Interactive Leaf Doctors: Leaf Segmentation and Health Prediction

Julia Ho, Autumn Nguyen, Sulagna Saha
Department of Computer Science, Mount Holyoke College
50 College Street, South Hadley, MA 01075
ho25l@mtholyoke.edu, ngoc54n@mtholyoke.edu, saha23s@mtholyoke.edu

Abstract

Leaf Doctors is an interactive program designed for predicting the health of a leaf using a laptop's camera. Employing various computer vision techniques, the program incorporates Sobel edge detection, LAB color space conversion, color channel masking, and blob detection for precise leaf segmentation. Subsequently, a convolutional neural network is built from scratch to classify the segmented leaves as healthy or unhealthy as an output of the program.

1 Introduction

Our project started with a vision of recognizing the health issues of indoor plants (over-watering, under-watering, pests, diseases etc) and providing the users with actionable advice. As we struggled to find a dataset of indoor plants categorized into different diseases, we tweaked our goal to only focus on predicting if a leaf is healthy or not. Currently, our project enables a user to take a real time picture using a webcam, segment the leaf from a complex background and predict the leaf's health using convolutional neural network (CNN). Our project techniques can potentially be an aid for learning in the botany classes where computers can help students learn the health of leaves from different species. Besides, our leaf segmentation process can be an initial step for segmenting a tree and extracting leaves to determine the health overall.

2 Related Work

Most leaf-based works in computer vision (CV) are related to crop health in the agricultural sector. As we wanted to work with leaves available on Mount Holyoke campus, we aimed to learn the techniques that we can use to predict the health of indoor plant leaves. Although most of the computer vision work related to plant health prediction mainly utilizes Machine Learning (ML), we initially wanted to use solely CV techniques for the whole project, not so that we could really leverage what we learned in the course.

Among the research papers that we explored, there was one research that inspired us by how they separated the leaves from the images by separating the foreground and background [5]. We wrote the code based on what we understood about their technique of extracting a leaf from uniform backgrounds, but a lot of novel parts were added to make it work with the complex backgrounds according to the specific features of Leaf Doctors, i.e users placing a leaf in the webcam in the center of the frame.

This research also used deep-learning models, which has reinforced our understanding that ML were a very faster and competent way to learn patterns of leaf health. We ultimately decided to predict the health of the segmented leaf using a Convolutional Neural Network (CNN), as it is a common choice for image classification, and it will allow us to leverage what we learned about image convolution

in the course. The most helpful resource for us in implementing the CNN is Nicholas Rennotte's tutorial [4].

3 Datasets and User Inputs

3.1 Datasets

We trained our CNN on a dataset of walnut leaves that have 680 unhealthy leaves and 1473 healthy ones. It is a subset of a public dataset on Tensorflow [1].

3.2 User Inputs

When users run our program, they are prompted to activate their laptop's camera for the purpose of capturing an image featuring a leaf. This image serves as the initial input for our implementation, starting with edge detection. To facilitate this interaction, our program accommodates video streaming as part of the graphical user interface (GUI), enabling users to engage with the camera in real-time. The video streaming functionality is implemented using various Python dependencies, including `IPython`, `cv2`, `numpy`, `PIL` (Pillow), and `io`.

4 Methods

4.1 Sobel Edge Detection

The important first step of the implementation is to detect all edges in the input camera image. As we are interested in leaf segmentation, identifying edges provide inputs for the subsequent implementation steps to extract key features that characterize the leaf from other objects in the image more easily. In addition, focusing on the edges help the program filter out less relevant information, reducing the impact of irrelevant details and noise.

Our approach to this step is to use Sobel edge detection, a technique that is covered in our COMSC-341CV course material. Sobel edge detection works by calculating the gradient of image intensity at each image pixel and finding the direction of the largest increase from light to dark and the rate of change in that direction. The result of Sobel edge detection shows how abruptly or smoothly the image change at each pixel, indicating how likely it represents an edge. [6]

Figure 1: Result of Sobel edge detection on a leaf image with a simple background.

4.2 Leaf Segmentation

Before diving into the approach, it is necessary to know why we are interested in leaf segmentation. We want to have a visualization for a user to see only the parts we are interested in extracting. We want the CNN's classification to be influenced only be the features of the leaf, not of any other thing in the background. Therefore, we segment out the leaves from the background before passing it into the CNN. The concepts used below are mentioned in the research paper [5], but we coded all the steps using available functions from class and `cv2` library and added our idea of recognizing the center blob.

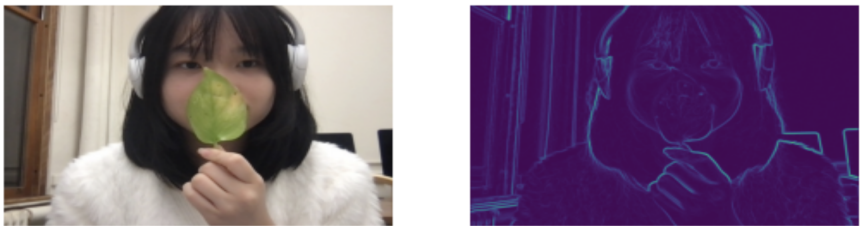

Figure 2: Result of Sobel edge detection on an example image input captured by the front camera.

4.2.1 Image Preprocessing

While working with the images, we don't want the lighting condition the user is taking picture influence the final result. But if we take an RGB image, there's a gradient in all the channels for different brightness. So, we use LAB color space instead of the RGB colorspace to work with the color channels and lightness separately.[1]

Figure 3: Original image, LAB color space, L channel, a channel, b channel

Then we blur the channels separately using the Gaussian blur (learned in class) to remove noises in the image.

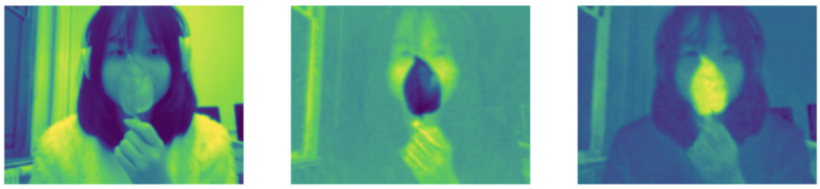

Figure 4: Blurred version using sigma value 2 and 5x5 kernel

4.2.2 Edge Detection and Background Color Detection

Using Sobel edge detection, we determine the edges of the images. The color of the edges tells us about the color characteristics of the whole images. After determining the colors of the edges, our code calculates the median color values `median_a and median_b` of the a and b channels for pixels corresponding to edges in the image.

4.2.3 Shifting the channels to recognize foreground

Currently, we are shifting the a and b channel by `median_a - 128` amounts. The value of 128 in the LAB color space corresponds to a neutral gray color. By subtracting the calculated median from 128, the code is effectively shifting the color distribution of the a and b channels so that the median color of the background becomes neutral gray. With the background color shifted towards gray, the color variations in the non-background areas (foreground) become more pronounced.

[1] LAB is a color space that represents colors in terms of three components: L (lightness), a (green to red), and b (blue to yellow)

4.2.4 Color Channel Masks

After correcting the channels, we apply the thresholding to create the mask including the area we are interested in. The code snippet involving the a and b channel masks is designed to segment and highlight specific color ranges within the LAB color space.

The a channel mask, identifies regions in the image where the a channel exhibits low positive values, indicative of a tendency towards greenish tones. Pixels with a values greater than 126 are set to 0 (black), and those less than or equal to 126 are set to 255 (white) in the binary mask.

Simultaneously, the b channel mask isolates areas where the b channel has high positive values, suggesting a prevalence of yellowish colors. In this binary mask, pixels with b values greater than 130 are set to 255 (white), while those less than or equal to 130 are set to 0 (black).

The resulting `combined_mask` retains pixels that are simultaneously highlighted in both the a and b channel masks, effectively pinpointing regions where both green and yellow color features coexist.

Figure 5: a_channel_mask, b_channel_mask, and combined_mask

4.2.5 Best Blob Selection

Initially, the mask is slightly blurred using a Gaussian Blur operation to smooth out small irregularities. The connected components within the blurred mask are then identified using the `cv2.connectedComponentsWithStats` function. Each connected component is assigned a score based on its size (area) and its distance from the center of the image. The component having the highest score is selected for the final mask.

4.2.6 Final Mask Application

In the final step, we apply the `best_blob_mask` to the original image and get the final image only with the leaf, making the background black.

Figure 6: Initial image, Final mask with best blob and Final image

4.3 Convolutional Neural Network to Predict Leaf's Health

We pass the **segmentation output** (an image with the leaf segmented from the background; all background pixels should be black) as an input into our CNN, which will do a **binary classification** to predict whether the leaf is **healthy or not**.

4.3.1 How CNN classifies images

The CNN works mainly by convolving the original image with a lot of different kernels to extract out the **key features** in the image. Key features are the most important features in an image that indicate

whether the leaf in the image is healthy or not. The model will then learn to predict a leaf's health by only looking at the key features, which is a much more efficient and effective process than learning from all original features.

In our course COMSC-341CV, we determined the values of a convolutional kernel to achieve a image filtering effect, but the values of the convolutional kernels in the CNN will be determined by backpropagation, which is the Machine Learning algorithm that aims to minimize the loss and maximize the prediction accuracy. Because it is too hard for us to figure out what convolution kernel can bring out the key features in any leaf image, we need Machine Learning to figure this out for us. The specific model we design uses 48 kernels together with several other weight and bias terms, totaling over 128 thousand parameters, to make a single prediction.

4.3.2 Input Pipeline

We use the `image_dataset_from_directory()` function from Tensorflow `keras` to generate small batches of data on the fly. The benefit of this is that if we load data in conventional ways, Google Colab would run out of RAM and crash.

4.3.3 Train, Validation, and Test Partition

We used roughly 70% of the data as the train set, 20% as the validation set, and 10% as the test set. Because our data has been divided into 12 batches, 8, 2 and 2 batches are used respectively for the train, validation, and test set.

4.3.4 Model Architecture

```
cnn_1 = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(16, (5,5), activation='relu', kernel_initializer='he_normal', input_shape=(256, 256, 3)),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
cnn_1.compile(optimizer = 'adam',
              loss = tf.losses.BinaryCrossentropy(),
              metrics=['accuracy'])
```

```
cnn_1.summary() # need to specify input shape in model creation first
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 252, 252, 16)      1216

 max_pooling2d (MaxPooling2  (None, 126, 126, 16)      0
 D)

 conv2d_1 (Conv2D)           (None, 124, 124, 32)      4640

 max_pooling2d_1 (MaxPoolin  (None, 62, 62, 32)        0
 g2D)

 flatten (Flatten)           (None, 123008)            0

 dense (Dense)               (None, 1)                 123009

=================================================================
Total params: 128865 (503.38 KB)
Trainable params: 128865 (503.38 KB)
Non-trainable params: 0 (0.00 Byte)
```

The first convolutional layer (`Conv2D`) has 16 convolutional kernels, and each kernel is a 5x5x3 kernel. The second `Conv2D` has 32 kernels, each kernel is 3x3x16. Note that we are doing a 3D convolution (convolution over volume), so the third dimension of the convolutional kernel matches the depth of the input. For a more detailed explanation of how the size of the kernels, the input, and the output of each layer is calculated through the image convolution process, you can check out Autumn's walk-through animated video [3]

The Max Pooling layer after each `Conv2D` reduces every block of 2x2 pixels down to one maximum pixel value in the block. The Flatten layer flattened the output of the `Conv2D` and the MaxPooling layers, which is a three-dimensional array, into a one-dimensional array to pass into the Dense layer.

We noticed that in another model with exactly the same architecture but one more Dense layer of 256 nodes, none of the accuracy, precision, recall, or loss was better by more than 5%, but the training time increased to be much longer. Thus, we only had one Dense layer.

We used the He initialization method to initialize the weights of the first Convolutional layer. He initialization is usually used to initialize networks that used the rectified linear (ReLU) activation function [2]. We chose ReLU and Sigmoid as our activation functions as commonly done in CNN.

4.3.5 Model Performance

Over 15 epoches, the loss decreases and the accuracy increases, though with a lot of jumps, especially in the validation accuracy.

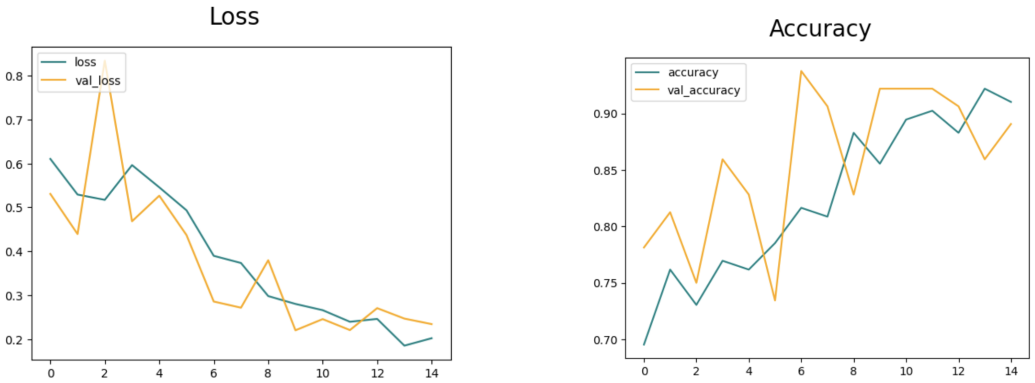

Figure 7: Loss and Accuracy of the model

We reach 84% precision score, which is the proportion of correct unhealthy predictions. Our recall score is 100%, meaning that all the unhealthy leaves are correctly identified. The overall binary classification accuracy is 87%.

4.3.6 Image Background In Training Data versus Leaf Segmentation Output

The dataset we trained the model on are close-up pictures of a single leaf on a plain grey background. Because the background in the images are all plain and gray, we believe that only features of the leaf will be considered by the CNN when it makes prediction. Therefore, we think our model can still make good predictions on the output from our segmentation step, where the background is all black.

4.3.7 Final Output

The final output of the model will be a float number ranging from 0 to 1. The closer it is to 1, the more likely it is that the input leaf is unhealthy. We choose 0.5 to be the threshold – the predicted class is Healthy if the output is less than 0.5, and Unhealthy otherwise. In the example input in this Figure 8, the prediction result for the segmented leaf is 0.902, which translates into 90.2% probability that it is unhealthy.

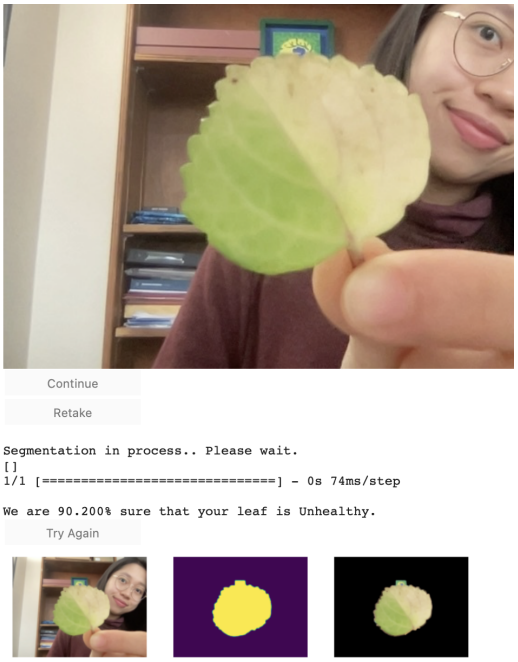

Figure 8: Final output of the whole program.

5 Conclusion

5.1 Limitations and Future Directions

Currently, even though our segmentation is working very well, there is still a thin border surrounded the segmented leaf, which is the area between the leaf and the background. That border might have confused the CNN and made it predict a healthy segmented leaf as unhealthy. In the future, we can work on making the segmentation more precise.

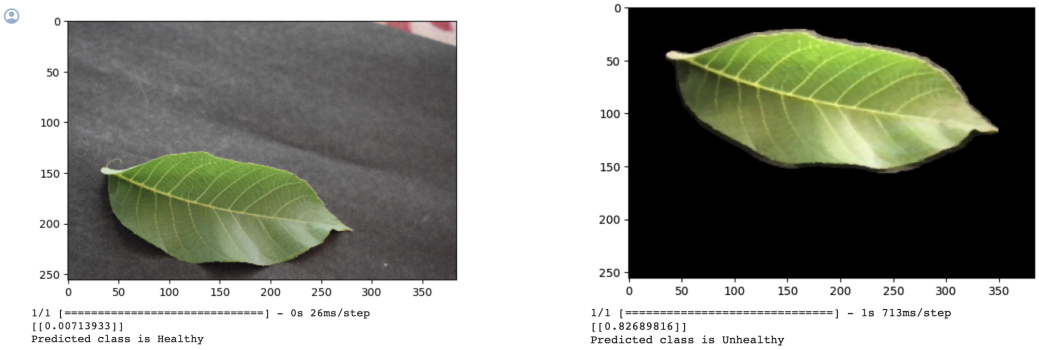

Figure 9: The difference in the prediction for the original image and the segmentation output. They are the same leaf, but in the original image, it is predicted as healthy, but as a segmented result, it is predicted as unhealthy.

The CNN is also quite black-boxed at the moment. To better understand how the model makes its the predictions, a future step would be to visualize feature maps of the CNN to see if it is correctly identifying the features that should be used to predict the leaf's health.

Moreover, our segmentation code can only work with green and yellow leaves, but potentially we want to work with leaves ranging in different colors and species to predict the health of the leaf

correctly. We also want to work with better dataset to train the model with diseases specific to species after making our segmentation code work without current limitations.

5.2 Lessons

This project has been an opportunity for us to learn computer vision techniques outside of the class. We formed a problem statement, found related work in the field of plant disease detection and implemented our solution to resolve the problem. It also gives us a chance to consider the user experience aspect while implementing the GUI of the program.

References

[1] Plantae. tensorflow datasets catalog. https://www.tensorflow.org/datasets/catalog/plantae.

[2] BROWNLEE, J. Weight initialization for deep learning neural networks. Blog, 2021.

[3] NGUYEN, A. Image convolution in cnn explained! https://www.youtube.com/watch?v=XELqHT8wkvg, 2023.

[4] NICHOLAS, R. Build a deep cnn image classifier with any images. https://youtu.be/jztwpsIzEGc?si=aWylvHLj49LvgNU9, 2022.

[5] S. P. MOHANTY, D. P. H., AND SALATHÉ, M. Using deep learning for image-based plant disease detection. *Front Plant Sci 7*, 3 (Sep 22, 2016), 1419.

[6] THE UNIVERSITY OF AUCKLAND, N. Z. Compsci 373: Computer graphics and image processing. https://www.cs.auckland.ac.nz/compsci373s1c/PatricesLectures/Edge%20detection-Sobel_2up.pdf.