PSTNET: POINT SPATIO-TEMPORAL CONVOLUTION ON POINT CLOUD SEQUENCES

Anonymous authors
Paper under double-blind review

ABSTRACT

Point cloud sequences are irregular and unordered in the spatial dimension while exhibiting regularities and order in the temporal dimension. Therefore, existing grid based convolutional operations cannot be applied to spatio-temporal modeling of raw point cloud sequences. In this paper, we propose a generic 3D point spatio-temporal (PST) convolution to achieve informative representations of point cloud sequences. The proposed PST convolution first disentangles space and time in point cloud sequences. Then, a spatial convolution is employed to capture the local structure of points in the 3D space, and a temporal convolution is used to model the dynamics of the spatial regions along the time dimension. Furthermore, we incorporate the proposed PST convolution into a deep network, namely PSTNet, to extract features of 3D point cloud sequences in a hierarchical manner. Extensive experiments on widely-used 3D action recognition and 4D semantic segmentation datasets demonstrate the effectiveness of PSTNet to model point cloud sequences.

1 INTRODUCTION

Modern robotic and automatic driving systems usually employ real-time depth sensors, such as LiDAR, to perceive the geometric information of scenes accurately while being robust to different lighting conditions. A scene geometry is thus represented by a 3D point cloud, *i.e.*, a set of measured point coordinates $\{(x, y, z)\}$. Moreover, when RGB images are available, they are often used as additional features associated with 3D points to enhance the discriminativeness of point clouds. However, unlike conventional grid based videos, dynamic point clouds are irregular and unordered in the spatial dimension while points are not consistent and even flow in and out over time. Therefore, existing 3D convolutions on grid based videos (Tran et al., 2015; Carreira & Zisserman, 2017; Hara et al., 2018) are not suitable to model raw point cloud sequences, as shown in Fig. 1.

To model the dynamics of point clouds, one solution is converting point clouds to a sequence of 3D voxels, and then applying 4D convolutions (Choy et al., 2019) to the voxel sequence. However, directly performing convolutions on voxel sequences require a large amount of computation. Furthermore, quantization errors are inevitable during voxelization, which may restrict applications that require precise measurement of scene geometry. Another solution (Liu et al., 2019d) is extending static point cloud methods, such as PointNet++ (Qi et al., 2017b), to point cloud sequences by appending 1D temporal dimension to 3D points. However, simply concatenating coordinates and time together and treating point cloud sequences as unordered 4D point sets may not facilitate neural networks to understand dynamic point clouds.

In this paper, we propose a generic 3D point spatio-temporal (PST) convolution to process raw point cloud sequences. As dynamic point clouds are spatially irregular but ordered in the temporal dimension, we aim to decouple the spatial and temporal information to model point cloud sequences. Specifically, PST convolution consists of (i) a point based spatial convolution that models spatial structure of 3D points and (ii) a temporal convolution that captures the temporal dynamics of point cloud sequences. In this fashion, PST convolution significantly facilitates the modeling of dynamic point clouds and reduces the impacts of the spatial irregularity of points on temporal modeling.

To enhance the feature extraction ability, we incorporate the proposed PST convolution into a deep network, namely PSTNet. Moreover, we extend our 3D PST convolution to a transposed version to address point-level prediction tasks. Different from the convolutional version, the PST transposed

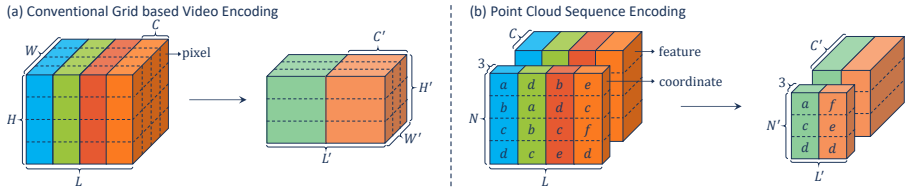


Figure 1: Illustration of grid based and point based convolutions on spatio-temporal sequences. **(a)** For a grid based video, each grid represents a feature of a pixel, where C, L, H and W denote the feature dimension, the number of frames, height and width, respectively. A 3D convolution encodes an input to an output of size $C' \times L' \times H' \times W'$. **(b)** A point cloud sequence consists of a coordinate part ($3 \times L \times N$) and a feature part ($C \times L \times N$), where N indicates the number of points in a frame. Our PST convolution encodes an input to an output composed of a coordinate tensor ($3 \times L' \times N'$) and a feature tensor ($C' \times L' \times N'$). Note that points in different frames are not consistent, and thus it is challenging to capture the spatio-temporal correlation.

convolution is designed to interpolate temporal dynamics and spatial features. Extensive experiments on widely-used 3D action recognition and 4D semantic segmentation datasets demonstrate the effectiveness of the proposed PST convolution and the superiority of PSTNet in modeling point clouds sequences. The contributions of this paper are fourfold:

- To the best of our knowledge, we are the first attempt to decompose spatial and temporal information in modeling raw point cloud sequences, and propose a generic point based convolutional operation, named PST convolution, to encode point cloud sequences.
- We propose a PST transposed convolution to decode raw point cloud sequences via interpolating the temporal dynamics and spatial feature for point-level prediction tasks.
- We construct deep networks based on the PST convolutions and transposed convolutions, dubbed PSTNets, to tackle sequence-level classification and point-level prediction tasks.
- Extensive experiments on four datasets indicate that our method remarkably improves the accuracy of 3D action recognition and 4D semantic segmentation.

2 Related Work

Learning Representations on Grid based Videos. Impressive progress has been made on generating compact and discriminative representations for RGB/RGBD videos due to the success of deep neural networks. For example, two-stream convolutional neural networks (Simonyan & Zisserman, 2014; Wang et al., 2016) adopt a spatial stream and an optical flow stream for video modeling. To summarize the temporal dependencies of videos, recurrent neural networks (Ng et al., 2015; Fan et al., 2018) are employed. In addition, by stacking multiple 2D frames into a 3D tensor, 3D convolutional neural networks (Tran et al., 2015; Carreira & Zisserman, 2017; Li et al., 2016; Tran et al., 2018; Hara et al., 2018; Ding et al., 2020) are widely used to learn spatio-temporal representations for videos, and achieve promising performance. For RGBD videos, state-of-the-art methods (Liu et al., 2019a; Hu et al., 2018) also adopt grid based methods to fuse RGB and depth information.

Static Point Cloud Processing. Static point cloud analysis has been widely investigated for many problems, such as classification, object part segmentation, scene semantic segmentation (Qi et al., 2017a;b; Li et al., 2018b; Wu et al., 2019; Thomas et al., 2019; Wang et al., 2019), reconstruction (Dai et al., 2017; Yu et al., 2018) and object detection (Chen et al., 2017; Qi et al., 2019). Most recent works aim to directly manipulate point sets without transforming coordinates into regular voxel grids. Since a point cloud is essentially a set of unordered points and invariant to permutations of its points, static point cloud processing methods mainly focus on designing effective point-based spatial correlation operations that do not rely on point orders.

Dynamic Point Cloud Modeling. Compared with static point cloud processing, dynamic point cloud modeling is a fairly new task but very important for intelligent systems to understand the dynamic world. Fast and Furious (FaF) (Luo et al., 2018) converts a point cloud in a frame into a bird's view voxel and then extracts features via 3D convolutions. MinkowskiNet (Choy et al., 2019) uses 4D Spatio-Temporal ConvNets on a 4D occupancy grid. MeteorNet (Liu et al., 2019d) extends 3D points to 4D points (x, y, z, t) and then appends a temporal dimension to PointNet++ (Qi et al., 2017b) to

process those 4D points. Different MeteorNet, our PSTNet decouples spatio-temporal information, and models temporal dynamics by leveraging the temporal regularity explicitly while addressing the spatial structure of point clouds by taking their spatial irregularity into account. Moreover, the grouping in MeteroNet only considers spatial radius. Without constructing temporal hierarchy with temporal kernels, pooling or stride, embedding points in a spatially local area along an entire sequence handicap capturing instant local dynamics of point clouds in MeteroNet. Without the temporal stride or pooling, MeteroNet needs to process all the frames in each layer and thus is not efficient. Besides, our PSTNet is the first attempt to construct convolution on raw point cloud sequences in a unified framework. Compared to MLP-based MeteroNet, our PSTNet is effective and efficient. 3DV (Wang et al., 2020) first integrates 3D motion information into a regular compact voxel set via temporal rank pooling and then applies PointNet++ to extract representations from the set for 3D action recognition. Besides, scene flow estimation (Liu et al., 2019c; Gu et al., 2019) aims to capture point motion between two frames. Niemeyer et al. (2019) learned a temporal and spatial vector field in which every point is assigned with a motion vector of space and time for 4D reconstruction. Prantl et al. (2020) learned stable and temporal coherent feature spaces for point-based super-resolution. CaSPR (Rempe et al., 2020) learns object-centric representations that can aggregate and encode spatio-temporal changes in object shape from point clouds, which can be used for reconstruction and camera pose estimation. Different from previous works, our PST convolution aims to directly capture the spatio-temporal structure in raw point cloud sequences in a convolutional manner.

3 Proposed Point Spatio-Temporal Convolutional Network

In this section, we first briefly review the grid based 3D convolution operation as our point spatio-temporal (PST) convolution is motivated by it. Then, we introduce how the proposed PST convolution extracts features from dynamic point clouds in detail. In order to address densely point prediction tasks, *i.e.*, semantic segmentation, we develop a PST transposed convolution. Finally, we incorporate our operations into deep hierarchical networks to address different dynamic point cloud tasks.

3.1 Motivation from Grid based 3D Convolution

The power of convolutional neural network (CNNs) comes from local structure modeling by convolutions and global representation learning via hierarchical architectures. Given an input feature map $\mathbf{F} \in \mathbb{R}^{C\times L\times H\times W}$, where C, L, H and W and denote the input feature dimension, length, height and width, 3D convolution is designed to capture the spatio-temporally local structure of $\mathbf{F}$, written as:

$$\mathbf{F}_t'^{(x,y)} = \sum_{k=-\lfloor l/2\rfloor}^{\lfloor l/2\rfloor} \sum_{i=-\lfloor h/2\rfloor}^{\lfloor h/2\rfloor} \sum_{j=-\lfloor w/2\rfloor}^{\lfloor w/2\rfloor} \mathbf{W}_k^{(i,j)} \cdot \mathbf{F}_{t+k}^{(x+i,y+j)}, \tag{1}$$

where $\mathbf{W} \in \mathbb{R}^{C'\times C\times l\times h\times w}$ is the convolution kernel, (l, h, w) is kernel size, C' represents the output feature dimension and $\cdot$ is matrix multiplication. The $\mathbf{W}_k^{(i,j)} \in \mathbb{R}^{C'\times C}$ denotes the weight at kernel position (k, i, j) and $\mathbf{F}_{t+k}^{(x+i,y+j)} \in \mathbb{R}^{C\times 1}$ denotes the feature of pixel at input position $(t + k, x + i, y + j)$. To effectively model local structure, CNNs usually adopt small kernel sizes, such as $(3, 3, 3)$, which are far less than input size. During performing convolution, stride (> 1) is used to subsample input feature maps so that following layers have larger receptive fields.

3.2 Point Spatio-Temporal (PST) Convolution

Let $\boldsymbol{P}_t \in \mathbb{R}^{3\times N}$ and $\boldsymbol{F}_t \in \mathbb{R}^{C\times N}$ denote the point coordinates and features of the t-th frame in a point cloud sequence, where N and C denote the number of points and feature channels, respectively. Given a point cloud sequence $\big([\boldsymbol{P}_1; \boldsymbol{F}_1], [\boldsymbol{P}_2; \boldsymbol{F}_2], \cdots, [\boldsymbol{P}_L; \boldsymbol{F}_L]\big)$, the proposed PST convolution will encode the sequence to $\big([\boldsymbol{P}_1'; \boldsymbol{F}_1'], [\boldsymbol{P}_2'; \boldsymbol{F}_2'], \cdots, [\boldsymbol{P}_{L'}'; \boldsymbol{F}_{L'}']\big)$, where L and L' indicate the number of frames, and $\boldsymbol{P}_t' \in \mathbb{R}^{3\times N'}$ and $\boldsymbol{F}_t' \in \mathbb{R}^{C'\times N'}$ represent the encoded coordinates and features.

3.2.1 Disentangling Space and Time in Point Cloud Sequences

Because point clouds are irregular and unordered, grid based 3D convolution cannot be directly applied to point cloud sequences. As point cloud sequences are spatially irregular and unordered

but temporally ordered, this motivates us to decouple these two dimensions in order to reduce the impacts of the spatial irregularity of points on temporal modeling. Moreover, the scales of spatial displacements and temporal differences in point cloud sequences may not be compatible. Treating them equally is not in favor of network optimization. By decoupling the spatio-temporal information in point cloud sequences, we do not only make l modeling easier but also significantly improve the ability to capture the temporal information. Therefore, our PST convolution is formulated as:

$$\begin{aligned} \boldsymbol{F}_t'^{(x,y,z)} &= \sum_{k=-\lfloor l/2 \rfloor}^{\lfloor l/2 \rfloor} \sum_{||(\delta_x,\delta_y,\delta_z)||\leq r} \mathbf{W}_k^{(\delta_x,\delta_y,\delta_z)} \cdot \boldsymbol{F}_{t+k}^{(x+\delta_x,y+\delta_y,z+\delta_z)}, \\ &= \sum_{k=-\lfloor l/2 \rfloor}^{\lfloor l/2 \rfloor} \sum_{||(\delta_x,\delta_y,\delta_z)||\leq r} \mathbf{T}_k^{(\delta_x,\delta_y,\delta_z)} \cdot \big(\mathbf{S}_k^{(\delta_x,\delta_y,\delta_z)} \cdot \boldsymbol{F}_{t+k}^{(x+\delta_x,y+\delta_y,z+\delta_z)}\big), \end{aligned} \tag{2}$$

where $(x, y, z) \in \boldsymbol{P}_t$, $(\delta_x, \delta_y, \delta_z)$ represents displacement and r is spatial search radius. The convolution kernel $\mathbf{W} \in \mathbb{R}^{C' \times C \times l} \times \mathbb{R}$ is decomposed into a spatial convolution kernel $\mathbf{S} \in \mathbb{R}^{C_m \times C \times l} \times \mathbb{R}$ and a temporal convolution kernel $\mathbf{T} \in \mathbb{R}^{C' \times C_m \times l} \times \mathbb{R}$ and C_m is the dimension of the intermediate feature. Moreover, because space and time are orthogonal and independent of each other, we further decompose spatial and temporal modeling as:

$$\boldsymbol{F}_t'^{(x,y,z)} = \sum_{k=-\lfloor l/2 \rfloor}^{\lfloor l/2 \rfloor} \mathbf{T}_k \cdot \sum_{||(\delta_x,\delta_y,\delta_z)||\leq r} \mathbf{S}^{(\delta_x,\delta_y,\delta_z)} \cdot \boldsymbol{F}_{t+k}^{(x+\delta_x,y+\delta_y,z+\delta_z)}, \tag{3}$$

where $\mathbf{S} \in \mathbb{R}^{C_m \times C} \times \mathbb{R}$ and $\mathbf{T} \in \mathbb{R}^{C' \times C_m \times l}$. The above decomposition can also be expressed as applying temporal convolution first and then spatial convolution to input features. However, doing so requires point tracking to capture point motion. As tracking points heavily relies on point colors (Liu et al., 2019d), it may fail to handle colorless point clouds. Thus, we opt to model the spatial structure of irregular points first, and then capture the temporal information from spatial regions. In this fashion, Eq. (3) can be rewritten as:

$$\boldsymbol{M}_t^{(x,y,z)} = \sum_{||(\delta_x,\delta_y,\delta_z)||\leq r} \mathbf{S}^{(\delta_x,\delta_y,\delta_z)} \cdot \boldsymbol{F}_t^{(x+\delta_x,y+\delta_y,z+\delta_z)}, \quad \boldsymbol{F}_t'^{(x,y,z)} = \sum_{k=-\lfloor l/2 \rfloor}^{\lfloor l/2 \rfloor} \mathbf{T}_k \cdot \boldsymbol{M}_{t+k}^{(x,y,z)}. \tag{4}$$

The spatial convolution captures spatially local structures of point clouds in 3D space, while the temporal convolution aims to describe the temporally local dynamics of point cloud sequences along the time dimension. In order to capture the distributions of neighboring points, it is required to learn a convolution kernel $\mathbf{S}$ for all displacements. However, this is impossible because point displacements are not discrete. To address this problem, we convert the kernel to a function of displacements, defined as:

$$\boldsymbol{M}_t^{(x,y,z)} = \sum_{||(\delta_x,\delta_y,\delta_z)||\leq r} f\big((\delta_x, \delta_y, \delta_z); \boldsymbol{\theta}\big) \cdot \boldsymbol{F}_t^{(x+\delta_x,y+\delta_y,z+\delta_z)}, \tag{5}$$

where $f : \mathbb{R}^{1\times 3} \rightarrow \mathbb{R}^{C_m \times C}$ is a function of $(\delta_x, \delta_y, \delta_z)$ parametrized by $\boldsymbol{\theta}$ to generate different $\mathbb{R}^{C_m \times C}$ according to different displacements. There can be many ways to implement the f function. In principle, the f function should be both computation-efficient and memory-efficient so that PST convolution is able to encode long sequences. Existing static point cloud convolution methods (Li et al., 2018b; Wu et al., 2019; Wang et al., 2019) usually contains multilayer perceptrons (MLPs), which are computation and memory consuming. In this paper, we design a lightweight implementation for f, which contains only a few parameters. Specifically, we further decompose the f function as $f\big((\delta_x, \delta_y, \delta_z); \boldsymbol{\theta}\big) = \boldsymbol{\theta}_d \cdot (\delta_x, \delta_y, \delta_z)^T \cdot \mathbf{1} \odot \boldsymbol{\theta}_s$, where $\boldsymbol{\theta} = [\boldsymbol{\theta}_d, \boldsymbol{\theta}_s]$, $\boldsymbol{\theta}_d \in \mathbb{R}^{C_m \times 3}$ is a displacement transform kernel, $\boldsymbol{\theta}_s \in \mathbb{R}^{C_m \times C}$ is a sharing kernel, $\mathbf{1} = (1, \cdots, 1) \in \mathbb{R}^{1\times C}$ is for broadcasting, and $\odot$ is element-wise product. The the sharing kernel $\boldsymbol{\theta}_s$ is to increase point feature dimension to improve the feature representation ability. The displacement kernel $\boldsymbol{\theta}_d$ is to capture spatial local structure based on displacements. In this fashion, f generates a unique spatial kernel for each displacement so that the lightweight spatial convolution is able to increase feature dimension and capture spatial local structure like conventional convolutions.

3.2.2 POINT TUBE

Grid based 3D convolutions can be easily performed on regular conventional videos by sliding along length, height and width. Because point cloud sequences are irregular and unordered in 3D space

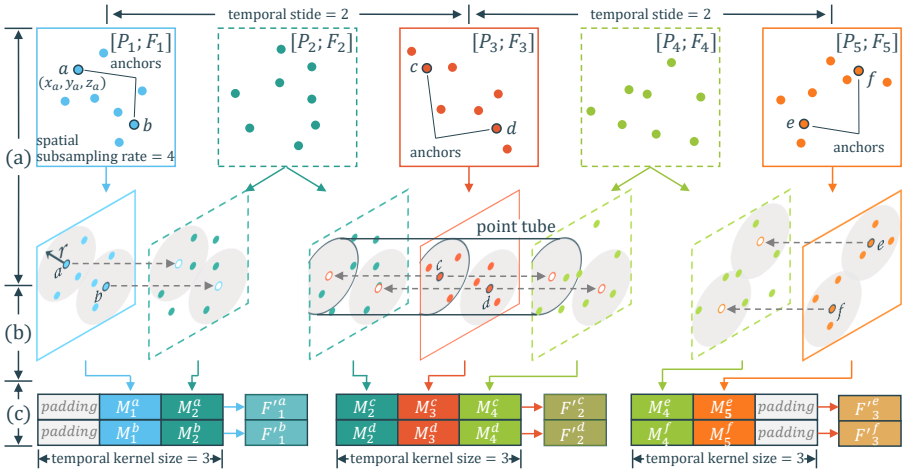


Figure 2: Illustration of the proposed point spatio-temporal (PST) convolution. The input contains $L = 5$ frames, with $N = 8$ points per frame. **(a)** Based on the temporal kernel size $l = 3$, temporal stride $s_t = 2$, and temporal padding $p = 1$, the 1st, 3rd, 5th frames are selected as temporal anchor frames. According to a spatial subsampling rate $s_s = 4$, 2 spatial anchor points are sampled by FPS in each anchor frame. The sampled anchor points are then transferred to the $\lfloor \frac{l}{2} \rfloor = 1$ nearest neighboring frames. A point tube is constructed with a spatial search radius r for the anchor points. **(b)** The spatial convolution encodes the local structure around each anchor point. **(c)** The temporal convolution encodes the l spatial features to a spatio-temporal feature.

and emerge inconsistently across different frames, it is challenging to perform convolution on them. To address this problem, we introduce a point tube to preserve the spatio-temporal structure. In contrast to pixel cubes in 3D convolution, in which pixels are distributed regularly, point tubes are dynamically generated according to input sequences so that dense areas have more tubes than sparse ones. Specifically, the point tube is constructed as follows:

Temporal anchor frame selection. For an entire sequence of point clouds, we need to select some anchor frames to generate our tubes. Temporal anchor frames in a point cloud sequence are automatically selected based on temporal kernel size (l), temporal stride (s_t) and temporal padding (p), where l is set to an odd number so that an anchor frame is located in the middle of a point tube. Moreover, we set $\lfloor \frac{l}{2} \rfloor \geq p$ to avoid selecting a padding frame as an anchor frame.

Spatial anchor point sampling. Once a temporal anchor frame is selected, we need to choose spatial anchor points that can represent the distribution of all the points in the frame. Given a spatial subsampling rate s_s, this operation aims to subsample N points to $N' = \lfloor \frac{N}{s_s} \rfloor$ points. We employ the farthest point sampling (FPS) (Qi et al., 2017b) to sample points in each anchor frame. Point tubes are generated according to sampled anchor points.

Transferring spatial anchor points. 3D convolutions can effectively capture local changes within a cube of a kernel size (l, h, w) rather than tracking a specific pixel. Following this idea, we propagate the positions of sampled anchor points to the neighboring frames without tracking, and they are regarded as the anchor points in those frames. Specifically, each anchor point is transferred to the $\lfloor \frac{l}{2} \rfloor$ nearest frames. The original and transferred anchor points form the central axis of a tube.

Spatial neighbor. This step aims to find the spatial neighbors of every anchor point in each frame for performing spatial convolution. As indicated in Eq. (5), radius neighborhood r is used to search neighbors within the tube slice, where local structure of points is depicted. Note that, padding is usually used in grid based convolution to align feature maps. However, our point based spatial convolution is not conducted on grids and thus spatial padding is not employed.

By performing PTS convolution along point tubes, our network is able to perceive the dynamic changes in local areas. The temporal kernel size l and spatial search radius r allow our PST convolution to capture temporal and spatial local structure, respectively. Frame subsampling (according to s_t) and point subsampling (according to s_s) make our PSTNet be both temporally and spatially hierarchical. Global movement can be summarized by merging the information from these tubes in the spatio-temporally hierarchical manner. An illustration of PST convolution is shown in Fig. 2.

3.3 POINT SPATIO-TEMPORAL TRANSPOSED CONVOLUTION

After PST convolution, the original point cloud sequence is subsampled. However, for point-level prediction tasks, we need to provide point features for all the original points. Thus, we develop a PST transposed convolution. Suppose $\big([\boldsymbol{P}'_1; \boldsymbol{F}'_1], [\boldsymbol{P}'_2; \boldsymbol{F}'_2], \cdots, [\boldsymbol{P}'_{L'}; \boldsymbol{F}'_{L'}]\big)$ is the encoded sequence of the original one $\big([\boldsymbol{P}_1; \boldsymbol{F}_1], [\boldsymbol{P}_2; \boldsymbol{F}_2], \cdots, [\boldsymbol{P}_L; \boldsymbol{F}_L]\big)$. PST transposed convolution propagates features $(\boldsymbol{F}'_1, \boldsymbol{F}'_2, \cdots, \boldsymbol{F}'_{L'})$ to the original point coordinates $(\boldsymbol{P}_1, \boldsymbol{P}_2, \cdots, \boldsymbol{P}_L)$, thus outputting new features $(\boldsymbol{F}''_1, \boldsymbol{F}''_2, \cdots, \boldsymbol{F}''_L)$, where $\boldsymbol{F}''_t \in \mathbb{R}^{C'' \times N}$ and C'' denotes the new feature dimension. To this end, PST transposed convolution first recovers the temporal length by a temporal transposed convolution:

$$\mathbf{T}' \cdot \boldsymbol{F}'^{(x,y,z)}_t = \big[\boldsymbol{M}'^{(x,y,z)}_{t-\lfloor l/2 \rfloor}, \cdots, \boldsymbol{M}'^{(x,y,z)}_{t+\lfloor l/2 \rfloor}\big], \tag{6}$$

where $\mathbf{T}' \in \mathbb{R}^{l \times C'_m \times C'}$ is the temporal transposed convolution kernel, and then increases the number of points by assigning features to original points. Inspired by (Qi et al., 2017b), the interpolated features are weighted by inverse distance between an original point and neighboring anchor points:

$$\boldsymbol{F}''^{(x,y,z)}_t = \boldsymbol{S}' \cdot \frac{\sum_{||(\delta_x,\delta_y,\delta_z)|| \leq r} w(\delta_x, \delta_y, \delta_z) \boldsymbol{M}'^{(x+\delta_x, y+\delta_y, z+\delta_z)}_t}{\sum_{||(\delta_x,\delta_y,\delta_z)|| \leq r} w(\delta_x, \delta_y, \delta_z)}, w(\delta_x, \delta_y, \delta_z) = \frac{1}{||(\delta_x, \delta_y, \delta_z)||}, \tag{7}$$

where $\boldsymbol{S}' \in \mathbb{R}^{C'' \times C'_m}$ is a sharing kernel to enhance the interpolated features and change the feature dimension from C'_m to C''. An illustration of PST transposed convolution is shown in Appendix A.

3.4 PSTNET ARCHITECTURES

PSTNet for 3D action recognition. We employ 6 PST convolution layers and a fully-connected (FC) layer for action recognition. In the 1st, 2nd, 4th, 6th layers, the spatial subsampling rate is set to 2 to halve the spatial resolution. In the 2nd, 4th, 6th layers, the temporal stride is set to 2 to halve the temporal resolution, and the temporal kernel size is set to 3 to capture temporal correlation. Temporal paddings are added when necessary. After the convolution layers, average and max poolings are used for spatial pooling and temporal pooling, respectively. The FC layer is then used as a classifier.

PSTNet for 4D semantic segmentation. We use 4 PST convolution layers and 4 PST transposed convolution layers for 4D semantic segmentation. The spatial subsampling rate is set to 4 to reduce the spatial resolution in each PST convolution layer. In the 3rd PST convolution and 2nd transposed convolution, the temporal kernel size is set to 3 to reduce and increase temporal dimension, respectively. Skip connections are added between the corresponding convolution layers and transposed convolution layers. After the last PST transposed convolution layer, a 1D convolution layer is appended for semantic predictions.

In addition, batch normalization and ReLU activation are inserted between the spatial convolution and the temporal convolution in PST convolution and PST transposed convolution, as well as between layers. Our PSTNet architectures are illustrated in Appendix B.

4 EXPERIMENTS

In this section, we conduct a sequence-level classification, *i.e.*, 3D action recognition, and a point-level classification, *i.e.*, 4D semantic segmentation, to demonstrate the effectiveness and superiority of PSTNet. Implementation details are provided in Appendix C. We will release our codes soon.

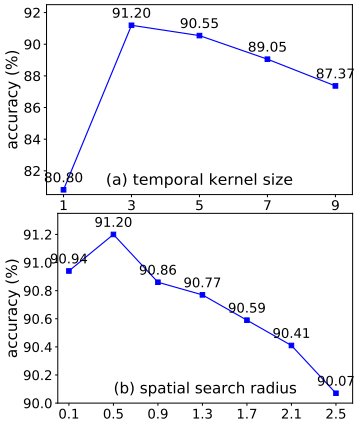
4.1 3D ACTION RECOGNITION

To show the effectiveness in sequence-level classification, we apply PSTNet to 3D action recognition. Following (Liu et al., 2019d; Wang et al., 2020), we sample 2,048 points for each frame. Point cloud sequences are split into multiple clips (with a fixed number of frames) as inputs. For training, sequence-level labels are used as clip-level labels. For evaluation, the mean of the clip-level predicted probabilities is used as the sequence-level prediction. Point colors are not used.

Table 1: Action recognition accuracy (%) on the MSR-Action3D dataset.

Method	Input	# Frames	Accuracy
Vieira *et al*. (Vieira et al., 2012)	depth	20	78.20
Kläser *et al*. (Kläser et al., 2008)	depth	18	81.43
Actionlet (Wang et al., 2012)	skeleton	all	88.21
PointNet++ (Qi et al., 2017b)	point	1	61.61
MeteorNet (Liu et al., 2019d)	point	4	78.11
		8	81.14
		12	86.53
		16	88.21
		24	88.50
PSTNets (ours)	point	4	81.14
		8	83.50
		12	87.88
		16	89.90
		24	**91.20**

Figure 3: Influence of temporal kernel size and spatial search radius on MSR-Action3D with 24 frames.



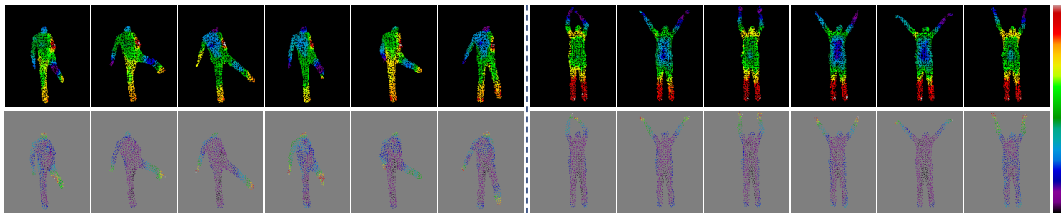


Figure 4: Visualization of PST convolution's output. Top: input point cloud sequences, where color encodes depth. Bottom: output of PST convolution, where brighter color indicates higher activation. Interestingly, PST convolution outputs high activation to salient motion. Best viewed in color.

4.1.1 MSR-ACTION3D

The MSR-Action3D (Li et al., 2010) dataset consists of 567 Kinect depth videos, with 20 action categories and 23K frames in total. We use the same training/test split as previous works (Wang et al., 2012; Liu et al., 2019d). We conduct experiments with 10 times and report the mean.

Comparison with the state-of-the-art. We compare our PSTNet with skeleton-based, depth-based and point-based 3D action recognition methods on this dataset. The performance comparisons are reported in Table 1. The proposed PSTNet significantly outperforms all the state-of-the-art methods, demonstrating the superiority of our PST convolution on feature extraction. Moreover, when encoding long point sequences, our spatio-temporally hierarchical PSTNet is more effective than the spatially hierarchical MeteorNet. From 16 frames to 24 frames, MeteorNet only achieves a slight improvement of 0.29% , while our method increases the accuracy by 1.30%.

What does PST convolution learn? To investigate what PST convolution learns, we visualize the output of the middle layer (*i.e*., the 3rd layer) in Fig. 4 (other layers' are shown in Appendix I). As ReLU is employed as the activation function, larger values indicate higher activation. As expected, PSTNet outputs higher activation on moving areas. This demonstrates that our PSTNet captures the most informative clues in action reasoning. We also visualize the learned features via t-SNE to show that PSTNet feature is impact and discriminative in Appendix F.

4.1.2 NTU RGB+D 60 AND NTU RGB+D 120

The NTU RGB+D 60 (Shahroudy et al., 2016) is the second largest dataset for 3D action recognition. It consists of 56K videos, with 60 action categories and 4M frames in total. The videos are captured using Kinect v2, with 3 cameras and 40 subjects (performers). The dataset defines two types of evaluation, *i.e*., cross-subject and cross-view. The cross-subject evaluation splits the 40 performers into training and test groups. Each group consists of 20 performers. The cross-view evaluation uses all the samples from camera 1 for testing and samples from cameras 2 and 3 for training.

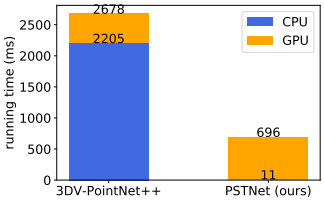
Table 2: Action recognition accuracy (%) on the NTU RGB+D 60 and NTU RGB+D 120 datasets.

Method	Input	NTU RGB+D 60		NTU RGB+D 120	
		Subject	View	Subject	Setup
SkeleMotion (Caetano et al., 2019)	skeleton	69.6	80.1	67.7	66.9
GCA-LSTM (Liu et al., 2017)	skeleton	74.4	82.8	58.3	59.3
FSNet (Liu et al., 2019b)	skeleton	-	-	59.9	62.4
Two Stream Attention LSTM (Liu et al., 2018)	skeleton	77.1	85.1	61.2	63.3
Body Pose Evolution Map (Liu & Yuan, 2018)	skeleton	-	-	64.6	66.9
AGC-LSTM (Si et al., 2019)	skeleton	89.2	95.0	-	-
AS-GCN (Li et al., 2019)	skeleton	86.8	94.2	-	-
VA-fusion (Zhang et al., 2019)	skeleton	89.4	95.0	-	-
2s-AGCN (Shi et al., 2019b)	skeleton	88.5	95.1	-	-
DGNN (Shi et al., 2019a)	skeleton	89.9	96.1	-	-
HON4D (Oreifej & Liu, 2013)	depth	30.6	7.3	-	-
SNV (Yang & Tian, 2014)	depth	31.8	13.6	-	-
HOG^2 (Ohn-Bar & Trivedi, 2013)	depth	32.2	22.3	-	-
Li *et al.* (Li et al., 2018a)	depth	68.1	83.4	-	-
Wang *et al.* (Wang et al., 2018)	depth	87.1	84.2	-	-
MVDI (Xiao et al., 2019)	depth	84.6	87.3	-	-
NTU RGB+D 120 Baseline (Liu et al., 2019a)	depth	-	-	48.7	40.1
PointNet++ (appearance) (Qi et al., 2017b)	point	80.1	85.1	72.1	79.4
3DV (motion) (Wang et al., 2020)	voxel	84.5	95.4	76.9	92.5
3DV-PointNet++ (Wang et al., 2020)	voxel + point	88.8	96.3	82.4	93.5
PSTNet (ours)	point	**90.5**	**96.5**	**87.0**	**93.8**

Table 3: Semantic segmentation results on the Synthia 4D dataset.

Method	Input	#Frms	#Params (M)	mIoU (%)
3D MinkNet14 (Choy et al., 2019)	voxel	1	19.31	76.24
4D MinkNet14 (Choy et al., 2019)	voxel	3	23.72	77.46
PointNet++ (Qi et al., 2017b)	point	1	0.88	79.35
MeteorNet (Liu et al., 2019d)	point	3	1.78	81.80
PSTNet ($l = 1$)	point	3	1.42	80.79
PSTNet ($l = 3$)	point	3	1.67	**82.24**

Figure 5: Comparison of recognition running time per video on NTU RGB+D 60.



The NTU RGB+D 120 (Liu et al., 2019a) dataset, the largest dataset for 3D action recognition, is an extension of NTU RGB+D 60. It consists of 114K videos, with 120 action categories and 8M frames in total. The videos are captured with 106 performers and 32 collection setups (locations and backgrounds). This dataset is challenging as it contains not only single-person actions but also multiple-person interactions. Besides cross-subject evaluation, the dataset defines a new evaluation setting, *i.e.*, cross-setup, where 16 setups are used for training, and the others are used for testing.

Comparison with state-of-the-art methods. As indicated in Table 2, PSTNet outperforms all the other approaches in all evaluation settings. Particularly, as indicated by the cross-setup evaluation on NTU RGB+D 120, PSTNet outperforms the second best 3DV-PointNet++ (Wang et al., 2020) by 4.6%. Moreover, compared to 3DV that extracts motion from voxels, PSTNet directly models the dynamic information of raw point cloud sequences and thus is efficient.

Computational efficiency. We provide a running time comparison with the second best 3DV-PointNet++ (Wang et al., 2020). The average running time per video is shown in Fig. 5. Experiments are conducted using 1 Nvidia RTX 2080Ti GPU on NTU RGB+D 60. Because the 3DV-PointNet++ method is not highly parallel and many time-consuming operations need to be performed on CPU, it takes much more CPU time than GPU time. Impressively, compared to 3DV-PointNet++, PSTNet reduces running time by about 2s, manifesting that PSTNet is efficient.

4.2 4D Semantic Segmentation

To demonstrate that our PST convolution can be generalized to point-level dense prediction tasks, we apply PSTNet to 4D semantic segmentation. Following the works (Choy et al., 2019; Liu et al.,

2019d), we conduct experiments on video clips with length of 3 frames. The widely-used standard mean Intersection over Union (mIoU) (Li et al., 2018b; Wu et al., 2019; Ding et al., 2020) is adopted as our evaluation metric.

Synthia 4D (Choy et al., 2019) uses the Synthia dataset (Ros et al., 2016) to create 3D video sequences. The Synthia 4D dataset includes 6 sequences of driving scenarios, where both objects and cameras are moving. Each sequence consists of 4 stereo RGB-D images taken from the top of a moving car. Following (Liu et al., 2019d), we reconstruct 3D point cloud sequences from RGB and depth videos, and use the same training/validation/test split, with 19,888, 815 and 1,886 frames, respectively.

We compare PSTNet with the state-of-the-art voxel-based and point-based methods on this dataset. Note that, although semantic segmentation can be achieved from a single frame, exploring temporal consistency would facilitate exploring the structure of scenes, thus improving segmentation accuracy and robustness to noise. As seen in Table 4, PSTNet ($l = 3$) exploits the temporal information and outperforms the state-of-the-art. Moreover, due to decoupling spatio-temporal modeling, our method saves 0.11M, relative 6% of parameters compared to the second best method MeteorNet (Liu et al., 2019d). We visualize a few segmentation examples in Appendix J.

4.3 Ablation Study

Clip length. Usually, information is not equally distributed in sequences along time. Short point cloud clips may miss key frames and thus confuse models as noise. Therefore, as shown in Table 1, increasing clip length (*i.e.*, the number of frames) benefits models for action recognition.

Temporal kernel size. The temporal kernel size l controls the temporal dynamics modeling of point cloud sequences. Fig. 3(a) shows the accuracy on MSR-Action3D with different l. (a) When l is set to 1, temporal correlation is not captured. However, PSTNet can still observe 24 frames and the pooling operation allows PSTNet to capture the pose information of an entire clip. Moreover, some actions (*e.g.*, "golf swing") can be easily recognized by a certain pose, and thus PSTNet with $l = 1$ can still achieve satisfied accuracy. (b) When l is greater than 1, PSTNet models temporal dynamics and therefore improves accuracy on actions that rely on motion or trajectory reasoning (*e.g.*, "draw x", "draw tick" and "draw circle"). (c) When l is greater than 3, the accuracy decreases. This mainly depends on motion in sequences. Because most actions in MSR-Action3D are fast (*e.g.*, "high arm wave"), using smaller temporal kernel size facilitates capturing fast motion, and long-range temporal dependencies will be captured in high-level layers. Since we aim to present generic point based convolution operations, we do not tune the kernel size for each action but use the same size. The influence of temporal kernel size on different actions is further discussed in Appendix E. Furthermore, when PSTNet is used in semantic segmentation, we observe that PSTNet ($l = 3$) improves mIoU by 1.45% compared to PSTNet ($l = 1$) without using temporal modeling (shown in Table 4).

Spatial search radius. The spatial search radius r controls the range of the spatial structure to be modeled. As shown in Fig. 3(b), using too small r cannot capture sufficient structure information while using large r will decrease the discriminativeness of local structure for temporal modeling.

5 Conclusion

In this paper, we propose an effective 3D point spatio-temporal (PST) convolution to learn informative representations from point cloud sequences. Considering 3D points are spatially irregular and temporally ordered, we decouple the feature extraction in two successive steps, *i.e.*, a spatial correlation operation and a temporal convolution operation. Then, we develop a PST transposed convolution for point-level dense prediction tasks. We also demonstrate that by incorporating the proposed convolutions into deep network, dubbed PSTNets, our method is competent to address various point-based tasks. Extensive experiments demonstrate that our PSTNet significantly improves the recognition and segmentation performance by effectively modeling point cloud sequences. However, as our method only exploits local operations and does not explicitly capture global dependencies, this may limit the ability of understanding scenes or spotting periodic actions. A potential improvement is to adopt non-local techniques to enhance the feature representations in spatial and temporal dimensions, which can be studied in the future.

REFERENCES

Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, 1992.

Carlos Caetano, Jessica Sena de Souza, François Brémond, Jefersson A. dos Santos, and William Robson Schwartz. Skelemotion: A new representation of skeleton joint sequences based on motion information for 3d action recognition. In *AVSS*, 2019.

João Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. In *CVPR*, 2017.

Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *CVPR*, 2017.

Christopher B. Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *CVPR*, 2019.

Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas A. Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, 2017.

Xinpeng Ding, Nannan Wang, Xinbo Gao, Jie Li, Xiaoyu Wang, and Tongliang Liu. Weakly supervised temporal action localization with segment-level labels. *arXiv*, 2007.01598, 2020.

Hehe Fan and Yi Yang. Pointrnn: Point recurrent neural network for moving point cloud processing. *arXiv*, 1910.08287, 2019.

Hehe Fan, Zhongwen Xu, Linchao Zhu, Chenggang Yan, Jianjun Ge, and Yi Yang. Watching a small portion could be as good as watching all: Towards efficient video classification. In *IJCAI*, 2018.

Xiuye Gu, Yijie Wang, Chongruo Wu, Yong Jae Lee, and Panqu Wang. Hplflownet: Hierarchical permutohedral lattice flownet for scene flow estimation on large-scale point clouds. In *CVPR*, 2019.

Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet? In *CVPR*, 2018.

Jianfang Hu, Wei-Shi Zheng, Jiahui Pan, Jianhuang Lai, and Jianguo Zhang. Deep bilinear learning for RGB-D action recognition. In *ECCV*, 2018.

Alexander Kläser, Marcin Marszalek, and Cordelia Schmid. A spatio-temporal descriptor based on 3d-gradients. In *BMVC*, 2008.

Junnan Li, Yongkang Wong, Qi Zhao, and Mohan S. Kankanhalli. Unsupervised learning of view-invariant action representations. In *NeurIPS*, 2018a.

Maosen Li, Siheng Chen, Xu Chen, Ya Zhang, Yanfeng Wang, and Qi Tian. Actional-structural graph convolutional networks for skeleton-based action recognition. In *CVPR*, 2019.

Wanqing Li, Zhengyou Zhang, and Zicheng Liu. Action recognition based on a bag of 3d points. In *CVPR Workshops*, 2010.

Xiaoyan Li, Tongliang Liu, Jiankang Deng, and Dacheng Tao. Video face editing using temporal-spatial-smooth warping. *ACM Trans. Intell. Syst. Technol.*, 7(3):32:1–32:28, 2016. doi: 10.1145/2819000.

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

Jun Liu, Gang Wang, Ping Hu, Ling-Yu Duan, and Alex C. Kot. Global context-aware attention LSTM networks for 3d action recognition. In *CVPR*, 2017.

Jun Liu, Gang Wang, Ling-Yu Duan, Kamila Abdiyeva, and Alex C. Kot. Skeleton-based human action recognition with global context-aware attention LSTM networks. *IEEE Trans. Image Processing*, 27(4):1586–1599, 2018. doi: 10.1109/TIP.2017.2785279.

Jun Liu, Amir Shahroudy, Mauricio Perez, Gang Wang, Ling-Yu Duan, and Alex C. Kot. Ntu rgb+d 120: A large-scale benchmark for 3d human activity understanding. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2019a. doi: 10.1109/TPAMI.2019.2916873.

Jun Liu, Amir Shahroudy, Gang Wang, Ling-Yu Duan, and Alex C. Kot. Skeleton-based online action prediction using scale selection network. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2019b. doi: 10.1109/TPAMI.2019.2898954.

Mengyuan Liu and Junsong Yuan. Recognizing human actions as the evolution of pose estimation maps. In *CVPR*, 2018.

Xingyu Liu, Charles R. Qi, and Leonidas J. Guibas. Flownet3d: Learning scene flow in 3d point clouds. In *CVPR*, 2019c.

Xingyu Liu, Mengyuan Yan, and Jeannette Bohg. Meteornet: Deep learning on dynamic 3d point cloud sequences. In *ICCV*, 2019d.

Wenjie Luo, Bin Yang, and Raquel Urtasun. Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net. In *CVPR*, 2018.

Nikolaus Mayer, Eddy Ilg, Philip Häusser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *CVPR*, 2016.

Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *CVPR*, 2015.

Joe Yue-Hei Ng, Matthew J. Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *CVPR*, 2015.

Michael Niemeyer, Lars M. Mescheder, Michael Oechsle, and Andreas Geiger. Occupancy flow: 4d reconstruction by learning particle dynamics. In *ICCV*, 2019.

Eshed Ohn-Bar and Mohan M. Trivedi. Joint angles similarities and HOG2 for action recognition. In *CVPR Workshops*, 2013.

Omar Oreifej and Zicheng Liu. HON4D: histogram of oriented 4d normals for activity recognition from depth sequences. In *CVPR*, 2013.

Lukas Prantl, Nuttapong Chentanez, Stefan Jeschke, and Nils Thuerey. Tranquil clouds: Neural networks for learning temporally coherent features in point clouds. In *ICLR*, 2020.

Charles R. Qi, Or Litany, Kaiming He, and Leonidas J. Guibas. Deep hough voting for 3d object detection in point clouds. In *ICCV*, 2019.

Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 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 *NeurIPS*, 2017b.

Davis Rempe, Tolga Birdal, Yongheng Zhao, Zan Gojcic, Srinath Sridhar, and Leonidas J. Guibas. Caspr: Learning canonical spatiotemporal point cloud representations. In *NIPS*, 2020.

Germán Ros, Laura Sellart, Joanna Materzynska, David Vázquez, and Antonio M. López. The SYNTHIA dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *CVPR*, 2016.

Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. NTU RGB+D: A large scale dataset for 3d human activity analysis. In *CVPR*, 2016.

Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. Skeleton-based action recognition with directed graph neural networks. In *CVPR*, 2019a.

Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. Two-stream adaptive graph convolutional networks for skeleton-based action recognition. In *CVPR*, 2019b.

Chenyang Si, Wentao Chen, Wei Wang, Liang Wang, and Tieniu Tan. An attention enhanced graph convolutional LSTM network for skeleton-based action recognition. In *CVPR*, 2019.

Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *NeurIPS*, 2014.

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

Du Tran, Lubomir D. Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, 2015.

Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. In *CVPR*, 2018.

Antônio Wilson Vieira, Erickson R. Nascimento, Gabriel L. Oliveira, Zicheng Liu, and Mario Fernando Montenegro Campos. STOP: space-time occupancy patterns for 3d action recognition from depth map sequences. In *CIARP*, 2012.

Jiang Wang, Zicheng Liu, Ying Wu, and Junsong Yuan. Mining actionlet ensemble for action recognition with depth cameras. In *CVPR*, 2012.

Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *ECCV*, 2016.

Pichao Wang, Wanqing Li, Zhimin Gao, Chang Tang, and Philip O. Ogunbona. Depth pooling based large-scale 3-d action recognition with convolutional neural networks. *IEEE Trans. Multimedia*, 20(5):1051–1061, 2018. doi: 10.1109/TMM.2018.2818329.

Yancheng Wang, Yang Xiao, Fu Xiong, Wenxiang Jiang, Zhiguo Cao, Joey Tianyi Zhou, and Junsong Yuan. 3dv: 3d dynamic voxel for action recognition in depth video. In *CVPR*, 2020.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph CNN for learning on point clouds. *ACM Trans. Graph.*, 38(5):146:1–146:12, 2019. doi: 10.1145/3326362.

Wenxuan Wu, Zhongang Qi, and Fuxin Li. Pointconv: Deep convolutional networks on 3d point clouds. In *CVPR*, 2019.

Yang Xiao, Jun Chen, Yancheng Wang, Zhiguo Cao, Joey Tianyi Zhou, and Xiang Bai. Action recognition for depth video using multi-view dynamic images. *Inf. Sci.*, 480:287–304, 2019. doi: 10.1016/j.ins.2018.12.050.

Xiaodong Yang and Yingli Tian. Super normal vector for activity recognition using depth sequences. In *CVPR*, 2014.

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

Pengfei Zhang, Cuiling Lan, Junliang Xing, Wenjun Zeng, Jianru Xue, and Nanning Zheng. View adaptive neural networks for high performance skeleton-based human action recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 41(8):1963–1978, 2019. doi: 10.1109/TPAMI.2019.2896631.

A PST TRANSPOSED CONVOLUTION

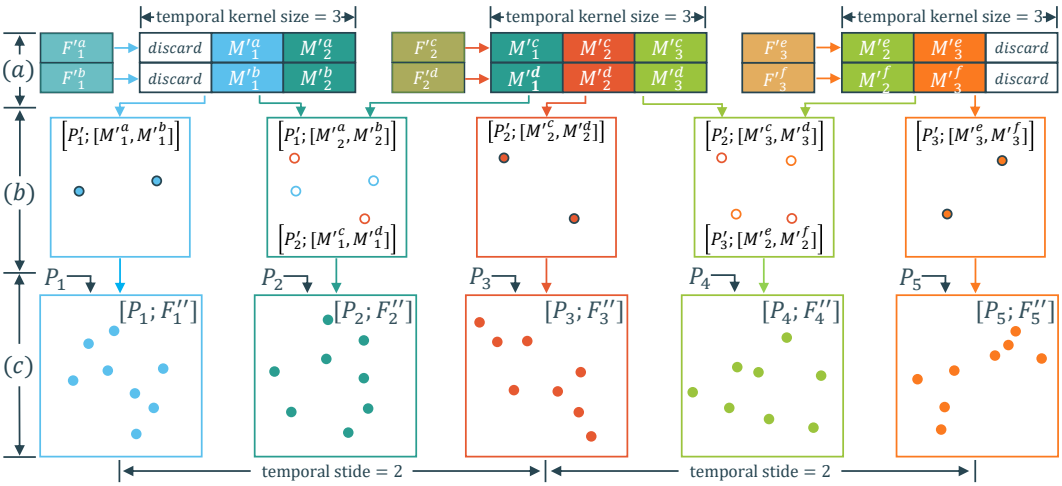


Figure 6: Illustration of the proposed PST transposed convolution. The input contains $L' = 3$ frames, with $N' = 2$ points per frame. The PST transposed convolution is to generate new original point features for the original coordinate sequence, which contains $L = 5$ frames, with $N = 8$ points per frame. The temporal kernel size l is 3 and the temporal stride is s_t is 2. (a) Temporal transposed convolution. (b) Temporal interpolation. (c) Spatial interpolation.

We illustrate an example of PST transposed convolution in Fig. 6. Given a convolved sequence $\big([\boldsymbol{P}'_1; \boldsymbol{F}'_1], [\boldsymbol{P}'_2; \boldsymbol{F}'_2], [\boldsymbol{P}'_{3'}; \boldsymbol{F}'_{3'}]\big)$, and an original coordinate sequence $\big(\boldsymbol{P}_1, \boldsymbol{P}_2, \boldsymbol{P}_3, \boldsymbol{P}_4, \boldsymbol{P}_5\big)$, PST transposed convolution aims to generate new features $(\boldsymbol{F}''_1, \boldsymbol{F}''_2, \boldsymbol{F}''_3, \boldsymbol{F}''_4, \boldsymbol{F}''_5)$ according to original point coordinates. First, according to the temporal kernel size $l = 3$, each input point feature is decoded to 3 features by a temporal transposed convolution. Second, 5 frames are interpolated in accordance with a temporal stride $s_t = 2$, and the input points with the decoded features are assigned to interpolated frames. In this way, the temporal cross-correlation is constructed. Third, given the original point coordinates (*e.g.*, $\boldsymbol{P}_1$) and an interpolated frame (*e.g.*, $\big[\boldsymbol{P}'_1; [\boldsymbol{M}'^{a}_1, \boldsymbol{M}'^{b}_1]\big]$), we generate features for all the original points (*e.g.*, $\boldsymbol{F}''_1$).

B PSTNET ARCHITECTURES

B.1 PSTNET FOR 3D ACTION RECOGNITION

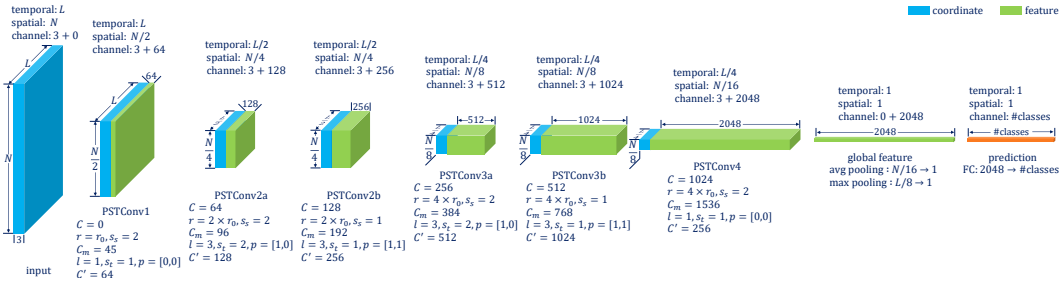


Figure 7: Hierarchical PSTNet architecture for 3D action recognition.

As shown in Fig. 7, the architecture of PSTNet for 3D action recognition consists of 6 PST convolutions, *i.e.*, PSTConv1, PSTConv2a, PSTConv2b, PSTConv3a, PSTConv3b, PSTConv4, and a fully-connected (FC) layer. In PSTConv1, PSTConv2a, PSTConv3a and PSTConv4, the spatial subsampling rate is set to 2 to halve the spatial resolution. In PSTConv2a and PSTConv3a, the temporal stride is set to 2 to halve the temporal resolution. The r_o denotes the initial spatial search

radius, which progressively increases to grow spatial receptive fields along the network. The temporal padding p is split as $[p_1, p_2]$, where p_1 and p_2 denote the left padding and the right padding, respectively. After these convolution layers, average and max poolings are used for spatial pooling and temporal pooling, respectively. The FC layer is then used as a classifier.

B.2 PSTNet for 4D Semantic Segmentation

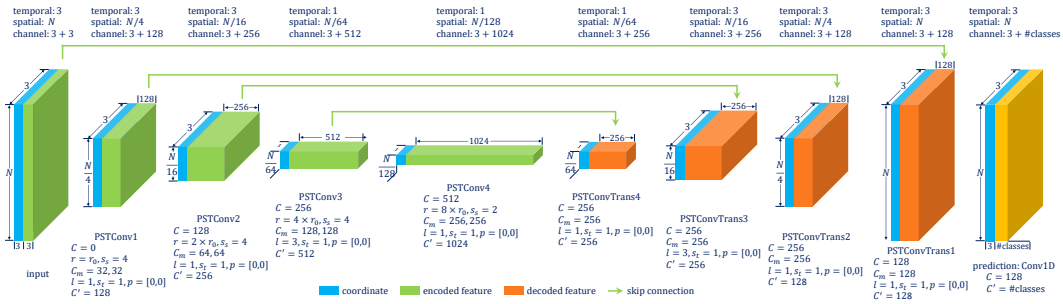


Figure 8: Hierarchical PSTNet architecture for 4D semantic segmentation.

As shown in Fig. 8, we employ 4 PST convolutional layers, *i.e.*, PSTConv1, PSTConv2, PSTConv3, and PSTConv4, and 4 PST transposed convolutional layers, *i.e.*, PSTConvTrans1, PSTConvTrans2, PSTConvTrans3, and PSTConvTrans4 for 4D Semantic Segmentation. The spatial subsampling rate is set to 4 to reduce the spatial resolution in each PST convolution layer. Following (Choy et al., 2019; Liu et al., 2019d), we conduct 4D semantic segmentation on clips with the length of 3 frames, *i.e.*, L is fixed to 3. In PSTConv3 and PSTConvTrans2, the temporal kernel size is set to 3. Skip connections are added from input to PSTConvTrans1, PSTConv1 to PSTConvTrans2, PSTConv2 to PSTConvTrans3, and PSTConv3 to PSTConvTrans4. After PSTConvTrans4, a 1D convolution layer is appended for semantic predictions.

C Implementation Details

To build a unified network and train the network with mini-batch, the number of spatial neighbors needs to be fixed. We follow existing point-based works (Qi et al., 2017b; Li et al., 2018b; Wu et al., 2019; Liu et al., 2019d) to randomly sample a fixed number of neighbors. For 3D action recognition, the number is set to 9. For 4D semantic segmentation, we follow (Liu et al., 2019d) to set the number to 32. If the actual number of neighbors of a point is less than the set one, we randomly repeat some neighbors.

We train our models for 35 epochs with the SGD optimizer. Learning rate is set to 0.01, and decays with a rate of 0.1 at the 10th epoch and the 20th epoch, respectively.

MSR-Action3D (Li et al., 2010). Following (Liu et al., 2019d), batch size is set to 16 and frame sampling stride are set to and 1, respectively. We set the initial spatial search radius r_o to 0.5.

NTU RGB+D 60 /120 (Shahroudy et al., 2016; Liu et al., 2019a). Following (Wang et al., 2020), batch size is set to 32. We set the frame sampling stride and the initial spatial search radius r_o to 2 and 0.1, respectively.

Synthia 4D (Choy et al., 2019). Following (Liu et al., 2019d), batch size is set to 16 and frame sampling stride are set to 12 and 1, respectively. We set the spatial search radius r_o to 0.9.

D Temporal Anchor Frame Selection and Temporal Interpolation

In this section, we provide the derivation about temporal anchor frame selection, temporal interpolation, and related constraints between sequence length, temporal kernel size, temporal stride and temporal padding.

D.1 TEMPORAL ANCHOR FRAME SELECTION FOR PST CONVOLUTION

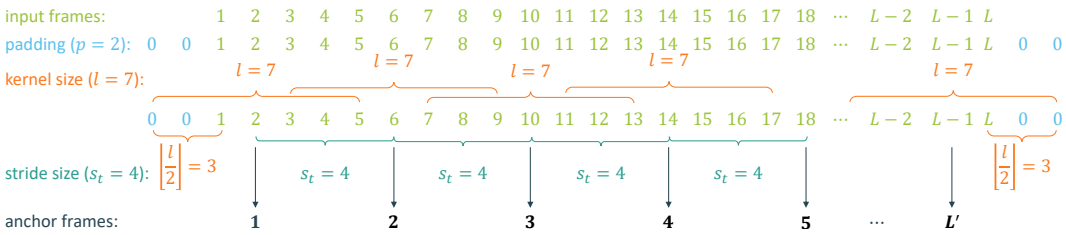


Figure 9: Illustration of temporal anchor frame selection for PST convolution.

Given a point cloud sequence with length L, after adding p paddings at the beginning and end, we obtain a new sequence with length $L + 2p$. In particular, the t-th frame in the input sequence become the $(t + p)$-th frame in the padded sequence. To perform a temporal convolution with kernel size l, the anchor frames should be selected from the $(\lfloor \frac{l}{2} \rfloor + 1)$-th frame in the padded sequence. Therefore, the first anchor frame should be the $(\lfloor \frac{l}{2} \rfloor + 1 - p)$-th frame in the input sequence. Because $L \geq 1$, the $\lfloor \frac{l}{2} \rfloor + 1 - p$ should be greater than or equal to 1. Then, we have $\lfloor \frac{l}{2} \rfloor \geq p$. As mentioned in the main paper, this also avoids selecting a padding frame as an anchor frame.

Let s_t be the temporal stride. The t'-th anchor frame is the $\big(\lfloor \frac{l}{2} \rfloor + 1 - p + (t' - 1) \times s_t\big)$-th frame in the input sequence. For example, as shown in Fig. 9, the 3rd anchor frame is the $\big(\lfloor \frac{7}{2} \rfloor + 1 - 2 + (3 - 1) \times 4\big) = 10$-th frame in the input sequence. The last anchor frame is the $(L + p - \lfloor \frac{l}{2} \rfloor)$-th frame in the original sequence.

Suppose the output sequence is with length L', then we have $\lfloor \frac{l}{2} \rfloor + 1 - p + (L' - 1) \times s_t = L + p - \lfloor \frac{l}{2} \rfloor$. Therefore, we have $L' = \frac{L + 2 \times p - 2 \times \lfloor \frac{l}{2} \rfloor - 1}{s_t} + 1$. Recall that l is set to an odd number so that an anchor frame is located in the middle of a point tube, therefore $-2 \times \lfloor \frac{l}{2} \rfloor - 1$ is equal to $-l$. Then, we have $L' = \frac{L + 2 \times p - l}{s_t} + 1$, where $L + 2 \times p - l$ should be exactly divisible by s_t.

In summary, we have the following conclusions:

- l is odd.
- $\lfloor \frac{l}{2} \rfloor \geq p$.
- $L + 2 \times p - l$ is exactly divisible by s_t.
- The t'-th anchor frame is the $\big(\lfloor \frac{l}{2} \rfloor + 1 - p + (t' - 1) \times s_t\big)$-th frame in the input sequence.
- The length of output sequence L' is $\frac{L + 2 \times p - l}{s_t} + 1$.

D.2 TEMPORAL INTERPOLATION FOR PST TRANSPOSED CONVOLUTION

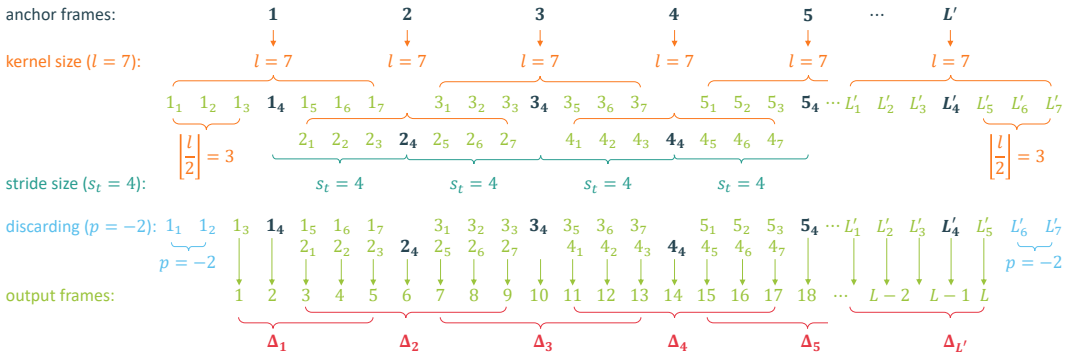
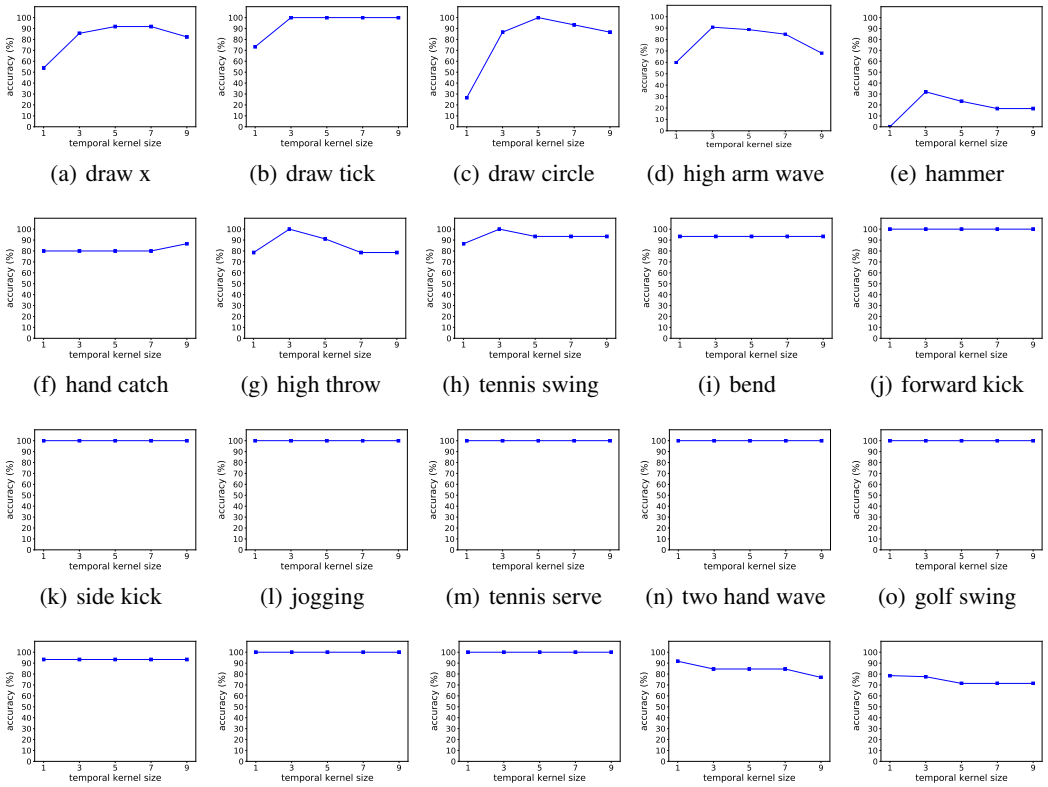


Figure 10: Illustration of temporal interpolation for PST transposed convolution.



(a) draw x (b) draw tick (c) draw circle (d) high arm wave (e) hammer

(f) hand catch (g) high throw (h) tennis swing (i) bend (j) forward kick

(k) side kick (l) jogging (m) tennis serve (n) two hand wave (o) golf swing

(p) pick up & throw (q) horizont arm wave (r) forward punch (s) hand clap (t) side-boxing

Figure 11: Influence of temporal kernel size on different actions of MSR-Action3D (Li et al., 2010).

To interpolate an encoded sequence (anchor frames) with length L' to the output sequence with length L, we first recover the positions of anchor frames in the output sequence. Because each anchor frame generates l frames (l is the kernel size of temporal transposed convolution), the first anchor frame is the $(\lfloor \frac{l}{2} \rfloor + p + 1)$-th frame in the output sequence, where p is padding size and usually less than 0 to discard additional generated frames by the first and last anchor frames. For example, as shown in Fig. 10, the 1st anchor frame generates $1_1, 1_2, 1_3, 1_4, 1_5, 1_6, 1_7$ frames by the temporal transposed convolution with $l = 7$. The 1_1 and 1_2 frames are discarded because $p = -2$. Therefore, 1_3 and 1_4 become the first frame and the second frame in the output sequence, respectively. To avoid discarding anchor frames, we set $\lfloor \frac{l}{2} \rfloor \geq -p$. Because middle generated frame 1_4 is corresponding to the 1st anchor frame, the 1st anchor frame is corresponding to the second output frame. Let s_t be the temporal stride. For the t'-th anchor frame, the corresponding output frame is $m_{t'} = \lfloor \frac{l}{2} \rfloor + p + 1 + (t' - 1) \times s_t$. In particular, for the last (L'-th) anchor frame, the corresponding output frame is $m_{L'} = \lfloor \frac{l}{2} \rfloor + p + 1 + (L' - 1) \times s_t$. Moreover, $m_{L'} + \lfloor \frac{l}{2} \rfloor + p$ should be equal to L. Therefore, we have $L = \lfloor \frac{l}{2} \rfloor + p + 1 + (L' - 1) \times s_t + \lfloor \frac{l}{2} \rfloor + p$. Recall that l is odd, we then have $L = l + 2p + (L' - 1) \times s_t$.

Once we find the corresponding output positions for all anchor frames, we can obtain their ranges of influence in the output sequence. For the t'-th anchor frame, its range of influence $\Delta_{t'}$ in the output sequence is $\left[m_{t'} - \lfloor \frac{l}{2} \rfloor, m_{t'} + \lfloor \frac{l}{2} \rfloor\right] = \left[p + 1 + (t' - 1) \times s_t, 2 \times \lfloor \frac{l}{2} \rfloor + p + 1 + (t' - 1) \times s_t\right]$. Because l is odd, we then have $\Delta_{t'} = [p + 1 + (t' - 1) \times s_t, l + p + (t' - 1) \times s_t]$.

Last, for the t-th frame in the original sequence, its anchor points are collected from the $(t - m_{t'} + \lfloor \frac{l}{2} \rfloor + 1)$-th frame in the t'-th anchor's generated frames, where $t \in \Delta_{t'}$. In this way, the decoded point features by temporal transposed convolution are assigned to the original frames and the temporal cross-correlation is reconstructed. Note that, for an interpolated frame, its anchor points does not need to be a subset of its original point points.

In summary, we have the following conclusions:

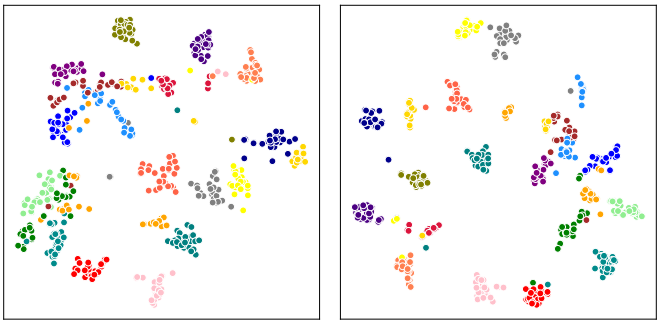
- l is odd.
- $\lfloor \frac{l}{2} \rfloor \geq -p$.
- For the t-th output frame, it anchor points are from the $\big(t+p+2+(t'-1)\times s_t\big)$-th generated frame of the t'-th anchor frame, where $t \in [p+1+(t'-1)\times s_t, l+p+(t'-1)\times s_t]$.
- The length of output sequence L is $l+2p+(L'-1)\times s_t$.

E INFLUENCE OF TEMPORAL KERNEL SIZE ON DIFFERENT ACTIONS

We show the influence of temporal kernel size on different actions of MSR-Action3D in Fig. 11. Clip length is 24.

When temporal kernel size $l = 1$, temporal correlation is not exploited. However, PSTNet can leverage pose information for recognition, expecfially for actions whose poses are unique in the dataset. For example, PSTNet with $l = 1$ can correctly recognize the "jogging" action because the action's pose is discriminative in the MSR-Action3D dataset. However, PSTNet with motion modeling ($l \geq 3$) can distinguish similar actions such as "draw x", "draw tick" and "draw circle". The results support our intuition that PSTNet effectively captures dynamics in point cloud sequences.

F FEATURE VISUALIZATION



(a) MeteorNet (b) PSTNet

Figure 12: Feature visualizations on MSR-Action3D using t-SNE. Each sequence is visualized as a point and sequences belonging to the same action have the same color. PSTNet features are semantically separable compared to MeteorNet suggesting that it is better for point cloud sequences.

We qualitatively evaluate PSTNet's ability to encode point cloud sequence by visualizing the learned features. We compare PSTNet with the sate-of-the-art MeteorNet (Liu et al., 2019d). Features are extracted from the layer before classifier (FC). These features are then projected to 2-dimensional space using t-SNE. As shown in Fig. 12, PSTNet feature is more compact and discriminative.

G SYNTHIA 4D SEMANTIC SEGMENTATION RESULT DETAILS

We list the segmentation result for each class in Table 4. The Synthia 4D dataset contains 12 categories. Our PSTNet achieves five best accuracies among them, demonstrating the effectiveness of PSTNet.

H PSTNET WITH TRACKING TUBES

Our PSTNet follows the idea of 3D convolution to capture local changes within a point tube rather than encoding the motion of a specific point. In this section, we investigate the influence of tracking on PSTNet. Because tracking requests point colors to estimate scene flow (Liu et al., 2019c), which

Table 4: Semantic segmentation result (mIoU %) details on the Synthia 4D dataset (Choy et al., 2019).

Method	# Frames	Bldn	Road	Sdwlk	Fence	Vegittn	Pole	Car	T. Sign	Pedstrn	Bicycl	Lane	T. Light	Average
3D MinkNet14 (Choy et al., 2019)	1	89.39	97.68	69.43	86.52	98.11	97.26	93.50	79.45	92.27	0.00	44.61	66.69	76.24
4D MinkNet14 (Choy et al., 2019)	3	90.13	98.26	73.47	87.19	**99.10**	97.50	94.01	79.04	**92.62**	0.00	50.01	68.14	77.46
PointNet++ (Qi et al., 2017b)	1	96.88	97.72	86.20	92.75	97.12	97.09	90.85	66.87	78.64	0.00	72.93	75.17	79.35
MeteorNet (Liu et al., 2019d)	3	**98.10**	97.72	88.65	94.00	97.98	**97.65**	93.83	**84.07**	80.90	0.00	71.14	**77.60**	81.80
PSTNet ($l = 1$)	3	96.32	98.07	85.40	94.66	97.16	97.51	94.83	76.65	76.99	0.00	75.39	76.45	80.79
PSTNet ($l = 3$)	3	96.91	**98.33**	**90.83**	**95.00**	96.96	97.61	**95.15**	77.45	85.68	0.00	**75.71**	77.28	**82.24**

are not available in MSR-Action3D or NTU RGB+D 60/120, we apply point tracking to Synthia 4D for semantic segmentation, following (Liu et al., 2019d). As shown in Tab. 5, tracking does not improve performance. We conjecture that this phenomenon might be caused by the following

Table 5: Semantic Segmentation on Synthia 4D with tracking.

Tracking	MeteorNet-m (Liu et al., 2019d)	PSTNet (ours)
No	**81.72**	**82.24**
Yes	81.47	81.61

reasons:

- In point cloud sequences, points are not consistent across different frames, and even flow in and out over time, making tracking significantly difficult.
- Inaccurate tracking results will hinder motion understanding. In other words, erroneous motion information can be propagated and enlarged in the following frames.

In contrast, without tracking, a model can sense when an object comes into or goes out of a tube. After merging the information from these tubes by hierarchical architecture, the model can summary the movement or trajectory of the object.

I Visualization of the Output of each Layer in PSTNet

We visualize the output of each layer in the PSTNet architecture trained on MSR-Action3D in Fig. 17. Because we use ReLU as the activation function, all outputs are greater than zero and large output indicate high activation. To visualize outputs, we squeeze output vectors to scalars via l_1 norm. We observe that:

- For PSTConv1, because temporal kernel size $l = 1$, it dose not capture the temporal correlation. In this case, PSTConv1 focuses on the appearance and therefore outputs high activation to performer contours.
- PSTConv2a to PSTConv4. When modeling the spatio-temporally structure of point cloud sequence, PST convolutions mainly focus on moving areas so as to achieve informative representations of actions.

The visualization results support our intuition that the proposed PST convolution effectively captures dynamics of point cloud sequences.

J Visualization of 4D Semantic Segmentation

We visualize segmentation results from the Synthia 4D dataset in Fig. 18 and Fig. 19. Our PSTNet can accurately segment most objects.

K PST CONVOLUTION WITHOUT DISENTANGLING SPATIO-TEMPORAL MODELING

To confirm the effectiveness of our disentangling structure for spatio-temporal modeling, we conduct 3D action recognition on MSR-Action3D with non-decomposing convolution. To combine space and time, timestamps and 3D coordinates are encoded together to generate convolution kernel. We evaluate our PSTNet with different clip lengths. As shown in Fig. 13, the disentangling PST convolution achieves better accuracy than the non-disentangling structure, demonstrating the effectiveness of the disentangling methodology.

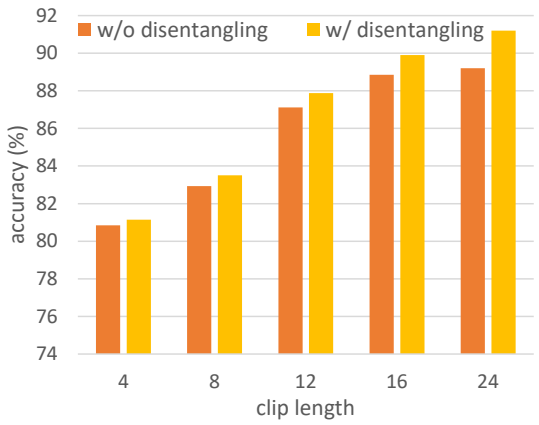


Figure 13: Comparison of PST convolution with (w/) and without (w/o) disentangling structure on MSR-Action3D.

L IMPACT OF DISPLACEMENT KERNEL AND SHARING KERNEL

In this paper, we design a lightweight implementation for the f function in Eq. (5), *i.e.*, $f\big((\delta_x, \delta_y, \delta_z); \boldsymbol{\theta}\big) = \boldsymbol{\theta}_d \cdot (\delta_x, \delta_y, \delta_z)^T \cdot \mathbf{1} \odot \boldsymbol{\theta}_s$, where $\boldsymbol{\theta}_d$ is a displacement transform kernel, $\boldsymbol{\theta}_s$ is a sharing kernel. In this section, we investigate the influence of these two kernels. We conduct 3D action recognition on MSR-Action3D. The experimental results are shown in Table 6.

Table 6: Impact of displacement kernel and sharing kernel on 3D action recognition accuracy (%). The MSR-Action3D dataset is used.

$\boldsymbol{\theta}_d$	$\boldsymbol{\theta}_s$	16-frame	24-frame
✓		58.25	61.67
	✓	86.35	87.46
✓	✓	**89.90**	**91.20**

Without $\boldsymbol{\theta}_s$, the spatial convolution degenerates to $\boldsymbol{M}_t^{(x,y,z)} = \sum_{||(\delta_x,\delta_y,\delta_z)||\leq r} \boldsymbol{\theta}_d \cdot (\delta_x, \delta_y, \delta_z)^T$. In this way, only the points in the final layer of our PSTNet are used. Therefore, the accuracy decreases dramatically.

Without $\boldsymbol{\theta}_d$, the spatial convolution degenerates to $\boldsymbol{M}_t^{(x,y,z)} = \sum_{||(\delta_x,\delta_y,\delta_z)||\leq r} \boldsymbol{\theta}_s \cdot \boldsymbol{F}_t^{(x+\delta_x, y+\delta_y, z+\delta_z)}$. In this fashion, the points in a neighborhood are treated equally. Point features are leveraged but their positions are ignored. Because spatial structure is not well captured, the accuracy decreases compared to the case where both $\boldsymbol{\theta}_d$ and $\boldsymbol{\theta}_s$ are used.

M COMPUTATION AND MEMORY EFFICIENCY STUDY

In this section, we evaluate the computation and memory efficiency, *i.e.*, the running time and number of parameters, of our method. We conduct 3D action recognition with a clip length of 16 on MSR-Action3D using 1 Nvidia Quadro RTX 6000 GPU.

As shown in Table 7, the proposed PSTNet is 41.5% faster than MeteorNet (Liu et al., 2019d). This is because, on one hand, MeteorNet is not temporally hierarchical. All frames are processed in each layer, which needs much computation. One the other hand, MeteorNet is based on MLPs, which are both computation and memory consuming. In contrast, our PSTNet is a spatio-temporally hierarchical architecture, in which points are subsampled along both spatial and temporal dimensions, and therefore saves running time. Moreover, our PST convolutions do not contain MLPs, and thus further saves running time and uses less parameters than MeteorNet.

Table 7: Comparison of parameter and running time on 3D action recognition accuracy (%). The MSR-Action3D dataset is used. Clip length is 16.

Method	# Parameters (M)	Running time per clip (ms)	Accuracy (%)
MeteorNet (Liu et al., 2019d)	17.60	54.56	88.21
PSTNet (original)	**8.26**	**31.92**	89.90
PSTNet (with PointConv)	20.46	102.72	**90.24**

Second, we study the impact of the number of PSTNet layers on running time and parameter. As shown in Fig. 14, when PSTNet becomes deep, the number of parameters significantly grows. This is because, like most deep neural networks, the point feature dimension exponentially increases in PSTNet, which needs larger kernels and more parameters to process. However, the running time does not increase significantly. This is because our PSTNet is spatio-temporally hierarchical, points are exponentially reduced along both spatial and temporal dimensions, thus saving running time.

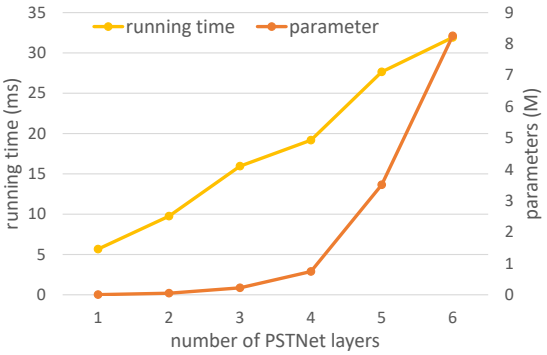


Figure 14: Impact of the number of PSTNet layers on running time and parameter.

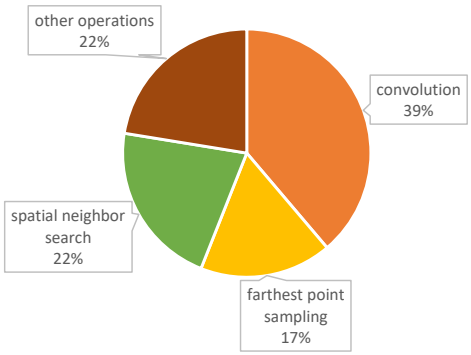


Figure 15: Running time proportion of main operations in PSTNet.

Finally, we investigate the running time of main operations in PSTNet, including convolution, farthest point sampling and spatial neighbor search. We show the running time proportion of these operations in Fig. 15. The running time is mainly spent on convolutions.

N PST CONVOLUTION WITH OTHER SPATIAL CONVOLUTION

The f function in Eq. (5) can be implemented in many ways. In this section, we replace our lightweight spatial convolution with PointConv (Wu et al., 2019). Experiments are conducted using 1 Nvidia Quadro RTX 6000 GPU.

As shown in Table 7, using PointConv as spatial operations only improves performance slightly compared to our lightweight spatial convolutions, but significantly increases parameters and running time. This is because PointConv adopts MLPs to process points and their features, which is memory

and computation consuming. Moreover, PointConv[1] uses kNN to search neighbors, in which sort is needed, and thus takes much more time than our lightweight spatial convolution.

O Irregular Frame Sampling

Our PSTNet can also process irregularly sampled point cloud sequences by frame interpolation. In this section, we randomly remove some frames in sequences. Specifically, we conduct 3D action recognition on MSR-Action with 24 frames. We randomly remove 8 frames from each sequence. Then, we use replication and Iterative Closest Point (ICP) (Besl & McKay, 1992) to interpolate missing frames, respectively. We compare our PSTNet with MeteorNet (Liu et al., 2019d), which explicitly encodes timestamps and does not need interpolation.

Table 8: 3D action recognition on MSR-Action3D with irregularly sampled frames. Clip length is originally 24 and then 8 frames are randomly removed from each clip.

Method	Accuracy (%)
MeteorNet (Liu et al., 2019d)	88.40
PSTNet (replication interpolation)	90.24
PSTNet (ICP interpolation)	**90.94**

As shown in Table 8, our PSTNet still achieves the best accuracy. This indicates that PSTNet is able to model irregularly sampled dynamic point clouds.

P Scene Flow Estimation

To evaluate our method on LiDAR data where points in a frame can be highly irregular, we apply our method to scene flow estimation. We follow the setting of MeteorNet (Liu et al., 2019d), *i.e.*, given a point cloud sequence, estimating a flow vector for every point in the last frame. However, because our point tube is constructed according to the middle frame, which is not applicable to the last-frame scene flow estimation, we adapt temporal anchor frame selection and spatial anchor point transferring. Specifically, we select the last frame in each tube as the anchor frame. Then, after spatial sampling, each anchor point is transferred to its previous nearest l frames. Following MeteorNet, we first train our model on the FlyingThings3D dataset (Mayer et al., 2016), and then fine-tune the model on the KITTI scene flow dataset (Menze & Geiger, 2015). Point tracking is not used.

Table 9: Scene flow estimation accuracy on the KITTI scene flow dataset.

Method	Input	# Frames	End-Point-Error
FlowNet3D (Liu et al., 2019c)	points	2	0.287
MeteorNet (Liu et al., 2019d)	points	3	0.282
PSTNet (ours)	points	3	**0.278**

As shown in Table 9, our PSTNet achieves promising accuracy on scene flow estimation.

Q Motion Recognition on the Synthetic Moving MNIST Point Cloud Dataset

In the real world, it is impossible to obtain point IDs. To evaluate our PSTNet in the scenario where points are correlated and can be accurately tracked across frames, we conduct motion recognition on a synthetic Moving MNIST Point Cloud dataset. To synthesize moving MNIST digits sequences, we use a generation process similar to that described in (Fan & Yang, 2019). Each synthetic sequence consists of 16 consecutive point cloud frames. Each frame contains one handwritten digit moving

[1] https://github.com/DylanWusee/pointconv_pytorch

and bouncing inside a 64×64 area. The digits are chosen randomly from the MNIST dataset. We sample 128 points for each digit. Point order is maintained across frames so that tracking can be employed. We design 144 motions, including

- Nine initial locations: $(1, 1), (1, 3), (1, 5), (3, 1), (3, 3), (3, 5), (5, 1), (5, 3), (5, 5)$.
- Eight velocities: $(1, 1), (-1, 1), (-1, -1), (1, -1), (2, 2), (-2, 2), (-2, -2), (2, -2)$.
- Two kinds of digit distortion: scaling digits horizontally or vertically, *i.e.*, $(|0.4 - 0.05t| + 0.6) \times (x - c_x) + c_x$ or $(|0.4 - 0.05t| + 0.6) \times (y - c_y) + c_y$, where (c_x, c_y) is the digit center and $t \in [1, 16]$.

A few motion examples are shown in Fig. 16. Because motion and appearance are independent of each other, it is challenging to recognize motion while avoiding interference from digit appearance.

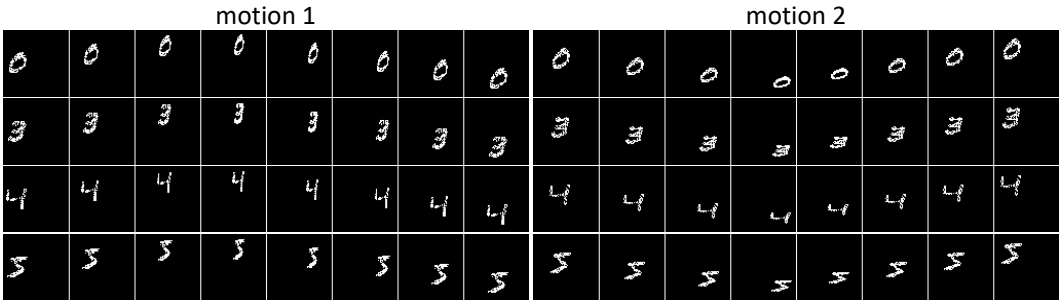


Figure 16: Motion examples in the the synthetic Moving MNIST Point Cloud dataset.

To exploit point correlation, anchor points and their neighbors are selected in the first frame, which are then propagated to other frames. In this fashion, anchors and their neighbors are tracked across frames.

Table 10: Motion recognition accuracy on the Moving MNIST Point Cloud dataset.

Method	Accuracy (%)
PSTNet (original)	97.15
PSTNet (with tracking)	**98.74**

As shown in Table 10, both original and tracking based PSTNet achieve promising accuracy on the Moving MNIST Point Cloud dataset. Our original PSTNet achieves similar accuracy with tracking based PSTNet in this simulated case, demonstrating that our PSTNet does not heavily rely on point IDs.

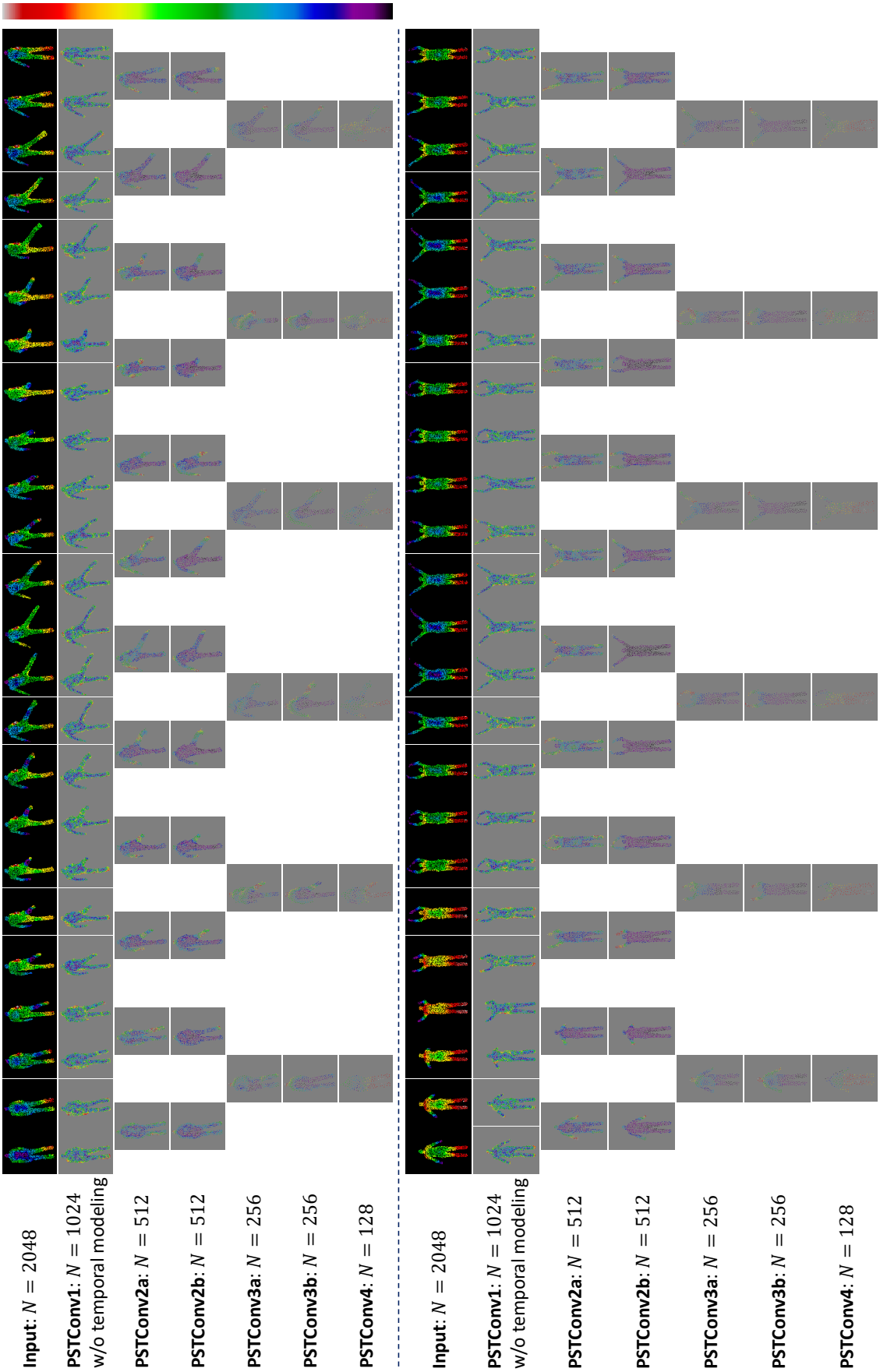


Figure 17: Visualization of the output of each layer in PSTNet. For the input point cloud sequence, color encodes depth. For the outputs, brighter color indicates higher activation. Input sequences consist of 24 frames. Due to the spatial subsampling s_s and the temporal stride s_t, points and frames progressively decrease along the network depth.

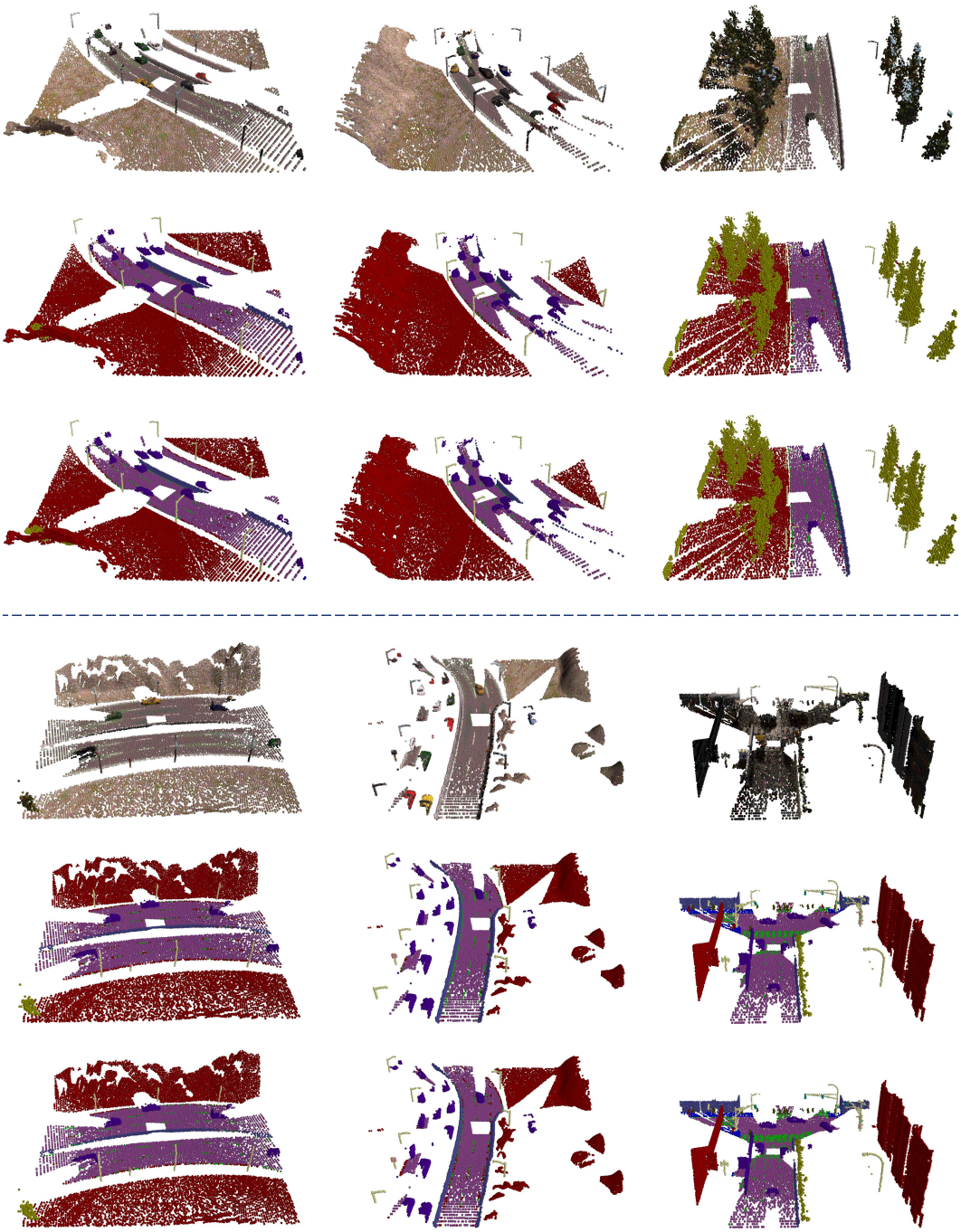


Figure 18: Visualization of semantic segmentation examples from Synthia 4D (I). Top: inputs. Middle: ground truth. Bottom: PSTNet predictions.

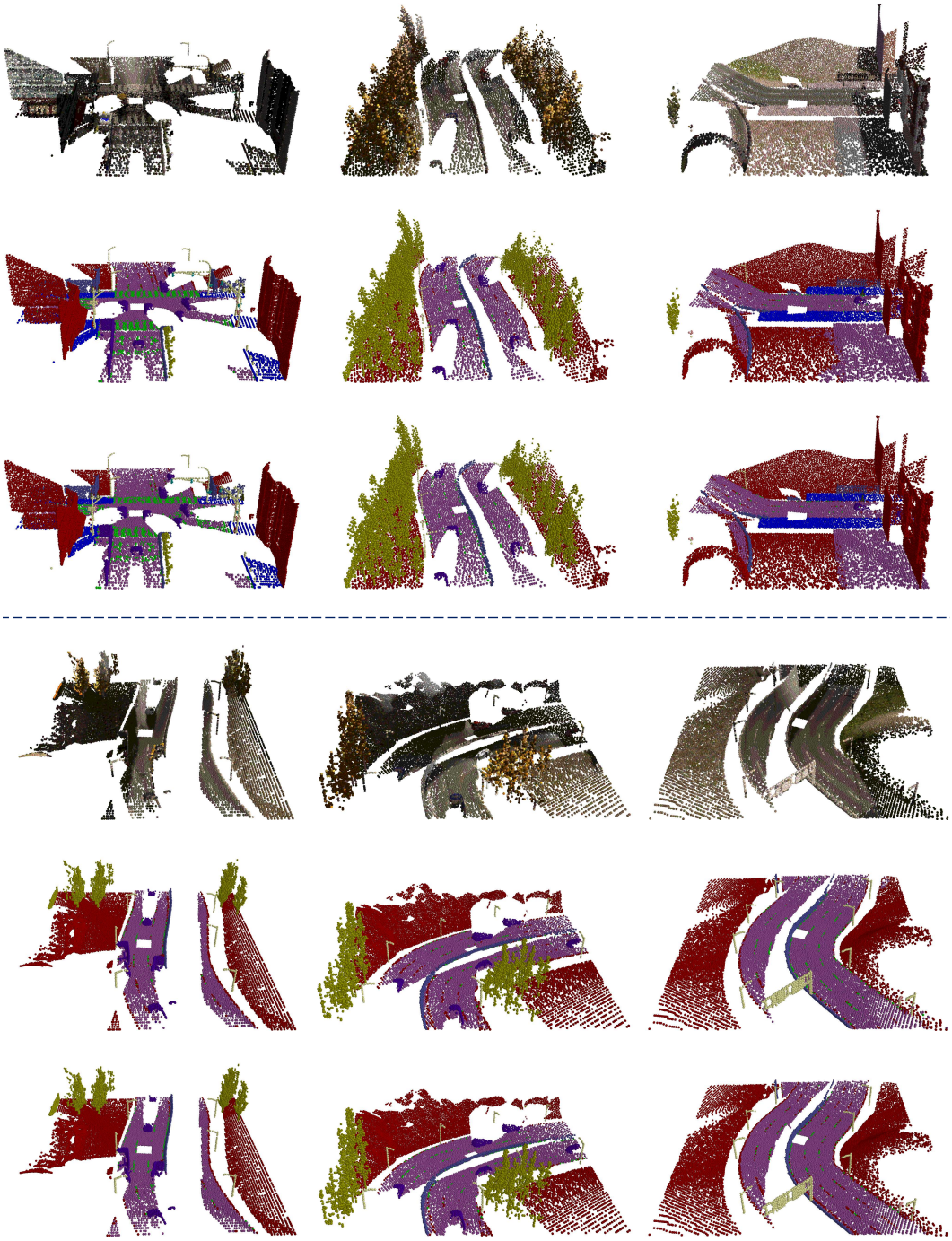


Figure 19: Visualization of semantic segmentation examples from Synthia 4D (II). Top: inputs. Middle: ground truth. Bottom: PSTNet predictions.