# Procedural Tree Generation

## Inverse modelling of 2-d trees using graph neural networks

**Erfan Mozafari**

**Supervisor(s): Elmar Eisemann, Petr Kellnhofer**

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
January 28, 2024

Name of the student: Erfan Mozafari
Final project course: CSE3000 Research Project
Thesis committee: Elmar Eisemann, Petr Kellnhofer, Marcel Reinders

## Abstract

The most established and widely used methods for analysing tree images for tasks such as geometry analysis, segmentation and classification often rely on pixels. In this paper, the applicability of analyzing tree geometry based on a graph representation rather than a pixel-based approach is pursued. To do so, 2D renders of different species of trees are converted to spatial graph structures capturing significant points on the tree skeleton. Two independent Graph Convolutional Network algorithms which learn node (coordinate) features are then applied on the obtained dataset to assess the reliability of graph based analysis. The first experiment explores a GCN for assigning correct species labels to the skeleton graph of the original tree image, demonstrating the association between geometry and tree metadata. The second experiment, an unsupervised representation learning, is conducted by using Graph Autoencoders to obtain an embedding for each skeleton graph which can be used to reconstruct partially the same graph, demonstrating the association between GCE latent representation and geometry. Promising results were found in both cases, reinforcing the reliability of the original proposition to rely on geometry as well as pixels for tree analysis tasks.

## 1 Introduction

Trees are inspiring creatures and they have inspired computer scientists to the extent that one of the most important data structures in the field is called trees. In nature, they seem to grow chaotically out of structure, which makes them tricky to model. In the context of two dimensional tree analysis which is that of this paper, attempts at processing tree images to for instance discover recursive patterns or classify trees often rely on pixels of the tree as input. This approach has proven to get the job done. However, this paper tries to picture the whys to instead model trees with a tree (graph) structure while demonstrating the applicability and the reasonableness of doing so.

!!! Priority C: The ambitions + Tree time lapse inspiration text !!!

The main research question explored in this paper is the following:

- **RQ:** How effective and reasonable is modelling tree geometry using graphs and Graph Convolutional Networks?

To this end, two subquestions are posed and answered:

- **SQ1:** Can spatial graphs composed of point coordinates sampled from skeletonised tree images be used for supervised learning tasks, showing the possibility to go from geometrical representation to tree metadata ?
- **SQ2:** Can (potentially sparse) spatial tree graphs be converted to low dimensional vectors holding significant geometrical information capable of partially reconstructing the original graph ?

Briefly explain each section to come

## 2 Related Work

According to Okura [8], plant modelling/reconstruction techniques are categorized into 5 groups: FSPM/L-system, Graphs, Mesh, Voxels, and Point cloud. redWe dont gie a fuck abouut mesh and voxel.

### 2.1 Plant modeling and reconstruction techniques

#### FSPM Representation

L-System vs at least one more representation

#### Graph Represenation

Concerning graph represenations, some (botanical) tree reconstruction methods from multiview images track the branch patterns from the root detected from a given image [11]. Also, graph/tree based structure has been used to analyze leaf veins, which is used for leaf species classification [12]. However, there are gaps in applying the same principle to e.g. classify trees in the two dimensional space, which is one of the aims of our paper.

#### Point Cloud Representation

PointNet++ [9] is a hierarchical neural network working recursively on a nested partitioning of the input point set while exploiting metric space distances, making it able to learn local patterns and features as the scale of the context increases. This model is adopted for the supervised classification task.

### 2.2 Imaging techniques

This section ...

#### GCN Usages

Examples of gcns used to analyze images

#### Pixel Based Approaches

Collecting input data for reconstructing realistic models of real trees through e.g. obtaining 3D point clouds using laser scanners or taking multiple photos from different angles around real trees can be too costly and time-consuming. Single-image-based tree reconstruction is a harder problem, but greatly simplifies the input requirements for recreating a realistic tree model.

1 [1] created a pipeline which only tries to reconstruct the volumetric shape of the tree by extruding its outline, generating a passable but unfaithful reconstruction of the tree. 10 [10] had users draw a stroke along the trunk and around the foliage, which are then fed into a growth engine to guide the generation of a tree.

The proposed method by 5 [5] involves training neural networks to predict the 3D structure of a tree from a single image. It segments the branches and leaves from the rest of the picture and generates disc-shaped bounding boxes and classifies the tree species, after which a procedural algorithm generates a more faithful 3D model of the tree than the aforementioned two papers. Another recent paper uses a Generative Adversarial Network (GAN) to infer the 3D skeleton of a tree from a single image and generate a realistic tree model [6].

On the topic of tree classification based on 2D images, several papers look at tree top classifications based on remote sensing data and drone imagery, in both RGB and hyperspectral images [3] [7] [13]. No papers were found solely on the topic image classification of trees based on 2D images, because that is problem would be too elementary for a research paper. The reason we get into this however is to evaluate another type of classification, as discussed in section 1.

# 3 Preliminaries

## 3.1 Adjacency Matrix

## 3.2 Feature Matrix

## 3.3 Graph Convolutional Networks

Graph neural networks (GNNs) are neural models that capture the dependence of graphs via message passing between the nodes of graphs. In recent years, variants of GNNs such as graph convolutional network (GCN), graph attention network (GAT), graph recurrent network (GRN) have demonstrated ground-breaking performances on many deep learning tasks. [14]

However, according to [2], GCNs lack the ability to take into account the ordering of node neighbors, even when there is a geometric interpretation of the graph vertices that provides an order based on their spatial positions. To remedy this issue, they propose Spatial Graph Convolutional Network (SGCN) which uses spatial features to efficiently learn from graphs that can be naturally located in space.

According to Kipf [4], Most graph neural network models have a somewhat universal architecture in common which we will refer to as Graph Convolutional Networks.

For these models, the goal is then to learn a function of signals/features on a graph which takes as input: A feature description $x_i$ for every node $i$ summarized in a $ND$ feature matrix X(N: number of nodes, D: number of input features) A representative description of the graph structure in matrix form; typically in the form of an adjacency matrix $A$ (or some function thereof) and produces a node-level output $Z$ (an $NF$ feature matrix, where $F$ is the number of output features per node). Graph-level outputs can be modeled by introducing some form of pooling operation.

Every neural network layer can then be written as a non-linear function $H^{l+1} = f(H^l, A)$, with $H^0 = X$ and $H^L = Z$ (or z for graph-level outputs), $L$ being the number of layers. The specific models then differ only in how $f(\cdot, \cdot)$ is chosen and parameterized.

## 3.4 (Variational) Graph Autoencoders

Variational graph autoencoders (VGAE) apply the idea of VAEs on graph-structured data. They have been successful on a number of citation network datasets such as Cora and Citesser.

The encoder (inference model) of GAE consists of graph convolutional networks (GCNs). It takes an adjacency matrix A and a feature matrix X as inputs and generates the latent variable Z as output. The first GCN layer generates a lower-dimensional feature matrix. The decoder (generative model) is defined by an inner product between latent variable Z. After encoding a node to a Z in the latent space, the similarity of each node embedding in that space is computed to generate the output adjacency matrix. Inner product calculates the cosine similarity of two vectors in that space.

# 4 Method

## 4.1 Dataset

120 images of trees all along the same camera angle were rendered using the software Tree-it [CITE]. The dimensions of each image was 512 x 512 pixels. It should be noted the dataset was obtained independently. The 120 trees comprised of 6 different species, more specifically: 50 Maple trees, 30 Acacia trees, 10 Beech trees, 10 Oak trees, 10 Pine trees, and 10 Willow trees.

### Skeletonization

Prior to skeletonization of each image, due to bugs in the software used, manual alpha blending had to be conducted on each image in order to be able to process them. After applying a threshold on the grayscale image equal to the mean value of the same image, sci-kit's [CITE] skeletonize function is used to obtain the skeleton of the tree. The skeleton object is then passed to a chain of dilation, erosion and closing. [FIGURE Original Tree, FIGURE Skeleton]

**Minimal Skeleton**

The minimal skeleton in this paper refers to a graph structure composed of nodes representing important parts of the original skeleton, mainly split points and segment tails. Edges in the graph represent connecting segments between the two points in the original image. These minimal graph skeletons form the dataset used for the autoencoder experiment. In order to locate the most important points on the obtained skeleton object, the skeleton is converted to a graph object whose independent connected paths are traversed in order to obtain different segments. The 25 longest segments are then identified and their start and end points are sampled, resulting in 50 points most of which represent split points on the trunk or the end point of the branches. Creating a graph only with edges connecting these nodes often results in disconnected graphs. Hence, a minimum spanning tree connecting all the obtained points is formed to arrive at a final representative graph structure. Each node has one attribute called pos which is a tuple of normalized coordinates. [FIGURE Skeleton, FIGURE Sampled Points, FIGURE Disjunctions, FIGURE MST]

**Dense Skeleton**

The graph classification experiment requires graphs with more sampled points. Therefore this time only 1/3 of the points in the skeleton object were sampled randomly. In this case, the input of the model is purely point clouds. The model to be used itself forms edges between the points. Both datasets, dense and minimal, loaded from node-link format are then converted into PyTorch custom datasets and batched and split for training and testing.

## 4.2 Experiments

**GCN-Based Skeleton Classification**

In this experiment, isolated nodes from the disjoint dense skeleton dataset (120 graphs) are fed into the PointNet++ [CITE] neural network, originally proposed for 3d point cloud classification/segmentation, which tries to capture meaningful local geometric patterns to be able to classify the entire graph. We only had to manually reshape the pos coordinates of the nodes to have a 0.0 third dimension as well for it to work. In PointNet++, the nodes are iteratively processed by following a simple grouping, neighborhood aggregation and downsampling scheme:

The grouping phase in our case uses a k-nn algorithm to construct a graph in which nearby points are connected. The constructed graph is then processed in a Graph Neural Network layer by aggregating coordinate values for each node from their direct neighbors. This layer is what allows PointNet++ to capture local structures at different scales. Finally, in the downsampling phase, a pooling scheme which summarizes local structures containing aggregated skeleton coordinate positions is adopted.

This experiment is the result of applying the PyG implementation of this model [CITE] to our dense skeleton dataset. As a variation of this experiment, the minimal skeleton dataset is also fed into the model. 80% of the graphs in both cases were used for training while 20% of them for testing.

Concerning implementation details, as visible in the provided code by PyG, a (k=16)-nearest neighbor graph based on the position pos of nodes is generated on which two graph-based convolutional operators are applied and enhanced by ReLU nonlinearities. The first operator takes in 3 input features (the positions of nodes) and maps them to 32 output features. The second operator is applied on the 32 dimension output from the previous step remaining in the same dimension. In the end a global graph readout function, global-max-pool, which takes the maximum value along the node dimension for each skeleton graph is applied which is then fed to a linear classifier mapping it to 6 different classes (1:Oak, 2:Maple, 3:Acacia, 4:Pine, 5:Beech, 6:Willow respectively). Three variations of this experiment, namely experimenting with different values for k, an additional convlutional middle layer and running the model on the minimal skeleton dataset as well as the dense one is also conducted.

[FIGURE VISUALISATION STEPS]

**GCN-Based Latent Representation With Graph Autoencoders**

The vector encodings associated with the skeleton graphs in the previous experiment lack one important characteristic. There are no guarantees the vectors are representative enough to be able to reconstruct the original skeleton back. Hence, in this experiment, a Graph Autoencoder is utilised in which the vectors produced by the encoder layer are made in such a manner that they are able to reconstruct the adjacency matrix in the original graph back ( done by minimizing the inner product between each node and its embedding during the process). For this experiment, the more challenging dataset containing the minimal skeleton graphs is used. This means the Graph AutoEncoder will learn to take a skeleton graph and compress the node positions and the edge connections into a low dimensional vector capable of being used to reconstruct the edges between different points on the skeletons, i.e constructing the links of the original graph.

# 5 Results

In this chapter, the results of the previously laid out two experiments are reported.

## 5.1 Skeleton Classification

As explained in the previous chapter, the PointNet++ [CITE] neural network was applied to our dataset. 82% accuracy in label prediction was observed with the first successful training and testing of the model on the dense skeleton dataset to assign a species label to each. Three seperate subexepriments were conducted to further examine and improve the prediction score, namely: applying the algorithm to the minimal skeleton dataset, restructuring the hidden layer of the model, and variations in the initial graph formation of the model. A streamline of the process looks as follows :

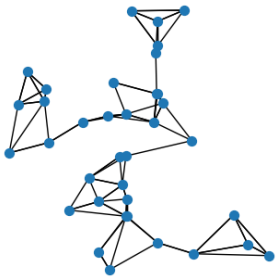

Figure 1: 20k steps of training and its performance on test sets

### Dataset Experimentation

Following the initial success on the dense dataset, the model was tested also on the minimal skeleton dataset stripped of the edges in the graph. Quite surprisingly, as observed in figure X, the model classified the skeletons more accurately, jumping up to 87% accuracy in the dataset with fewer points (35 points on average for each minimal skeleton vs 761 points on average for each dense skeleton).

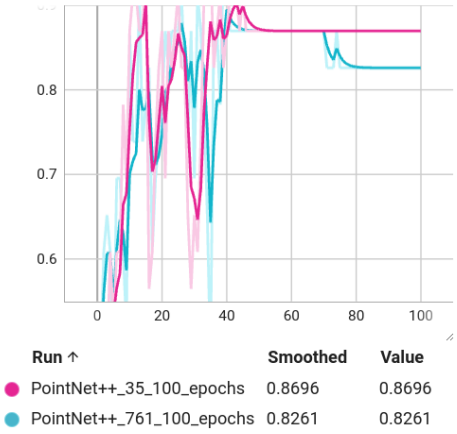


Figure 2: PointNet++ performance on dense (blue) vs the minimal dataset (pink)

### Hidden Convolutional Layer Experimentation

The used implementation of the model originally had placed two convolutional 32x32 layers for aggregating the node coordinates. As expected with other standard deep neural nets, the addition of an extra 32x32 layer to the process, saw the model jump 4% in the predictive power up to 95% as observed in figure X.


Figure 3: PointNet++ accuracy with three convolution layers (red) instead of two (green)

### Input Formation Experimentation

A critical part in the implementation of PointNet++ concerns the formation of a custom k-nn graph since the model expects a point cloud. Originally the value was set to k=16. Given the two dimensional nature of this task and the special case of points representing tree skeletons, the model was adjusted and tested for three more cases of k=1, k=4, and k=8 both in the case of three convolution layers, and with two, resulting in eight different variations.

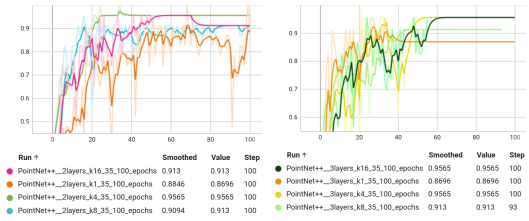

Figure 4: 20k steps of training and its performance on test sets

In both architectures, k=4 was found to produce the best results: 95% testing accuracy. In the case of 3 convolution layers the original k=16 performed as well as k=4. In both cases k=1 performs the worst. The performance for these values of k are understandable because, in the case of a minimal skeleton which this experiment was based on, message

passing and association between a reasonable number of neighbours is the optimal mode of analysis as geometrical information is not isolated in each node as in the case of k=1 and not too spread out in the whole structure in the case of k=16 or k=8 with two convolution layers.

In summary, the customized PointNet++ model described above, was able to predict the original tree label with a 95% accuracy through learning to map shared and convolved geometrical patterns between the critical points in the minimal skeleton of the trees to metadata.

### 5.2 Skeleton Representation

## 6 Discussion

### Limitations and Future Work

Even though for each skeleton graph, in the previous experiment the model learns to arrive at a latent space which can be linearly classified to different labels by using only pure geometrical information (node positions) and no other node or edge features whatsoever, the argument can be made that such a supervised classification method could learn to cheat its way by learning special easy mappings from input to labels. There are two possible answers to this: 1. There is no back propagation in this model which means the final classifier layer and the tree labels do not influence the previous layers producing the low dimensional vectors. 2. Even if the model learns shortcuts from the graph, it would only do so purely based on the positions of the nodes, and not based on node degrees for example, since the skeletons are fed as raw point cloud data with no edges. Since there are no other node or edge features, this would actually demonstrate the model has learned key geometrical patterns specific to that species of trees, allowing it to cheat with a master key. It is argued that if there are learned cheat patterns, they can be treated as a key representative pattern allowing for further analysis based on only that sub graph instead of the original graph.

With that said, to further practically demonstrate the previously mentioned arguments, a second experiment based on the concept of Autoencoders was designed to gauge the viability of embedding the skeleton graphs independent of metadata.

### Conclusion

### Ethical Considerations

## A Appendix

## References


[1] Oscar Argudo, Antonio Chica, and Carlos Andujar. Single-picture reconstruction and rendering of trees for plausible vegetation synthesis. *Computers & Graphics*, 57:55–67, 2016.

[2] Tomasz Danel, Przemysław Spurek, Jacek Tabor, Marek Śmieja, Łukasz Struski, Agnieszka Słowik, and Łukasz Maziarka. Spatial graph convolutional networks, 2020.

[3] Fabian Ewald Fassnacht, Hooman Latifi, Krzysztof Stereńczak, Aneta Modzelewska, Michael Lefsky, Lars T Waser, Christoph Straub, and Aniruddha Ghosh. Review of studies on tree species classification from remotely sensed data. *Remote sensing of environment*, 186:64–87, 2016.

[4] Thomas Kipf. Graph convolutional neural networks, 2016.

[5] Bosheng Li, Jacek Kałużny, Jonathan Klein, Dominik L Michels, Wojtek Pałubicki, Bedrich Benes, and Sören Pirk. Learning to reconstruct botanical trees from single images. *ACM Transactions on Graphics (TOG)*, 40(6):1–15, 2021.

[6] Zhihao Liu, Kai Wu, Jianwei Guo, Yunhai Wang, Oliver Deussen, and Zhanglin Cheng. Single image tree reconstruction via adversarial network. *Graphical Models*, 117:101115, 2021.

[7] Somayeh Nezami, Ehsan Khoramshahi, Olli Nevalainen, Ilkka Pölönen, and Eija Honkavaara. Tree species classification of drone hyperspectral and rgb imagery with deep learning convolutional neural networks. *Remote Sensing*, 12(7):1070, 2020.

[8] Fumio Okura. 3d modeling and reconstruction of plants and trees: A cross-cutting review across computer graphics, vision, and plant phenotyping. *Breeding Science*, 72(1):31–47, 2022.

[9] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space, 2017.

[10] Ping Tan, Tian Fang, Jianxiong Xiao, Peng Zhao, and Long Quan. Single image tree modeling. *ACM Transactions on Graphics (TOG)*, 27(5):1–7, 2008.

[11] Ping Tan, Gang Zeng, Jingdong Wang, Sing Bing Kang, and Long Quan. Image-based tree modeling. In *ACM SIGGRAPH 2007 papers*, pages 87–es. 2007.

[12] Xin Wei, Ruixuan Yu, and Jian Sun. View-gcn: View-based graph convolutional network for 3d shape analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[13] Chen Zhang, Kai Xia, Hailin Feng, Yinhui Yang, and Xiaochen Du. Tree species classification using deep learning and rgb optical images obtained by an unmanned aerial vehicle. *Journal of Forestry Research*, 32(5):1879–1888, 2021.

[14] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.