GOING DEEPER WITH LEAN POINT NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

In this work we train deeper and more accurate networks by introducing three point processing blocks that improve accuracy and memory consumption: a *convolution-type block* for point sets that blends neighborhood information in a memory-efficient manner; a *multi-resolution point cloud processing block*; and a *crosslink block* that efficiently shares information across low- and high-resolution processing branches. By combining these blocks, we design significantly wider and deeper architectures. We extensively evaluate the proposed architectures on multiple point segmentation benchmarks (ShapeNet-Part, ScanNet, PartNet). We report systematic accuracy and memory consumption improvements by using our generic modules in conjunction with multiple architectures (PointNet++, DGCNN, SpiderNet, PointCNN). All of our code will become publicly available.

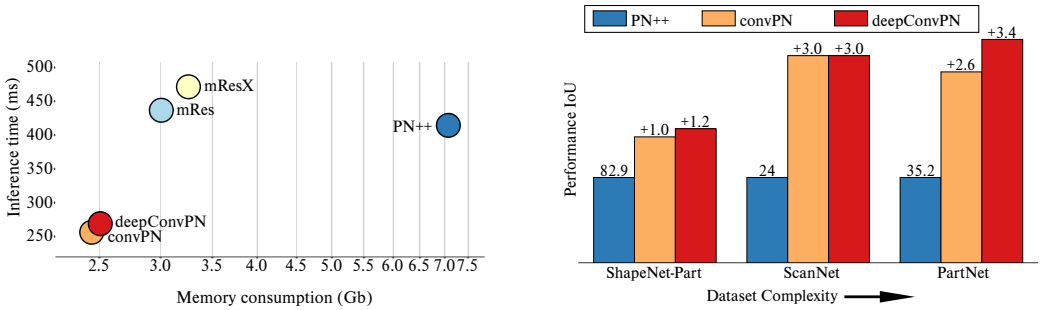

Figure 1: (Left) Memory footprint and inference speed of network variations: our multi-resolution (mRes), and crosslink (X) blocks decrease the memory footprint, while our convolution-type block (conv) decreases both memory consumption (-67%) and inference time (-41%) compared to the PointNet++ (PN++) baseline. (Right) Improvements in accuracy for three segmentation benchmarks of increasing complexity. On the –most complex– PartNet dataset our deep network outperforms the shallow PointNet++ baseline by 3.4%(spread of +0.6), yielding a 9.7% (spread of +3.4) relative increase.

1 INTRODUCTION

Geometry processing has recently started profiting from applying deep learning to graphics and 3D shape analysis (Qi et al., 2017b; Wang et al., 2018b; Dai et al., 2017; Chang et al., 2017) with networks that guarantee desirable properties of point cloud processing, such as permutation-invariance and quantization-free representation (Su et al., 2015; Wang et al., 2017; 2018a). Despite these advances, several differences still impede the breakthroughs made in computer vision.

The different nature of 3D data dictates re-inventing for geometry processing the functionality of basic image processing blocks, such as multi-resolution processing or convolution operations. When operating with unstructured point clouds, one has to resort to elementary local pooling operations that group information within a neighborhood based on Euclidean distance. Exemplary methods such as the PointNet/PointNet++ architectures (Qi et al., 2017a;b) make design choices that potentially compromise performance.

In particular, the computation and memory demands of point network blocks can affect both training speed and, more crucially, inference time. One of the main bottlenecks for point networks is

their memory-intensive nature: 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.

In this work, we enhance point processing networks by introducing a set of modules that improve memory footprint and accuracy, without compromising on inference speed. We call the result architectures Lean Point Networks, to highlight their lightweight memory budget. We build on the decreased memory budget to go deeper with point networks. As has been witnessed repeatedly in the image domain (He et al., 2016; Huang et al., 2016; Zagoruyko & Komodakis, 2016), we show that going deep also increases the prediction accuracy of point networks.

We start in Sec. 3.2 by replacing 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, while exhibiting favorable training properties due to more effective mixing of information across neighborhoods.

We then turn in Sec. 3.3 to improving the information flow across layers and scales within point networks through three techniques: a *multi-resolution* variant for multi-scale network which still delivers the multi-scale context but at a reduced memory and computational cost, *residual links*, and a new *cross-link block* that broadcasts multi-scale information across the network branches. By combining these advances we are able to successfully train deeper point networks that allow us to leverage upon larger, recently introduced datasets.

In Sec. 4 we thoroughly validate our contributions on the ShapeNet-Part, ScanNet and PartNet segmentation benchmarks, reporting systematic improvements over the PointNet++ baseline. As shown in Fig. 1, 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 PartNet, the most complex dataset, our deep architecture achieves a 9.7% relative increase in IoU while decreasing memory footprint by 57% and inference time by 47%.

Having thoroughly ablated our design choices on the PartNet++ baseline, in Sec. 4.3 we turn to confirming the generic nature of our blocks. We extend the scope of our experiments to three additional networks, (i) DGCNN (Wang et al., 2018b), (ii) SpiderCNN (Xu et al., 2018) and (iii) PointCNN (Li et al., 2018b) and report systematic improvements in memory efficiency and performance.

2 Related Work

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 (Qi et al., 2017a) and several extensions such as PointNet++ (Qi et al., 2017b) and Dynamic Graph CNNs (Wang et al., 2018b) for shape segmentation and classification, PCPNet (Guerrero et al., 2018) for normal and curvature estimation, P2P-Net (Yin et al., 2018) and PU-Net (Yu et al., 2018b) for cross-domain point cloud transformation. Although many alternatives to PointNet have been proposed (Su et al., 2018; Li et al., 2018a;b; Hermosilla et al., 2018; Zaheer et al., 2017) to achieve higher performance, the simplicity and effectiveness of PointNet and its extension PointNet++ make it popular for many other tasks (Yu et al., 2018a).

Taking PointNet++ as our starting point, our work facilitates the transfer of network design techniques developed in computer vision to point cloud processing. In particular, significant accuracy improvements have been obtained with respect to the original AlexNet network (Krizhevsky et al., 2013) by engineering the scale of the filtering operations (Zeiler & Fergus, 2014; Simonyan & Zisserman, 2015), the structure of the computational blocks (Szegedy et al., 2014; Xie et al., 2016), and the network's width and depth (He et al., 2016; Zagoruyko & Komodakis, 2016). A catalyst for experimenting with a larger space of network architecture, however, is the reduction of memory consumption - this motivated us to design lean alternatives to point processing networks. Notably, Zhang et al. (2019) introduce a new operator to improve point cloud network efficiency, but only focus on increasing the convergence speed by tuning the receptive field. Li et al. (2019) has investigated how residual/dense connections and dilated convolution could help mitigate vanishing gradient observed for deep graph convolution networks but without solving memory limitations. By

contrast our work explicitly tackles the memory problem with the objective of training deeper/wider networks and shows that there are clear improvements over strong baselines.

Memory-Efficient Networks. The memory complexity of the standard back-propagation implementation grows linearly with network's depth as 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 bypass this problem by trading off speed with memory. Checkpointing techniques (Chen et al., 2016; Gruslys et al., 2016) use anchor points to free up intermediate computation results, and re-compute them in the backward pass. This is 1.5x 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 (Gruslys et al., 2016) but is not as easy for general Directed Acyclic Graphs, such as U-Nets, or multi-scale networks like PointNet++. One needs to manually identify the graph components, making it cumbersome to experiment with diverse variations of architectures.

Reversible Residual Networks (RevNets) (Gomez et al., 2017) limit the computational block to come in a particular, invertible form of residual network. This is also 1.5x slower during training, but alleviates the need for anchor points altogether. Unfortunately, it is unclear what is the point cloud counterpart of invertible blocks.

We propose generic blocks to reduce the memory footprint inspired from multi-resolution processing and efficient implementations of the convolution operation in computer vision. As we show in Sec. 4.3, our blocks can be used as drop-in replacements in generic point processing architectures (PointNet++, DGCNN, SpiderNet, PointCNN) without any additional network design effort.

3 METHOD

This Section introduces a set of modular blocks, shown in Fig. 2, that can be applied to most state-of-the-art point networks. We start with a brief introduction of the PointNet++ network, which serves as an example point network baseline. We then introduce our modules and explain how their design decreases memory footprint and improves information flow.

3.1 POINTNET AND POINTNET++ ARCHITECTURES

PointNet++ (Qi et al., 2017b) builds on top of PointNet. 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 is assigned back to the corresponding point $\mathbf{p}_i$. While demonstrated to be extremely effective, PointNet++ has two main shortcomings: first, because of explictly carrying around k-nearest neighbor information for each point, the network layers are memory intensive; and second, being reliant on PointNet, it also delays transmission of global information until the last stage.

3.2 MEMORY EFFICIENT NEIGHBORHOOD CONVOLUTION

As shown in Fig. 2(a), the existing PointNet++ grouping operation exposes the neighborhood of any point i by concatenating all of its K neighboring D-dimensional vectors $\mathbf{v}_{[i,k]}$ to form a tensor: $T = [\ \mathbf{v} \quad \mathbf{v}_{[.,1]} \quad \dots \quad \mathbf{v}_{[.,K]}\ ]$ 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 point 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.

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 convo-

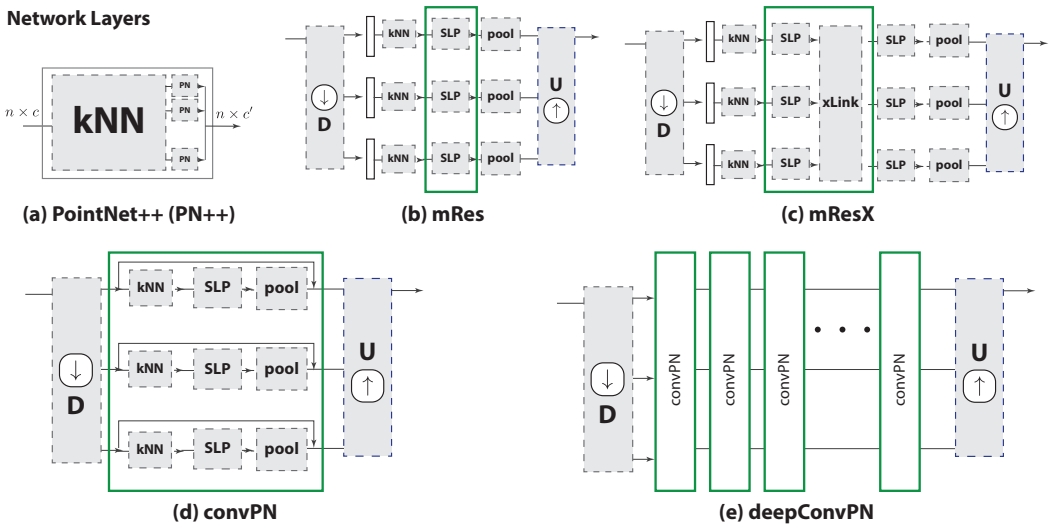

Figure 2: The standard PN++ layer in (a) amounts to the composition of a neighborhood-based lookup and a PointNet element. In (b) we propose to combine parallel PointNet++ blocks in a multi-resolution architecture, and in (c) allow information to flow across branches of different resolutions through 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 stack the block in (d) to grow in depth and build our deep architecture. 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.

lutions with General Matrix-Matrix Multiplication (GEMM) (Jia, 2014). 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.

In particular we free the memory as soon as the forward pass computes its output, rather than maintaining the matrix in memory. 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 refer to Algorithm 1 and Algorithm 2 in Appendix B for a detailed description).

Using the on-the-fly re-computation of the tensor $\mathcal{T}$ has a positive impact on both the forward and backward pass. Instead of applying the SLP on the neighbourhood feature matrix, we can first apply the SLP on the flat feature matrix and then reconstruct the neighbourhood just before the max-pooling layer. The same can be used for the backward pass. In our unoptimized code, our convolution-type architecture shortens the time spent for the forward pass and the backward pass by 41% and 68% respectively on average.

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 $L \times (N \times D) + (N \times D \times K)$, where the term, $L \times (N \times D)$ accounts for the memory required to store the layer activations, while the second term $N \times D \times K$ is the per-layer memory consumption of a single neighborhood convolution layer. As L grows larger, this results in a K-fold drop, shown on Fig. 3. This reduction opens up the possibility of learning much deeper networks, since memory demands now grow substantially more slowly in depth. With minor, dataset-dependent, fluctuations, the memory footprint of our convolution type architecture is on average 67% lower than the PointNet++ baseline, while doubling the number of layers comes with a memory overhead of 2.7%.

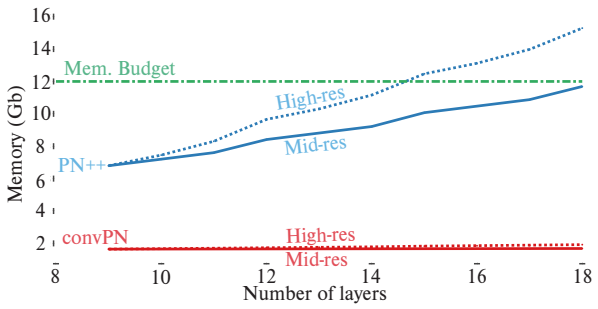

Figure 3: Evolution of memory consumption as the number of layers increases for PointNet++ and convPN (convolution block counterpart) on ShapeNet-Part. Doubling the number of layers for convPN results only in an increase in memory by +2.3% and +16.8% for mid- and high- resolution respectively, which favorably compares to the +72% and +125% increases for PointNet++.

3.3 Improving Information Flow

We now turn to methods that allow for a more efficient propagation of information through the network. As has been repeatedly shown in computer vision, this can drastically impact the behavior of the network during training. Our experiments indicate that this is also the case for point processing. Due to lack of space, the illustrations clarifying the operation of these blocks are only available in the Appendix.

a- 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++ mostly uses a neighborhood of fixed radius, leaving only to the later blocks of the network the use of bigger radii and point set sampling. Hence, at early stages of the network, the points only has access to very local information.

We observe that this 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 (Schwartz, 1977; Tola et al., 2010; Kokkinos & Yuille, 2008) where a 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 point cloud. In the coordinates of the original point cloud this amounts to increasing the effective grouping area, but it now comes with a much smaller memory budget. We observe a 58% decrease in memory footprint on average on the three tested datasets. Please refer to Fig. 4 in Appendix A.1.2 for an illustration of the difference between both types of processing.

b- Residual Links. We use the standard Residual Network architecture (He et al., 2016), which helps to train deep networks reliably. Residual networks change the network's connectivity to improve gradient flow during training: identity connections provide early network layers with access to undistorted versions of the loss gradient, effectively mitigating the vanishing gradient problem. As our results in Sec. 4 show, this allows us to train deeper networks.

c- Cross Links. We further introduce Cross-Resolution Links in order to better propagate information in the network during training. We draw inspiration from the Multi-Grid Networks (Ke et al., 2016) and the Multiresolution Tree Networks (Gadelha et al., 2018) 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. 5 in Appendix A.1: unlike Gadelha et al. (2018), an MLP transforms the output of one branch to the right output dimensionality so that it can be combined with the output of another branch. Each resolution branch can focus on its own representation and the MLPs 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 lookup-SLP-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

Dataset and evaluation measures. We evaluate our modules on the point cloud segmentation task on three different datasets. The datasets consist of either 3D CAD models or real-world scans. We quantify the 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 - as such we can draw more informative conclusions from more complex datasets. We order the datasets by increasing complexity: ShapeNet-Part (Chang et al., 2015), ScanNet (Dai et al., 2017) and PartNet (Mo et al., 2018) for fine-grained segmentation. By its size (24,506 samples) and its granularity (251 labeled parts), PartNet is the most complex dataset we have experimented on. In order to stay consistent with reported benchmarks on each dataset, we use two different metrics to report the Intersection over Union (IoU): (i) the mean Intersection over Union (mIoU) and (ii) the part Intersection over Union (pIoU). Please refer to Appendix C.2 for further explanation about both metrics.

4.1 EFFECT OF IMPROVED INFORMATION FLOW

We report the performance of our variations for PointNet++ on the Shapenet-Part, ScanNet and PartNet datasets (Table 1). Our lean and deep architectures can 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 the 17 classes instead of having to train a separate network for each category in contrast to the original paper (Mo et al., 2018).

Our architectures substantially improve the memory efficiency of the PointNet++ baseline while also delivering an increase in performance for more complex datasets (see Fig. 1). Indeed, as the data complexity grows, having efficient information flow has a larger influence on the network performance. On PartNet, the spread between our architectures and the vanilla PointNet++ becomes significantly high: our multiresolution (mRes) network increases relative performance by +5.7% over PointNet++ and this gain reaches +6.5% with cross-links (mResX). Our convolution-type network (convPN) outperforms other architectures when dataset complexity increases (+12.5% on ScanNet and +7.4% on PartNet) by more efficiently mixing information across neighbours.

Table 1: Performance of our modules compared to PointNet++ baseline. The impact of our modules becomes most prominent as the dataset complexity grows. On PartNet our deepConvPN network increases pIoU by 9.7% over PointNet++, outperforming its shallow counterpart by +2.1%.

	ShapeNet-Part (13,998 samp.)	ScanNet (1,201 samp.)		PartNet (17,119 samp.)
	mIoU (%)	Accuracy (%)	pIoU (%)	pIoU (%)
PN++	82.9 (+0.0%)	79 (+0.0%)	24 (+0.0%)	35.2 (+0.0%)
mRes	83.9 (+1.2%)	76 (-3.8%)	22 (-8.3%)	37.2 (+5.7%)
mResX	83.8 (+1.1%)	76 (-3.8%)	22 (-8.3%)	37.5 (+6.5%)
convPN	83.9 (+1.2%)	**81 (+2.5%)**	**27 (+12.5%)**	37.8 (+7.4%)
deepConvPN	**84.1 (+1.4%)**	80 (+1.3%)	**27 (+12.5%)**	**38.6 (+9.7%)**

4.2 IMPROVEMENT OF ACCURACY BY GOING DEEPER

The aforementioned memory savings give the opportunity to design significantly deeper networks. Naively increasing network depth can harm performance He et al. (2016); instead, we use residual connections to improve convergence for our deep network. The exact design of this architecture is more thoroughly detailed in Appendix A.1.4 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 as shown in Table 1. On PartNet, this margin reaches +2.1% over the shallow convPN and +9.7% over the vanilla PointNet++. We underline the extremely low growth of memory as a function of depth, shown in Fig. 3.

In Table 2 we compare against Deep GCN (Li et al., 2019) in terms of the overall accuracy and the mean IoU on S3DIS dataset (Armeni et al., 2016). We attain similar performance to the current state-of-the-art, while relying on our generic memory-efficient network blocks and while based on a weaker baseline compared to Deep GCN (i.e. DGCNN); as we show in the following section, these blocks comes with the advantage of being applicable to many SOTA point processing networks.

Table 2: Performance of our deepConPN network compared to other SOTA methods on S3DIS. The difference in performance observed on each class can be explained by the different approaches networks have for point convolution. Our deep network clearly outperforms PointNet++ baseline by a spread of +6.8% for mIoU. We achieve similar performance compared to Deep GCN while relying on a weaker baseline (PointNet++ against DGCNN)

Method	OA	mIOU	ceiling	floor	wall	beam	column	window	door	table	chair	sofa	bookcase	board	clutter
MS+CU	79.2	47.8	88.6	**95.8**	67.3	36.9	24.9	48.6	52.3	51.9	45.1	10.6	36.8	24.7	37.5
G+RCU	81.1	49.7	90.3	92.1	67.9	44.7	24.2	52.3	51.2	58.1	47.4	6.9	39.0	30.0	41.9
3DRNN+CF	**86.9**	56.3	92.9	93.8	73.1	42.5	25.9	47.6	59.2	60.4	66.7	24.8	**57.0**	36.7	51.6
DGCNN	84.1	56.1	-	-	-	-	-	-	-	-	-	-	-	-	-
ResGCN-28	85.9	**60.0**	**93.1**	95.3	**78.2**	33.9	**37.4**	**56.1**	**68.2**	64.9	61.0	**34.6**	51.5	51.1	54.4
PointNet	78.5	47.6	88.0	88.7	69.3	42.4	23.1	47.5	51.6	54.1	42.0	9.6	38.2	29.4	35.2
PointNet++	-	53.2	90.2	91.7	73.1	42.7	21.2	49.7	42.3	62.7	59.0	19.6	45.8	48.2	45.6
DeepConvPN	85.7	**60.0**	91.0	95.6	76.1	**50.3**	25.9	55.1	56.8	**66.3**	**74.3**	25.8	54.0	**52.3**	**55.3**

4.3 Evaluation on more architectures

We have introduced building blocks for point processing networks based on two key ideas, (i) a memory efficient convolution and (ii) a multi-resolution approach. Our blocks make it really efficient to capture, process and diffuse information in a point neighbourhood. Diffusing information across neighborhood is the main behavior that most networks, if not all, share. We validate the generality of the proposed modular blocks in the context of other state-of-the-art point-based learning setups, as shown in Table 3. Each of our macro-blocks can be stacked together, extended into a deeper block by duplicating the green boxes (see Figure 2) or even be modified by changing one of its components by another. We test our framework on three additional networks among the latest state-of-the-art approaches, (i) Dynamic Graph CNN (Wang et al., 2018b), (ii) PointCNN (Li et al., 2018b) and (iii) SpiderCNN (Xu et al., 2018). The efficiency of our modules for the KPConv network (Thomas et al., 2019) is explored in Appendix C.4. These networks involve a diverse set of point convolution approaches and thus allows us to assess the generic nature of our modular blocks.

All three of the networks 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 two metrics: (i) memory footprint and (ii) IoU in Table 3. Our lean counterparts consistently improve both the IoU (from +1.0% up to +8.0%) and the memory consumption (from -19% up to -69%).

Table 3: Performance of our blocks on three different architectures (DGCNN, PointCNN and SpiderCNN) on three datasets using two different metrics: (i) memory consumption and (ii) IoU. Our lean counterparts improve significantly both the IoU (up to +8.0%) and the memory consumption (up to -69%).

		DGCNN			PointCNN			SCNN		
		ShapeNet-P	ScanNet	PartNet	ShapeNet-P	ScanNet	PartNet	ShapeNet-P	ScanNet	PartNet
Mem. (Gb)	Vanilla	2.62 (+0%)	7.03 (+0%)	9.50 (+0%)	4.54 (+0%)	5.18 (+0%)	6.83 (+0%)	1.09 (+0%)	4.33 (+0%)	5.21 (+0%)
	Efficient	**0.81 (-69%)**	**3.99 (-43%)**	**5.77 (-39%)**	**1.98 (-56%)**	**3.93 (-24%)**	**5.55 (-19%)**	**0.79 (-28%)**	**3.25 (-25%)**	**3.33 (-36%)**
Perf. (%)	Vanilla	79.8 (+0.0%)	70 (+0.0%)	20.5 (+0.0%)	82 (+0.0%)	73 (+0.0%)	25.0 (+0.0%)	78.0 (+0.0%)	69 (+0.0%)	17.9 (+0.0%)
	Efficient	**81.3 (+1.9%)**	**71 (+1.4%)**	**21.9 (+6.8%)**	**82.8 (+1.0%)**	**76 (+4.1%)**	**27.0 (+8.0%)**	**79.6 (+2.1%)**	**71 (+2.9%)**	**18.4 (+2.8%)**

Our modular blocks can thus be applied to a wide range of state-of-the-art networks and improve significantly their memory consumption while having a positive impact on performance.

4.4 Ablation study

In this section, we report our extensive experiments to assess the importance of each block of our network architectures. 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 (Table 1), (ii) the memory footprint, (iii) the inference time and (iv) the length of the backward pass. Our main experimental findings

regarding network efficiency are reported in Table 4 and ablate the impact of our proposed design choices for point processing networks.

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. Please refer to Appendix C.3 for absolute values.

	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++	+0.0%	+0.0%	+0.0%	+0.0%	+0.0%	+0.0%	+0.0%	+0.0%	+0.0%	+0.0%	+0.0%	+0.0%
mRes	-17.0%	-17.6%	-16.6%	-69.3%	-56.5%	-47.6%	+14.8%	+59.2%	-19.4%	-68.8%	**-53.8%**	-63.2%
mResX	-10.6%	-10.7%	-10.1%	-65.0%	-53.2%	-46.3%	+28.2%	+60.9%	-12.5%	-29.5%	+0.0%	-25.4%
convPN	+13.8%	+13.4%	+12.6%	-75.7%	**-66.6%**	**-57.9%**	**-45.6%**	**-30.3%**	**-47.9%**	**-82.7%**	-42.3%	**-78.9%**
deepConvPN	**+54.3%**	**+54.0%**	**+50.8%**	**-79.1%**	-65.4%	-57.0%	-40.4%	-25.6%	-46.5%	-78.6%	-11.5%	-72.4%

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 1.2% on ShapeNet-Part and 5.7% 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. The training time is quicker due to 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 PartNet, the most complex dataset, with these extra-links by 0.8%.

Memory-efficient Convolutions: As described in Sec. 3.2, 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 network. Our convolution-type network wins on all counts, both on performance and efficiency. Indeed, the IoU is increased by 12.5% on ScanNet and 7.4% on PartNet compared to PointNet++ baseline. Regarding its efficiency, the memory footprint is decreased 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.

5 Conclusion

In this work we have introduced new generic building blocks for point processing networks, that exhibit 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 marginal cost in terms of efficiency and achieves the best IoU on PartNet (+9.7% 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 -69%) and increase in IoU (up to +8.0%). 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 (Mo et al., 2018; Koch et al., 2018).

REFERENCES

Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2016.

Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niener, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. 09 2017.

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher 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.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *CoRR*, abs/1604.06174, 2016. URL http://arxiv.org/abs/1604.06174.

Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

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)*, pp. 103–118, 2018.

Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. The reversible residual network: Backpropagation without storing activations. *CoRR*, abs/1707.04585, 2017. URL http://arxiv.org/abs/1707.04585.

Audrunas Gruslys, Remi Munos, Ivo Danihelka, Marc Lanctot, and Alex Graves. Memory-efficient backpropagation through time. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 4125–4133. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6221-memory-efficient-backpropagation-through-time.pdf.

Paul Guerrero, Yanir Kleiman, Maks Ovsjanikov, and Niloy J. Mitra. PCPNet: Learning local shape properties from raw point clouds. *CGF*, 37(2):75–85, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 2016.

P. Hermosilla, T. Ritschel, P-P Vazquez, A. Vinacua, and T. Ropinski. Monte carlo convolution for learning on non-uniformly sampled point clouds. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2018)*, 37(6), 2018. doi: 10.1145/3272127.3275110.

Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. URL http://arxiv.org/abs/1608.06993.

Yangqing Jia. *Learning Semantic Image Representations at a Large Scale*. PhD thesis, University of California, Berkeley, USA, 2014. URL http://www.escholarship.org/uc/item/64c2v6sn.

Tsung-Wei Ke, Michael Maire, and Stella X. Yu. Neural multigrid. *CoRR*, abs/1611.07661, 2016. URL http://arxiv.org/abs/1611.07661.

Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. ABC: A big CAD model dataset for geometric deep learning. *CoRR*, abs/1812.06216, 2018.

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

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

Guohao Li, Matthias Mller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns?, 2019.

Jiaxin Li, Ben M Chen, and Gim Hee Lee. So-net: Self-organizing network for point cloud analysis. pp. 9397–9406, 2018a.

Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. 2018b.

Kaichun Mo, Shilin Zhu, Angel X. Chang, Li Yi, Subarna Tripathi, Leonidas J. Guibas, and Hao Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. *CoRR*, abs/1812.02713, 2018. URL `http://arxiv.org/abs/1812.02713`.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CVPR*, 1(2):4, 2017a.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, pp. 5099–5108, 2017b.

E. L. Schwartz. Spatial mapping in the primate sensory projection: Analytic structure and relevance to perception. *Biological Cybernetics*, 1977.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2015.

Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. ICCV*, 2015.

Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. Splatnet: Sparse lattice networks for point cloud processing. pp. 2530–2539, 2018.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL `http://arxiv.org/abs/1409.4842`.

Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. *arXiv preprint arXiv:1904.08889*, 2019.

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.

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 (SIGGRAPH)*, 36(4), 2017.

Peng-Shuai Wang, Chun-Yu Sun, Yang Liu, and Xin Tong. Adaptive O-CNN: A Patch-based Deep Representation of 3D Shapes. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 37(6), 2018a.

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*, 2018b.

Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016.

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

Kangxue Yin, Hui Huang, Daniel Cohen-Or, and Hao Zhang. P2p-net: Bidirectional point displacement net for shape transform. *ACM TOG*, 37(4):152:1–152:13, July 2018. ISSN 0730-0301.

Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Ec-net: an edge-aware point set consolidation network. pp. 386–402, 2018a.

Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Pu-net: Point cloud upsampling network. In *CVPR*, 2018b.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016. URL http://arxiv.org/abs/1605.07146.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. 2017.

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. 2014.

Zhiyuan Zhang, Binh-Son Hua, and Sai-Kit Yeung. Shellnet: Efficient point cloud convolutional neural networks using concentric shells statistics. In *International Conference on Computer Vision (ICCV)*, 2019.

A DESIGN OF OUR ARCHITECTURES

In this section, we provide more details about how we design our lean architectures to ensure reproducible results for all tested architectures, (i) PointNet++(Qi et al., 2017b), (ii) Dynamic Graph CNN (Wang et al., 2018b), (iii) SpiderCNN (Xu et al., 2018) and (iv) PointCNN (Li et al., 2018b). We implement each networks in Pytorch following the original code in Tensorflow and implant our blocks.

A.1 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

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 (see Fig. 5).

A.1.1 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.1.2 MRES

The mRes architecture consists in changing the way the sampling is done in the network to get a multiresolution approach (Fig. 4). We provide the details only for the encoding part of the network as we keep the decoding part unchanged from PointNet++.

Encoding1:

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.

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

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 architecture. 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.1.3 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 its 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 one of 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:

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

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 work on the same feature dimension.

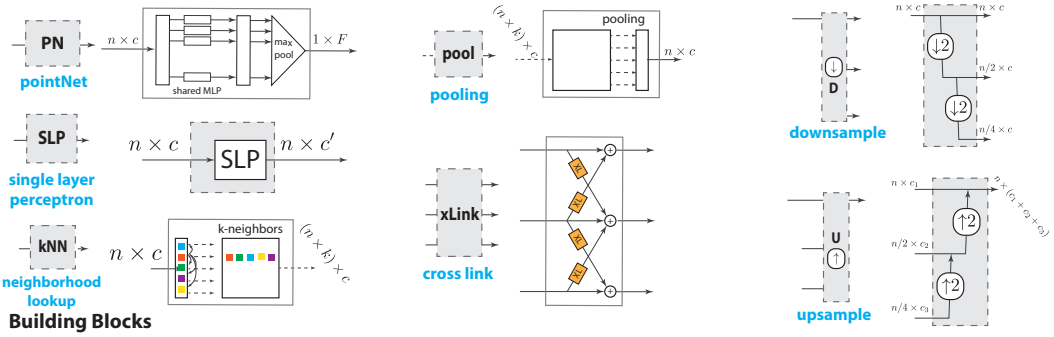

Figure 5: Elementary building blocks for point processing and network layers obtained from their composition. Apart from standard pooling and SLP layers, we introduce cross-link layers across scales, and propose multi-resolution up/down sampling blocks for point processing.

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.1.4 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.

A.2 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 build their MLPs).

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.

A.3 SPIDERCNN BASED ARCHITECTURE

A SpiderConv block can be seen as a bilinear operator on the input features and on a non-linear transformation of the input points. This non-linear transformation consists of changing the space where the points live.

In the original architecture, an 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 outputs are obtained. We thus modify the backward pass to reconstruct the needed tensors for gradient computation.

A.4 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.

A.5 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 pointclouds are downsampled using the furthest point sampling (FPS) algorithm or randomly upsampled.

We keep the exact same parameters as the original networks regarding most of parameters.

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.

B Details on our memory-efficient block

The 2D counterpart of convolution operates in three steps: (i) neighbourhood exposure, (ii) matrix multiplication and (iii) pooling through a sum operator. Our convPN block follows the same steps but it can be seen as non-standard convolution as each weight matrix is constrained to be identical for all neighbours.

To expose the neighbourhood, an intermediate tensor needs to be built to store neighborhood features. This tensor can be then used to gather and refine local information for each patch. This process has been used for image processing as the so-called **im2col** operation to rearrange discrete image blocks in a two-dimensional tensor. Exposing the neighbourhood simplifies the convolution to a simple matrix multiplication and thus fastens the convolution operation but does have a critical memory footprint if not handled properly. Indeed, neighborhood features as any other activations will be saved into memory to allow gradients to flow downward the graph. We design our memory efficient block to build the neighborhood matrix *on-the-fly* (see Algorithm 1 and 2) without the need to store neighborhood features.

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 $R^{D'}$ */

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

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

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

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

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

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

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

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

end

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

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

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

end

C Details and additional evaluation results

C.1 Datasets

We evaluate our network on the point cloud segmentation task on three different datasets, ordered by increasing complexity:

- ShapeNet-Part (Chang et al., 2015): CAD models of 16 different object categories composed of 50 labeled parts. The dataset provides $13,998$ samples for training and $2,874$ samples for evaluation. Point segmentation performance is assessed using the mean point Intersection over Union (mIoU).
- ScanNet (Dai et al., 2017): Scans of real 3D scenes (scanned and reconstructed indoor scenes) composed of 21 semantic parts. The dataset provides $1,201$ samples for training and 312 samples for evaluation. We follow the same protocol as in Qi et al. (2017a) and report both the accuracy and the part Intersection over Union (pIoU).
- PartNet (Mo et al., 2018): Large collection of CAD models of 17 object categories composed of 251 labeled parts. The dataset provides $17,119$ samples for training, $2,492$ for validation and $4,895$ for evaluation. 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 our networks on a more challenging segmentation task using the same part Intersection over Union (pIoU) as in ScanNet.

C.2 Evaluation metrics

To report our results, we use two versions of the Intersection over Union metric:

- 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 sample. 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 sample s.

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

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

  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.

To take into account the randomness of point cloud sampling when performing coarsening, we use the average of 'N' forward passes to decide on the final segmentation during evaluation.

C.3 Detailed results

The following section provides more details on the evaluation experiments introduced in the paper. We present the per-class IoU on both ShapeNet-Part (Table 5) and PartNet (Table 6) datasets for each of the PointNet++ based architecture. Due to the high number of points per sample and the level of details of the segmentation, PartNet can be seen as a much more complex than ShapeNet-Part. As additional reference, we provide on Table 8 the performance of our lean blocks applied to three architectures when training one network per-object category on PartNet (on *Chairs* and *Tables* that represents 60% of the dataset).

On PartNet, the spread between an architecture with an improved information flow and a vanilla one becomes significant. Our PointNet++ based networks perform consistently better than the original architecture on each of the PartNet classes.

Increasing the depth of the network allows to achieve a higher accuracy on the most complex classes such as Chairs or Lamps composed of 38 and 40 different part categories respectively. As shown on Fig. 6, our deep architecture is able to better capture the boundaries between parts and thus to predict the right labels very close from part edges. When a sample is itself composed of many parts, having a deep architecture is a significant advantage.

For reference, we provide as well the absolute values for the efficiency of each of those networks measured by three different metrics on Table 7: (i) memory footprint, (ii) inference time and (iii) length of backward pass.

Table 5: Performance on ShapeNet-Part. The table reports the mIoU performance based on a training on the whole dataset all 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 part categories. All of our network architectures outperform PointNet++ baseline by at least +1.2%. 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**

C.4 Additional results with KPConv

We provide on Table 9 some additional experiments with KPConv network (Thomas et al., 2019), a kernel point convolution approach. We report the efficiency with and without our modules, evaluated both in terms of memory consumption and forward/backward speed. Our modules successfully help to reduce the memory by up to 52.5% while having no impact on the speed of the forward or backward pass.

Table 6: Performance on PartNet. The table reports the Part IoU performance based on a training on the whole dataset all at once in contrast with Mo et al. (2018). The fine details 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 the objects properly. Our different architectures outperform PointNet++ with a spread of at least 2.0% (+5.7% increase). With this more complex dataset, deeper networks become significantly better: our deepConvPN network achieves to increase pIoU by +9.7% over PointNet++ baseline, outperforming its shallow counterpart by +2.1%.

	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 7: 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

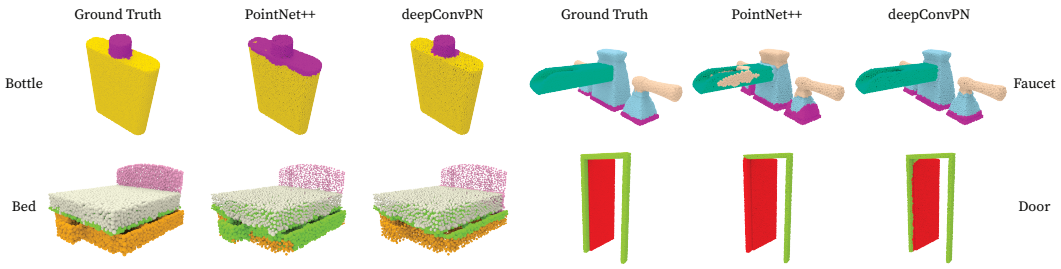

Figure 6: Segmentation prediction on the test set for both PointNet++ and our deepConvPN network compared to the ground truth. While PointNet++ struggles to detect accurately the boundaries between different parts, our deep architecture performs a much finer segmentation in those frontier areas.

Table 8: Per-class IoU on PartNet when training a separate network for each category, evaluated for three different architectures for *Chairs* and *Tables* (60% of the whole dataset). Our efficient networks achieve here similar performance as their vanilla counterpart while delivering significant savings in memory.

	DGCNN		PointCNN		SpiderCNN	
	Vanilla	Efficient	Vanilla	Effiicent	Vanilla	Efficient
Chair	**29.2 (+0.0%)**	24.2 (-17.1%)	40.4 (+0.0%)	**41.4 (+2.5%)**	30.8 (+0.0%)	**31.1 (+1.0%)**
Table	22.5 (+0.0%)	**28.9 (+28.4%)**	32.1 (+0.0%)	**33.1 (+3.1%)**	**21.3 (+0.0%)**	21.2 (-0.5%)

Table 9: Efficiency of our memory efficient blocks applied to a KPConv backbone. Our blocks allow to decrease significantly the memory footprint by up to 52.5% with no impact both on inference and backward speed.

Network	Dataset	Version	Memory (Gb)	Forward pass Time (ms)	Backward pass Time (ms)
KPConv	ShapeNet-Part	Vanilla	7.38 (0%)	**51 (+0%)**	171 (+0%)
		Efficient	**3.89 (-47%)**	**51 (+0%)**	**166 (-3%)**
	PartNet	Vanilla	4.48 (+0.0%)	**42 (+0.0%)**	240 (+0.0%)
		Efficient	**2.13 (-52.5%)**	43 (+2.4%)	**236 (-1.7%)**