overhead lower than 3.6%, depending on the dataset.

3.4 Improved Information Flow

Having described our contributions in decreasing the memory footprint of point processing networks, we now turn to improving their information flow.

Firstly, we use the standard Residual Network architecture [13], which helps to train deep networks reliably. Residual networks change the network's connectivity to improve the flow of gradient information during training: identity connections provide early network layers with ac-

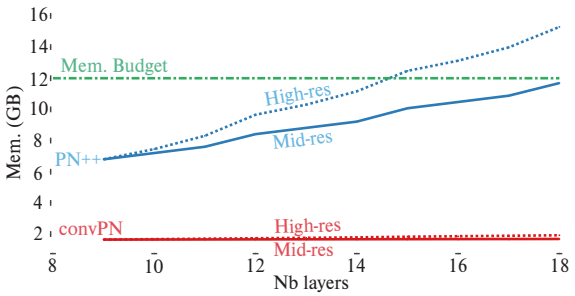

Figure 5: Evolution of memory consumption as the number of layer increases for PointNet++ and convPN (convolution block counterpart) on ShapeNet-Part. As PointNet++ processes the input point cloud at two intermediate resolutions before extracting a global feature, we evaluate the impact of adding layers either at high- or mid- resolution. Our blocks not only decrease the fixed cost to replicate initial architectures but are bolstered by an extremely low marginal cost of depth. Doubling the number of layers results only in an increase of memory by +2.3% and +16.8% for mid- and high- resolution respectively, to compare with +72% and +125% for PointNet++.

cess to undistorted versions of the loss gradient - effectively mitigating the vanishing gradients problem. As our results in Sec. 4 show, this improves optimization during training for deeper networks.

We further introduce Cross-Resolution Links in order to more effectively use information during training. We draw inspiration from the Multi-Grid Networks [16] and the Multiresolution Tree Networks [7] and allow layers that reside in different resolution branches to communicate with each other, thereby exchanging low- mid- and high-resolution information throughout the network processing, rather than fusing multi-resolution information at the end of each block.

Cross-links broadcast information across resolutions as shown in Fig. 3: unlike [7], an MLP transforms the output of one branch to the output dimensionality that can be combined with the output of another. Each resolution can focus on its own representation and the MLP will be in charge of making the translation between them. Taking in particular the case of a high-resolution branch communicating its outputs to a mid-resolution branch, we have $N \times D^H$ feature vectors at the output of a convolution/max-pooling block cascade, which need to be communicated to the $N/2 \times D^M$ vectors of the mid-resolution branch. We first downsample the points, going from N to $N/2$ points, and then use an MLP that transforms the vectors to the target dimensionality. Conversely, when going from low- to higher dimensions we first transform the points to the right dimensionality and then upsample them. We have experimented with both concatenating and summing multi-resolution features, and have observed that summation behaves systematically better in terms of both training speed and test performance.

4 Evaluation

4.1 Dataset and Evaluation measures

We evaluate our network on the point cloud segmentation task on three different datasets. The datasets consist of either 3D CAD models or real-world scans. We quantify complexity of each dataset based on (i) the number of training samples, (ii) the homogeneity of the samples and (iii) the granularity of the segmentation task. Note that a network trained on a bigger and diverse dataset would be less prone to overfitting. We order the datasets by increasing complexity:

- ShapeNet-Part [3]: CAD models of 16 different object categories composed of 50 labeled parts. The dataset provides 13998 samples for training and 2874 samples for testing. Point segmentation performance is assessed using the mean points Intersection over Union (mIoU).

- ScanNet [5]: Scans of real 3D scenes (scanned and reconstructed indoor scenes) composed of 21 semantic parts. The dataset provides 1201 samples for training and 312 samples for testing. We followed the same protocol as in [25] for evaluation and report both the per-voxel accuracy and the part Intersection over Union (pIoU).

- PartNet [24]: Large collection of CAD models of 17 object categories composed of 251 labeled parts. The dataset provides 17119 samples for training, 2492

for validation and 4895 for testing. The dataset provides a benchmark for three different tasks: fine-grained semantic segmentation, hierarchical semantic segmentation and instance segmentation. We report on the first task to evaluate the networks on a more challenging segmentation task using the same part Intersection over Union (pIoU) as in ScanNet.

In order to stay consistent with reported benchmarks on each dataset, we use two different metrics to report IoU:

- mIoU: To get the per sample mean-IoU, the IoU is first computed for each part belonging to the given object category, whether or not the part is in the object. Then, those values are averaged across the parts. If a part is neither predicted nor in the ground truth, the IoU of the part is set to 1 to avoid this indefinite form. The mIoU obtained for each sample is then averaged to get the final score as,

$$\text{mIoU} = \frac{1}{n_{\text{samples}}} \sum_{s \in \text{samples}} \frac{1}{n_{\text{parts}}^{\text{cat(s)}}} \sum_{p^i \in \mathcal{P}_{\text{cat(s)}}} \text{IoU}_s(p^i)$$

  with n_{samples} the number of samples in the dataset, cat(s), $n_{\text{parts}}^{\text{cat(s)}}$ and $\mathcal{P}_{\text{cat(s)}}$ the object category where s belongs, the number of parts in this category and the sets of its parts respectively. $\text{IoU}_s(p^i)$ is the IoU of part p^i in s.

- pIoU: The part-IoU is computed differently. The IoU is first computed over the whole object category for each part and then, the values obtained are averaged across the parts as,

$$\text{pIoU} = \frac{1}{n_{\text{parts}}} \sum_{p \in \text{parts}} \frac{\sum_{s \in \text{samples}} \text{I}_s(p^i)}{\sum_{s \in \text{samples}} \text{U}_s(p^i)}$$

  with n_{parts} the number of parts in the dataset, $\text{I}_s(p^i)$ and $\text{U}_s(p^i)$ the intersection and union for samples s on part p^i respectively.

4.2 Implementation details

In all our experiments, we process the dataset to have the same number of points N for each sample. To reach a given number of points, input pointcloud is downsampled using the furthest point sampling (FPS) algorithm or randomly upsampled.

Inside our architectures, every downsampling module is itself based on FPS to decrease the resolution of the input point cloud. To get back to the original resolution, upsampling layers proceed to linear interpolation in the spatial space using the K_u closest neighbours. To generate multiple resolutions of the same input point cloud, a downsampling ratio of 2 is used for every additional resolution.

We keep the exact same parameters as the original networks regarding most of other parameters (see Appendix for details).

To regularize the network, we interleave a dropout layer between the last fully connected layers and parameterize it to zero 70% of the input neurons. Finally, we add a weight decay of 5e-4 to the loss for all our experiments.

All networks are trained using the Adam optimizer to minimize the cross-entropy loss. The running average coefficients for Adam are set to 0.9 and 0.999 for the gradient and its square, respectively.

4.3 Comparison

We report the performance of our variations for PointNet++ on the Shapenet-Part (Table 1), ScanNet (Table 2) and PartNet (Table 3) datasets. Our lean and deep architectures can indeed be easily deployed on large and complex datasets. Hence, for PartNet, we choose to train on the full dataset all at once on a segmentation task across 251 part labels instead of having to train a separate network for each category in contrast to the original paper [24].

To take into account the randomness of point cloud sampling when performing the coarsening (as detailed in the Appendix), we use the average of the 'N' predictions to decide on the final segmentation during evaluation.

We observe that our architectures substantially improve the memory efficiency of PointNet++ benchmark while also delivering an increase in performance for more complex dataset (see Figure 1). Note that, as the data complexity grows, the spread in performance increases between the different networks. Despite a good performance of our multiresolution approach (wrt. PointNet++ baseline), our convolution-type network starts to clearly outperform other architectures when dataset complexity in-

creases (+3% on ScanNet and +2.6% on PartNet). Finally, the deep counterpart of convPN takes naturally the lead over shallow architectures (+3.4% over PointNet++ baseline on PartNet) on the most challenging dataset.

4.4 Ablation study

In this section, we report extensive experiments to assess the importance of each component of our network architectures. Indeed, our lean structure allows us to adjust the network architectures by increasing its complexity, either by (i) adding extra connections or by (ii) increasing the depth. We analyze our networks along four axes: (i) the performance measured in IoU, (ii) the memory footprint, (iii) the inference time and (iv) the length of a backward pass.

Our main experimental findings concerning network efficiency are reported in Table 4 and ablate the impact of our proposed design choices for point processing networks.

Multi-Resolution: Processing different resolutions at the same stage of a network has been shown to perform well in shallow networks. Indeed, mixing information at different resolutions helps to capture complex features early in the network. We adopt that approach to design our mRes architecture. Switching from a PointNet++ architecture to a multi-resolution setting increases the IoU by 1% on ShapeNet-Part and 2% on PartNet. More crucially, this increase in performance come with more efficiency. Although the inference time is longer (18% longer on average) due to the extra downsampling and upsampling operations, the architecture is much leaner and reduces by 58% the memory footprint. However, the training time is quicker because of a 62% faster backward pass.

Cross-links: Information streams at different resolutions are processed separately and can be seen as complementary. To leverage this synergy, the network is provided with additional links connecting neighborhood resolutions. We experiment on the impact of those cross-resolution links to check their impact on the optimization. At the price of a small impact on memory efficiency (+8% wrt. mRes) and speed (+7% on inference time wrt. mRes), the performance can be improved on the most complex dataset with these extra-links by 0.3% (on PartNet).

Memory-efficient Convolutions: As described in Sec. 3.3, our leanest architeture is equivalent to constraining each PointNet unit to be composed of a single layer network, and turning its operation into a memory-efficient block by removing intermediate activations from memory. In order to get a network of similar size, multiple such units are stacked to reach the same number of layers as the original PointNet++ network. Our convolution-type network win on all counts, both on performance and efficiency. Indeed, the IoU is increased by 3% on ScanNet and 2.6% on PartNet compared to PointNet++ baseline. Concerning its efficiency, the memory footprint is decrease by 67% on average while decreasing both inference time (-41%) and the length of the backward pass (-68%). These improvements in speed can be seen as the consequence of processing most computations on flatten tensors and thus reducing drastically the complexity compared to PointNet++ baseline.

4.5 Going deeper

The aforementioned memory savings have given the opportunity to design deeper networks. Increasing naively the depth of the network results in a drastic drop in the performance of the network. Instead, we use residual connections to prevent any bad behavior of our deep network. The exact design of this architecture is more thoroughly detailed in the Appendix but consists in doubling the number of layers in the encoding part. While keeping the impact on efficiency very small (+6.3% on inference time on average and +3.6% on memory consumption at most compared to the shallow convPN), the performance is improved by a significant margin. On PartNet, this margin reaches 0.8% over the shallow PointNet++. This extremely low cost of depth (Fig. 5) lead us to believe that such architecture would help researchers to solve the next generation datasets using very deep architectures.

4.6 Evaluation on more architectures

We introduce building blocks for point processing networks based on two key ideas, (i) a multi-resolution approach and (ii) a memory efficient convolution. Our blocks make it really efficient to capture, process and diffuse information in a point neighbourhood. This is the main behavior that most networks, if not all, share. We experiment the generality of the proposed modular blocks

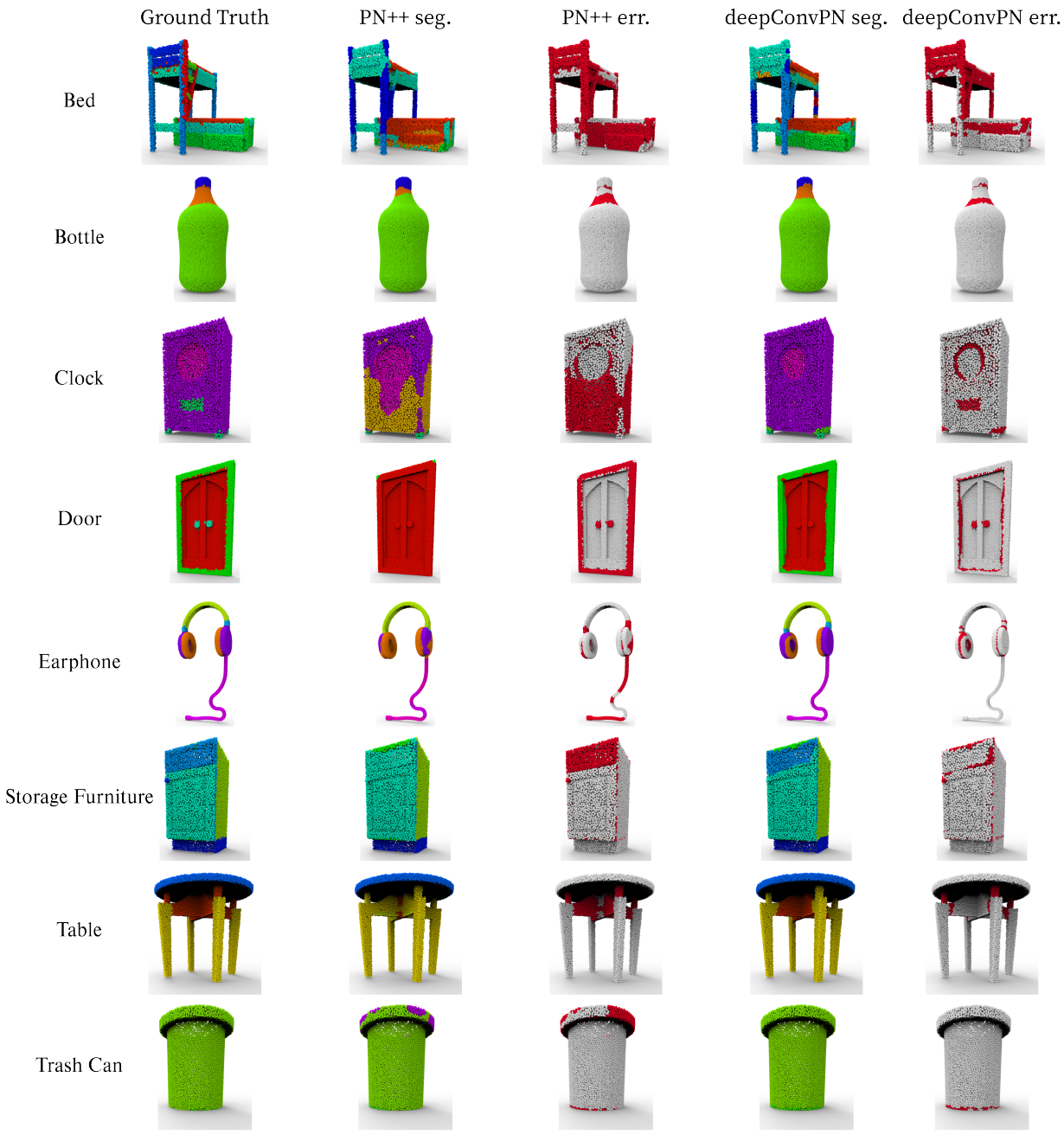

Figure 6: Segmentation results on PartNet on the 8 object categories. PointNet++ misses to segment parts on flat areas with small relief as the frame of the door which is better captured by our architectures.

Table 1: Performance on ShapeNet-Part. The table reports the mIoU performance based on a training on the whole dataset at once. Although the number of samples in the dataset is quite high, learning the segmentation on Shapenet-Part does not necessarily need deep networks because of the simplicity of the shapes and the low number of object parts. All of our network architectures outperform PointNet++ baseline by at least 0.9%. Our deep architecture still improve the performance of its shallower counterpart by a small margin of 0.2%.

	Tot./Av.	Aero	Bag	Cap	Car	Chair	Ear	Guitar	Knife	Lamp	Laptop	Motor	Mug	Pistol	Rocket	Skate	Table
No. Samples	13998	2349	62	44	740	3053	55	628	312	1261	367	151	146	234	54	121	4421
PN++	82.9	80.8	76.8	84.4	77.7	88.9	70.1	90.5	86.3	76.2	**96.0**	70.9	94.3	80.5	**62.8**	76.3	80.2
mRes	83.9	**81.9**	77.5	85.7	78.8	89.5	73.4	**91.6**	**88.2**	78.4	95.6	**73.9**	95.2	81.4	59.7	76.3	81.1
mResX	83.8	81.5	76.7	85.2	78.5	89.4	67.0	91.5	87.9	78.1	95.8	73.3	**95.4**	82.4	57.1	77.1	81.3
convPN	83.9	81.6	76.6	**87.2**	79.2	**89.7**	71.9	90.6	**88.2**	78.1	95.7	73.8	**95.4**	**83.2**	60.9	76.4	80.9
deepConvPN	**84.1**	81.1	**79.8**	81.6	**79.8**	89.5	**75.1**	**91.6**	88.1	**79.0**	95.4	71.9	95.3	**83.2**	61.9	**77.2**	**81.5**

Table 2: Performance Voxel accuracy and part IoU on ScanNet. Although ScanNet is a real-world dataset, the low number of samples in the dataset makes it easy for networks to overfit. Our convolutional networks (convPN and deepConvPN) still improve the benchmark by a significant +3% spread on pIoU

	Acc.	Part IoU
PN++	79	24
mRes	76	22
mResX	76	22
convPN	**81**	**27**
deepConvPN	80	**27**

in the context of other state-of-the-art point-based learning setups, as shown in Table 5. Each macro-block can be stacked together, extended into a deeper block by duplicating the green boxes (see Figure 3) or even be modified by changing one of its component by another. We experiment on three additional networks among the latest state-of-the-art approaches, (i) Dynamic Graph CNN [37], (ii) SpiderCNN [39] and (iii) PointCNN [21].

All three of the methods make extensive use of memory which is a bottleneck to depth. We implant our modules directly in the original networks, making, when needed, some approximations from the initial architecture. We report the performance of each network with our lean counterpart on three metrics: (i) IoU, (ii) memory footprint and (iii) inference time in Table 5. Our lean counterparts consistently improve both the IoU (from +1.0% up to +2.1%) and the memory consumption (from -27% up to -63%) with a low impact on inference time.

5 Conclusion

In this work we have introduced new generic building blocks for point processing networks, that exhibit extremely favorable memory, computation, and optimization properties when compared to the current counterparts of state-of-the-art point processing networks. When based on PointNet++, our lean architecture convPN wins on all counts, memory efficiency (-67% wrt. PointNet++) and speed (-41% and -68% on inference time and length of backward pass). Its deep counterpart has a very marginal cost in terms of efficiency and achieves the best IoU on PartNet (+3.4% over PointNet++). Those generic and modular blocks exhibit similar performance on all of the additional tested architectures with a significant decrease in memory (up to -63%) and increase in IoU (up to +2.1%). From the promising results on PartNet and the extremely low cost of depth in our architectures, we anticipate that adding these components to the armament of the deep geometry processing community will allow researchers to train the next generation of point processing networks by leveraging upon the advent of larger shape datasets [24, 17].

References

[1] Spatial mapping in the primate sensory projection: Analytic structure and relevance to perception. *Biological Cybernetics*, 1977. 5

[2] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niener, M. Savva, S. Song, A. Zeng, and Y. Zhang. Matterport3d: Learning from rgb-d data in indoor environments. 09 2017. 1

[3] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su,

Table 3: Performance on PartNet. The table reports the Part IoU performance based on a training on the whole dataset at once in contrast with [24]. The fine detail of the segmentation and the high number of points to process make the training much more complex than any former datasets. PointNet++, here, fails to capture enough features to segment objects properly. Our different architectures outperform PointNet++ with a spread of at least 2%. With this more complex dataset, deeper networks become significantly better: our deepConvPN network achieves to increase pIoU by 3.4% over PointNet++ baseline, outperforming its shallow counterpart by 0.8%.

	Tot./Av.	Bed	Bott	Chair	Clock	Dish	Disp	Door	Ear	Fauc	Knife	Lamp	Micro	Frid	Storage	Table	Trash	Vase
No. samples	17119	133	315	4489	406	111	633	149	147	435	221	1554	133	136	1588	5707	221	741
PN++	35.2	30.1	32.0	39.5	30.3	29.1	81.4	31.4	35.4	46.6	37.1	25.1	31.5	32.6	40.5	34.9	33.0	56.3
mRes	37.2	29.6	32.7	40.0	34.3	29.9	80.2	**35.0**	**50.0**	**56.5**	41.0	26.5	**33.9**	**35.1**	41.0	35.4	35.3	57.7
mResX	37.5	32.0	37.9	40.4	30.2	31.8	80.9	34.0	43.0	54.3	42.6	26.8	33.1	31.8	**41.2**	**36.5**	40.8	57.2
convPN	37.8	**33.2**	40.7	40.8	**35.8**	31.9	81.2	33.6	48.4	54.3	41.8	26.8	31.0	32.2	40.6	35.4	41.1	57.2
deepConvPN	**38.6**	29.5	**42.1**	**41.8**	34.7	**33.2**	**81.6**	34.8	49.6	53.0	**44.8**	**28.4**	33.5	32.3	41.1	36.3	**43.1**	**57.8**

Table 4: Efficiency of our network architectures measured with a batch size of 8 samples on a Nvidia GTX 2080Ti GPU. All of our lean architectures allow to save a substantial amount of memory on GPU wrt. the PointNet++ baseline from 58% with mRes to a 67% decrease with convPN. This latter convolution-type architecture wins on all counts, decreasing both inference time (-41%) and the length of backward pass (-68%) by a large spread. Starting form this architecture, the marginal cost of going deep is extremely low: doubling the number of layers in the encoding part of the network increases inference time by 6.3% on average and the memory consumption by only 3.6% at most compared to convPN)

	Parameters (M)			Memory Footprint (Gb)			Inference Time (ms)			Length Backward pass (ms)		
	ShapeNet-Part	ScanNet	PartNet	ShapeNet-Part	ScanNet	PartNet	ShapeNet-Part	ScanNet	PartNet	ShapeNet-Part	ScanNet	PartNet
PointNet++	1.88	1.87	1.99	6.80	6.73	7.69	344	238	666	173	26	185
mRes	1.56	1.54	1.66	2.09	2.93	4.03	395	379	537	54	12	68
mResX	1.68	1.67	1.79	2.38	3.15	4.13	441	383	583	122	26	138
convPN	2.14	2.12	2.24	**1.65**	**2.25**	**3.24**	**187**	**166**	**347**	**30**	**15**	**39**
deepConvPN	**2.90**	**2.88**	**3.00**	1.42	2.33	3.31	205	177	356	37	23	51

Table 5: Performance of our blocks on four different architectures based on ShapeNet-Part using three different metrics: IoU, memory consumption and inference time. Our lean counterparts improve significantly both the IoU (up to +2.1%) and the memory consumption (up to -63%) with a low impact on inference time.

		IoU (%)	Mem. (GB)	Inf. (ms)
PN++	Vanilla	82.9	6.8	345
	Lean	83.9 **+1.2%**	1.84 **-73%**	273 **-21%**
DGCNN	Vanilla	79.8	2.62	42
	Lean	81.3 **+1.9%**	0.97 **-63%**	32 **-24%**
SpiderCNN	Vanilla	78.0	1.08	22
	Lean	79.6 **+2.1%**	0.79 **-27%**	31 **+41%**
PointCNN	Vanilla	82.0	4.54	175
	Lean	82.8 **+1.0%**	1.97 **-57%**	246 **+41%**

J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 7

[4] T. Chen, B. Xu, C. Zhang, and C. Guestrin. Training deep nets with sublinear memory cost. *CoRR*, abs/1604.06174, 2016. 3

[5] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017. 1, 2, 7

[6] M. Dominguez, F. P. Such, S. Sah, and R. Ptucha. Towards 3d convolutional neural networks with meshes. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3929–3933, 2017. 2

[7] M. Gadelha, R. Wang, and S. Maji. Multiresolution tree networks for 3d point cloud processing. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 103–118, 2018. 7

[8] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse. The reversible residual network: Backpropagation without storing activations. *CoRR*, abs/1707.04585, 2017. 3

[9] A. Gruslys, R. Munos, I. Danihelka, M. Lanctot, and A. Graves. Memory-efficient backpropagation through time. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4125–4133. Curran Associates, Inc., 2016. 3

[10] P. Guerrero, Y. Kleiman, M. Ovsjanikov, and N. J. Mitra. PCPNet: Learning local shape properties from raw point clouds. *CGF*, 37(2):75–85, 2018. 3

[11] K. Guo, D. Zou, and X. Chen. 3d mesh labeling via deep convolutional neural networks. *ACM Trans. Graph.*, 35(1):3:1–3:12, Dec. 2015. 2

[12] X. Han, Z. Li, H. Huang, E. Kalogerakis, and Y. Yu. High-resolution shape completion using deep neural networks for global structure and local geometry inference. pages 85–93, 2017. 2

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. 2016. 2, 3, 6

[14] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. 2

[15] Y. Jia. *Learning Semantic Image Representations at a Large Scale*. PhD thesis, University of California, Berkeley, USA, 2014. 6

[16] T. Ke, M. Maire, and S. X. Yu. Neural multigrid. *CoRR*, abs/1611.07661, 2016. 7

[17] S. Koch, A. Matveev, Z. Jiang, F. Williams, A. Artemov, E. Burnaev, M. Alexa, D. Zorin, and D. Panozzo. ABC: A big CAD model dataset for geometric deep learning. *CoRR*, abs/1812.06216, 2018. 3, 11

[18] I. Kokkinos and A. Yuille. Scale invariance without scale selection. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008. 5

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2013. 3

[20] J. Li, B. M. Chen, and G. Hee Lee. So-net: Self-organizing network for point cloud analysis. pages 9397–9406, 2018. 3

[21] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. Pointcnn: Convolution on x-transformed points. 2018. 3, 11, 14

[22] C. Liu, L. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. *CoRR*, abs/1901.02985, 2019. 2, 3

[23] H. Liu, K. Simonyan, and Y. Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018. 2, 3

[24] K. Mo, S. Zhu, A. X. Chang, L. Yi, S. Tripathi, L. J. Guibas, and H. Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. *CoRR*, abs/1812.02713, 2018. 3, 7, 8, 11, 12

[25] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CVPR*, 1(2):4, 2017. 2, 3, 5, 7

[26] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, pages 5099–5108, 2017. 1, 2, 3, 5, 14

[27] A. Ranjan, T. Bolkart, S. Sanyal, and M. J. Black. Generating 3d faces using convolutional mesh autoencoders. *CoRR*, abs/1807.10267, 2018. 2

[28] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018. 2

[29] A. Sharma, O. Grau, and M. Fritz. Vconv-dae: Deep volumetric shape learning without object labels. pages 236–250, 2016. 2

[30] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. 2015. 3

[31] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M.-H. Yang, and J. Kautz. Splatnet: Sparse lattice networks for point cloud processing. pages 2530–2539, 2018. 3

[32] H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. ICCV*, 2015. 2

[33] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. 3

[34] E. Tola, V. Lepetit, and P. Fua. Daisy: An efficient dense descriptor applied to wide-baseline stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(5):815–830, 2010. 5

[35] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong. O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis. *ACM Transactions on Graphics (SIGGRAPH)*, 36(4), 2017. 2

[36] P.-S. Wang, C.-Y. Sun, Y. Liu, and X. Tong. Adaptive O-CNN: A Patch-based Deep Representation of 3D Shapes. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 37(6), 2018. 2

[37] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018. 1, 3, 11, 14

[38] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016. 3

[39] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 87–102, 2018. 3, 11, 14

[40] K. Yin, H. Huang, D. Cohen-Or, and H. Zhang. P2p-net: Bidirectional point displacement net for shape transform. *ACM TOG*, 37(4):152:1–152:13, July 2018. 3

[41] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng. Ec-net: an edge-aware point set consolidation network. pages 386–402, 2018. 3

[42] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng. Pu-net: Point cloud upsampling network. In *CVPR*, 2018. 3

[43] S. Zagoruyko and N. Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016. 2, 3

[44] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. 2017. 3

[45] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. 2014. 3

In the following sections, we provide more details about how we design our lean architectures to ensure reproducible results for all tested architectures, (i) PointNet++ [26], (ii) Dynamic Graph CNN [37], (iii) SpiderCNN [39] and (iv) PointCNN [21]. Please refer to the code for further explanations.

A PointNet++ based architectures

To keep things simple and concise in this section, we adopt the following notations:

- S(n): Sampling layer of n points;
- rNN(r): query-ball of radius r;
- MaxP: Max Pooling along the neighbourhood axis;
- $\bigoplus$: Multi-resolution combination;
- Lin(s): Linear unit of s neurons;
- Drop(p): Dropout layer with a probability p to zero a neuron

A.1 PointNet++

In all our experiments, we choose to report the performance of the multi-scale PointNet++ (MSG PN++) as it is reported to beat its alternative versions in the original paper on all tasks. We implement our own implementation of PointNet++ in Pytorch and choose the same parameters as in the original code.

For segmentation task, the architecture is designed as follow:

Encoding1:

$$\text{S}(512)\rightarrow \begin{bmatrix} \text{rNN}(.1) \rightarrow \text{mLP}([32, 32, 64]) \rightarrow \text{MaxP} \\ \text{rNN}(.2) \rightarrow \text{mLP}([64, 64, 128]) \rightarrow \text{MaxP} \\ \text{rNN}(.4) \rightarrow \text{mLP}([64, 96, 128]) \rightarrow \text{MaxP} \end{bmatrix} \bigoplus$$

Encoding2:

$$\text{S}(128)\rightarrow \begin{bmatrix} \text{rNN}(.2) \rightarrow \text{mLP}([64, 64, 128]) \rightarrow \text{MaxP} \\ \text{rNN}(.4) \rightarrow \text{mLP}([128, 128, 256]) \rightarrow \text{MaxP} \\ \text{rNN}(.8) \rightarrow \text{mLP}([128, 128, 256]) \rightarrow \text{MaxP} \end{bmatrix} \bigoplus$$

Encoding3:

$\text{S}(1)\rightarrow \text{mLP}([256, 512, 1024]) \rightarrow \text{MaxP}$

Decoding1: $\text{Interp}(3)\rightarrow \text{mLP}([256, 256])$

Decoding2: $\text{Interp}(3)\rightarrow \text{mLP}([256, 128])$

Decoding3: $\text{Interp}(3)\rightarrow \text{mLP}([128, 128])$

Classification: $\text{Lin}(512)\rightarrow \text{Drop}(.7)\rightarrow \text{Lin}(\text{nb}_{\text{classes}})$

We omit here skiplinks for sake of clarity: they connect encoding and decoding modules at the same scale level.

A.2 mRes

The mRes architecture consists in changing the way the sampling is done in the network. We provide the details for the encoding part of the network as we keep the decoding part unchanged from PointNet++.

Encoding1:

$$\begin{bmatrix} S(512) \rightarrow \text{rNN}(.1) \rightarrow \text{mLP}([32, 32, 64]) \rightarrow \text{MaxP} \\ S(256) \rightarrow \text{rNN}(.2) \rightarrow \text{mLP}([64, 64, 128]) \rightarrow \text{MaxP} \\ S(128) \rightarrow \text{rNN}(.4) \rightarrow \text{mLP}([64, 96, 128]) \rightarrow \text{MaxP} \end{bmatrix} \bigoplus$$

Encoding2:

$$\begin{bmatrix} S(128) \rightarrow \text{rNN}(.2) \rightarrow \text{mLP}([64, 64, 128]) \rightarrow \text{MaxP} \\ S(96) \rightarrow \text{rNN}(.4) \rightarrow \text{mLP}([128, 128, 256]) \rightarrow \text{MaxP} \\ S(64) \rightarrow \text{rNN}(.8) \rightarrow \text{mLP}([128, 128, 256] \rightarrow \text{MaxP} \end{bmatrix} \bigoplus$$

Encoding3:

$\text{S}(1)\rightarrow \text{mLP}([256, 512, 1024]) \rightarrow \text{MaxP}$

Starting from this architecture, we add Xlinks connection between each layer of mLPs to get our mResX archi-

tecture. A Xlink connection connects two neighbouring resolutions to merge information at different granularity. On each link, we use a sampling module (either downsampling or upsampling) to match the input to the target resolution. We use two alternatives for feature combination: (a) concatenation, (b) summation. In the later case, we add an additional sLP on each Xlink to map the input feature dimension to the target. To keep this process as lean as possible, we position the SLP at the coarser resolution, i.e. before the upsampling module or after the downsampling module.

A.3 convPN

To simplify the writing, we adopt the additional notations:

- *Sampling* block $S([s_1, s_2, .., s_n]^T)$ where we make a sampling of s_i points on each resolution i. When only one resolution is available as input, the block $S([., s_1, s_2, ..., s_{n-1}]^T)$ will sequentially downsample the input point cloud by s_1, s_2, .. points to create the desired number of resolutions.

- *Convolution* block $C([r_1, r_2, ..., r_n]^T)$ is composed itself of three operations for each resolution i: neighborhood lookup to select the r_iNN for each points, an sLP layer of the same size as input and a max-pooling.

- *Transition* block $T([t_1, t_2, ..., t_n]^T)$ whose main role is to change the channel dimension of the input to the convolution block. An sLP of ouput dimension t_i will be apply to the resolution i.

Residual connections are noted as *.
Encoding1:

$$S\begin{bmatrix}.\\512\\256\end{bmatrix} \rightarrow T\begin{bmatrix}32\\64\\64\end{bmatrix} \rightarrow C^*\begin{bmatrix}.1\\.2\\.4\end{bmatrix} \rightarrow T\begin{bmatrix}32\\64\\96\end{bmatrix} \rightarrow C^*\begin{bmatrix}.1\\.2\\.4\end{bmatrix} \rightarrow T\begin{bmatrix}64\\128\\128\end{bmatrix} \rightarrow C^*\begin{bmatrix}.1\\.2\\.4\end{bmatrix} \rightarrow S\begin{bmatrix}512\\256\\128\end{bmatrix} \rightarrow \bigoplus$$

Encoding2:

$$S\begin{bmatrix}.\\128\\96\end{bmatrix} \rightarrow T\begin{bmatrix}64\\128\\128\end{bmatrix} \rightarrow C^*\begin{bmatrix}.2\\.4\\.8\end{bmatrix} \rightarrow C^*\begin{bmatrix}.2\\.4\\.8\end{bmatrix} \rightarrow T\begin{bmatrix}128\\256\\256\end{bmatrix} \rightarrow C^*\begin{bmatrix}.2\\.4\\.8\end{bmatrix} \rightarrow S\begin{bmatrix}128\\96\\64\end{bmatrix} \rightarrow \bigoplus$$

Encoding3:
S(1)$\rightarrow$ mLP($[256, 512, 1024]$) $\rightarrow$ MaxP

Note here that there is no *Transition* block between the first two C blocks in the Encoding2 part. This is because those two *Convolution* blocks works on the same feature dimension.

We also add Xlinks inside each of the C blocks. In this architecture, the features are combined by summation and the links follow the same design as for mResX.

A.4 deepConvPN

Our deep architecture builds on convPN to design a deeper architecture. For our experiments, we double the size of the encoding part by repeating each convolution block twice. For each encoding segment, we position the sampling block after the third convolution block, so that the first half of the convolution blocks are processing a higher resolution point cloud and the other half a coarsen version.

B DGCNN based architecture

Starting from the authors' exact implementation, we swap each edge-conv layer, implemented as an MLP, by a sequence of single resolution convPN blocks. The set of convPN blocks replicates the succession of SLP used in the original implementation.

To allow the use of residual links, a transition block is placed before each edge-conv layer to match the input dimension of our convPN blocks to their output dimension.

C SpiderCNN based architecture

A SpiderConv block can be seen as a bilinear operator on the input features and a non-linear transformation of the input points. This non-linear transformation consists of changing the point coordinates into a new coordinate system obtained by computing any initial coordinate product of order 3 and less.

In the original architecture, a SLP is first applied to the transformed points to compute the points' Taylor expansion. Then, each output vector is multiplied by its corresponding feature. Finally a convolution is applied on the product. Therefore, the neighbourhood features can be built on-the-fly within the block and deleted once the output is obtained. We thus modify the backward pass to reconstruct the needed tensors for gradient computation.

D PointCNN based architecture

For PointCNN, we modify the χ-conv operator to avoid having to store the neighbourhood features for the backward pass. To do so, we make several approximations from the original architecture.

We replace the first MLP used to lift the points by a sequence of convPN blocks. Thus, instead of learning a feature representation per neighbour, we retain only a global feature vector per representative point.

We change as well the first fully connected layer used to learn the χ-transformation matrix. This new layer now reconstructs the neighbourhood features on-the-fly from its inputs and deletes it from memory as soon as its output is computed. During the backward pass, the neighbourhood features tensor is easily rebuilt to get the required gradients.

We implement the same trick for the convolution operator applied to the transformed features. We further augment this layer with the task of applying the χ-transformation to the neighbourhood features once grouped.

Finally, we place transition blocks between each χ-conv operation to enable residual links.