Dynamic Graph CNN for learning on Point Clouds.

Mérigot-Lombard Matthieu ; Vindry Guillaume

Learning on point clouds

Point cloud from the ModelNet40 dataset

Tasks:

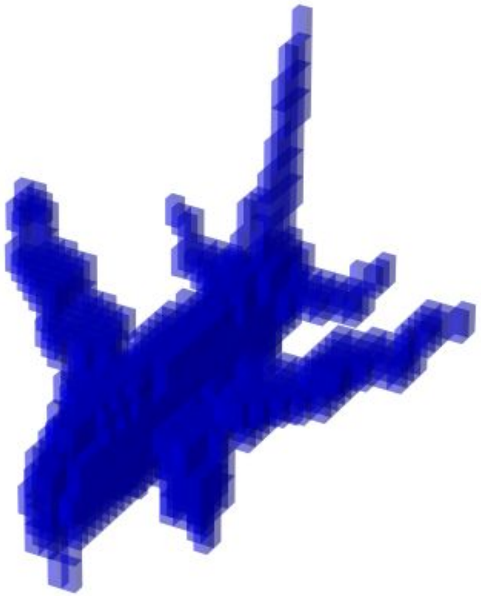
- **Classification**: Assign a label to an entire point cloud (identifying an object as a car or a tree)
- **Segmentation**: Partition the point cloud into meaningful regions (separating ground, vehicles, and pedestrians in an urban scene)

Applications:

- **Autonomous Navigation**: Use LiDAR sensors to map and understand the environment for self-driving cars or robotic systems
- **Medical Imaging**: Analyze 3D scans such as MRI or CT data, to identify anatomical structures or detect anomalies

[...]

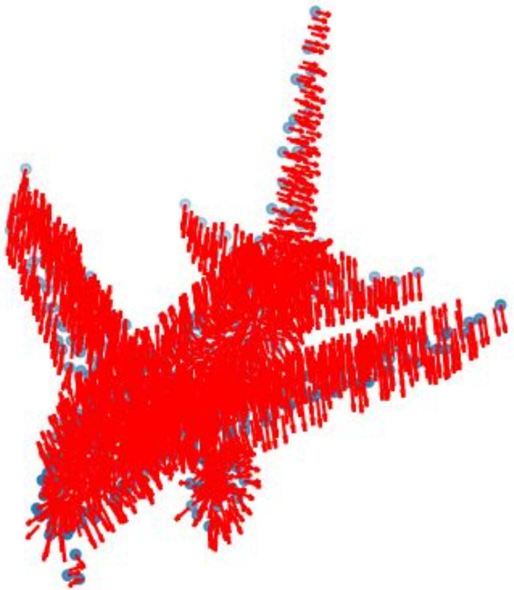
Handcrafted geometric features



Voxels

Transform the point cloud before applying a deep learning model:

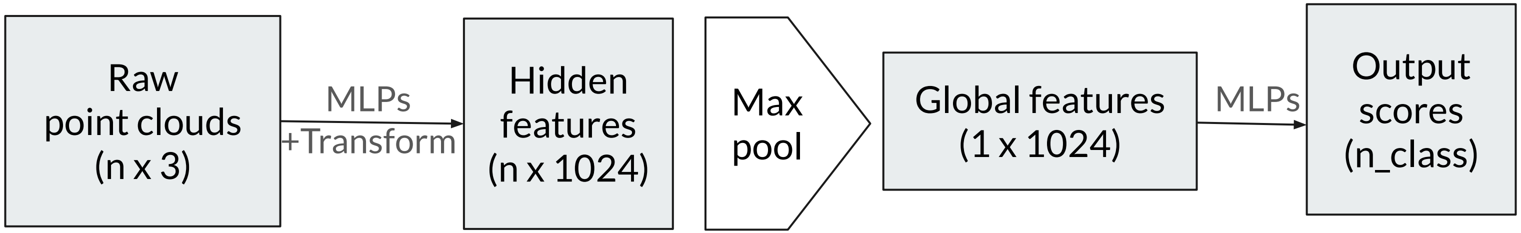
- **Normals**: Compute surface normals
- **Voxelization**: Discretize the point cloud into a 3D grid of voxels
- **Grid Representation**: Use other methods to project the point cloud into a structured grid, enabling convolution operations



Normals

PointNet and PointNet++

PointNet architecture



Extension to PointNet++

Step 1: Operates **hierarchically** by applying PointNet to **local neighborhoods**, capturing finer-grained geometric structures

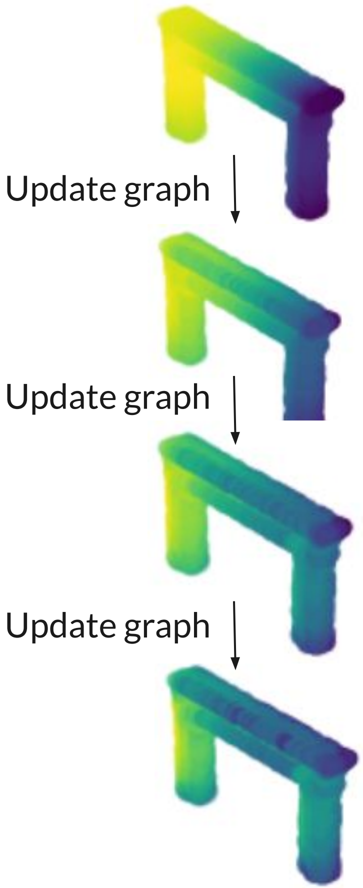
Step 2: Successively **pools local features**, creating coarser representations until summarizing the **entire cloud**

Dynamic Graph CNNs main ideas

Input: Raw Point Cloud Input
Retain the input as a raw point cloud, preserving the **original spatial and geometric information**

Step 1: Local Feature Extraction
Construct a graph within the **local neighborhood of points** to capture local features. Apply graph operations to update point features

Step 2: Non-Local Interactions
Dynamically update the graph as features evolve, enabling interactions between distant but similar points, enhancing **non-locality**



Local feature extraction: Edge Convolution

Objective:

Capture **local geometric structure** by updating point features based on these relationships

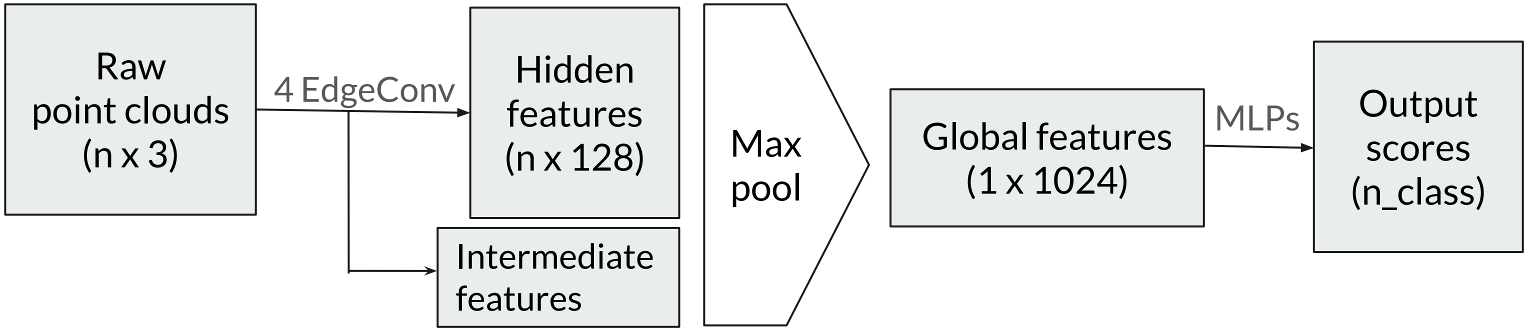
General EdgeConv definition

$$\text{EdgeConv}(\mathbf{x_i}) = \underset{x_j \in \mathcal{N}_i \cup \{x_i\}}{\square} h_\Theta(x_i, x_j)$$

$$\text{EdgeConv}_\Theta^{\text{PointNet}}(x_i) = \text{MLP}_\Theta(x_i)$$

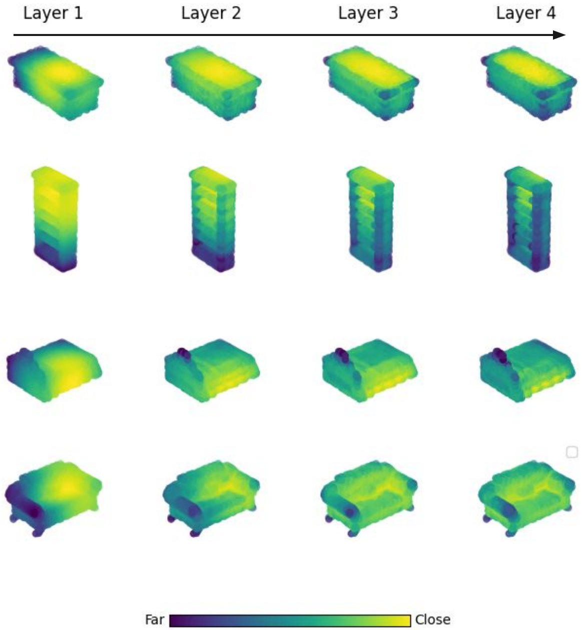
$$\text{EdgeConv}_{\theta,\phi}^{\text{DGCNN}}(x_i) = \max_{x_j \in \mathcal{N}_i \cup \{x_i\}} \text{ReLU}(\theta \cdot (x_j - x_i) + \phi \cdot x_i)$$

DGCNN Classification Architecture

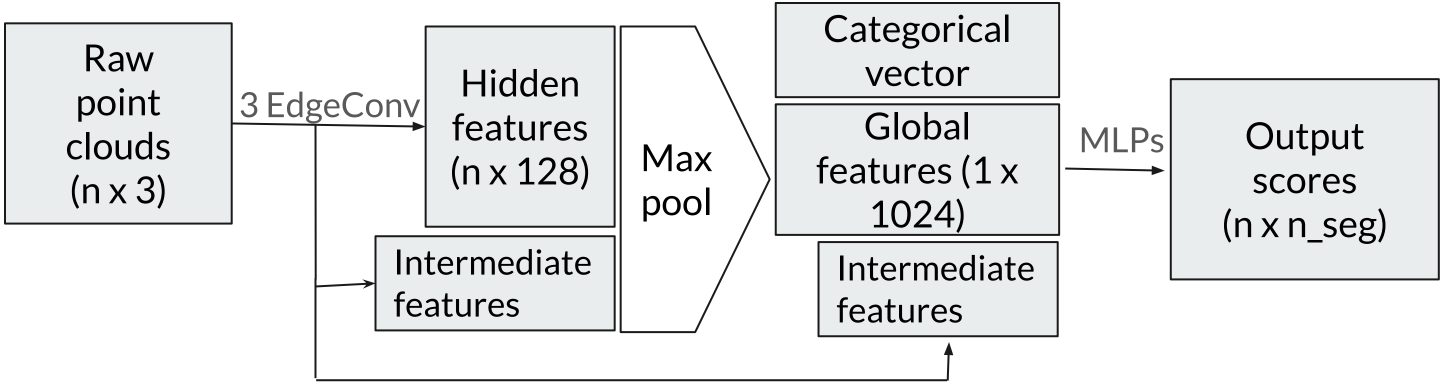


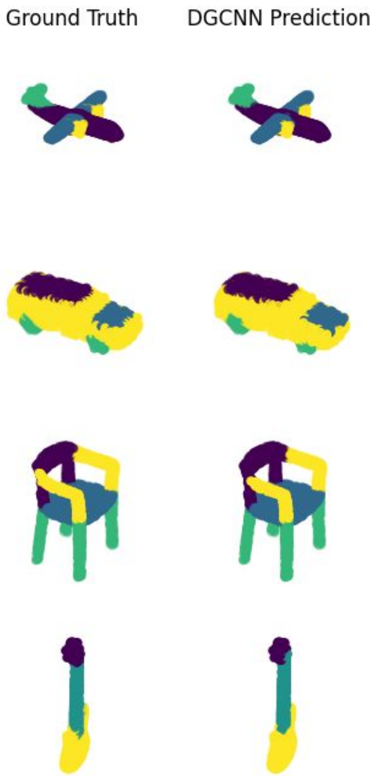
The graph is updated multiple times to capture **interactions between distant but similar points** helping the model detect **non-local patterns**.

Dynamic graphs : evolution of the distance metric for the k-nearest neighbors



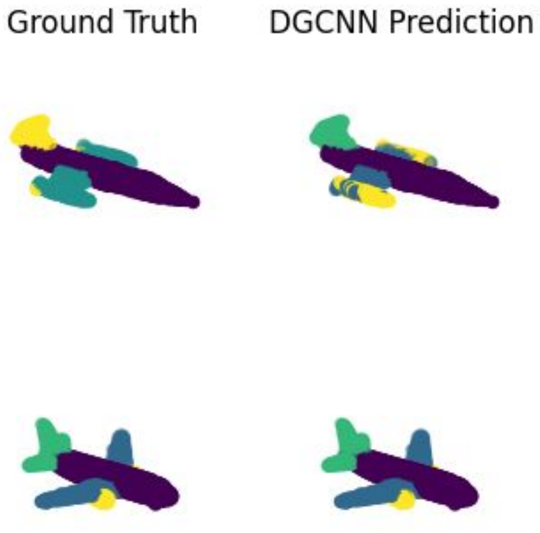
DGCNN Segmentation Architecture





Similar Core Structure: Builds on the previous architecture.
Categorical Vectors: Incorporates vectors providing information about the number of segments per class.

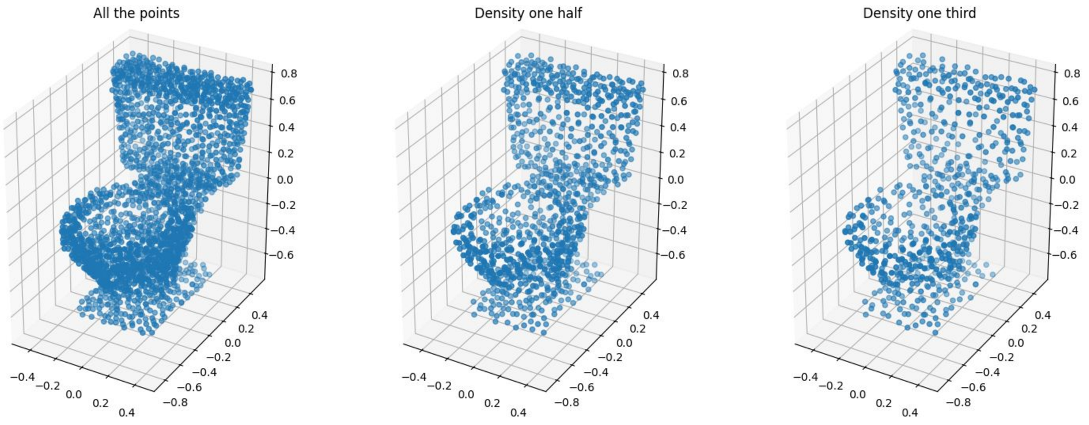
Fixed number of class of segmentation

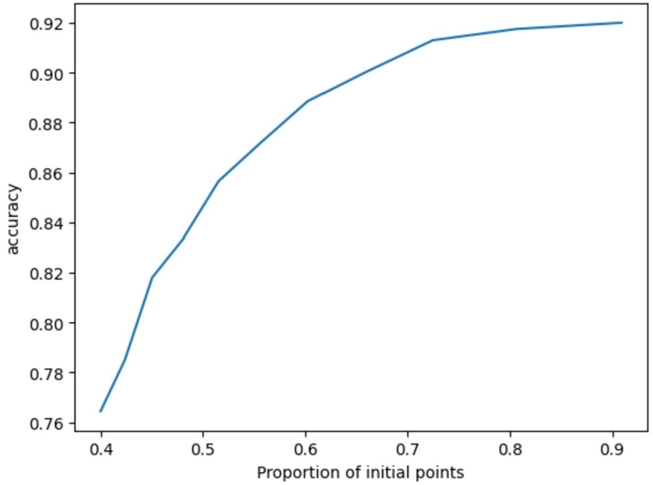


Potential loss of generalization: Fixed number of segmentation classes per object class which may not capture variations in complexity or structure

Scalability Issue: Adding point clouds with new segmentation classes would probably require retraining

Dependency to the number of points and density





Model on fixed number of points: k-NN approach could struggle with point clouds with different sizes as the one it is trained on
→ Could **modify k** to compensate

Density inhomogeneity: k-NN could also struggle as the number of necessary neighbors to generate an expressive feature can vary

Efficiency of k-NN computations

Drawback: k-NN calculation is **computationally intensive** and **challenging to parallelize**

Performance Comparison: From our computations, PointNet runs **~50x faster** during inference than DGCNN for classification task with 2048 points

Efficiency: From the authors, dynamic graph computation improved accuracy from PointNet by **less than 1%**