# Project Report: Leaf Segmentation and Classification

## 1. Introduction

This project aims to automate the classification of plant leaves using image processing and machine learning techniques. The pipeline involves loading leaf images, segmenting the leaf from the background using morphological operations, extracting geometric features, and training a Random Forest classifier to predict the plant species.

---

## 2. Methodology and Source Code

### a) Core Libraries and Data Loading

We utilize `cv2` and `skimage` for image manipulation, `numpy` for matrix operations, and `sklearn` for model training[1].

```Python
import cv2
import numpy as np
import pandas as pd
from skimage.measure import label, regionprops
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report

def load_dataset(images_dir):

    dataset = []
    classes = sorted(os.listdir(images_dir))
    for cls in classes:
        img_folder = os.path.join(images_dir, cls)
        img_files = sorted(os.listdir(img_folder))
        for img_file in img_files:
            dataset.append({
                'class': cls,
                'img_path': os.path.join(img_folder, img_file)
            })
    return dataset
```

**Explanation:** This block sets up the environment and defines a function to traverse the dataset directory structure, mapping each image file to its corresponding plant species class.

## b) Image Segmentation (Preprocessing)

The segmentation function is the core image processing step. It isolates the leaf from the background to create a binary mask using a 6-step pipeline.

Python

```
def segment_leaf_image(img):
    # 1. Convert to Grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # 2. Enhance Contrast (CLAHE)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    contrast = clahe.apply(gray)

    # 3. Reduce Noise (Median Blur)
    denoised = cv2.medianBlur(contrast, 5)

    # 4. Otsu's Thresholding
    _, binary = cv2.threshold(denoised, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    # 5. Background Correction (Ensure leaf is white)
    h, w = binary.shape
    corners = [binary[0,0], binary[0, w-1], binary[h-1, 0], binary[h-1, w-1]]
    if np.mean(corners) > 127:
        binary = cv2.bitwise_not(binary)

    # 6. Keep Largest Contour Only
    contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    final_mask = np.zeros_like(binary)
    if contours:
        largest_contour = max(contours, key=cv2.contourArea)
        cv2.drawContours(final_mask, [largest_contour], -1, 255, thickness=cv2.FILLED)

    return final_mask
```

**Detailed Step-by-Step Explanation:**

1. **Grayscale Conversion:** Reduces computational complexity by removing color data, as intensity is sufficient to separate the leaf from the background
2. **Contrast Enhancement (CLAHE):** Enhances local details and edge definitions. It prevents noise amplification in flat regions while making leaf veins distinct
3. **Denoising (Median Blur):** Removes "salt-and-pepper" noise without blurring sharp edges, preventing small noise spots from being mistaken for leaf tissue
4. **Otsu's Thresholding:** Automatically calculates the optimal threshold to separate foreground (leaf) from background based on pixel intensity
5. **Background Correction:** Checks image corners to ensure the background is black. If the background is lighter than the leaf, the mask is inverted to maintain consistency
6. **Contour-Based Masking:** Removes artifacts (shadows/dirt) by identifying all shapes and keeping only the largest connected object

## c) Feature Extraction

We extract geometric properties from the binary mask to represent the leaf numerically[10].

```
Python

# Extract features using RegionProps
props = regionprops(label(binary_mask))[0]
features = [
    props.area,
    props.perimeter,
    props.eccentricity,  # Measure of how circular the leaf is
    props.solidity,      # Measure of the leaf margin (smooth vs serrated)
    props.extent,
    props.major_axis_length,
    props.minor_axis_length
]
# (Hu Moments are also extracted to capture complex shape characteristics)
```

**Key Features:**

- **Eccentricity:** Distinguishes round leaves (e.g., *Lemon*) from elongated ones (e.g., *Mango*).
- **Solidity:** Indicates edge roughness; leaves with deep lobes have lower solidity.
- **Hu Moments:** Invariant descriptors that capture complex shape characteristics independent of rotation or scale.

### d) Classifier Training (Random Forest)

The features are scaled and fed into a Random Forest classifier.

```
Python

# Feature Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split Data (75% Train, 25% Test)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.25, random_state=42, stratify=y
)

# Initialize and Train Model
rf = RandomForestClassifier(
    n_estimators=1000,
    min_samples_leaf=5,
    class_weight='balanced_subsample',
    random_state=42
)
rf.fit(X_train, y_train)
```

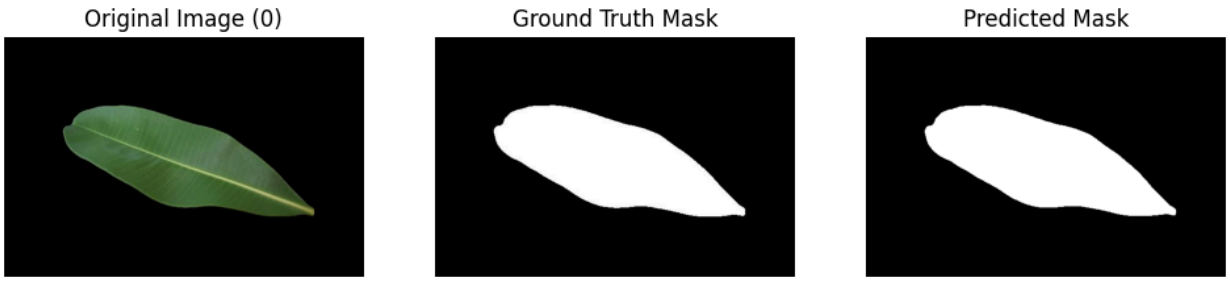
**Explanation:** We use 1000 decision trees for stability. The `balanced_subsample` weight helps the model address class imbalances (e.g., under-represented classes like *Chinar*).

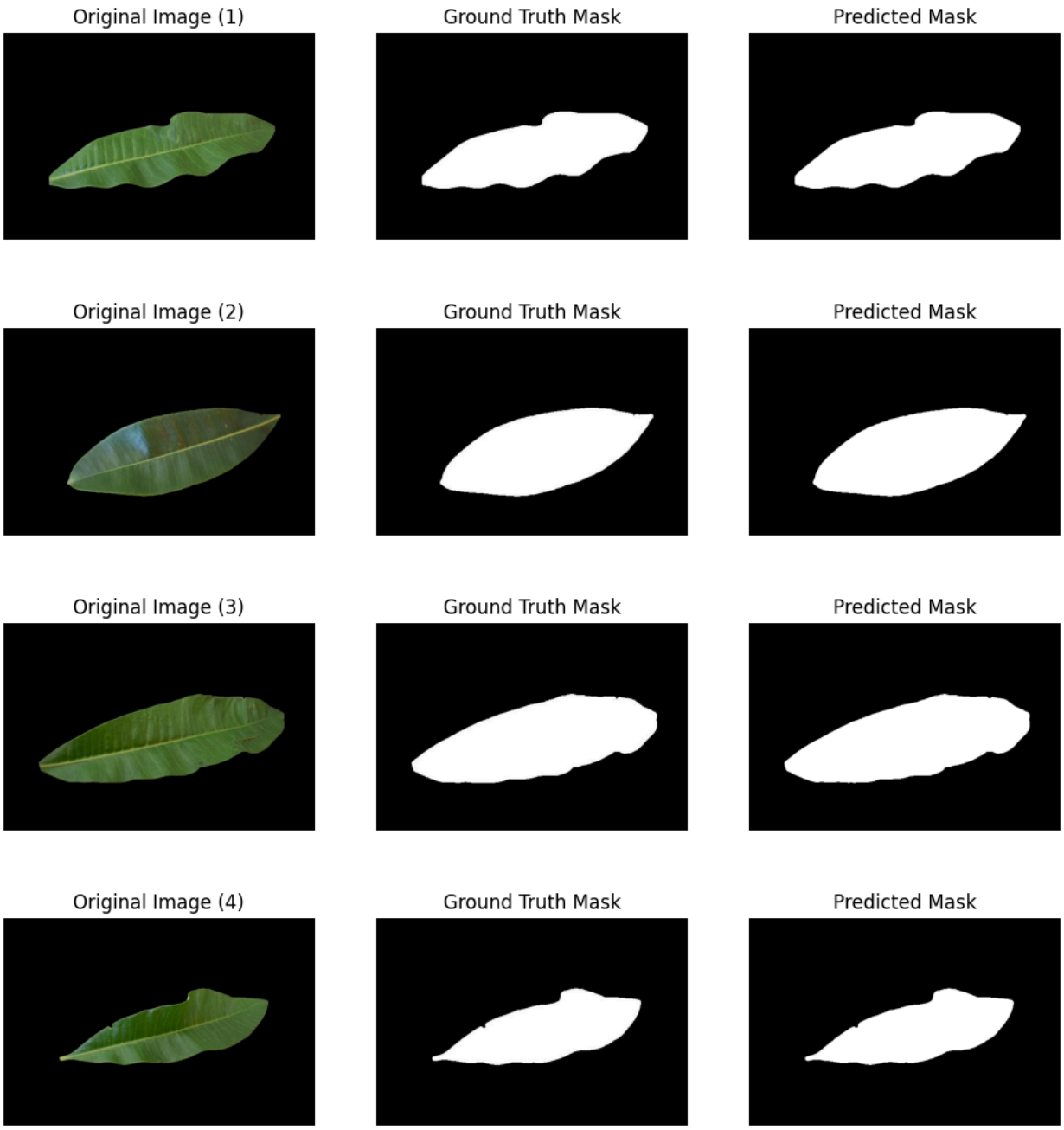
---

## 3. Visual Results

### a) Segmentation Pipeline

The following images illustrate the transformation of a raw sample into a segmented binary mask.

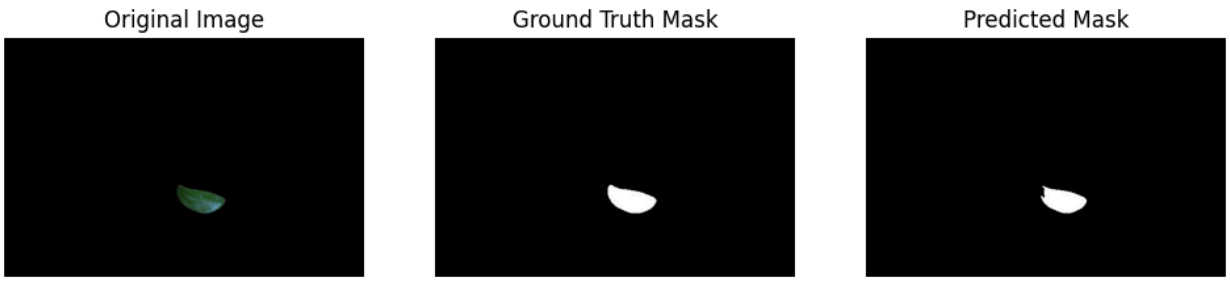




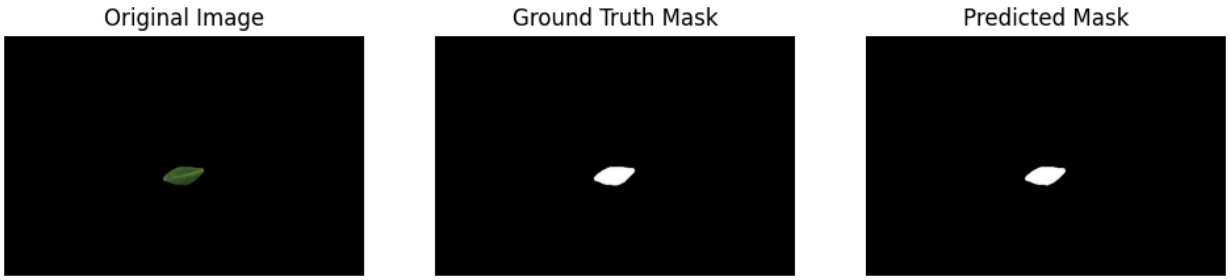

## b) Analysis of Bad Predictions

Despite robust processing, segmentation can fail under specific conditions.

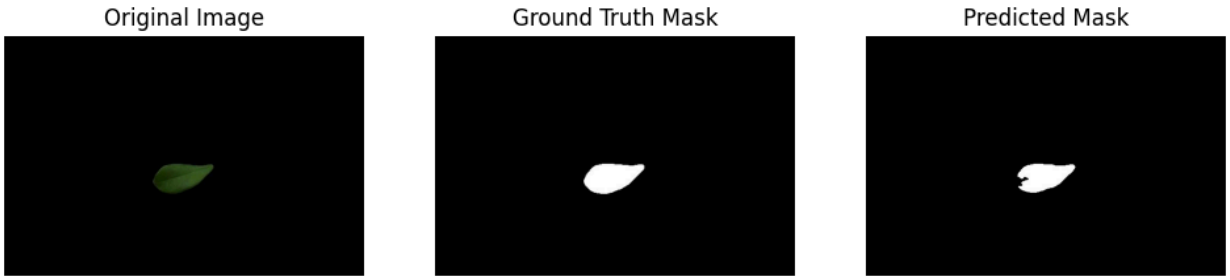
*Class: Pomegranate_(P8) IoU: 0.7919*



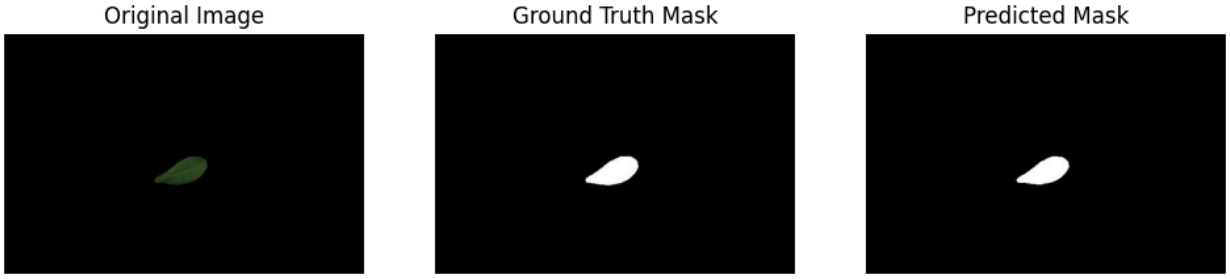

*Class: Arjun_(P1) IoU: 0.8040*



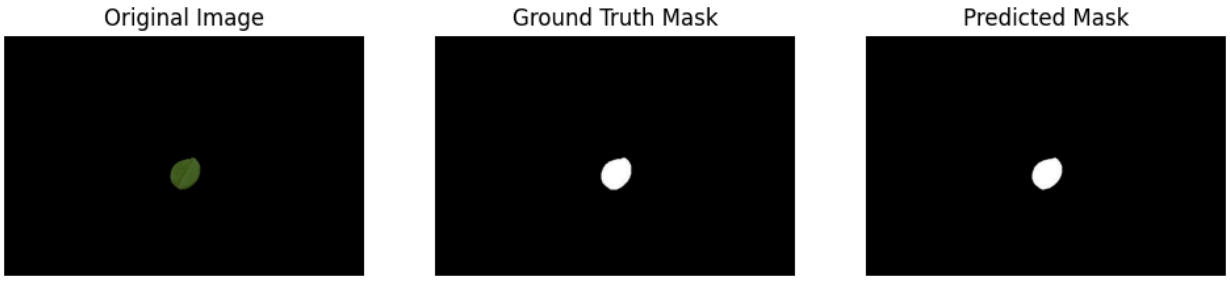

*Class: Pomegranate_(P8) IoU: 0.8141*




*Class: Pomegranate_(P8) IoU: 0.8173*




*Class: Pomegranate_(P8) IoU: 0.8205*




**Common Failure Modes:**

1. **Low Contrast:** Green backgrounds similar to the leaf color confuse Otsu's thresholding.
2. **Shadows and Glare:** Strong shadows may be interpreted as holes in the leaf, while glare can be mistaken for leaf tissue.
3. **Fragmented Leaves:** If a leaf is damaged or insect-eaten, the algorithm may discard valid parts of the leaf, keeping only the largest fragment.

---

# 4. Model Evaluation

## a) Overall Performance

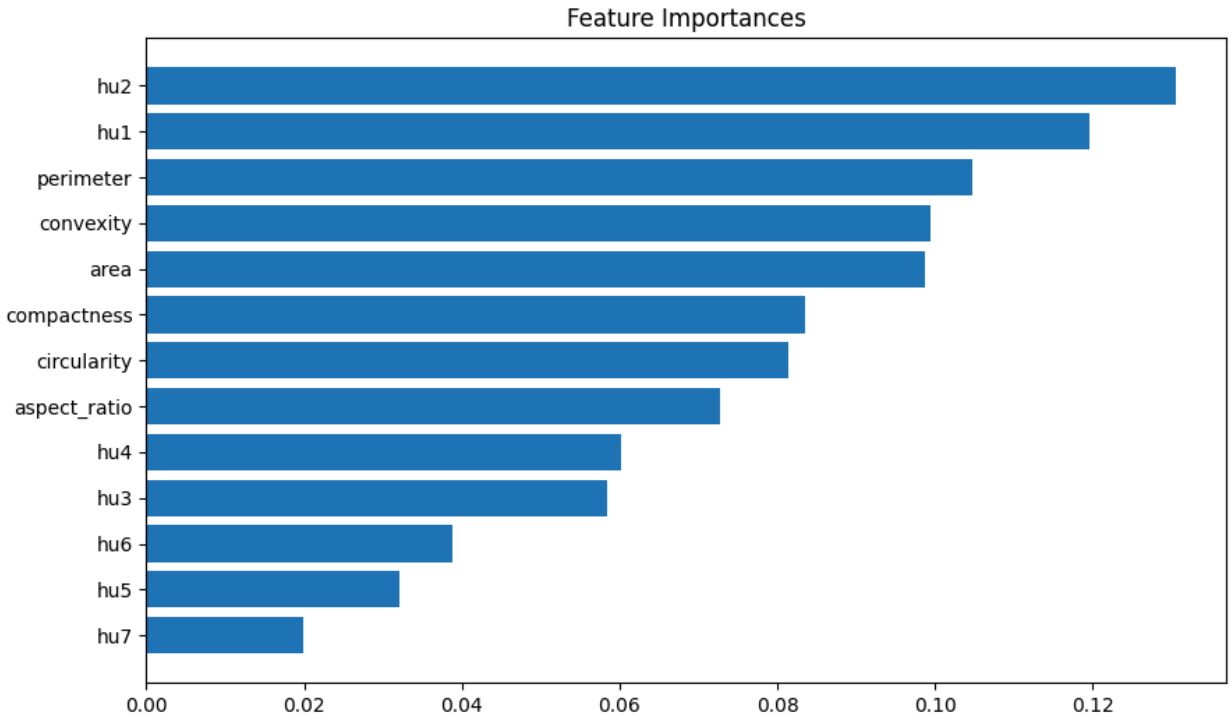
- **Accuracy: 74.65%** [16]
- **Test Set Size:** 493 samples [17]

## b) Classification Report

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| **Alstonia (P0)** | 0.61 | 0.70 | 0.65 | 43 |
| **Arjun (P1)** | 0.71 | 0.60 | 0.65 | 53 |
| **Basil (P2)** | 0.70 | 0.70 | 0.70 | 37 |
| **Chinar (P3)** | **0.93** | **0.96** | **0.94** | 26 |
| **Jamun (P4)** | 0.72 | 0.70 | 0.71 | 67 |
| **Jatropha (P5)** | **0.94** | **0.88** | **0.91** | 33 |
| **Lemon (P6)** | 0.58 | 0.78 | 0.67 | 40 |
| **Mango (P7)** | 0.84 | 0.88 | 0.86 | 42 |

| | | | | |
|---|---|---|---|---|
| **Pomegranate (P8)** | 0.73 | 0.68 | 0.71 | 72 |
| **Pongamia (P9)** | 0.83 | 0.78 | 0.80 | 80 |

## c) Feature Importance




**Analysis:**

- **Top Features:** *Hu Moments (hu2, hu1)* and *Perimeter* were the most critical for classification.
- **Best Classes:** *Chinar* and *Jatropha* achieved F1-scores >90% due to their distinct geometric shapes.
- **Challenges:** *Alstonia* and *Lemon* (Precision ~60%) were frequently confused due to their generic oval shapes.

---

# 5. Discussion: Why Random Forest?

For this classification task, **Random Forest** was chosen over Deep Learning (CNNs) or Linear Regression for the following reasons:

1. **Suitability for Tabular Data:** Random Forest is optimized for structured data (numerical features like Area and Perimeter), whereas CNNs are designed for raw pixel data.
2. **Robustness to Overfitting:** By averaging predictions from 1000 trees ("Bagging"), the model generalizes better than a single decision tree.
3. **Interpretability:** The model provides clear Feature Importance scores, offering biological insight into which shapes distinguish species (e.g., Eccentricity).
4. **Handling Non-Linearity:** It effectively captures complex, non-linear boundaries between similar leaf shapes that a simple linear model cannot.
5. **Efficiency:** It performs exceptionally well on small-to-medium datasets (~2,000 images) and trains in seconds without requiring a GPU.

---

# 6. Conclusion

The implemented pipeline successfully segments and classifies plant leaves with ~75% accuracy. While distinct shapes are classified with high precision, generic oval leaves remain a challenge. Future work could improve performance by incorporating texture analysis or vein patterns to supplement geometric features.

# 7. Appendices

## Appendix A: Complete Feature Set

Features (14 total):
- area, perimeter, aspect_ratio, circularity
- compactness, convexity
- hu1, hu2, hu3, hu4, hu5, hu6, hu7
- eccentricity (derived from major/minor axis)

## Appendix B: Segmentation Pipeline Visualization

The segmentation process transforms raw leaf images through the following stages (detailed examples in Section 4):

**Raw Image → Grayscale → CLAHE → Denoising → Otsu Threshold → Background Correction → Final Mask**

Sample IoU scores demonstrate excellent segmentation quality:

- Average IoU over dataset: 0.9427
- Minimum IoU: 0.7919
- Maximum IoU: 0.9769

## Appendix C: Dataset Distribution

**Class Distribution in Dataset (1,971 total samples):**

| Class | Training Samples | Test Samples | Total |
|---|---|---|---|
| Pongamia (P9) | ~240 | 80 | ~320 |
| Pomegranate (P8) | ~216 | 72 | ~288 |
| Jamun (P4) | ~201 | 67 | ~268 |
| Arjun (P1) | ~159 | 53 | ~212 |
| Mango (P7) | ~126 | 42 | ~168 |
| Alstonia (P0) | ~129 | 43 | ~172 |
| Lemon (P6) | ~120 | 40 | ~160 |
| Basil (P2) | ~111 | 37 | ~148 |
| Jatropha (P5) | ~99 | 33 | ~132 |
| Chinar (P3) | ~78 | 26 | ~104 |

The stratified split ensures proportional representation of each class in training and test sets.

---

**Project Repository**: https://github.com/MohamedKhalidmk/Segmentation_IoU_Classification
**Date**: December 2025