PointNet for 3D Shape Analysis

Pau Azpeitia Bergós

January 30, 2026

Abstract

In recent years, new and increasingly efficient strategies for acquiring 3D data have emerged, the ability to efficiently process and analyze 3D geometric data has become crucial for applications ranging from autonomous driving to medical imaging. Traditional methods have often relied on converting 3D data into voxel grids or image collections, introducing quantization artifacts and computational overhead. This paper reviews PointNet, a deep learning architecture that consumes raw, unordered point clouds directly. The theoretical foundation in permutation invariance is analyzed, specifically its relation to the DeepSets framework, along with an explanation of how symmetric functions are used to approximate global shape features. Furthermore, PointNet is compared with hierarchical approaches and modern Transformer-based methods, followed by a discussion of its lasting impact as a milestone in 3D computer vision.

Contents

1 Introduction

1.1 Motivation and Applications

The amount of 3D data available today has grown significantly because sensors like LiDAR and RGB-D cameras are becoming much easier to access. This data is essential for modern technology: self-driving cars use it to detect obstacles, robots use it to grab and manipulate objects, and AR/VR systems use it to reconstruct realistic environments.

However, standard Deep Learning models, such as Convolutional Neural Networks (CNNs), are designed to work with structured grids like images or 3D voxels. Point clouds are different; they are irregular sets of points without a fixed grid structure. Because of this, researchers traditionally had to convert point clouds into 3D grids or collections of images before the AI could process them. This conversion step is problematic because it makes the data unnecessarily large and introduces errors that can hide the true details of the shape.

To solve these issues, PointNet was introduced, a neural network designed to process raw point clouds directly [**Qi2017**]. By avoiding the need to convert the data into voxels, PointNet handles the input efficiently and respects the fact that the order of points does not matter. This results in a unified architecture that works effectively for tasks ranging from classifying objects to segmenting specific parts of a scene.

1.2 Challenges in 3D Data Processing

Processing 3D data presents unique challenges compared to 2D images. While images possess a regular grid structure, point clouds are irregular and unordered sets of vectors. A point cloud of N points has $N!$ possible permutations, all representing the same geometry. A robust neural network must be invariant to these permutations, as well as to rigid transformations like rotation and translation.

1.3 Main Tasks

Deep learning on 3D data generally focuses on two main tasks:

- **Classification:** Assigning a single semantic label to a 3D object.
- **Segmentation:** An extension of classification, that assigns a label to each point (shape segmentation) or partitions an object into functional parts (part segmentation).

1.4 Goal of the Paper

This paper examines PointNet [**Qi2017**] as a fundamental method for direct point cloud analysis. The mechanisms used to solve the permutation invariance problem are explored, and the architecture is compared to subsequent modern approaches. Before delving into the specifics of PointNet, it is essential to establish the theoretical background. This includes an overview of 3D data representations, the fundamental tasks of classification and segmentation, and key concepts such as Deep Sets.

2 Background: 3D Shape Analysis

2.1 3D Data Representations

To appreciate why PointNet was a significant breakthrough, we must first address the limitations associated with transforming raw point clouds into structured formats such as voxel grids or multi-view images. Therefore, this section defines these 3D data representations and outlines their fundamental differences [He+21].

RGB-D. Often referred to as 2.5D data, RGB-D combines standard color information (RGB) with a depth map (D) capturing the distance from the camera to the object. While useful for scene reconstruction and obtainable via sensors like Microsoft Kinect, it represents a single viewpoint rather than a complete 3D shape.

Voxel Grids. A voxel (volumetric pixel) is the 3D equivalent of a pixel. Voxel data represents scenes using a regular grid of volume elements, where each voxel stores attributes like occupancy or density. While this structure allows for standard 3D convolutions, it is computationally expensive and memory-intensive at high resolutions.

Meshes. Meshes represent 3D surfaces using a collection of vertices, edges, and faces (usually triangles). They are standard in computer graphics and animation because they efficiently capture surface topology. However, their irregular connectivity makes them difficult to process directly with standard neural networks.

Point Clouds. A point cloud is a set of data points in space, usually defined by coordinates (x, y, z) and often including attributes like intensity or RGB color. This is the raw output format of most 3D sensors, such as LiDAR and laser scanners.

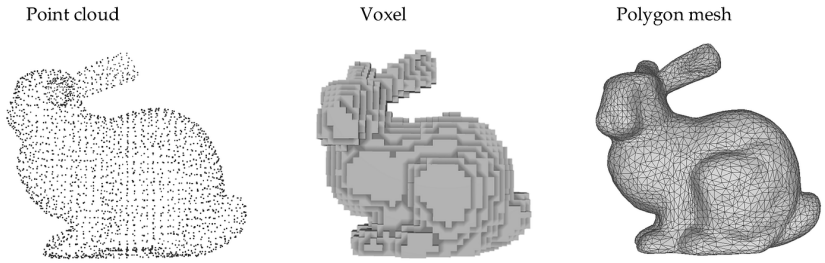

Figure 1: Visual comparison of 3D representations: Point cloud, voxel, and polygon mesh. Source: [Gao+22].

Comparison. Comparing these representations reveals significant trade-offs. RGB-D is limited by its single perspective, while meshes require complex preprocessing to handle their irregular graph structures. Voxel grids solve the structure problem but suffer from cubic complexity; increasing resolution drastically increases memory usage. Point clouds emerge as the optimal representation for 3D classification and segmentation. They are the native output of 3D sensors, preserving raw geometric information without the quantization errors of voxels or the complexity of mesh generation. Processing point clouds directly allows for efficient, high-fidelity learning, making them the preferred format for modern deep learning architectures like PointNet.

2.2 3D Classification and Segmentation

The analysis of 3D data is generally categorized into two primary tasks: classification and segmentation.

- **3D Classification:** The goal of classification is to assign a single global label to a 3D shape (e.g., identifying a scan as a "car" or "table"). This requires the network to extract a global feature vector that captures the overall geometry of the object.
- **3D Segmentation:** This is a fine-grained task where the goal is to assign a label to each primitive (point) in the data. It is essential for understanding the local structure of objects or scenes.

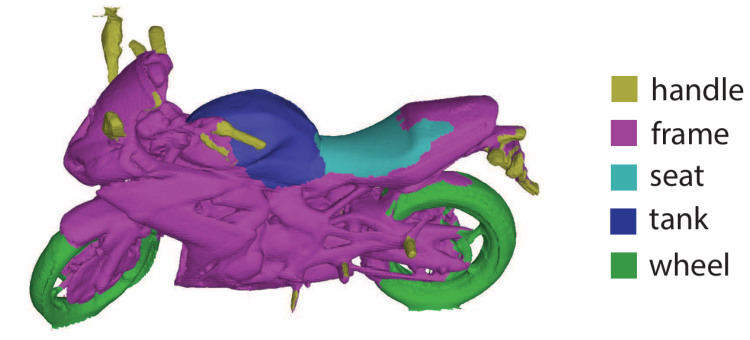

Figure 2: In segmentation, the network assigns a semantic label to each point, distinguishing functional parts. Source: [Kal+16].

Types and Datasets. Segmentation is further divided into:

- *Semantic Segmentation:* Labeling points based on category (e.g., road vs. building). Common datasets include S3DIS [Arm+16] and ScanNet [Dai+17].
- *Part Segmentation:* Decomposing an object into functional parts (e.g., legs of a chair). The ShapeNet dataset is the standard benchmark for this, containing thousands of models with part annotations.
- *Instance Segmentation:* Distinguishing between separate objects of the same class. For classification, ModelNet40 is the most widely used benchmark.

Challenges for Point Clouds. Processing point clouds is uniquely challenging because they are *irregular* and *unordered*. Unlike images which have a fixed grid structure, a point cloud is a set of vectors. A neural network must be invariant to the permutation of input points (the order in which points are fed) and robust to rigid transformations like rotation and translation.

Traditional Methods. Prior to deep learning, 3D analysis relied on hand-crafted features. Early deep learning approaches avoided the irregularity of point clouds by converting them into intermediate representations. These included *Multi-view* methods (projecting 3D shapes into 2D images) and *Voxel-based* methods (rasterizing points into volumetric grids). However, these conversions introduce quantization artifacts, lose geometric detail, or incur high computational costs.

Modern Trend: Direct Processing. To overcome these limitations, the field has shifted towards methods that consume raw point clouds directly without intermediate transformation. PointNet pioneered this direction, proposing an architecture capable of learning directly from unordered point sets, preserving the original geometric information.

2.3 Deep Sets and Permutation Invariance

A fundamental challenge in processing point clouds is that they are unstructured and unordered sets. Unlike images, where pixel $(0, 0)$ is always the top-left corner, a point cloud is represented as a set $S = \{x_1, x_2, \ldots, x_N\}$ with $x_i \in \mathbb{R}^d$. The order in which these points are stored in memory does not alter the geometric shape they represent.

Consequently, any deep learning architecture designed for point clouds must satisfy the property of Permutation Invariance. Formally, if f is the function learned by the network and π is any permutation of the indices $\{1, \ldots, N\}$, the network must satisfy:

$$f(\{x_1, \ldots, x_N\}) = f(\{x_{\pi(1)}, \ldots, x_{\pi(N)}\}) \tag{1}$$

This requirement restricts the types of architectures we can use. We cannot simply feed the point coordinates into a standard fully connected layer or a recurrent neural network (RNN) directly, as these are sensitive to input order.

The Deep Sets Theorem. Theoretical groundwork on learning from sets (often referred to as the Deep Sets framework) proves that a continuous set function f that is permutation invariant can be approximated by decomposing it into two learned functions, ϕ and ρ:

$$f(S) = \rho\left(\mathcal{A}_{x \in S}\{\phi(x)\}\right) \tag{2}$$

Where:

- ϕ is a transformation applied to each element x independently (typically a shared MLP).
- $\mathcal{A}$ is a *symmetric aggregation function* (such as sum, mean, or max) that effectively removes the ordering information.
- ρ is a post-processing function that operates on the aggregated global feature.

Relevance to PointNet. Understanding this theoretical framework is essential because PointNet is essentially a specific implementation of this theorem. PointNet employs Multi-Layer Perceptrons (MLPs) to approximate ϕ and uses Max Pooling as the symmetric function $\mathcal{A}$ to aggregate information from all points. This mathematical foundation guarantees that the network can universally approximate any continuous set function, validating its capability to classify unordered point clouds.

3 PointNet

3.1 Motivation and Key Idea

As mentioned before, PointNet was a turning point in the field of graphic analysis. The ability to process point clouds directly as inputs has led to significant improvements and advanced numerous applications, which are detailed in later sections. Having established the necessary context, the core principles of the architecture can now be explained. Fundamentally, PointNet is a deep learning framework that directly consumes unordered point sets as inputs and produces as output, depending on the objective, a label to classify the set or distinct labels for each point for segmentation.

3.2 Architecture Overview

The PointNet architecture is designed as a dual-branch network that shares a common feature extraction backbone. The pipeline accepts n raw points as input and processes them through a series of Multi-Layer Perceptrons (MLPs) and learned transformation matrices (T-Nets). Note that I have simplified the original architecture to make the explanation clearer, since my goal is to orient it toward an audience with little or no prior experience in this field, which was my own situation before I started studying PointNet. Figures 3 and 4 (manually created by me) are therefore not entirely identical to the originals presented in the paper; nevertheless, although simplified, they are faithful to the method and the idea conveyed by their authors.

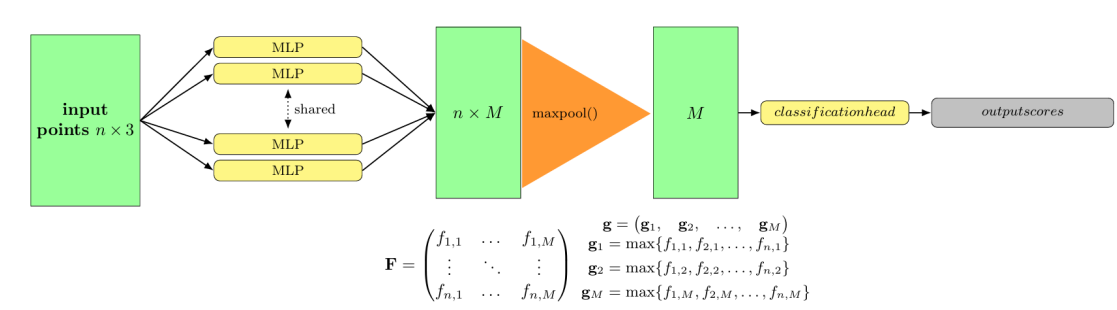

Figure 3: PointNet architecture diagram for classification, showing the feature matrix F and the global feature vector g. Here, n denotes the number of input points, 3 represents the point coordinates (x, y, z), and M is the dimension of the learned feature vector for each point. The classification head MLP takes the global feature vector and transforms it through learned weights and biases to produce a score for each possible class.

3.3 Overall Method Flow

To provide a clear understanding of the method, the complete flow is first explained for classification, followed by an extension to the case of segmentation.

As previously mentioned, the input to the network is directly the point cloud, represented as an $n \times 3$ matrix. Each point is defined, in its most basic form, by its three spatial coordinates (x, y, z). The main challenge lies in the fact that point clouds are unordered: the neural network must be capable of processing the points correctly regardless of their order.

As shown in Figure 3, the first major step of the architecture is the feature transformation. Each point is processed individually by a Multi-Layer Perceptron (MLP) that is identical for all points. In other words, the same MLP is applied independently to each of the n points.

Technically, the MLP is a shared function h. As specified in the network architecture (see Figure 2 in [Qi+17b]), this transformation is implemented via two sequential shared MLP blocks: the first with layer sizes $(64, 64)$ and the second with sizes $(64, 128, 1024)$. This structure receives the 3-coordinate input vector and progressively maps it into a higher-dimensional feature space (resulting in a 1024-dimensional vector), effectively unpacking the compressed geometric information contained in the raw coordinates. The overall function learned by the network can be expressed as:

$$f(\{x_1, \ldots, x_n\}) \approx g(h(x_1), \ldots, h(x_n)),$$

where h is applied independently to each point, and g is a symmetric aggregation function. As demonstrated by Theorem 1 (Universal Approximation) in [Qi+17b], this architecture is theoretically capable of approximating any continuous set function. This guarantees that the network can learn effectively because small perturbations in the input point cloud will not cause drastic changes in the output.

This step is crucial because it transforms raw spatial coordinates into implicit local geometric features, such as relative position or structural potential. Importantly, at this stage, there is still no interaction between points. Each point is analyzed independently, which preserves permutation invariance. If the network were to compare points directly at this stage, the ordering of points would affect the result.

It is worth noting that the original architecture includes an additional crucial component known as T-Nets (Joint Alignment Networks), which are omitted here for clarity. In the full model, these mini-networks are inserted before the MLP blocks to predict transformation matrices (both 3×3 and 64×64). They align the input points and intermediate features to a canonical space, ensuring the network remains invariant to geometric transformations such as object rotation.
In the context of the Deep Sets framework, this stage effectively implements the function ϕ. By processing points independently, it ensures that permutation invariance is preserved before the aggregation step.

After the shared MLP, an intermediate feature matrix of size $n \times M$ is obtained, where each of the n points is represented by an M-dimensional feature vector.

The next key component of the architecture is the max pooling operation, which is the core idea behind the network. Since the goal is to obtain a global representation of the object that is independent of point ordering, a symmetric function is required. The max pooling operation selects, for each feature channel, the maximum value across all points.

Intuitively, this is similar to observing a crowd and selecting the person with the largest value for a particular attribute (e.g., the largest nose), regardless of their position or order in the crowd. This operation reduces the n feature vectors to a single global feature vector that represents the entire object.

In terms of the Deep Sets framework previously discussed, this max pooling operation implements the symmetric aggregation function $\mathcal{A}$. By collapsing the set of features in a permutation-invariant manner, it satisfies the critical requirement for processing unordered sets.

This global feature vector is then passed to a standard neural network, referred to as the classification head. This network outputs a probability distribution over the predefined classes, and the class with the highest probability is selected as the classification result. Finally, the output scores (logits) are transformed into a valid probability distribution using the Softmax function. This allows the network to be trained using a classification loss and ensures that the outputs sum to one. As we can see, this final step provides the last component ρ required to complete the Deep Sets theorem. This proves that the PointNet architecture is a specific implementation of it, effectively solving the permutation invariance challenge.

Choice of Symmetric Function. While the Deep Sets theoretical framework allows for any symmetric function $\mathcal{A}$ to guarantee permutation invariance, the authors demonstrated that Max Pooling is the optimal choice for 3D point clouds. They compared Max Pooling against Mean Pooling and Sum Pooling. The results showed that Max Pooling significantly outperformed the others due to its unique geometric properties.

Unlike averaging, which aggregates information from all points (including noise and empty space), Max Pooling acts as a strict feature selector. It captures only the most prominent features—effectively identifying the "critical set" or the *skeleton* of the shape—while ignoring weaker signals. This makes the architecture exceptionally robust to data corruption and varying point densities; as long as the critical points (e.g., the corners of a table) are preserved, the global feature vector remains stable, whereas an average would be skewed by missing data or outliers.

To ensure stable training and robust generalization, the PointNet architecture systematically incorporates the following components across its processing pipeline:

- ReLU Activation: The standard non-linear activation function applied after each linear transformation. It enables the model to learn complex, non-linear mappings while maintaining computational efficiency.
- Batch Normalization (BN): Applied to all layers, including both the point-wise shared MLPs and the global classification head. BN is critical for stabilizing the distribution of activations throughout the network, which accelerates convergence and provides a slight regularizing effect.
- Dropout: Specifically utilized in the final fully connected layers of the classification head. By randomly deactivating units during training, it prevents the model from over-relying on specific feature combinations, thereby reducing overfitting.

3.4 Extension to Segmentation

For segmentation, things become more complicated. Now the output cannot simply be the most probable label/class; instead, we must return that label for each point. We achieve this by modifying/adding to the architecture as follows:

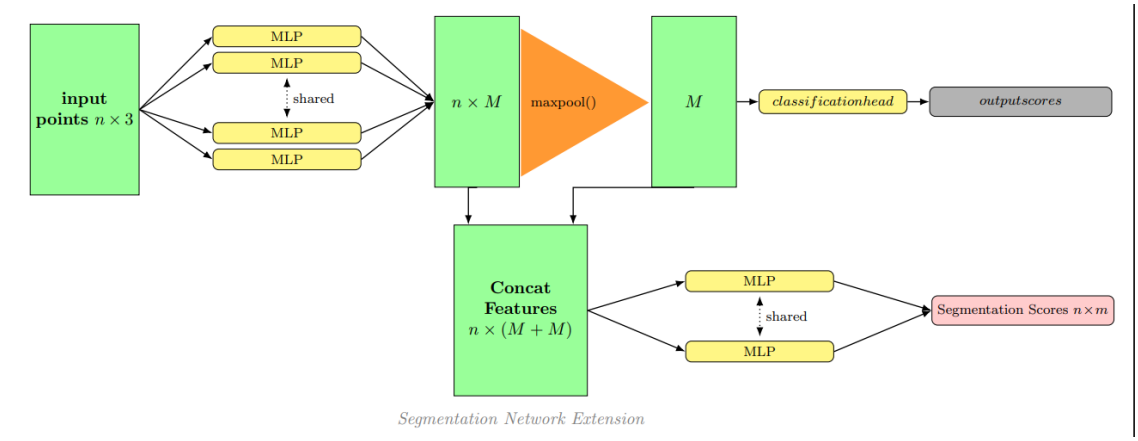

Figure 4: PointNet architecture extended for semantic segmentation. The top section illustrates the classification stream where global features are aggregated via max pooling. To perform segmentation, the global feature vector (M) is duplicated and concatenated with the local point features $(n \times M)$ extracted from the intermediate layers. This combined representation $(n \times (M+M))$ captures both local geometry and global context, allowing the subsequent shared MLPs to predict semantic scores $(n \times m)$ for each individual point.

As said, while the classification network summarizes the input point cloud into a single global feature vector to predict a category for the entire shape, semantic segmentation requires a fine-grained prediction for each individual point. To achieve this, the network must rely on both local geometric structures and global context.

As illustrated in the Segmentation Network extension, we simply extend the classification architecture by reintroducing the local point features that were computed before the max pooling operation. Specifically, the global feature vector is duplicated and concatenated with the local feature vector of each point. This results in a combined feature vector for every point that encapsulates both its local geometry and the global semantic context of the object.

These combined features are then processed by new shared MLP layers to extract point-wise features. Finally, a shared fully connected layer outputs the semantic label scores for each point, effectively transforming the global understanding of the shape into detailed per-point predictions.

3.5 A Concrete Walkthrough: Processing a Chair

To solidify the understanding of the architecture, let's trace the dimensions of the data as a single 3D object passes through the network. Suppose our input is a point cloud representing a chair, consisting of $n = 1024$ points.

1. Input State (1024×3):
   We start with a matrix of size 1024×3. Each row is a point (x, y, z) on the chair. At this stage, the network sees 1024 isolated coordinates with no concept of "legs" or "seat".

2. Feature Transformation (Local Features):
   The points pass through the shared MLPs. The network expands the information of each point.

   - First, it might map, for example, 3 coordinates to 64 features (1024×64).
   - Finally, it maps them to a high-dimensional space of 1024 features.

   Current State: A matrix of 1024×1024. We still have 1024 individual points, but each is now described by a rich vector of 1024 learned mathematical properties (e.g., curvature, verticality).

3. Max Pooling:
   The symmetric function is applied. We can understand this process also as a bottleneck. For each of the 1024 feature columns, we pick the single highest value among all 1024 points. This is the Global Feature, a single descriptor for the entire chair.
   Current State: A vector of 1×1024.

4. Branch A: Classification:
   This 1×1024 vector goes through the final MLP to produce scores for k classes (e.g., 40 classes in ModelNet40).
   Output: A vector of 1×40. If the "Chair" index has the highest value, the network predicts "Chair".

5. Branch B: Segmentation:
   To label the legs of the chair, we need to go back to n points.

   - We take the Global Feature (1×1024) and duplicate it 1024 times (1024×1024).
   - We retrieve the Local Features from Step 2 (let's say we took the layer with size 1024×64).
   - Concatenation: We glue them together. Each point now has its specific 64 local features + the 1024 global features of the chair.

   Current State: A matrix of 1024×1088.
   The network now processes this to output a label for each point.
   Final Output: A matrix of $1024 \times m$ (where m is the number of part types, e.g., 4: leg, seat, back, arm).

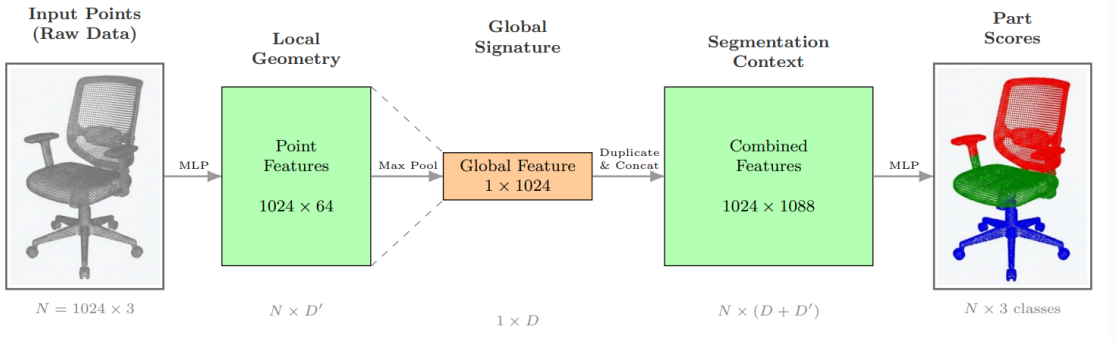

Figure 5: Visual walkthrough of the data flow for a chair instance ($N = 1024$). The diagram illustrates the tensor transformations at each stage, highlighting how the Global Feature (1×1024) is concatenated with local geometry to enable per-point segmentation predictions.

3.6 Strengths and Limitations

We have already seen and illustrated how PointNet works, so we can now concisely list and explain the strengths and limitations of this neural network. I have chosen to highlight the most notable attributes for this purpose.

3.6.1 Strengths

Computational Efficiency PointNet is highly efficient in terms of memory usage and computational speed compared to voxel-based or multi-view methods. With a time complexity of $O(N)$, it can process over one million points per second on a standard GPU, making it suitable for real-time applications [Qi+17b].

Robustness to Data Corruption The architecture demonstrates remarkable stability against input data corruption. Due to the use of a symmetric max-pooling function that selects only a critical set of points (the "skeleton" of the object), the network maintains high classification accuracy even with 50% missing data or significant outlier noise [Guo+20].

Permutation Invariance As mentioned before, PointNet was the pioneering deep learning architecture to natively solve the problem of unordered point sets. By processing points independently and aggregating them via a symmetric function, it guarantees consistent outputs regardless of the input order.

3.6.2 Limitations

Lack of Local Context The most significant limitation of the original PointNet is its inability to capture local structures induced by the metric space. Since the network learns features for each point independently until the global max-pooling step, it fails to recognize fine-grained geometric patterns and relationships between neighboring points. This limitation motivated the development of hierarchical architectures such as PointNet++ [Qi+17a].

Sensitivity to Global Transformations Although PointNet handles small noise well, it struggles when the entire object is rotated. Since the coordinates change completely with rotation, the network does not automatically recognize it is the same shape. As mentioned in section 3.3, PointNet attempts to solve rotation issues with T-Nets, which align the object to a standard position before processing. However, adding this extra step makes the training process more difficult.

Fixed Global Scale The network processes the entire point cloud at a single scale. This makes it challenging to generalize to complex scenes where objects appear at different scales or densities, a problem later addressed by methods (again PointNet++ [Qi+17a], which will be comparing later) employing multi-scale grouping.

3.7 Applications

Following the analysis of the core theoretical tasks supported by PointNet—primarily object classification and segmentation—this section explores real-world applications where this methodology has significantly improved efficiency and precision. However, in this section I would like to go a step further and highlight real-world examples of situations where, thanks to this methodology, tasks that were previously not performed with such efficiency and precision have been carried out more effectively.

Robotic Disassembly A standout example of PointNet in the industry is its use to help robots automatically identify and disassemble car parts, specifically turbochargers. In this kind of task, standard 2D cameras often fail because they cannot capture the full 3D structure, and used parts can vary a lot in appearance.

The most innovative part of this work was how they solved the data problem. Instead of manually scanning thousands of real physical parts—which is very slow—they created "synthetic" 3D scans directly from digital computer models (CAD). They even simulated the typical glitches and noise that real depth sensors produce. The results showed that PointNet could correctly identify 12 different complex mechanical parts (like housings and bearings) more than 90% of the time, even when the robot could only see a partial view or the data was noisy. This proves that PointNet is a robust tool for automating remanufacturing processes that used to be slow manual bottlenecks. These conclusions were drawn from this work: [Zhe+22]

Autonomous Driving Self-driving cars rely heavily on LiDAR scanners to see the world in 3D. A major challenge is that these scanners produce massive point clouds representing the entire street, which are too large for the original PointNet to process all at once effectively.

To address efficiency, researchers initially developed "Attentional PointNet" [Pai+19]. This method functions like a flashlight in a dark room: instead of processing the entire scene, it uses a visual attention mechanism to focus on small, specific 3D boxes. A standard PointNet then analyzes these regions to detect cars, enabling real-time performance.

However, relying only on one detection source can be risky. To improve safety and robustness, recent advancements like Multi-View Frustum PointNet (MVFP) [Cao+19] combine PointNet with other sensor data. While standard methods often rely on RGB cameras to tell PointNet where to look (Frustum PointNet), they can fail in poor lighting or shadows. MVFP solves this by adding a Bird's-Eye View (BEV) detector that scans the raw LiDAR data from above. If the camera misses a pedestrian or cyclist due to shadows, the BEV module detects the object and feeds it back to the PointNet system for confirmation. This multi-view approach significantly reduces missed detections, making autonomous navigation safer in complex urban environments.

Medical Diagnosis & Genetics Finally, it is fascinating to see how PointNet works beyond physical 3D objects like cars or chairs. A recent study [Lu+24] applied this technology to medicine, specifically for diagnosing cancer.

Typically, doctors analyze genetic data (gene expression) using complex spreadsheets or standard charts. However, these researchers had a creative idea: they treated the genetic data as if it were a point cloud. Instead of coordinates in a room (x, y, z), the "points" represented biological information about genes. They fed this abstract "cloud of genes" into a PointNet-based model called Gene PointNet. The result? The AI could classify tumors with over 99% accuracy. This proves that PointNet's ability to understand unordered sets of data is a powerful tool not just for robots, but also for saving lives.

4 Comparison with Other Methods

4.1 PointNet++

As mentioned in the limitations section, the original PointNet is relatively ineffective at capturing local relationships. Since it processes each point independently until the final stage (Max Pooling), it fails to recognize that contiguous points share significant geometric connections. To address this, the same authors developed PointNet++ [Qi+17a]. This can be summarized in one sentence: PointNet++ was presented as a direct extension of PointNet to tackle its main limitation: the failure to capture local context within point clouds. But how does it solve this?

PointNet++ introduces an idea that proved highly successful in image CNNs: analyzing small patches and then aggregating them. Here, the famous "Divide and Conquer" strategy applies. Instead of processing the entire point cloud at once, PointNet++ first partitions the cloud into many small spheres and applies a "mini-PointNet" to each sphere to understand the precise shape of that local instance [Qi+17a]. It then groups these patches into larger spheres and re-applies PointNet, repeating this process until the complete shape is formed.

This achieves two key improvements. First, it establishes a distinct hierarchical structure. It learns local features and fine details, progressively combining them to form global features [Qi+17a]. This works particularly well for understanding complex scenes, not just isolated objects. Furthermore, when comparing PointNet with PointNet++, we observe that in distant objects within a scene—which have few points—and nearby objects—which have many—the original PointNet struggled to distinguish the density differences. In contrast, PointNet++ successfully handles varying densities using mathematical functions such as Multi-Scale Grouping (MSG) to robustly process regions regardless of whether they have many or few points [Qi+17a].

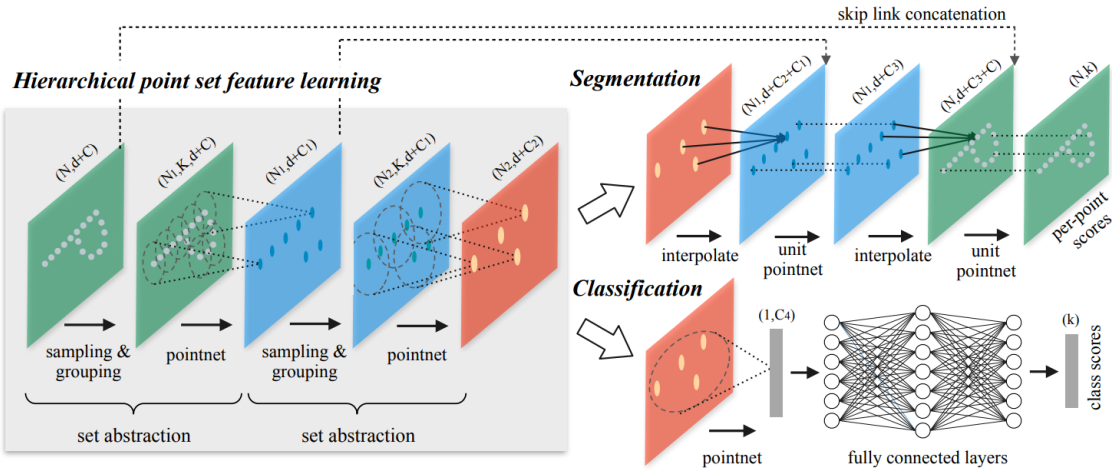

Figure 6: The PointNet++ architecture for visual comparison with PointNet. Extracted directly without modification from the original paper [Qi+17a].

4.2 Voxel-based methods

Before PointNet, the standard way to analyze 3D data was a technique called voxelization. Methods like 3D ShapeNets [Wu+15] and VoxNet [MS15] converted the cloud of points into a 3D grid of blocks, exactly like pixels in a photo but with volume.

This trick allowed computers to use the same Artificial Intelligence tools that already worked well for normal 2D photos. However, this "blocky" approach has two major problems that PointNet solves:

- Loss of Detail: To fit the points into a grid, the computer has to round off their positions to the nearest block. This makes smooth objects look jagged and pixelated, causing the loss of fine geometric details.
- Huge Memory Cost (Cubic Complexity $O(N^3)$): The main problem is efficiency. If you want to capture small details, you need a grid with very tiny blocks (high resolution). This increases the number of blocks to process massively (cubically). For example, a small grid of $30 \times 30 \times 30$ is easy to handle, but a high-definition grid of $256 \times 256 \times 256$ requires huge amounts of memory and computing power, even if most of those blocks are just empty air (sparsity).

PointNet processes the points directly, avoiding these bulky grids. This keeps the original shape perfect without wasting computer memory on empty space.

4.3 Graph-based and local convolution methods

While PointNet processes points independently, newer methods attempt to recreate the connections between neighboring points, similar to how pixels in an image relate to each other. We can highlight two main approaches that outperform the original PointNet:

1. Graph-based Methods: Here I have chosen the most widely used and most important method in this field: DGCNN. Imagine PointNet treats points like strangers in a crowd. Dynamic Graph CNN (DGCNN) [Wan+19] treats them like a social network. It connects each point to its nearest neighbors, creating a "graph" where information flows between them. Crucially, this graph is dynamic: the connections change deeper in the network as the AI learns which points are semantically related (e.g., grouping all points belonging to a car's wheel), not just physically close. The main advantage of DGCNN is that it explicitly captures local geometry (like edges, corners, and curves), whereas PointNet often misses these fine details because it processes points in isolation. However, PointNet retains the advantage in computational efficiency. Calculating these connections (graphs) at every step makes DGCNN significantly slower and heavier to run than the simple, linear processing of PointNet. Related to this comparison, I found the paper *Processing Laser Point Cloud in Fully Mechanized Mining Face Based on DGCNN* [Xin+21]. In this work, the authors demonstrate that their improved version of DGCNN has much lower position error (0.14m) compared to PointNet (0.72m) or PointNet++ (0.57m), which allows for safer autonomous mining.

2. Local Convolution Methods: Standard image recognition uses a sliding window (convolution) that moves over pixels. Doing this in 3D is hard because points are scattered. Methods like KPConv [Tho+19] solve this by placing flexible reference points (kernels) in 3D space. These act like 3D stamps or molds that check the local geometry around every point, allowing the network to detect fine details like corners or smooth curves much better than PointNet. While KPConv is far superior in detecting fine geometric details, PointNet wins in simplicity and speed. Implementing deformable kernels in 3D is mathematically complex and computationally expensive. PointNet, being a simple architecture based on basic matrix multiplications, is much lighter and faster to train, making it preferable for devices with limited processing power.

4.4 Modern methods

While investigating recent trends and the future direction of the state of the art, I came across a 2021 paper that, in my view, marks a definitive major change: the Point Transformer [Zha+21].
So far, we have seen how PointNet processes points efficiently but in isolation, and how subsequent methods attempted to "patch" this limitation by searching for local neighbors. However, this work proposes something radically different: bringing the Transformer revolution (the same technology underlying Large Language Models like ChatGPT) into the 3D world. The key concept here is the Self-Attention mechanism. Instead of using rigid filters or static groupings, this network allows each point in the cloud to dynamically decide which other points to "attend to" based on their semantic relevance, rather than just their physical proximity.

What convinces me that this is the future is not just the theory, but the empirical results. This model achieved state-of-the-art performance on major benchmarks such as S3DIS (the standard for indoor segmentation), significantly outperforming all PointNet variants and 3D convolution networks [Zha+21]. This demonstrates that, when hardware resources allow, the ability to understand global context through attention mechanisms is far superior to traditional local operations.

While Transformers are great for accuracy, they are heavy. For processing massive LiDAR scans of entire cities in real-time (Autonomous Driving), the industry standard is often Sparse Convolution (e.g., MinkowskiEngine [CGS19]). Unlike old voxel methods that waste memory processing empty space, Sparse Convolutions only compute features where points actually exist. This combines the efficiency of PointNet with the regularity of CNNs, making it the go-to solution for large-scale outdoor perception.

Method	**Input Rep.**	**Local Context**	**Complexity**	**Key Strength**	**Main Limitation**
PointNet	Raw Points	None (Global Pool)	$O(N)$ (Linear)	Efficiency & Speed	No local context
Voxel-based	Voxel Grid	3D Convolution	$O(N^3)$ (Cubic)	Mature CNN tools	Memory heavy & Artifacts
PointNet++	Hierarchical Points	Multi-Scale Grouping	$O(N)$ (Higher const.)	Multi-scale features	Sampling latency
DGCNN	Dynamic Graph	EdgeConv (k-NN)	$O(k \cdot N)$	Semantic grouping	Slow graph construction
Transformers	Points + Embeddings	Self-Attention	$O(N^2)$ or Heavy	SOTA Accuracy	Computational cost

Table 1: Trade-off analysis of 3D deep learning methods.

5 Conclusion

This paper has explored the trajectory of 3D deep learning, positioning PointNet not merely as another neural network architecture, but as a fundamental change in how machines perceive three-dimensional space. Before its introduction, the field was constrained by the need to adapt 3D data to 2D-centric architectures through voxelization or multi-view projection. These methods, while effective to a degree, imposed severe computational penalties and introduced quantization artifacts that obscured fine geometric details.

PointNet broke this barrier by proposing a remarkably elegant solution: accepting the raw, unordered nature of point clouds. Through the theoretical application of the Deep Sets framework, utilizing shared Multi-Layer Perceptrons and a symmetric Max Pooling function, PointNet solved the permutation invariance problem. This approach provides high-speed processing and significant robustness against data corruption, making it particularly suitable for real-time applications.

However, my analysis also highlighted the inherent limitations of this pioneering design. By processing points independently, the original PointNet suffers from a "local blindness," failing to capture the geometric relationships between neighboring points that define fine-grained structures. We discussed how this specific weakness catalyzed a wave of innovation, leading to hierarchical architectures like PointNet++, which applies the "divide and conquer" strategy to learn features at different scales, and graph-based methods like DGCNN, which dynamically link points to understand local topology.

Furthermore, looking at the current state of the art, we observe that the field is moving in two distinct directions. On one hand, the rise of 3D Transformers represents the frontier of accuracy, achieving unprecedented results in complex segmentation benchmarks by enabling global context awareness. On the other hand, the industry—particularly in autonomous driving and robotics—often favors efficiency over marginal gains in precision. In this context, PointNet (and efficient derivatives like Sparse Convolutions) remains the industry standard for deploying AI on hardware-constrained devices.

Personally, I was struck by the simplicity of the entire neural network in relation to the significant progress it represented. When I first started looking for topics for this seminar, the field seemed overwhelming due to my limited experience. However, studying PointNet revealed a remarkable elegance: the architecture is not merely a heuristic solution, but a direct, clean specification of the Deep Sets theorem. Seeing how abstract mathematical components (ϕ, ρ, and $\mathcal{A}$) translate so naturally into standard deep learning layers gave me a clear intuition for the method. This combination of strong theory and simplicity makes the paper very easy to understand. I can now clearly see why my supervisor considers it one of his favorites.

In conclusion, while newer and more complex methods continue to emerge, PointNet remains the "Hello World" of 3D Deep Learning. Its simplicity, speed, and theoretical soundness make it the foundational building block upon which modern 3D computer vision is built. Understanding PointNet is, therefore, not just looking back at a historical milestone, but essential for understanding the present and the future of 3D Shape Analysis.

References

[Arm+16] Iro Armeni et al. "3D Semantic Parsing of Large-Scale Indoor Spaces". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 1543–1552.

[Cao+19] Pei Cao et al. "Multi-View Frustum PointNet for Object Detection in Autonomous Driving". In: *Proceedings of the IEEE International Conference on Image Processing (ICIP)*. 2019, pp. 3896–3900.

[CGS19] Christopher Choy, JunYoung Gwak, and Silvio Savarese. "4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3075–3084.

[Dai+17] Angela Dai et al. "ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 5828–5839.

[Gao+22] Mengran Gao et al. "Deep Neural Network for 3D Shape Classification Based on Mesh Feature". In: *Sensors* 22.18 (2022). DOI: 10.3390/s22187040.

[Guo+20] Yulan Guo et al. "Deep Learning for 3D Point Clouds: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.12 (2020), pp. 4338–4364.

[He+21] Yulan He et al. "Deep Learning Based 3D Segmentation: A Survey". In: *arXiv preprint arXiv:2103.05423* (2021).

[Kal+16] Evangelos Kalogerakis et al. "3D Shape Segmentation with Projective Convolutional Networks". In: *arXiv preprint arXiv:1612.02808* (2016).

[Lu+24] Hao Lu et al. "Gene pointNet for tumor classification". In: *Neural Computing and Applications* 36 (2024), pp. 21107–21121.

[MS15] Daniel Maturana and Sebastian Scherer. "Voxnet: A 3d convolutional neural network for real-time object recognition". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 922–928.

[Pai+19] Anshul Paigwar et al. “Attentional PointNet for 3D-Object Detection in Point Clouds”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2019.

[Qi+17a] Charles R Qi et al. “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space”. In: *arXiv preprint arXiv:1706.02413* (2017).

[Qi+17b] Charles R. Qi et al. “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 652–660.

[Tho+19] Hugues Thomas et al. “KPConv: Flexible and Deformable Convolution for Point Clouds”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 6411–6420.

[Wan+19] Yue Wang et al. “Dynamic Graph CNN for Learning on Point Clouds”. In: *ACM Transactions on Graphics (TOG)* 38.5 (2019), pp. 1–13.

[Wu+15] Zhirong Wu et al. “3d shapenets: A deep representation for volumetric shapes”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1912–1920.

[Xin+21] Zhizhong Xing et al. “Processing Laser Point Cloud in Fully Mechanized Mining Face Based on DGCNN”. In: *ISPRS International Journal of Geo-Information* 10.7 (2021), p. 482.

[Zha+21] Hengshuang Zhao et al. “Point Transformer”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 16259–16268.

[Zhe+22] Senjing Zheng et al. “Automatic identification of mechanical parts for robotic disassembly using the PointNet deep neural network”. In: (2022). Published online: 15 Mar 2022.