Graph-Based Deep Learning for Component Segmentation of Maize Plants

J. I. RUIZ[a], A. MENDEZ-VAZQUEZ[a], E. RODRIGUEZ-TELLO[b]

[a] *Cinvestav Unidad Guadalajara, Av. del Bosque 1145, Zapopan, 45017, Jalisco, Mexico*
[b] *Cinvestav Unidad Tamaulipas, Km. 5.5 Carretera Victoria-Soto La Marina, Victoria, 87130, Tamaulipas, Mexico*

Abstract

In precision agriculture, one of the most important tasks when exploring crop production is identifying individual plant components. There are several attempts to accomplish this task by the use of traditional 2D imaging, 3D reconstructions, and Convolutional Neural Networks (CNN). However, they have several drawbacks when processing 3D data and identifying individual plant components. Therefore, in this work, we propose a novel Deep Learning architecture to detect components of individual plants on Light Detection and Ranging (LiDAR) 3D Point Cloud (PC) data sets. This architecture is based on the concept of Graph Neural Networks (GNN), and feature enhancing with Principal Component Analysis (PCA). For this, each point is taken as a vertex and by the use of a K-Nearest Neighbors (KNN) layer, the edges are established, thus representing the 3D PC data set. Subsequently, EdgeConv layers are used to further increase the features of each point. Finally, Graph Attention Networks (GAT) are applied to classify visible phenotypic components of the plant, such as the leaf, stem, and soil. This study demonstrates that our graph-based deep learning approach enhances segmentation accuracy for identifying individual plant components, achieving percentages above 80% in the IoU average, thus outperforming other existing models based on point clouds.

Keywords: 3D, LiDAR, computational vision, graphs, PointNet, clustering, point cloud, maize, Segmentation, EdgeConv.

Email addresses: jesus.ruiz@cinvestav.com (J. I. RUIZ), andres.mendez@cinvestav.com (A. MENDEZ-VAZQUEZ), ertello@cinvestav.com (E. RODRIGUEZ-TELLO)

arXiv preprint — June 2025

1. Introduction

Accurate identification of plant components, such as leaves, stems, and soil regions, are a crucial task in precision agriculture [1]. Therefore, precise plant component identification allows for better biomass estimation [2], targeted resource allocation [3], crop productivity [4], plant diseases discovery [5], and finally yield productivity estimation [6].

In order to address the problem of plant component identification, Deep Learning Neural Networks are proposed to identify the primary structures of plants. Deep Learning (DL) models have seen significant growth and evolution [7, 8], making them essential for point segmentation in DL novel projects.

A key advancement in Deep Learning has been the development of Convolutional Neural Networks (CNNs). Although CNNs have predominantly been applied to 2D image processing tasks [9], extending them to handle 3D data is challenging due to the irregular and unordered nature of point clouds datasets. Unlike pixels in 2D images, which possess a uniform neighborhood structure, points in 3D PC have varying neighbor densities and distributions, which complicate the direct application of CNN models. Previous studies, such as the work by Bernhard et al [10, 11], have explored methods to adapt conventional 2D CNN-based algorithms for effective processing of 3D data. In contrast, other novel models took a different approach by processing the point clouds directly. For example, the novel model PointNet [12] is a pioneer in this field. Unlike traditional convolutional models, it operates directly on raw point datasets without voxelization or projection. On the other hand, it employs symmetric aggregation functions such as max pooling and T-Net transformation to maintain permutation invariance. Therefore, PointNet learns global features while preserving the geometric structure in point cloud data.

However, PointNet has limitations in capturing fine-grained local structures, and the relationships among neighboring points, which are crucial for segmenting complex geometries. In order to overcome these shortcomings, other models have been proposed, such as Graph Convolutional Networks (GCN) [13]. These models take the PC dataset and convert it into a graph, where each point acts as a vertex, and edges are defined by local neighborhood relationships. This representation allows for explicit modeling of both local and global dependencies through message passing, offering a more flexible framework for tasks that involve complex spatial structures [14]. In

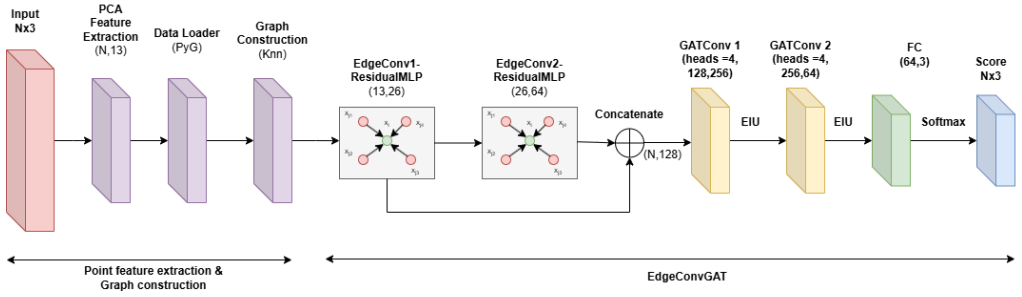

Figure 1: Our proposed model is a graph based EdgeGAT architecture. The first stage enhances point features by incorporating PCA-based features and constructing the graph representation. The second stage refines features using EdgeConv and leverages Graph Attention Networks for pretraining.

order to further improve the relationship between the points and their neighbors, another model such as Dynamic Graph CNN (DGCNN) [15] enhances this idea by dynamically constructing graphs based on feature space, allowing the network to update neighborhood connections at each layer. This dynamic construction, combined with edge-based convolutions, has proven highly effective for point cloud segmentation tasks, offering a more flexible and spatially aware representation than fixed neighborhood approaches.

Building upon these advancements, Graph Attention Networks (GAT) [16] introduce an attention mechanism into the graph-based learning paradigm, enabling the model to assign varying levels of importance to neighboring vertices during feature aggregation. Unlike GCN and DGCNN, which rely on uniform or implicitly learned weights for neighborhood aggregation, GAT explicitly compute attention coefficients for each edge in the graph. This allows the network to focus more on relevant points while down-weighting less informative ones. Such a mechanism is particularly beneficial in the context of point cloud segmentation, where certain regions may exhibit higher structural or semantic importance than others. By incorporating attention-based weighting, GAT enable the model to capture complex spatial patterns more effectively, adapting dynamically to the heterogeneity of plants point cloud data. A common limitation of these previous architectures is the insufficient explicit modeling of feature interactions among neighboring points, neglecting to fully exploit local geometric context.

In order to address this challenge, we propose a novel architecture named EdgeGAT. The general architecture of this model can be seen in Figure 1,

which leverages graph-based processing for enhanced point-cloud segmentation. For this, the proposed model applies Principal Component Analysis (PCA) as a preprocessing step. This step extracts geometric components present in the raw PC such as normals, curvature, planarity, and others; this also reduces noise and redundancy, but also improves the subsequent neighborhood construction and feature extraction phases. Following this, local geometric features are extracted and expanded using EdgeConv layers, which aggregate information from neighboring points while capturing directional edge relationships. These enriched features are then passed to a hierarchy module composed of GAT layers. These layers apply attention mechanisms over the constructed graph to selectively focus on the most relevant neighbors during feature propagation. This hierarchical refinement boosts the model ability to generalize across complex geometries.

The remainder of this paper is divided into the following sections: Section II discusses related work in point cloud segmentation and graph-based methods. Section III presents an overview of graph-based architectures used in this study. Section IV describes our proposed EdgeGAT model in detail. Section V outlines the experimental setup and presents the obtained results. Finally, Section VI offers conclusions and directions for future work.

2. Previous Work

In this section, we review the theoretical foundations and recent advances related to graph-based methods for PC segmentation. We first introduce fundamental graph concepts and their applicability to irregularly structured data. Subsequently, we discuss several prominent deep-learning architectures specifically designed for graph-structured data, including both classic and state-of-the-art approaches.

2.1. Graphs

Graphs are versatile and powerful structures that enable the modeling of complex relationships between data in various applications [17]. Each graph consists of vertices and edges that funge as connections between them. In the context of point cloud processing, graphs offer an efficient way to preserve the topology details of the dataset. Translating a point cloud into a graph enables capturing local relationships between points. For this, graphs are a good option to represent irregular grids data. Thus, rather than 2D pixels, 3D PC can present a complex structure, specially when leaf and stem on a

plant-based PC show. With this in mind, we can construct a graph from our PC, given that a graph is represented as $G = (V, E)$ where $|V|$ denotes the set of vertices and E the set of edges. Therefore, we take as a vertex each point of the sub-sampled PC. With each point on the PC acting as a vertex, the final step is to create an edge for each point using the Euclidean distance between points. This can be performed using the K-nearest neighbor method, which connects two vertices p, q by an edge if the distance is the k-th smallest distance from p to any other point from PC. These edges can be represented with an adjacency matrix, which indicates the presence or absence of an edge between a pair of vertices.

2.2. Graph-Based Architectures

In order to address the segmentation task on 3D point clouds, we consider six different deep learning architectures based on graph or point representations: (1) Graph Attention Network [16], (2) Graph Convolutional Network (GCN) [18], (3) PointNet [12], (4) Dynamic Graph CNN (DGCNN) [15], (5) GCN-UNet [19], and (6) our proposed model, EdgeGAT. These models are selected for their strong performance and conceptual relevance in processing unordered point clouds or graphs. In the following subsections, we briefly describe each model, highlighting their key components and main mathematical formulation.

2.2.1. Graph Attention Networks

Graph Attention Networks introduce the attention mechanism to graph-structured data, allowing each vertex to weigh the importance of its neighbors during the message-passing process. This enables the model to dynamically adjust the influence of different neighbors, which is particularly useful in irregular and sparse structures like point clouds. The main operation in GAT can be described as:

$$\mathbf{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} \mathbf{W}\mathbf{h}_j\right),$$

where: $\mathbf{h}_i$ is the input feature vector of vertex i, and $\mathbf{h}_j$ is vector of neighbor j, $\mathbf{W}$ is a shared weight matrix, $\mathbf{h}'_\mathbf{i}$ is the new node layer. α_{ij} is the attention coefficient between vertices i and j, calculated as:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_j]\right)\right)}{\sum_{k \in \mathcal{N}(i)} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_k]\right)\right)}$$

2.2.2. Graph Convolutional Networks (GCN)

GCN generalizes the concept of convolution to non-Euclidean data by aggregating information from neighboring vertices. In its basic form, each vertex updates its representation by averaging (or summing) the features of its neighbors, followed by two operations: a linear transformation and an activation function operation. The propagation rule for a GCN layer is:

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right),$$

Where: $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix with added self-loops, $\tilde{\mathbf{D}}$ is the degree matrix of $\tilde{\mathbf{A}}$, $\mathbf{W}^{(l)}$ is the trainable weight matrix for layer l, and σ is a nonlinear activation function (e.g., ReLU).

2.2.3. PointNet

PointNet was one of the first architectures designed to directly process raw 3D point clouds without converting them into voxel grids or meshes. It treats each point independently using shared MLPs, followed by a global symmetric aggregation function (max pooling) to ensure permutation invariance.

$$f(\{x_1, ..., x_n\}) \approx g\left(\underset{i=1,...,n}{\mathrm{MAX}}\left(h(x_i)\right)\right),$$

where: n is the number of points in the PC, h is a point-wise feature extractor (MLP), MAX is a symmetric function that aggregates features across all points, and g is a final MLP for classification or segmentation.

2.2.4. Dynamic Graph CNN (DGCNN)

DGCNN extends GCNs by dynamically updating the graph structure at each layer based on feature space distances rather than fixed spatial proximity. This allows the model to adapt its neighborhood structure as feature representations evolve. Its core operation is the **EdgeConv**, defined as:

$$\mathbf{h}_i^{(l+1)} = \underset{j\in\mathcal{N}(i)}{\mathrm{MAX}}\ \phi\left(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)} - \mathbf{h}_i^{(l)}\right),$$

where: ϕ is a learnable function (usually an MLP), $\mathcal{N}(i)$ denotes the k-nearest neighbors of point i, and edge features are dynamically recomputed at each layer.

2.2.5. GCN-UNet

GCN-UNet follows a U-shaped architecture inspired by UNet in CNNs, integrating pooling and unpooling operations adapted to graph data. It combines local context preservation with hierarchical abstraction, making it suitable for segmentation tasks where spatial consistency is important.

Its downsampling uses graph pooling techniques such as top k pooling, while upsampling is done through interpolation or learned upsampling methods over the graph structure. The main idea behind GCN-UNet is to hierarchically encode and decode graph features:

$$\begin{aligned} \text{Encoder:} \quad & \mathbf{H}^{(l)} \xrightarrow{\text{Pool}} \text{Pool}\left(\mathbf{H}^{(l)}\right) \xrightarrow{\text{GCN}} \mathbf{H}^{(l+1)}, \\ \text{Decoder:} \quad & \mathbf{H}^{(l)} \xrightarrow{\text{Unpool}} \text{Unpool}\left(\mathbf{H}^{(l)}\right) \xrightarrow{\text{GCN}} \mathbf{H}^{(l+1)}. \end{aligned}$$

2.2.6. EdgeGAT

Our proposed model, EdgeGAT, is designed to leverage both the edge-level geometric relationships from EdgeConv and the adaptive feature weighting of attention mechanisms from GAT. Inspired by DGCNN [15], we first construct a dynamic graph based on k nearest neighbors in the feature space. Then, we apply an attention-based convolution that operates over vertex features. The EdgeConv operation captures local geometric context through the function:

$$\mathbf{e}_{ij} = h_\theta(\mathbf{h}_i, \mathbf{h}_j - \mathbf{h}_i),$$

where: - $\mathbf{h}_i$ and $\mathbf{h}_j$ are the features of vertices i and j, - h_θ is a learnable MLP. We integrate attention weights α_{ij} into this formulation as:

$$\mathbf{h}'_i = \sum_{j \in \mathcal{N}(i)} \alpha_{ij} \cdot \mathbf{e}_{ij},$$

This allows the model to both learn geometric structure (via edge features) and adaptively weight neighbors (via attention), enhancing the expressive power for point cloud segmentation tasks.

3. Proposed graph-based EdgeGAT model

We propose a novel architecture, called EdgeGAT, that integrates enhanced point features with graph-based attention mechanisms to classify

points in PC datasets based on previously labeled point cloud data. This architecture is designed around the following concepts:

1. A point feature extraction and generation,
2. A graph extraction for KNN,
3. EdgeConv layers for vertex representation,
4. Graph Attention layers for global and local context,
5. An explanation on the use of a residual MLP.

Finally, we have a section to describe the training of this entire architecture.

3.1. Point Feature Extraction

Raw point cloud data, initially represented by only the 3D spatial coordinates (x, y, z), lacks sufficient geometric context for complex semantic tasks. To enrich this representation, we incorporate a series of local geometric descriptors derived from Principal Component Analysis (PCA) computed on each point's local neighborhood [20, 21, 22, 23].

The final input vector for each point expands from 3 to 13 dimensions, incorporating the following features:

- Coordinates: The original input with 3 channels (x, y, z)
- Normals:

$$\mathbf{C} = \frac{1}{k} \sum_{i=1}^{k} (\mathbf{p}_i - \bar{\mathbf{p}})(\mathbf{p}_i - \bar{\mathbf{p}})^T,$$

  where $\mathbf{p}_i$ denotes the i-th neighboring point and $\bar{\mathbf{p}}$ is the centroid of the neighborhood. The eigenvector corresponding to the smallest eigenvalue λ_0 of $\mathbf{C}$ defines the estimated surface normal at point $\mathbf{p}$ [24].

- Curvature: A measure of local surface variation, defined as:

$$\text{Curvature} = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2},$$

  where $\lambda_0 \leq \lambda_1 \leq \lambda_2$ are the eigenvalues of the covariance matrix.

- Linearity: Indicates how well points align along a line in the neighborhood. Defined as:

$$\text{Linearity} = \frac{\lambda_2 - \lambda_1}{\lambda_2},$$

  a value close to 1 indicates strong alignment along one direction (e.g., stems), while lower values suggest more isotropic or planar regions.

- Planarity: Measures how well the local neighborhood fits a plane. Defined as:

$$\text{Planarity} = \frac{\lambda_1 - \lambda_0}{\lambda_2},$$

  high planarity values suggest that the points are spread over a 2D surface, such as plants components.

- Scaterring: Reflects how dispersed the local neighborhood is in 3D space. It is computed as:

$$\text{Scattering} = \frac{\lambda_0}{\lambda_2},$$

  a low value indicates structured data (e.g., lines or planes), while a high value suggests volumetric or noisy regions.

- Omnivariance: A measure of volumetric dispersion in 3D space, calculated as the geometric mean of the eigenvalues:

$$\text{Omnivariance} = (\lambda_0 \cdot \lambda_1 \cdot \lambda_2)^{1/3},$$

  higher values represent more volumetric or isotropic neighborhoods.

- Anisotropy: Measures the degree of directionality in the local neighborhood. Computed as:

$$\text{Anisotropy} = \frac{\lambda_2 - \lambda_0}{\lambda_2},$$

  higher values indicate strong alignment along a principal direction, such as stems or elongated structures.

- Eigenentropy: Describes the entropy of the local eigenvalue distribution, capturing the complexity or randomness of point dispersion. Defined as:

$$\text{Eigenentropy} = -\sum_{i=0}^{2} \tilde{\lambda}_i \log(\tilde{\lambda}_i) \quad \text{where} \quad \tilde{\lambda}_i = \frac{\lambda_i}{\lambda_0 + \lambda_1 + \lambda_2},$$

  where higher values suggest more uniform distribution of variance, while lower values imply strong directional structure.

Here, $\lambda_0 \leq \lambda_1 \leq \lambda_2$ are the eigenvalues obtained by performing PCA over the covariance matrix of the point local neighborhood. These values describe the spatial distribution and variation of neighboring points, offering geometric intuition such as flatness (planarity), elongation (linearity), or compactness (scattering) [25].

This 13-dimensional feature vector enables the model to distinguish subtle structural differences between different plant components, such as stems, leaves, and surrounding soil. The inclusion of curvature and entropy, for instance, helps in capturing surface smoothness and complexity, which are useful for clasification in semantic segmentation tasks like in plants datasets [23, 20] .

3.2. Edge and graph construction

Following the feature extraction step, each point in PC is now represented by a 13-dimentional feature vector, encompassing not only the original spatial coordinates (x, y, z) but also a set of geometric additional features derived from PCA feature generation. Therefore, in order to capture local geometric relationships between points, we construct a graph using the KNN algorithm in the spatial features.

The graph construction process considers only the 3D spatial position of each point, denoted as $\mathbf{x}_i \in \mathbb{R}^3$, for neighbor identification. In order to construct the edges of the graph, we use the Euclidean distance between two points defined as:

$$d(\mathbf{p}_i, \mathbf{p}_j) = \|\mathbf{p}_i - \mathbf{p}_j\|_2 , \tag{1}$$

where $\|\cdot\|_2$ is the ℓ_2-norm. For each point $\mathbf{p}_i$, its k-nearest neighbors are determined by

$$\mathcal{N}_k(\mathbf{p}_i) = \left\{ \mathbf{p}_j \;\middle|\; j \in \arg\min_{j \neq i} d(\mathbf{p}_i, \mathbf{p}_j),\ |\mathcal{N}_k| = k \right\}. \tag{2}$$

Once the neighborhood is identified, we construct an undirected edge between $\mathbf{p}_i$ and each of its neighbors $\mathbf{p}_j \in \mathcal{N}_k(\mathbf{x}_i)$. The graph is encoded via an edge index matrix $\mathbf{E} \in \mathbb{R}^{2\times|E|}$, where each column defines a pair of connected vertex indices. While the edge structure is based on spatial proximity, the 13-dimensional features associated with each vertex are described as $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N\} \subset \mathbb{R}^{13}$, and serve as the vertex attributes used in downstream processing stages such as EdgeConv and Graph Attention layers. This method is well-established for pattern recognition and graph construction [26] and is performed by other applications such as Pointnet++ [27] which leverage KNN to model local geometric relationships in point clouds.

3.3. Residual MLP for Enhanced Feature Propagation

In our architecture, rather than using a conventional Multi-Layer Perceptron in the EdgeConv blocks, we employ a **Residual MLP**. A standard MLP performs a sequence of linear transformations followed by non-linear activations. For instance, a two-layer MLP can be expressed as:

$$\mathbf{y} = \sigma\big(\mathbf{W}_2\,\sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2\big), \tag{3}$$

where $\mathbf{x}$ is the input vector, $\mathbf{W}_1, \mathbf{W}_2$ are learnable weight matrices, $\mathbf{b}_1, \mathbf{b}_2$ are biases, and $\sigma(\cdot)$ denotes a non-linear activation function, specifically Leaky ReLU in our implementation.

While standard MLPs perform adequately in shallow configurations, they can suffer from vanishing gradients and degraded performance in deeper networks due to the loss of original input information through successive transformations. To address these limitations, we adopt the **Residual MLP**, inspired by the residual learning paradigm introduced by He et al. [28]. This approach employs a skip (residual) connection that directly propagates the input signal to the output, facilitating improved gradient flow and better feature preservation. Residual learning has demonstrated superior performance in various deep graph neural network architectures [29, 30, 31]. Formally, given an input feature vector $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$, the Residual MLP computes:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}) + \mathcal{S}(\mathbf{x}), \tag{4}$$

where $\mathcal{F}(\mathbf{x})$ denotes the sequence of linear transformations, batch normalization, Leaky ReLU activations, and dropout layers. The skip connection, $\mathcal{S}(\mathbf{x})$, is either a linear projection $\mathbf{W}_s \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$ if input and output dimensions differ ($d_{\text{in}} \neq d_{\text{out}}$), otherwise, in the case of the same input and output dimensions ($d_{\text{in}} = d_{\text{out}}$), it returns the initial vector.

In the context of EdgeConv implementation, described subsequently in Section 3.4, the Residual MLP consists specifically of two hidden layers, expanding input feature dimensions sequentially. This facilitates progressive feature refinement while ensuring stable training through residual connections.

3.4. EdgeConv Integration for Graph vertex Features

Building upon the Residual MLP previously described, our proposed architecture integrates local geometric information into vertex representations using the **EdgeConv** operator, initially introduced by Wang et al. [32] for point cloud analysis. EdgeConv enriches vertex embeddings by explicitly capturing both absolute and relative geometric features from each vertex and its neighbors, as illustrated in Figure 2.

Formally, given a vertex v_i with an initial enriched feature vector $\mathbf{f}_i \in \mathbb{R}^{13}$, EdgeConv computes edge features for each neighbor edge $(i, j) \in E$ as follows:

$$\mathbf{e}_{ij} = \Phi\Big([\mathbf{f}_i \parallel \mathbf{f}_j - \mathbf{f}_i]\Big), \tag{5}$$

where $\Phi(\cdot)$ is the Residual MLP described previously, and $\parallel$ denotes concatenation.

Thus, we apply two consecutive EdgeConv blocks to progressively refine vertex embeddings:

- **First EdgeConv block:** Initially, the relative neighbor differences $\mathbf{f}_j - \mathbf{f}_i \in \mathbb{R}^{13}$ are computed, capturing local geometric context around each vertex. These differences are concatenated within the vertex original features $\mathbf{f}_i$, producing an enhanced 26-dimensional edge vector ($\mathbf{x}_{ij} = [\mathbf{f}_i \| \mathbf{f}_j - \mathbf{f}_i] \in \mathbb{R}^{26}$). This vector is subsequently processed by the ResidualMLP, expanding dimensions from 26 to 32, and then from 32 to 64 dimensions through a simple MLP. After applying sum aggregation over neighboring edges, the first refined vertex embedding $\mathbf{h}_i^{(1)} \in \mathbb{R}^{64}$ is obtained.

- **Second EdgeConv block:** The refined vertex embeddings $\mathbf{h}_i^{(1)} \in \mathbb{R}^{64}$ from the previous block serve as input. Following the same methodology, relative differences are computed as $\mathbf{h}_j^{(1)} - \mathbf{h}_i^{(1)}$, producing vectors of 64 dimensions. Concatenation of these relative differences with the

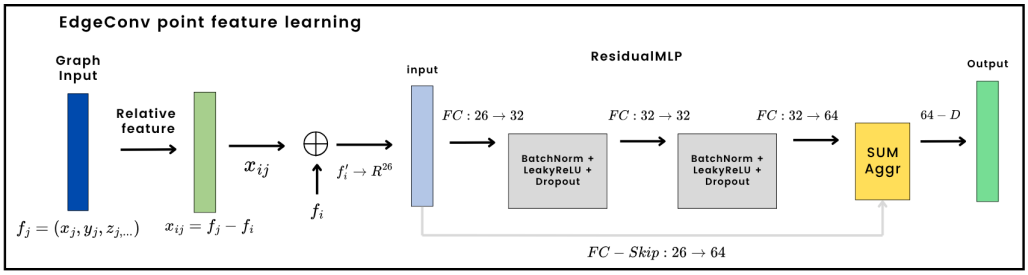

Figure 2: Architecture of the EdgeConv module for feature enhancement. The Residual MLP plays a key role in refining features after concatenation with the original input, resulting in an enhanced feature representation.

vertices own features ($\mathbf{h}_i^{(1)}$) yields 128-dimensional edge vectors, which are subsequently processed by a second Residual MLP, this time reducing dimensionality from 128 to 32 and then expanding back to 64. After mean aggregation over neighbors, we obtain the second refined embeddings, $\mathbf{h}_i^{(2)} \in \mathbb{R}^{64}$.

Finally, to preserve the initial geometric information and facilitate feature reuse, we concatenate the original input vector $\mathbf{f}_i$ with the output of the second EdgeConv $\mathbf{h}_i^{(2)}$, yielding a final vertex representation:

$$\mathbf{h}_i^{\text{final}} = [\mathbf{f}_i \parallel \mathbf{h}_i^{(2)}] \in \mathbb{R}^{77}. \tag{6}$$

This enriched embedding captures both low-level geometric structure and high-level edge-based contextual information. These representations then serve as input for subsequent graph-attention layers, as detailed in the following section.

3.5. Graph Attention layers

Following the EdgeConv stages, each point in the graph is represented by a feature vector of dimension 77. This dimensionality arises from the concatenation of the original 13 input features with the outputs of the two successive EdgeConv blocks, 26 and 64 dimensions respectively, with partial overlaps due to shortcut connections. These enriched representations serve as input to our GAT, which is designed to further refine vertex features by capturing both local and global dependencies through attention mechanisms [16, 33].

In each GAT layer, a shared linear transformation $\mathbf{W}$ is applied to every vertex feature vector $\mathbf{h}_i \in \mathbb{R}^{77}$, yielding a transformed feature space. To model the importance of neighboring vertices, the attention coefficient between vertex i and its neighbor j is computed as:

$$\alpha_{ij} = \text{softmax}_j \left(\sigma \left(\mathbf{a}^\top \left[\mathbf{W}\mathbf{h}_i \parallel \mathbf{W}\mathbf{h}_j \right] \right) \right), \tag{7}$$

where $\mathbf{a}$ is a learnable weight vector and σ is a Leaky ReLU activation function. These coefficients quantify the attention that vertex i assigns to each of its neighbors.

The vertex feature update is then performed through an attention-weighted aggregation of neighbor features:

$$\mathbf{h}'_i = \sum_{j \in \mathcal{N}(i)} \alpha_{ij} \, \mathbf{W}\mathbf{h}_j. \tag{8}$$

This attention mechanism enables the model to selectively focus on informative neighbors while down-weighting less relevant ones. In our implementation, two stacked GAT layers are used, allowing the network to progressively capture richer features at different levels of neighborhood abstraction.

3.6. Training with Graph Attention Networks

After enriching the point features to a 77-dimensional representation with Residual MLPs and EdgeConv, we employ GAT to perform vertex classification on the constructed graph $G = (V, E)$. Each vertex in the graph represents a point in the point cloud and is associated with a 77-dimensional feature vector.

Unlike traditional GCNs, which use uniform aggregation of neighboring features, GAT layers incorporate a learnable attention mechanism to assign different weights to neighboring vertices. This allows the model to emphasize relevant context while maintaining spatial adaptability. As introduced by Veličković et al. [16], each attention head independently processes the neighborhood of a vertex, capturing diverse relational patterns.

The first GAT layer receives an input of shape $N \times 77$, where N is the number of vertices, and produces a higher-dimensional representation using four parallel attention heads, each with 64 hidden units. The outputs of these heads are concatenated, resulting in a representation of shape $N \times 256$:

$$\mathbf{H}_1 = \text{GAT}_{\text{concat}} \left(\mathbf{X}, \mathbf{E} \right) \in \mathbb{R}^{N \times 256}$$

where $\mathbf{X} \in \mathbb{R}^{N \times 77}$ is the input feature matrix and $\mathbf{E}$ the set of edges, constructed via a fixed k-nearest neighbor strategy with $k = 16$.

A second GAT layer then processes this intermediate representation. Again using four attention heads, but this time averaging their outputs, the resulting representation has shape $N \times 64$:

$$\mathbf{H}_2 = \mathrm{GAT}_{\mathrm{avg}}\left(\mathbf{H}_1, \mathbf{E}\right) \in \mathbb{R}^{N \times 64}$$

A final fully connected layer projects this output into the classification space of three classes:

$$\mathbf{Y} = \mathrm{Linear}(\mathbf{H}_2) \in \mathbb{R}^{N \times 3}$$

During training, the model minimizes a cross-entropy loss over all vertices. Optimization is performed using the Adam algorithm [34] with a learning rate of 0.001, for 100 epochs, and a batch size of 8. A fixed graph structure with $k = 16$ neighbors is used throughout, though the architecture allows for extension to dynamic graph constructions based on spatial radius.

This attention-based architecture effectively leverages the rich 77-dimensional vertex features and the expressiveness of multi-head attention to capture both fine-grained and global contexts in complex plant structures.

4. Experiments

The objective of these experiments is to classify each vertex with our model into one of three predefined categories: soil, stem, or leaf. For this, each step carried out with our dataset is described below.

4.1. Data

The data was obtained from the work of the authors of [35], who allowed us to use the point clouds they had collected on corn. We refer to this dataset as Ao dataset in reference to its first author. These point clouds were acquired using terrestrial LiDAR. We also used a second dataset, this was obtained from Pheno4D [36] which had labeled point clouds of corn and tomato plants, of which we focused on the corn dataset, we kept the original Pheno4D name for analysis.

Each individual corn plant file originally contained over 50,000 points, a number too large to be directly processed. To address this, we applied

a data augmentation step consisting of random jittering, rotation, translation, and scaling to enhance data variability and model robustness. After augmentation, uniform random downsampling was performed to reduce the number of points to 1,024 prior to training in both datasets. This sampling was done using the `Open3D` library, which selects a random subset of point indices without replacement.

Mathematically, this corresponds to selecting a subset $S \subset P$ such that $|S| = 1024$, where P denotes the augmented set of points, and the selection is governed by a uniform probability distribution over the indices. For the PCA-based feature extraction process, we generated various geometric descriptors discussed in section 3.1. Throughout this document, these descriptors are denoted by specific acronyms for convenience, as detailed in Table 1.

Table 1: Acronyms used for PCA-derived geometric feature sets

Acronym	**Feature Combination**
XYZ	Raw 3D coordinates
N	Surface normals
C	Curvature
L	Linearity
P	Planarity
S	Scattering
O	Omnivariance
A	Anisotropy
E	Eigenentropy

4.2. Data preprocessing

The points used for classification are manually selected and visualized using MeshLab an Open-Source Mesh Processing Tool [37]. Once the dataset is fully prepared, the model can be trained to perform these classifications with our model. The distribution of the three classes is given by Table 2.

Due to limited data access, the maize plants in the Ao dataset had an overall age of approximately 30 days and presented plant heights ranging from 80 cm to 130 cm. In contrast, the Pheno4D dataset contained younger maize plants aged between 4 and 10 days, with plant heights typically ranging from

Table 2: Table 1: Distribution of classes in Ao maize dataset (%)

	Leaves (%)	**Soil (%)**	**Stem (%)**
Maize train model	50.15	29.95	19.91
Maize test model	53.36	28.34	18.30
Maize valid model	53.82	24.42	21.77

10 cm to 30 cm. Both datasets exhibit significant structural variations, especially in leaf and stem configurations. Given the early developmental stage of plants in Pheno4D, the maize spike was grouped with the stem class, as it was not fully developed. Additionally, the upper and lower plant segments were combined into the stem category, simplifying classification. For clear visualization of the ground truth labels, distinct colors were assigned to represent each class: green for leaves, red for soil, and blue for stem. Examples of labeled maize plants from the Ao dataset are presented in Figure 3, while Figure 4 illustrates labeling results for the Pheno4D dataset.

The classified datasets were evaluated using a five-fold cross-validation strategy. Specifically, each dataset was partitioned into five equally sized subsets. In each iteration, four folds, that contain 80% of the data, are used to train the model, while the remaining fold is employed as a validation set for hyperparameter tuning and to detect potential overfitting. This procedure was repeated five times, ensuring each fold served exactly once as the validation set. The overall performance metrics were then calculated by averaging the results across all folds, thus providing a more robust model.

4.3. Ablation study on feature enhancement with PCA-derived attributes

Given the features obtained by PCA, we conducted a series of experiments, evaluating the impact of incorporating these features derived from PCA geometric descriptors. These experiments are performed principally in Ao dataset, and across four architectures: GCN, GAT, UNET, and UNET2. Where UNET and UNET2 are architectures based on the GCN following an encoder-decoder structure best described in Appendix A. Also, we employ a fixed graph construction strategy with $k = 16$.

For this, we start with the simplest feature configuration using only the spatial coordinates, and progressively include other descriptors such as: normals, curvature and normals and finally all our 10 PCA features. While the

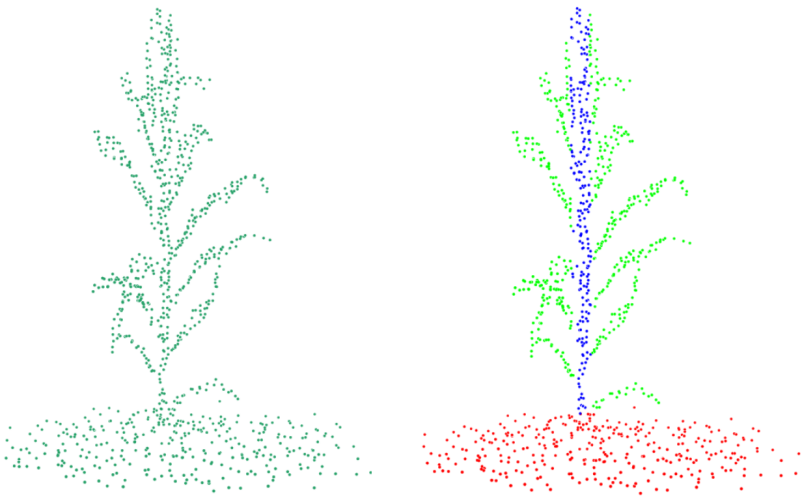

Figure 3: Three-class point cloud representation of a maize plant in Ao dataset.(a) The raw point cloud data, (b) The classified point cloud, where the stem is labeled in blue, the leaves in green, and the soil in red.

inclusion of curvature or normals individually led to only marginal improvements in validation mIoU. A more substantial boost is observed in Figure 5 when combining both curvature and normal features. This is the reason behind the exploration of a richer 13-dimensional feature set derived from PCA.

Figure 5(a) presents the baseline performance using only the raw XYZ coordinates. Under this scenario, all four models exhibit comparable mIoU results after 60 epochs, with GAT and GCN showing similar performance, and both U-Net-based models converging below the 60% threshold.

Figure 5(b) adding surface-normal vectors to the XYZ coordinates, a notable performance differentiation emerges among the models. Here, the GAT model achieves the highest performance, followed closely by GCN. The two U-Net models display improvements as well, clustering their performances around the 65% mark. Nevertheless, GAT consistently outperforms all other models in this configuration.

Expanding further on this analysis, Figure 6(a) presents additional experiments. This figure includes curvature, alongside XYZ and normals, showing a clear positive trend across all models. The GCN model achieves the highest

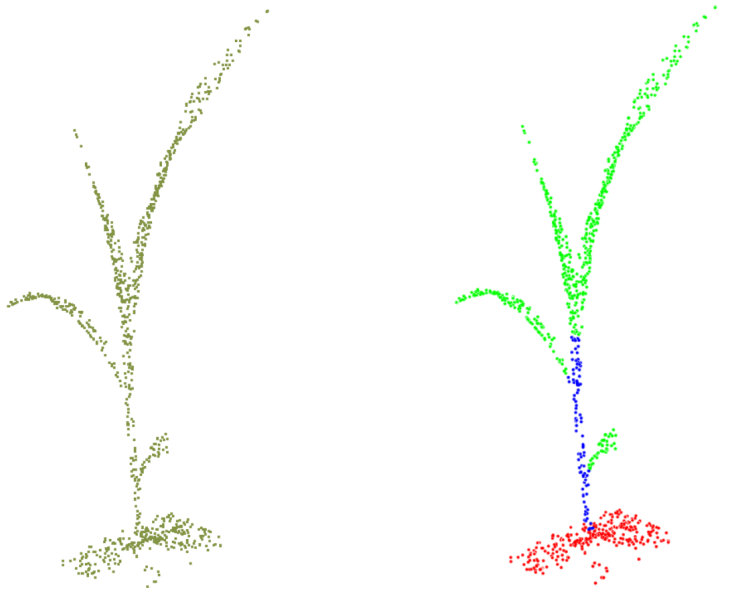

Figure 4: Three-class point cloud representation of a maize plant used for model training in Pheno4D dataset. (a) The raw point cloud data, (b) The classified point cloud, where the stem is labeled in blue, the leaves in green, and the soil in red.

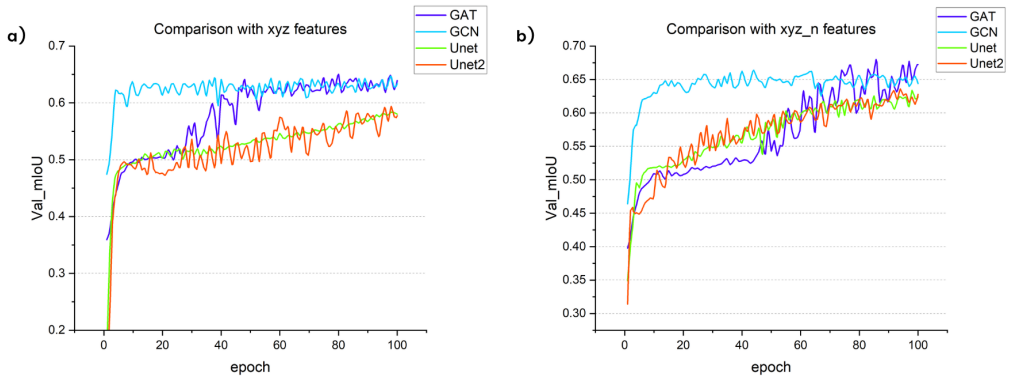

Figure 5: Validation mIoU results across different feature configurations for GCN, GAT, UNET, and UNET2 on Ao dataset. Each subplot shows the evolution of validation performance when including: (a) only xyz, (b) xyz + normals.

mIoU at approximately 67%, followed closely by GAT, then U-Net2 and U-Net in that order. This demonstrates a beneficial impact from incorporating curvature as an additional feature, raising overall performance from roughly 60% to 67%.

Figure 6 (b) evaluates the impact of using an expanded feature set comprising XYZ, normals, curvature, and the complete set of 10 PCA-derived features. Here, we observe further performance improvement, with GCN narrowly surpassing GAT, reaching a value slightly above 70% after epoch 35. The GAT model achieves a similar level, but only after epoch 90. The two U-Net variants lag behind, remaining consistently below the 70% mark.

Given the previously observed improvements in model performance when integrating additional geometric features, we conduct an analysis comparing the training times of all evaluated architectures under the different feature configurations. Table 3 summarizes the time per epoch (in seconds) for each model and feature set. As seen, GCN and GAT architectures consistently exhibit significantly faster training speeds compared to the U-Net–based encoder–decoder models.

The results highlight a clear efficiency advantage of GCN and GAT models. Specifically, GCN achieved consistently low training durations, that have approximately 12.0 seconds per epoch across all feature configurations, while GAT maintained similar efficiency with epoch durations of around 11.0 seconds. Conversely, the U-Net based GCN models exhibited notably longer training times, ranging from approximately 28.3 to 31.5 seconds per epoch,

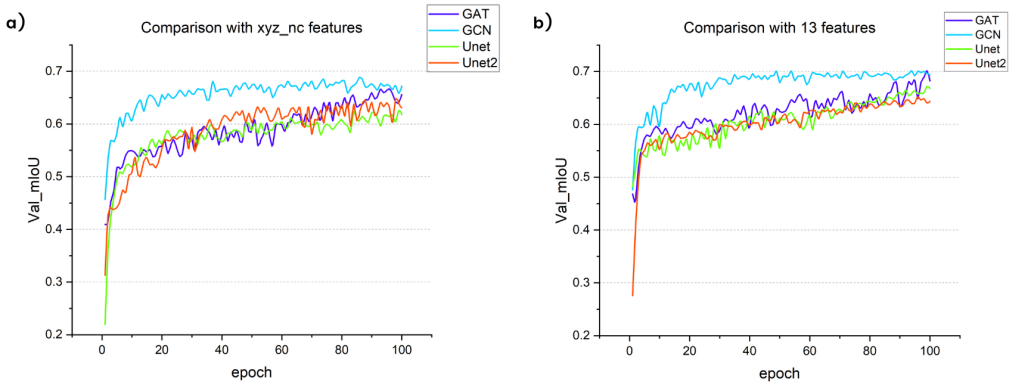

Figure 6: Validation mIoU results across different feature configurations for GCN, GAT, UNET, and UNET2 on Ao dataset. Each subplot shows the evolution of validation performance when including: (a) xyz + normals + curvature, (b) xyz + 10 PCA features.

Table 3: Comparison of Training Times for Different Graph Architectures on Ao dataset

Features	**GCN (s)**	**UNet (s)**	**UNet2 (s)**	**GAT (s)**
XYZ	12.04	31.24	28.97	11.23
XYZ-N	12.08	31.36	30.32	11.09
XYZ-NC	12.14	31.52	28.92	11.21
XYZ-NCLPSOAE	12.21	31.09	28.30	11.14

depending on the specific feature set. This substantial efficiency gap, combined with the competitive or superior performance previously demonstrated, strongly supports the adoption of GNN-based architectures—particularly GCN and GAT—for this type of point cloud segmentation task.

In order to further evaluate the impact of adding 13 PCA-derived features with our model, we extended this comparison to include performance metrics such as validation loss and mIoU, as shown in Table 5.

As shown in Table 5, the GAT model exhibited relatively short training times while maintaining good validation performance, achieving an mIoU of 67.40% and an accuracy of 79.25%. Interestingly, despite having the longest training time of 22.49 seconds per epoch, our proposed EdgeGAT model attained the highest performance metrics among all evaluated architectures, reaching a validation mIoU of 73.44% and an accuracy of 82.28%.

Table 4: Training Time and Performance Metrics Using 13 PCA Features in Ao dataset

Model	Training Time (s)	mIoU (%)	Val. Accuracy (%)
GCN	12.25	67.33	79.20
GAT	**11.70**	67.40	79.25
UNet	25.07	64.07	77.53
UNet2	30.92	65.92	79.50
EdgeGAT	22.49	**73.44**	**82.28**

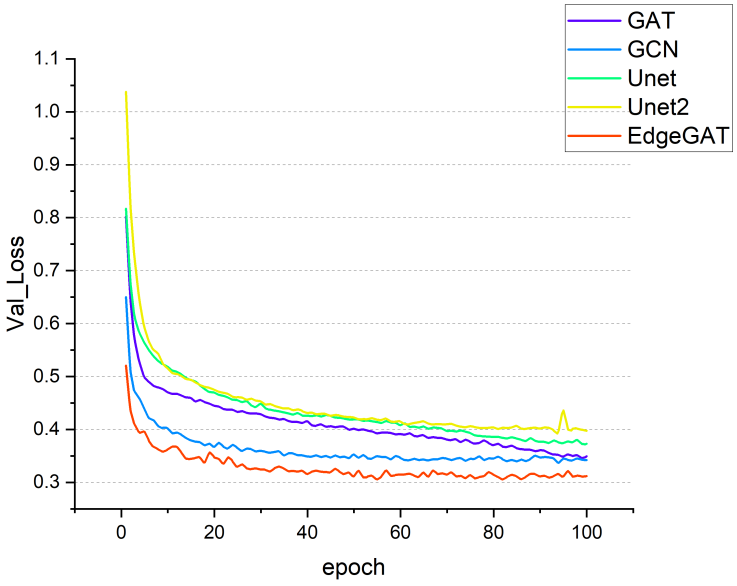

Figure 7: Validation Loss of GCN, GAT, UNet, UNet2, and EdgeGAT using 13 PCA-extracted features in Ao dataset.

The effectiveness of incorporating the complete set of 13 PCA-extracted geometric features is further illustrated in Figures 9 and 8. Figure 9 highlights the impact of these features on the loss function during training, clearly showing that EdgeGAT achieved the fastest initial decline, consistently maintaining the lowest loss values throughout training, followed closely by GCN and GAT. Complementarily, Figure 8 demonstrates a noticeable improvement in validation mIoU across all models when PCA-derived features were

included. Notably, EdgeGAT surpassed the 70% mIoU threshold early on, outperforming traditional models such as GCN and GAT. Moreover, EdgeGAT reached higher mIoU levels as early as epoch 20, further underscoring the effectiveness of combining PCA feature extraction with advanced graph-based learning techniques.

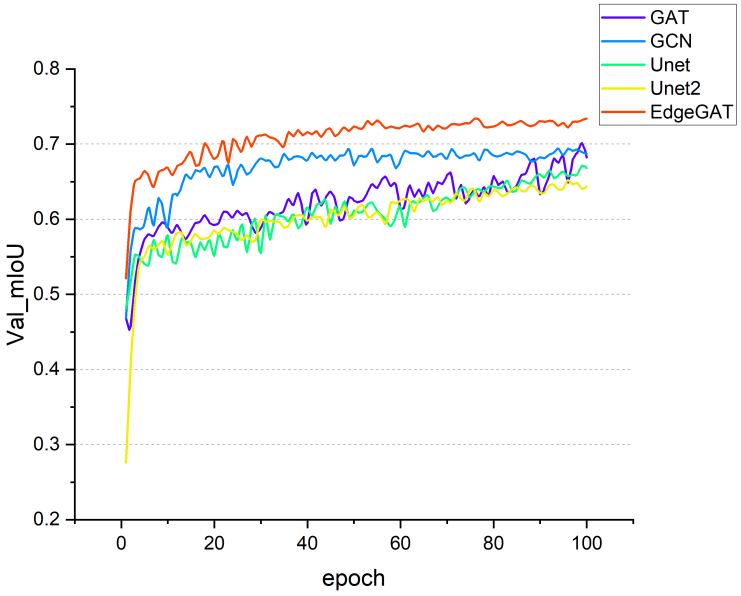

Figure 8: Validation mIoU of GCN, GAT, UNet, UNet2, and EdgeGAT using 13 PCA-extracted features in Ao dataset.

Table 5: Training Time and Performance Metrics with 13 PCA features on Pheno4D dataset

Model	**Training Time (s)**	**mIoU (%)**	**Val. Accuracy (%)**
GCN	33.60	80.28	87.80
GAT	**31.30**	82.12	88.93
UNet	82.50	77.08	85.80
UNet2	80.00	76.21	85.50
EdgeGAT	62.80	**93.20**	**96.40**

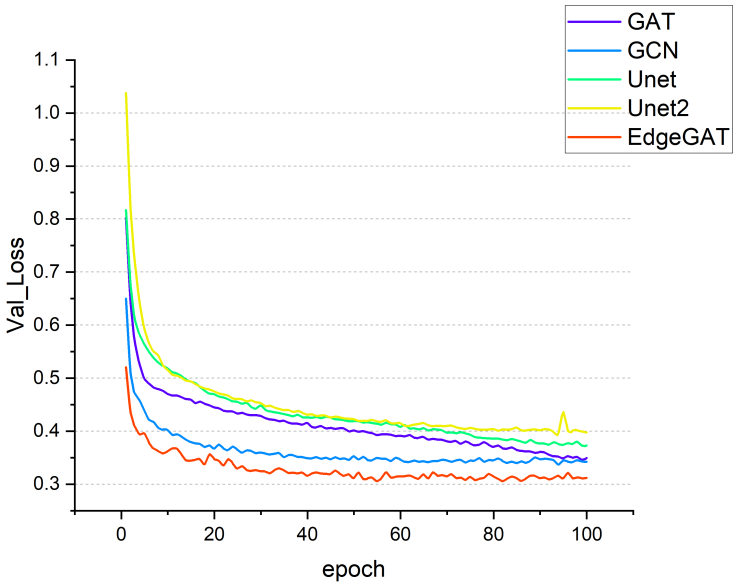

Figure 9: Validation Loss of GCN, GAT, UNet, UNet2, and EdgeGAT using 13 PCA-extracted features.

Additionally, we assessed the models on the Pheno4D dataset using the full set of 13 PCA-derived features. Table 5 summarises the results. Consistent with previous experiments, the GAT model achieved the shortest training time (31.3 s per epoch), closely followed by the GCN baseline, whereas the U-Net variants remained the slowest. Our proposed EdgeGAT required a moderate 62 s per epoch, yet delivered the best segmentation performance, attaining an mIoU of 93.20 %. The next-best model was GAT at 82.12 %, followed by GCN, with both U-Net models trailing behind. Validation accuracy mirrored the mIoU ranking, confirming that the attention-based architectures outperformed their convolution counterparts, with EdgeGAT offering the overall highest accuracy despite its longer training time. Figure 10 visualises the evolution of the validation mIoU for the same models on the Pheno4D dataset. The curve confirms the numerical trends reported in Table 5: **EdgeGAT** rises sharply within the first ten epochs and stabilises above the 90 % mark, surpassing the other graph-based architectures by nearly ten percentage points. The standard GAT model follows with values consistently above 80 %, while the GCN baseline converges just below that level. Both U-Net variants remain well behind, never exceeding the 70 %

threshold. The early and sustained lead of EdgeGAT highlights not only its higher final accuracy but also its faster convergence and greater stability throughout training.

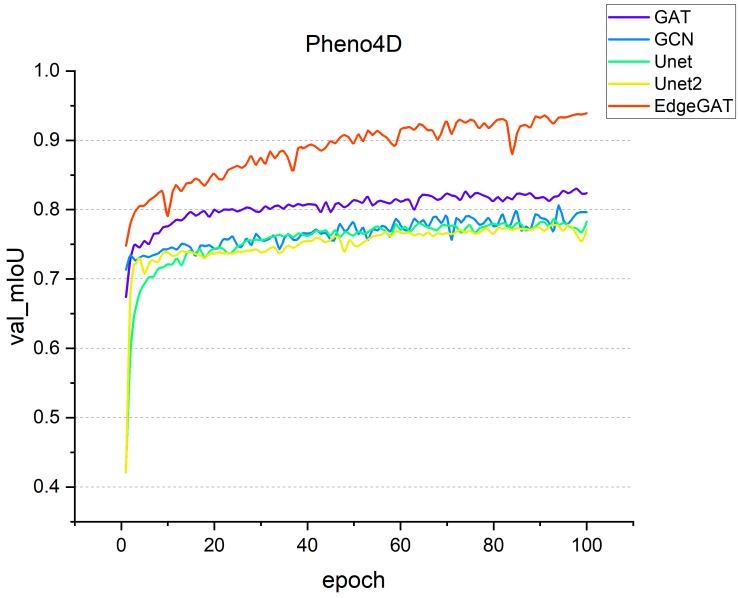

Figure 10: Validation mIoU of GCN, GAT, UNet, UNet2, and EdgeGAT using 13 PCA-extracted features in Pheno4D dataset.

This trade-off between computational efficiency and predictive performance highlights the practical considerations when selecting an architecture. While GCN and GAT offer excellent speed–performance balance, models like EdgeGAT are better suited when maximum mIoU is critical, even at the expense of longer training durations.

4.4. Comparative Analysis and Advantages of EdgeGAT over other Point Cloud Models

In this section, we conduct additional comparative experiments involving both point-based (PointNet and DGCNN) and graph-based (GAT and EdgeGAT) architectures, aiming to evaluate the relative effectiveness of our proposed EdgeGAT model. The analysis specifically focuses on the validation loss and mean Intersection over Union metrics.

Figure 11 illustrates the validation loss across epochs for each architecture. The EdgeGAT model demonstrates superior convergence, achieving the

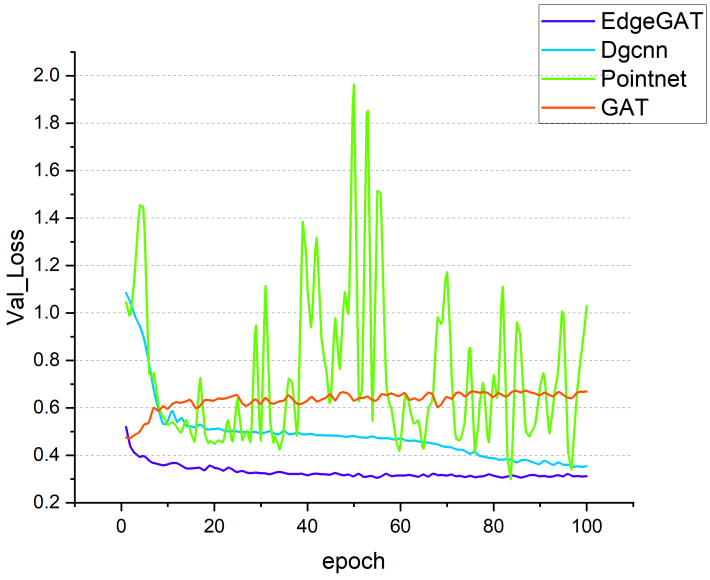

Figure 11: Validation Loss of point and graph-based models using 13-PCA features: EdgeGAt, DGCNN, Pointnet and GAT.

lowest loss values rapidly after approximately epoch 30. In contrast, PointNet exhibits high variability without reaching a stable equilibrium, reflecting its difficulty in consistently modeling the three classes. The DGCNN model shows a steady but slower decrease in validation loss, approaching values similar to EdgeGAT only after epoch 80. Interestingly, GAT experiences signs of potential overfitting or training instability, as its loss initially starts at around 0.50 but gradually increases to approximately 0.60, highlighting potential challenges in its convergence behavior when using the complete set of PCA-derived features.

The validation mIoU for each architecture is presented in Figure 12. Here, our EdgeGAT model achieves the highest and most consistent mIoU performance, surpassing 73%. DGCNN follows, reaching a maximum near 70%, but only towards the final epochs of training. GAT remains consistently below 70%, aligning with its observed loss instability. PointNet occasionally reaches peaks around 70%, yet its performance fluctuates significantly, ultimately averaging around 61%, underscoring limited consistency in capturing class-specific features.

In addition to the results illustrated previously, Table 6 summarizes a

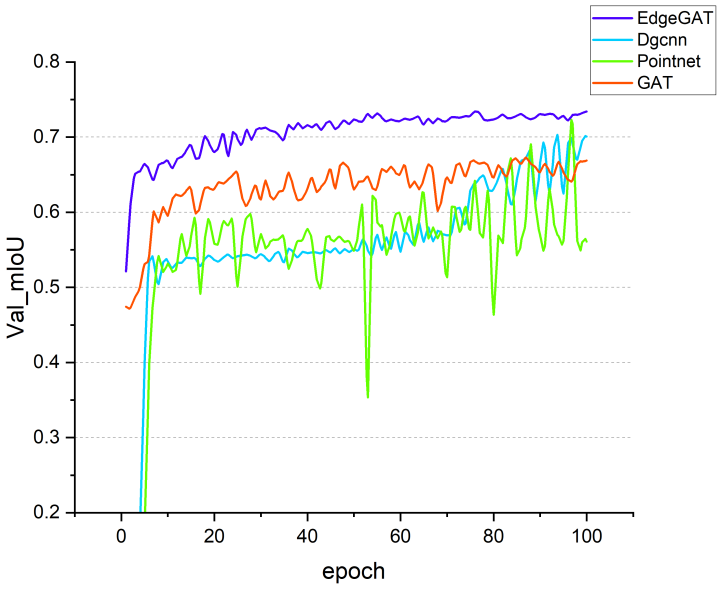

Figure 12: Validation mIoU of point and graph-based models using 13-PCA features: EdgeGAT, DGCNN, Pointnet and GAT.

Table 6: Performance Comparison of Evaluated Architectures in Ao Dataset

Metric	**EdgeGAT**	**DGCNN**	**PointNet**	**GAT**
Loss (%)	**29.28**	31.71	43.13	32.85
mIoU (%)	**73.35**	70.54	60.07	70.31
Accuracy (%)	**85.09**	83.53	77.53	83.05
Precision (%)	82.89	**83.14**	55.82	81.21
T. Time (s)	22.49	189.29	16.20	**11.29**

detailed quantitative comparison across all evaluated architectures on the validation dataset. Specifically, it includes performance metrics such as loss, mean IoU, accuracy, precision, and training time per epoch. While DGCNN achieved the highest precision of 83.14%, it required the longest training duration of approximately 189 seconds per epoch. Conversely, the GAT model recorded the shortest training time at 11.09 seconds per epoch; however, it did not reach the performance obtained by our proposed EdgeGAT

model. Our EdgeGAT architecture, despite having a moderate training time of 22.30 seconds per epoch, outperformed other models by achieving the highest values for mIoU and accuracy, along with the lowest loss values. Lastly, the PointNet model demonstrated a relatively quick training time of 16.20 seconds per epoch but exhibited the lowest performance metrics among all evaluated architectures.

Evaluation on the Pheno4D Dataset. Table 7 summarizes the performance of all evaluated architectures on the more complex and diverse Pheno4D dataset, confirming the robustness and generalizability of our EdgeGAT model. Consistent with earlier results from the Ao dataset, EdgeGAT significantly surpasses the other architectures, achieving a notably high mIoU of 93.20%. This result outperforms the next-best models, GAT and DGCNN, by approximately 11 percentage points, as both models converge to values slightly above 80%. In contrast, PointNet exhibits considerably lower performance, achieving an mIoU of only around 60%, highlighting its limited capacity for consistently capturing complex geometric features present in this dataset. These trends are clearly reflected in Figure 13, where the validation mIoU curves illustrate that EdgeGAT not only reaches superior accuracy levels, but also attains this performance much earlier during training, emphasizing its stability and efficiency in learning rich feature representations from challenging real-world point cloud data.

Table 7: Performance Comparison of Evaluated Architectures in Pheno4D Dataset

Metric	**EdgeGAT**	**DGCNN**	**PointNet**	**GAT**
Loss (%)	**04.09**	12.29	24.96	16.94
mIoU (%)	**93.20**	82.05	60.82	82.12
Accuracy (%)	**96.40**	95.43	86.28	88.93
Precision (%)	**97.20**	89.12	80.25	89.46
T. Time (s)	62.80	452.17	36.00	**31.30**

Collectively, these results emphasize that our proposed EdgeGAT model not only reaches superior mIoU and validation accuracy values, but also converges faster and more reliably compared to both point-based and alternative graph-based approaches when integrating the full set of PCA-extracted features.

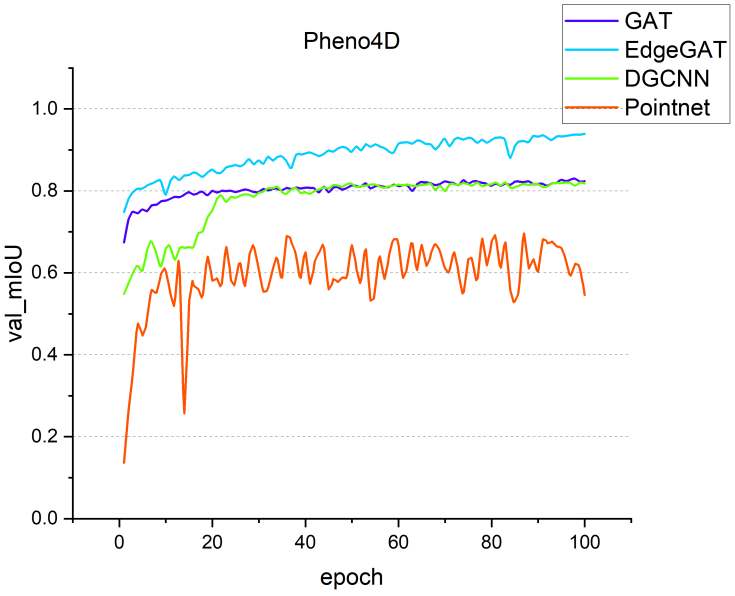

Figure 13: Validation mIoU curves for EdgeGAT, DGCNN, PointNet, and GAT on the Pheno4D dataset.

In order to better visualize the performance of each model in a real-world scenario, we selected a raw 3D point cloud from the test set and evaluated it using the best checkpoint from each model. These results are shown in Figure 14. Part (a) presents the original input point cloud. Part (b) displays the prediction output from DGCNN, while part (c) corresponds to the result obtained with PointNet. Finally, part (d) shows the prediction from our EdgeGAT model. As observed, PointNet only predicted two out of the three trained classes, demonstrating limited expressiveness. DGCNN performed noticeably better, closely matching the output of our model. However, EdgeGAT exhibited more accurate class separation and clearer segmentation boundaries.

5. Discussion

In recent years, methods for point cloud segmentation have significantly evolved, transitioning from traditional convolutional neural network approaches, often hindered by the irregularity and unordered nature of point cloud data, to graph-based methods that explicitly exploit the relational structure inher-

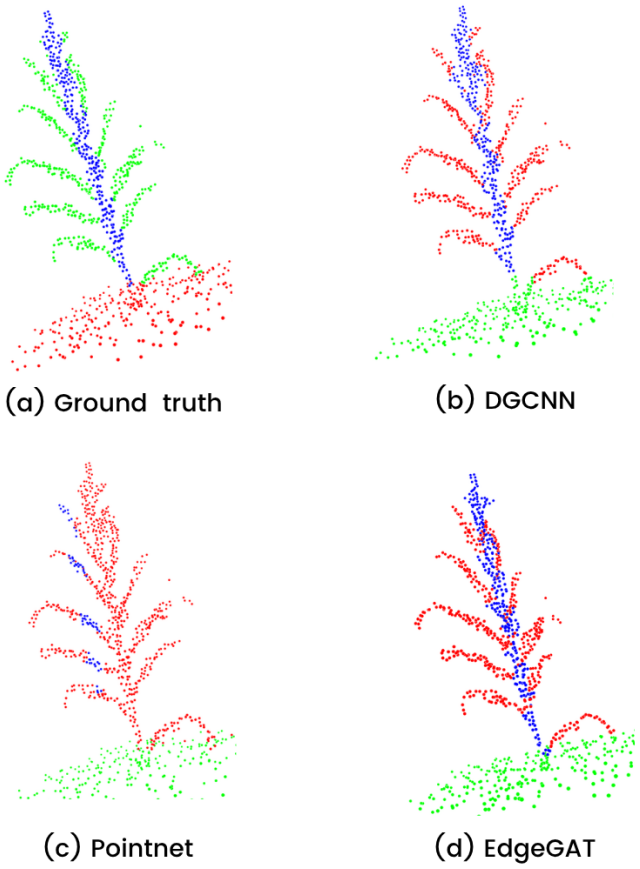

Figure 14: Visualization of point cloud classification: (a) Ground truth of maize plant, (b) Predicted with DGCNN model, (c) Predicted with pointnet model, (d) Predicted with our EdgeGAT model.

ent to 3D information. Graph-based models represent point clouds as structured graphs, treating each point as a vertex interconnected by edges that capture local geometric relationships. By aggregating local context through these edges, these models facilitate precise feature extraction, crucial for effectively segmenting complex structures such as maize plants.

The integration of attention mechanisms within graph-based models further refines segmentation performance by adaptively weighting the contribution of neighboring vertices. Such attention-based modules allow the model to dynamically prioritize relevant local features, addressing challenges related to uneven feature representation and local ambiguities. In this context,

our proposed EdgeGAT model clearly demonstrates these advantages. By combining edge convolutional operations enhanced with Residual MLPs and attention-based mechanisms from Graph Attention Networks, the EdgeGAT architecture achieves superior feature representation, faster convergence, and significantly improved segmentation accuracy compared to both standard GAT and alternative point-based models.

Our extensive evaluations across two different datasets (Ao and Pheno4D) further reinforced these observations. On the Ao dataset, EdgeGAT outperformed alternative architectures, achieving higher and more consistent mIoU and accuracy metrics. Interestingly, evaluations on the Pheno4D dataset, which consists of younger maize plants approximately ten days old, yielded even higher performance metrics due to the reduced structural complexity of the plants at this developmental stage. Specifically, EdgeGAT attained an mIoU exceeding 90%, substantially outperforming DGCNN and GAT, both of which reached approximately 80%. PointNet consistently lagged behind, underscoring its limitations in reliably capturing geometric features even in less structurally complex scenarios. Despite EdgeGAT relatively higher computational demands compared to simpler models, the substantial gains in accuracy, stability, and rapid convergence strongly justify its adoption, particularly when precise segmentation and robust performance across different developmental stages of plants are critical.

6. Conclusion

In this work, we demonstrated the effectiveness of our proposed model, EdgeGAT, for accurate segmentation and classification of maize plant components. By enriching the original 3D coordinates through PCA-based geometric feature extraction, we significantly improved the representation of local geometric details, enabling the construction of more informative graphs and facilitating accurate differentiation among plant structures.

Our proposed EdgeGAT architecture, which integrates EdgeConv modules enhanced with Residual MLPs and Graph Attention layers, proved highly effective in capturing both fine-grained local characteristics and broader contextual relationships within the point clouds. Specifically, by leveraging edge-based convolutions and dynamic neighbor aggregation, EdgeGAT achieved superior performance, exhibiting notably faster convergence and higher segmentation accuracy compared to traditional graph-based methods and other state-of-the-art point cloud architectures.

However, our EdgeGAT model also presents certain limitations. Although it achieves strong accuracy and precision in segmentation tasks, even with limited samples, it may not generalize effectively to other tasks, such as classification of 2D images or datasets in which the axis alignment among classes is less consistent than in structured scenarios like plant segmentation. Future work should therefore explore the model generalizability by evaluating its performance on diverse point cloud datasets involving semantic segmentation in different contexts and applications.

Overall, the progression toward graph-based segmentation models, particularly those leveraging edge-based convolution and attention mechanisms, presents significant potential for improving the accuracy and efficiency of point cloud segmentation. This paradigm is particularly relevant in agricultural applications, where precise identification and segmentation of plant components—such as stems, leaves, and soil regions—are crucial for precision agriculture tasks, including crop monitoring, biomass estimation, and targeted agricultural interventions.

Acknowledgements

The authors wish to express their sincere gratitude to CINVESTAV for their institutional support throughout this project. We also thank SECIHTI for the financial assistance provided, which was instrumental in enabling this research. Additionally, we extend our appreciation to the authors of the original datasets, whose efforts in data collection and sharing made this work possible. The source code developed and labeled datasets for this research is available at: `https://github.com/Ivanrms1/EdgeGAT-graph-model`.

Appendix A. Evaluated Sub-architectures

This appendix provides detailed descriptions of two alternative GCN-based sub-architectures evaluated during preliminary experiments: a Simple GCN model and a Graph U-Net architecture.

A.1 Simple Graph Convolutional Network (Simple GCN)

The Simple GCN model consists of multiple stacked GCN layers (`GCNConv`), each followed by a ReLU activation and dropout regularization, except for the final output layer, which outputs raw logits directly.

Mathematically, a single GCN convolutional layer can be described as follows:

$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right), \tag{A.1}$$

where $\mathbf{H}^{(l)}$ represents vertex embeddings at layer l, $\mathbf{W}^{(l)}$ are learnable weight matrices, $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix of the graph with added self-connections, and $\hat{\mathbf{D}}$ is the diagonal degree matrix of $\hat{\mathbf{A}}$. The activation function $\sigma(\cdot)$ is ReLU in intermediate layers.

The final classification logits $\mathbf{Z}$ for vertices are computed directly from the last GCN layer without activation:

$$\mathbf{Z} = \hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(L-1)}\mathbf{W}^{(L-1)}, \tag{A.2}$$

where L is the total number of layers.

GCN Model: The diagram should depict a clear stack of multiple GCNConv layers, each followed by ReLU and dropout, showing explicitly the skip of activation in the last layer.

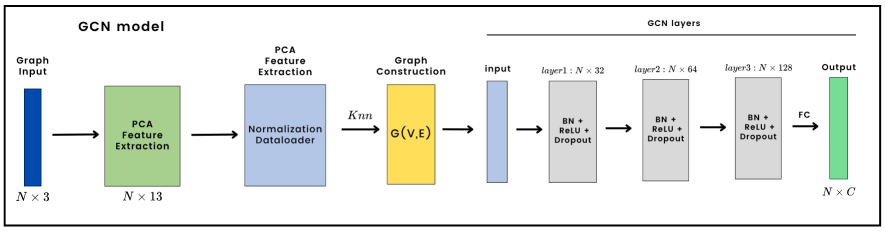

Figure A.15: Visualization of GCN model utilized in ablation studies

A.2 Graph U-Net with GCN (GCN-UNet)

The second architecture is a Graph U-Net, which integrates pooling and unpooling layers with GCN convolutions. It is inspired by the U-Net structure commonly used in computer vision, enabling the network to capture hierarchical features at multiple scales within graph data.

GCN Unet Model: The diagram should depict a clear stack of multiple GCNConv layers, each followed by ReLU and dropout, showing explicitly the skip of activation in the last layer.

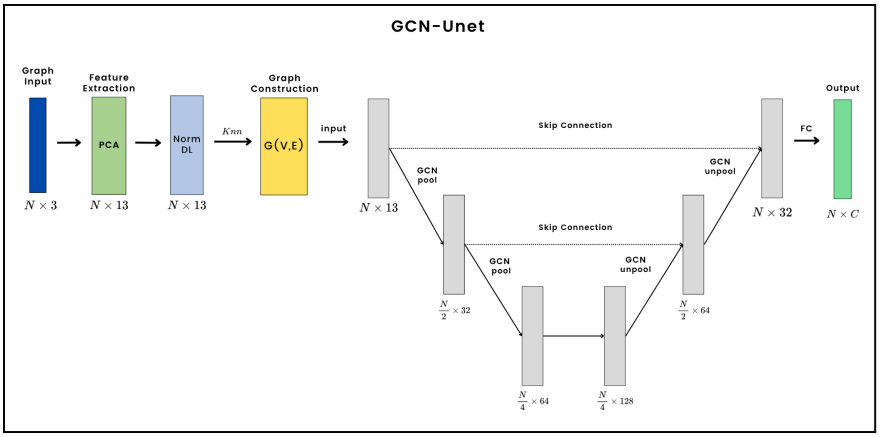

Figure A.16: Visualization of GCN-Unet model utilized in ablation studies

The Graph U-Net operates through three main steps:

1. **Encoder:** Repeated application of GCN layers and graph pooling to reduce the number of vertices progressively.
2. **Bottleneck:** A latent representation is obtained at the deepest level.
3. **Decoder:** Gradual reconstruction of the original graph resolution by graph unpooling and additional GCN layers, integrating skip connections from the encoder stage.

Formally, at each encoding and decoding step, the vertex embeddings are computed as:

$$\mathbf{H}^{(l+1)} = \mathrm{GCN}\left(\mathbf{H}^{(l)}, \mathbf{A}^{(l)}\right), \tag{A.3}$$

where $\mathbf{H}^{(l)}$ is the vertex embedding at layer l, and $\mathbf{A}^{(l)}$ denotes the adjacency matrix at layer l, updated via pooling/unpooling operations.

The final vertex representation, incorporating skip connections, is given by:

$$\mathbf{H}^{(\text{final})} = \mathbf{H}^{(L)} + \sum_{l=1}^{L-1} \mathbf{H}^{(l)}, \tag{A.4}$$

where $\mathbf{H}^{(L)}$ is the output of the decoder, and the summation represents the residual connections from the encoder.

A.3 Enhanced Graph U-Net2

We also evaluated an enhanced version of the Graph U-Net, referred to as **GCN-UNet2**, which closely follows the original architecture described previously but incorporates two significant additions aimed at improving stability and regularization. Specifically, after each GCN convolutional layer within both encoder and decoder stages, a **Batch Normalization** operation followed by **Dropout** ($p = 0.2$) was applied. This modification ensures more stable training dynamics, reduces potential overfitting, and helps maintain consistent gradient flow throughout the network.

Formally, the vertex embeddings after each enhanced convolutional layer are computed as:

$$\mathbf{H}^{(l+1)} = \text{Dropout}\left(\text{BatchNorm}\left(\text{GCN}\left(\mathbf{H}^{(l)}, \mathbf{A}^{(l)}\right)\right)\right), \tag{A.5}$$

where $H^{(l)}$ denotes the vertex embedding at layer$l$, and $A^{(l)}$ is the corresponding adjacency matrix. All other structural components, including the encoder-decoder arrangement, pooling, unpooling, and skip connections, remain identical to the original Graph U-Net implementation.

A.4 Graph Attention Network (

The GAT architecture uses the attention mechanisms to dynamically weigh the influence of neighboring vertices. In contrast to standard graph convolutional networks, GAT employs a self-attention mechanism to assign adaptive importance scores to each neighbor during aggregation, allowing it to effectively capture varying degrees of local relevance in the input graph.

The GAT architecture comprises two key steps at each convolutional layer:

1. **Feature Transformation:** The initial vertex features $\mathbf{h}_i$ are linearly projected into a higher-dimensional space through learnable weight matrices.

2. **Attention-based Aggregation:** Attention coefficients α_{ij} are computed for each pair of connected vertices (i, j), determining the contribution of vertex j's transformed features to the updated representation of vertex i. Formally, the attention coefficients are computed as:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top [\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_j]\right)\right)}{\sum_{k \in \mathcal{N}(i)} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top [\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_k]\right)\right)}, \tag{A.6}$$

where $\mathbf{a}$ is a learnable attention weight vector, $\mathbf{W}$ represents the linear transformation matrix, and $\mathcal{N}(i)$ denotes the set of neighboring vertices i.

Finally, the updated vertex embedding is computed by aggregating these features weighted by their attention scores:

$$\mathbf{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} \mathbf{W}\mathbf{h}_j\right), \tag{A.7}$$

where $\sigma(\cdot)$ denotes a nonlinear activation function, typically a LeakyReLU.

The complete GAT architecture used in our experiments employs two consecutive GAT convolutional layers, each followed by batch normalization, nonlinear activation, and dropout regularization, as illustrated in Figure A.17.

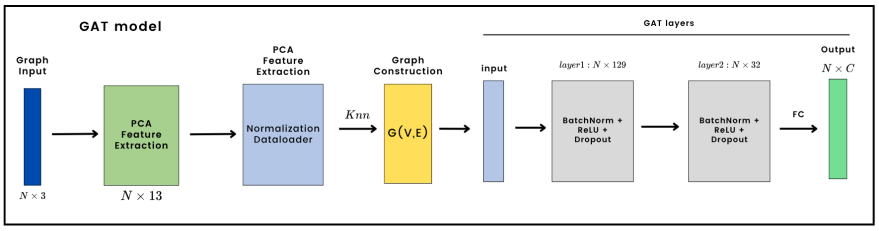

Figure A.17: Visualization of the Graph Attention Network (GAT) architecture employed as a baseline in the comparative studies.

Additional Notes

The architectures described in this appendix served as baseline models to benchmark the performance of more advanced graph-based approaches, particularly EdgeConv and GAT-based networks. Results obtained from these

simpler models provided valuable insights into the effectiveness of incorporating residual connections, multi-scale graph operations, and attention mechanisms, which ultimately guided the design and refinement of our proposed EdgeGAT architecture.

References

[1] C. Carroll, C. Carter, R. Goodhue, C.-Y. C. L. Lawell, Crop Disease and Agricultural Productivity, Tech. Rep. w23513, National Bureau of Economic Research, Cambridge, MA (Jun. 2017). doi:10.3386/w23513.
URL http://www.nber.org/papers/w23513.pdf

[2] C. Zhang, et al., Biomass and crop height estimation of different crops using uav-based 3d point cloud, Remote Sensing 12 (1) (2020) 17.

[3] Geopard, The power of crop identification, Geopard Tech Blog (2022).
URL https://geopard.tech/blog/crop-identification/

[4] Picsellia, Precision agriculture: Using computer vision for crop health monitoring, Picsellia Blog (2024).
URL https://www.picsellia.com/post/precision-agriculture-computer-vision-crop-health-monitoring

[5] A.-K. Mahlein, Plant disease detection by imaging sensors – parallels and specific demands for precision agriculture and plant phenotyping, Plant Disease 100 (2) (2016) 241–251.

[6] Geopard, Crop yield prediction with remote sensing data in precision agriculture, Geopard Tech Blog (2022).
URL https://geopard.tech/blog/predicting-crop-yield-with-remote-sensing-data/

[7] M. Bertolini, D. Mezzogori, M. Neroni, F. Zammori, Machine learning for industrial applications: A comprehensive literature review, Expert Systems with Applications 175 (2021) 114820. doi:10.1016/j.eswa.2021.114820.
URL https://www.sciencedirect.com/science/article/pii/S095741742100261X

[8] S. O. Araújo, R. S. Peres, J. C. Ramalho, F. Lidon, J. Barata, Machine learning applications in agriculture: Current trends, challenges,

and future perspectives, Agronomy 13 (12) (2023). doi:10.3390/agronomy13122976.
URL https://www.mdpi.com/2073-4395/13/12/2976

[9] K. O'Shea, R. Nash, An introduction to convolutional neural networks (2015). arXiv:1511.08458.
URL https://arxiv.org/abs/1511.08458

[10] B. Japes, J. Mack, F. Rist, K. Herzog, R. Töpfer, V. Steinhage, Multi-View Semantic Labeling of 3D Point Clouds for Automated Plant Phenotyping, arXiv:1805.03994 [cs] (May 2018).
URL http://arxiv.org/abs/1805.03994

[11] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, M. Bennamoun, Deep learning for 3d point clouds: A survey, IEEE Transactions on Pattern Analysis and Machine Intelligence 43 (12) (2021) 4338–4364. doi:10.1109/TPAMI.2020.3005434.

[12] C. R. Qi, H. Su, K. Mo, L. J. Guibas, Pointnet: Deep learning on point sets for 3d classification and segmentation, arXiv preprint arXiv:1612.00593 (2016).

[13] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: Proceedings of the International Conference on Learning Representations (ICLR), 2017.
URL https://arxiv.org/abs/1609.02907

[14] Y. Guo, H. Wang, Q. Hu, Q. Liu, L. J. Guibas, Deep learning on point clouds: A survey, IEEE Transactions on Neural Networks and Learning Systems 32 (11) (2020) 4333–4352.

[15] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, J. M. Solomon, Dynamic graph cnn for learning on point clouds (2019). arXiv:1801.07829.
URL https://arxiv.org/abs/1801.07829

[16] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, Graph attention networks, arXiv preprint arXiv:1710.10903 (2017).
URL https://arxiv.org/abs/1710.10903

[17] J. A. Bondy, U. S. R. Murty, Graph Theory with Applications, Elsevier, New York, 1976.

[18] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint arXiv:1609.02907 (2016).

[19] H. Gao, S. Ji, Graph u-nets, in: Proceedings of the 36th International Conference on Machine Learning, PMLR, 2019, pp. 2083–2092.

[20] H. Hotelling, Analysis of a complex of statistical variables into principal components, Journal of Educational Psychology 24 (6) (1933) 417–441. doi:10.1037/h0071325.

[21] K. Pearson, On lines and planes of closest fit to systems of points in space, Philosophical Magazine 2 (11) (1901) 559–572. doi:10.1080/14786440109462720.

[22] I. T. Jolliffe, Principal Component Analysis, 2nd Edition, Springer Series in Statistics, Springer, New York, 2002. doi:10.1007/b98835.

[23] M. Weinmann, B. Jutzi, C. Mallet, Feature extraction and classification of 3D point clouds for mapping of urban facades, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences II-3/W4 (2015) 57–64. doi:10.5194/isprsannals-II-3-W4-57-2015.

[24] M. Pauly, M. Gross, L. P. Kobbelt, Efficient simplification of point-sampled surfaces, Proceedings of the conference on Visualization (2002) 163–170doi:10.1109/VISUAL.2002.1183788.

[25] T. Hackel, J. D. Wegner, K. Schindler, Fast semantic segmentation of 3d point clouds with strongly varying density, in: ISPRS annals of the photogrammetry, remote sensing and spatial information sciences, Vol. 3, 2016, pp. 177–184. doi:10.5194/isprs-annals-III-3-177-2016.

[26] T. M. Cover, P. E. Hart, Nearest neighbor pattern classification, IEEE Transactions on Information Theory 13 (1) (1967) 21–27.

[27] C. R. Qi, L. Yi, H. Su, L. J. Guibas, Pointnet++: Deep hierarchical feature learning on point sets in a metric space (2017). arXiv:1706.02413.
URL https://arxiv.org/abs/1706.02413

[28] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR) (2016) 770–778.

[29] Y. Rong, W. Huang, T. Xu, J. Huang, Dropedge: Towards deep graph convolutional networks on node classification, in: International Conference on Learning Representations (ICLR), 2020.
URL https://arxiv.org/abs/1907.10903

[30] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.

[31] I. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, F. Heide, et al., Mlp-mixer: An all-mlp architecture for vision, arXiv preprint arXiv:2105.01601 (2021).

[32] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, J. M. Solomon, Dynamic graph cnn for learning on point clouds, in: ACM Transactions on Graphics (TOG), Vol. 38, ACM, 2019, pp. 1–12.

[33] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems, 2017.

[34] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization (2017). arXiv:1412.6980.
URL https://arxiv.org/abs/1412.6980

[35] Z. Ao, F. Wu, S. Hu, Y. Sun, Y. Su, Q. Guo, Q. Xin, Automatic segmentation of stem and leaf components and individual maize plants in field terrestrial lidar data using convolutional neural networks, The Crop Journal 10 (5) (2022) 1239–1250, crop phenotyping studies with application to crop monitoring. doi:10.1016/j.cj.2021.10.010.
URL https://www.sciencedirect.com/science/article/pii/S2214514121002191

[36] D. Schunck, F. Magistri, R. A. Rosu, A. Cornelißen, N. Chebrolu, S. Paulus, J. Léon, S. Behnke, C. Stachniss, H. Kuhlmann, L. Klingbeil, Pheno4d: A spatio-temporal dataset of maize and tomato plant

point clouds for phenotyping and advanced plant analysis, PLOS ONE 16 (8) (2021) 1–18. doi:10.1371/journal.pone.0256340.
URL https://doi.org/10.1371/journal.pone.0256340

[37] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, G. Ranzuglia, MeshLab: an Open-Source Mesh Processing Tool, in: V. Scarano, R. D. Chiara, U. Erra (Eds.), Eurographics Italian Chapter Conference, The Eurographics Association, 2008. doi:10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136.