Green AI Challenge

proposed solution by Team

“Clear Vision”

Our Works : https://github.com/Mahos-H/ClearVision


NATURE IS NOT A
PLACE TO VISIT,
IT IS HOME.
–GARY SNYDER

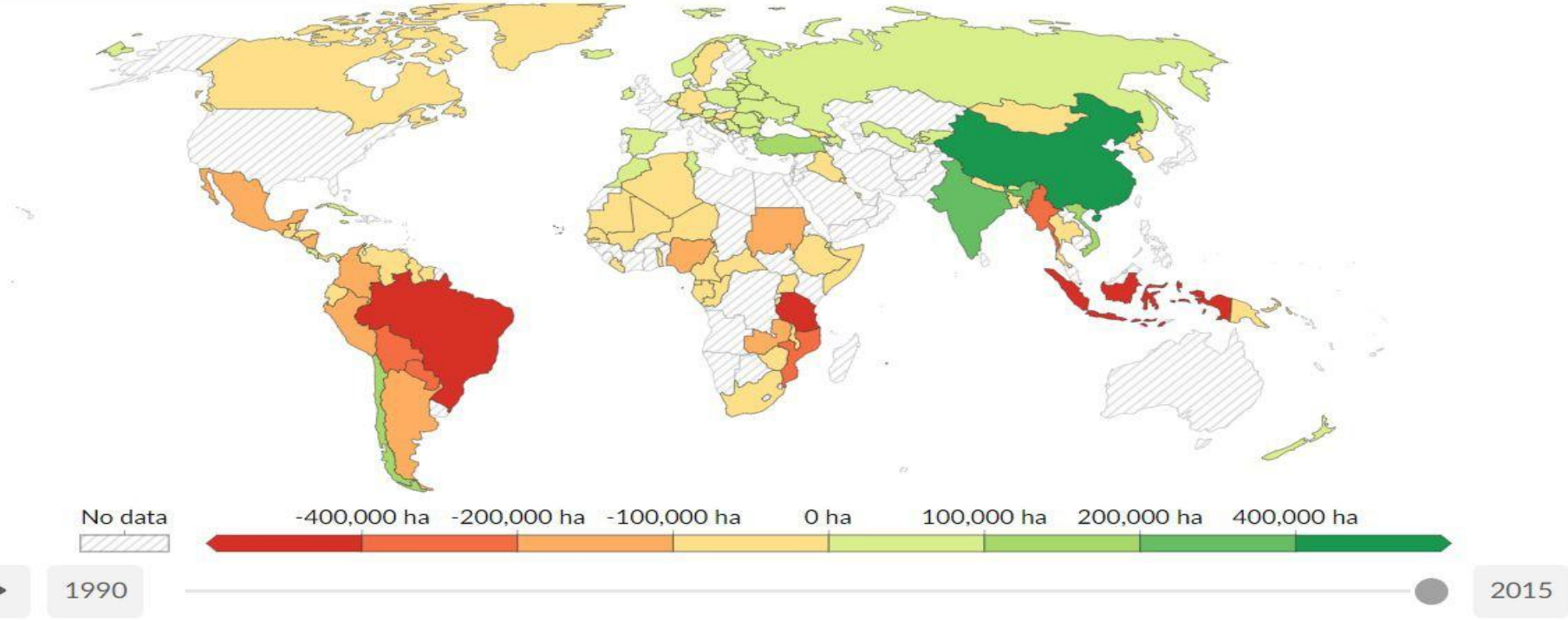
Annual change in forest area, 2015

Net change in forest area measures forest expansion (either through afforestation or natural expansion) minus deforestation

Our World in Data

Table | Map | Chart

World



https://ourworldindata.org/grapher/annual-change-forest-area?time=2015

Problem Statement:

Using ML/AI to compare drone images of an afforestation patch and point out casualties.

Odisha forest department plants nearly 5 crore trees every year across the state. Afforestation programs are taken up over patches of irregular shapes and sizes and models that depend on the existing vegetation density of the patch.

Post OP 1

OP1: During March-to-May period of Year 1, pits of size 45x45x45cm are dug up and left to the sun to kill any microbes. This is carried out by manual labor or digging machines (JCBs) with fixed size buckets and is easily visible from the sky.

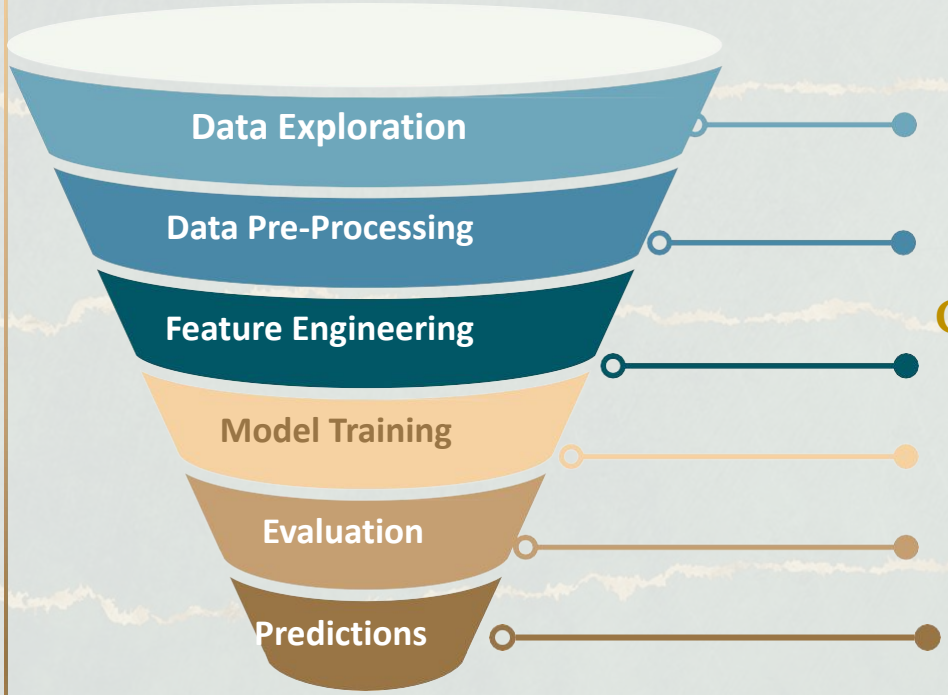
Post OP2

OP2: During the first monsoon rains and subsequent month (June 15 to July 30), planting of saplings is carried out. The saplings are 18 months old and are 4 to 6 ft tall depending on the species with minimal foliage.

Post OP3

OP3: During Oct-Nov of Year 1, for weeding, a diameter of 1m soil is cleared around the sapling, which is visible from the sky. The same operation is conducted during Year 2 and Year 3 during the same period

Pipeline for the Project



Open Drone Mapping, Pix4D, Python Libraries

Geodata Extraction, Pit Detection and Indexing

OpenCV, K Means Clustering, Rasterio, Scikit-Images

NDVI Sapling Detection, Health Estimation

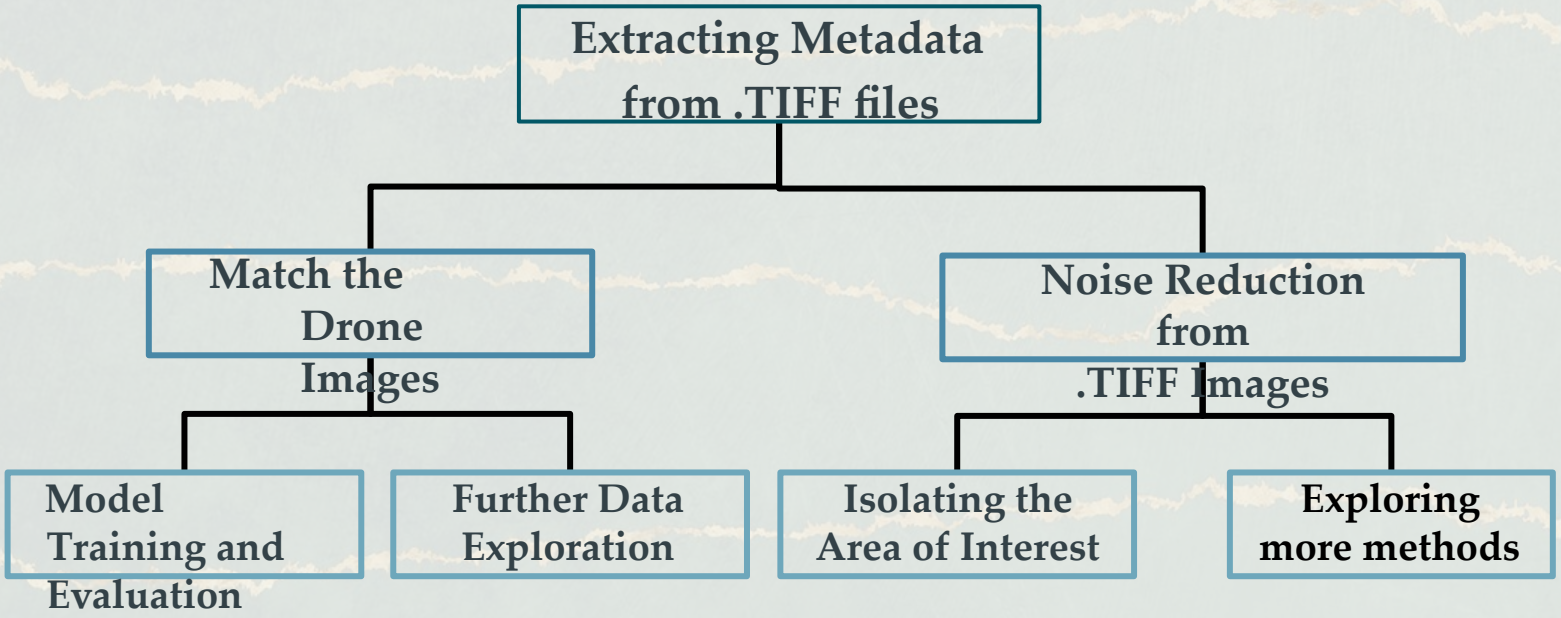
Mixed Precision evaluation + Health Embeddings

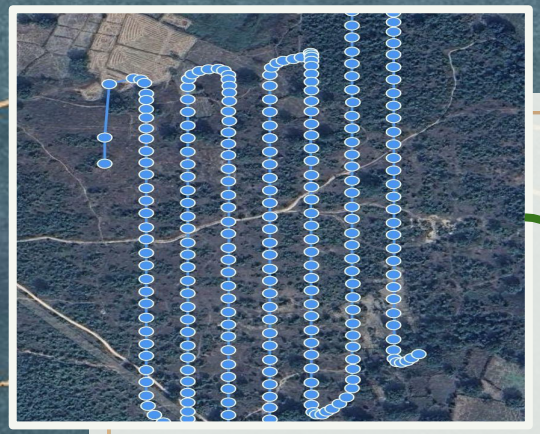
Survival Rate Calculation, Possible Factors of Death

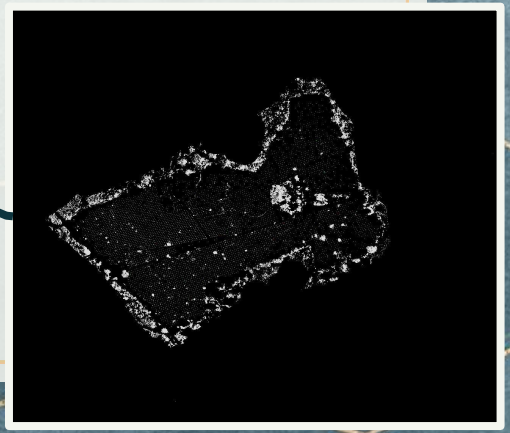
Step 1 : Understanding the Dataset

The main challenge we faced.

FlowChart of Data-Exploration







Mapping the small images with the Drone Path and reconstructing the pits in the plot

Software toolkits used for Data Exploration

```
pre_pitting_Benkmura = pd.read_csv("./Benkmura CSVs/post_pitting_metadata.csv")
post_pitting_Benkmura = pd.read_csv("./Benkmura CSVs/post_planting_metadata.csv")
post_planting_Benkmura = pd.read_csv("./Benkmura CSVs/post_sw_metadata.csv")
post_sw_Benkmura = pd.read_csv("./Benkmura CSVs/pre_pitting_metadata.csv")
```

```
post_pitting_Debadihi = pd.read_csv("./Debadihi CSVs/post_pitting_metadata.csv")
post_planting_Debadihi = pd.read_csv("./Debadihi CSVs/post_planting_metadata.csv")
post_sw_Debadihi = pd.read_csv("./Debadihi CSVs/post_SW_metadata.csv")
```

```
# Add location column to the original dataframes
pre_pitting_Benkmura['Location'] = 'Benkmura'
post_pitting_Benkmura['Location'] = 'Benkmura'
post_planting_Benkmura['Location'] = 'Benkmura'
post_sw_Benkmura['Location'] = 'Benkmura'

post_pitting_Debadihi['Location'] = 'Debadihi'
post_planting_Debadihi['Location'] = 'Debadihi'
post_sw_Debadihi['Location'] = 'Debadihi'
```

```
# Recreate combined dataframes with location info
combined_post_pitting = pd.concat([post_pitting_Benkmura, post_pitting_Debadihi])
combined_post_planting = pd.concat([post_planting_Benkmura, post_planting_Debadihi])
combined_post_sw = pd.concat([post_sw_Benkmura, post_sw_Debadihi])
```

Drone Mapping and Image Analysis Using ODM

Overview

We use Open Drone Mapping (ODM) to track drone flight paths, stitch images accurately, and create image masks for identifying pit holes within plots.

Pros :--

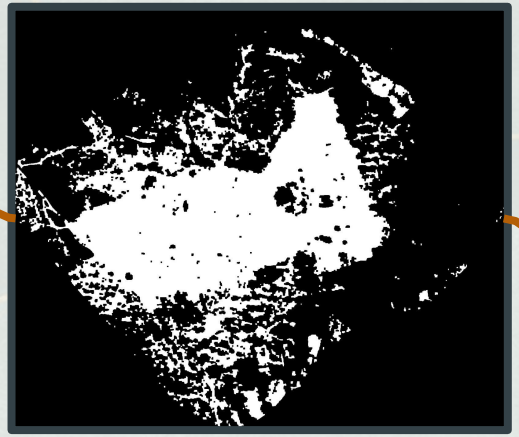
1. **Enhanced Accuracy: Precise mapping improves the quality of flight data.**
2. **Increased Efficiency: Automated stitching saves time and labor efforts.**
3. **Thorough Analysis: Image masks ensure detailed pit examinations.**
4. **Better Visualization: Isolating pits enhances clarity and understanding.**
5. **Versatility: Adaptable to various terrains and plot types.**
6. **Comprehensive Data Integration: Allows merging with other datasets for holistic analysis.**

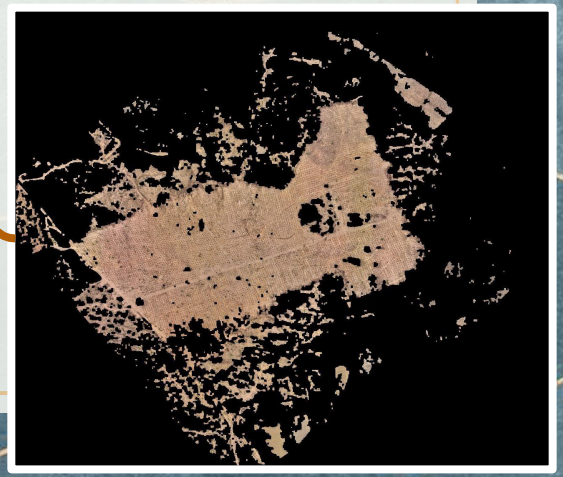
Cons :--

1. Inconsistent Image Alignment: Altitude and angle changes disrupt image matching.
2. Time-Consuming Manual Stitching: Stitching many images manually takes significant amount of time.
3. Overlapping Pixels: Overlaps cause confusion in sub-images.

Step 2 : Isolating the Useful Areas using K Means

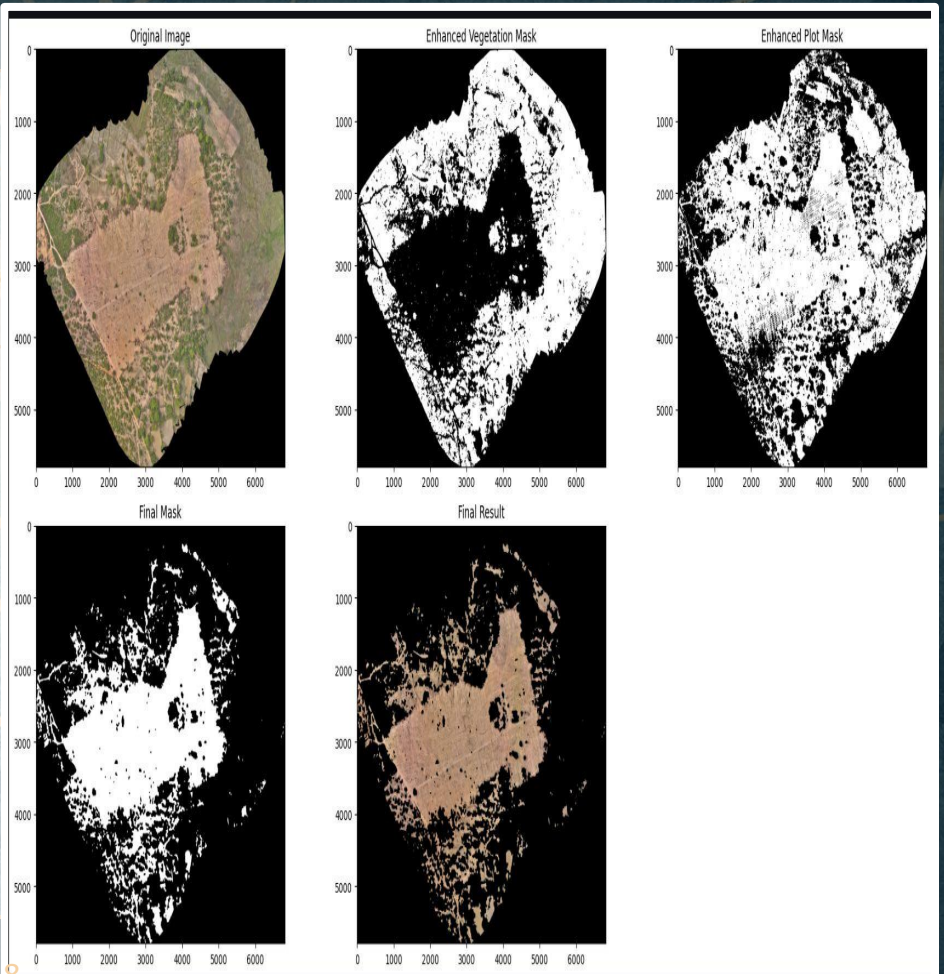
Pre-Processing the Data





Isolation of Area of Interest for Pitt Numbering and Comparison

```
def create_enhanced_vegetation_mask(self, img):
    """
    Create vegetation mask using color spaces and KMeans clustering
    """
    # Convert to multiple color spaces
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)

    # Initial vegetation mask using traditional method
    green_ranges = [
        (np.array([35, 40, 40]), np.array([85, 255, 255])),  # Bright green
        (np.array([25, 40, 0]), np.array([85, 255, 60])),    # Dark green
        (np.array([25, 40, 40]), np.array([85, 255, 255])),  # Yellow-green
        (np.array([20, 30, 30]), np.array([85, 180, 180]))   # Olive green
    ]

    # Create initial vegetation mask
    init_mask = np.zeros(img.shape[:2], dtype=np.uint8)
    for lower, upper in green_ranges:
        mask = cv2.inRange(
        init_mask = cv2.bit

    # Prepare data for
    features = np.concatena
        hsv.reshape(-1, 3),
        lab.reshape(-1, 3)
    ], axis=1)

    # Initialize KMeans
    kmeans = KMeans(n_clust

    # Fit on pixels that
    veg_pixels = features[i
    if len(veg_pixels) > 0:
        kmeans.fit(veg_pixe

        # Find distances
        distances = cdist(f
        min_distances = np.

        # Create threshold
        distance_threshold

        # Create enhanced
        enhanced_mask = {m
        enhanced_mask = en

        # Clean up the mask
        kernel = np.ones((5
        enhanced_mask = cv2
        enhanced_mask = cv2

        return enhanced_mas

    return init_mask

def create_plot_mask(self,
    """
    Create mask for the agr
    """
    # Convert to HSV
    hsv = cv2.cvtColor(img,

    # Initial brown mask us
    lower_brown = np.array(
    upper_brown = np.array(
    init_brown_mask = cv2.i

    # Prepare features for
    features = hsv.reshape(

    # Initialize KMeans
    kmeans = KMeans(n_clust

    # Fit on pixels that
    plot_pixels = features[init_brown_mask.flatten() > 0]
    if len(plot_pixels) > 0:
        kmeans.fit(plot_pixels)

        # Find distances to centroids for all pixels
        distances = cdist(features, kmeans.cluster_centers_)
        min_distances = np.min(distances, axis=1)

        # Create threshold for identifying similar pixels
        distance_threshold = np.percentile(min_distances[init_brown_mask.flatten() > 0], 75)

        # Create enhanced mask including similar pixels
        enhanced_mask = (min_distances < distance_threshold).reshape(img.shape[:2])
        enhanced_mask = enhanced_mask.astype(np.uint8) * 255
```



K-Means Clustering for Region of Interest Detection

Overview

Incorporated KMeans Clustering Algorithm along with CV2 to isolate green patches from yellow ones using a black-and-white mask for clear differentiation.

Pros

1. Easy Edge Detection: Simplifies identifying edges within images.
2. Simple Algorithm: KMeans is easy to implement and adapt.
3. Time-Efficient: Optimizes processing for quicker results.

Cons

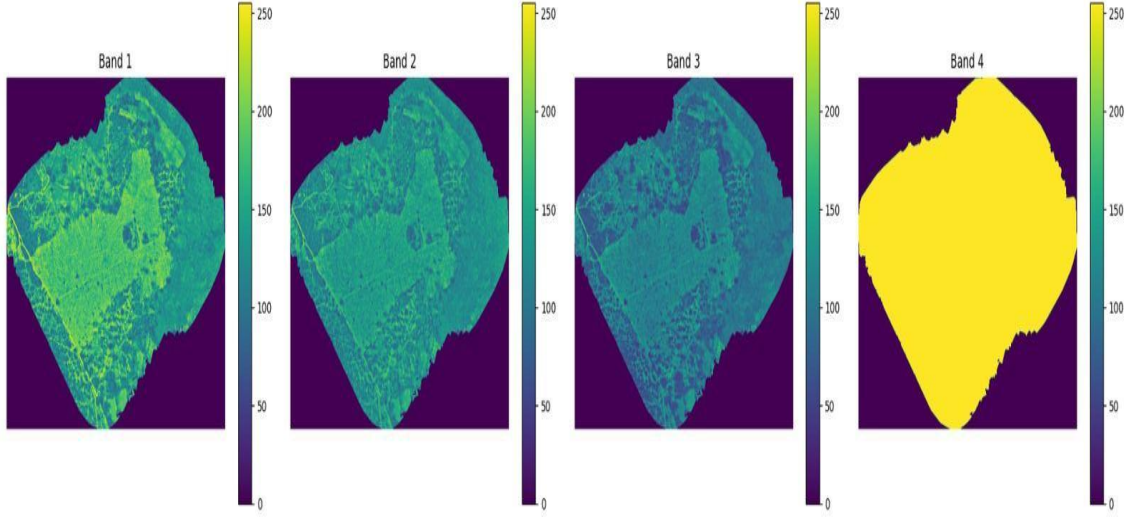
1. Jittery Boundaries: Patch edges may appear inconsistent.
2. Boundary Noise: Noise at edges can reduce segmentation accuracy.

Step 3 : Indexing the Pitts for Model Training

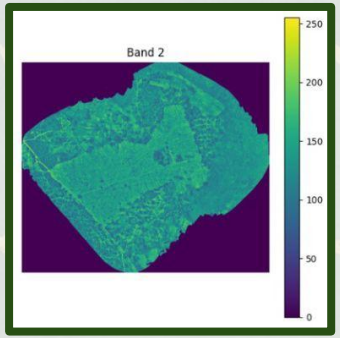
Feature Engineering Step

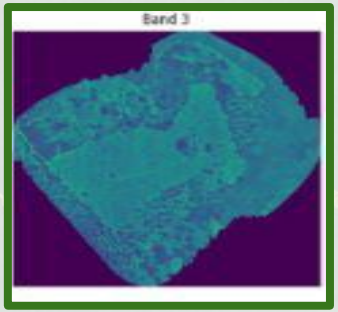
Using Rasterio to separate the Color Bands of the .Tiff file

```python
def extract_raster_bands(raster_path):
    with rasterio.open(raster_path) as src:
        # Get basic metadata
        print(f"Number of bands: {src.count}")
        print(f"Width: {src.width}")
        print(f"Height: {src.height}")
        print(f"Coordinate Reference System: {src.crs}")
        print(f"Transform: {src.transform}")

        # Read all bands
        bands = []
        for i in range(src.count):
            band = src.read(i + 1)
            bands.append(band)

            # Print band statistics
            print(f"\nBand {i + 1} statistics:")
            print(f"Min: {band.min()}")
            print(f"Max: {band.max()}")
            print(f"Shape: {band.shape}")

        # Vi
        fig,
        if src.count == 1:
            axes = [axes]

        for
            im = a
            ax.set_title(f'Band {idx + 1}')
            fig.colorbar(im, ax=ax)

            # Save individual band
            plt.imsave(f'band_{idx+1}
                       cmap='viridis')

        plt.tight_layout()
        plt.savefig('all_bands.png')
        plt.show()

        return bands

# Use the function
raster_path = 'BenkmuraPostSW.tif'  # Replace with your raster file path
bands = extract_raster_bands(raster_path)

# If you want to work with specific bands
# For example, to create a composite of bands 4,3,2:
if len(bands) >= 4:
    rgb = np.dstack((bands[3], bands[2], bands[1]))  # bands3,2

    plt.figure(figsize=(10, 10))
    plt.imshow(rgb/rgb.max())  # Normalize values for display
    plt.title('RGB Composite (4-3-2)')
    plt.axis('off')
    plt.savefig('rgb_composite.png')
    plt.show()
```

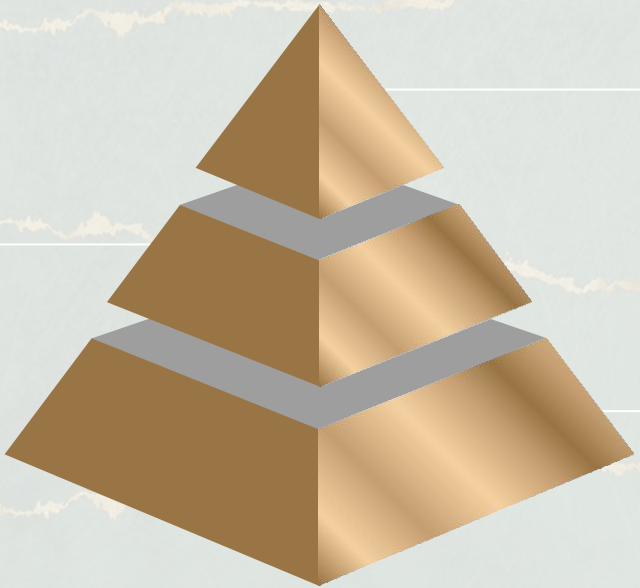


Color Band Extraction

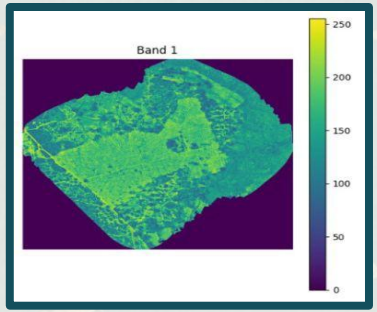




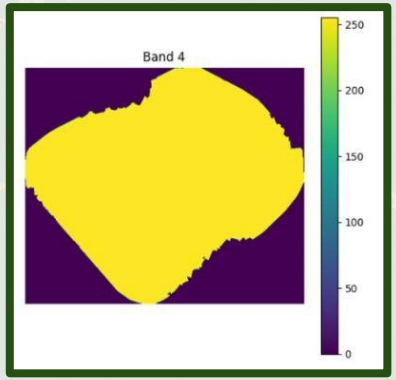
2



1



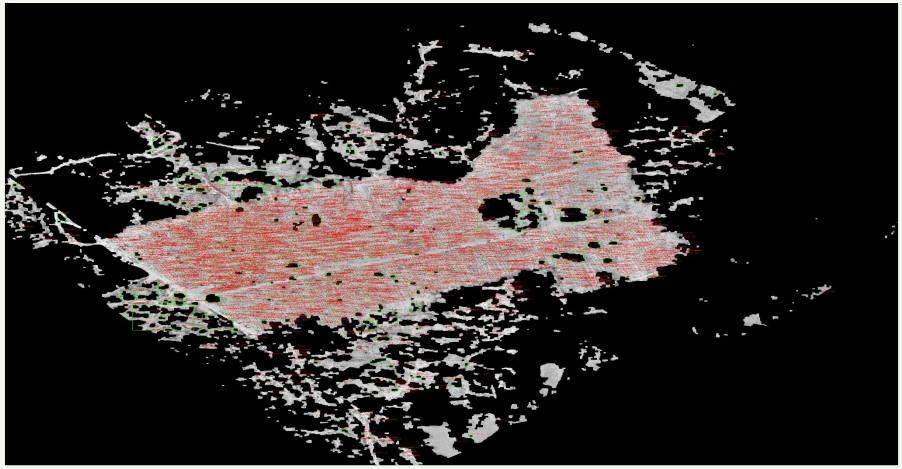
4

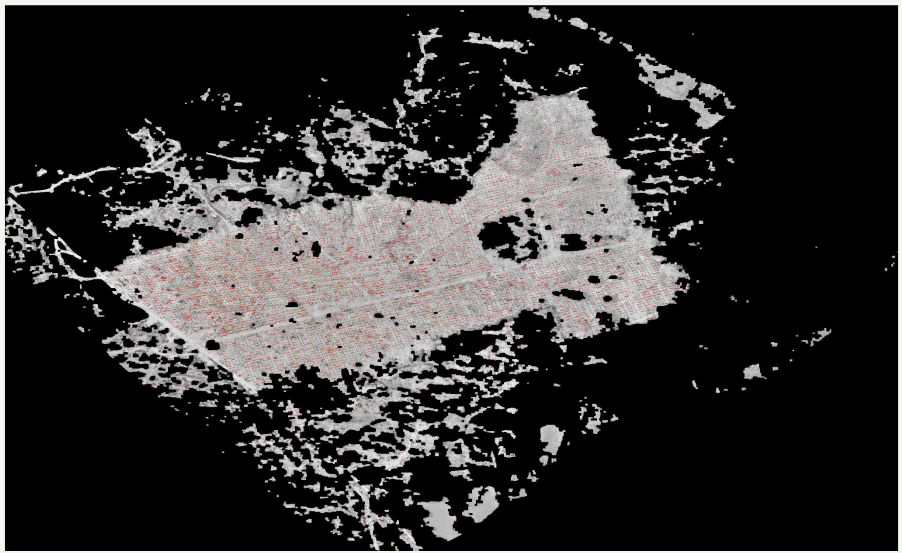


```python
# Constants
BLACK_THRESH = 100
WHITE_THRESH = 150
BORDER_SIZE = 3
WHITE_BORDER_RATIO = 0.2
MAX_PIT_SIZE = 10  # Maximum pit size in pixels

def check_pit_border(image, region, border_size=BORDER_SIZE):
    """
    Check if a dark region is surrounded by whitish pixels and meets size constraints.
    """
    minr, minc, maxr, maxc = region.bbox
    height, width = region.image.shape

    # Check pit size
    if (maxr - minr) > MAX_PIT_SIZE or (maxc - minc) > MAX_PIT_SIZE:
        return False

    # Get expanded border region
    border_minr = max(minr - border_size, 0)
    border_minc = max(minc - border_size, 0)
    border_maxr = min(maxr + border_size, image.shape[0])
    border_maxc = min(maxc + border_size, image.shape[1])

    border_region = image[border_minr:border_maxr, border_minc:border_maxc]

    # Create border mask
    border_mask = np.ones((border_maxr - border_minr, border_maxc - border_minc), dtype=bool)
    pit_mask = region.image
    border_mask[pit_mask] = False

    # Check if pit area is dark enough
    pit_area = image[minr:maxr, minc:maxc][pit_mask]
    is_dark_enough = np.mean(pit_area) < BLACK_THRESH

    # Check if border is whitish enough
    whitish_pixels = np.sum(border_region[border_mask] > WHITE_THRESH)
    total_border_pixels = np.sum(border_mask)
    white_ratio = whitish_pixels / total_border_pixels if total_border_pixels > 0 else 0

    return is_dark_enough and white_ratio > WHITE_BORDER_RATIO

def detect_and_label_pits(image_path):
    """
    Detect pits with size constraints and smaller labels.
    """
    with rasterio.open(image_path) as src:
        image = src.read(1)

    image = cv2.normalize(image, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8U)
    dark_regions = image < BLACK_THRESH
    labeled_regions = label(dark_regions)
    regions = regionprops(labeled_regions)

    # Filter regions
    valid_regions = [region for region in regions if check_pit_border(image, region)]

    image_rgb = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
    coordinates = []
```

Pit Detection Algorithm

using CV2 after the color bandas have be optimised





Naive Green Check along with ConvMixer / NDVI Fusion:

Goal: Detect 45×45×45 cm pits in the above Pitt-Detected picture.

Implementation:

1. **Simple Threshold & Labeling:**
   - **A thresholding technique identifies dark pixels.**
   - **Connected-component labeling is used to outline pit shapes.**

Advanced Segmentation:

1. Conv-Mixer or MobileNet has been used for compressed segmentation.

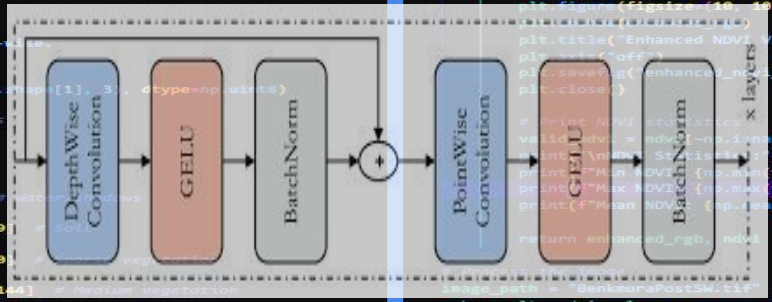
2. Generates binary masks for pits and computes centroids for precise localization.

3. This approach blends simplicity with advanced segmentation techniques for efficient and accurate pit detection.

Step 4 : Sapling Detection and Health Calculation

Model Training Phase

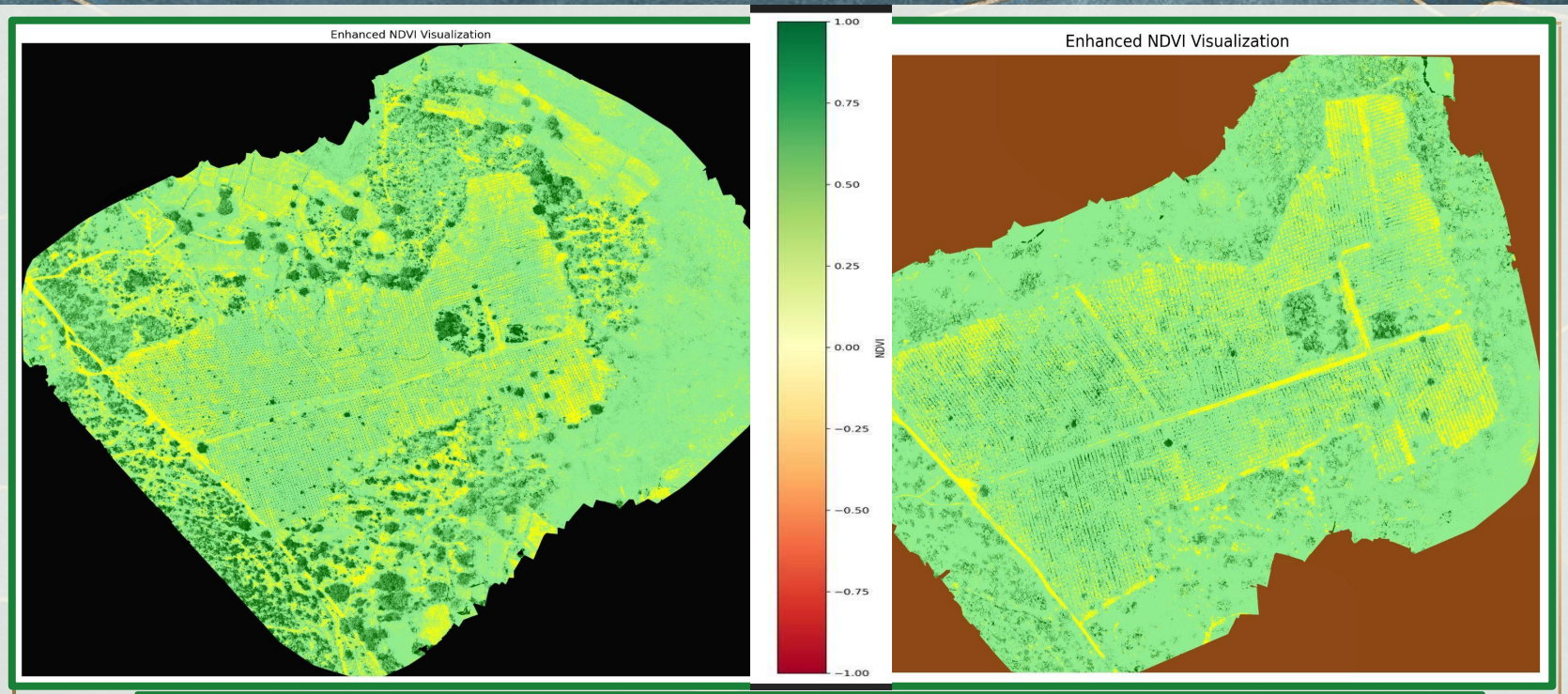
```python
import rasterio    Import "rasterio" could not be resolved
import numpy as np
import matplotlib.pyplot as plt
import cv2

def compute_ndvi_pixelwise(red_band, nir_band):
    """
    Compute NDVI for each pixel with precise handling of values.
    """
    epsilon = 1e-10  # To prevent division by zero
    denominator = nir_band + red_band + epsilon

    # NDVI computation
    ndvi = (nir_band - red_band) / denominator

    # Clip NDVI values to the range [-1, 1]
    return np.clip(ndvi, -1, 1)

def apply_pixelwise_color(ndvi):
    """
    Map NDVI values to RGB colors pixel-wise.
    """
    # Create RGB array
    rgb = np.zeros((ndvi.shape[0], ndvi.shape[1], 3), dtype=np.uint8)

    # Map NDVI ranges to specific colors
    for i in range(ndvi.shape[0]):
        for j in range(ndvi.shape[1]):
            value = ndvi[i, j]
            if value <= -0.2:
                rgb[i, j] = [0, 0, 0]  # Water/Shadows
            elif value <= 0.1:
                rgb[i, j] = [139, 69, 19]  # Soil
            elif value <= 0.3:
                rgb[i, j] = [255, 255, 0]  # Sparse vegetation
            elif value <= 0.6:
                rgb[i, j] = [144, 238, 144]  # Medium vegetation
            else:
                rgb[i, j] = [0, 100, 0]  # Dense vegetation
    return rgb

def enhance_contrast_locally(image, grid_size=(4, 4)):
    """
    Apply CLAHE for local contrast enhancement with a fine filter.
    """
    lab = cv2.cvtColor(image, cv2.COLOR_RGB2LAB)
    l, a, b = cv2.split(lab)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=grid_size)
    cl = clahe.apply(l)
    enhanced_lab = cv2.merge((cl, a, b))
    return cv2.cvtColor(enhanced_lab, cv2.COLOR_LAB2RGB)
```



ConvMixer Model

```python
def process_image_optimized(image_path):
    with rasterio.open(image_path) as src:
        # Read red and NIR bands (assumes 3 = red, 4 = NIR)
        red = src.read(3).astype(np.float32)
        nir = src.read(4).astype(np.float32)

        # Compute NDVI pixel-wise
        ndvi = compute_ndvi_pixelwise(red, nir)

        # Map NDVI values to RGB pixel-wise
        rgb_ndvi = apply_pixelwise_color(ndvi)

        # Enhance contrast locally with a 4x4 grid filter
        enhanced_rgb = enhance_contrast_locally(rgb_ndvi, grid_size=(4, 4))

        # Display and save the enhanced image
        plt.figure(figsize=(10, 10))
        plt.title("Enhanced NDVI Visualization")
        plt.axis("off")
        plt.savefig("enhanced_ndvi_pixelwise.png", dpi=300, bbox_inches="tight")
        plt.close()

        # Print NDVI statistics
        valid_ndvi = ndvi[~np.isnan(ndvi)]
        print("\nNDVI Statistics:")
        print(f"Min NDVI: {np.min(valid_ndvi):.3f}")
        print(f"Max NDVI: {np.max(valid_ndvi):.3f}")
        print(f"Mean NDVI: {np.mean(valid_ndvi):.3f}")

        return enhanced_rgb, ndvi

image_path = "BenkmuraPostSw.tif"
rgb_result, ndvi_values = process_image_optimized(image_path)

# Create a legend for NDVI visualization
fig, ax = plt.subplots(figsize=(10, 2))
categories = ['Water/Shadows', 'Soil', 'Sparse Vegetation',
              'Medium Vegetation', 'Dense Vegetation']
colors = [[0, 0, 0], [139/255, 69/255, 19/255], [1, 1, 0],
          [144/255, 238/255, 144/255], [0, 100/255, 0]]
patches = [plt.Rectangle((0, 0), 1, 1, fc=color) for color in colors]
plt.legend(patches, categories, loc='center', ncol=5)
plt.axis("off")
plt.savefig("ndvi_legend_pixelwise.png", dpi=300, bbox_inches="tight")
plt.close()
```

+++++
+++++
+++++



NDVI Index Output of Post-Planting and Post-SW

Normalized Difference Vegetation Index (NDVI)

Overview

NDVI measures vegetation presence and health, commonly used in remote sensing.

How It Works:

- **Data Source: Uses satellite data to measure visible and near-infrared light reflection.**
- **Value Range: NDVI values range from -1 to 1:**
  - **Negative Values: Indicate water or clouds.**
  - **Positive Values: Indicate vegetation, with higher values representing denser vegetation.**

Purpose:

An essential tool for monitoring vegetation health and density.

NDVI Formula: NDVI = (NIR+Red) / (NIR–Red)

- **NIR: Reflectance of near-infrared light**

  (healthy vegetation reflects more NIR).

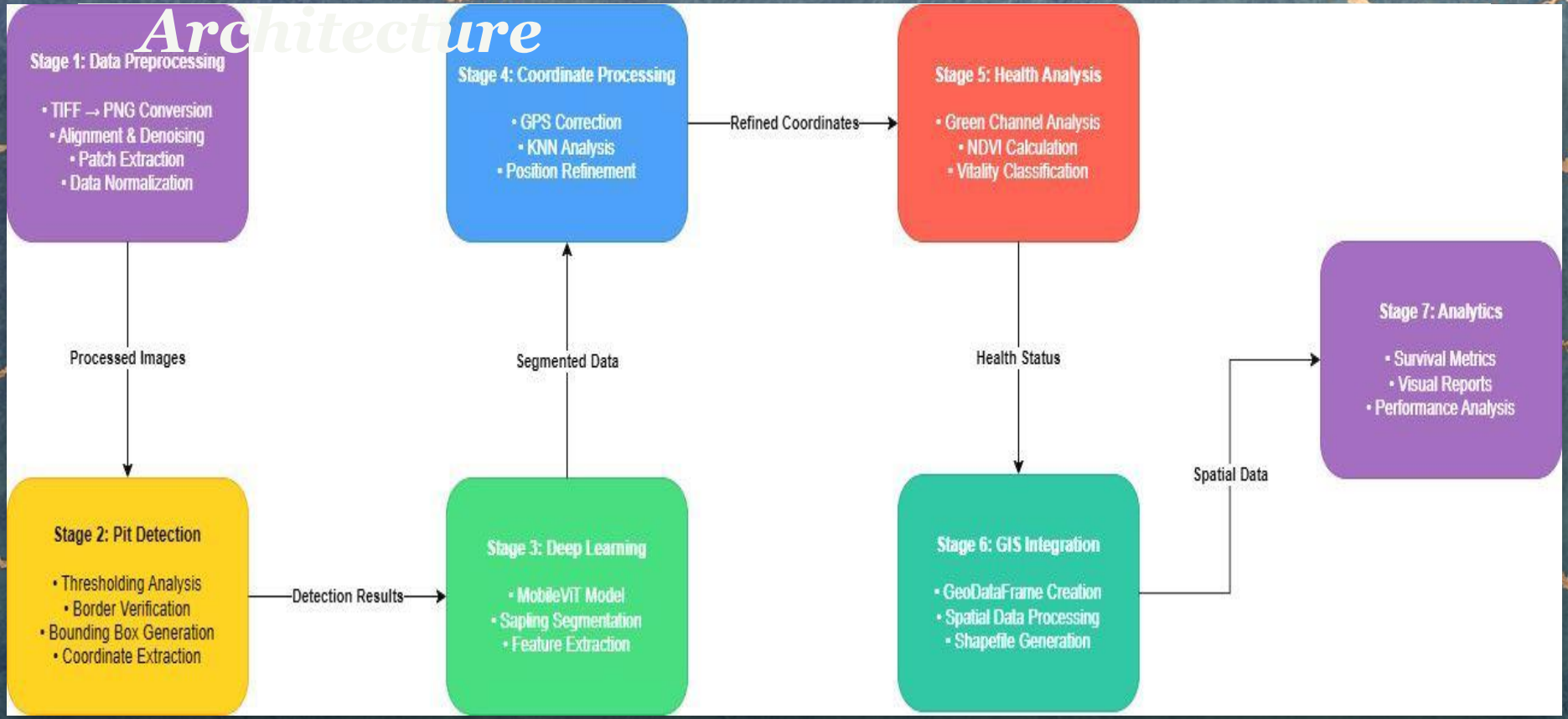
- **Red: Reflectance of red light**

  (absorbed more by chlorophyll in healthy vegetation).

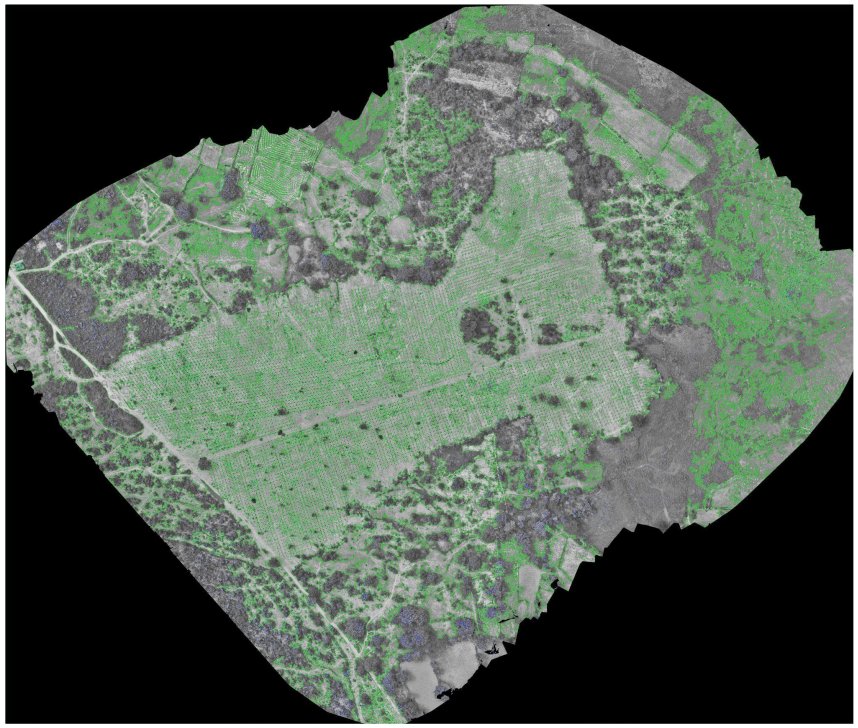
Why It Works:

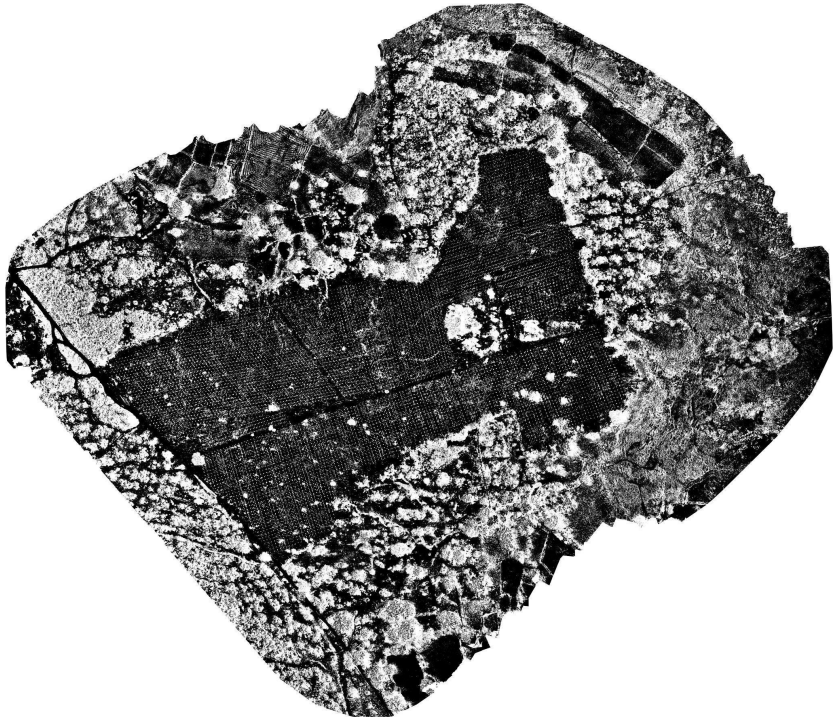
Healthy vegetation strongly reflects NIR and absorbs red light, making NDVI a reliable measure of vegetation density and health.

Complete Pipeline of the Model Architecture



Stage 2 and Stage 3 Results



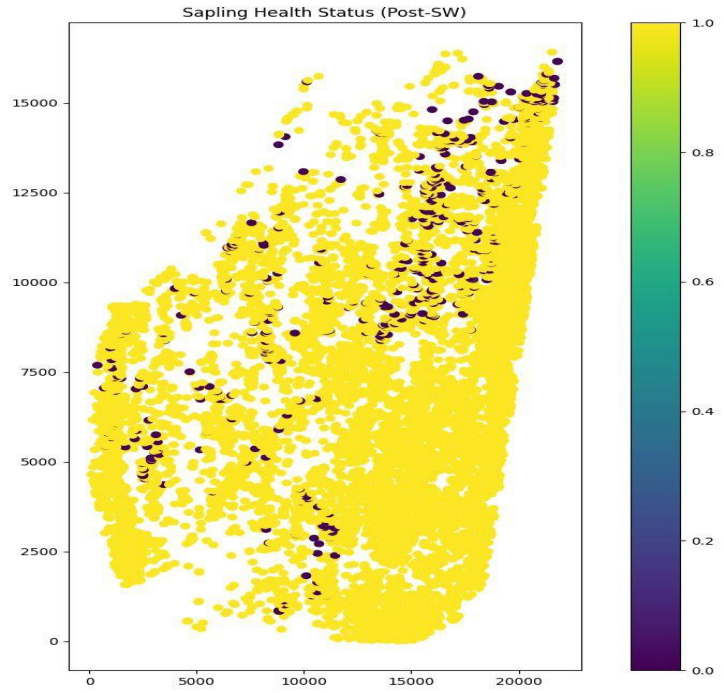


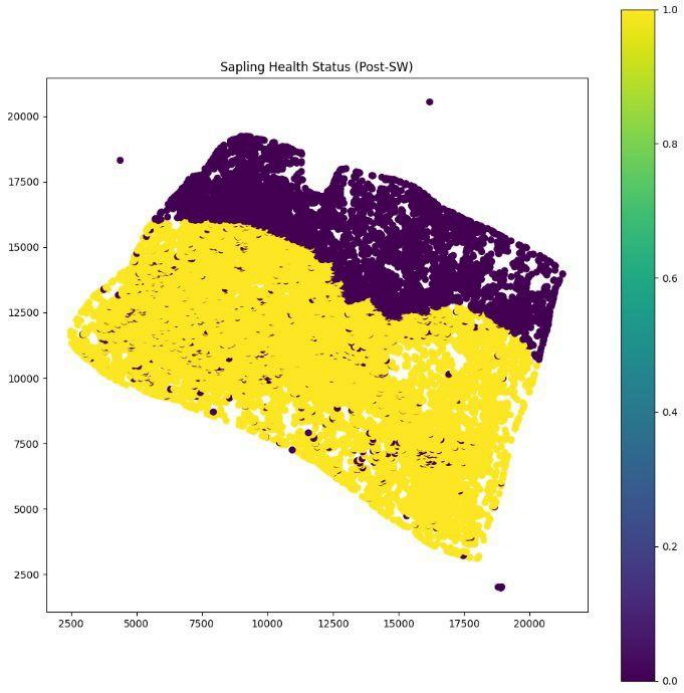
Stage 4 Results

D4 | 83.8213498611111

	A	B	C	D	E	F
1	Filename	Timestamp	Latitude	Longitude	Altitude	Location
2	DJI_20241228160059_0001_V.JPG	1.73538E+12	21.65479238888889	83.82134808333333	207.083	Benkmura
3	DJI_20241228160100_0002_V.JPG	1.73538E+12	21.65477458333333	83.82134883333333	207.077	Benkmura
4	DJI_20241228160101_0003_V.JPG	1.73538E+12	21.65471408333333	83.82134986	207.054	Benkmura
5	DJI_20241228160103_0004_V.JPG	1.73538E+12	21.65460858333333	83.82135047222222	206.983	Benkmura
6	DJI_20241228160104_0005_V.JPG	1.73538E+12	21.65448730555556	83.82135041666666	206.93	Benkmura
7	DJI_20241228160106_0006_V.JPG	1.73538E+12	21.65435961111111	83.82134991666666	206.913	Benkmura
8	DJI_20241228160107_0007_V.JPG	1.73538E+12	21.654241638888887	83.82135091666666	206.997	Benkmura
9	DJI_20241228160108_0008_V.JPG	1.73538E+12	21.654114055555556	83.82135066666666	207.027	Benkmura
10	DJI_20241228160110_0009_V.JPG	1.73538E+12	21.653996416666665	83.82135036	207.051	Benkmura
11	DJI_20241228160111_0010_V.JPG	1.73538E+12	21.65386958333333	83.82135038888889	207.07	Benkmura
12	DJI_20241228160112_0011_V.JPG	1.73538E+12	21.65374283333333	83.82135008333333	207.07	Benkmura
13	DJI_20241228160114_0012_V.JPG	1.73538E+12	21.65362502777777	83.82134952777777	207.077	Benkmura
14	DJI_20241228160115_0013_V.JPG	1.73538E+12	21.653498027777776	83.82135036	207.074	Benkmura
15	DJI_20241228160116_0014_V.JPG	1.73538E+12	21.653380277777774	83.82135163888888	207.078	Benkmura
16	DJI_20241228160118_0015_V.JPG	1.73538E+12	21.653252694444443	83.82135122222222	207.071	Benkmura
17	DJI_20241228160119_0016_V.JPG	1.73538E+12	21.65312675	83.82134994444444	207.043	Benkmura
18	DJI_20241228160120_0017_V.JPG	1.73538E+12	21.65300908333333	83.82135522222222	206.96	Benkmura
19	DJI_20241228160122_0018_V.JPG	1.73538E+12	21.652920305555558	83.82139638888889	206.991	Benkmura
20	DJI_20241228160123_0019_V.JPG	1.73538E+12	21.652871694444443	83.82148183333332	206.983	Benkmura
21	DJI_20241228160125_0020_V.JPG	1.73538E+12	21.652854583333333	83.82156622222222	206.921	Benkmura
22	DJI_20241228160126_0021_V.JPG	1.73538E+12	21.652898138888887	83.82161016666666	207.029	Benkmura
23	DJI_20241228160127_0022_V.JPG	1.73538E+12	21.65296397222222	83.82161613888888	207.109	Benkmura
24	DJI_20241228160129_0023_V.JPG	1.73538E+12	21.65306625	83.82161677777778	207.199	Benkmura
25	DJI_20241228160130_0024_V.JPG	1.73538E+12	21.653186194444444	83.82161652777778	207.148	Benkmura
26	DJI_20241228160131_0025_V.JPG	1.73538E+12	21.653315416666665	83.82161666666666	207.008	Benkmura

Stage 5 Results





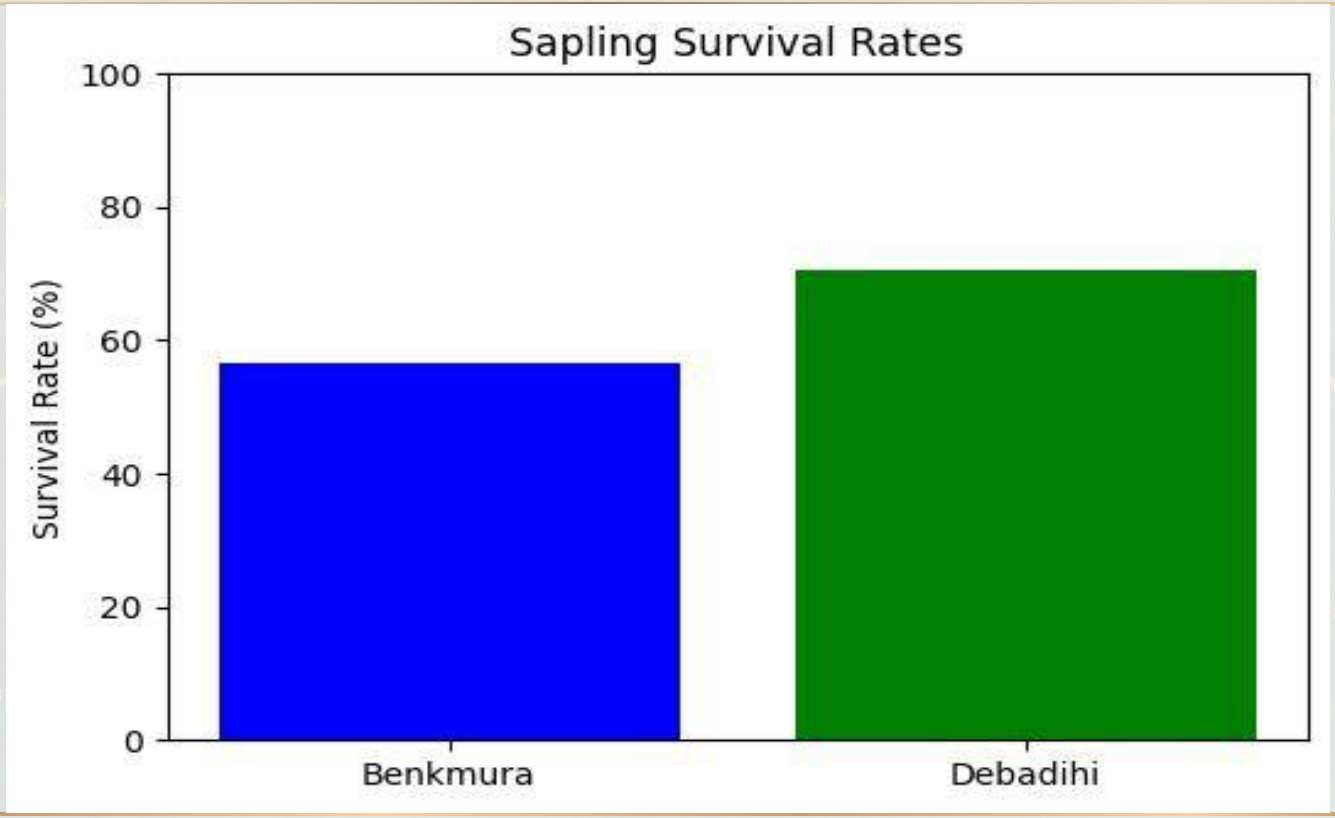
Final Results

```
--- REPORT ---
Detected Pits (Post-Pitting):8883
Alive Saplings (Post-SW): 5037
Dead Saplings (Post-SW): 3845
Survival Rate: 56.71% of 8883 originally planted)
```

Benkmura Health Analysis

```
--- REPORT ---
Detected Pits (Post-Pitting):7126
Alive Saplings (Post-SW): 5037
Dead Saplings (Post-SW): 2089
Survival Rate: 70.68%
```

Debadihi Health Analysis


Sapling Survival Rates
Survival Rate (%)
100
80
60
40
20
0
Benkmura
Debadihi

Project Timeline

1. *The Model Takes a .TIFF of the OrthoMosaic Image*
2. *Pre-processed the large Orthomosaic to produce a streamlined .png with reduced metadata for easier processing.*
3. *Extracting the useful part of each Orthomosaic(The pitted areas)*
4. *Contouring out the Pitts in the land area along with indexing them for future use*
5. *Using NDVI along with ConvMixer to detect the population of healthy plants*
6. *Model Training, Evaluation Metric and Performance result*

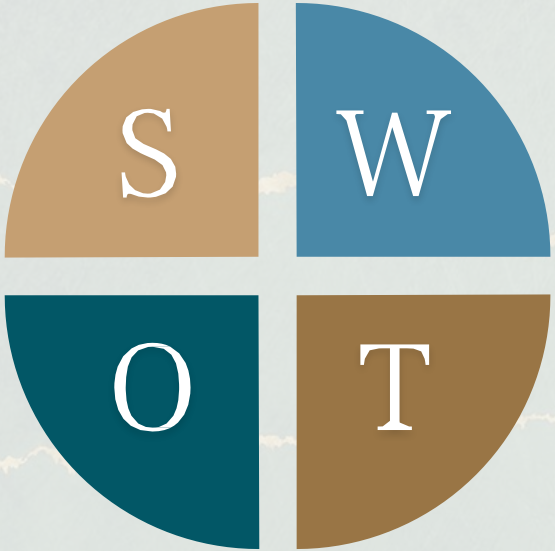
SWOT ANALYSIS

STRENGTHS

- Dynamic Nature
- Low-Latency Architecture
- Simple mechanics, great for Edge Computing

OPPORTUNITIES

- Improve Data Quality: Use higher-resolution multispectral or hyperspectral imagery.
- Better Preprocessing: Enhance image alignment and apply advanced noise reduction techniques.



WEAKNESSES

- The data complexity demands extensive pre-processing, with optimal results achievable only using infrared cameras.

THREATS

- Shadows, dense vegetation, or weather may mislead detection.
- Real-time processing needs optimized hardware.
- GPS inaccuracies affect spatial matching.

42,538,764

Each day, the count of Dead Trunks continues to Rise!

Let’s fix it together