PointConvFormer: Revenge of the Point-based Convolution

Wenxuan Wu\textsuperscript{1,2}, Li Fuxin\textsuperscript{1,3}, Qi Shan\textsuperscript{3}
\textsuperscript{1}Oregon State University, \textsuperscript{2}CASIA, \textsuperscript{3}Apple, Inc.
\{wuwen, lif\}@oregonstate.edu, \{fli2, qshan\}@apple.com

Abstract

We introduce PointConvFormer, a novel building block for point cloud based deep network architectures. Inspired by generalization theory, PointConvFormer combines ideas from point convolution, where filter weights are only based on relative position, and Transformers which utilize feature-based attention. In PointConvFormer, attention computed from feature difference between points in the neighborhood is used to modify the convolutional weights at each point. Hence, we preserved the invariances from point convolution, whereas attention helps to select relevant points in the neighborhood for convolution. PointConvFormer is suitable for multiple tasks that require details at the point level, such as segmentation and scene flow estimation tasks. We experiment on both tasks with multiple datasets including ScanNet, SemanticKitti, FlyingThings3D and KITTI. Our results show that PointConvFormer offers a better accuracy-speed tradeoff than classic convolutions, regular transformers, and voxelized sparse convolution approaches. Visualizations show that PointConvFormer performs similarly to convolution on flat areas, whereas the neighborhood selection effect is stronger on object boundaries, showing that it has got the best of both worlds. The code will be available.

1. Introduction

Depth sensors for indoor and outdoor 3D scanning have significantly improved in terms of both performance and affordability. Hence, their common data format, the 3D point cloud, has drawn significant attention from academia and industry. Understanding the 3D real world from point clouds can be applied to many application domains, e.g. robotics, autonomous driving, CAD, and AR/VR. However, unlike image pixels arranged in regular grids, 3D points are unstructured, which makes applying grid based Convolutional Neural Networks (CNNs) difficult.

Various approaches have been proposed in response to this challenge. \cite{3, 5, 31, 35, 37, 65} introduced interesting ways to project 3D point clouds back to 2D image space and apply 2D convolution. Another line of research directly voxelizes the 3D space and apply 3D discrete convolution, but it induces massive computation and memory overhead \cite{47, 62}. Sparse 3D convolution operations \cite{9, 20} save a significant amount of computation by computing convolution only on occupied voxels.

Some approaches directly operate on point clouds \cite{41, 57, 58, 64, 69, 81}. \cite{57, 58} are pioneers which aggregate information on point clouds using max-pooling layers. Others proposed to reorder the input points with a learned transformation \cite{42}, a flexible point kernel \cite{69}, and a convolutional operation that directly work on point clouds \cite{75, 81} which utilizes a multi-layer perceptron (MLP) to learn convolution weights implicitly as a nonlinear transformation from the relative positions of the local neighbourhood.

The approach to directly work on points is appealing to us because it allows direct manipulation of the point coordinates, thus being able to encode rotation/scale invariance/equivariance directly into the convolution weights \cite{41, 42, 58, 94}. These invariances serve as priors to make the models more generalizable. Besides, point-based approaches require less parameters than voxel-based ones, which need to keep e.g. $3 \times 3 \times 3$ convolution kernels on all input and output channels. Finally, point-based ap-
proaches can utilize k-nearest neighbors (kNN) to find the local neighborhood, thus can adapt to variable sampling densities in different 3D locations.

However, so far the methods with the best accuracy-speed tradeoff have still been the sparse voxel-based approaches or a fusion between sparse voxel and point-based models. Note that no matter the voxel-based or point-based representation, the information from the input is exactly the same, so it is unclear why fusion is needed. Besides, fusion adds significantly to model complexity and memory usage. This leads us to question the component that is indeed different between these representations: the generalization w.r.t. the irregular local neighbourhood. The shape of the kNN neighbourhood in point-based approaches varies in different parts of the point cloud. Irrelevant points from other objects, noise and the background might be included in the neighborhood, especially around object boundaries, which can be detrimental to the performance and the robustness of point-based models.

To improve the robustness of models with kNN neighborhoods, we refer back to the generalization theory of CNNs, which indicates that points with significant feature correlation should be included in the same neighborhood [40]. A key idea in this paper is that feature correlation can be a way to filter out irrelevant neighbors in a kNN neighborhood, which makes the subsequent convolution more generalizable. We introduce PointConvFormer, which computes attention weights based on feature differences and uses that to reweight the points in the neighborhood in a point-based convolutional model, which indirectly “improves” the neighborhood for generalization.

The idea of using feature-based attention is not new, but there are important differences between PointConvFormer and other vision transformers [16,54,96]. PointConvFormer combines features in the neighborhood with point-wise convolution, whereas Transformer attention models usually adopt softmax attention in this step. In our formulation, the positional information is outside the attention, hence viewpoint-invariance can be introduced into the convolutional weights. We believe that invariance helps generalizing across neighborhood (size/rotation) differences between training/testing sets, especially with a kNN neighborhood.

We evaluate PointConvFormer on two point cloud tasks, semantic segmentation and scene flow estimation. For semantic segmentation, experiment results on the indoor ScanNet [11] and the outdoor SemanticKitti [2] demonstrate superior performances over classic convolution and transformers with a much more compact network. The performance gap is the most significant at low resolutions, e.g. on ScanNet with a 10cm resolution we achieved more than 10% improvement over MinkowskiNet with only 15% of its parameters (Fig. 1). We also apply PointConvFormer as the backbone of PointPWC-Net [82] for scene flow estimation, and observe significant improvements on FlyingThings3D [48] and KITTI scene flow 2015 [52] datasets as well. These results show that PointConvFormer could potentially compete with sparse convolution as the backbone choice for dense prediction tasks on 3D point clouds.

2. Related Work

Voxel-based networks. Different from 2D images, 3D point clouds are unordered and scattered in 3D space. A standard approach to process 3D point clouds is to voxelize them into regular 3D voxels. However, directly applying dense 3D convolution [47,62] onto the 3D voxels can incur massive computation and memory overhead, which limits its applications to large-scale real world scenarios. The sparse convolution [9,20] reduces the convolutional overhead by only working on the non-empty voxels. There are also some work [67,68] that working on making the network inference more efficient for voxel processing, which could also be extended to point-based methods.

Point-based networks. Point-based approaches [41,57,58,64,76,81] directly process point clouds without relying on the voxel structure. [57] propose to use MLPs followed by max-pooling layers to encode and aggregate point cloud features. However, max-pooling can lead to the loss of critical geometric information in the point cloud. [58] improves over [57] with a hierarchical structure by gradually downsampling and grouping the point cloud with k-nearest neighbours or a query ball method. A number of works [19,30,38,49,50,73] build a kNN graph from the point cloud and conduct message passing using graph convolution. To better encode the local information, [17,41,42,46,69,75,81,86] conduct continuous convolution on point clouds. [75] represents the convolutional weights with MLPs. SpiderCNN [86] uses a family of polynomial functions to approximate the convolution kernels. [64] projects the whole point cloud into a high-dimensional grid for rasterized convolution. [69,81] formulate the convolutional weights to be a function of relative position in a local neighbourhood, where the weights can be constructed according to input point clouds. [41] introduces hand-crafted viewpoint-invariant coordinate transforms on the relative position to increase the robustness of the model.

Dynamic filters and Transformers. Recently, the design of dynamic convolutional filters [4,6,7,12,29,30,45,63,70,74,77,79,83,85,88,91,93,97] has drawn more attentions. This line of work [6,45,88,93] introduces methods to predict convolutional filters, which are shared across the whole input. [30,70,77,91] propose to predict complete convolutional filters for each pixel. However, their applications are constrained by their significant runtime and high memory usage. [97] introduces decoupled dynamic filters w.r.t. the input features on 2D classification and upsampling tasks. [63,66] propose to re-weight 2D convolutional ker-
Figure 2. (a) **PointConvFormer** can be seen as a point convolution, but modulated by a scalar attention weight for each point in the neighborhood, so that the neighborhood is selectively chosen to perform convolution; (b) Visualization of the reweighting effect in PointConvFormer. The colors are computed by the difference of the maximal attention and the minimal attention in each neighborhood. Red areas have stronger reweighting and blue areas behave similar to convolution. It can be seen that the reweighting effect is stronger at object boundaries where the neighborhoods are more likely to be problematic, whereas on smoother surfaces PointConvFormer behaves more similarly to convolution (more details in Sec. 4.4).

With recent success in natural language processing [13, 15, 71, 80, 90] and 2D images analysis [16, 23, 59, 95], transformers have drawn more attention in the field of 3D scene understanding. Some work [36, 43, 84, 89] utilize global transformers have drawn more attention in the field of 3D scene analysis [15, 71, 80, 90] and 2D images analysis [16, 23, 59, 95], transformations introduce heavy computation overhead and are unable to extend to large-scale real-world scenes, which usually contain over 100k points per point cloud scan. Recently, the work [54, 83, 96] introduce point transformer with local attention to reduce the computation overhead, which could be applied to large scenes. Compared to transformers, the attention of the PointConvFormer modulates convolution kernels instead of softmax aggregation.

3. **PointConvFormer**

3.1. **Point Convolutions and Transformers**

Given a continuous input signal \( x(p) \in \mathbb{R}^{c_{\text{in}}} \) where \( p \in \mathbb{R}^s \) with \( s \) being a small number (2 for 2D images or 3 for 3D point clouds, but could be any arbitrary low-dimensional Euclidean space), \( x(\cdot) \) can be sampled as a point cloud \( P = \{p_1, \ldots, p_n\} \) with the corresponding values \( x_p = \{x(p_1), \ldots, x(p_n)\} \), where each \( p_i \in \mathbb{R}^s \). The continuous convolution at point \( p \) is formulated as:

\[
\text{Conv}(w, x)_p = \int_{\Delta p \in \mathbb{R}^s} \langle w(\Delta p), x(p + \Delta p) \rangle d\Delta p \tag{1}
\]

where \( w(\Delta p) \in \mathbb{R}^{c_{\text{in}}} \) is the continuous convolution weight function. [61, 76, 81] discretize the continuous convolution on a neighbourhood of point \( p \). The discretized convolution on point clouds is written as:

\[
x'_p = \sum_{p_i \in \mathcal{N}(p)} w(p_i - p) \top x(p_i) \tag{2}
\]

where \( \mathcal{N}(p) \) is a neighborhood that is conventionally chosen as the \( k \)-nearest neighbor or \( \epsilon \)-ball neighborhood of the center point \( p \). The function \( w(p_i - p) : \mathbb{R}^s \rightarrow \mathbb{R}^{c_{\text{in}}} \) can be approximated as an MLP and learned from data. Moreover, because now \( p_i - p \) can be explicitly controlled, one can concatenate invariant coordinate transforms on \( p_i - p \) as input to \( w(\cdot) \), e.g., \( \|p_i - p\| \) would be rotation invariant. [41] has found that concatenating a set of rotation and scale-invariant coordinate transforms with \( p_i - p \) significantly improves the performance of point convolutions.

In PointConv [81], an efficient formulation was derived when \( w(p_i - p) \) has a linear final layer \( w(p_i - p) = W_l h(p_i - p) \), where \( h(p_i - p) : \mathbb{R}^3 \rightarrow \mathbb{R}^{c_{\text{mid}}} \) is the output of the penultimate layer of the MLP and \( W_l \in \mathbb{R}^{c_{\text{out}} \times c_{\text{mid}}} \) is the learnable parameters in the final linear layer. Eq. (2) can be equivalently written as:

\[
X'_p = \left\{ \text{vec}(W_l) \cdot \text{vec}\left( \sum_{p_i \in \mathcal{N}(p)} h(p_i - p)x(p_i) \top \right) \right\} . \tag{3}
\]

where \( \text{vec}(\cdot) \) turns the matrix into a vector. Note that \( W_l \) represents parameters of a linear layer and hence independent of \( p_i \). Thus, when there are \( c_{\text{out}} \) convolution kernels, \( n \) training examples with a neighborhood size of \( k \) each, there is no longer a need to store the original convolution weights \( w(p_i - p) \) for each point in each neighborhood with a dimensionality of \( c_{\text{out}} \times c_{\text{in}} \times k \times n \). Instead, the dimension of all the \( h(p_i - p) \) vectors in this case is only \( c_{\text{mid}} \times k \times n \), where \( c_{\text{mid}} \) is significantly smaller (usually 4 to 16) than \( c_{\text{out}} \times c_{\text{in}} \) (which could go higher than \( 10^4 \)). This efficient PointConv enables applications to large-scale networks on 3D point cloud processing. Besides, its model size is significantly smaller than sparse 3D convolutions, which require \( 27 \times c_{\text{in}} \times c_{\text{out}} \) parameters, whereas PointConv only requires
$c_{mid} \times c_{in} \times c_{out}$ for its most costly linear layer $W_i(\cdot)$ ($h(\cdot)$ requires only less than 100 parameters). As we will see later, $c_{mid}$ can be as small as 4 for reasonable performance.

Recently, transformer architectures are popular with 2D images. 3D point cloud-based transformers have also been proposed (e.g. [54, 96]). Transformers compute an attention model between points (or pixels) based on the features of both points and the positional encodings of them. Relative positional encoding was the most popular which encodes $w(p_i - p)$, similar to Eq. (2). It has been shown to outperform absolute positional encodings [10, 60, 96]. Adopting similar notations to Eq. (2), we can express the softmax attention model used in transformers as:

$$Attention(p) = \sum_{p_i \in N(p)} \text{softmax}(q(x(p)))k(x(p)) + w(p_i - p) \cdot v(x(p))$$

where $q(\cdot), k(\cdot), v(\cdot)$ are transformation to the features to form the query, key and value matrices respectively, usually implemented with MLPs. Comparing PointConv [81] and attention, one can see that both employ $w(p_i - p)$, but in PointConv that is the sole source of the convolutional kernel which is translation-invariant. In attention models, the matching between the query transform $q(x(p))$ and the key transform $k(x(p))$ of the features are also considered. Besides, having $h(p_i - p)$ outside the attention allows PointConv to utilize an invariant coordinate transform (e.g. scale/rotation) which helps with robustness. Transformers usually adopt 2 fully connected layers after the attention module, which are analogous with the $h(\cdot)$ and $W(\cdot)$ layers. Coincidentally, the first fully-connected layer in transformers usually also adopt a dimensionality expansion of $4 \times$, which is similar to the minimal $c_{mid}$ we found to be performant in PointConv as well. Hence, an entire transformer block can be seen as similar to a PointConv block, but with the features $x(p)$ also participating in the process of generating weights.

However, one needs to note that in Eq. (3), $h(p_i - p)$ creates a set of nonlinear transforms of $p_i - p$ with $c_{mid} > 1$, which does not exist in Eq. (4) if only a single head is used. This indicates that with Eq. (4) only nonnegative combinations of the input features can be made. In transformers, this is remedied with the multi-head design where different heads are capable to use different $v(\cdot)$s to enable negative contributions of neighbors. Such a design is not required in PointConv where different $h(\cdot)$ combined with $W_i$ enables both positive and negative contributions of each neighbor.

### 3.2. CNN Generalization Theory and The PointConvFormer Layer

We are interested in adopting the strengths of attention-based models, while still preserve some of the benefits of convolution and explore the possibility of having negative weights. To this end, we first look at theoretical insights in terms of which architecture would generalize well. We note the following bound proved in [40]:

$$\hat{G}_N(F) \leq C \max_{p, p' \in \mathcal{N}(p)} \sqrt{E_{x, p}(\|x(p) - x(p')\|^2)}$$

where $\hat{G}_N(F)$ is the empirical Gaussian complexity on the function class $F$: a one-layer CNN followed by a fully-connected layer, and $C$ is a constant. A smaller Gaussian complexity leads to better generalization [1]. To minimize the bound in Eq. (5), one should select points that has high feature correlation to belong to the same neighborhood. In images, nearby pixels usually have the highest color correlation [40], hence conventional CNNs achieve better generalization by choosing a small local neighborhood (e.g. $3 \times 3$). Although the bound does not directly apply to transformer models, the idea of using attention to exclude less correlated neighbors could directly improve the generalization bound in Eq. (5) for CNNs. In 3D point clouds, noisy points can be included in the kNN neighborhood, which motivates us to attempt to filter out those noisy points by explicitly checking their $x(p) - x(p')$, keeping only the relevant points in the neighborhood.

Inspired by the discussion above, we define a novel convolution operation, PointConvFormer, which takes into account both the relative position $p_i - p$ and the feature difference $x(p_i) - x(p)$. The PointConvFormer layer of a point $p$ with its neighborhood $\mathcal{N}(p)$ can be written as:

$$x'_{p} = \sum_{p_i \in \mathcal{N}(p)} w(p_i - p)^T \psi([x(p_i) - x(p), p_i - p])x(p_i)$$

where the function $w(p_i - p)$ is the same as defined in Eq. (2), the scalar function $\psi([x(p_i) - x(p), p_i - p]) : \mathbb{R}^{c_{in}} \rightarrow \mathbb{R}$ is the function of both feature differences $X_{p_i} - X_p$ and position differences. In practice, $\psi(\cdot)$ is approximated with an MLP followed by an activation layer.

If we fix the function $\psi(\cdot) = 1$, the PointConvFormer layer is equivalent to Eq. (2) and reduces to traditional convolution. In Eq. (6), the function $w(p_i - p)$ learns the weights respect to the relative positions, and the function $\psi(\cdot)$ learns to select useful points in the neighborhood, which works similarly to the attention in transformer. However, different from the transformer whose non-negative weights are directly used as a weighted average on the input, the output of $\psi(\cdot)$ only modifies the convolutional filter $w(p_i - p)$, which allows each neighborhood point to have both positive and negative contributions. Hence, our design is more lightweight than [83] in which $\psi$ outputs a vector for each head. Besides, leaving $w(p_i - p)$ outside $\psi$ allows it to take full advantage of invariance-aware positional encodings.

Since $\psi(\cdot)$ is outside the convolution, we adopt the same approach as PointConv [81] to create an efficient version of
the PointConvFormer layer. Following Eq. (3), we have:
\[ x'_p = W_l \sum_{p_i \in N(p)} h(p_i - p)\psi(x(p_i), x(p))x(p_i)^T \]  

(7)

where \( W_l \) and \( h(\cdot) \) are the same as in Eq. (3).

**Multi-Head Mechanism** As in Eq.(7), the weight function \( \psi(\cdot) : \mathbb{R}^{c_{in}} \rightarrow \mathbb{R} \) learns the relationship between the center point feature \( x(p) \in \mathbb{R}^{c_{in}} \) and its neighbourhood features \( x(p_i) \in \mathbb{R}^{c_{in}} \), where \( c_{in} \) is the number of the input feature dimension. To increase the representation power of the PointConvFormer, we use the multi-head mechanism to learn different types of neighborhood filtering mechanisms. As a result, the function \( \psi : \mathbb{R}^{c_{in}} \rightarrow \mathbb{R} \) becomes a set of functions \( \psi_i : \mathbb{R}^{c_{in}} \rightarrow \mathbb{R} \) with \( i \in \{1, \ldots, z\} \), \( z \) being the number of heads.

### 3.3. PointConvFormer Block

To build deep neural networks for various computer vision tasks, we construct bottleneck residual blocks with PointConvFormer layer as its main components. The detailed structures of the residual blocks are illustrated in Fig. 3. The input of the residual block is the input point features \( X \in \mathbb{R}^{c_{in}} \) along with its coordinates \( p \in \mathbb{R}^3 \). The residual block uses a bottleneck structure, which consists of two branches. The main branch is a linear layer, followed by PointConvFormer layer, followed by another linear layer. Following ResNet and KPConv [22, 69], we use one-fourth of the input channels in the first linear layer, conduct PointConvFormer with the smaller number of channels, and finally upsample to the amount of output channels. We have found this strategy to significantly reduce the model size and computational cost while maintaining high accuracy for both PointConv and PointConvFormer.

![Figure 3. The residual blocks of PointConvFormer.](image-url)

**Downsampling and Deconvolution** We use the grid-subsampling method [69] to downsample the point clouds similar to the \( 2 \times 2 \) downsampling in 3D convolutions, which had been shown to outperform random and furthest point downsampling [69]. This subsampling choice makes PointConvFormer comparable with 3D convolution backbones at the same voxelization levels (e.g. 2cm, 5cm) as both use the same voxelization and downsampling strategies. For upsampling layers, we cannot apply PointConvFormer because no feature is available for points that are not part of the downsampled cloud. Instead, we note that in (3) of PointConv, \( p \) itself does not have to belong to \( N(p) \), thus we can just apply PointConv layers for deconvolution without features \( x(p) \) as long as coordinates \( p \) are known. This helps us to keep the consistency of the network and avoid arbitrary interpolation layers that are not learnable.

### 4. Experiments

We conduct experiments in a number of domains and tasks to demonstrate the effectiveness of the proposed PointConvFormer in tasks that require point-level accuracy. For 3D semantic segmentation, we use the challenging ScanNet [11], a large-scale indoor scene dataset, and the SemanticKitti dataset [2], a large-scale outdoor scene dataset. Besides, we conduct experiments on the scene flow estimation from 3D point clouds with the synthetic FlyingThings3D dataset [48] for training and the KITTI scene flow 2015 dataset [52]. We also conduct ablation studies to explore the properties of the PointConvFormer.

**Implementation Details.** We implement PointConvFormer in PyTorch [55]. The viewpoint-invariant coordinate transform in [41] is concatenated with the relative coordinates as the input to the \( h(\cdot) \) function. We use the AdamW optimizer with \((0.9, 0.999)\) betas and 0.05 weight decay. For the ScanNet dataset, we train the model with an initial learning rate 0.001 and dropped to half for every 8 epochs for 40 epochs. A weighted cross entropy loss is used. To ensure fair comparison with published approaches, we did not employ the recent Mix3D augmentations [55] except when it is clearly marked that we used it. For the scene flow estimation, we follow the exact same training pipeline as in [82] for fair comparison.

### 4.1. Indoor Scene Semantic Segmentation

We conduct indoor 3D semantic scene segmentation on the ScanNet [11] dataset. We use the official split with 1,201 scenes for training and 312 for validation. We compare against both voxel-based methods such as MinkowskiNet42 [9] and SparseConvNet [20], as well as point-based approaches [41, 57, 69, 81, 87]. Recently, there are work adopting transformer to point clouds. We chose the Point Transformer [96] and the Fast Point Transformer [54] as representative transformer based methods. Since the Point Transformer does not report their results on the ScanNet dataset, we adopt their point transformer layer (a standard multi-head attention layer) with the same network structure as ours. Hence, it serves as a direct comparison between PointConvFormer and multi-head attention. There
exists some other approaches [8, 26, 27, 33] which use additional inputs, such as 2D images, which benefit from ImageNet [14] pre-training that we do not use. Hence, we excluded these methods from comparison accordingly, but we are comparable to the best of them.

We adopt a general U-Net structure with residual blocks in the encoding layers as our backbone model. Through experiments we found out that the decoder can be very lightweight without sacrificing performance (shown in supplementary). Hence, we set $c_{mid}$ to be 1 in the decoder throughout the experiments, and just have consecutive PointConv upsampling layers without any residual blocks. Please refer to the supplementary for detailed network structure. Following [54], we conduct experiments on different input voxel sizes, reported in Fig. 1 and Table 1. We re-implemented PointConv using the bottleneck architecture which yielded significantly better performance than the original paper, yet still significantly trails our PointConvFormer using the same codebase.

We compare the top results among work in the literature in Table 2. According to Fig. 1, Table. 1 and Table. 2, our PointConvFormer offers the best accuracy-speed trade-off regardless of the input grid size. Especially, our PointConvFormer outperforms MinkowskiNet42 [9] by a significant 10.1% with 10cm input grid, 7.3% with 5cm input grid, and 2.3% with 2cm input grid, while being faster than it in the first two cases. It is also significantly faster than all the transformer approaches. On top of this, mix3D could further improve results at 10cm and 5cm resolutions. On 2cm, although mix3D did not improve results on our model with 9.4M parameters, it helped to propel a much smaller model with 5.8M parameters to a similar accuracy.

We also provide lightweight versions named PointConvFormer-Lite, which utilized less layers in each stage. They are faster and more memory efficient with minimal degradation in performance. Fig. 4 is the visualization of the comparison among PointConv [81], Point Transformer [96] and PointConvFormer on the ScanNet dataset [11]. We observe that PointConvFormer is able to achieve better predictions with fine details comparing with PointConv [81] and Point Transformer [96]. Interestingly, it seems that PointConvFormer is usually able to find the better prediction out of PointConv [81] and Point Transformer [96], showing that its novel design brings the best out of both operations.

### 4.2. Outdoor Scene Semantic Segmentation

SemanticKitti [2, 18] is a large-scale street view point cloud dataset built upon the KITTI Vision Odometry Benchmark [18]. The dataset consists of 43, 552 point cloud scans sampled from 22 driving scene sequences. Each point cloud scan contains 10 to 13k points. We follow the training and validation split in [2] and 19 classes are used for training and evaluation. For each 3D point, only the $(x, y, z)$ coordinates are given without any color information. It is a challenging dataset because the scanning density is uneven as faraway points are more sparse in LiDAR scans.

Table 3 reports the results on the semanticKitti dataset. Because this work mainly focus on the basic building block, PointConvFormer, which is applicable to any kind of 3D point cloud data, of deep neural network, we did not compare with work [7, 98] whose main novelties work mostly on LiDAR datasets due to the additional assumption that there are no occlusions from the bird-eye view. From the table, one can see that our PointConvFormer outperforms both point-based methods and point+voxel fusion methods. Especially, our method obtains better results comparing with SPVNAS [68], which utilizes the network architecture search (NAS) techniques and fuses both point and voxel branches. We did not utilize any NAS in our system which would only further improve our performance.

### 4.3. Scene Flow Estimation from Point Clouds

Scene flow is the 3D displacement vector between each surface in two consecutive frames. As a fundamental tool for low-level understanding of the world, scene flow can be used in many 3D applications. Traditionally, scene flow was estimated directly from RGB data [28, 51, 72]. However, with the recent development of 3D sensors such as LiDAR and 3D deep learning techniques, there is increasing interest on directly estimating scene flow from 3D point clouds [21, 44, 56, 78, 82]. In this work, we adopt PointConvFormer into the PointPWC-Net [82], which utilizes a coarse-to-fine framework for scene flow estimation, by replacing the PointConv in the feature pyramid layers with the
Figure 4. ScanNet result visualization. We visualize the ScanNet prediction results from our PointConvFormer, PointConv [81] and Point Transformer [96]. The red ellipses indicates the improvements of our PointConvFormer over other approaches. Points with ignore labels are filtered for a better visualization. (Best viewed in color)

Table 2. Semantic segmentation results on ScanNet dataset. We compare both the ScanNet [11] validation set and test set. Numbers for baselines are taken from the literature. The numbers for test set are from the official ScanNet benchmark.

<table>
<thead>
<tr>
<th>Methods</th>
<th># Params(M)</th>
<th>Input</th>
<th>Runtime(ms)</th>
<th>Val mIoU(%)</th>
<th>Test mIoU(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointNet++ [58]</td>
<td>-</td>
<td>Point</td>
<td>-</td>
<td>53.5</td>
<td>55.7</td>
</tr>
<tr>
<td>PointConv [81]</td>
<td>-</td>
<td>Point</td>
<td>83.1</td>
<td>65.1</td>
<td>66.6</td>
</tr>
<tr>
<td>KPConv deform [69]</td>
<td>14.9</td>
<td>Point</td>
<td>-</td>
<td>69.2</td>
<td>68.4</td>
</tr>
<tr>
<td>PointASNL [87]</td>
<td>-</td>
<td>Point</td>
<td>-</td>
<td>63.5</td>
<td>66.6</td>
</tr>
<tr>
<td>RandLA-Net [24]</td>
<td>-</td>
<td>point</td>
<td>-</td>
<td>-</td>
<td>64.5</td>
</tr>
<tr>
<td>VI-PointConv [51]</td>
<td>15.5</td>
<td>Point</td>
<td>88.9</td>
<td>70.1</td>
<td>67.6</td>
</tr>
<tr>
<td>SparseConvNet [20]</td>
<td>-</td>
<td>Voxel</td>
<td>-</td>
<td>69.3</td>
<td>72.5</td>
</tr>
<tr>
<td>MinkowskiNet42 [5]</td>
<td>37.9</td>
<td>Voxel</td>
<td>115.6</td>
<td>72.2</td>
<td>73.6</td>
</tr>
<tr>
<td>PointTransformer [96]</td>
<td>-</td>
<td>Point</td>
<td>-</td>
<td>70.6</td>
<td>-</td>
</tr>
<tr>
<td>Fast Point Transformer [54]</td>
<td>37.9</td>
<td>Voxel</td>
<td>312.0</td>
<td>72.0</td>
<td>-</td>
</tr>
<tr>
<td>Stratified Transformer [34]</td>
<td>18.8</td>
<td>point</td>
<td>1689.3</td>
<td>74.3</td>
<td>74.7</td>
</tr>
<tr>
<td>PointTransformerV2 [83]</td>
<td>12.8</td>
<td>point</td>
<td>266</td>
<td>75.4</td>
<td>75.2</td>
</tr>
<tr>
<td>PointConvFormer(ours)</td>
<td>9.4</td>
<td>Point</td>
<td>145.5</td>
<td>74.5</td>
<td>74.9</td>
</tr>
</tbody>
</table>

Table 3. Semantic segmentation results on SemanticKitti validation set.

<table>
<thead>
<tr>
<th>Method</th>
<th>#MACs (G)</th>
<th># Params (M)</th>
<th>Input</th>
<th>mIoU (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RandLA-Net [25]</td>
<td>66.5</td>
<td>1.2</td>
<td>Point</td>
<td>57.1</td>
</tr>
<tr>
<td>FusionNet [92]</td>
<td>-</td>
<td>-</td>
<td>Point+Voxel</td>
<td>63.7</td>
</tr>
<tr>
<td>KPRNet [32]</td>
<td>-</td>
<td>-</td>
<td>Point+Range</td>
<td>64.1</td>
</tr>
<tr>
<td>MinkowskiNet [9]</td>
<td>113.9</td>
<td>21.7</td>
<td>Voxel</td>
<td>61.1</td>
</tr>
<tr>
<td>SPVCNN [68]</td>
<td>118.6</td>
<td>21.8</td>
<td>Point+Voxel</td>
<td>63.8</td>
</tr>
<tr>
<td>SPVNAS [68]</td>
<td>64.5</td>
<td>10.8/12.5</td>
<td>Point+Voxel</td>
<td>64.7</td>
</tr>
<tr>
<td>PointConvFormer</td>
<td>91.1</td>
<td>8.1</td>
<td>Point</td>
<td>67.1</td>
</tr>
</tbody>
</table>

PointConvFormer and keeping the rest of the structure the same as the original version of PointPWC-Net.

We name the new network ‘PCFPWC-Net’ where PCF stands for PointConvFormer. To train the PCFPWC-Net we follow the training pipeline in [82]. For a fair comparison, we use the same dataset configurations, hyper-parameters and training pipelines used in [82]. Please check supplementary for more details. From Table 4, we can see that PCFPWC-Net outperforms previous methods in almost all the evaluation metrics. Comparing with PointPWC-Net [82], our PCFPWC-Net achieves around 10% improvement in all metrics. On the KITTI dataset, our PCFPWC-Net also shows strong result for scene flow estimation by improving the EPE3D by more than 30% (0.0694 → 0.0479) over PointPWC-Net [82]). The qualitative results can be found in supplementary.

4.4. Visualization of Reweighted Scores

We visualize the difference of the learned attention for a few example scenes in the ScanNet [11] dataset. The differ-
Table 4. Evaluation results on Scene Flow Datasets. All approaches are trained on FlyingThings3D with the supervised loss. On KITTI, the models are directly evaluated on KITTI without any fine-tuning.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Method</th>
<th>EPE3D(m)</th>
<th>Acc3DS↑ (%)</th>
<th>Acc3DR↑ (%)</th>
<th>Outliers3D↓ (%)</th>
<th>EPE2D(px)↓ (%)</th>
<th>Acc2D↑ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flyingthings3D</td>
<td>FlowNet3D [44]</td>
<td>0.1136</td>
<td>0.4125</td>
<td>0.7706</td>
<td>0.6016</td>
<td>5.9740</td>
<td>0.5692</td>
</tr>
<tr>
<td></td>
<td>HPLFlowNet [21]</td>
<td>0.0804</td>
<td>0.6144</td>
<td>0.8555</td>
<td>0.4287</td>
<td>4.6723</td>
<td>0.6764</td>
</tr>
<tr>
<td></td>
<td>HCRF-Flow [39]</td>
<td>0.0488</td>
<td>0.8337</td>
<td>0.9507</td>
<td>0.2614</td>
<td>2.5652</td>
<td>0.8704</td>
</tr>
<tr>
<td></td>
<td>FLOT [56]</td>
<td>0.052</td>
<td>0.732</td>
<td>0.927</td>
<td>0.357</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>PV-RAFT [78]</td>
<td>0.0461</td>
<td>0.8169</td>
<td>0.9574</td>
<td>0.2924</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>PointPWC-Net [82]</td>
<td>0.0588</td>
<td>0.7379</td>
<td>0.9276</td>
<td>0.3424</td>
<td>3.2390</td>
<td>0.7994</td>
</tr>
<tr>
<td></td>
<td>PCFPWC-Net (ours)</td>
<td>0.0416</td>
<td>0.8645</td>
<td>0.9658</td>
<td>0.2263</td>
<td>2.2967</td>
<td>0.8871</td>
</tr>
<tr>
<td>KITTI</td>
<td>FlowNet3D [49]</td>
<td>0.1767</td>
<td>0.3738</td>
<td>0.6677</td>
<td>0.5271</td>
<td>7.2141</td>
<td>0.5933</td>
</tr>
<tr>
<td></td>
<td>HPLFlowNet [21]</td>
<td>0.1169</td>
<td>0.4783</td>
<td>0.7776</td>
<td>0.4103</td>
<td>4.8055</td>
<td>0.5938</td>
</tr>
<tr>
<td></td>
<td>HCRF-Flow [39]</td>
<td>0.0531</td>
<td>0.8631</td>
<td>0.9444</td>
<td>0.1797</td>
<td>2.0700</td>
<td>0.8656</td>
</tr>
<tr>
<td></td>
<td>FLOT [56]</td>
<td>0.056</td>
<td>0.755</td>
<td>0.908</td>
<td>0.242</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>PV-RAFT [78]</td>
<td>0.0560</td>
<td>0.8226</td>
<td>0.9372</td>
<td>0.2163</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>PointPWC-Net [82]</td>
<td>0.0694</td>
<td>0.7281</td>
<td>0.8884</td>
<td>0.2648</td>
<td>3.0602</td>
<td>0.7673</td>
</tr>
<tr>
<td></td>
<td>PCFPWC-Net (ours)</td>
<td>0.0479</td>
<td>0.8659</td>
<td>0.9332</td>
<td>0.1731</td>
<td>1.7943</td>
<td>0.8924</td>
</tr>
</tbody>
</table>

The feature difference is computed by $\max_{x_i \in N(x_0)} \psi(x_i) - \min_{x_i \in N} \psi(x_i)$, where $\psi$ is the attention score. A larger difference indicates that some points are discarded from the neighborhood. A smaller difference indicates a nearly constant $\psi$ in the neighbourhood, where PointConvFormer would reduce to regular point convolution. We visualize the difference in Fig. 2(b), where it can be seen that larger differences happen mostly in object boundaries. For smooth surfaces and points from the same class, the difference of reweighted scores is low. This visualization further confirms that PointConvFormer is able to utilize feature differences to conduct neighborhood filtering.

4.5. Ablation Studies on Attention Types

In this section, we perform ablation experiment on different attention types in PointConvFormer. The ablation studies are conducted on the ScanNet [11] dataset with the PointConvFormer-Lite model structure at 5cm grid size. Many other ablation studies on different aspects of PointConvFormer are shown in the supplementary.

We compare the PointConvFormer with three different kind of attention types: Point Transformer [96], Dot-Product Attention [71], and no attention. The network shares the same backbone architecture, i.e. the same ResNet structure and the same number of layers. The dot product version of the attention is shown in equation:

$$X'_p = \sum_{p \in \mathcal{N}(p)} w(p_i - p) \varphi(\frac{1}{\sqrt{d}}\mathbf{q}(X_p)\mathbf{k}(X'_p))X_p,$$

(8)

Where $d$ is the dimension of the $\mathbf{q}$ and $\mathbf{k}$ transforms of the input feature. Theoretically, the computational cost and memory usage of eq. (8) are slightly smaller than the subtractive attention $\psi(X_i - X_j)$ we use in PointConvFormer.

The results in Table 5 show that the PointConvFormer attention is significantly better than Point Transformer attention as well as only using the viewpoint-invariant point convolution [41] without attention. The experiment results in Table 5 also show that the feature difference achieves better results than dot-product attention, which are also confirmed in [96]. The dot-product version has a bit more parameters due to the two MLPs for $Q$ and $K$ instead of a single one as in eq.(6). It in principle uses a bit less memory and computation during inference, however the savings is not very significant in practice due to the small neighborhood size of $K = 16$.

Table 5. Different Attention Types. With the same model architecture and training parameters we change the attention layer of the model. The experiment is performed at the 5cm voxel grid level with the lite model architecture.

<table>
<thead>
<tr>
<th>Attention Type</th>
<th># Params (M)</th>
<th>mIoU(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointTransformer Attention</td>
<td>2.9</td>
<td>71.6</td>
</tr>
<tr>
<td>No Attention (VI-PointConv only)</td>
<td>1.9</td>
<td>71.1</td>
</tr>
<tr>
<td>Dot-Product Attention</td>
<td>2.1</td>
<td>72.0</td>
</tr>
<tr>
<td>PointConvFormer-Lite</td>
<td>2.0</td>
<td>73.3</td>
</tr>
</tbody>
</table>

5. Conclusion

In this work, we propose a novel point cloud layer, PointConvFormer, which can be widely used in various computer vision tasks on point cloud data. Unlike traditional convolution where convolutional kernels are functions of the relative position, the convolutional weights of the PointConvFormer are modified by an attention score computed from feature differences and relative position. Hence, PointConvFormer incorporates benefits of attention models, which could help the network to focus on points with high feature correlation during feature encoding. Experiments on a number of point cloud tasks showed that PointConvFormer is the first point-based model that provides a better accuracy-speed tradeoff w.r.t. sparse 3D convolutional networks.

Acknowledgements

We thank Dr. Shuangfei Zhai for helpful discussions.
References


[25] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham.


[50] Iaroslav Melekhov, Aleksei Tuliipin, Torsten Sattler, Marc Pollefeys, Esa Rahtu, and Juho Kannala. Dgc-net: Dense ge-


21813