Training Deep Learning Algorithms on Synthetic Forest Images for Tree Detection

Vincent Grondin[1 2], François Pomerleau[1] and Philippe Giguère[1]

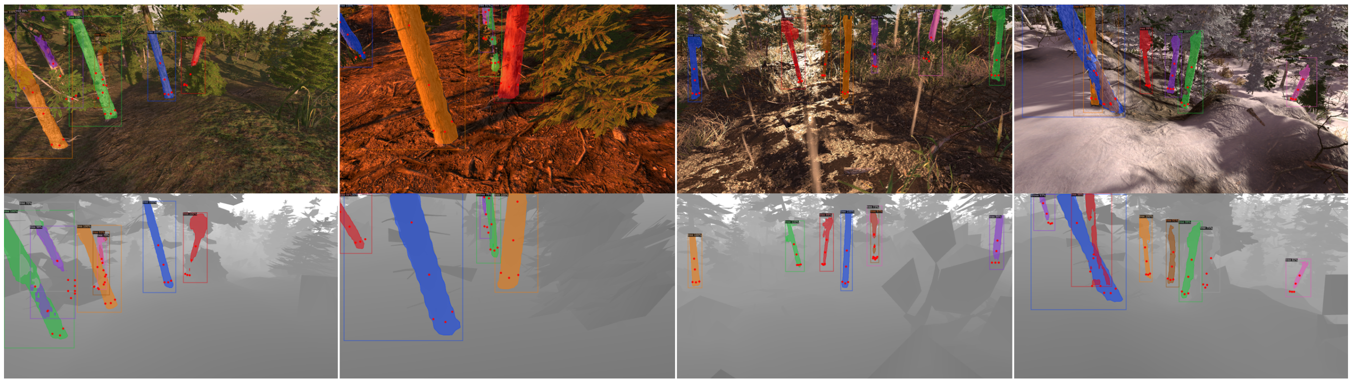

Fig. 1: Predictions by `ResNeXt-101` on synthetic RGB images (top row) and by `ResNet-101` on depth images (bottom row). The models achieve qualitatively good results under challenging simulated conditions such as light variation, occlusion, instances overlapping and weather effects.

***Abstract*— Vision-based segmentation in forested environments is a key functionality for autonomous forestry operations such as tree felling and forwarding. Deep learning algorithms demonstrate promising results to perform visual tasks such as object detection. However, the supervised learning process of these algorithms requires annotations from a large diversity of images. In this work, we propose to use simulated forest environments to automatically generate 43 k realistic synthetic images with pixel-level annotations, and use it to train deep learning algorithms for tree detection. This allows us to address the following questions: i) what kind of performance should we expect from deep learning in harsh synthetic forest environments, ii) which annotations are the most important for training, and iii) what modality should be used between RGB and depth. We also report the promising transfer learning capability of features learned on our synthetic dataset by directly predicting bounding box, segmentation masks and keypoints on real images.**

I. Introduction

Deep learning gained much attention in the field of forestry as it can implement knowledge into machines to tackle problems such as tree detection or tree health/species classification [1]. However, deep learning is a data centric approach that needs a sufficient amount of annotated images to learn distinctive object features. Creating an image dataset is a cumbersome process requiring a great deal of time and human resources, especially for pixel-level annotations. Accordingly, few datasets specific to forestry exist, and this limits deep learning applications, as well as task automation requiring high-level cognition.

In order to avoid hand-annotation and include as many realistic conditions as possible in images, we propose to fill the data gap by creating a large dataset of synthetic images containing over 43 k images, which we name the SynthTree43k dataset. Based on this dataset, we train Mask R-CNN [2], the most commonly-used model for instance segmentation [1], and measure its performances for tree detection and segmentation. Because our simulator allows for quick annotation, we also experiment with keypoint detection to provide information about tree diameter, inclination and felling cut location.

Even though the detection performances obtained on SynthTree43k will not directly transfer to real world images because of the reality gap, a result analysis can guide us towards building an optimal real dataset. Notably, synthetic datasets can be used to evaluate preliminary prototypes [3], and sometimes they can improve detection performance when combined with real-world datasets [3], [4]. In that sense, we shed light on which annotations are the most impactful on learning, and if adding the depth modality in the dataset is pertinent. Lastly, we demonstrate the reality gap by qualitatively testing the model on real images, showing transfer learning potential.

II. Related Work

Deep learning for tree detection in forestry has demonstrated success on relatively small real image datasets. For instance, [5] implement a U-Net architecture to perform tree specie classification, detection, segmentation and stock volume estimation on trees. When trained on their (private) dataset of 3 k images, they achieve 97.25 % precision and 95.68 % recall rates. Similarly, [6] uses a mix of visible and thermal images to create a dataset of 2895 images extracted from video sequences, and solely include bounding box annotations. They trained five different one-shot detectors

*This work is supported by the FORAC Consortium.

[1] Department of Computer Science and Software Engineering at Laval University, Canada

[2] Corresponding author: `vincent.grondin.2@ulaval.ca`

on their dataset and achieved 89.84 % precision, and 89.37 % F1-score.

We believe these aforementioned methods could benefit from training on synthetic images. The Virtual KITTI dataset [3] is one of the first to explore this approach to train and evaluate models for autonomous driving applications. By recreating real-world videos with a game engine, they generate synthetic data comparable to real data. The models trained on their virtual dataset show that the gap between real and virtual data is small, and it can substitute for data gaps in multi-object tracking. Meanwhile, [7] explore object detection using a synthetic dataset for autonomous driving, and they report that training models on realistically rendered images could produce good segmentations by themselves on real datasets while dramatically increasing accuracy when combined with real data. Quantitatively, they improved per-class accuracy by more than 10 points (and in some cases, as far as 18.3 points).

Although synthetic datasets cannot completely replace real world data, multiple works demonstrate that it is a cost-effective alternative that offers good transferability [3], [4], [7]. Therefore, developing synthetic forest datasets will potentially improve the current state of tree perception methods in forestry.

III. Methodology

In this section, we detail how SynthTree43k was created. Then, we describe the deep learning architecture and backbones, as well as the training details.

A. Simulator and Dataset

SynthTree43k is generated by employing the Unity[1] game engine to render realistic virtual forests. This virtual world generator can be configured via Gaia[2] to procedurally terra-form the landscape, texture the terrain and spawn objects. In this simulation, the forest density is controlled through various spawn rules such as altitude, terrain slope and the number of neighbouring objects in a given area.

The forest is populated with realistic tree models from Nature Manufacture[3]. In order to extend visual variability, texture on the six tree models is modified to create 17 new, distinctive tree models. Other object models from Nature Manufacture are also included in scenes such as grass, stumps, scrubs and branches.

For additional realism and variety, meteorological conditions are also simulated in this virtual world. We simulate snow using snow texture, and wet effect using decals. Particle systems are employed to recreate snowflakes, raindrops or fog effects. To simulate different moments of the day, we adjust illumination to morning, daylight, evening and dusk. The object shadows adapt to illumination cast on the scene.

From each generated scene, we add between 200-1000 images to the dataset. Each image includes bounding box, segmentation mask and keypoint annotations. The pipeline annotates trees within a 10 m radius from the camera, which corresponds to the reach of a harvester [8]. Five keypoints are assigned per trees to capture the essential information that an autonomous tree felling system would need: the felling cut location, diameter and inclination.

Collectively, this pipeline can generate an unlimited amount of synthetic images with an annotation speed of approximately 20 frames/minute, for we consider RGB and depth images as one frame. Overall, SynthTree43k contains over 43 k RGB and depth images, and over 162 k annotated trees.

B. Models

The Mask R-CNN architecture is composed of *i)* a convolutional feature extraction backbone, *ii)* a Region Proposal Network (RPN), and *iii)* prediction heads. The original Mask R-CNN network is slightly adapted for our tree detection problem, in that, an optional keypoint branch is added to the prediction head. Therefore, it can be used for classification, bounding box regression, segmentation, and keypoint prediction.

Predictions are made via a two-stage process. In the first stage, the RPN proposes regions of interests (RoI) from the feature maps of the backbone. These correspond to a region that potentially contains a tree. The generation of a RoI follows along the default nine box anchors, corresponding to three area-scales (8, 16 and 32) and three aspect ratios (0.5, 1.0, and 2.0). In our experiment, we employ three different backbone architectures: ResNet-50, ResNet-101 [9], and ResNeXt-101 [10]. We use ResNet backbone for feature extraction as it gives excellent gains in both accuracy and speed, which we use as a baseline for our results. When comparing the 50-layer to its 101-layer counterpart, there is a possibility that they perform similarly when the dataset is small [2], yet we expect that using SynthTree43k will be enough to demonstrate that the 101-layer can outperform the 50-layer. In regards to ResNeXt, it introduces a cardinality

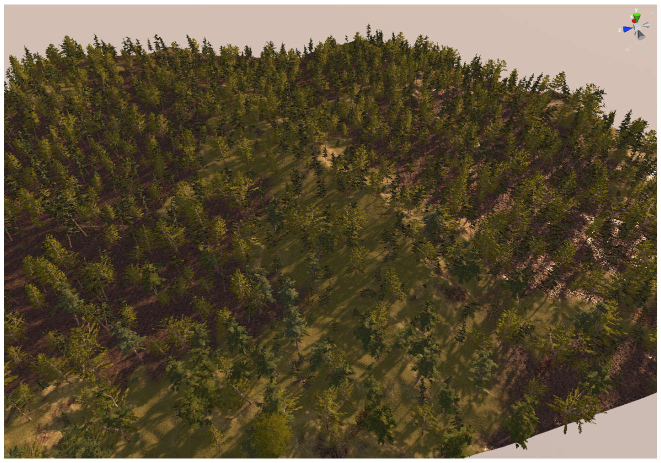
Fig. 3: A general view of a simulated forest environment. In this scene, three terrain textures are used to simulate moss, roots and mud conditions. The tree models are fir and beech, accompanied by scrubs, branches, grass and stomps under a morning light effect.

[1]https://unity.com
[2]https://assetstore.unity.com/packages/tools/terrain/gaia-2-terrain-scene-generator-42618
[3]https://naturemanufacture.com/

hyper-parameter, which is the number of independent paths, providing a way to adjust the model capacity without going deeper or wider. More details about backbone parameters are provided in Table I

TABLE I: Backbone parameters. The number of learnable parameters (#Params), computational complexity (GFLOPs) and frames per second (FPS) at inference time on 800×800 images.

Backbone	#Params	GFLOPs	FPS
`ResNet-50-FPN`	25.6 M	3.86	18
`ResNet-101-FPN`	44.7 M	7.58	15
`ResNeXt-101-FPN`	44 M	7.99	10

Subsequently, RoIAlign [2] uses bilinear interpolation to map the feature maps of the backbone into a 7×7 input feature map within each RoI area. Features from each RoI then go through the network head to simultaneously predict the class, box offset, binary segmentation mask, and an optional binary mask for each keypoint.

C. Training Details

We use Detectron2 [11] implementations of Mask R-CNN. It has been shown that pre-training helps regularize models [12], and facilitate transfer learning to a target domain [13]. Therefore, the Mask R-CNN models employed in our experiments are pre-trained on the COCO Person Keypoint dataset [14], which is a large-scale dataset containing more than 200 k images and 250 k person instances labeled with 17 keypoints per instance. Before training or fine-tuning the models, the first two convolutional layers of the backbone are frozen. The hardware for model training and testing is an NVIDIA RTX-3090-24GB GPU and an Intel Core i9-10900KF CPU.

To train the model, SynthTree43k is split into three subsets: 40 k in the train set, 1 k for the validation set and 2 k in the test set. The model learns from the train set by using an stochastic gradient descent (SGD) optimizer with a momentum of 0.9, and a weight decay of 0.0005. During training, we improve model generalization, and reduce dataset overfitting by employing data augmentation techniques such as image resizing, horizontal flipping, sheering, saturation, rotation, and cropping. No data augmentations are used at validation and test time. Model overfitting is monitored via the validation set, which is also used for early stopping.

Depth images are gray scales of 8-bit 1-channel. At train time, they are converted to 8-bit 3-channel to fit the RGB format from pre-trained models. In our case, a single image channel could be possible, but it would require randomly initiated models, which takes an enormous amount of pre-training time for backbones such as `ResNet` and `ResNeXt`.

Hyperparameter optimization is conducted for `ResNet-50-FPN` only, and these hyperparameters are used for every model. We use early stopping based on the highest validation set Average Precision (AP) to determine when to stop training.

IV. Experimental Results

We base our performance evaluation on the standard COCO metrics for each task, AP^{bb} and AP^{mask}, we train Mask R-CNN on our synthetic forest images and compare detection performance between the three backbones and RGB/Depth modality. We also conduct an analysis of keypoint prediction by measuring the pixel error of each predicted keypoint. Lastly, we test detection on real images, qualitatively demonstrating the reality gap between synthetic and real images.

A. Tree Detection and Segmentation

Six models, corresponding to all combinations between the three different backbones and two modalities, are trained and tested on SynthTree43k. From Table II, we observe that all models trained on the depth modality outperform models trained on the RGB modality. In fact, the detection task based on the depth modality improves AP^{bb} by 9.49 % on average, even though all of the models were pre-trained on COCO Person — an RGB dataset. This suggests two things: 1) that the depth modality helps networks reject trees located further than our 10 m annotation threshold, and 2) depth images are possibly easier to interpret. In regard to the segmentation task, very little gain is obtained by using depth images instead of RGB, and in the case of `ResNeXt-101` it decreases. Surprisingly, the `ResNeXt` architecture has more trouble transferring to depth images compared to the `ResNet` architecture, which make `ResNet-101` the best backbone for depth.

On RGB images, we achieve the best detection results using `ResNeXt-101`. This is similar to previous research [10], [15], and it is a result of the cardinality used in `ResNeXt` as it is more effective than going deeper or wider when the model capacity is increased. The predictions on synthetic images can be observed in Figure 1.

TABLE II: Results for models trained and tested on SynthTree43k. All models achieved better performances using the depth modality.

Backbone	Modality	AP^{bb}	AP^{mask}	AP50^{bb}	AP50^{mask}
`R-50`	RGB	55.20	31.13	87.74	69.36
	Depth	66.70	31.52	89.67	70.66
`R-101`	RGB	56.79	31.72	88.51	70.53
	Depth	**68.20**	**31.98**	**89.89**	**71.65**
`X-101`	RGB	58.34	31.77	88.91	71.07
	Depth	63.90	28.86	87.41	68.19

Table III shows that adding the mask branch consistently improves AP^{bb} and AP^{kp}. Comparatively, adding the keypoint branch reduces AP^{bb} and AP^{mask}. These findings align with [2], as they found that the keypoint branch benefits from multitask training, but it does not help the other tasks in return. Better bounding box and keypoint detections can occur by learning the features specific to segmentation. In fact, a richer and detailed understanding of image content requires pixel-level segmentation, which can play an important role in precisely delimiting the boundaries of individual trees [16].

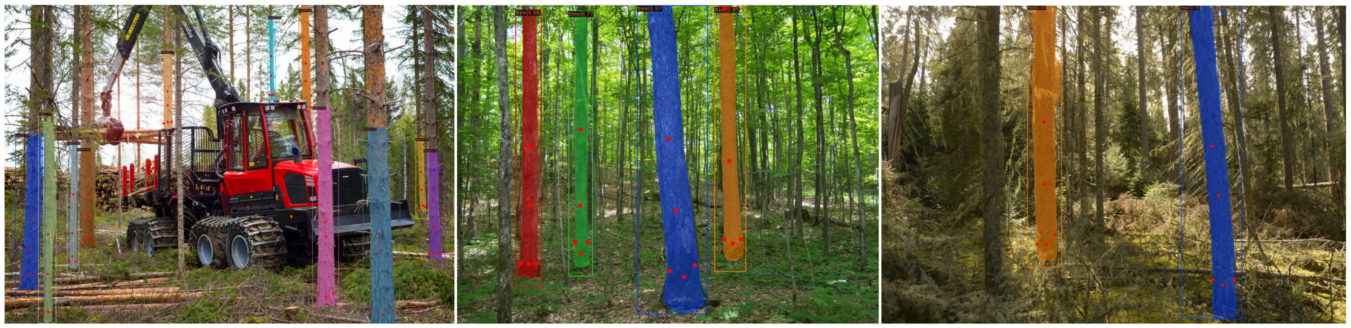

Fig. 4: Predictions on real images from `ResNeXt-101` trained only on synthetic images. We observe that even with the reality gap, the model can still detect trees with high precision, but suffers from low recall rates.

TABLE III: Impact of multi-task learning on bounding box, segmentation mask, and keypoints. Results are from `ResNeXt-101` on real RGB images.

Tasks	AP^{bb}	AP^{mask}	AP^{kp}
mask-only	**59.25**	**32.65**	-
keypoint & mask	58.34	31.77	**80.19**
keypoint-only	57.71	-	80.13

B. Keypoint Detection

A keypoint detection analysis based on error in pixels allows for meaningful and straightforward interpretations of tree-felling tasks and their error distribution. Therefore, we compute the pixel error between the ground truth and the predicted keypoint, and report the results in Figure 5.

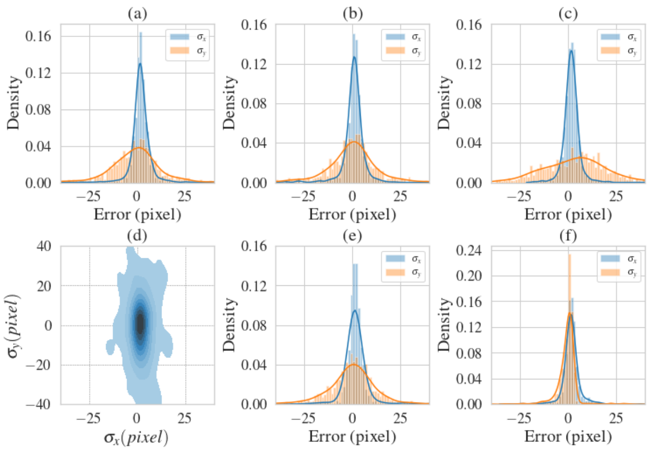

Fig. 5: Keypoint error distributions (in pixels) for our best detection model, `ResNet-101` on depth images. Blue is the horizontal error and orange is the vertical error distribution for the **(a)** felling cut keypoint, **(b)** right and **(e)** left diameter keypoint; **(c)** middle keypoint and **(f)** top keypoint. Density map of the felling cut keypoint is shown in **(d)**.

Our tree detection model achieves a mean error of 5.2 pixels when estimating tree diameters. This accuracy is promising for automation applications, depending on the distance between the tree and the camera. Since the conversion from pixel error to metric error depends on the depth, the error increases when the tree is further away, and in turn decreases accuracy.

We observe a significant difference between horizontal and vertical error, where σ_y is about three times the value of σ_x, for the felling cut, right and left diameter keypoint, and middle keypoint. We expected a larger σ_y than σ_x, because the horizontal keypoint position is either located on the side or center of the trees. In comparison, the vertical position of each keypoint is subjectively more difficult to estimate due to the inability of extracting precise vertical information. Hence, the difficulty in estimating the σ_y error greatly impacts the estimated felling cut position. If it is estimated too high on the stem, felling the tree will leave behind high stumps that are against current harvesting practices [17]. Keypoint predictions with high vertical values often occur when a dense understorey restricts the line of sight to the tree base. This causes faulty predictions to position above the understorey, which results in an inappropriate felling cut height. In practice, a simple solution to this issue is to place the felling head on the predicted point, and roll it down to the base [17].

C. Prediction on Real Images

We test the transferability of our model on real images. Due to the lack of real image datasets for tree detection and segmentation, the models are not fine-tuned on real images. Qualitative results can be observed in Figure 4. Visually, we see that not only bounding boxes are well predicted, but segmentation masks along with keypoint predictions are also transferred successfully. The model seems to be more precise than accurate, which indicates that it is unable to detect trees that are too different from the ones trained on in the synthetic dataset. Adding different tree models to our simulation could help generalize to the real world. Virtual pre-training is a promising practice given the current data gap in forestry compared to other domains, like autonomous driving or industrial automation.

V. Conclusion and Future Works

In short, we explored the use of synthetic images to train deep learning algorithms for tree detection. We provide quantitative experimental evidence suggesting that the segmentation task is important and helps to improve both bounding box and keypoint predictions. Therefore, the creation of a real image dataset in forestry should include these annotations. We also show that the depth modality significantly outperforms the RGB modality in the synthetic world. Finally, we qualitatively demonstrate that direct transfer to real world images suffer from low accuracy, while the precision is relatively good. Models publicly available[4].

In future works, we plan to evaluate tree detection performances on a real images dataset and assess its possible use in forestry related operations.

References

[1] Y. Diez, S. Kentsch, M. Fukuda, M. L. L. Caceres, K. Moritake, and M. Cabezas, "Deep learning in forestry using uav-acquired rgb data: A practical review," *Remote Sensing*, vol. 13, no. 14, p. 2837, 2021.

[2] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *IEEE International Conference on Computer Vision*, 2017, pp. 2961–2969.

[3] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig, "Virtual worlds as proxy for multi-object tracking analysis," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4340–4349.

[4] C. R. de Souza12, A. Gaidon, Y. Cabon, and A. M. López, "Procedural generation of videos to train deep action recognition networks," 2017.

[5] J. Liu, X. Wang, and T. Wang, "Classification of tree species and stock volume estimation in ground forest images using deep learning," *Computers and Electronics in Agriculture*, vol. 166, p. 105 012, 2019.

[6] D. Q. da Silva, F. N. Dos Santos, A. J. Sousa, and V. Filipe, "Visible and thermal image-based trunk detection with deep learning for forestry mobile robotics," *Journal of Imaging*, vol. 7, no. 9, p. 176, 2021.

[7] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3234–3243.

[8] O. Lindroos, O. Ringdahl, P. La Hera, P. Hohnloser, and T. H. Hellström, "Estimating the position of the harvester head–a key step towards the precision forestry of the future?" *Croatian Journal of Forest Engineering: Journal for Theory and Application of Forestry Engineering*, vol. 36, no. 2, pp. 147–164, 2015.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[10] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1492–1500.

[11] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, *Detectron2*, `https://github.com/facebookresearch/detectron2`, 2019.

[12] D. Erhan, A. Courville, Y. Bengio, and P. Vincent, "Why does unsupervised pre-training help deep learning?" In *International conference on artificial intelligence and statistics*, JMLR Workshop, 2010, pp. 201–208.

[13] D. Mahajan, R. Girshick, V. Ramanathan, *et al.*, "Exploring the limits of weakly supervised pretraining," in *European Conference on Computer Vision (ECCV)*, 2018, pp. 181–196.

[14] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, "Microsoft COCO: Common objects in context," in *IEEE European Conference on Computer Vision (ECCV)*, Springer, 2014, pp. 740–755.

[15] M. Wu, H. Yue, J. Wang, *et al.*, "Object detection based on rgc mask r-cnn," *IET Image Processing*, vol. 14, no. 8, pp. 1502–1508, 2020.

[16] L. Liu, W. Ouyang, X. Wang, *et al.*, "Deep learning for generic object detection: A survey," *International Journal of Computer Vision*, vol. 128, no. 2, pp. 261–318, 2020.

[17] D. Ireland and G. Kerr, "Ccf harvesting method development: Harvester head visibility," 2008.

[4]https://github.com/norlab-ulaval/PercepTreeV1