Biomass Estimation:
Tree Detection & Segmentation

Adityaa Chandna, Alex Tang, Audrey Romanik,
Jesse Hoogs, Morgan McCarty, Nate Sawant,
Noah Flott, Sarah Kolasa, Sophia He

Table of Contents

Background......3
Objective......3
Models & Libraries......3
  DeepForest......3
  samgeo......9
  DetecTree......11
Benchmarking......12
  Annotated Data......12
  Annotation Process......13
  Evaluation Metrics......13
Results......14
  Deep Forest......14
  DetecTree......15
Analysis......16
  DeepForest......16
  DetecTree......17
  Comparison......17
Fine-Tuning and Training......19
  Segment Anything Model (SAM) for Geospatial Data (samgeo)......19
  DetecTree......20
  DeepForest......20
  Further Training Ideas......21
Data......21
  GeoTiff......21
  Google Maps and Tiles......21
  Northern Territory Dataset......22
Conclusion......22

Background

As part of the ever changing landscapes and dangers of climate change, estimating the above ground biomass is needed to monitor how different areas differ and change over time. To estimate the above ground biomass, the tree canopy's coverage can be detected and segmented, which should correlate with the amount of stored carbon in the vegetation.

Objective

The main goals of this report and project are to identify existing tools that can be used to help estimate tree coverage and compare their performances to each other. Recommendations will be made based on the results on how to optimize the workflow, and for the models that are capable of doing so, recommendations will be made on how to fine-tune and optimize parameters.

Models & Libraries

There are a lot of different types of computer vision libraries that are openly available to the public, but few that are specialized to detect trees. For the purpose of this project, several different models/libraries, including DeepForest, samgeo, DetecTree, PointcloudITD, YOLOv7, sentinel-tree-cover, TreeDetection, Detectron2, and more were researched. Several of these were found to be unsuitable or inaccessible for the scope of this project, especially taking into account the limited time for this project.

The three biggest and most promising libraries that were found were DeepForest, samgeo, and DetecTree.

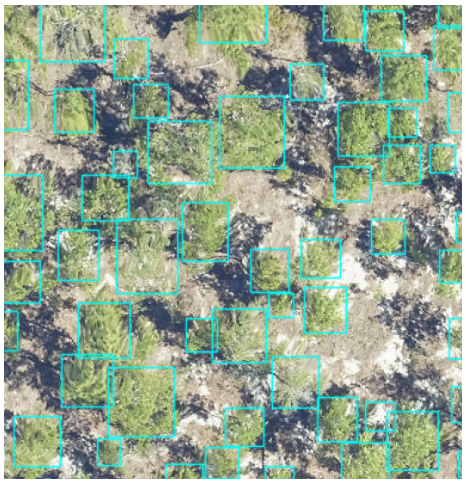
DeepForest

DeepForest is a Python library that detects and draws boxes around specific objects, primarily trees and birds. It has pretrained models for predicting trees, although there are ways to fine tune those models if you supply more data.

The 'default' function for predicting trees in an image is predict_image. Here is an example script that uses it:

```
from deepforest import main
from deepforest import get_data
import matplotlib.pyplot as plt

model = main.deepforest()
model.use_release()

sample_image_path = get_data("OSBS_029.png")
img = model.predict_image(path=sample_image_path, return_plot=True)

#predict_image returns plot in BlueGreenRed (opencv style), but matplotlib
likes RedGreenBlue, switch the channel order. Many functions in deepforest
will automatically perform this flip for you and give a warning.
plt.imshow(img[:,:,::-1])
```

This will output an image with predicted bounding boxes for the image at OSBS_029.png using the default tree detection model. This script works best with photos that were taken lower to the ground, such as the included OSBS_029.png from Florida. Here is the result of this script on that photo:

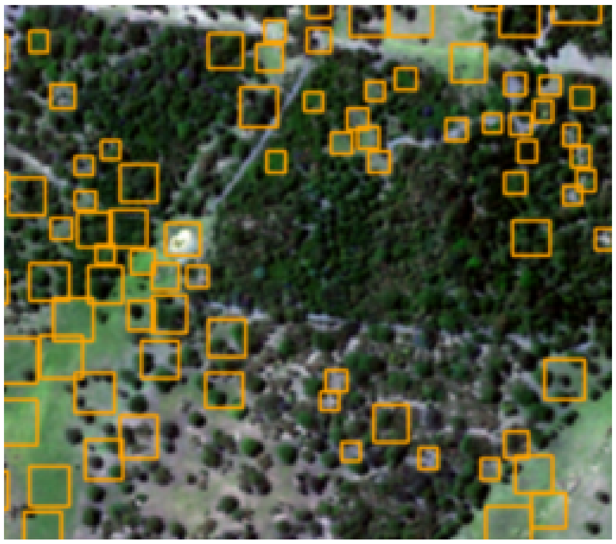


Results from sample image from Florida provided by DeepForest

The provided sample photo is too big and taken from too far away for this default script to be appropriate.

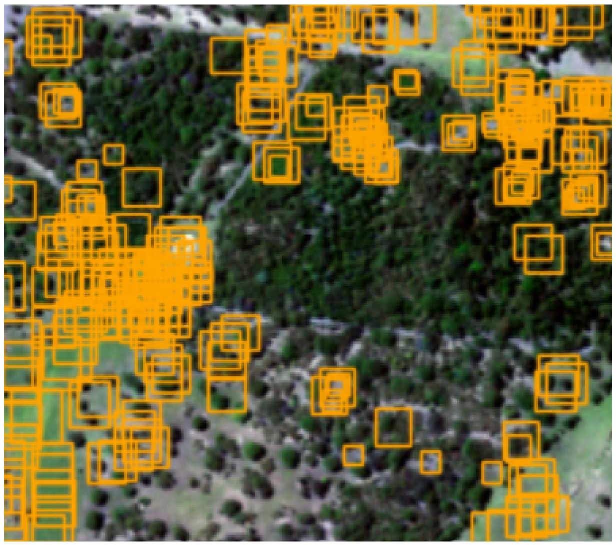
Results of sample image from rural Queensland provided by Prof. Bin Liang

Running the script on a small section of the image also does not give a great result.



Results of small section of previous rural Queensland imagery, nms_thresh = 0.05

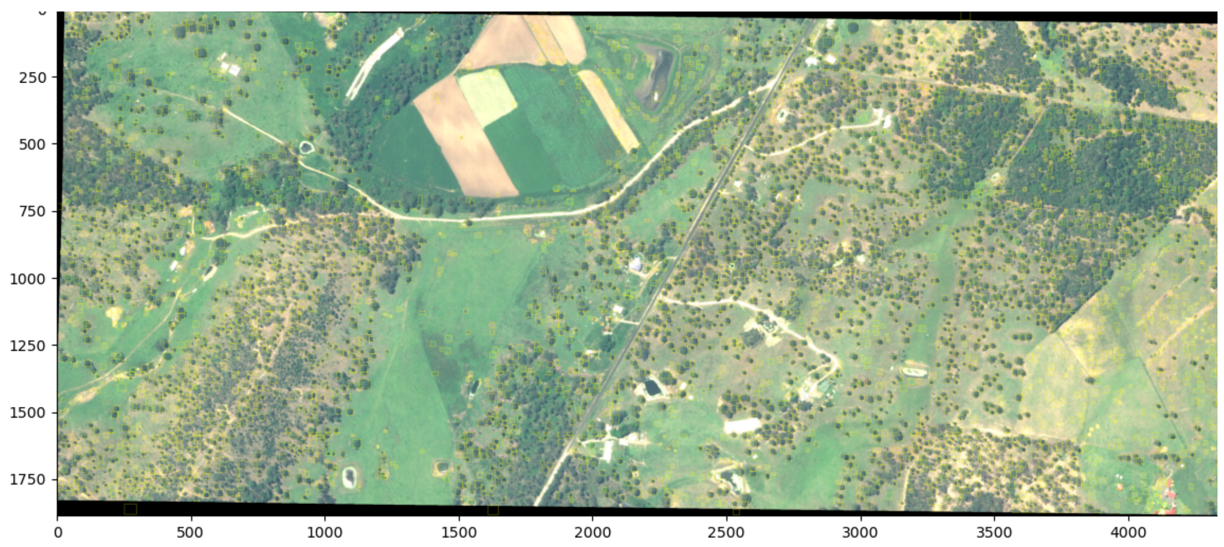
The only parameter that could be edited were nms_thresh, which was edited from the deepforest_config.yml file. This parameter stands for Non-Maximum Suppression Threshold. It determines how much the predicted tree boxes are allowed to overlap. The default value is 0.05. Editing this parameter doesn't help predict the sample data any better, since the amount of overlap is not the problem. Here is the result when the nms_threst was set to 0.5:



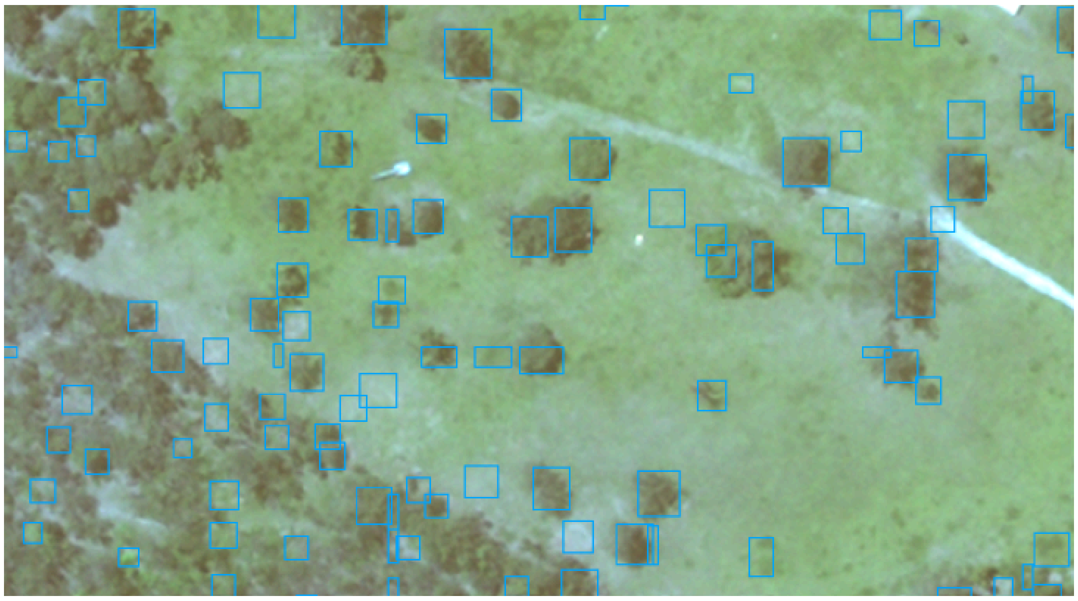
Results of small section of previous rural Queensland imagery, nms_thresh = 0.5

Since predict_image was very unhelpful without any extra training, another function which works better with larger images, called predict_tile was found. While predict_image predicts the entire image at once, predict_tile splits the image into multiple 'tiles' and runs a prediction algorithm on those. Here is the sample script for using predict_tile:

```
from deepforest import main
from deepforest import get_data
import matplotlib.pyplot as plt

model = main.deepforest()
model.use_release()

raster_path = get_data("sample_QLD.tif")
# Window size of 300px with an overlap of 25% among windows for this small tile.
predicted_raster = model.predict_tile(raster_path, return_plot = True,
patch_size=300, patch_overlap=0.25)

# View boxes overlayed when return_plot=True, when False, boxes are returned.
plt.imshow(predicted_raster)
plt.show()
```

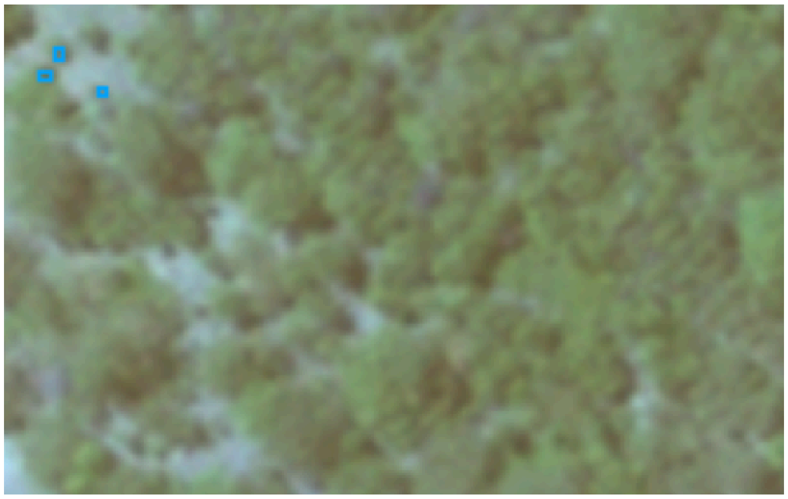


Result of running above script with the sample image



Previous image zoomed in more

predict_tile also has more parameters that can be adjusted without having to re-train the model. One of those parameters is patch_size, which determines the size of the patches that the image is broken into. The default patch size is 300. If there is a patch size that is too small, then the patches will be smaller than the trees themselves, so the bounding boxes will not be visable over entire trees. Smaller patch sizes means more patches are generated, so very small patch sizes takes a long time to run. This is a portion of the sample image run with a patch size of 30. It took 7 hours to run the entire sample image with this parameter.



Results from patch_size = 30

Installation and Running:

To use deepforest, conda is required to be installed within the local environment. Conda then has to be activated to be able to run conda commands. In Windows, this is done by opening the Anaconda Powershell Prompt app. On Linux, this is done by running a script like source anaconda2/bin/activate. Then, a deepforest environment can be created that will be saved within the local conda environments. Every time a script that requires DeepForest is ran, conda must first be activated, followed by the activation of the deepforest environment.

Windows installation steps for DeepForest:

1. Install conda (run the following commands in Powershell)

```
$ curl https://repo.anaconda.com/miniconda/Miniconda3-latest-Windows-x86_64.exe -o miniconda.exe

$ start miniconda.exe

$ del miniconda.exe
```

2. Navigate to the Anaconda Powershell Prompt app on your computer

3. On the Anaconda Powershell Prompt, install and set up DeepForest (run the following commands)

```
$ pip install DeepForest
$ conda create -n deepforest python=3 pytorch torchvision -c pytorch
$ conda activate deepforest
$ conda install deepforest -c conda-forge
```

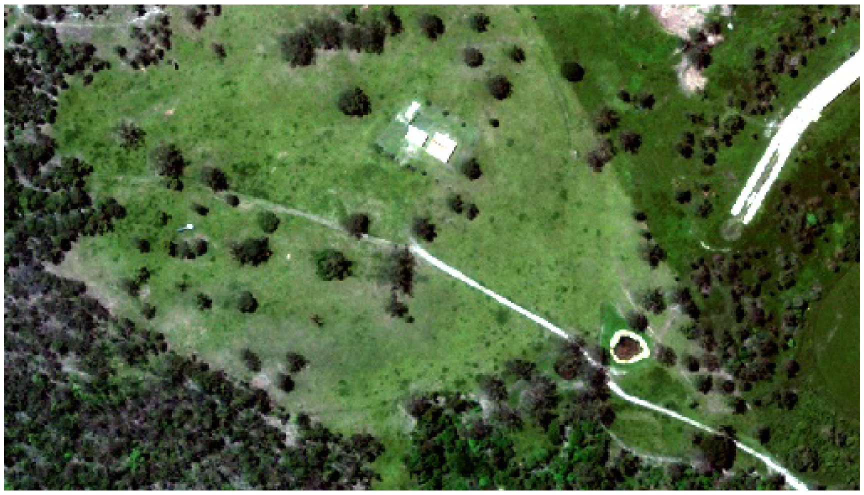
Running a DeepForest Script on Windows:

1. Open the Anaconda Powershell Prompt
2. Run the following command: conda activate deepforest
    a. You need to run this command every time you open up the Anaconda Powershell Prompt
3. Make sure all the files the script needs is in this folder:
    a. <path to your miniconda3 folder> /miniconda3/envs/deepforest/Lib/site-packages/deepforest/data
    b. If you don't know where your folder is, run the script with whatever data it has, and it should show up in the error message
    c. Your deepforest_config.yml file should also be in here if you want to edit it
4. Run the python script from the Anaconda prompt

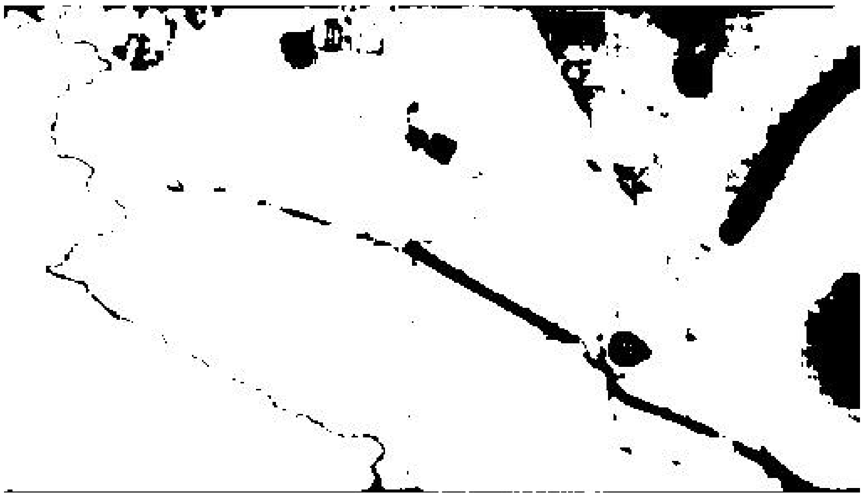
samgeo

samgeo is a Python package to run the segmentation on geospatial data. It uses Meta AI's Segment Anything Model (SAM), but has geospatial specific tools to aid in the segmentation of trees and forests in this case. The model itself is extremely heavy due to the fact that it's supposed to work on any image and isn't specialized to trees specifically even if there is a library that allows it to work better with geospatial objects such as pools, houses, trees, etc. In order to be able to try and fine tune the samgeo tool set, SAM must be trained on the desired location's trees to gain any viable results.

As part of a small test to see general performance, a small section of the Queensland satellite imagery was used, however performance was underwhelming.



Queensland sample imagery subset



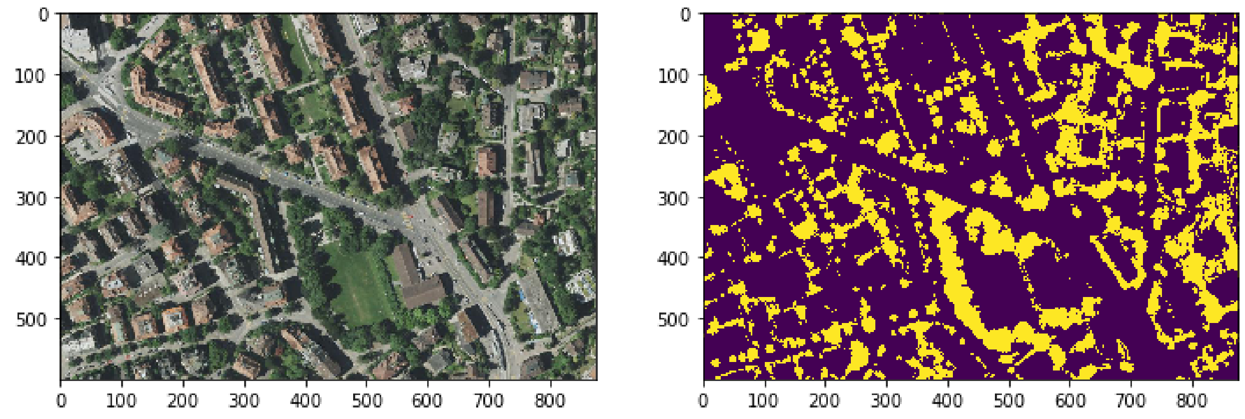
Result of samgeo using default settings LangSAM

As seen directly above, the performance of samgeo was terrible. The white are what samgeo thinks are trees, and samgeo greatly overestimates the canopy coverage. It can only confidently detect the high contrast areas (e.g. where the roads meet the trees/grass). This image in particular took approximately 90 seconds, around 45 times as long as it took for the other models.

Additionally, there were only two parameters to modify, box_threshold and text_threshold. The first parameter determines the threshold/confidence value required to identify geospatial features while the text_threshold determines the threshold/confidence value required to identify trees from the geospatial features. However, while testing these parameters, it was found there was no significant difference in scores when testing, within 1% of each other, so it was not found to be tunable and not a good candidate to move further along with.

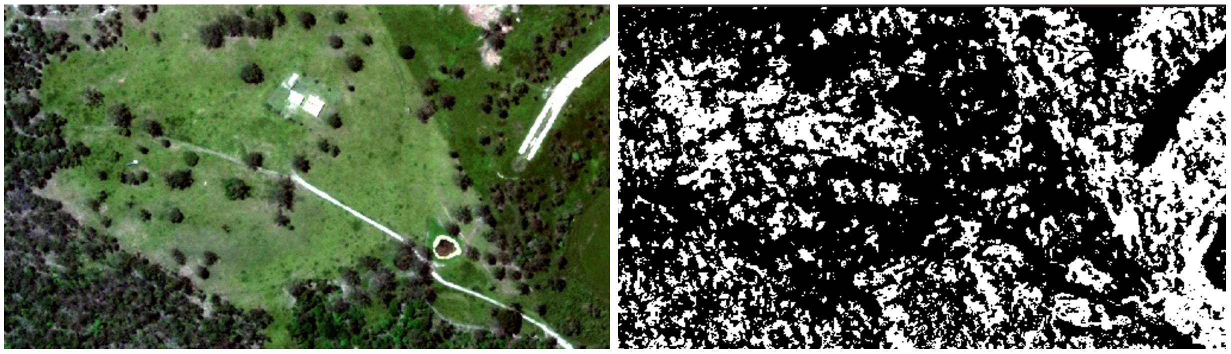
DetecTree

DetecTree is a Python library that segments aerial imagery into tree/non-tree pixels using semantic segmentation. From this machine learning algorithm, it generates a mask of the image where the default black represents no trees and white represents trees. The format of the output mask can be altered with different color schemes, but a white/black layer mask is generally easiest to work with. A pre-trained model for tree detection is available with the library, trained on sample data from Zurich. The model has strong values for IoU (0.87635), precision (0.785237), recall (0.756414), and f1 (0.770556) when tested on aerial images in Zurich.



Zurich sample with the model generating a purple/yellow mask

However, these values are not replicated when used on Australian aerial imagery; IoU (0.290873), precision (0.73206), recall (0.325529), and f1 (0.450661). This library also has the functionality to train your own model based off of an aerial imagery dataset which would likely yield better results when working with Australian canopy.



Australian canopy sample with the model generating a black/white mask

The classifier for the pretrained model also has a built-in refine and beta_refine method, which work to remove isolated pixels and smooth the output, which has an effect on the metrics.

Windows Installation Steps for DetecTree:

1. Install conda (run the following commands)
    a. curl https://repo.anaconda.com/miniconda/Miniconda3-latest-Windows-x86_64.exe -o miniconda.exe
    b. start miniconda.exe
    c. del miniconda.exe
2. Navigate to the Anaconda Prompt app on your computer
3. On the Anaconda prompt, install DetecTree (run the following command)
    a. conda install -c conda-forge detectree

Running a DetecTree script with the default pre-trained model:

1. Import all required packages
    a. matplotlib.pyplot
    b. rasterio
    c. detectree
2. Run the following prompt
    a. detectree.Classifier().predict_img(tile_filename, output_filepath="")
        i. tile_filename is the file path for the image you want segmented
        ii. output_filepath is where the mask will be saved to
        iii. The library works on image files but not GeoTIFF files so conversion is necessary if using GeoTIFF
        iv. Outputs a .png

Benchmarking

Annotated Data

In order to compare the different models and libraries, an apples-to-apples comparison must be made. However, some of the outputs across libraries are different, so it might be difficult to compare accurately. As part of this project, an effort was made to annotate satellite images using QGIS using bounding boxes around the trees. Some were also annotated using an image mask, where the pixels with trees were filled with white pixels and other areas were filled with black pixels.

Annotation Process

There are two ways to annotate data depending on what types of models were being used: bit mask annotations and bounding box annotations.

To annotate the bit masks, it's a relatively simple process using some sort of image processing software such as Photoshop or GIMP. A layer mask can be created where white are trees and black are non-tree pixels. Then the layer can be exported as a mask and used for evaluation and/or training.

To annotate the images, QGIS was used to produce bounding boxes around trees in sample data. The following steps detail how to accomplish this task.

1. Open a QGIS project and drag and drop a PNG image into it.
2. In the Toolbar, click on "New Shapefile Layer"
3. Name the layer, set the Geometry Type to "Polygon" and click "OK"
4. Click on "Toggle Editing", then "Add Polygon Feature", then select "Rectangle From Extent"
5. To draw bounding boxes, left click to start the rectangle, right click to end it; the Image ID box can be left as NULL (hit Enter and move on to the next box)

Evaluation Metrics

Since there were multiple types of outputs, there were a few metrics that were chosen. The four evaluation metrics were Accuracy, Precision, Recall, and F1 Score. The Intersection over Union score was used for bounding boxes annotations and the others are used for image masks. The image mask metrics were done pixel by pixel. The following scores can be calculated with the formulas below.

Accuracy

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Predictions}$$

Precision

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Recall

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

F1 Score

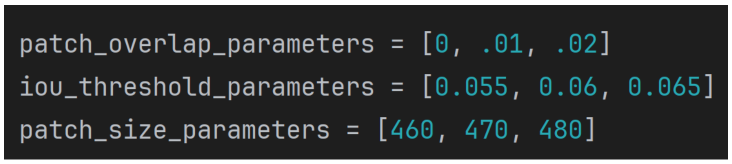
$$F_1 = \frac{2 \times (Precision \times Recall)}{Precision + Recall}$$

Results

After doing small tests with all three models/libraries, it was decided that samgeo would not be further tested due to its massive performance overhead and significant performance difference when compared to the others due to the SAM model being extremely general.

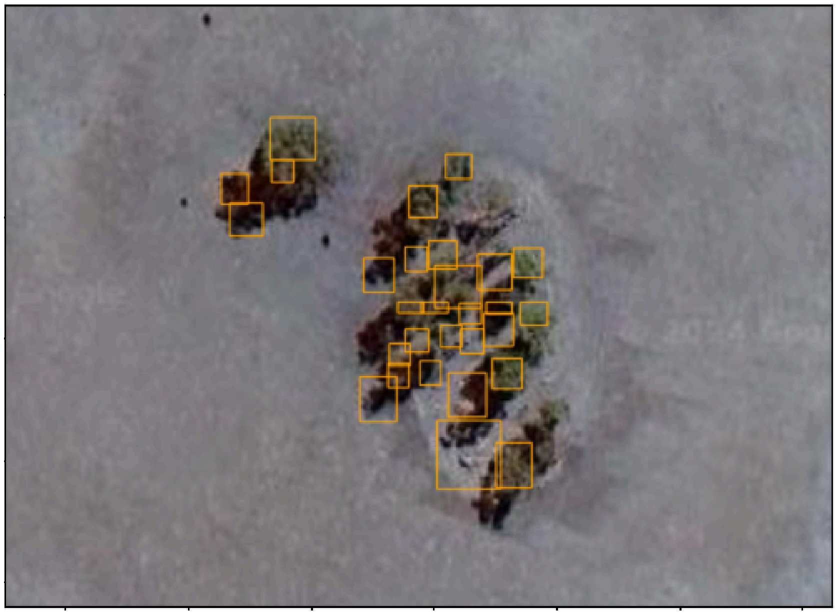
Deep Forest

Given the range of evaluation metrics to test on the results of model predictions, high precision and recall was the driving factor. For DeepForest, IoU was used to calculate the model's score. The ground truth annotations were given in the form of a CSV file containing the coordinate points of bounding boxes. Using a 550px by 250px test image, the model generated a prediction with an IoU of 0.163. Grid Search was run using the following parameters to find the optimal values for patch overlap, IoU threshold, and patch size.



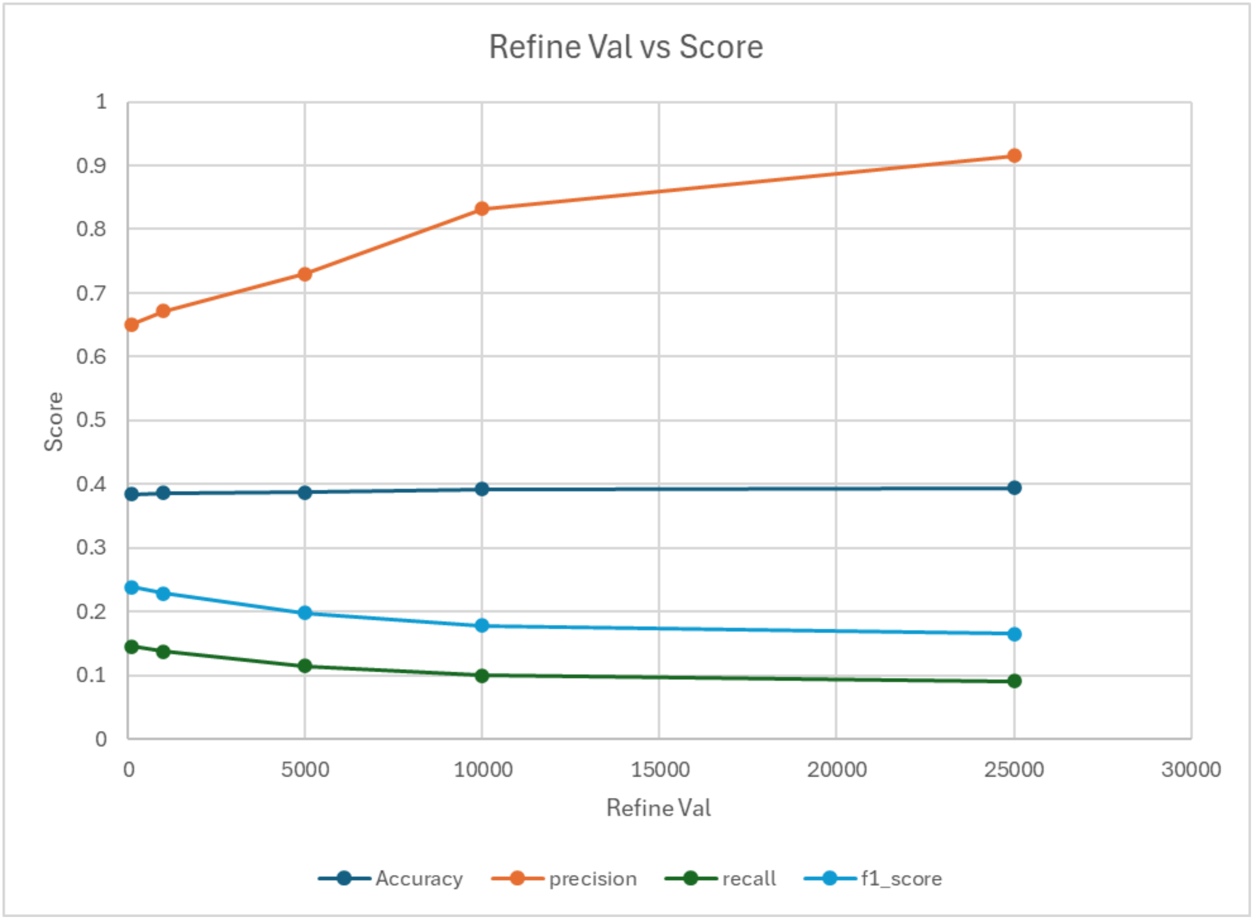
```
patch_overlap_parameters = [0, .01, .02]
iou_threshold_parameters = [0.055, 0.06, 0.065]
patch_size_parameters = [460, 470, 480]
```

The result was a max IoU of 0.163 with {'overlap': 0.02, 'threshold': 0.06, 'patch_size': 480} as the optimal parameters. The following image is zoomed in on a tree section to better visualize the results.



DetecTree

When comparing the model generated mask to the ground truth bitmask, the base DetecTree model excelled in the precision metric and overall had mediocre scores for IoU. These metrics were affected by the level of beta_refine, or refining, done to the images. When doing this, the precision values excelled while all other values diminished. The highest IoU value was at a beta-refine value of 0, with a score of 0.294. IoU though is best for the bounding boxes, so the calculations, accuracy, precision, recall, and F1 are best for the bitmask.



Metrics tested at different beta_refine values

Analysis

DeepForest

When choosing an evaluation metric, it was important to find a balance between precision and recall. Since precision measures accuracy of identifying trees and recall measures the completeness of detecting all trees, IoU was the best balance. DeepForest excelled at allowing flexibility with modifying parameters such as patch size and IoU threshold. The built-in predict tile function was helpful when detecting trees in large geotiff files or aerial images. It is able to segment a larger image into smaller ones then recombining the patches. The shortcomings of DeepForest was the inability to optimize IoU beyond a certain capacity, since the model is limited by test data. The scope of this project did not allow for enough resources or data to retrain the model given the limited time, but it is certainly a task that can be accomplished by future researchers. As a result, beyond grid search, there is not much more tuning of parameters that can be done to improve tree detection.

DetecTree

DetecTree excelled when using precision as an evaluation metric. However, IoU does not work as well due to its pixel-by-pixel detection of trees. DetecTree also fell short of differentiating between grass and trees; with refine_value being the only major parameter that was changeable, not much fidelity was achievable.

Comparison

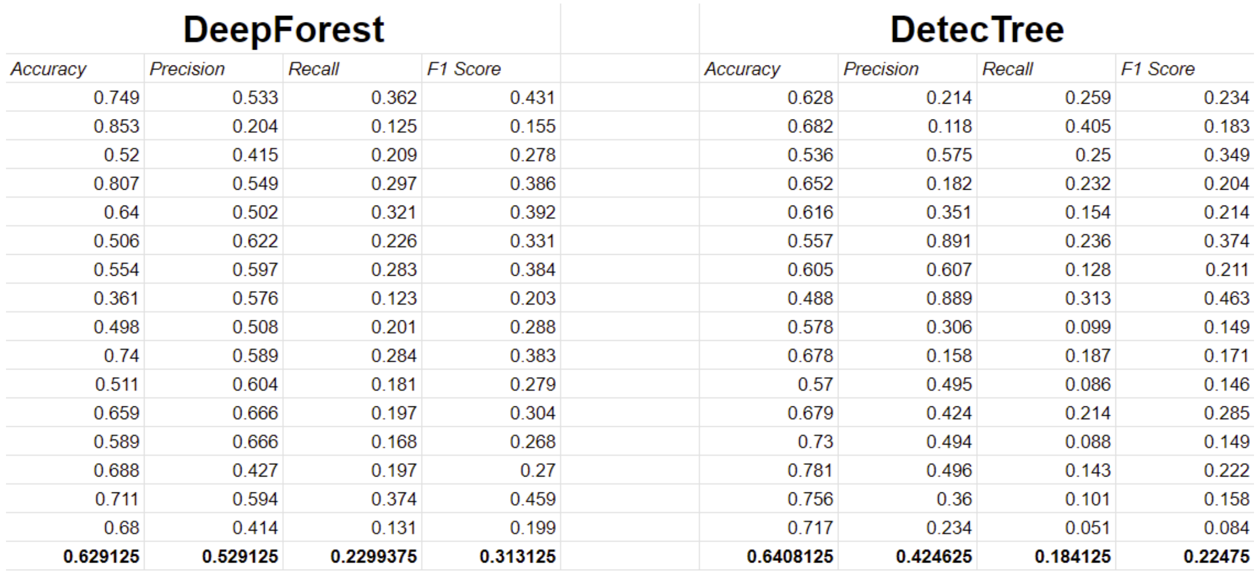
To compare DeepForest and DetecTree, a pipeline program was created in Python using conda environments to implement the models. The program uses Postman as an endpoint to send in an image file, select a processor (either DeepForest or DetectTree), and obtain an output "predicted" file. For DeepForest, this consists of an image with bounding boxes. For DetecTree, a bitmask is produced.

This provides the user with a streamlined workflow to obtain side-by-side predictions. However, to evaluate the accuracy of these models, a ground truth file must be compared with the predictions. For the purpose of this project, the images were simplified into black and white images – white pixels represent trees, while black pixels represent not trees (much like DetecTree). The ground truth files were annotated by hand, creating a pixel mask for DetecTree and a bounding box-like mask for DeepForest. These ground truths were then passed through an evaluation program which calculated accuracy, precision, recall, and F-1 score.

```
for w in range(width):
  for h in range(height):
    true_positive += 1 if expected[w][h] and actual[w][h] else 0
    true_negative += 1 if not expected[w][h] and not actual[w][h] else 0
    false_positive += 1 if not expected[w][h] and actual[w][h] else 0
    false_negative += 1 if expected[w][h] and not actual[w][h] else 0

accuracy = (true_positive + true_negative) / (width * height)
precision = true_positive / (true_positive + false_positive)
recall = true_positive / (true_positive + false_negative)
f1_score = (2 * precision * recall ) / (precision + recall)

return accuracy, precision, recall, f1_score
```

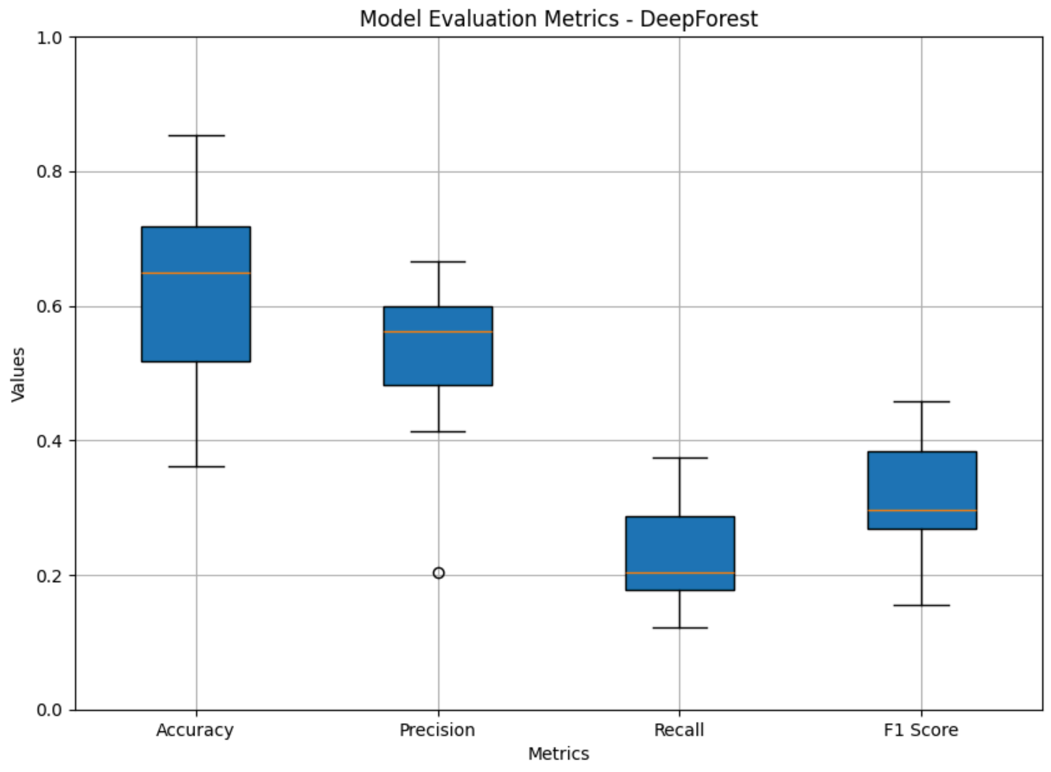
Evaluation Code on GitHub

The results were as follows:

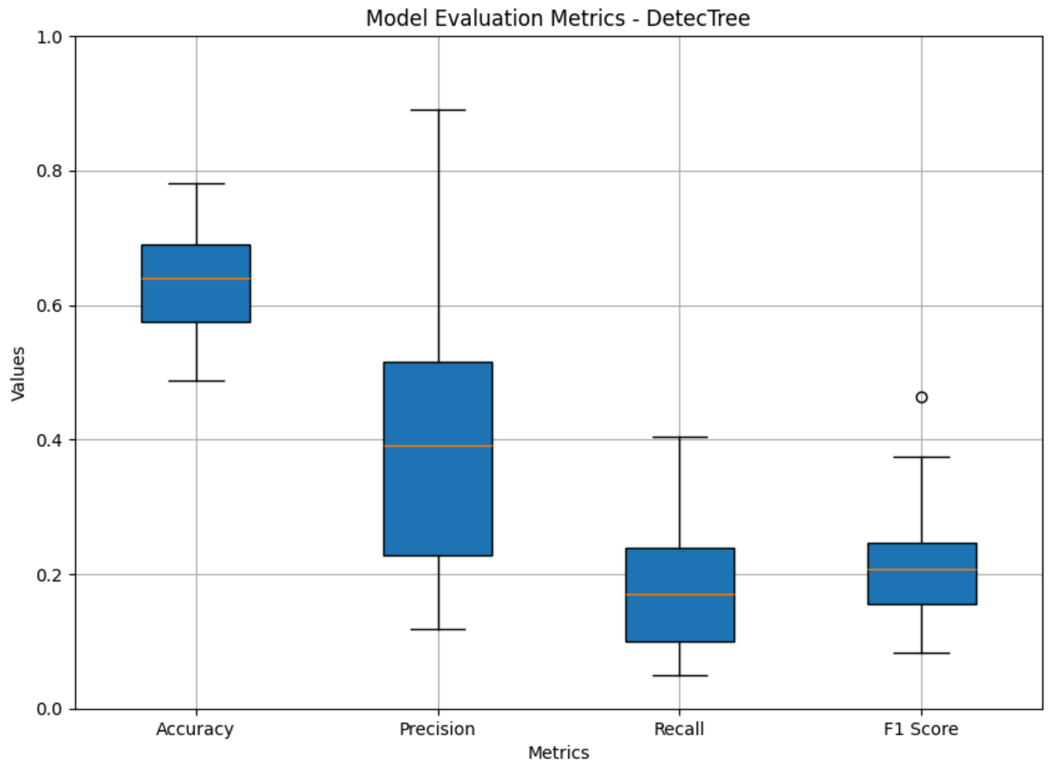


DeepForest				DetecTree			
Accuracy	*Precision*	*Recall*	*F1 Score*	*Accuracy*	*Precision*	*Recall*	*F1 Score*
0.749	0.533	0.362	0.431	0.628	0.214	0.259	0.234
0.853	0.204	0.125	0.155	0.682	0.118	0.405	0.183
0.52	0.415	0.209	0.278	0.536	0.575	0.25	0.349
0.807	0.549	0.297	0.386	0.652	0.182	0.232	0.204
0.64	0.502	0.321	0.392	0.616	0.351	0.154	0.214
0.506	0.622	0.226	0.331	0.557	0.891	0.236	0.374
0.554	0.597	0.283	0.384	0.605	0.607	0.128	0.211
0.361	0.576	0.123	0.203	0.488	0.889	0.313	0.463
0.498	0.508	0.201	0.288	0.578	0.306	0.099	0.149
0.74	0.589	0.284	0.383	0.678	0.158	0.187	0.171
0.511	0.604	0.181	0.279	0.57	0.495	0.086	0.146
0.659	0.666	0.197	0.304	0.679	0.424	0.214	0.285
0.589	0.666	0.168	0.268	0.73	0.494	0.088	0.149
0.688	0.427	0.197	0.27	0.781	0.496	0.143	0.222
0.711	0.594	0.374	0.459	0.756	0.36	0.101	0.158
0.68	0.414	0.131	0.199	0.717	0.234	0.051	0.084
0.629125	**0.529125**	**0.2299375**	**0.313125**	**0.6408125**	**0.424625**	**0.184125**	**0.22475**

List of values for 16 images and average values



Box and Whisker chart for DeepForest



Box and Whisker chart for DetecTree

This allowed us to obtain a parallel comparison for the results generated by DeepForest and DetecTree.

Fine-Tuning and Training

Each of the three main models has a different process for training/fine-tuning. As such, the focus of the project was to find the model that had the most accessible and continuable fine-tuning process. DeepForest, ultimately, had the best documentation for continuing training and the form of data required seemed the most feasible to create given the constraints of the data. All three models are possible to fine tune, however, the level of success of each would significantly vary depending on the circumstances.

Segment Anything Model (SAM) for Geospatial Data (samgeo)

As mentioned previously, samgeo is a set of tools for using Meta's Segment Anything Model on Geospatial data. This means that in order to fine tune the model towards higher accuracy on tree segmentation/identification (especially if specifically Australian/Queensland trees). As SAM is a vision transformer the fine-tuning process would be much more similar towards fine-tuning a model like GPT-4o than any of the other models we tested. Due to its out-of-the-box performance being high, fine-tuning this model has the highest potential for excellent results, however the shape of the data as well as the large number of model parameters makes it very difficult to acquire and manipulate sufficient data for training.

DetecTree

Similarly to samgeo, DetecTree has great potential for training, but one major point held it back from being a focus on fine-tuning. Namely, that the model requires image-mask data for fine-tuning rather than metadata associated with the image representing the location of trees. However, the overall training process for DetecTree is explained in the documentation similarly to DeepForest and training the model would be feasible given sufficient data.

DeepForest

DeepForest and DetecTree have the greatest capability for continued training, but both require two different types of data. While DetecTree uses image-masks, DeepForest uses metadata bounding-box annotations. This type of data is easier to create quickly and possible on lower resolution images. However, there are outstanding questions related to how accurate the data needs to be in order to successfully fine-tune. DeepForest's documentation explains quite well how to fine-tune the model, however there are several key issues that are holding back the training process. The first of which pertains directly to the quality of the data: namely resolution and bounding-box accuracy. In order to fine-tune these models a large amount of data is required - enough to create a reasonable train test split. Drawing bounding boxes accurately on an image necessitates high enough resolution to visibly differentiate trees from one another. Additionally there are possible issues with the exact positioning of bounding boxes. Boxes placed over the edge of the image or overlapping may cause issues when attempting to load data into the model trainer. While attempting to fine-tune DeepForest this issue occurred several times. Furthermore, if DeepForest is installed through Anaconda the training process does not work due to several dependencies being outdated. Training can be done through Notebooks (e.g. Colab or Sagemaker) and checkpoints can be saved. Given the large amount of errors only a pipeline was created (no checkpoints). There still exist questions about how much data would be necessary to get good results. DeepForest mentions that approximately one hour of annotation creation would lead to improved results, however with data as diverse as that from Queensland and as low resolution as the data we found it's very hard to say how well this would work. It is possible that training against the GeoTIFF and its sub-images would lead to improved performance in closely related images, but this likely wouldn't be adaptable to more diverse regions and lower resolution images.

Further Training Ideas

The two models with the most potential for future training are DeepForest and SAM. SAM is a very strong segmentation model, which, if fine-tuned for tree specific data and leveraged with the samgeo toolset could be excellent at tree canopy estimation. DeepForest has the most accessible pipeline for training, but lacks sufficient data. Both of these problems could be potentially approached using Low Rank Adaption (LoRA) training. SAM is a massive model which has a large number of parameters, by training only a select few of them which correspond to specific types of tree canopies it would be possible to train the model more efficiently and be more accessible towards others (allow others to adapt their base models more quickly by applying the LoRA). This, of course, would require more data, and, critically, more accurately annotations.

Data

The data was divided into three separate parts. First, the provided GeoTIFF file, for which an image-mask was created and annotations were created (~4000 tree annotations). Second, an image set of one hundred screenshots from Google Maps from throughout Australia of which around fifteen were manually annotated (~10000 tree annotations). Last, a dataset discovered in the last week which had very high resolution images from the Northern Territory as well as ~3000 tree annotations.

GeoTiff

The GeoTiff image was split into sixteen separate sub-images of approximately the same resolution as the Google Maps screenshots. These sub-images then had DeepForest annotations created. Due to the high resolution of this image it was possible to accurately find individual trees as well as create an image-mask.

Google Maps and Tiles

Before the Google Maps Tile API was set up, a dataset of approximately one hundred images was created through screenshotting various regions in Australia, primarily in Queensland, New South Wales, and Victoria. These images were then collated into a repository and used for annotation creation. Of the one hundred images, approximately fifteen were annotated totalling in around 10000 individual tree annotations. These images are of fairly low quality so it is very hard to determine how accurately the trees were annotated. Additionally, the annotations often overlapped and occasionally fell outside the bounds of the images. These discrepancies led to the aforementioned issues with fine-tuning.

Northern Territory Dataset

The Northern Territory dataset contained a set of seven large aerial photographs. These are of very high resolution, showing detail in individual branches of trees. Additionally each image also had a shapefile containing a set of annotations for the image. Notably, none of the images were completely annotated leading to some additional issues in attempting to train. This dataset has the most potential in further annotations due to its high quality unless further images from the GeoTIFF dataset are able to be acquired. Overall this dataset contained approximately 3000 good annotations, but no image was fully annotated.

Conclusion

The findings of this project prove that further data is a necessity to finetune the models. Both DeepForest and DetecTree produce average precision values of around 50%, which is extremely low for a machine learning model. Further annotated data is required to provide the model with enough ground-truth observations to produce accurate results.

However, the project identifies and explores several different parameters that are important to the functioning of DeepForest and DetecTree. Furthermore, it implements these models and modifies these parameters through Python scripts, evaluating them on the basis of Intersection over Union, Accuracy, Precision, Recall, and F-1 Score. The workflow for these models is detailed and optimized, allowing for easy use of them by future researchers.

These findings of this project can be helpful in determining above ground biomass and CO_2 absorption by trees. The predictions from both models can be used to calculate the total area with canopy coverage. Fast growing trees absorb CO_2 faster than older trees which can be determined by the size and type of tree. These factors can be incorporated into the models to predict CO_2 absorption rate and climate change. The flexibility of DeepForest and precision of DetecTree allow expandability and ability to visualize and calculate the square footage of canopy cover.