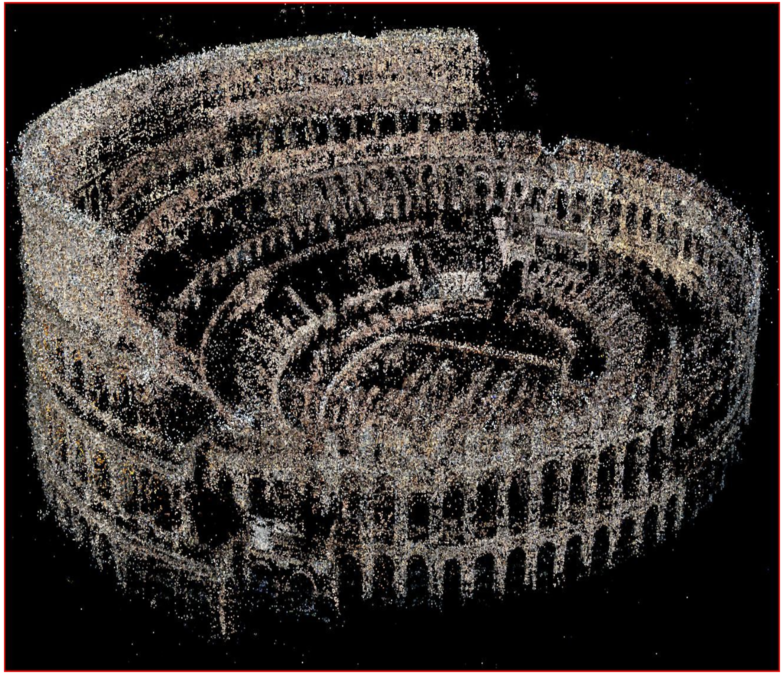
Survey of GNN on 3D Point Cloud Semantic Segmentation

Presenters: Jiayue Sun, Hengda Shi, Benlin Liu

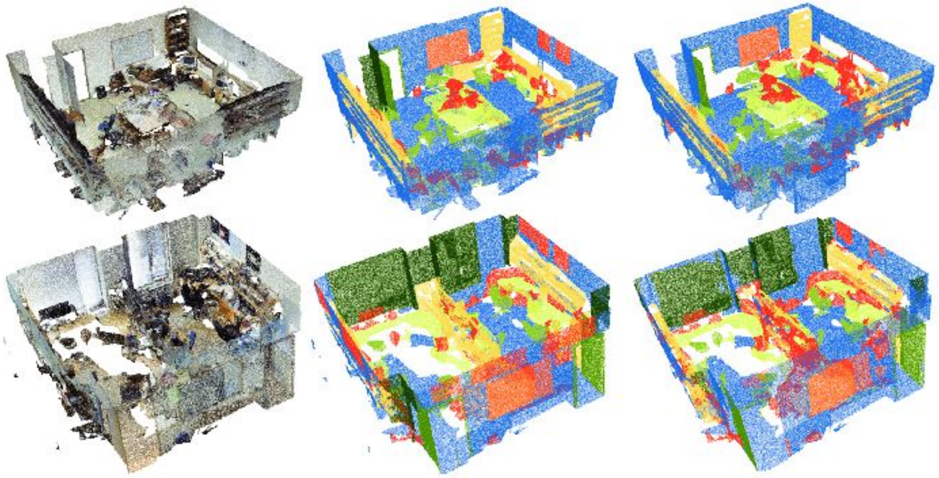
Why point Cloud? Why GNN?

- Point Cloud contains depth information!
- Captured using 3D scanner, like RGB-D camera
- Geometric information
  - Also color information
- Survey Scope: GNN methods to encode point cloud for semantic segmentation (part segmentation and scene segmentation)



Data & Evaluation Metrics

- Data:
  - ShapeNet, ModelNet40
  - S3DIS (area 5, 6-fold)
  - KITTI
- Metrics:
  - Class-wise mean of intersection over union (mIoU)
  - Class-wise mean of accuracy (mAcc)
  - Point-wise overall accuracy (OA)



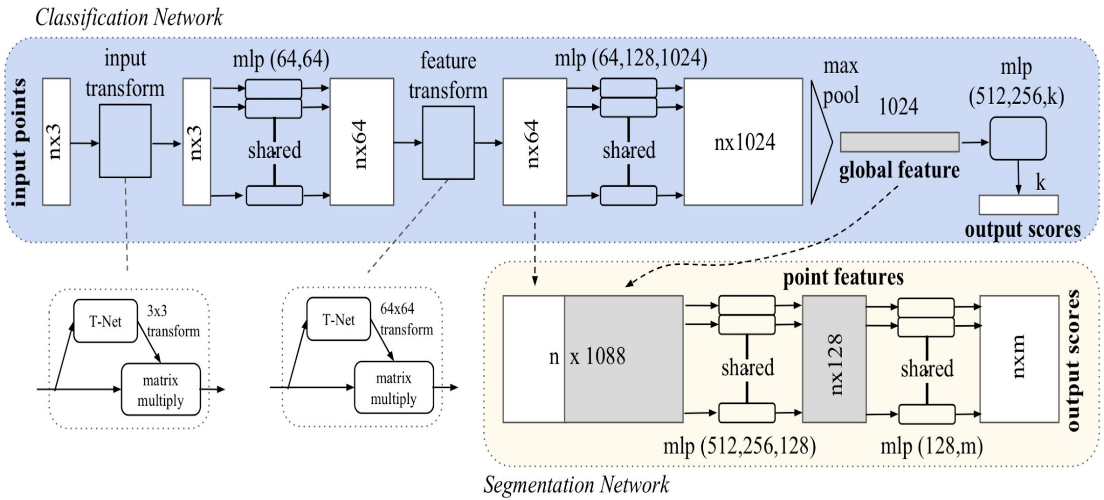
Learning on Point Cloud

- Multi View
  - Loss of structure info
- Voxel
  - Sparse
- Can we directly learn on the point cloud data?

Some Properties of Point Cloud

- Unordered
  - Permutation shouldn't change the result
- Interaction among points
  - Capture the local structure
- Invariant under certain transformation
  - Like rotation and translation

PointNet



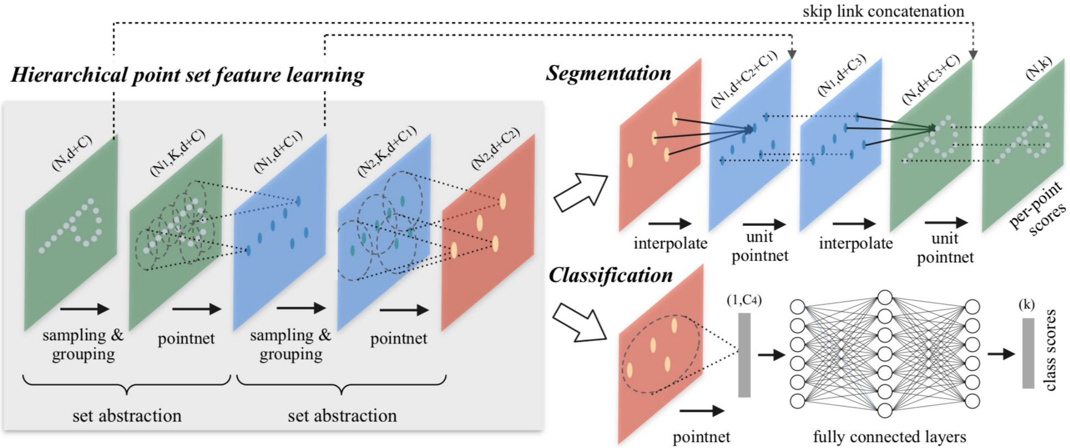
- Unordered
  - Using Maxpool as symmetry function
- Interaction among points
  - Concatenating the global feature and local feature
- Invariant under certain transformation
  - Align input to a canonical space

PointNet++

Drawbacks of PointNet:

- MLP cannot well capture the local structure!
  - Like convolution neural network
- Solution：
  - Hierarchical point feature learning based on PointNet

PointNet++



Drawbacks of PointNet:

- Sampling
  - Farthest point sampling
- Grouping
  - Ball query
- Aggregation
  - PointNet

EdgeConv (Dynamic GCNN)

- Dynamic graph setting
  - Reconstruct the graph by connecting central vertex with k-nearest neighbors.
- A generalization of the PointNet architecture, the output of EdgeConv at the i-th vertex is given by:

$$\mathbf{x}'_i = \square_{j:(i,j)\in\mathcal{E}} h_{\Theta}(\mathbf{x}_i, \mathbf{x}_j).$$

$\mathbf{x}'_i$: output for each vertex $\mathbf{x}_i$

$\mathbf{x}_j$: neighboring vertices

$h_{\Theta}(\mathbf{x}_i, \mathbf{x}_j)$: mapping function between central vertex and neighboring vertex

$\square_{j:(i,j)\in\mathcal{E}}$: aggregation function (e.g. max, avg)

EdgeConv (Dynamic GCNN)

- PointNet can be seen as a special case (ignores neighboring points).

$$h_{\mathbf{\Theta}}(\mathbf{x}_i, \mathbf{x}_j) = h_{\mathbf{\Theta}}(\mathbf{x}_i)$$

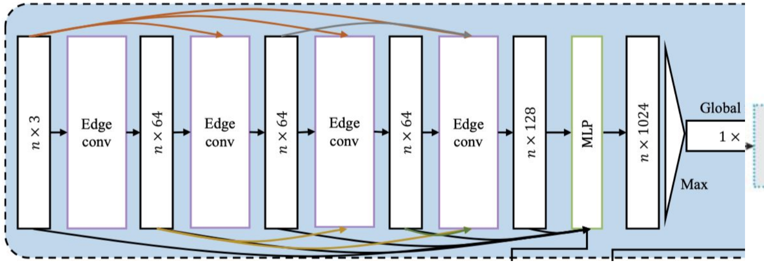
- DGCNN uses the following aggregation function and mapping function:

$$h_{\mathbf{\Theta}}(\mathbf{x}_i, \mathbf{x}_j) = \bar{h}_{\mathbf{\Theta}}(\mathbf{x}_i, \mathbf{x}_j - \mathbf{x}_i)$$

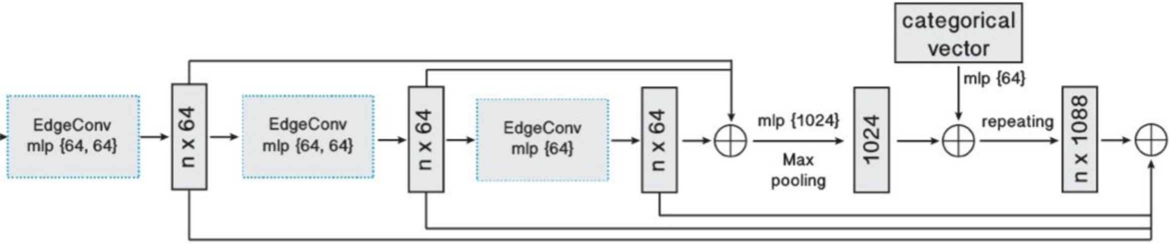
- Takes both central vertex information and relative information from neighboring points.
- Other improvements:
  - Skip connection
  - Transformation matrix (same as PointNet) to ensure transformation invariance.

Linked Dynamic GCNN

- It has the following difference from DGCNN.
    a. Different skip connection (see below figures).
    b. The pose normalization step is removed.
    c. At a certain point, the EdgeConv layers will be freezed and later MLPs will be retrained.



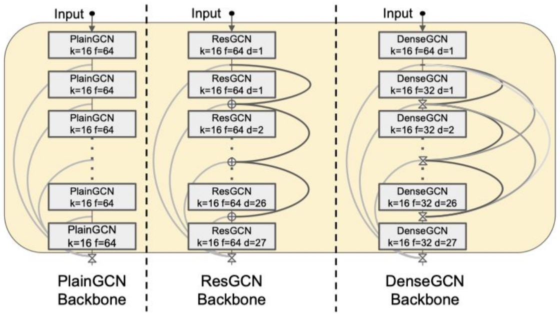
LDGCNN

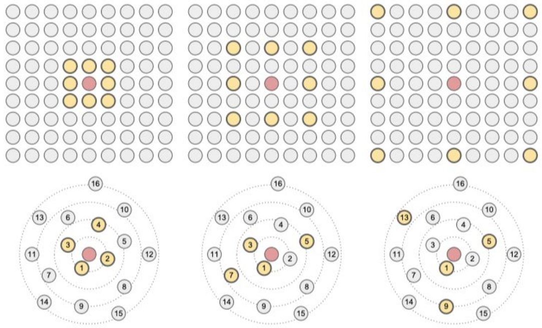


DGCNN

DeepGCNs

- extensively studies the different skip connection setups
- The PlainGCN is the setup for DGCNN, and DenseGCN is the setup for LDGCNN.
- ResGCN closely follows the skip connection setup for ResNet.
- They constructs the graph by **dilated k-nearest neighbor** to enlarge the receptive field.
- i.e. If the true 4-nearest neighbor is [1, 2, 3, 4], then dilated 4-nearest neighbor is [2, 4, 6, 8].
- Based on the experiment, ResGCN-28 which stacks 28 GCNs with ResNet-like skip connection seems to balance well between the model size and performance.





Regularized GCNN

- RGCNN also adopts the dynamic graph update at each layer.
- However, they connect **every vertex with all other vertices** on the graph, and **assign weight based on the distance metric**:

$$a_{i,j} = \exp(-\beta \|\mathbf{p}_i - \mathbf{p}_j\|_2^2)$$

- RGCNN further generalizes the DGCNN EdgeConv to aggregate all points up to k-th hop from the central point (same setup as ChebNet).

$$\mathbf{y} = g_\theta(\mathcal{L})\mathbf{x} = \sum_{k=0}^{K-1} \theta_k T_k(\mathcal{L})\mathbf{x}$$

- When **K=1**, it's the setup of **PointNet** where no neighbor information is aggregated.
- When **K=2**, it's the setup of **classic GCN**, **DGCNN** and **LDGCNN** where only the first hop neighbors are aggregated.
- What's more, they proposed to add a smoothness prior to the adjacency matrix to enforces the features of adjacent vertices to be more similar:

$$\sum_{i \sim j} a_{i,j}(y_i - y_j)^2 < \epsilon, \ \forall i, j,$$

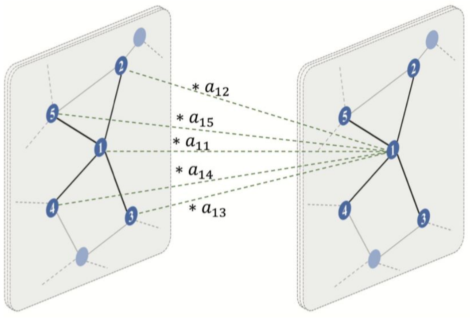
GAC-Net

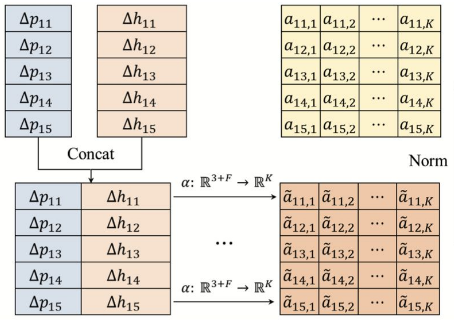
- GAC-Net instead incorporates the attention mechanism to assign different weights to neighbor nodes.
- The attention is formulated as $\alpha(\Delta p_{ij}, \Delta h_{ij}) = M_\alpha([\Delta p_{ij} || \Delta h_{ij}])$.

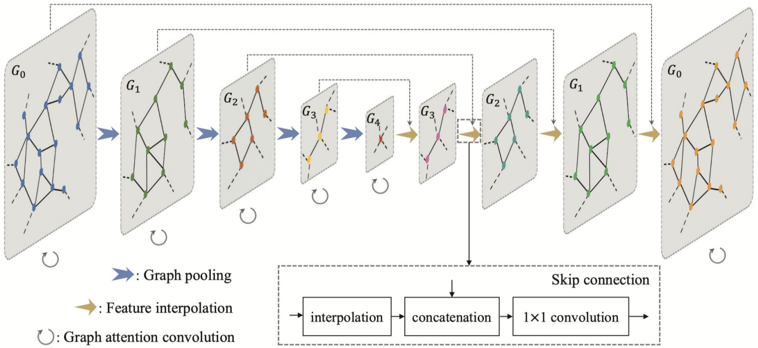
  $\Delta p_{ij} = p_i - p_j$: distance between two points (p_i : point coordinate).

  $\Delta h_{ij} = M_g(h_i) - M_g(h_j)$: difference between two transformed feature vectors. (h_i: feature vector).

  M_α, M_g are MLP layers, and $||$ is concatenation.
- In addition to the attention mechanism, GAC-Net models the network structure similar to the PointNet++ (encoder-decoder architecture), and use k-NN and FPS sampling to alternatively reconstruct or subsample at each layer.



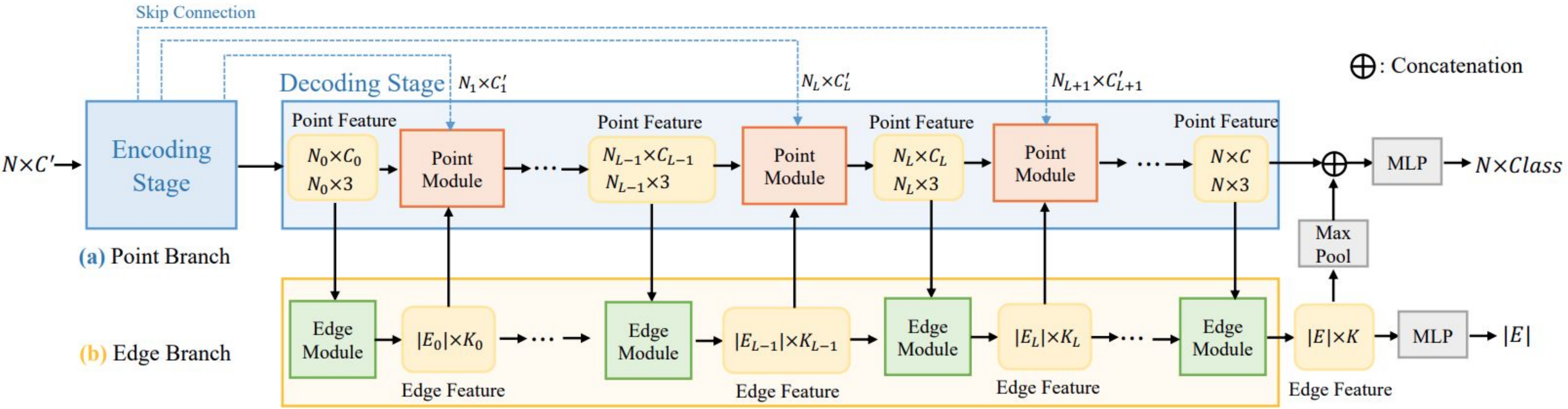




Pros and Cons

- DGCNN, LDGCNN, and ResGCN:
  - Pros: Dynamic graph update helps the model to learn a larger receptive field.
  - Cons: Each layer’s kNN recomputation is always computationally expensive.
- RGCNN:
  - Pros: The graph construction computation at each layer takes less time.
  - Cons: The adjacency matrix is always dense and hence is memory inefficient.
- GAC-Net:
  - Pros: Leverages the attention mechanism to assign different weights on neighbor nodes.
  - Cons: No dynamic graph update at each layer, and hence the receptive field is limited.

HPEIN: Hierarchical Point-Edge Interaction Network



- **Edge branch**: progressively integrate point features in different layers.
- **Loss function:** final edge features are supervised by semantic-consistency prediction of each edge. (whether the two end-points are in the same category or not)

$$L = \lambda_1 L_{point} + \lambda_2 L_{edge} \qquad l_{i,j}^e = \begin{cases} 1, & \text{if } l_i^p = l_j^p \\ 0, & \text{if } l_i^p \neq l_j^p \end{cases}$$

Edge Module

- Point features F and edge features H

$$\mathbb{H}_{E_L} = M_{encoder}\left(\mathbb{F}_{V_L}, M_{upsample}\left(\mathbb{H}_{\mathbb{E}_{L-1}}\right)\right)$$

- Encode edge with **upsampled edge feature** from prev layer and **point features** as input.

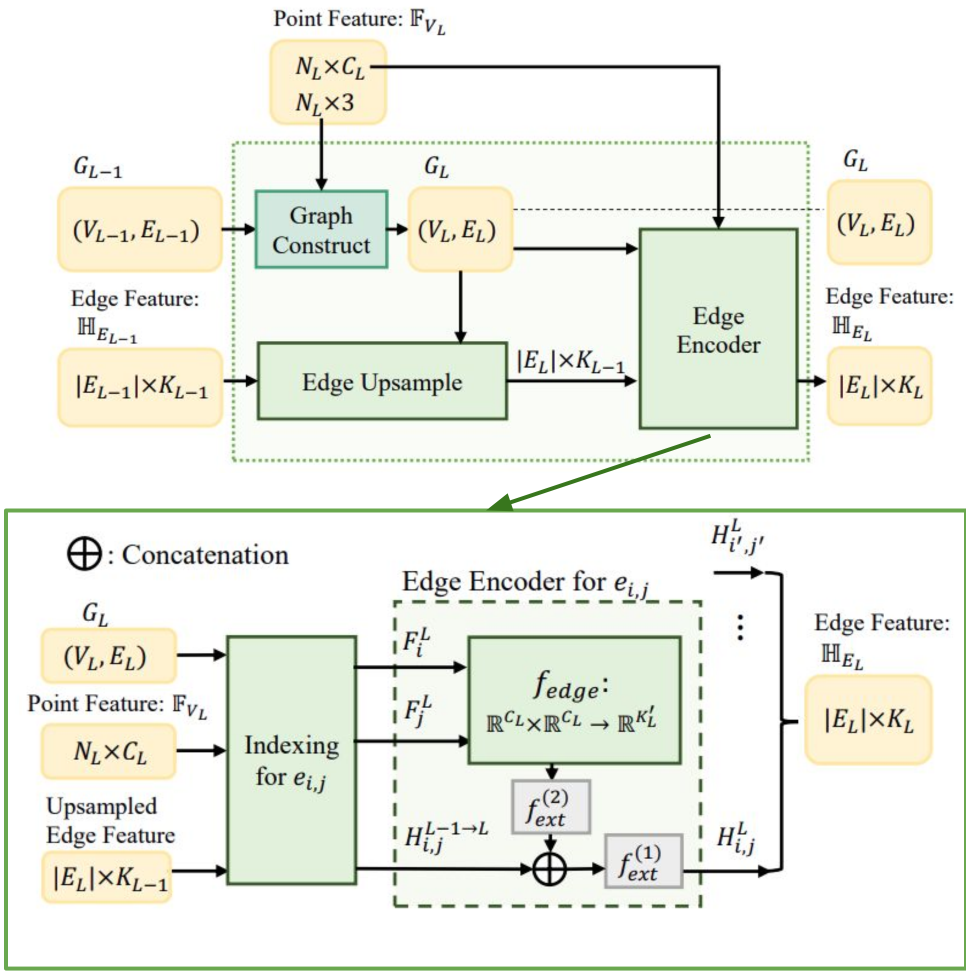
$$H^L_{i,j} = f^{(1)}_{ext}\left(\left[f^{(2)}_{ext}\left(f_{edge}\left(F^L_i, F^L_j\right)\right), H^{L-1\to L}_{i,j}\right]\right)$$

$$f_{edge}\left(F^L_i, F^L_j\right) = \left[(p_i - p_j), F^L_i, F^L_j\right]$$

- The edge features are aggregated by **max-pooli** and are used to **update point feature**

$$\mathbb{H}_{E_L(p_i)} = \left\{H^L_{i,j} \mid (p_i\ ,\ p_j) \in E_L(p_i)\right\}$$

$$\left(F^L_i\right)_{new} = \left[F^L_i, \mathrm{MaxPool}\left(\mathbb{H}_{E_L(p_i)}\right)\right]$$



Hierarchical Graph Construction

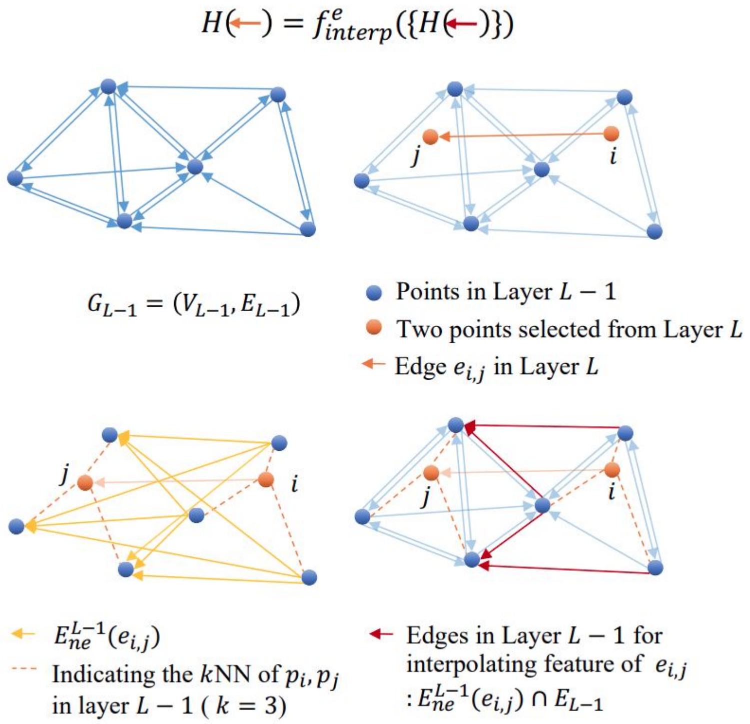
- Graph G at layer L:
  - finding the KNN points for each point in input vertices.
  - Add edges between two KNN neighbor points in the coarser(prev) layer if they are connected.

- **Edge Upsampling:**
  Propagate edge features from previous layer to current layer by interpolating **KNN edge features** from prev layer:

$$H_{i,j}^{L-1\to L} = f_{interp}^{e}\left(\left\{H_{i',j'}^{L-1\to L} \mid (p_{i'}, p_{j'}) \in E_{ne}^{L-1}(e_{i,j}) \cap E_{L-1}\right\}\right)$$

- The **interpolation weights** are based on the normalized **inverse distance** of the two pairs of end points.

$$w_{i',j'} = \frac{1}{(\|p_i - p_{i'}\|^t + \epsilon) \cdot (\|p_j - p_{j'}\|^t + \epsilon)}$$



HDGCN: Hierarchical Depthwise Graph CNN

Find KNN neighbor points: $\left\{p_{j_{i_1}}, \cdots, p_{j_{i_k}}\right\}$

Spatial Graph Convolution:

$$x_i' = \sum_{m=1}^{k} MLP(L(i, j_{i_m}); \theta) x_{j_{im}} \quad L(i, j_{i_m}) = p_{j_{im}} - p_i$$

Mem: $n \times k \times O \times I$

Depthwise Graph Convolution:

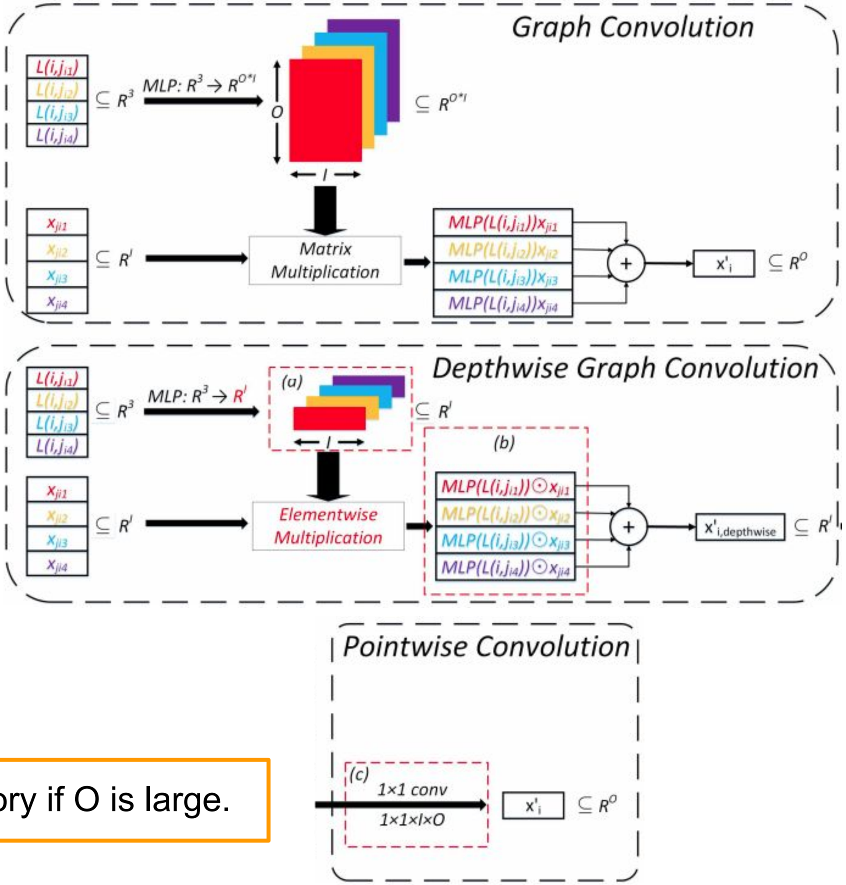
- Use graph convolution to aggregate local features channel-wisely (depthwise graph convolution)

$$x'_{i,depthwise} = \sum_{m=1}^{k} MLP(L(i, j_{i_m}); \theta) \odot x_{j_{im}}$$

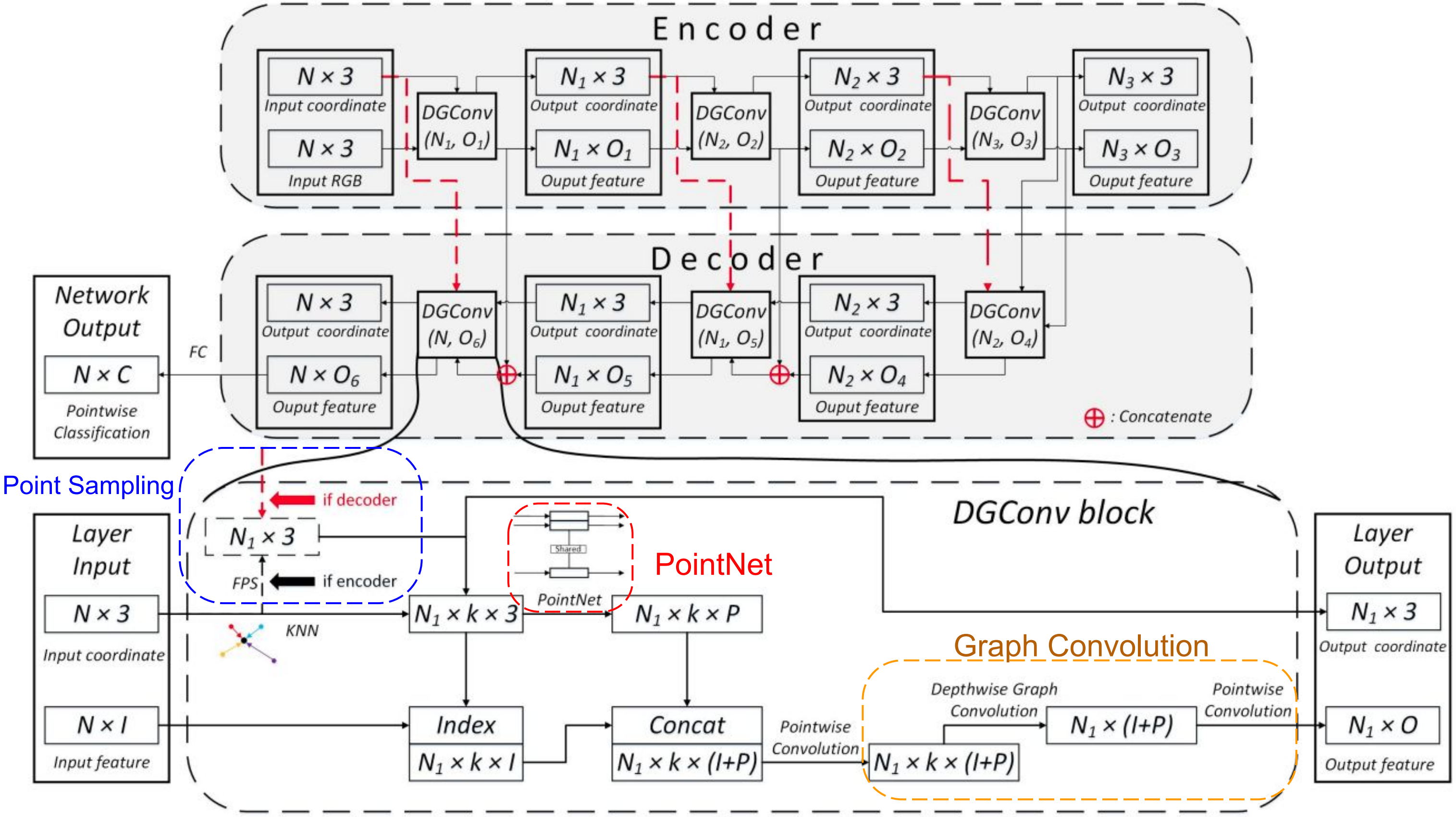
- Pointwise convolution (MLP) to project local depthwise feature to output channel size R^O

Mem: $n \times k \times I \times 2 + I \times O$

Use much less memory if O is large.



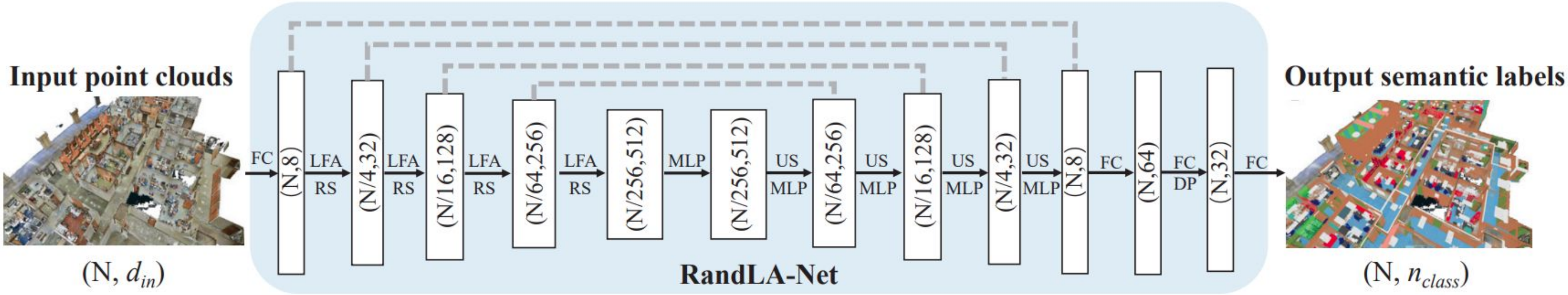
HDGCN: Hierarchical Depthwise Graph CNN



RandLA-Net

➔ Scalability issue of previous methods:

- **Point-sampling** are computationally expensive or memory inefficient (**Farthest Point Sampling**)
- Expensive **local feature learner**: kernelization or graph construction
- Existing local feature learners are incapable of capturing complex structures due to their **limited size of receptive fields**.



➔ Random Sampling (RS): downsampling in each neural layer

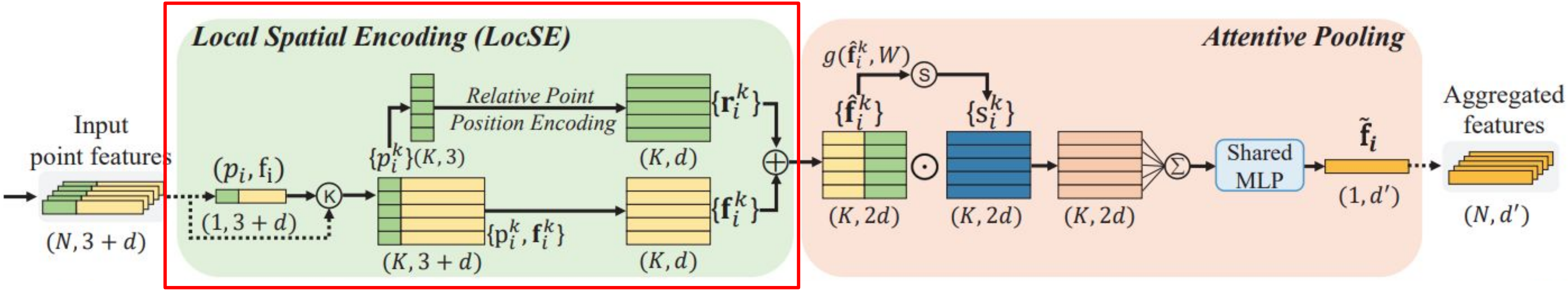
- **Computationally efficient**: agnostic to the number of input points. ✔️
- Computation does not require extra memory. ✔️
- Can **discard important information**, especially for objects with sparse points ❌

RandLA-Net

Local Spatial Encoding (LocSE)

- Find **KNN points** with point-wise euclidean distances
- Encode **relative point position** for each neighbor $\mathbf{r}_i^k = MLP\big(p_i \oplus p_i^k \oplus (p_i - p_i^k) \oplus \|p_i - p_i^k\|\big)$
- Point feature augmentation: $\hat{\mathbf{f}}_i^k = \mathbf{f}_i^k \oplus \mathbf{r}_i^k$
  concatenate **relative** point position $\mathbf{r}_i^k$ and point **feature** $\mathbf{f}_i^k$

$$\hat{\mathbf{F}}_i = \left\{\hat{\mathbf{f}}_i^1 \cdots \hat{\mathbf{f}}_i^k \cdots \hat{\mathbf{f}}_i^K\right\}$$

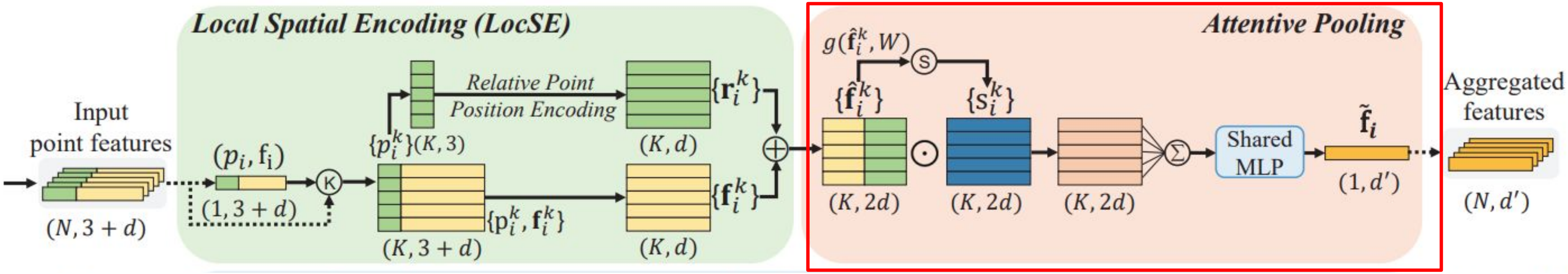


RandLA-Net

Attentive Pooling

- Aggregate the set of neighbor point features $\hat{\mathbf{F}}_i = \left\{\hat{\mathbf{f}}_i^1 \cdots \hat{\mathbf{f}}_i^k \cdots \hat{\mathbf{f}}_i^K\right\}$
- Compute attention scores: function **g** is a shared MLP followed by softmax
- Weighted summation features by their attention scores:

$$\tilde{\mathbf{f}}_i^k = \sum_{k=1}^{K} \hat{\mathbf{f}}_i^k \odot \mathbf{s}_i^k \qquad \mathbf{s}_i^k = g\left(\hat{\mathbf{f}}_i^k, \mathbf{W}\right)$$

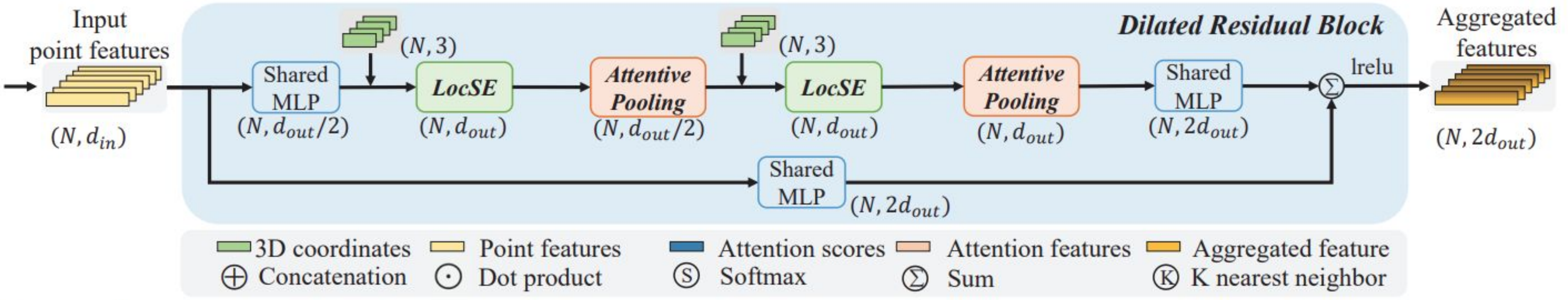


RandLA-Net

Dilated Residual Block

- **LocSE+Attentive Pooling operation**:
  Dilate the receptive field and expand the effective neighborhood through feature propagation.

- Stack **two** sets of LocSE and Attentive the Pooling in the block to achieve a balance between efficiency and effectiveness.



Contributions

- HPEIN (main changes in decoding stage)
  - New edge branch that hierarchically encodes edge features and interacts with the point branch
  - Hierarchical graph construction when upsampling and upsampled edge features
- HDGCN
  - Memory-efficient graph convolution
  - DGConv block that uses PointNet to extract point features
- RandLA-Net
  - Lightweight model that only consists of MLP to effectively process large-scale point clouds
  - Random sampling that is computationally and memory efficient.
  - Dilated residual block aggregate effectively increase

	Total time (seconds)	Parameters (millions)	Maximum inference points (millions)
PointNet (Vanilla)	192	0.8	0.49
PointNet++ (SSG)	9831	0.97	0.98
PointCNN	8142	11	0.05
SPG	43584	**0.25**	-
KPConv	717	14.9	0.54
RandLA-Net (Ours)	**185**	1.24	**1.03**

Results

	ShapeNet		ModelNet40		S3DIS (6-fold)			S3DIS (area-5)			ScanNet		
	OA	mIoU	OA	mAcc	OA	mAcc	mIoU	OA	mAcc	mIoU	OA	mAcc	mIoU
PointNet		83.7	89.2	86.0	78.5	66.2	47.6	-	48.98	41.09			
PointNet++		85.1	90.7		81.0	67.1	54.5	-	-	-			33.9
DGCNN	-	**85.2**	**93.5**	90.7	84.1	-	56.1	-	-	-	-	-	-
LDGCNN	-	85.1	92.9	90.3	-	-	-	-	-	-	-	-	-
ResGCN-28	-	-	-	-	85.9	-	60.0	-	-	52.49	-	-	-
RGCNN	-	84.3	90.5	87.3	-	-	-	-	-	-	-	-	-
GAC-Net	-	-	-	-	-	-	-	**87.79**	-	**62.85**	-	-	-
HPEIN	-	-	-	-	**88.2**	76.26	67.83	87.18	**68.3**	61.85	-	-	**61.8**
HDGCN					-	76.11	66.85	-	65.81	59.33			
RandLA-Net					88.0	**82.0**	**70.0**						

Summary

- Dynamic graph update does not provide significant improvement compared to the hierarchical encoder-decoder structure of PointNet++, when applying to large scene segmentation datasets.

- Hierarchical structure enables larger receptive field, which enables the model to perform better on more complex scene understanding task.

- Sampling cost in hierarchical encoder-decoder structure can be a huge bottleneck in terms of both time and memory.
    - Farthest point sampling (FPS) is very expensive