CMPE257
San José Urban Forest
Final Report

Group 04

William Parker - 017461899
Manan Choksi - 017448899
Martin Alvarez Lopez - 012328706
Sai Yaaminie Ganda - 017054700
Vaibhavi Savani - 017456972

Date: 12/01/2023

1. Idea

1.1. Description

Tree canopy cover, also referred to as urban forest canopy cover, is defined as an area of land covered by trees, their leaves, and branches, when viewed from above.

The City of San José's tree canopy has been shrinking at an alarming rate, with citywide coverage dropping from 15.36 percent in 2012 to 13.54 percent in 2018, representing a 1.82% loss that accounts for 11.8% of the total canopy cover, equivalent to the loss of 1,728 acres or 2.7 square miles [1]. To recover just 1% of this lost canopy, the city will need to plant a substantial 40,000 trees with a 35-foot canopy spread. It's important to note that losing a tree is immediate while replacing it can take around 30-40 years. Preserving the existing canopy coverage is identified as the most cost-effective and efficient means of increasing canopy cover. Projections indicate that if the current trend persists, the city's overall canopy coverage could potentially plummet to less than 10% by 2030.

In San José, removing a tree from private property often requires a permit which typically requires replacing it with a certain number of trees, the specifics of which depend on the size and species of the tree being removed and the property type; however, if planting replacements on the same property isn't possible, applicants can opt to pay an in-lieu fee, which the Department of Transportation (DOT) uses to plant a tree in another location [2]. Furthermore, the City Auditor, Joe Rois, found that there were mass discrepancies in permits submitted towards the in-lieu fees, which resulted in significantly fewer trees being planted. Additionally, San José currently does not have a verification process set in place to ensure the necessary trees were replanted and if trees were planted, the city has no way of verifying if they are suitable species or appropriate size to replace the initial tree with. Presently, the city relies on a complaint-based system for Code Enforcement to investigate illegal tree removals, leaving unreported cases to result in the loss of existing trees without the requirement for replacement.

The City of San José directly manages an estimated 300,000 public space trees and it last completed a tree inventory in 2014. Most cities often update their tree inventory when planting, maintaining, or pruning trees.

However, because in San José private property owners are responsible for around 86% of street trees, the city does not have a reliable inventory system. As a result, the city does not have a reliable tree inventory of all public space trees. Conservatively, a citywide tree inventory would cost around $4 per tree which means a tree inventory for all 300,000 trees would cost more than 1 million dollars.

Due to the City's insufficient resources, a solution should be cost-effective, efficient, and conscious of staffing. To be able to manage its trees, the City will need data. City employees need reliable, up-to-date information, to make decisions. Frequently updated data to track trees can empower arborists to make more efficient and better decisions for the urban forest's well-being.

1.2 Goals/Objectives

We propose a solution that utilizes available resources such as yearly aerial imagery to generate data on individual trees to track all of San José's Tree canopy, both in public and private spaces within the city limits. This solution will be cost-effective with publicly available data and open-source software.

Essentially, our solution will be able to break down available aerial imagery datasets and describe them for future work. From the aerial imagery, we will be producing location information for individual trees. The location information is derived from tree bounding boxes interpreted as coordinates. Additionally, from our tree bounding boxes, we will be able to calculate the area and give an estimate for the total tree canopy coverage the City has with each year's aerial imagery.

2. Work Developed

2.1 Adopted ML algorithms/techniques

2.1.1 WMS-tile-get

We use aerial images to detect the tree canopy. To obtain the data for our project, we utilize the WMS-tile-get service [3]. WMS-tile-get requires several inputs, such as the server's address, image size, zoom level, output path, and GeoJSON file. The GeoJSON file outlines the boundary limits of

the city of San José, which is in the shape of a polygon. WMS-tile-get downloads images from the WMS server based on the provided input. The zoom level and image size are crucial factors. If the image is overly zoomed in, we will have more images to process, requiring significant computational resources and time. Conversely, if the image is too zoomed out, the model may struggle to clearly identify trees. Zoom level 18 appears to be the optimal choice. For each year, we had a total of 33,768 images, each sized 256 x 256 pixels. WMS-tile-get downloads files in the slippy map format.

2.1.2 Resampling Tiles

After downloading the images from the server, we need to process them for further use in our machine learning model. Each image is converted from PNG format to geo-referenced TIFF format [4]. Additionally, we resize the images and introduce some overlapping. Resizing images from 256 x 256 to 1312 x 1312 enables them to offer higher spatial resolution.

When processing the images, it's important to note that edges may behave differently than central regions, potentially leading to edge effects where the model may not perform as well near the borders. To address this, we provide an overlap of 96 pixels to handle the edge effect and ensure that the model learns to recognize object boundaries effectively. After resampling, there will be a total of 1844 images for each year. We will also generate a metadata file containing the “nodata” ratio for each resampled image.

2.1.3 Tile Selection for Labeling

The DeepForest algorithm can be fine-tuned to San José urban cover by using supervised learning. The images for training can be selected randomly from the set of 1844 images, but randomly selected images may not be representative of the entire dataset. The most efficient approach is to use an unsupervised learning algorithm to capture details in the image. In this project, we have used k-means clustering. The k-means algorithm creates groups out of the images that capture the diversity in the images and then selects images from these clusters for training.

The “nodata” ratio is the ratio of an image that contains no information. Selecting images with a high nodata ratio will not provide much information

for the algorithm to learn. We select the images with less than a 30% nodata ratio for the k-means algorithm to form a training set.

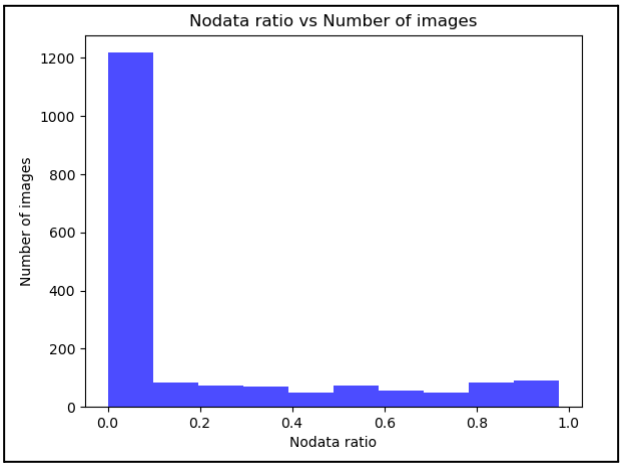


Figure 1: A histogram showing the number of images vs nodata ratio

Total images - 1844

‘nodata’ ratio	Number of images
<= 0.25	1344
<= 0.3	1375
<= 0.4	1445
> 0.3	469

Table 1: Histogram statistics

Gist descriptors are the vector representation of the characteristics of an image. This captures the features like color, texture, and shape of an image. DeepForest expects an 832 component gist descriptor for each image [5].

As the k-means algorithm with an 832-dimensional gist descriptor for each image will take a lot of computational resources, PCA is performed to reduce the dimensionality of the gist descriptors.

Number of principal components	Variance captured
3	31.5%
12	55.9%
24	71.2%
64	91.25%

Table 2: The selection of principal components using PCA

We use the train_test_split method in the TrainingSelector class from the DetecTree library [6], which internally computes gist descriptors for each image and then uses k-means clustering combined with PCA to select the images to form the training set.

This makes two first-level clusters, which split the dataset into 817 and 558 respectively into cluster-0 and cluster-1. We then apply K-means again to the first-level clusters by setting the number of clusters to be 2% of the data in their respective first-level clusters. Then an image nearest to the second-level cluster centroid is selected from all the second level clusters.

First Level Clusters	Number of Images	Second Level Clusters
0	816	17
1	559	12

Table 3: K-means selection results

Finally, we got 29 images from 1345 images for the train set.

2.1.4 Labeling

We installed the LabelMe tool and divided the labeling of the 29 images amongst all 5 group members. We labeled the images and saved the results as JSON files where each label was an entry in the file with the top left and bottom right coordinates of the rectangle saved. An example of this can be seen in Figure 2.

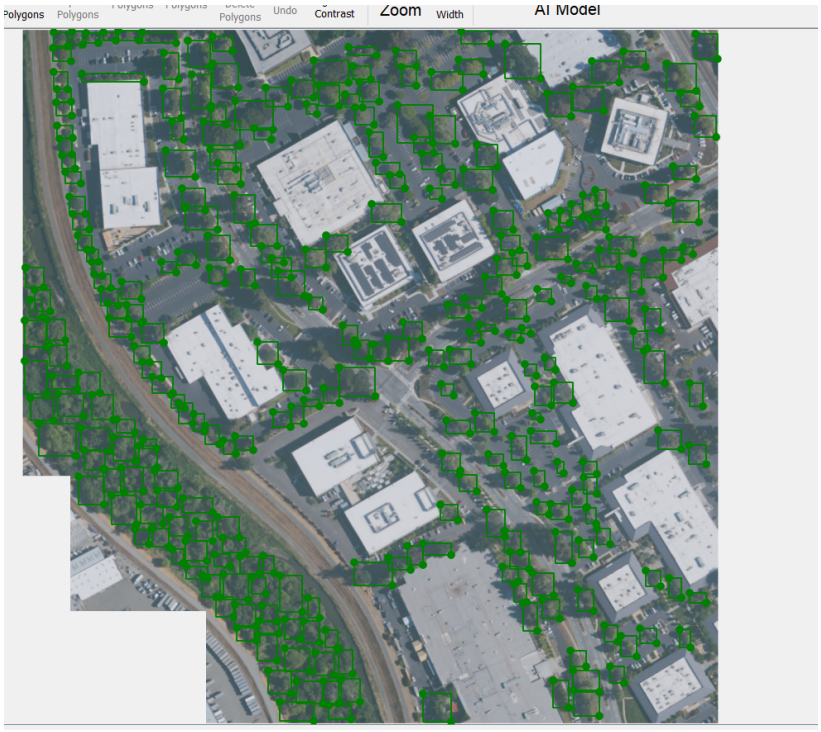


Figure 2: Labeling using LabelMe

Next, we took advantage of an Urban Tree function called “generate_response” (Figure 3) which gathered all labels in every JSON file and collated the data in a single CSV file. The generated CSV file contained the image path and respective coordinates of each bounding box. Luckily, the “generate_response” function also contained functionality that helped us split the labels into two CSV files: one that contained train data and another that contained validation data which helped make the fine-tuning process more efficient.

```
def generate_response(dataset_train_dir, dataset_response_dir,
                      model_trainer_min_bbox_size,
                      model_trainer_min_bbox_ratio,
                      model_trainer_validation_ratio,
                      model_trainer_patch_sizes=None,
                      model_trainer_patch_overlap_size=32,
                      **ignored):
  """
  generate response of training dataset with labelme annotation results.

  Parameters
  ----------
  dataset_train_dir : str
    the path to input training dataset folder with labelme annotation results in json
  dataset_response_dir : str
    the path to output training response folder for further training with torch vision
  model_trainer_min_bbox_size : int
    minimal size of object bbox which should be trained
  model_trainer_min_bbox_ratio : float
    minimal ratio (short_side/long_side) of object bbox
  model_trainer_validation_ratio : float
    train/validation split ratio
```

Figure 3: Generate Response Code

Most of the labeling we did was conducted on images of size 1312 x 1312 which was too large for the DeepForest neural network. [7] The DeepForest neural network was trained to detect trees in images of square size around 400 pixels. The “generate_response” function implemented DeepForest’s “split_raster” (Figure 4) function which helped to split our images into square blocks of 450 x 450.

```
df.to_csv("full_annotations.csv",index=False)
annotations = preprocess.split_raster(
    path_to_raster=image_path,
    annotations_file="full_annotations.csv",
    patch_size=450,
    patch_overlap=0,
    base_dir=directory_to_save_crops,
    allow_empty=False
)
```

Figure 4: Code Snippet of Splitting Raster

2.1.5 DeepForest

DeepForest is a deep-learning neural network that has been trained to detect tree crowns where each layer helps to understand the color and shape of different canopies. [8] DeepForest contains a pre-built model that was trained on data from the National Ecological Observatory Network (NEON). [9] The model was pre-trained using an unsupervised

LiDAR-based algorithm to generate millions of annotations. The model was then retrained on RGB images to based on the LIDAR annotations in order to train the network to recognize tree canopies through RGB alone.

```
# Example run with short training
# annotations_file = "/content/drive/My Drive/deepForest/combined.all.csv_"
annotations_file = "/content/drive/My Drive/deepForest/combined.train.final.csv"
validation_file = "/content/drive/My Drive/deepForest/combined.valid.final.csv"
df = pd.read_csv(annotations_file)

model = main.deepforest()
model.config["epochs"] = 1
model.config["save-snapshot"] = False
model.config["train"]["csv_file"] = annotations_file
model.config["train"]["root_dir"] = os.path.dirname(annotations_file)
model.config["validation"]["csv_file"] = validation_file
model.config["validation"]["root_dir"] = os.path.dirname(validation_file)
model.use_release()
model.create_trainer()
```

Figure 5: Code Snippet of Fine Tuning Model

For the purposes of San José tree canopy detection, we wanted to be able to fine-tune the existing model to fit our use case (Figure 5). Hence, we used the validation and training data sets we generated from the labeling process as inputs to the DeepForest model's pre-trained network.

2.2 Performance Metrics

To measure the effectiveness of our fine-tuned model we measured the box recall and precision on our validation data set. Due to the unique nature of our labels, the DeepForest library recommends we use an IntersectionOverUnion (IoU) metric. The IoU metric measures the area of overlap of two bounding boxes over the total union area. A representation of this metric can be seen in Figure 6. DeepForest recommends using a 0.4 threshold rather than the conventional 0.5 because it is more useful for ecological purposes.

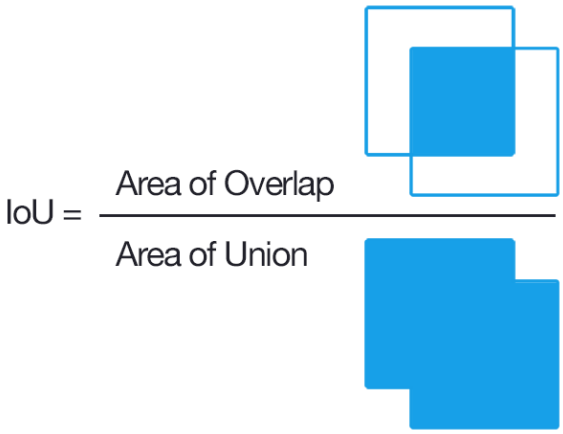


Figure 6: Representation of IoU Metric

After running the evaluation method DeepForest provides, our overall box recall was 0.65 and our precision was 0.47 (Table 1). While these scores may seem relatively low it's worthwhile to keep in mind that state-of-the-art LIDAR algorithms possessed a 0.71 box recall. Hence, it's fair to say that the RGB model performs fairly well given its limitations.

We also decided to run a comparison with the pre-trained model to see how well the fine-tuned model did. As you can see from Table 4, our fine-tuned model did almost twice as much better than the pre-trained version.

	Pre-Trained Model	Fine-Tuned Model
Box Recall	0.38	0.65
Box Precision	0.35	0.47

Table 4: Box recall and precision for the pre-trained and our fine-tuned model

2.3 Evaluation

We used two additional methods to evaluate its utility. Firstly, we chose to visually evaluate a portion of the resulting boundary boxes generated by the model to confirm that it was detecting the trees as expected. Secondly, we compared the total canopy size detected to the City of San José's official 2017 calculations.

Visually proving that the model is accurately detecting trees is a simple way to show the capabilities of the model to a general audience. In Figure 7, we can see that larger trees in a city can be detected. While it does not detect all trees, it can detect canopy at a much higher rate than the base model in this image, which only detected two trees in this image. While we cannot fully rely on this output, we can start to drive data from it.

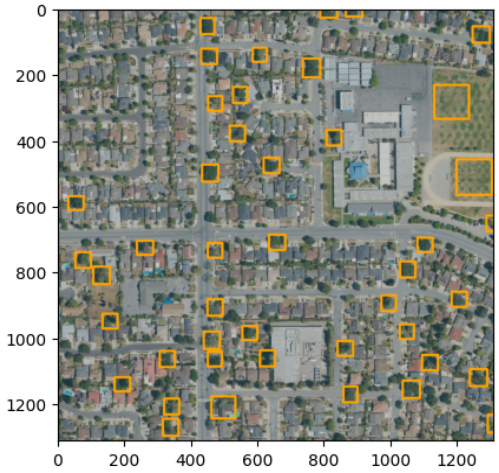


Figure 7: The fine-tuned model on an aerial image of San José

The size of any tree canopy can be calculated using the results from DeepForest. The image stored the coordinates of the corners of the image and the relative positions of the corners of the tree boundary box. Using this data, we could measure the area of tree canopies. After calculating the tree canopy area for every tree in San José and summing the values, our model predicted that there was a total of 61,318,479 meters squared of tree canopy across the city in 2022.

The City of San José [1], reported approximately 63,138,374 meters squared of canopy cover in 2018. They also reported a loss of 1.8% of tree canopy cover per year, making the expected total 58,713,685 meters squared in 2022. This puts our model on a close track with the city's reported expected total for 2022, indicating alignment between our predictions and official calculations. However, a difference of about 2.6 million square meters exists between our model's prediction of 61,318,479 square meters and the city's estimated 58,713,685 square meters for 2022.

2.4 Future Work

In our future work, we will gather aerial imagery with better resolution than the one provided by the California Department of Fish and Wildlife [10]. With the resolution we had, it was difficult for us to manually label trees to fine-tune the model. More resolution means our model will have more information to identify trees.

Additionally, we will focus our efforts on developing map layers, or shapefiles, to highlight trees in a map by year and also to identify tree changes by year such as trees being removed or added.

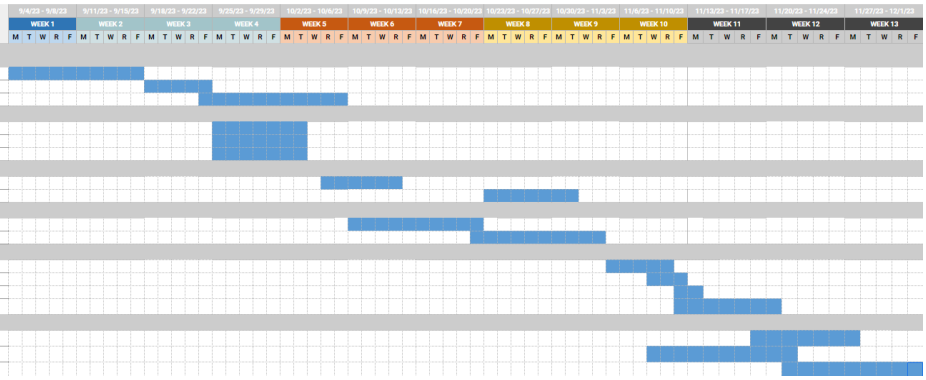
Ultimately, our team would like to develop a tree inventory for the City of San Jose using the data and techniques we developed in this project. As part of that, we want to determine if we can incorporate Lidar or multispectral data to refine our model and increase its performance.

3. Project Management

3.1. Final Schedule & Task Distribution

PROJECT TITLE	San Jose's Urban Forest Project
PROJECT MANAGER	Martin

Sr. no	TASK TITLE	TASK OWNER	START DATE	DUE DATE
1	**Project Definition and Planning (Initial Phase)**			
1.1	Problem definition	Martin	9/4/23	9/15/23
1.2	Initial Report	All	9/18/23	9/22/23
1.3	Data Collection	Martin	9/22/23	10/6/23
2	**Existing Research**			
2.1	Understand DeepForest	William, Vaibhavi	9/25/23	10/3/23
2.2	Understand DetectTree	Manan, Yaaminie	9/25/23	10/3/23
2.3	Find Other Tools	Martin	9/25/23	10/3/23
3	**Presentations**			
3.1	Mid-term Presentation	All	10/5/23	10/12/23
3.2	Case Study	All	10/22/23	10/31/23
4	**Project Conception and Initiation**			
4.1	Data Collection Cont.	Martin	10/9/23	10/20/23
4.2	Resampling and training data selection	Vaibhavi, Yaaminie	10/20/23	11/2/23
5	**Training the model**			
5.1	Preparing DeepForest	All	11/3/23	11/9/23
5.2	Labeling	All	11/8/23	11/10/23
5.3	Label Fixing	Manan	11/10/23	11/13/23
5.4	Train Model	Manan	11/10/23	11/20/23
6	**Testing/Evaluating the model**			
6.1	Validate Model	William	11/17/23	11/27/23
6.2	Testing Model	Manan	11/8/23	11/21/23
6.3	Final Report and Presentation	All	11/21/23	12/1/23



The GANTT chart is in the folder in addition to the following link: https://docs.google.com/spreadsheets/d/1OjgqV3Ug_HcFn9W0uZlG8wpbg5B9AWLI1GT9xFKq_dw/edit?usp=sharing

3.2. Challenges

3.2.1. Learning GIS

This project was difficult for our team because we needed Machine Learning techniques and then also Geographic Information System (GIS) technology on the fly. There was a steep learning curve where we spent substantial time at the beginning of the project learning terminology and how to gather the aerial imagery we needed. Also, as part of learning new GIS technology, we needed to work with several different data formats that we never encountered before.

3.2.2. Low-Resolution Aerial Imagery

Lower resolution made it harder for both the hand labeling and final detection phases of the project. Labelers had difficulty distinguishing trees from other objects. Lower resolution also may have hurt the final quality of results, as more pixels per meter would allow more data for the model to process.

3.2.3. Tile Selection

Large dimensions to capture details in the image make it difficult to visualize the clusters in the K-means algorithm. Trial and error to select the number of components of the gist descriptors with PCA algorithm to capture at least 90% variance among the images.

3.2.4. Labeling

Labeling aerial imagery took a significant amount of time and resources while also being difficult to do accurately. Differences in color and shape often caused second-guessing by labelers, as at times it could not be determined if it was a tree or bush. Overlapping trees in forests was also a problem, as it was difficult to distinguish between individual trees.

3.2.5. Coordinate Conversions

Using the provided EPSG:3857 caused problems as the format is in meters but as a projection from the center of the world map. This caused issues when trying to calculate tree canopy size, as we calculated values significantly off. We needed to convert to EPSG:4326 and then use a library to find the difference in the coordinate points in that system to correctly calculate tree canopy size.

3.3. Learning Outcomes

3.3.1. How to Select the Most Representative Data

We learned how unsupervised learning can be used over random or manual selection to form the training set with the most representative data to fine-tune the deep forest algorithm. In this process, we also learned that to capture features of the image, a high-dimensional vector representation is used. For computational efficiency, we used PCA to reduce dimensionality, followed by K-means to select images closest to centroids to form the training set.

3.3.2. How to Fine-tune a Deep Learning Model

We tried to utilize transfer learning techniques to build upon a proven model for detecting tree crowns. We realized that although the model was good, it was not well suited to detecting trees in urban settings. The fine-tuning process simply involved taking the pre-trained deep neural network with its pre-calculated parameters and retraining it on our data set. Usually, the output layer retrains from scratch, however, the model uses the new data and updates the existing weights to better suit the new data.

References

[1] City of San José, “Community Forest Management Plan | City of San José,” www.sanjoseca.gov, 2021. https://www.sanjoseca.gov/your-government/departments-offices/transportation/landscaping/trees/community-forest-management-plan (accessed Dec. 1, 2023).

[2] City of San Jose - Office of the City Auditor, “City of San José - File #: 23-016,” sanjose.legistar.com, 2022. https://sanjose.legistar.com/LegislationDetail.aspx?ID=5971493&GUID=150C3E74-F97B-4C41-8BF0-C2AC14D69077 (accessed Dec. 1, 2023).

[3] tsungi,“wms-tile-get” 2021. https://github.com/easz/wms-tile-get (accessed Dec. 1, 2023).

[4] tsungi, “urban-tree” 2023. https://github.com/easz/urban-tree (accessed Dec. 1, 2023).

[5] Lin Yang, Xiaqing Wu, Emil Praun, and Xiaoxu Ma, “Tree detection from aerial imagery,” presented at the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Seattle, WA, USA, Nov. 2009. [Online]. Available:

https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/35659.pdf
[6] M. Bosch, “DetecTree: Tree detection from aerial imagery in Python,” Journal of Open Source Software, vol. 5, no. 50, p. 2172, Jun. 2020, doi: 10.21105/joss.02172.
Library - https://detectree.readthedocs.io/en/latest/train_test_split.html
[7] tsungi, “urban-tree” 2023. https://github.com/easz/urban-tree (accessed Dec. 1, 2023).
[8] B. G. Weinstein, S. Marconi, S. Bohlman, A. Zare, and E. White, “Individual Tree-Crown Detection in RGB Imagery Using Semi-Supervised Deep Learning Neural Networks,” Remote Sensing, vol. 11, no. 11, p. 1309, Jan. 2019, doi: 10.3390/rs11111309.
[9] B. G. Weinstein, S. Marconi, S. A. Bohlman, A. Zare, and E. P. White, “Geographic Generalization in Airborne RGB Deep Learning Tree Detection.” bioRxiv, Oct. 02, 2019. doi: 10.1101/790071.
[10] “Map Services,” wildlife.ca.gov. https://wildlife.ca.gov/Data/GIS/Map-Services (accessed Dec. 1, 2023).