Tree Instance Segmentation with Temporal Contour Graph

Adnan Firoze Cameron Wingren† Raymond A. Yeh Bedrich Benes Daniel Aliaga
Department of Computer Science, Purdue University
Department of Forestry and Natural Resources†, Purdue University
{afiroze, cwingren, rayyeh, bbenes, aliaga}@purdue.edu

Input Human Annotation Ours

Figure 1. **Tree Instance Segmentation.** A flyover from a UAV and the corresponding tree instance segmentation during green leaf season when the tree crowns are most similar to their neighbors, and occlusion is complex adding to the difficulty in segmentation.

Abstract

We present a novel approach to perform instance segmentation and counting for densely packed self-similar trees using a top-view RGB image sequence. We propose a solution that leverages pixel content, shape, and self-occlusion. First, we perform an initial over-segmentation of the image sequence and aggregate structural characteristics into a contour graph with temporal information incorporated. Second, using a graph convolutional network and its inherent local messaging passing abilities, we merge adjacent tree crown patches into a final set of tree crowns. Per various studies and comparisons, our method is superior to all prior methods and results in high-accuracy instance segmentation and counting despite the trees being tightly packed. Finally, we provide various forest image sequence datasets suitable for subsequent benchmarking and evaluation captured at different altitudes and leaf conditions.

1. Introduction

Trees in forests are tightly spaced, partially overlapping 3D objects with complex boundaries. Tree instance segmentation is critical in several domains. For example, ecosystem services and agriculture need to segment and count trees in large areas in order to obtain information about the ecological balance, environmental health, and timber inventory. Counting trees from the ground perspective is inefficient, does not scale, and is challenging to automate because of many occlusions with branches and low accesibility. In this paper, we address tree instance segmentation and counting using overhead RGB image sequences captured by unmanned aerial vehicles (UAVs), especially during the green-leaf season when trees are most self-similar; see Fig. 1 for an illustration.

There is significant prior work in segmentation and counting, particularly in the field of instance segmentation. While some approaches make use of LiDAR or RGB-D images (see the survey paper [**?**]), we focus on using easier-to-obtain uncalibrated RGB image sequences. Prior works based on uncalibrated RGB images can be largely organized into three groups. The first group seeks to count individual objects. These approaches often use density estimation and do not focus on segmentation [**?** **?**]. The second group of methods relies on convolutional neural networks (CNNs) applied directly to image pixels, such as Mask R-CNN [**?**]. However, in the case of abutting and self-similar objects, e.g., trees, distinguishing individual instances is hard. The third group of techniques (e.g., **?** **?**]) model object contour as a graph, where each pixel corresponds to a node, and makes use of graph convolutional networks to complete individual contours; nevertheless, abutting and self-similar instances also hinder these methods. Yet other methods, such as those in digital forestry research, exploit domain-specific features such as assumed differences between tree species or fall leaf coloring (i.e., during one brief time period of the year, the leaf color of adjacent crowns is often

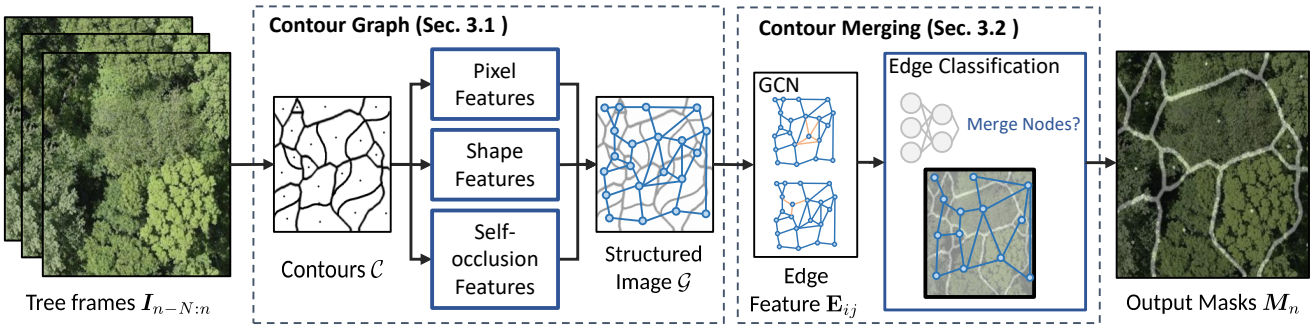

Figure 2. **Pipeline:** The input image sequence is analyzed. Initial contours and features are detected and organized into a contour graph that is refined by merging edges and nodes, resulting in the final output mask.

different).

Our approach is motivated by a key observation that two tree crown leaf patterns tightly packed together are highly similar, and additional features beyond leaf patterns are necessary to perform segmentation. Hence, we consider features based on the tree crown shape because it is unlikely to observe a rectangular tree crown. Beyond shape features, we also use the changing self-occlusion patterns captured in the different frames to aid segmentation.

At a high level, our proposed approach simultaneously exploits pixel content, shape, and self-occlusion, which collectively define a graph-based structure that we call a *contour graph*. Each node corresponds to an initial oversegmentation of a tree crown fragment; i.e., each node corresponds to a *region enclosed* by a closed contour. We then learn features on this contour graph via graph convolutional network (GCN) to determine which nodes should be merged. Our method decides whether two nearby regions correspond to the same tree crown. Notably, a tree crown fragment is subject to various simultaneous features that we can exploit to discern one tree crown from another, even if one tree crown is of the same species and has a very similar leaf pattern and color to an adjacent tree crown. Altogether this leads to an instance segmentation method that can process overhead RGB image sequences of dense forests even when all leaves are mostly similar in color during summer. See Fig. 2 for an overview of our approach.

When creating and evaluating model performance, we are not aware of suitable databases on dense forests with subsequent frames. To address this: (a) We leveraged developmental tree models [**?** **?**] to produce a synthetic dataset with annotated tree crowns (5,157 trees in total); (b) We manually labeled real-world image sequences captured by UAV over three large forests (6,527 trees in total), collectively spanning approximately 3,680,000 m^2. We will make our self-collected datasets publicly available.

On these datasets, we show that the proposed method achieves a segmentation accuracy of 73.6 and a count accuracy of 89.8% on average, which is compared to multiple recent instance segmentation approaches.

Our main contributions include:

- an instance segmentation method to robustly process densely packed trees where the instances are abutting, partially overlapping, and self-similar,
- tree crown counting, which is beneficial to ecosystem services and digital forestry, and
- a curated dataset of multiple labeled and unlabeled temporally continuous dense forests suitable for future research.

2. Related Work

Counting Methods. Instance segmentation and counting have been pursued by various approaches, including methods using input data beyond uncalibrated RGB image sequences. Counting methods use stochastic mechanisms to estimate the number of instances. For example, **?**] use RGB-D data and density estimation to distinguish instances at a relatively low resolution of individuals in a dense crowd of humans. **?**] perform a density-based estimation of trees but use 12 months of satellite data. **?**] use RGB-thermal imagery to perform crowd counting using a multi-modality deep network and exploiting the thermal signatures of humans. In our case, each instance is relatively larger (i.e., occupies many pixels). Thus resolving them is not a challenge because of limited resolution but rather difficult due to similarity, semi-transparency, tight spacing, and partial overlaps with complex boundaries. A density-based method would be too inaccurate and would not provide a segmentation.

Detection and Instance Segmentation. Object detection methods largely fall into two-stage and single-stage methods. Two-stage detection, such as Faster R-CNN [**?**] and Fast R-CNN [**?**], predict object masks based on region proposal and bounding box regression heads using features extracted from convolutional neural networks (CNN). Single-stage detection [**?** **?** **?** **?** **?**], bypasses the regional proposal stage. Notably, YOLO-based methods [**?** **?** **?** **?**] achieve efficient detection of objects in real-time. Nevertheless, when applied to abutting and self-similar content, e.g., tree crowns, the precision of these approaches is low

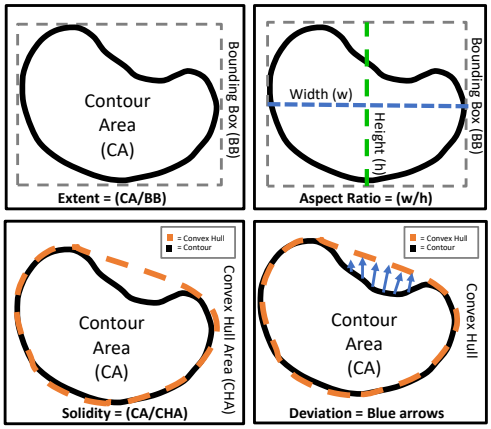

Figure 3. **Shape Features.** Illustration of the extracted shape features, including extent, aspect ratio, solidity, and deviation.

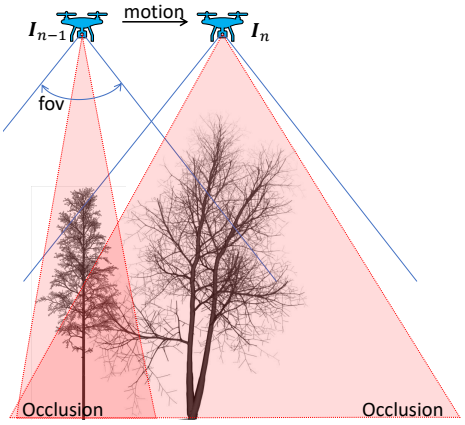

Figure 4. **Self-Occlusion.** Consecutive frames $\boldsymbol{I}_{n-1}$ and $\boldsymbol{I}_n$ from the moving UAV capture slightly different self-occlusion patterns.

for tree counting.

Many instance segmentation methods [**?** **?** **?** **?** **?** **?** **?** **?**] have been developed. Earlier works use CNN features and build on the detection methods, e.g., Mask R-CNN [**?**]. More recently, several transformer-based instance segmentation methods [**?** **?** **?**] have been proposed as well, for instance, for instance, SwinTransformer [**?** **?**] uses a transformer-based backbone together with Mask R-CNN. Some methods use graph convolutional networks (GCN) to perform instance segmentation [**?** **?**]. For example, **?**] models a graph corresponding to a sequence of pixels (nodes) defining the contour (connectivity) of object instances aiming to model occlusions among the objects. Different from our approach, we define a node to be the "region" enclosed by a contour and exploit the change in "self-occlusion", through time, within each object (i.e., tree crown). In this work, we compare several of the recent instance segmentation methods, including Mask-RCNN with ResNet and Swin-T backbone [**?** **?**], TraDes [**?**], and BoundaryFormer [**?**].

Digital Forestry and Remote Sensing often use CNNs (e.g., see summary by **?**] on remote sensing) and exploit domain-specific characteristics. One methodology is to acquire LIDAR data from "underneath" the tree crowns using UAVs [**?**]. The data can be used to measure trunk diameter and counts, but unfortunately, it does not scale as flying through large dense forests is a significant challenge. Another strategy is to use NDVI, an image-based vegetation index, to distinguish vegetation from non-vegetation. However, this index cannot separate one tree crown from another. Another application is tree classification, which also relies on CNNs [**?**] and the instance segmentation of trees is suggested as future work.

Other approaches exploit the visual differences between tree species. For example, **?**] uses a CNN and UAV captured overhead RGB data to segment and classify trees. However, the segmentation component of this work is done in the peak fall season (when leaves of adjacent trees are most likely a different color) and then subject to *manual correction* to determine a good set of segmented trees. They claim accurate instance segmentation of trees as a challenging task for future work.

Tree modeling has a long history in computer vision and computer graphics [**?**]. Early methods focused on fractal-based approaches [**?** **?** **?**], and later methods simulate botanical plant model [**?**] development [**?** **?** **?**], including competition for resources [**?** **?**], climbing vegetation [**?**], ecosystems [**?**], or plants interacting with wind [**?**] or fire [**?**]. Synthetic tree models are perceived as highly realistic [**?**] and provide sufficient details to bridge the simulation-to-real gap for vision-based approaches. We leverage these tree models to help develop the approach presented in this paper.

3. Tree Crown Instance Segmentation

We formulate our problem as computing a set of instance segmentation masks for an uncalibrated input image sequence. The input image in a sequence is denoted by $\boldsymbol{I}_k \in (I_1, \ldots, I_N)$ and the corresponding set of predicted instance segmentation masks for $\boldsymbol{I}_k$ is denoted by $\boldsymbol{M}_k = \{M_{t,k}\ \forall t \in \mathcal{T}_k\}$ for the tree crowns $t \in \mathcal{T}_k$ in the current frame $\boldsymbol{I}_k$. Each segmentation mask of a tree crown is represented by a set of pixels $\boldsymbol{M}_{t,k} = \{(u, v)\}$. Our approach creates a contour graph representation capturing an initial over-segmentation of the forest and formulates the prediction of tree crown masks as merging the over-segmented regions.

3.1. Contour Graph Creation

We create a graph-based data structure $\mathcal{G} = (\mathcal{C}, \mathcal{E})$ (*contour graph*) for assisting with differentiating tightly-spaced and highly-similar tree crowns. The contour graph readily stores different types of feature sets and supports merging nodes via edge collapse operations.

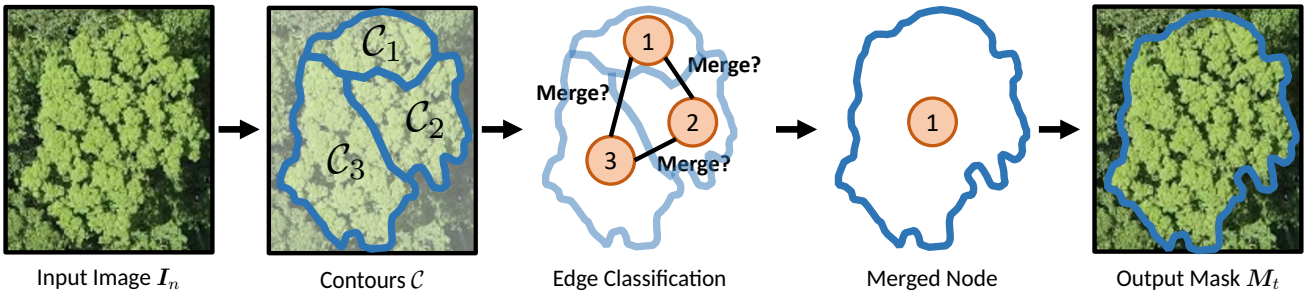

Figure 5. **Merging Process:** Illustrative presentation of our process to/from contour and graph spaces to merge noisy contours using edge classification (detailed in Sec. 3.2).

Contour Graph Construction. We create a graph where each node corresponds to a closed contour $\mathcal{C}_t = \{(u, v)\}$, i.e., the set of pixels of a tree crown fragment in the image. Two contours (nodes) have an edge between them $(\mathcal{C}_i, \mathcal{C}_j) \in \mathcal{E}$, if they are adjacent to each other in the image.

We extract the initial contours by using deep edge detection [**?**] that produces an edge map with plausible tree crown fragments. The extracted edge map is often noisy so we apply Guo-Hall algorithm [**?**] to skeletonize the edge map. Then, a simple polygonal contour can be used to represent the pixels of each closed region (i.e., each tree crown fragment). In the following, we discuss how to extract three feature set types that are stored within each node.

Pixel Features represent the pixel content within a contour. We use the following features: *Patch pixels*: we extract the center $p \times p$ RGB image patch within each contour, using $p = 30$; *Pixel similarity*: for a contour, we compute the similarity of its center patch to the center patches of neighboring contours. We measure similarity using LPIPS [**?**] which corresponds highly to human perception.

Shape Features. We capture the geometric shape features of a contour (see Fig. 3) including its *area*, i.e., $|\mathcal{C}_i|$, *extent* given as the ratio of contour area to its bounding rectangle area, *aspect ratio* of $\mathcal{C}_i$'s bounding box, *solidity* given by the ratio of the area to the contour's convex hull, and *deviation* of a contour from its convex hull.

Self-occlusion Features. We capture the self-occlusion pattern of a contour. Fig. 4 depicts a scenario of two adjacent trees. The pattern of how the leaves and branches of the left tree exhibit their occlusion relationship as the image sequence was captured is likely different than the pattern of the right tree. This is true for trees of different species but also for trees of similar species. Although both branching patterns of adjacent trees originate from approximately the middle of the tree crown, they are not the same pattern. The optical flow of the pixels within each tree crown fragment can be compared and thus used to help differentiate one self-occlusion pattern from another.

We use the Gunnar-Farneback [**?**] optical flow algorithm between images $\boldsymbol{I}_n$ and $\boldsymbol{I}_{n-1}$. We then extract the center $p \times p$ patch from the flow map to create the flow feature vector for each contour.

3.2. Contour Merging

With the contour graph constructed, we observe that a single tree crown is often split into multiple contours. In other words, to predict accurate instance segmentation, one would need to merge these contours (Fig. 5). Hence, we formulate contour merging as an edge classification problem on a graph.

Edge Classification. For a set of contours $\mathcal{C}$, we represent whether pairs of contours belong to the same instance via a matrix $\boldsymbol{Y} \in \{0, 1\}^{|\mathcal{C}|\times|\mathcal{C}|}$, where $\boldsymbol{Y}_{ij} = 1$ indicates that contours $\mathcal{C}_i$ and $\mathcal{C}_j$ should be merged. Given a contour graph input, we aim to build a model that predicts this merge matrix. We formulate this task as an edge classification problem of the contour graph.

We have extracted various features associated with each node in the graph (Sec. 3.1). We concatenate these features into a node feature matrix $\boldsymbol{H}^0 \in \mathbb{R}^{|\mathcal{C}|\times d}$. Next, we represent the edges via an adjacency matrix $\boldsymbol{A} \in [0, 1]^{|\mathcal{C}|\times|\mathcal{C}|}$.

Graph Convolution. Using a graph convolution layer [**?**], we perform L rounds of message-passing to aggregate information from each node's neighborhood:

$$\boldsymbol{H}^{(l+1)} = f^{(l)}_{\text{GCN}}(\boldsymbol{H}^{(l)}) = \sigma\left(\tilde{\boldsymbol{A}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}\right), \tag{1}$$

where $\tilde{\boldsymbol{A}}$ denotes the degree normalized adjacency matrix, σ denotes an element-wise non-linearity, and $\boldsymbol{W}^{(l)}$ denotes the trainable weights at the l^{th} layer.

To create the edge features for classification, following **?**], we average the L^{th} node features $\boldsymbol{H}^{(L)}$ for the pair of nodes in an edge $(\mathcal{C}_i, \mathcal{C}_j)$:

$$\boldsymbol{E}_{ij} = \frac{1}{2}\left(\boldsymbol{H}^{(L)}_i + \boldsymbol{H}^{(L)}_j\right). \tag{2}$$

Given the edge features $\boldsymbol{E}_{ij}$, we predict whether a pair of nodes should be merged using a multi-layer perceptron, i.e,

$$\hat{\boldsymbol{Y}}_{ij} = \text{MLP}(\boldsymbol{E}_{ij})\ \forall(\mathcal{C}_i, \mathcal{C}_j \in \mathcal{E}). \tag{3}$$

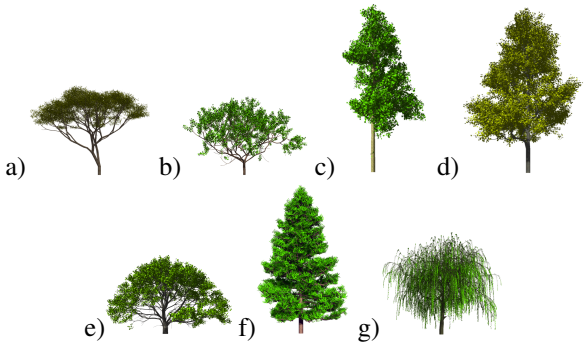

Figure 6. **Tree Models.** Examples of synthetic trees: a) acacia, b) apple, c) birch, d) maple, e) oak, f) pine, and g) willow).

To train this model, we minimize the binary cross-entropy loss between the predicted merge matrix $\hat{\boldsymbol{Y}}$ and the ground truth merge matrix $\boldsymbol{Y}$ over the edges:

$$\mathcal{L} = - \sum_{\{(i,j):\boldsymbol{A}_{ij}=1\}} \boldsymbol{Y}_{ij} \log(\hat{\boldsymbol{Y}}_{ij}) + (1 - \boldsymbol{Y}_{ij}) \log(1 - \hat{\boldsymbol{Y}}_{ij}).$$

Finally, we merge the contours according to the predicted merge matrix $\hat{\boldsymbol{Y}}$ and output an instance mask for each of the merged contours. In other words, for a tree t, the predicted mask is the union of all pixel locations within merged contours, i.e.,

$$\boldsymbol{M}_t = \bigcup_{j:\boldsymbol{Y}_{tj}=1} \mathcal{C}_j. \tag{4}$$

3.3. Collection of Datasets

We have prepared four datasets to develop and evaluate our method. We have made the synthetic and real-world dataset (with videos) publicly available. Please visit https://github.com/adnan0819/Tree-Instance-Segmentation-using-Temporal-Structured-Images.

Synthetic Dataset. A first dataset is created synthetically and enables us to control the density, species, and coloring of the trees. We use the algorithm for plant competition for resources [**?**] implemented as a developmental model [**?**] controlled by user-defined parameters [**?** **?**] and its realism has been validated by a prior large user study [**?**]. The trees have been placed by ecosystem simulation [**?** **?**]. Our implementation supports acacia, apple, willow, maple, birch, oak, and pine trees (see Fig. 6). We can generate terrains of arbitrary size full of trees.

Real-world Datasets. We have prepared three real-world forest datasets. Forest A (Martell Foret, West Lafayette, IN) was collected by our team with a UAV flying over a large forest and we made this dataset available publicly. To our knowledge, it is the largest and most comprehensive UAV-captured overhead image sequence dataset for forest research. Moreover, although not used here, each image includes georegistered coordinates of the camera location.

Methods	GT	Pred.	Acc. ↑
YOLOv7	2172	3442	41.5
YOLOv7 + Flow	2172	3134	55.7
YOLOv7 + Flow (Med.)	2172	3016	61.1
YOLOv7 + Flow (Mean)	2172	2838	69.3
Ours	2172	2373	**90.7**

Table 1. **Preliminary experiments.** Count accuracy on synthetic forest verifying the effectiveness of capturing self-occlusion using optical flow. Med. and Mean correspond to thresholding the optical flow based on the median or mean magnitude.

The UAV was piloted by a trained forestry pilot and used two cameras: DJI P1 and DJI H20T. Three distinct forest regions were captured using 23 flights and covering approximately 368 hectares (3,680,000 m^2) in total. The image sequences were captured at 80m, 100m, and 120m altitudes with the camera sensors pointing straight down (i.e., nadir). The DJI P1 camera had a field of view (FOV) of $63.5°$, and DJI H20T had a FOV of $82.9°$. All flight speeds were 5m/s. Each image has a resolution of 3,840 $\times$ 2,160 pixels and was captured at the rate of 60 images per second.

To evaluate the generalization capability (i.e., *out-of-distribution*), we collected Forest B (Kentucky Ridge State Forest, KY) and C (Olympic National Forest, WA) from Google Earth. We annotated twenty images at 832 $\times$ 732 pixels with tree crown information. Forest B and C are only used for validation. Notably, these forests have different characteristics. Forest A has a combination of natural and plantation vegetation and combines both deciduous and coniferous species like white oak, black cherry, red oak, etc. In contrast, Forest B contains more color variation without any plantation-type forest with species like sugar maple, tulip poplar, various oaks, hemlocks, etc. Differently, Forest C contains both seasonally changing and evergreen species. These differences in datasets B and C make them suitable for out-of-distribution evaluation.

4. Results

We evaluate our proposed approach using the Synthetic, Forest A, B, and C datasets. We report on the task of tree crown instance segmentation and counting. Note that instance segmentation systems trivially generalize to counting by predicting the number of detected instances. Beyond quantitative comparisons with baselines, we also conduct ablation studies verifying the efficacy of our proposed components and qualitative demonstrations.

Experiment Setup. For synthetic data experiments, we use 80% of the data to train and report evaluation metrics on the remaining 20%. In real-world experiments, we train on Forest A also using an 80-20 train and validation split. Using the same model trained on A, we report performance

Methods	Synthetic Forest			Forest A			Forest B			Forest C		
	AP ↑	AP_{50} ↑	AP_{70} ↑	AP ↑	AP_{50} ↑	AP_{70} ↑	AP ↑	AP_{50} ↑	AP_{70} ↑	AP ↑	AP_{50} ↑	AP_{70} ↑
Mask-RCNN (ResNet)	27.1	53.6	50.1	33.4	57.3	54.1	39.2	58.8	56.1	35.1	57.9	58.4
Mask-RCNN (Swin-T)	59.8	70.2	68.3	64.6	74.1	70.5	69.5	**77.3**	**72.4**	63.2	72.6	70.3
TraDeS	61.1	55.2	64.4	58.1	71.3	66.8	63.7	73.9	70.5	59.6	70..4	64.1
BoundaryFormer	56.3	65.1	57.5	60.9	72.9	66.2	64.1	73.4	69.2	58.9	71.2	61.8
Mask2Former	62.4	65.2	63.9	59.7	64.1	63.2	64.1	67.5	63.8	61.9	63.1	62.9
MS-RCNN	66.2	68.4	65.8	64.8	71.3	68.5	66.4	70.2	69.3	64.2	69.3	65.8
OCISIS	59.1	63.5	61.8	55.7	58.6	58.1	60.8	68.2	66.8	60.2	62.8	60.9
SLIC (Superpixel)	23.7	27.3	24.2	20.7	24.8	23.6	22.5	27.3	23.7	20.1	27.5	24.6
Aerial Laser	71.1	72.9	70.8	70.4	75.2	71.1	65.1	68.2	66.8	65.3	68.7	64.8
Ours	**74.6**	**73.1**	**69.5**	**74.5**	**81.6**	**72.8**	**69.8**	76.2	71.5	**70.1**	**75.4**	**72.5**

Table 2. **Segmentation.** Comparisons of segmentation performance to prior instance segmentation baselines.

Methods	Synthetic Forest			Forest A			Forest B			Forest C		
	GT	Pred.	Acc. ↑	GT	Pred.	Acc. ↑	GT	Pred.	Acc. ↑	GT	Pred. ↑	Acc. ↑
Mask-RCNN (ResNet)	5157	3291	63.8	2314	1691	73.1	2041	1281	62.8	2172	1403	64.6
Mask-RCNN (Swin-T)	5157	4275	82.9	2314	1816	78.5	2041	1655	81.1	2172	1791	82.5
TraDeS	5157	4131	80.1	2314	1987	85.9	2041	1596	78.2	2172	1611	74.2
BoundaryFormer	5157	3981	77.2	2314	1950	84.3	2041	1549	75.9	2172	1713	78.9
Mask2Former	5157	3290	63.8	2314	1708	73.8	2041	1601	78.4	2172	1681	77.4
MS-RCNN	5157	3527	68.4	2314	1886	81.6	2041	1645	80.6	2172	1752	80.7
OCISIS	5157	4373	84.8	2314	1969	85.1	2041	1643	80.5	2172	1822	83.9
SLIC (Superpixel)	5157	8142	42.1	2314	3566	45.9	2041	3298	38.4	2172	3521	37.9
Aerial Laser	5157	4399	85.3	2314	1960	84.7	2041	1639	80.3	2172	1887	86.9
Ours	5157	5636	**90.7**	2314	2510	**91.5**	2041	1771	**86.8**	2172	2384	**90.2**

Table 3. **Counts.** Comparisons of tree crown count performance to prior instance segmentation baselines.

on B and C to evaluate out-of-distribution generalization.

Evaluation Metrics. We follow the evaluation protocol from the prior work [**?**]. For *tree instance segmentation*, we report the mask Average Precision (AP) averaged over different intersection-over-union (IoU) thresholds and at IoU thresholds of 0.5 and 0.7, denoted as AP_{50} and AP_{70} respectively. For *tree counting*, we report the raw number of counts and the count accuracy, which is expressed as a percentage denoted by Acc.

Baselines. We compare our approach to the following instance segmentation methods: Mask-RCNN based on ResNet [**?**], Mask-RCNN based on the recent transformer backbone (Swin-T) [**?**], Track-to-Detect-and-Segment(TraDeS) [**?**], BoundaryFormer [**?**] (a recent mask-supervised polygonal boundary approach to instance segmentation using transformers), Mask2Former [**?**], Mask Scoring RCNN [**?**], Object Counting and Instance Segmentation with Image-level Supervision [**?**], SLIC (superpixels) [**?**], and Aerial Scanning Using Laser and Deep Model (a model specific to trees) [**?**].

4.1. Synthetic Data Experiments

Preliminary Experiments. We used our synthetic data to aid the development of our method. In particular, we used it to understand the impact of determining the change in self-occlusion, and we experimented with several settings using our synthetic dataset.

First, we varied the density of the tree crowns in the dataset until YOLOv7 [**?**] had difficulty in determining the tree crown count (see Tab. 1).

Second, we used the graphics rendering engine to determine the ground truth optical flow as a virtual UAV flies over the same dense synthetic forest. We extended YOLOv7 [**?**] to include this flow data as an indicator of changing self-occlusion patterns. Specifically, we incorporate flow as additional channels to the input.

We found that different optical flow thresholds led to improved count accuracies. Hence, Tab. 1 reports the counting performance using flow when not thresholded, thresholded by the median and the mean. The incorporation of the optical flow improves the performance by 33%. However, the counting performance of YOLOv7 remains unsatisfactory.

To further improve performance, we developed our method to include pixel content features, shape features and changed the underlying model to the graph-based approach described in Sec. 3 – this produced the highest count accuracy of 90.7%, as seen in Tab. 1. In contrast, the next best model, YOLOv7+Flow(mean), achieves a count accuracy of only 69.3%.

Quantitative Results. We compared the count accuracy

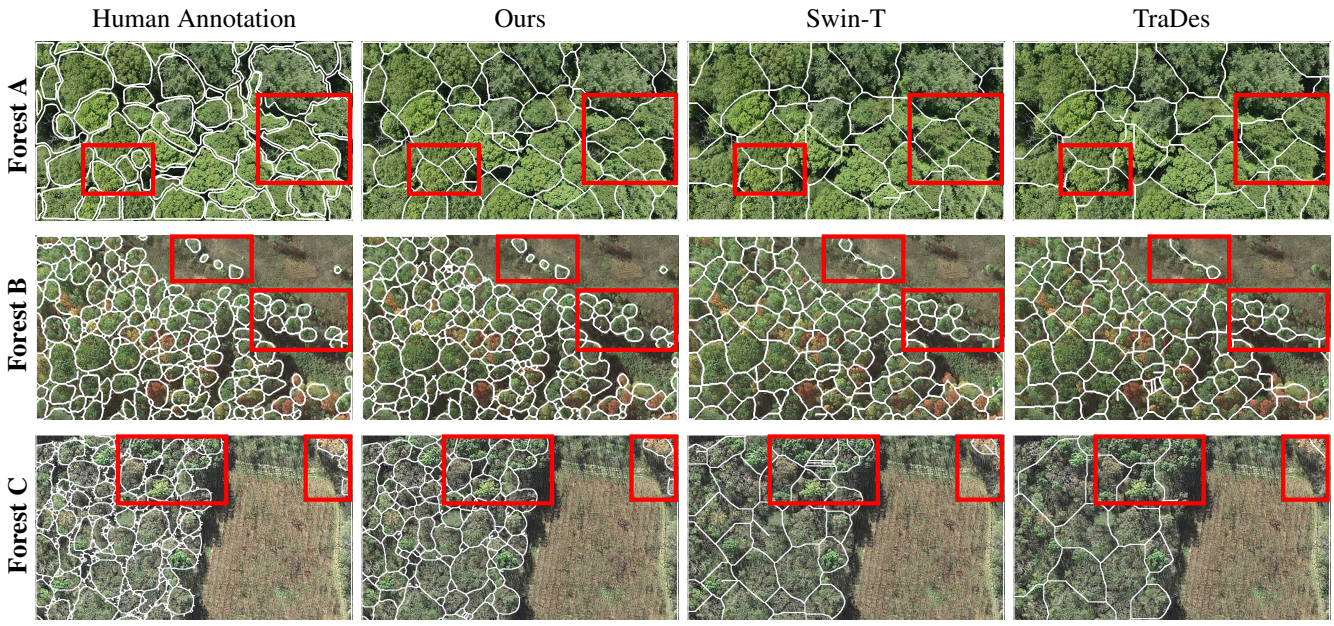

Figure 7. **Qualitative comparisons.** Comparisons of our approach vs. Swin-T and TraDes. We observe that our approach produces instance segmentation masks that more closely match the human annotation. Comparisons to more baselines are illustrated in the Appendix.

of our method using the synthetic dataset to our baselines in Tab. 3 (first three columns). We observe that Mask-RCNN (Swin-T) is the best-performing baseline with a count accuracy of 82.9%, followed by TraDeS with 80.1%, BoundaryFormer with 77.2%, and Mask-RCNN (ResNet) with 63.8%. Our method outperforms all baselines in count accuracy with 90.7%. We also compared the AP performance of using the synthetic dataset to our baselines in Tab. 2. Our method again outperforms all the baselines.

4.2. Real-World Data Experiments

We also conducted experiments on our multiple real-world forest datasets. Tab. 2 contains the AP comparisons to the baselines. Our approach outperforms all methods in all forests except for AP_{50} and AP_{70} in Forest B. Overall, our method achieves an AP of 74.5, 69.8 and 70.1 on Forest A, B, and C respectively.

Tab. 3 reports the counting accuracy comparison to our baselines. The accuracy metric is defined as $1 - MAE$ (normalized) as a percentage where MAE is the mean absolute error. Our method exceeds the performance of all prior methods across all forests. On Forest A, our method achieves a count accuracy of 91.5% compared to the next best baseline (TraDeS) of 85.9%. On Forest B, our method achieves 86.8 compared to the next best baseline (Swin-T) of 81.1%. On Forest C, our method achieves 90.2% compared to the next best baseline (Aerial Laser Scanning Model) of 86.9%.

Ablated feature	AP	AP_{50}	AP_{75}	Acc.
All Features	74.5	81.6	72.8	91.5
All — aspect ratio	67.6	72.9	68.4	82.4
All — solidity	69.2	74.1	66.2	80.1
All — self-occlusion	62.9	77.3	71.1	78.2
All — patch	51.3	56.8	63.2	71.8
All — deviation	59.8	62.2	64.1	70.9
All — area	55.1	58.8	61.6	68.3
All — neighbor sim.	57.4	61.7	59.6	63.5

Table 4. **Feature ablations.** Performance of our approach on Forest A as we ablate each feature (sorted by count accuracy). Features with lower accuracy have a higher impact on the prediction.

4.3. Qualitative Study

Beyond the quantitative results, we also qualitatively study the method's behavior. Fig. 7 shows the results of our method and baselines applied to the three real-world datasets. We generally observe that the predicted instance segmentation closely matches the ground truth annotations. On the other hand, we observe that a typical failure case for TraDes and Swin-T is that they often group multiple tree crowns into one; See Fig. 7 for regions boxed in red. –Similar illustrations of qualitative comparisons with more baselines are given in the Appendix.

Next, we study the merging behavior (Sec. 3.2). Fig. 8 shows the contour map before and after merging. We observed that our approach merged small contours into larger ones to match the tree crown boundaries more closely.

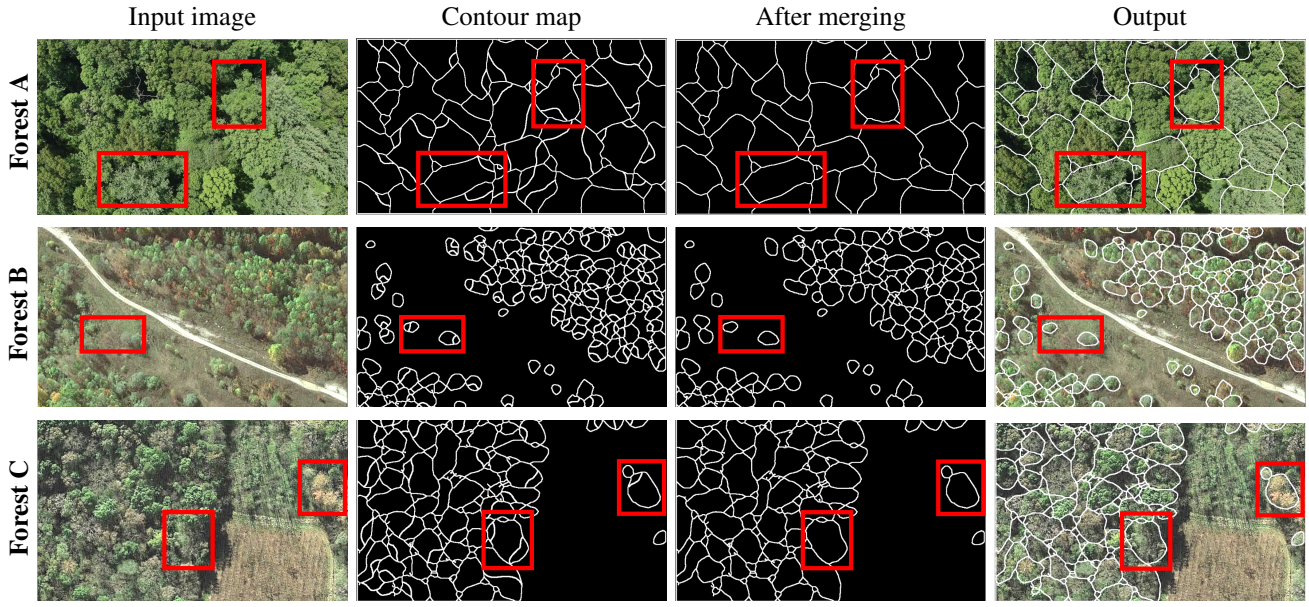

Figure 8. **Visual illustration of data flow.** We visualize the contour map before and after contour merging.

Frame Δ	1	2	3	4	5	no flow
AP ↑	74.5	72.8	68.1	65.4	64.8	62.9
Acc ↑	91.5	86.1	83.5	82.9	81.8	78.2

Table 5. **Frame-rate ablations.** The performance drop as optical flow is computed at different image intervals.

4.4. Ablation Study

To quantify the importance of features of our contour graph, we perform an ablation study using Forest A. In Tab. 4, we sort the ablations by descending in count accuracy – i.e., the feature in the top rows are the features that least impact model performance. We observe that 'neighbor similarity' impacts count accuracy the most, and the patch feature has the most impact on AP.

The other shape and self-occlusion features contribute significantly to the segmentation. The ablation shows a 13.3% increase in count accuracy when the self-occlusion features are used (driven by optical flow). Not using this feature in dense scale forests, similar to the samples of all real forest trees reported in Tab. 3, would lead to an over or underestimation of approximately 868 trees.

Moreover, to test the capability of the optical flow algorithm used [**?**], we skipped progressively more frames and recomputed the flow in each case (ablating the self-occlusion feature). The different flow values were then used and evaluated in terms of AP (see Tab. 5) and tree count accuracy. We can see in Tab. 5 the largest shift happens from skipping one frame to two frames, as the granularity of flow (using motion) halves. The subsequent drops indicate lower frame rate adversely affects the performance. Notably, even at lower frame rates, the performance metrics were higher than having no optical flow. In summary, we observe that our method generally functions well even for images captured between 30-60 fps.

5. Limitations & Conclusion

Our method is limited by the quality of the input image. For example, trees include very high-frequency details, e.g., small branchlets, that are below the resolution of the UAV cameras, which our approach cannot capture. Other image degradation factors, such as blurring caused by the wind and the UAV motion, also lead to similar challenges. Also, some shadows can be identified as trees. An obvious limitation is that our approach does not work for a single image and requires an image sequence.

In summary, we introduced a novel approach to instance segmentation and counting tree crowns in forests. Our approach uses image sequences from increasingly available UAVs. Our key contribution is leveraging the partial occlusion between successive images, shape features of the contours, and encoding these features into a contour graph that is updated between successive images. The result is a method that produces instance segmentation masks that separate the individual tree crowns. Finally, we provide synthetic and real-world datasets that can facilitate future research in tree crown instance segmentation and counting.

Acknowledgements: This research is at least partially supported by Purdue Digital Forestry Center (esp. Dr. Songlin Fei) and NSF grants 1835739 and 2106717.

Appendix: Tree Instance Segmentation with Temporal Contour Graph

A1. Additional Results

In this section, we expand upon the methods and results we presented in our main paper. We further explain the reasoning and rationales for making certain choices in our methodology.

Additional qualitative results. Fig. A1 illustrates several additional input-to-output data flows on various types of forest appearances. In particular, we selected varying shadows, lighting, shapes, colors, and distributions. This example shows the robustness of our method.

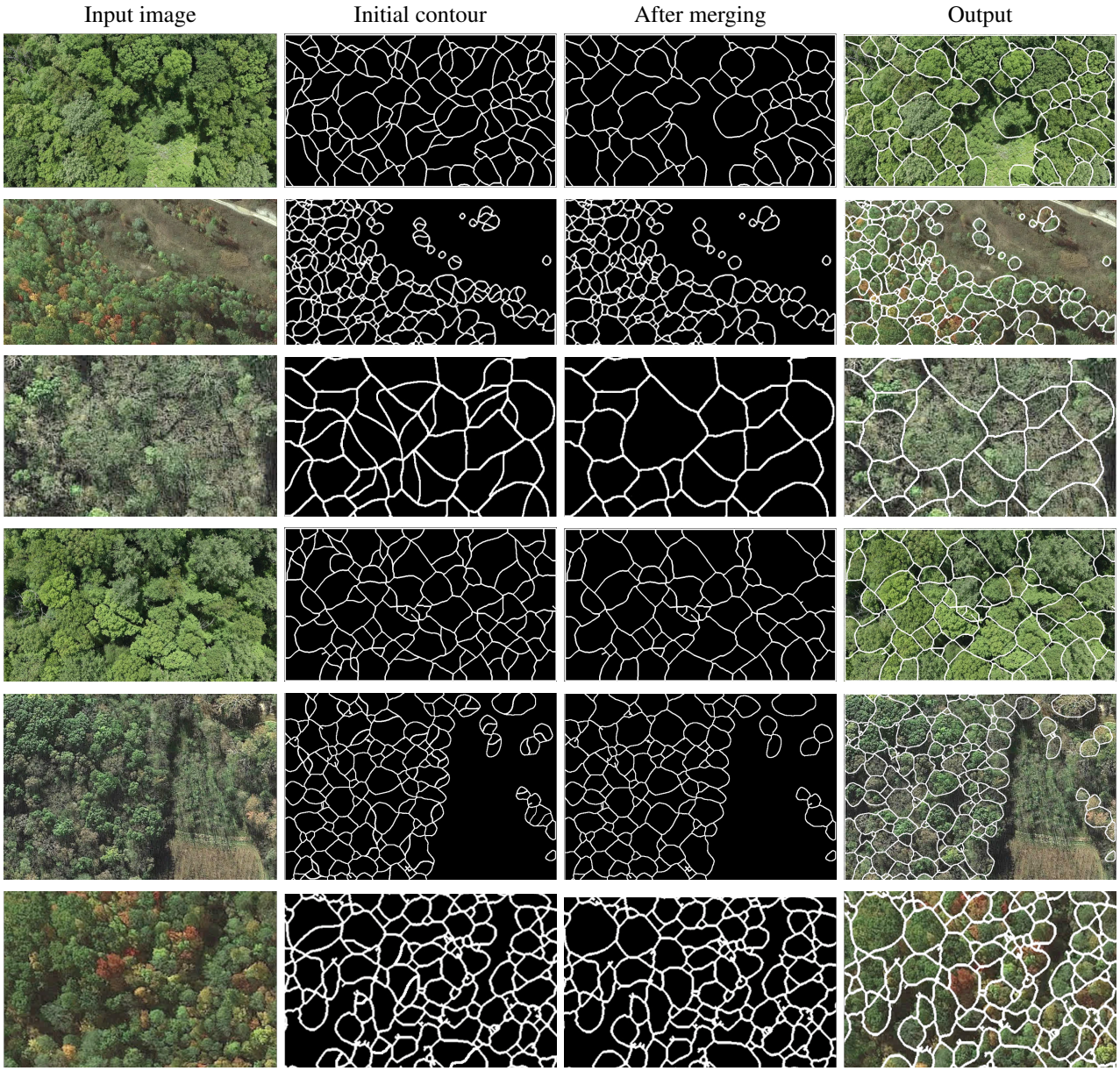

Figure A1. Additional qualitative results. We visualize the contour map before and after contour merging.

Additional visual comparisons to baselines: Here, we present more qualitative comparisons to other baselines in Figs. Fig. A2 and Fig. A3.

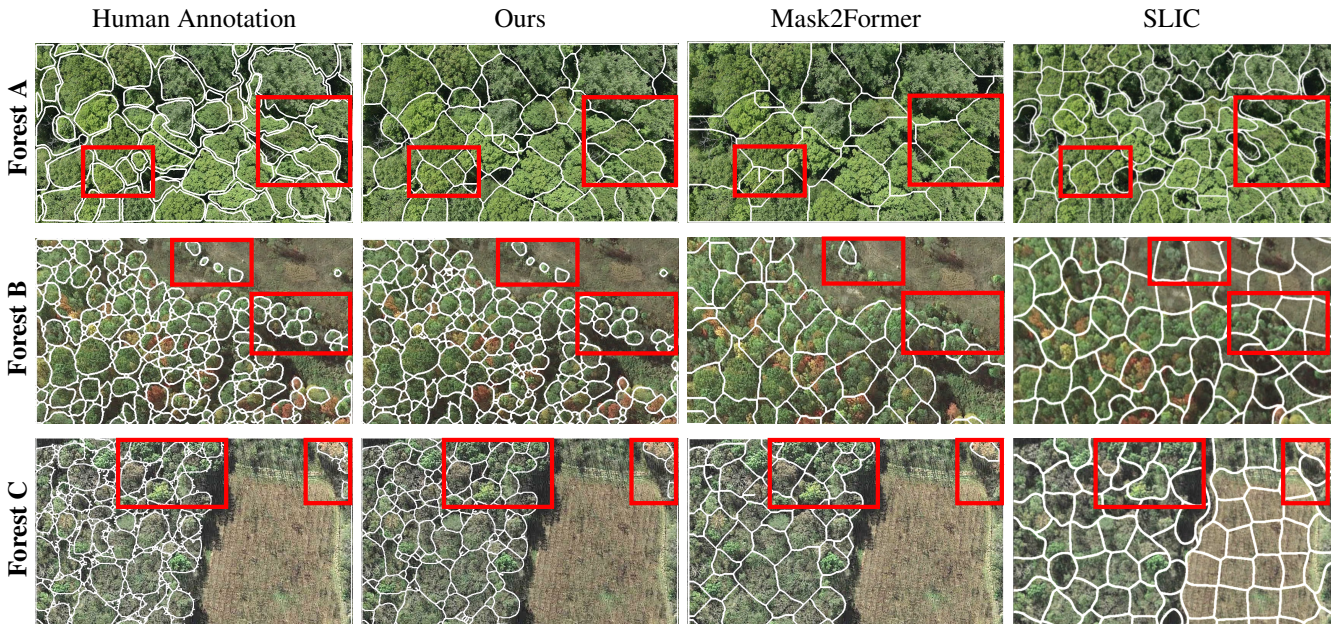

Figure A2. **Extended Qualitative Comparisons.** Comparisons of our approach vs. Mask2Former and SLIC. We observe that our approach produces instance segmentation masks that more closely match the human annotation.

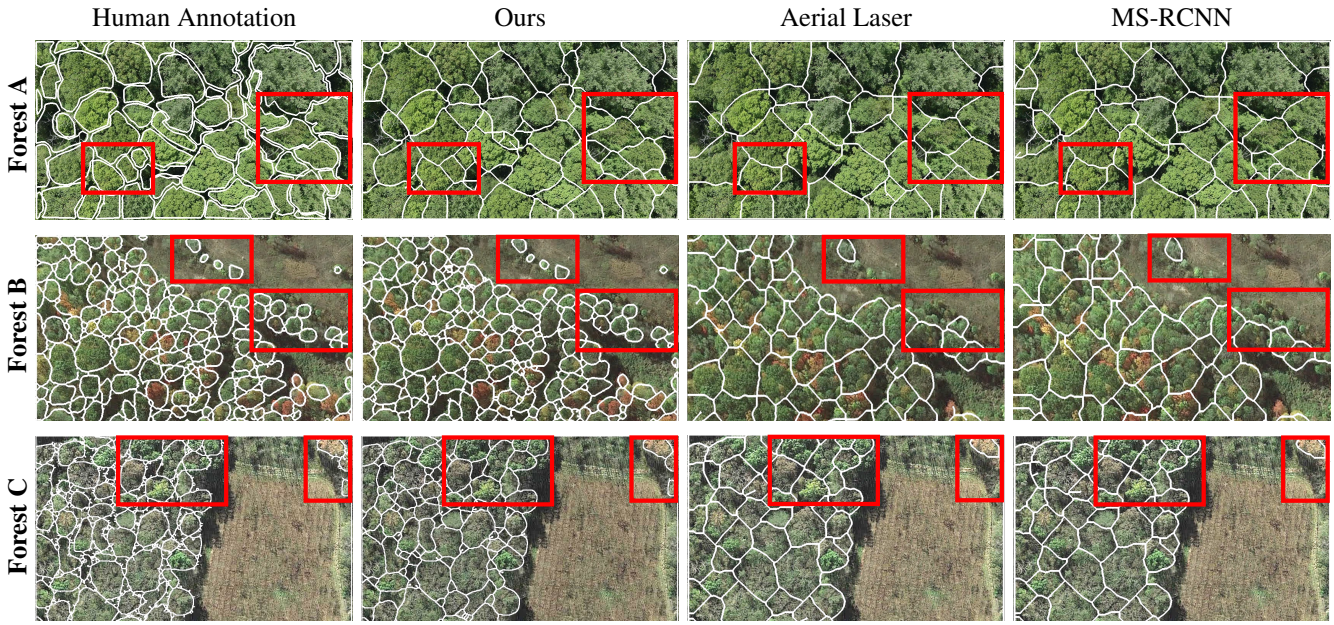

Figure A3. **More Qualitative comparisons.** Comparisons of our approach vs. Aerial Laser Model and Mask Scoring RCNN. We observe that our approach produces instance segmentation masks that more closely match the human annotation.

Comprehensive ablation. To offer a more detailed ablation on the effect of out-of-distribution data, we conduct ablations and report quantitative metrics for Forest B and Forest C (in addition to Forest A, which was presented in the main paper). In addition, we show the (very low) performance of just using the initial contours as determined by the edge detection network. Tab. A1 presents the aforementioned metrics.

Justification for Guo-Hall skeletonization [?]. Our method uses Guo Hall (GH) skeletonization [?] to convert and simplify grayscale contours to binary maps. We illustrate the effects of using GH in Fig. A4, and the resulting segmentation

Ablated feature	Forest A				Forest B				Forest C			
	AP ↑	AP_{50} ↑	AP_{75} ↑	Acc. ↑	AP ↑	AP_{50} ↑	AP_{75} ↑	Acc. ↑	AP ↑	AP_{50} ↑	AP_{75} ↑	Acc. ↑
All Features	74.5	81.6	72.8	91.5	69.8	76.2	71.5	86.8	70.1	75.4	72.5	90.2
All — aspect ratio	67.6	72.9	68.4	82.4	55.3	60.7	57.5	62.2	57.5	62.2	57.7	56.1
All — solidity	69.2	74.1	66.2	80.1	63.5	75.1	70.2	78.8	65.9	73.6	65.9	71.3
All — self-occlusion	62.9	77.3	71.1	78.2	56.2	58.4	52.5	62.7	56.5	61.8	60.2	66.5
All — patch	51.3	56.8	63.2	71.8	60.2	57.9	61.1	65.9	57.2	54.5	51.9	59.2
All — deviation	59.8	62.2	64.1	70.9	61.3	64.6	62.1	60.4	50.5	62.9	61.4	65.3
All — area	55.1	58.8	61.6	68.3	51.1	52.5	44.6	54.5	54.7	57.2	56.5	59.4
All — neighbor sim.	57.4	61.7	59.6	63.5	54.9	59.7	58.9	59.9	68.6	72.2	70.7	79.5
All — Guo-Hall	32.6	39.2	30.8	41.7	28.8	36.3	31.6	44.2	30.7	38.1	34.7	42.2
Initial Contour	25.7	29.1	27.3	38.5	21.6	22.5	21.8	31.3	28.4	34.8	29.7	36.9

Table A1. Ablation analysis of all forest datasets: Performance of our approach on real-world forest datasets as we ablate each feature. Also, we show performance when not using Guo-Hall skeletonization and also when directly using the initial contours. The table is sorted on count accuracy of Forest A to stay consistent with base paper.

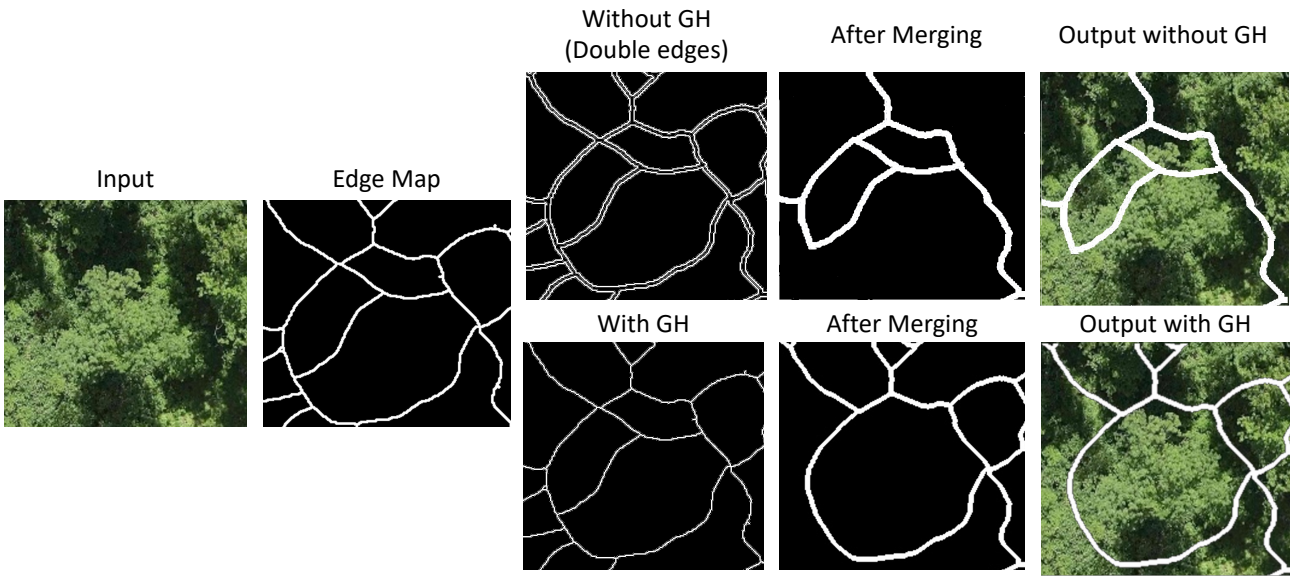

Figure A4. Impact of Guo-Hall Skeletonization (GH) [**?**]. We visualize, with and without Guo-Hall, the resulting intermediates throughout the proposed method.

is different as can be seen in the top and bottom rows. The initial edge detector generates edges with doubled contours. When constructing the contour graph, without GH, we would potentially receive two instead of one contour for each side (see Fig. A4). This results in double edges in the edge maps and leads to incorrect connectivity between the nodes. This leads to incorrect contour merging. Such effect is also shown quantitatively in Tab. A1.

Hyperparameter tuning. We used 30×30 pixels for each node's representative patch. We selected this dimension through a hyperparameter tuning experiment using several choices of patch sizes as shown in Tab. A2. The resolution of 30×30 was chosen because it outperformed both larger and smaller patches. Using larger patches would pick up pixels from neighboring nodes that are undesirable at this point of the pipeline. Graph Fig. A5 shows the values of AP and Acc. with varying patch sizes.

Patch Size (px)	AP ↑	AP_{50} ↑	AP_{75} ↑	Acc. ↑
10 x 10	52.1	61.8	57.9	74.4
20 x 20	61.5	74.8	68.1	87.2
30 x 30	**68.9**	**77.3**	**71.1**	**90.1**
40 x 40	64.2	75.9	70.3	88.3

Table A2. Hyperparameter tuning for patch size on Forest A.

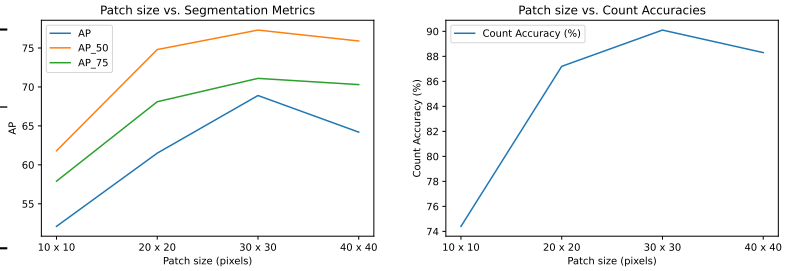

Figure A5. Plots showing AP performance and count accuracies vs. patch sizes.

A2. Implementation Details

Compute resource. All our experiments and training were done on a machine with AMD EPYC 7302 16-Core Processor with 3 GHz clock speed, and 128 GB RAM. Our model is trained with a single NVIDIA RTX 3090 GPU.

Implementation details. Our work is implemented in Python using PyTorch. We utilize Pytorch Geometric [**?**] for graph processing and OpenCV for image processing. The deep contour/edge map generation is generated using a deep edge detection network named Pixel Difference Networks for Efficient Edge Detection (PIDINet) [**?**] from repository. We retrained the edge detection network with our annotated dataset. It took approximately 22 hours to train the network with a base learning rate of 0.01 and weight decay of 0.0005 for 200 epochs. The optimizer used was ADAM [**?**]. Then, the network could generate the initial edge maps necessary for our pipeline.

We use a GCN [**?**] to perform message passing of node embedding, where input features capture each node's accompanying neighborhood information. In our GCN implementation, we set our walk length to two, which corresponds to K in the original GCN paper[**?**]. In this way, we aggregate information from at most two steps (i.e., neighbor of neighbors) away for each node embedding. We used a base learning rate of 0.1 and weight decay of 0.005 with ADAM [**?**] and trained for 200 epochs. It took approximately 15 hours to process and train the model on our dataset.

Finally, the MLP used for edge classification is trained with a base learning rate of 0.001 and momentum of 0.9. using ADAM [**?**]. It took under two hours to train the MLP classifier for 500 iterations with early stopping at 10 steps.

Baseline implementation details. We document how we train/use the nine baseline models on our dataset.

*Mask-RCNN [**?**].* We use the publicly available implementation Matterport official GitHub repository (which was acknowledged by the original authors). We used the ResNet-101 backbone and retrained on our labeled dataset. We train the modeling using SGD with a learning rate of 0.001, a momentum of 0.9, and a weight decay of 0.0001. It took approximately four days to train 200 epochs on our dataset.

*Swin-T [**?**] backbone with Mask-RCNN [**?**].* We use the official branch in Github from the author's affiliation – Microsoft Research, which was specific for instance segmentation. We train the model using a base learning rate of 0.001 and weight decay of 0.0001 using ADAM [**?**]. This baseline was trained for 200 epochs, which takes approximately three days.

*TraDes [**?**].* We use the official repository on GitHub. We used the MoT17 [**?**] pre-trained weights and evaluated our validation dataset to confirm consistency. We note that MoT17 also had trees and other vegetation classes subjectively similar to ours.

*BoundaryFormer [**?**].* We use the official implementation on GitHub. We retrained the model using our own dataset. It took approximately 2 days to finish the training for 200 epochs. We set the base learning rate is set to 0.02 and use the ADAM [**?**] optimizer.

*Mask Scoring RCNN [**?**].* We use the official implementation on GitHub. We retrained the model using our own dataset in COCO format. It took approximately one day to finish the training for 100 epochs. We set the base learning rate to 0.02 with weight decay of 0.0001 and use the SGD optimizer.

*Mask2Former [**?**].* We use the official implementation on GitHub. We retrained the model using our own dataset in COCO format. It took approximately eight hours to finish the training for 100 epochs. We set the base learning rate to 0.0001, weight decay of 0.05 and use the ADAM [**?**] optimizer.

*Aerial Laser Model [**?**].* It is an improved parameterized version of YOLOv4. We make the instructed changes to the PyTorch implementation on GitHub. We retrained the model using our own dataset. It took approximately 12 hours to finish the training for 200 epochs. We set the base learning rate to 0.0013, weight decay of 0.0005, the momentum of 0.949 and use the ADAM [**?**] optimizer.

*SLIC (Superpixel) [**?**].* As this is a traditional algorithm as opposed to a deep learning model, we used the SLIC implementation from the Python Scikit-Learn package.

*OCISIS [**?**].* We use the official implementation on GitHub. We retrained the model using our own dataset in COCO format. It took approximately two days to finish the training for 200 epochs. We set the base learning rate to 0.01, weight decay of 0.0001, and momentum of 0.9 and used the SGD optimizer.

[10pt,twocolumn,letterpaper]article [rebuttal]cvpr
graphicx amsmath amssymb booktabs multirow contour [normalem]ulem soul
0.8pt
[pagebackref,breaklinks,colorlinks,bookmarks=false]hyperref
xcolor xspace
symbols
Rebuttal: Tree Instance Segmentation using Temporal Structured Images

We thank the reviewers for their constructive feedback. We address individual questions from each reviewer and include the requested new comparisons. *Our method outperforms all newly requested baselines across all datasets.*

To Reviewer Qvio

Q1. Quality of human annotations. Experienced forest research collaborators prepared our validation data by physically surveying the forests and by using images with green-leaf, fall-colored, and leaf-off (only branches and trunks).

Q2. Comparision with traditional superpixel segmentation. We added an experiment using SLIC [**?**] in Tab. A3. SLIC achieves less than one-third of the AP of our method.

Q3. Comparision with tree specific segmentation. We are not aware of dense tree-specific methods that are able to address our goal. However, one similar recent work in a top venue is [**?**], which focuses on bounding boxes and not segmentation masks. We provide additional comparisons (suggested by Reviewer MpxE) in Tabs. A3 and A4. Our method achieves better performance than the compared methods.

Q4. Novelty. **First**, our work addresses the challenging problem of densely packed trees. Prior works focus on counting and not instance segmentation for dense trees (see ℓ.96). **Second**, our approach computes an over-segmentation and then performs a refined instance segmentation posed as an edge classification problem modeled via graph nets. Our approach differs from the generic instance segmentation pipelines. **Third,** we discuss the relation, and how we differ, to prior works in Sec. 2.

To Reviewer MpxE

Q5. Requested additional baselines. Tab. A3 compares to the requested additional baselines [**? ? ? ? ? ?**]. Our method across all datasets *at least* outperforms the next-best model [**?**] by 25% in MAE and by 5% in AP for segmentation and [**?**] by 26% in MAE for counting. Average improvement over other methods 49.6% (AP) and 50.2% (MAE).

Q6. Evaluation metric for counting (Why not MAE?). Sorry for the confusion, we briefly explained Acc. in ℓ.584-586. For clarity, Acc. is computed as $1-\text{MAE}$ (normalized) where $\text{MAE} = \frac{1}{N}\sum_{n=1}^{N}|\hat{y}_n - y_n|$, with $\hat{y}_n$, y_n, N denoting the tree prediction (binary), ground-truth (binary) and total ground-truth tree count, respectively. We use Acc. as higher is better, i.e., its direction is consistent with AP. See AP/MAE in Tab. A3.

Q7. Interest to the community. **Algorithmic interest:** Counting tightly packed moving structures, e.g., trees, has not been done extensively before, and we believe this may stimulate future works in this direction. Please also see **Q4** where we summarize the paper's novelty. **Societal interest:** This is directly linked to the theory of human-induced climate changes and also to ecological balance and environmental health (as suggested by Reviewer rjNH).

Table A3. Additional benchmarks against crowd, object, tree/planet segmentation approaches (**AP** ↑ **/ MAE** ↓)

Method/Dataset	Synth.	Fors. A	Fors. B	Fors. C
Ours	**74.6/9.3**	**74.5/8.5**	**69.8/13.2**	**70.1/9.8**
Mask2Former [**?**]	62.4/36.2	59.7/26.4	64.1/21.6	61.9/22.6
MS-RCNN[**?**]	66.2/31.6	64.8/18.4	66.4/19.1	64.2/19.5
OCISIS[**?**]	59.1/15.2	55.7/14.9	60.8/19.5	60.2/16.1
Aerial Laser[**?**]	71.1/14.7	70.4/15.3	65.1/19.7	65.3/13.1
SLIC[**?**]	23.7/57.9	20.7/54.1	22.5/61.6	38.4/62.1

Table A4. Additional benchmarks against object and crowd counting models - our method performs best (**MAE** ↓)

Method/Dataset	Synth.	Fors. A	Fors. B	Fors. C
Ours	**9.3**	**8.5**	**13.2**	**9.8**
DMCC[**?**]	13.8	14.3	17.8	16.4
OCISIS[**?**]	15.2	14.9	19.5	16.1

To Reviewer rjNH

Q8. Comparision with crowd detection. We report the requested comparison with [**?**] in Tab. A4 (and other new comparisons in Tab. A3, see **Q5**). We observe, among the crowd-based models, ours *at least* outperformed the next-best models in segmentation (AP) [**?**] by 5%, and the next-best model in counting (MAE) [**?**] by 26% across all datasets.

Q9. Additional dataset details. Dataset details are in Sec 3.3 (ℓ.519-524), where we described the captured areas, framerates, speed, FOV, and resolution. We purposefully captured UAV data at three heights (80, 100, and 120 meters) and three forests of varying species distribution. The trees have a range of heights (Forest A: 3.4-15.6 meters, Forest B: 18-25 meters; Forest C: 10-33 meters). We choose Forest B and C to evaluate the out-of-distribution setting, as written in Sec. 3.3. When anonymity is lifted, we will report actual forest names and full-length UAV videos; example videos are already included in the supplement. Finally, we will update the paper and appendix with more dataset details.