

Lecture 5 – Supplementary

主讲人 Jiaxin Li

Aptiv 自动驾驶
新加坡国立大学 博士
清华大学 本科

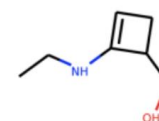
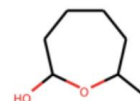
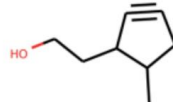
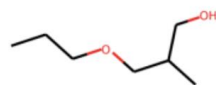
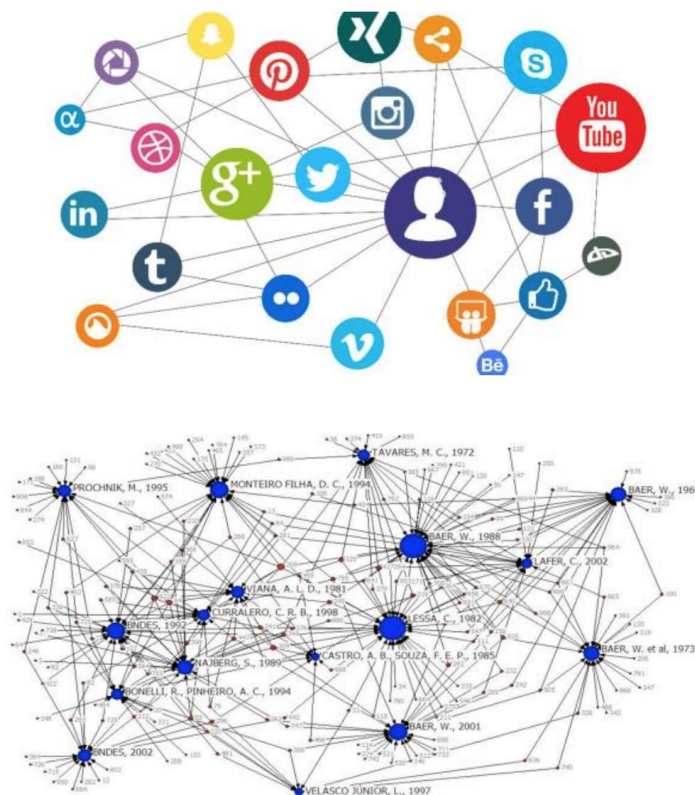




- An instance of GCN for Point Cloud
 - Dynamic Graph CNN for Learning on Point Clouds (DGCNN)
- General GCN
 - Some common GCNs
 - GCN vs. DGCNN
- Classification vs. Semantic Segmentation



Graph Data



- Social Network
- Citation Network
- Molecules
- **Point Cloud**
- 3D Mesh
-

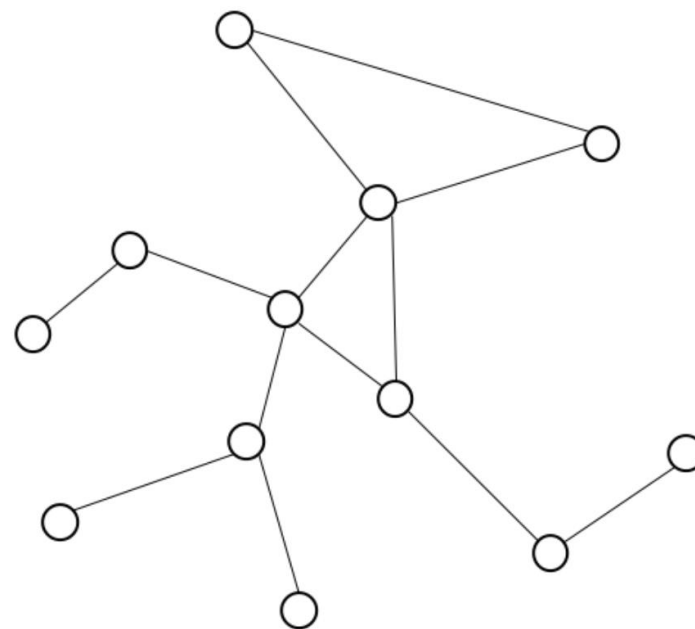


- Graph - $G(V, E)$
 - V : a set of vertices
 - E : a set of edges
- Represent a point cloud by graph?
 - One point – one vertex
 - Edge
 - Fix – by coordinate based kNN/RadiusNN
 - Dynamic – **DGCNN**

Vertex info:

Coordinate: \mathbb{R}^3

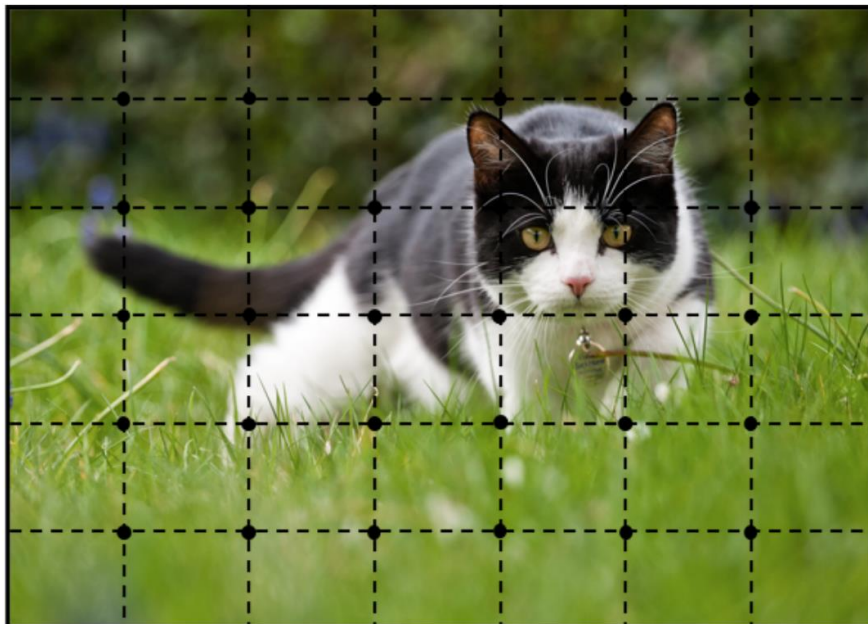
Feature: \mathbb{R}^m



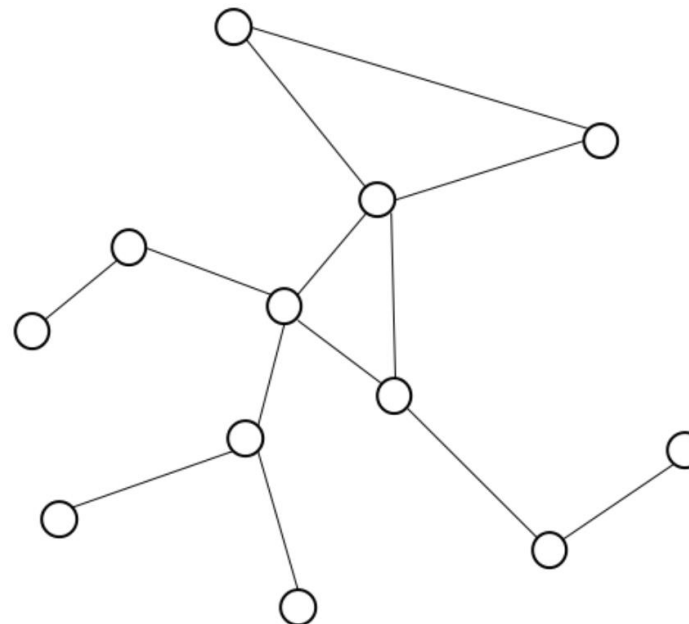
Graph – $G(V, E)$



What is GCN - Convolution on Graph



Regular Euclidean Data (Grid)
Normal convolution / pooling,
etc.

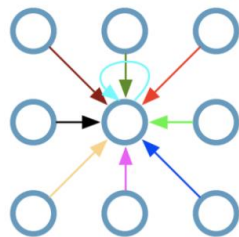
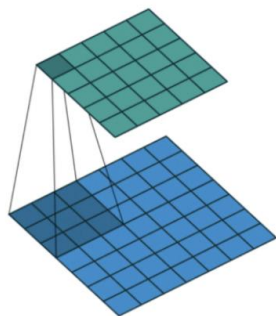


Graph in Non-Euclidean Space:
Convolution?
Pooling?

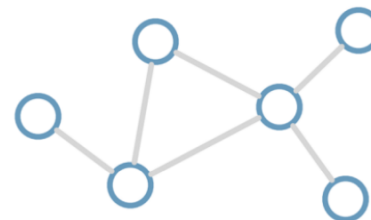


How to Define Convolution on Graph?

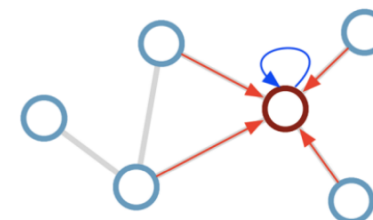
Single CNN layer
with 3x3 filter:



Consider this
undirected graph:



Calculate update
for node in red:



A **unified formulation** of convolution on images and graphs:

Updated
vertex/pixel $\mathbb{R}^{c'}$

$$x'_i = x_i W_0 + \sum_{j=1}^k x_j W_j$$

The vertex/pixel we
are working on \mathbb{R}^c

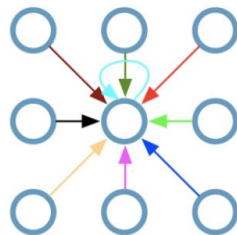
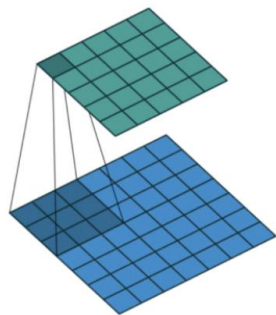
Neighboring
vertex/pixel \mathbb{R}^c

Trainable parameter $\mathbb{R}^{c \times c'}$

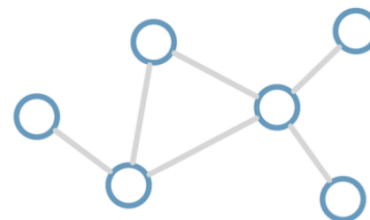


How to Define Convolution on Graph?

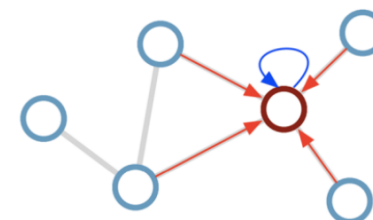
Single CNN layer
with 3x3 filter:



Consider this
undirected graph:



Calculate update
for node in red:



- Problem solved? No!
 - Number of neighbor is not fixed on graph.
 - DGCNN for Point Cloud: Fixed neighbor number
 - There are edge weights not utilized.
 - DGCNN for Point Cloud: Assume identical edge weight. (Ignore edge weight)



- EdgeConv
 - Aggregation of neighboring vertex

Aggregation function, e.g.,
sum, avg, min, max

Edge function (feature transform),
with trainable parameter θ

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

The vertex we are working on

Vertices that are connected to x_i



EdgeConv

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

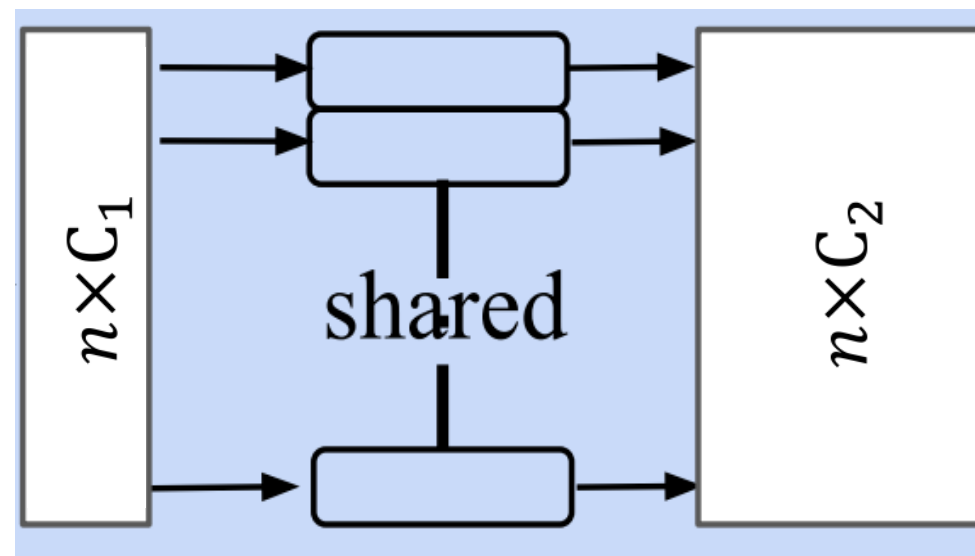
$$h_{\Theta}(\mathbf{x}_i, \mathbf{x}_j) = h_{\Theta}(\mathbf{x}_i)$$

Identity
Function

MLP

Point Feature, \mathbb{R}^m

PointNet





EdgeConv

Maxpool

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

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

MLP

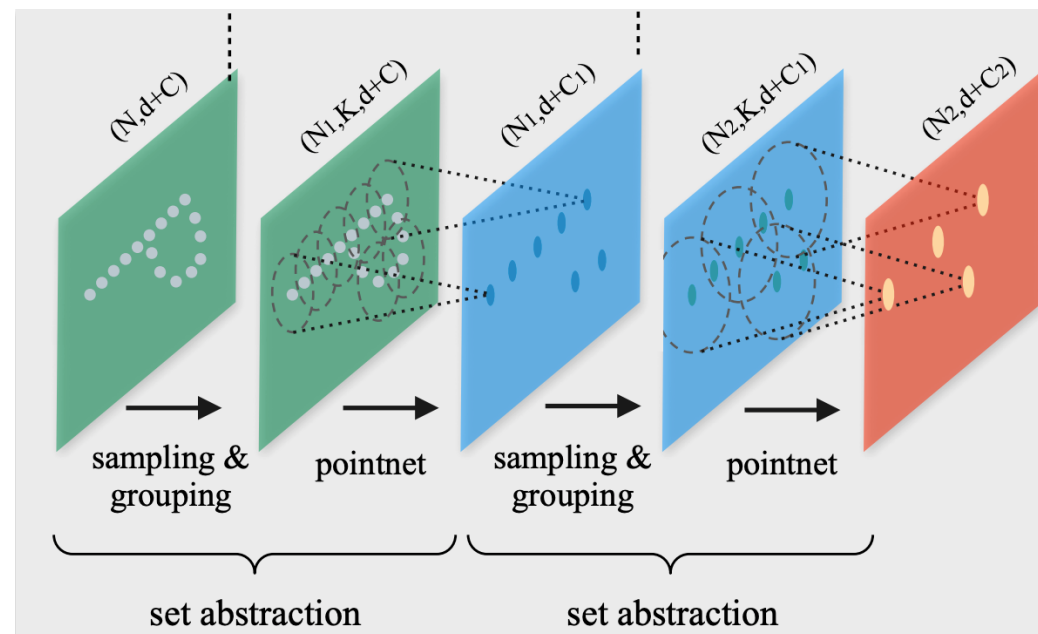
Features of
dimension C, C_1, C_2

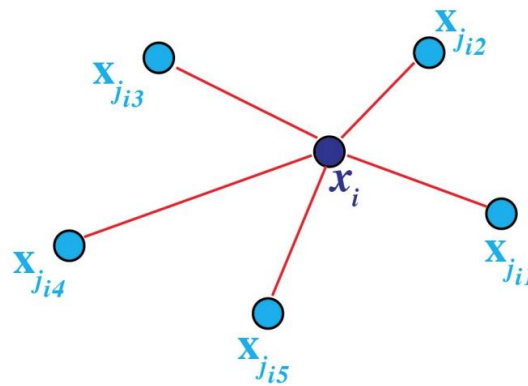
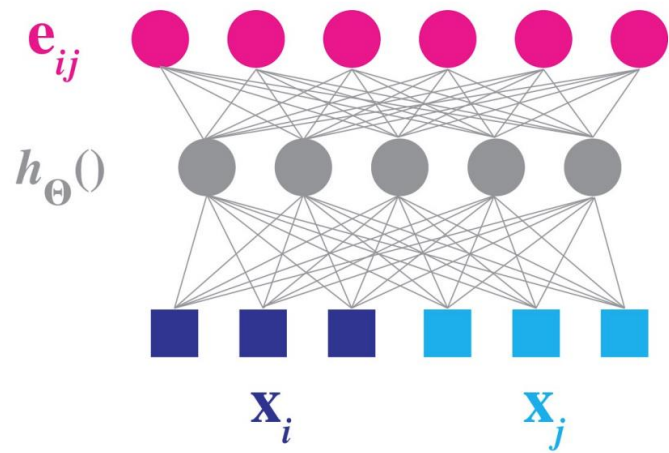
$$h_{\Theta}(\mathbf{x}_i, \mathbf{x}_j) = h_{\Theta}(\mathbf{x}_j - \mathbf{x}_i)$$

MLP

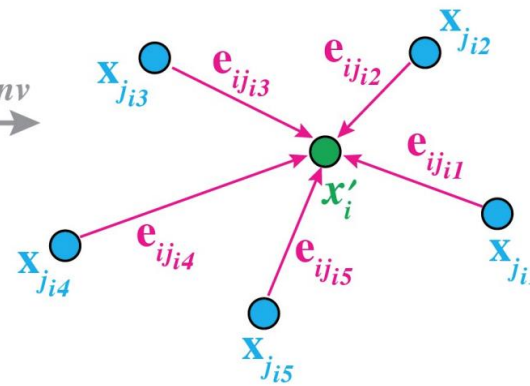
Coordinate of
dimension $d = 3$

PointNet++





EdgeConv



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

MLP with trainable param Θ

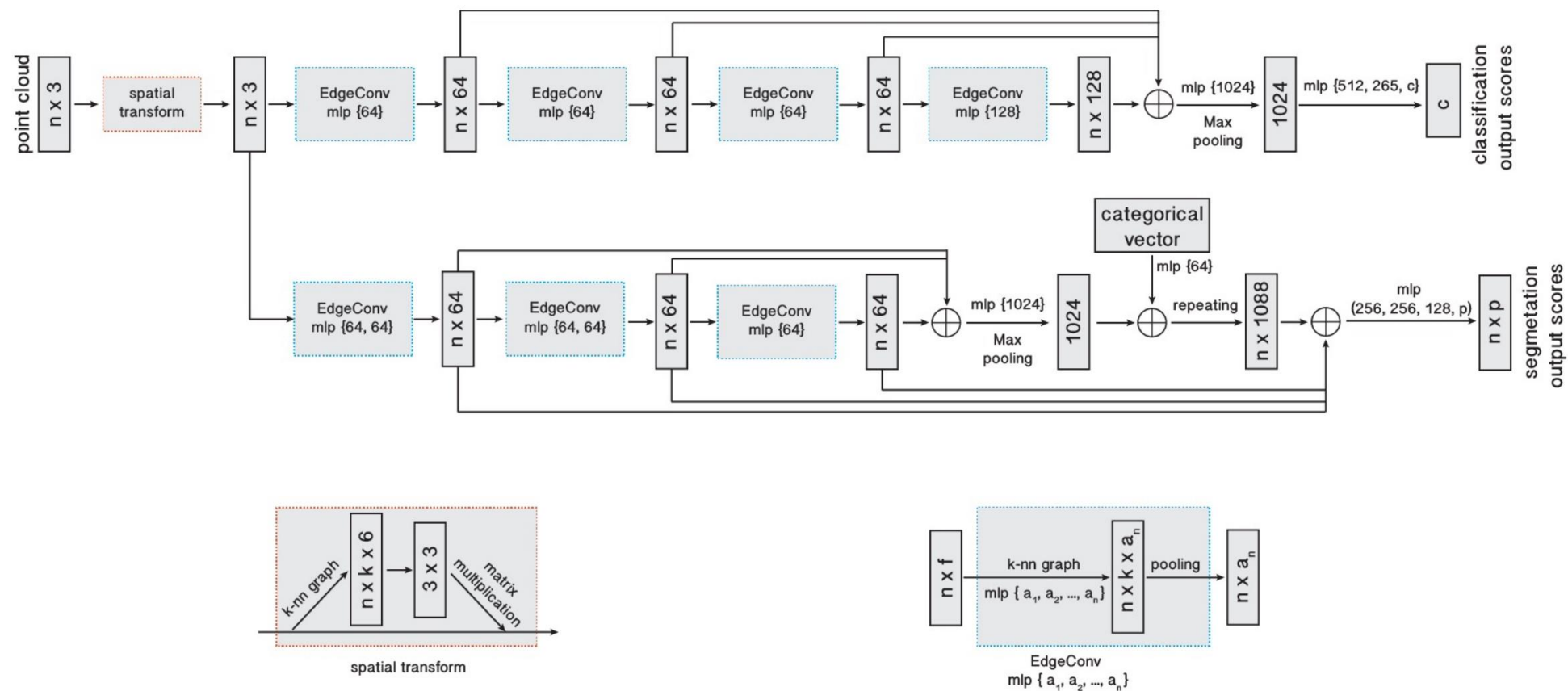
MLP with trainable param ϕ

$$h_{\Theta}(x_i, x_j) = h_{\Theta}(x_j - x_i) + h_{\phi}(x_i)$$

Neighboring
vertex, \mathbb{R}^m

The vertex we are
working on, \mathbb{R}^m

$$\mathbf{x}'_i = \maxpool_j \left(h_{\Theta}(\mathbf{x}_i, \mathbf{x}_j) \right)$$

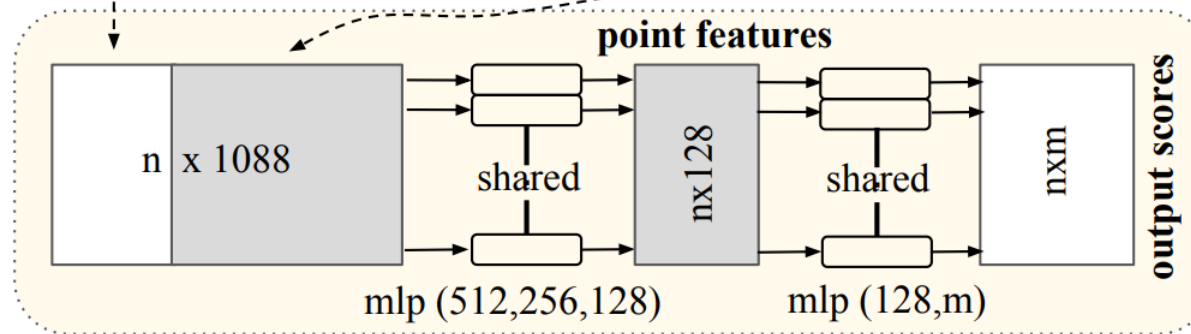
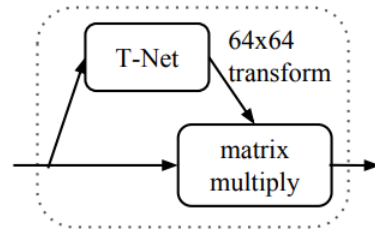
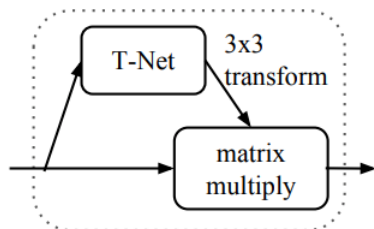
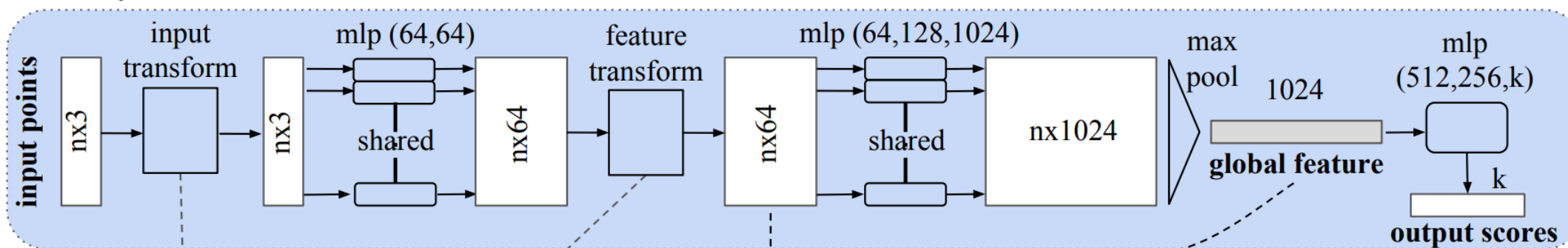


- Similar to T-Net of PointNet,
- NOT presented in the author's open-source code

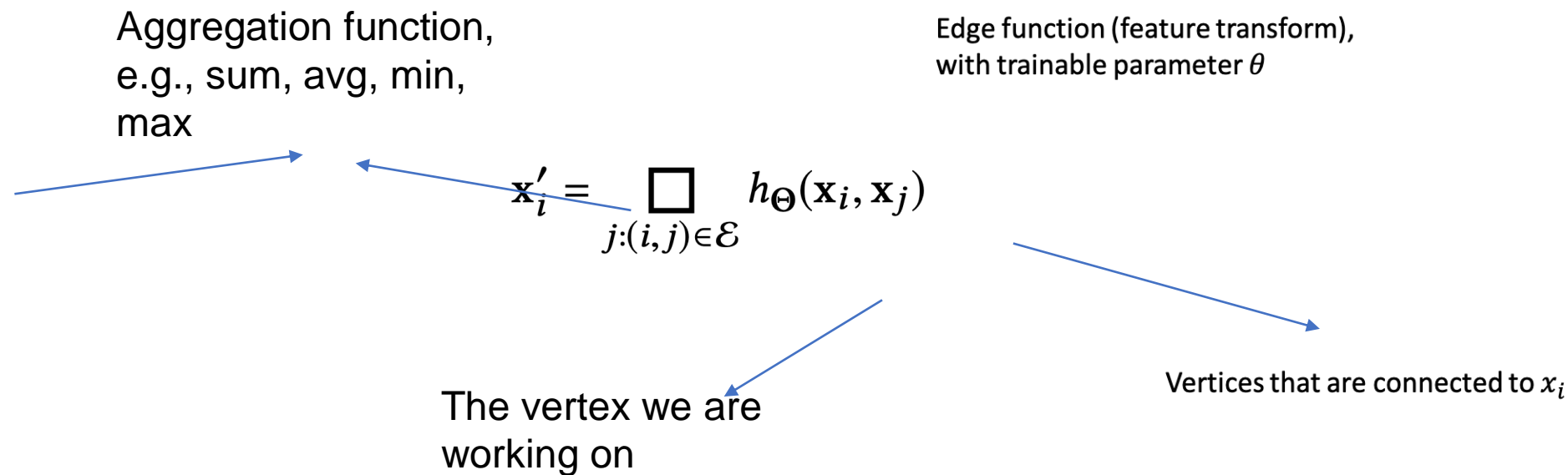
n points, each is $x'_i = \maxpool(h_\theta(x_j - x_i) + h_\phi(x_i))$



Classification Network



Segmentation Network



- How to define “neighbor x_j ”?
 - For first EdgeConv – kNN on coordinates ($x_i \in \mathbb{R}^3$)
 - For subsequent EdgeConv – **kNN on features** ($x'_i \in \mathbb{R}^m$)
 - Graph is **dynamic**, not static.



- Left

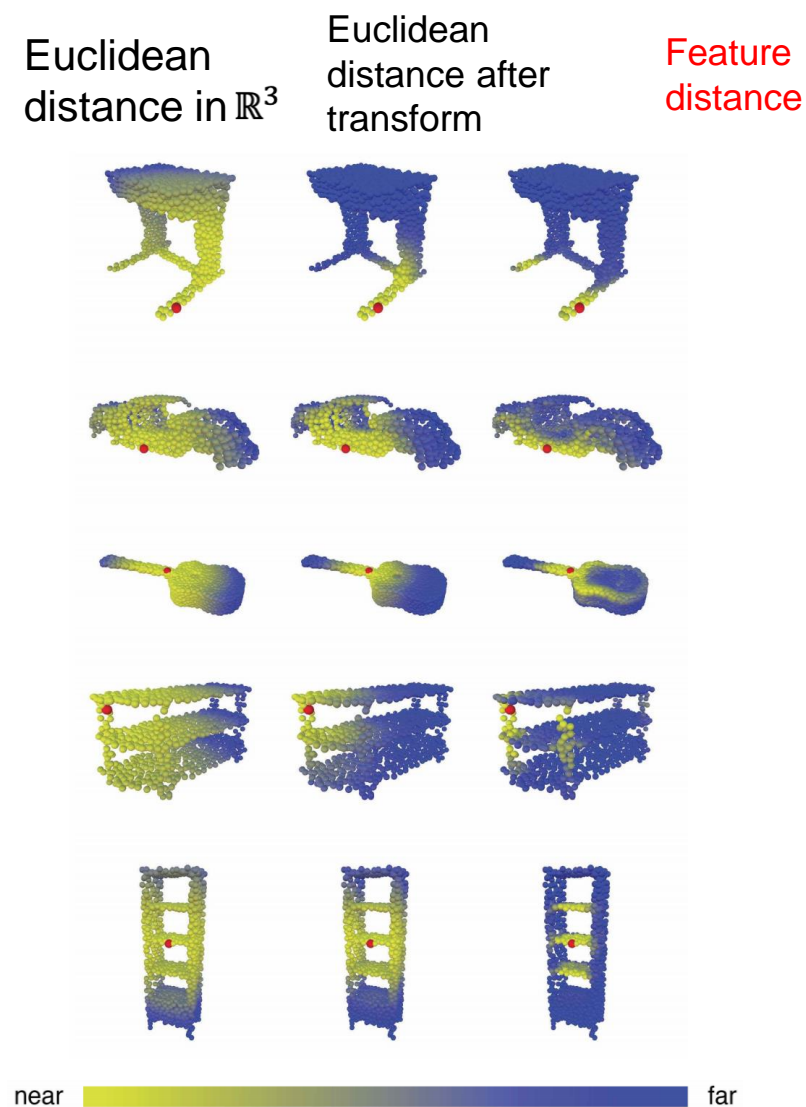
- Euclidean distance in \mathbb{R}^3

- Middle

- Euclidean distance in \mathbb{R}^3
 - After 3×3 matrix transform

- Right

- Feature distance in \mathbb{R}^3
 - From the last layer
 - Semantic & dynamic





Classification on ModelNet40 – Outperforms PointNet++

| | MODEL SIZE(MB) | TIME(MS) | ACCURACY(%) |
|---------------------------------------|----------------|----------|-------------|
| POINTNET (BASELINE) [QI ET AL. 2017B] | 9.4 | 6.8 | 87.1 |
| POINTNET [QI ET AL. 2017B] | 40 | 16.6 | 89.2 |
| POINTNET++ [QI ET AL. 2017C] | 12 | 163.2 | 90.7 |
| PCNN [ATZMON ET AL. 2018] | 94 | 117.0 | 92.3 |
| OURS (BASELINE) | 11 | 19.7 | 91.7 |
| OURS | 21 | 27.2 | 92.9 |

Table 3. Complexity, forward time, and accuracy of different models

| NUMBER OF NEAREST NEIGHBORS (K) | MEAN CLASS ACCURACY(%) | OVERALL ACCURACY(%) |
|---------------------------------|---------------------------|------------------------|
| 5 | 88.0 | 90.5 |
| 10 | 88.9 | 91.4 |
| 20 | 90.2 | 92.9 |
| 40 | 89.4 | 92.4 |

Table 5. Results of our model with different numbers of nearest neighbors.



- Graph structure is represented by **Similarity Matrix $A \in \mathbb{R}^{n \times n}$**

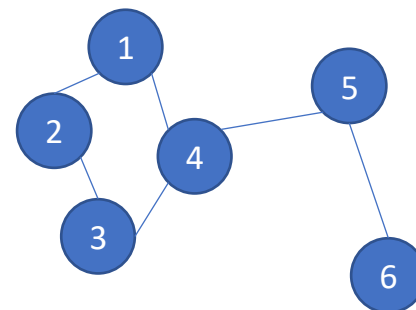
- Un-connected: 0
- Connected: similarity score, e.g., 1

- Concepts derived from A (Please refer to Lecture 4)

- Degree matrix $D \in \mathbb{R}^{n \times n}$
- Laplacian matrix $L = D - A \in \mathbb{R}^{n \times n}$
- Normalized Laplacian matrix
$$L_{sym} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}$$

- Why** do we need these A, D, L, L_{sym} ?

Graph



Connectivity / Similarity matrix A

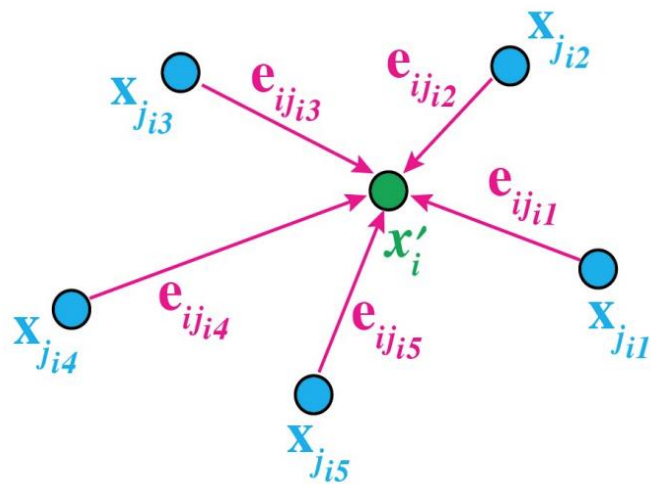
$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Degree Matrix D

$$\begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$



- Input: $X \in \mathbb{R}^{n \times C_{in}}$
- Output: $H \in \mathbb{R}^{n \times C_{out}}$
- A GCN layer consists of two steps
 - Aggregation
 - Gather features from neighbors
 - Update
 - Apply learnable layer to transform the aggregated features





General GCN—Similarity Matrix

- Input: $X \in \mathbb{R}^{n \times C_{in}}$
- Output: $H \in \mathbb{R}^{n \times C_{out}}$

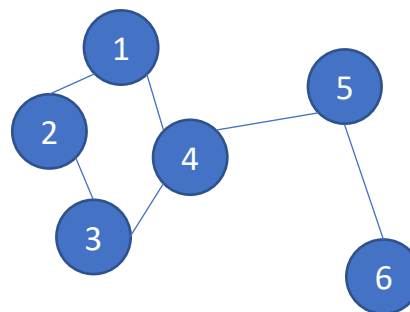
- Aggregation

- Weighted sum of **neighbors**
- $N = A \cdot X$

- Update

- **Linear (Fully Connected) Layer** with learnable param $W \in \mathbb{R}^{C_{in} \times C_{out}}$
- Activation function σ , e.g, ReLU
- $H = \sigma(N \cdot W) = \sigma(AXW)$

Graph



Connectivity / Similarity matrix A

| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 |



- $H = \sigma(N \cdot W) = \sigma(\textcolor{red}{A}XW)$

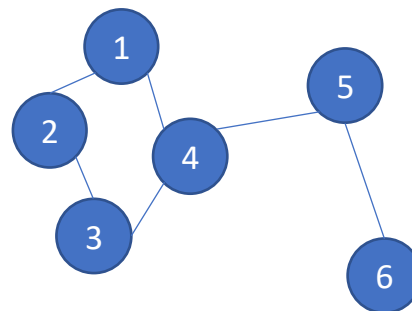
- **Problem:**

- Each node itself is not considered

- Diagonal elements of A is 0

- **Solution: Laplacian matrix**

Graph



Connectivity / Similarity matrix A

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

- Naïve weighted sum puts more weights on nodes with more connections

- A is not normalized

- **Solution: Normalized Laplacian matrix**



General GCN — Lapacian Matrix

- Input: $X \in \mathbb{R}^{n \times C_{in}}$
- Output: $H \in \mathbb{R}^{n \times C_{out}}$

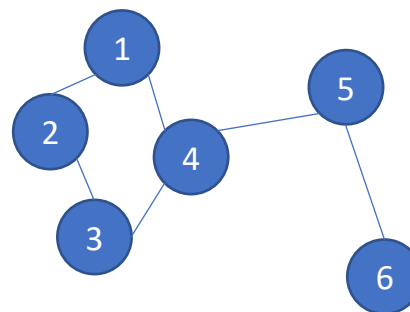
- Aggregation

- Weighted sum of **neighbors and itself**
- $N = (D - A) \cdot X = LX$
- In vector form: $N_i = \sum_j (A_{ij}(X_i - X_j))$

- Update

- **Linear (Fully Connected) Layer** with learnable param $W \in \mathbb{R}^{C_{in} \times C_{out}}$
- Activation function σ , e.g, ReLU
- $H = \sigma(N \cdot W) = \sigma(LXW)$

Graph



Laplacian Matrix L

$$\begin{bmatrix} 2 & -1 & 0 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ -1 & 0 & -1 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$



General GCN — Normalized Laplacian Matrix

- Input: $X \in \mathbb{R}^{n \times C_{in}}$
- Output: $H \in \mathbb{R}^{n \times C_{out}}$

- Aggregation

- **Normalized** weighted sum of **neighbors and itself**

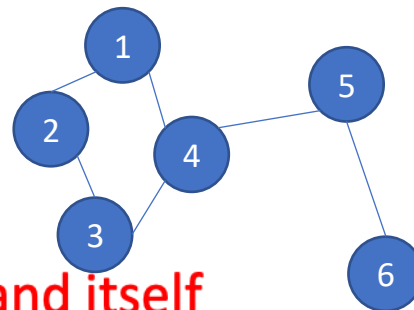
- $N = D^{-1/2} L D^{-1/2} \cdot X = L_{sym} X$

- In vector form: $N_i = \sum_j \left(A_{ij} (X_i - X_j) \cdot \frac{1}{\sqrt{D_{ii} D_{jj}}} \right)$

- Update

- **Linear (Fully Connected) Layer** with learnable param $W \in \mathbb{R}^{C_{in} \times C_{out}}$
 - Activation function σ , e.g, ReLU
 - $H = \sigma(N \cdot W) = \sigma(LXW)$

Graph



Normalized Laplacian Matrix L_{sym}

$$\begin{bmatrix} 1 & -\frac{1}{\sqrt{2}\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}\sqrt{3}} & 0 & 0 \\ -\frac{1}{\sqrt{2}\sqrt{2}} & 1 & -\frac{1}{\sqrt{2}\sqrt{2}} & 0 & 0 & 0 \\ 0 & -\frac{1}{\sqrt{2}\sqrt{2}} & 1 & -\frac{1}{\sqrt{2}\sqrt{3}} & 0 & 0 \\ -\frac{1}{\sqrt{2}\sqrt{3}} & 0 & -\frac{1}{\sqrt{2}\sqrt{3}} & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -\frac{1}{\sqrt{2}\sqrt{1}} \\ 0 & 0 & 0 & 0 & -\frac{1}{\sqrt{2}\sqrt{1}} & 1 \end{bmatrix}$$



DGCNN vs. General GCN

- DGCNN

- For each i
- MLP on neighbors \rightarrow get **multiple** feature vectors
- **Maxpool** \rightarrow get final feature vector for i

$$h_{\Theta}(x_i, x_j) = h_{\Theta}(x_j - x_i) + h_{\phi}(x_i)$$

MLP with param Θ MLP with param ϕ

Neighboring vertex, \mathbb{R}^m The vertex we are working on, \mathbb{R}^m

$$x'_i = \maxpool_j \left(h_{\Theta}(x_i, x_j) \right)$$

- General GCN

- For each i
- Aggregation: Weighted **sum** of neighbors \rightarrow get **one** feature vector
- Update: MLP on the feature vector \rightarrow get final feature vector for i

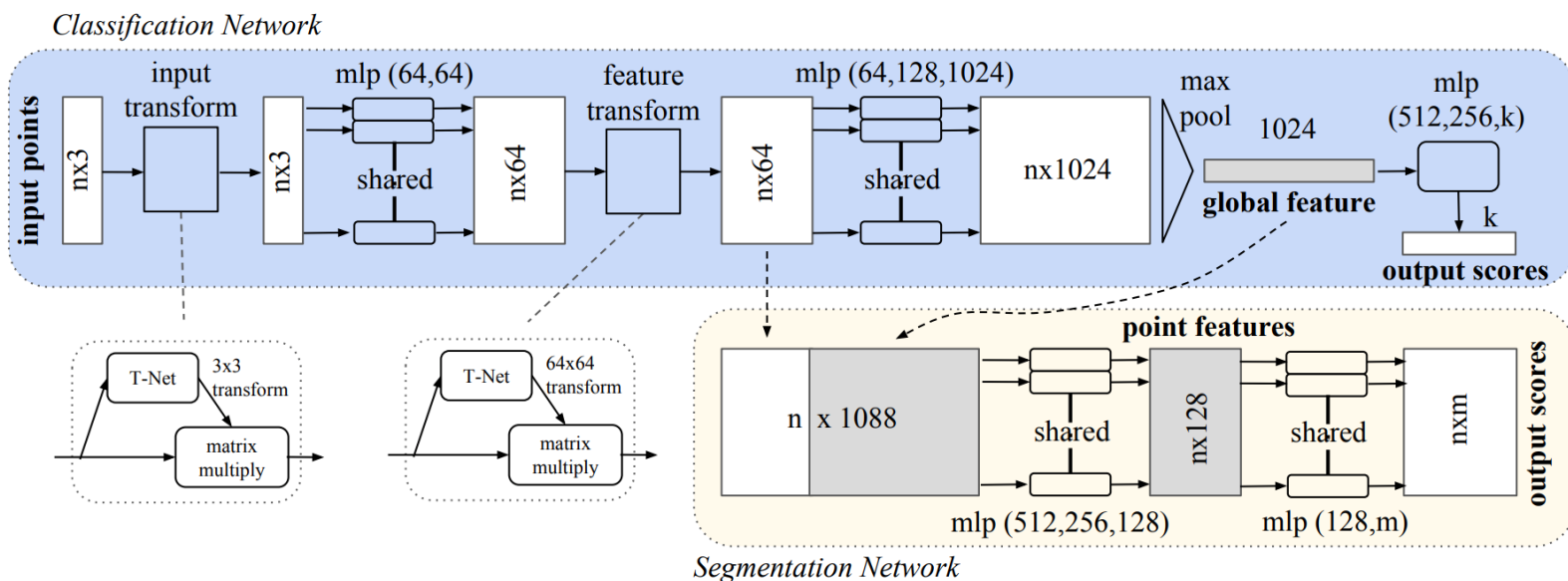


Some Other Comments On Deep Learning



Classification vs. Semantic Segmentation

- Similarity
 - Both of them are classification by Softmax
- Difference
 - Classification – Per Object. Segmentation – Per Point
 - Network design
 - Classification: Global feature
 - Segmentation: Global + Local feature





- **Want practical tricks/experience?**

- Do it yourself
 - Dataset preparation
 - Design, write, train & tune your own network
 - Don't be scared. DL is simple for implementation.
 - You have to experience it to get "experience"
- Develop your ability to invent "tricks"
 - You won't get a good job by saying you know lots of tricks

- **Common methods?**

- Core ideas are valid for a long time
 - PointNet series, voxel grid, data augmentation...
- Methods/Networks are different every company, every year.

- **Some critical topics in industry**

- Data/repo management
 - Coding ability
- Runtime
 - TensorRT
 - Hardware-aware network design
 - Network Pruning, Quantization, etc.
- Performance optimization
 - Active learning, e.g., data mining
 - Network Architecture Search (NAS)