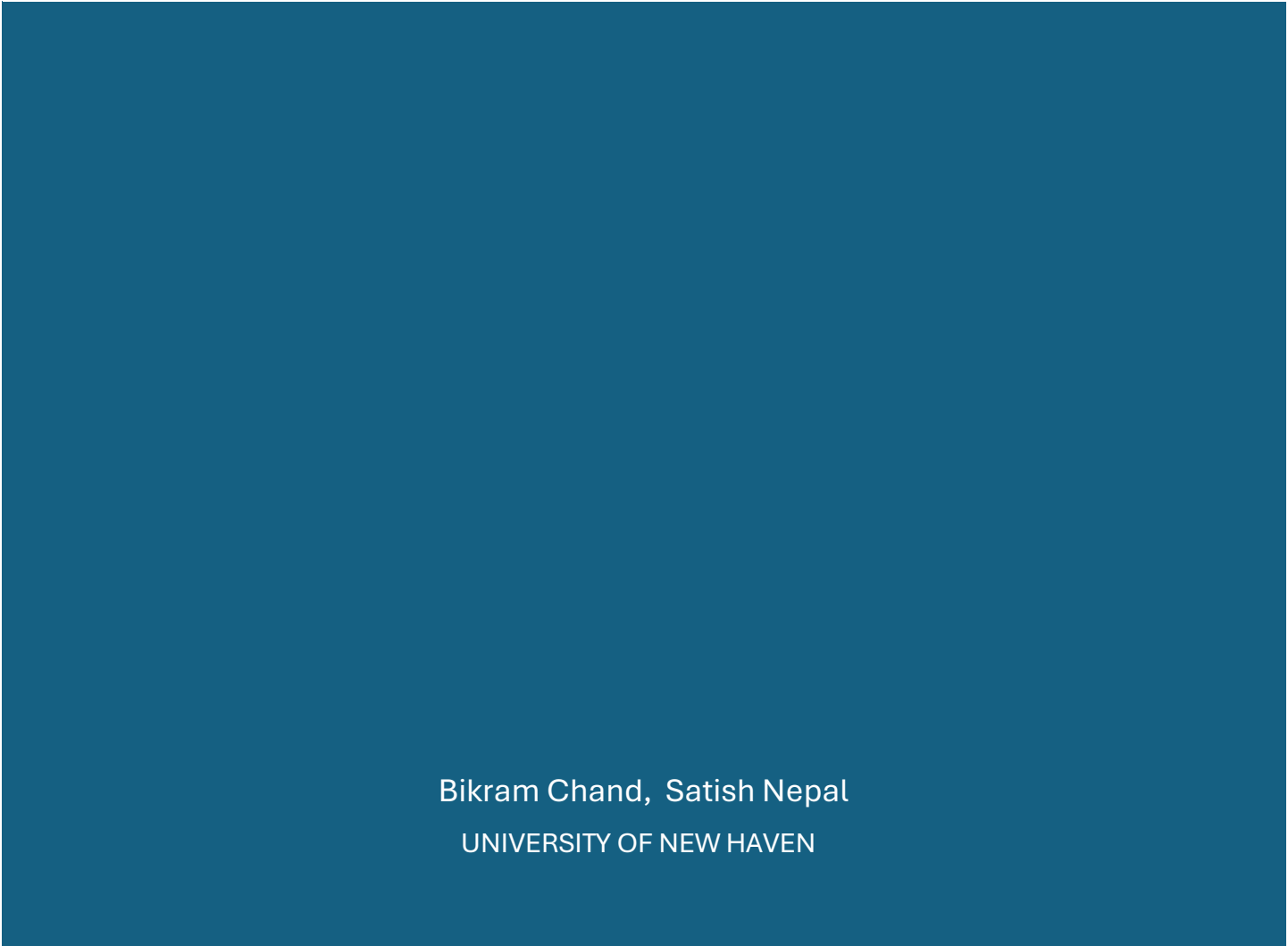




# **Leaf Segmentation and Quantification Using Deep Learning**



Bikram Chand, Satish Nepal  
UNIVERSITY OF NEW HAVEN

# 1. Introduction

Leaf detection and quantification is an important challenge in deep learning, particularly in environmental monitoring and agricultural studies. This project focuses on segmenting and counting leaves in images by labeling each pixel to identify individual leaves accurately. Two models have been employed for this purpose: ResNet-UNet for leaf type segmentation with five classes (background and four leaf types) and ResNet-50 for leaf counting.

A dataset of 200 images of fallen leaves was collected around the campus, and comprehensive preprocessing steps, including data augmentation and normalization, were applied to enhance the dataset's quality. The pre-trained ResNet-UNet model was fine-tuned by loading existing weights and biases while replacing the final output layer to accommodate five segmentation classes. Additionally, a suitable hyperparameter optimization strategy was implemented to improve the model's performance.

This paper presents the methodology, experiments, and results, along with a detailed analysis of the findings, demonstrating the effectiveness of the proposed models in addressing leaf segmentation and counting challenges.

## 2. Dataset and Preprocessing

The dataset for this project comprises custom images of fallen leaves, captured using a smartphone camera. There are 200 images in total, representing four different types of leaves: eastern cottonwood, sugar maple, camellia, and river birch. All images are in JPEG format. The images have been annotated with masks created using the VGG Image Annotator (VIA) tool, which provides the necessary labeling for training the model. The full dataset, including the images and masks, are hosted here [Google Drive Link](#).

In the preprocessing pipeline, the dataset begins with the extraction information from annotation json files. This includes filenames, image sizes, region details, and leaf types. The data is then organized into a DataFrame, which is grouped by filename and size. The leaf count for each image is added by merging the DataFrame with an external CSV file containing this information.

For the image processing, a custom dataset class is defined using PyTorch's `Dataset` module. This class loads images and their corresponding annotation data, where each region's shape is used to generate a mask.

The data augmentation process applies several image transformations that includes random resizing, flipping, rotation, color adjustments, and Gaussian blur. For masks, similar transformations are applied without the color changes or grayscale conversion. To prepare the data for model input, the images are resized to 224x224 pixels and normalized based on pre-calculated mean and standard deviation values from the training set which are mean=[0.4829, 0.4384, 0.3892], std=[0.2369, 0.2155, 0.2078].

Finally, the dataset is partitioned into 80% training, 10% validation, and 10% test sets. A DataLoader is used to load mini-batches of 8 images at a time for each set.

### 3. Method

The proposed solution involves two deep learning networks: one for semantic segmentation and another for leaf counting. The first network, a semantic segmentation model, produces a mask that indicates which pixels correspond to the leaves. This mask is used as input for the second network, the leaf counting model, which then counts the leaves in the plant regions of the image. The research paper “[A Segmentation-Guided Deep Learning Framework for Leaf Counting](#)” has been used as a reference for this.

For the respective model implementations, you can refer to the following GitHub code links:

- Leaf Segmentation Model: <https://github.com/gvil-research/leaf-segmentation-unet/blob/main/model/model.py>
- Leaf Counter Model: [https://github.com/IgorFreik/Leaf\\_Counter\\_ComputerVision/blob/main/src/resnet\\_unet.py](https://github.com/IgorFreik/Leaf_Counter_ComputerVision/blob/main/src/resnet_unet.py)

### 3.1 ResNetUnet Architecture (Leaf Segmentation)

The architecture uses a ResNet-18 as the backbone model for the segmentation. The network uses a series of convolutional layers to extract features, and then these features are refined using upsampling layers to generate the segmentation mask.

#### Custom Layers

- **Convolutional Layers:** These layers are added to the ResNet-18 backbone to refine the features before upsampling.
- **Upsampling Layers:** After extracting features through ResNet-18, the network uses bilinear upsampling to upsample the feature maps and progressively recover spatial resolution.
- **Skip Connections:** The output from earlier layers (e.g., layer0, layer1) is concatenated with the upsampled features from the higher layers (e.g., layer4) to preserve fine-grained details, which is essential for accurate segmentation.
- **Final Convolution Layer:** The network ends with a 1x1 convolutional layer to produce the final segmentation mask.

#### Forward Pass

The input image first goes through initial convolution layers to extract basic features. It is then passed through the ResNet backbone, from layer0 to layer4, to learn more complex features. The feature maps from each layer are combined and passed through upsampling layers. Finally, these refined features are processed through convolutional layers to produce the final plant segmentation mask.

## **Final Output**

The output is a segmentation mask where each pixel is classified as belonging to the leaf or not.

### **3.2 Resnet50 Architecture (Leaf Counting)**

The LeafCounter model is based on ResNet-50 and is used to count the leaves in the segmented part of the image. The backbone uses ResNet-50 pre-trained on ImageNet. This model is frozen except for the layers that are used for leaf counting (e.g., the first convolutional layer and the fully connected layer).

The first convolutional layer is modified to accept 4 channels (3 for the image and 1 for the segmentation mask). The final fully connected layer is adjusted to output a single value representing the number of leaves.

#### **Forward Pass:**

The input consists of a stack of the original image and its corresponding segmentation mask, forming a 4-channel input. This input is then passed through the modified ResNet-50 layers, where the model extracts relevant features. The resulting feature map is subsequently processed through a fully connected layer, which outputs a single value representing the number of leaves in the image.

## Final Output

The output is a scalar value, which represents the count of leaves in the image.

### 3.3 Loss Function

#### 3.3.1 Cross-Entropy Loss

For the segmentation task, Cross-Entropy Loss is used, which is effective for multi-class classification problems, such as semantic segmentation with multiple object categories. In this case, the model is tasked with classifying each pixel into one of five classes: four different types of leaves and one background class. Cross-Entropy Loss calculates the difference between the predicted class probabilities and the actual class labels for each pixel, penalizing incorrect predictions. By minimizing this loss, the model learns to accurately segment the plant and background regions, while distinguishing between the different types of leaves.

#### 3.3.2 MSE

For the leaf counting task, Mean Squared Error (MSE) Loss is employed. As a regression problem, the goal is for the model to predict the number of leaves present in the image. MSE Loss measures the squared difference between the predicted leaf count and the actual count, and the model aims to minimize this difference. This approach ensures that the counting model provides a more accurate numerical estimate of the total number of leaves,

making it suitable for regression tasks where exact values are required.

### 3.4 Model Training and Freezing Layers

The segmentation model was initialized using pretrained weights, which were downloaded from [here](#) and subsequently uploaded to my personal Google Drive for ease of access and integration.

```
✓ [13] pretrained_model_path = "/content/drive/My Drive/DLProjectDataset/latest_weights.pth"
11s state_dict = torch.load(pretrained_model_path, map_location=torch.device('cpu'))
print(len(state_dict))

<ipython-input-13-f42f8ac7bf02>:2: FutureWarning: You are using `torch.load` with `weights_only=False`
state_dict = torch.load(pretrained_model_path, map_location=torch.device('cpu'))
268

✓ [32] state_dict
1s
-0.0159, -0.0303, 0.0089, -0.0082, -0.0135, 0.0330, -0.0002, 0.0298,
0.0252, -0.0234, 0.0176, 0.0457, 0.0121, -0.0329, -0.0125, 0.0381,
0.0233, 0.0215, 0.0528, 0.0315, 0.0389, -0.0180, -0.0199, 0.0344]],
('conv_original_size2.0.weight',
 tensor([[[[-8.2783e-11, -6.7351e-11, -5.5781e-11],
 [-7.9251e-11, -6.1531e-11, -5.0908e-11],
 [-7.3290e-11, -5.5916e-11, -4.2023e-11]],
 [[-2.4307e-03, -1.9148e-02, -1.5946e-02],
 [-1.5325e-02, 4.3216e-03, -2.2712e-02],
 [-3.5879e-03, -1.5958e-02, -1.8158e-02]],
```

After loading the pretrained weights, the last convolutional layer of the segmentation model is modified to output a 5-class mask (instead of the original single class).

```
✓ ▶ seg_model.conv_last = nn.Conv2d(64, 5, kernel_size=(1, 1), stride=(1, 1))
s print(seg_model)
```



Once the segmentation model is loaded with the pretrained weights, all layers except the final convolutional layer (conv\_last) are frozen to prevent further training. This allows fine-tuning only the final layers that are important for the specific task:

```
▶ # Freeze all layers in the model
for param in seg_model.parameters():
    param.requires_grad = False

for param in seg_model.conv_last.parameters():
    param.requires_grad = True

# Check the requires_grad status of all parameters in the model
for name, param in seg_model.named_parameters():
    print(f"Layer: {name}, requires_grad: {param.requires_grad}")
```

```
↳ Layer: base_model.bn1.bias, requires_grad: False
   Layer: base_model.layer1.0.conv1.weight, requires_grad: False
   Layer: base_model.layer1.0.bn1.weight, requires_grad: False
```

This ensures that the model retains the useful features learned during pretraining while adapting to the specific task of segmenting plants in images.

## 4. Experiments and Results

### 4.1 HyperParameter Tuning

In this experiment, we performed hyperparameter tuning to optimize the learning rate and momentum for training the segmentation model. Our goal was to find the best combination of hyperparameters that would minimize the validation loss and improve model performance.

### 4.1.1 Learning Rate

We first focused on identifying the optimal learning rate for the model. To achieve this, we tested a range of learning rates by using a logarithmic scale from 0.000001 to 0.1. We employed a learning rate scheduler, iterating over the learning rates in the list and training the model for 20 epochs. The model was trained using the Stochastic Gradient Descent (SGD) optimizer, and we used early stopping based on the validation loss, with a patience of 10 epochs. This method allowed us to monitor and select the learning rate that resulted in the best validation performance.

After conducting the experiment, we identified that the best learning rate found is 0.0999 which resulted in the lowest validation loss.

```
· Stopping early at lr: 0.00000 because of no improvement for 10 epochs
  1/5: LR: 0.00000, Train Loss: 0.21365, Val Loss: 0.26407
  2/5: LR: 0.00002, Train Loss: 0.20942, Val Loss: 0.25179
  3/5: LR: 0.00032, Train Loss: 0.12717, Val Loss: 0.15661
  4/5: LR: 0.00562, Train Loss: 0.11650, Val Loss: 0.13954
  5/5: LR: 0.10000, Train Loss: 0.10788, Val Loss: 0.12645
  best lr: 0.09999999999999999
```

---

### 4.1.2 Momentum

Next, we focused on finding the best momentum value. Using the learning rate identified from the previous step, we performed a search for the optimal momentum by testing a few common values: 0, 0.5, 0.9, and 0.99. Similar to the learning rate search, we trained the model using the SGD optimizer with the selected learning rate and varying momentum values. Again, we used early stopping with a

patience of 10 epochs to prevent overfitting and to identify the momentum value that resulted in the best validation loss.

After conducting the experiment, we identified that the best momentum found is 0.5.

```
Stopping early at momentum: 0.00000 because of no improvement for 10 epochs
1/4: Momentum: 0.00000, Train Loss: 0.10223, Val Loss: 0.11775
Stopping early at momentum: 0.50000 because of no improvement for 10 epochs
2/4: Momentum: 0.50000, Train Loss: 0.10762, Val Loss: 0.13304
3/4: Momentum: 0.90000, Train Loss: 0.12124, Val Loss: 0.14599
4/4: Momentum: 0.99000, Train Loss: 0.58597, Val Loss: 0.86513
best_mom 0.5
```

Ac  
86%

### 4.1.3 Optimizer and Learning Rate Scheduler

For this experiment, the optimizer used was the SGD optimizer, with the following configuration:

```
[ ] from torch.optim.lr_scheduler import StepLR
    optimizer = torch.optim.SGD(
        filter(lambda p: p.requires_grad, seg_model.parameters()), lr=best_lr, momentum=best_mom, weight_decay=0.001
    )

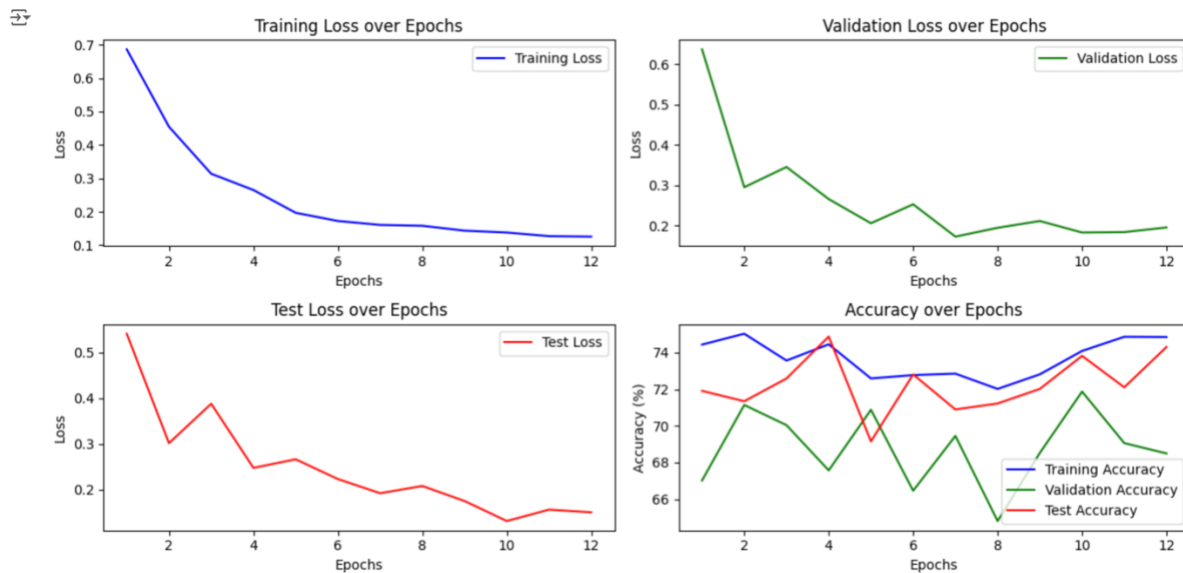
# Example of using a learning rate scheduler
scheduler = StepLR(optimizer, step_size=10, gamma=0.1)
```

This configuration, along with the learning rate scheduler, helped adjust the learning rate during training, ensuring better convergence. The StepLR scheduler decayed the learning rate by a factor of 0.1 every 10 epochs, allowing the model to adapt and achieve better generalization.

## 4.2 Evaluation

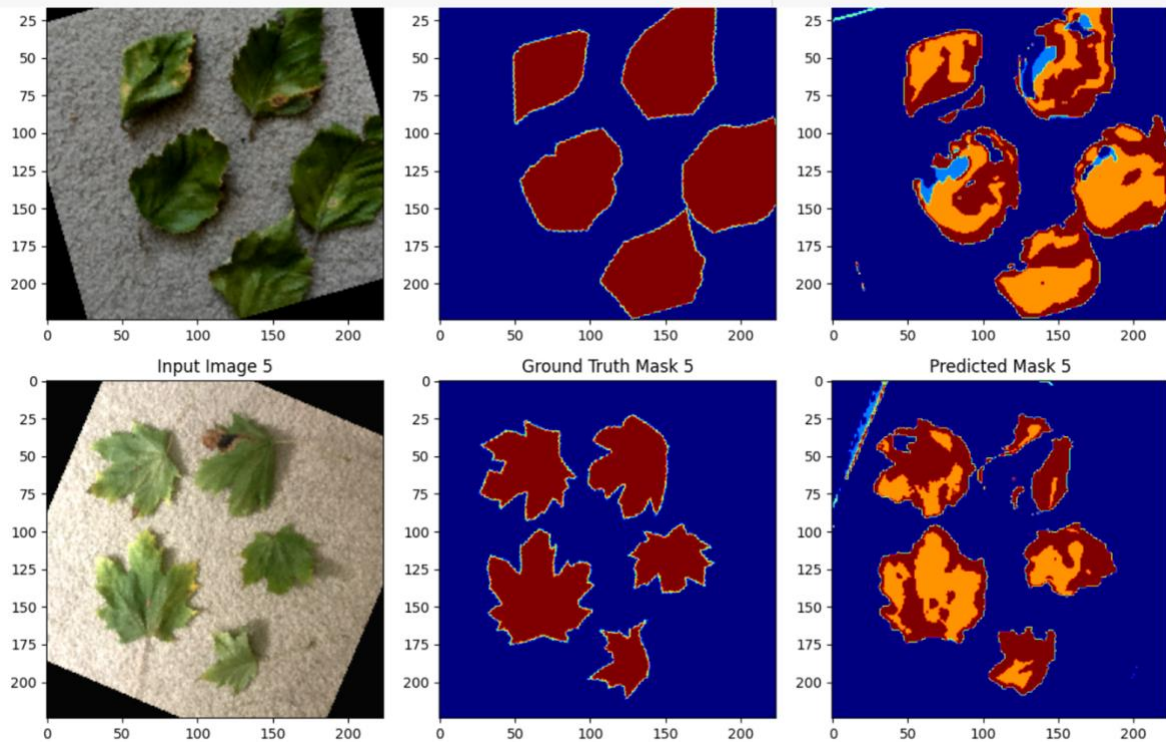
The final results from the evaluation showed that the model achieved a training accuracy of 74.85%, a validation accuracy of

68.49%, and a test accuracy of 74.30%. The training loss decreased to 0.126, while the validation loss ended at 0.195 and the test loss at 0.150. These results indicate that the model performed well in terms of both training and generalization, with some fluctuations in validation performance but consistent improvement in test accuracy.



### 4.3 Predicted Segmentation Mask

To further illustrate the performance of the model, we include visual examples of the predicted segmentation masks generated on the test data. These masks demonstrate how effectively the model segments objects of interest in the input images. By comparing the predicted masks with the ground truth, we can observe the alignment between the model's output and the expected results.



## IOU

To evaluate the model's performance, we calculated the IoU (Intersection over Union) scores for the predicted masks. The IoU measures how well the predicted segmentation aligns with the ground truth. The average IoU scores for the images in the test batch ranged from 0.1404 to 0.3467. These values indicate that the model's predictions currently show limited alignment with the ground truth, suggesting the need for further improvements in the model architecture, training process, or dataset quality.

## 4.4 Predicted Leaf Counts

---

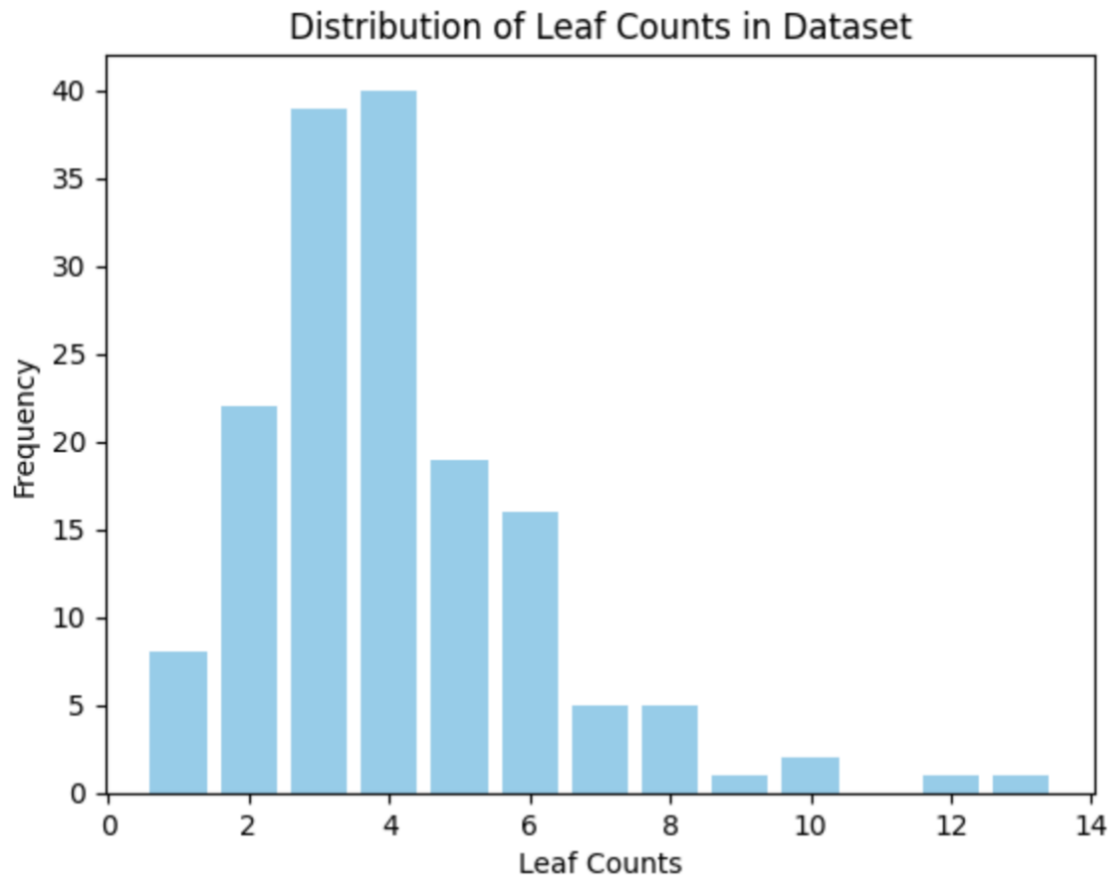
```
✓ Predicted Count (Test): 5, Actual Count (Test): 6
  Predicted Count (Test): 5, Actual Count (Test): 6
  Predicted Count (Test): 6, Actual Count (Test): 8
  Predicted Count (Test): 5, Actual Count (Test): 3
  Predicted Count (Test): 6, Actual Count (Test): 6
  Predicted Count (Test): 6, Actual Count (Test): 6
  Predicted Count (Test): 5, Actual Count (Test): 4
  Predicted Count (Test): 5, Actual Count (Test): 2
  Predicted Count (Test): 6, Actual Count (Test): 3
  Predicted Count (Test): 5, Actual Count (Test): 4
  Predicted Count (Test): 5, Actual Count (Test): 3
  Predicted Count (Test): 5, Actual Count (Test): 4
  Predicted Count (Test): 5, Actual Count (Test): 5
  Predicted Count (Test): 5, Actual Count (Test): 1
  Predicted Count (Test): 6, Actual Count (Test): 3
  Predicted Count (Test): 5, Actual Count (Test): 8
  Predicted Count (Test): 6, Actual Count (Test): 1
  Predicted Count (Test): 5, Actual Count (Test): 1
  Predicted Count (Test): 5, Actual Count (Test): 6
  Test Loss: 7.9029, Test MAE: 2.2883
```

---

During the test phase, the model's performance was evaluated by comparing the predicted leaf counts with the actual counts. The predictions were rounded to the nearest whole number to make the comparison easier. The Mean Squared Error (MSE) was 7.9029, showing how far off the predictions were on average. The Mean Absolute Error (MAE) was 2.2883, which is the average of the absolute differences between predicted and actual counts.

The model performed well in some cases, with predictions close to the actual counts (e.g., predicted = 6, actual = 6). However, in some cases, the predictions were quite different from the actual values

(e.g., predicted = 6, actual = 1, or predicted = 5, actual = 8). This is likely because the dataset is imbalanced, meaning it has too many examples of certain leaf counts (like 3 or 4) and not enough of others. Fixing this by balancing the dataset could help the model make better predictions



## Conclusion

In conclusion, this project successfully implemented a two-model approach for leaf segmentation and counting using deep learning. The ResNet-UNet model effectively segmented leaf regions into five classes, while the ResNet-50 model accurately counted the number of leaves based on the segmentation masks. The dataset, consisting of 200 annotated images of fallen leaves, was carefully preprocessed with data augmentation and normalization to improve model performance. Hyperparameter tuning identified optimal values for learning rate and momentum, leading to improved training outcomes. Despite the promising results, with training accuracy of 74.85% and test accuracy of 74.30%, the model's performance showed room for improvement, particularly in the segmentation task, as indicated by the IoU scores. Future work will focus on refining the model architecture and enhancing the dataset for better alignment between predicted and ground truth segmentation.



## References

- Ferik, I. (n.d.). *Leaf counting with Deep Learning*. Medium. Retrieved from <https://medium.com/@i.freik.work/leaf-counting-with-deep-learning-762320a0da55>.
- Fan, X., Zhou, R., Tjahjadi, T., Das Choudhury, S., & Ye, Q. (2022). *A Segmentation-Guided Deep Learning Framework for Leaf Counting*. Frontiers in Plant Science. Retrieved from <https://www.frontiersin.org/articles/10.3389/fpls.2022.844522/full>

## Appendix A: Dataset and Code

### **Google Colab Link:**

<https://colab.research.google.com/drive/1ymQ0G9uqcdcEkGeDfaiQNE6YtCwT4Uh-?usp=sharing>

### **Dataset and Annotations Link:**

<https://drive.google.com/drive/folders/1Xy8s-oeuITBed9zCKjbuNDM8VPGTbdxu?usp=sharing>