Semi-Supervised Object Recognition to classify leaves as Poison Ivy

Kushal Kale (ksk7657@rit.edu) Swetna Tribhuvan (st9252@rit.edu)

Abstract

People usually have a hard time recognizing the poison ivy plant and its leaves. As the name rightly says, Poison Ivy is poisonous and we want people to avoid touching it. In this project, the objective is to develop a program that isolates and recognizes poison ivy, given a picture clicked by a human. The program takes images of leaves as input and applies several image processing and computer vision techniques like segmentation, edge detection,etc. on them to extract information which can then be used to build a binary classifier which can tell if the plant in an image is Poison Ivy or not. The images being fed to the program are preprocessed first. This is because the images often contained other irrelevant objects like grass, dead leaves, etc. Also, some of the images were not ideally shot and so some contrast enhancing methods are used to make the images better suitable for further processing. The program successfully segments the input images while keeping the necessary details from the image. The segmentation output is then used to extract features unique to the Poison Ivy plant which are used to recognize it in a given image.

Introduction

Poison Ivy (*Figure 1*) is an allergenic plant found in the USA. If humans touch it, it causes contact dermatitis which has unpleasant symptoms like itchy skin and painful rashes. This plant is present both on and around the RIT campus in Rochester. People on the campus usually can not identify the plant or are completely unaware about it and so end up coming in contact with the plant which results in them suffering the above mentioned effects. The goal is to keep people on and around the RIT campus safe from this plant.

Figure 1

One way to achieve the goal is to teach people from a young age that the poison ivy plant has three leaves and to tell them to avoid contact with it. But, multiple other plants like strawberries, raspberries, virginia creeper, etc also have three leaves and are not poisonous and so this way is not very effective because after a significant amount of time people will avoid all three leaf plants.

Another way to tackle this problem is to use computer vision and machine learning techniques to help people recognize poison ivy in real time by clicking a picture of the plant and telling if it is poison ivy or not.

To be able to build a software that is able to do this, we need a big training dataset of images which we can use to build a classifier. And to build a classifier, we need to extract relevant information from images which distinguishes poison ivy from other plants (other 3 leaf plants specifically). We then use this information to create features specific to poison ivy and use these features to build the classifier which can recognize whether a plant in an image is poison ivy or not.

However, getting relevant information from images and then converting it to usable features and deciding which features to use and which to discard are arduous tasks. The first challenge is that the training dataset can contain images which are clicked using different lenses and cameras, by different people, under different lighting conditions. Basically, the images are not clicked in a controlled environment and so are not standardized for processing. To overcome the first challenge, we use a bunch of image processing techniques which include preprocessing the images to change their size to a standard value, then checking if the image is suitable for further processing by checking if the exposure and contrast are right and if not fix it. Once this is done, we need to segment the image into foreground and background which will give us the subject of interest, the leaf.

The second challenge is that even after segmentation, the images will have multiple attributes which need to be converted into features. Also, there is no direct way to know which features to use and which ones to reject. As the no free lunch theorem says, we have to test all of them to find out which one works the best in this case.

In this project, we first focus on overcoming the first challenge. That is, we try to pre-process images so that they can be fed to complex image processing techniques further for segmentation,etc. so that we are able to extract physical properties of its leaves to create features. We then create a repository of such segmented images which can be used as the training dataset for a classifier. Second, we use the extracted features to classify whether a given image has poison ivy plant in it or not.

Methodology

1. Firstly, we added the TEST_IMAGES folder to the path so that everytime we run the code, it will be able to read in all the images directly from the path once they are all put in that folder.

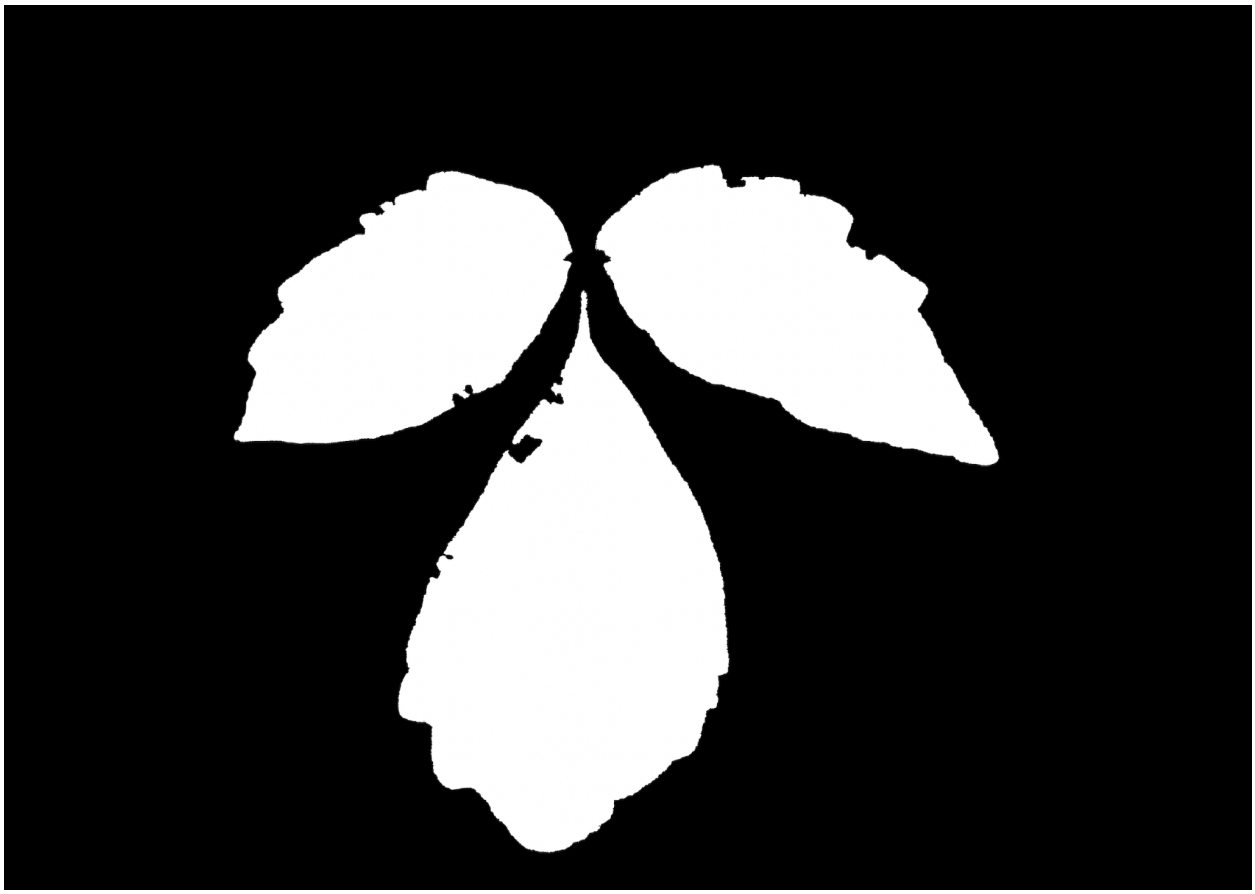
2. Then, we read each image from the disk and crop the outer region to get rid of some unnecessary objects in the image as the subject of interest is always at the center. This also fastens the execution.

3. We call the **segment_and_get_edges** method which returns a feature matrix of that image. In this function, we do the following:
    a. Call the segment function in which we first calculate the contrast value of the current image and check if contrast enhancement is necessary by comparing the value to a threshold found by trial and error.
    b. If it is necessary, we convert the image from RGB to LAB format and extract the L channel from it.
    c. Then we perform adaptive histogram equalization on the normalized L channel and keep the A and B channels the same.
    d. We convert the image back to RGB. This process increased the contrast because we only manipulated the L channel that is the lightness of an image which controls the exposure (and consequently the contrast).

Original Image	Cropped and Contrast Enhanced Image
	

4. Once the image is ready, we use the **ginput** method to locate the subject (poison_ivy leaves) and background pixels by clicking on both multiple times at different locations to get multiple pixel coordinates and then store the color values of both from those positions.

5. Then, using the Mahalanobis distance, we found out the distance of every pixel in the image from both the selected foreground and background data.
6. Using this measure we create a new binary image based on whether a pixel value is closer to the background color or the subject color. If it is closer to the subject color, the resulting pixel is white, otherwise it is black. We use Mahalanobis distance here because it enables us to calculate the distance of a point (pixel value) from a distribution (pixel data of the entire image). This helps us segment the image into leaf and background. Below we can see the output of this procedure when 2 different images were given as inputs. We can see that the code properly differentiates between the background and the leaf. And it has segmented the 3 leaves as expected.



7. One important thing to note here is that the quality and accuracy of the segmentation depends on how well we provide the color data to the **mahal** function. By trial and error we have found that it works well if we select 7 to 10 points each for both leaf and background.

8. As seen from the segmented output image, we have successfully separated the leaf from the background. We can use canny edge detection on this segmented image to get a hollow representation of the leaf (the program saves this in the out/ directory). Using this image we can also calculate properties of the leaf like area, lengths of major and minor axes, perimeter, circularity, etc. These attributes can be used to create useful features for classification (e.g. ratio of the minor axis to the major axis). However, in doing so, we have lost the details present on the leaf like the veins patterns and the presence of spots which is unique to poison ivy and thus can be useful in classification.

9. So, to retain this aspect of the input image we use the edge details of the original image. We do this in following steps:

a. We create spatial filters to find out horizontal and vertical edges in the given image by using the **imfilter** function as follows:

```
dIdy = imfilter( rgb2gray(im), fltr_dIdy);
dIdx = imfilter( rgb2gray(im), fltr_dIdx);
```

We can note here that before passing the image to the filter function, we have converted the image to its grayscale version because the **imfilter** function needs 2-dimensional input.

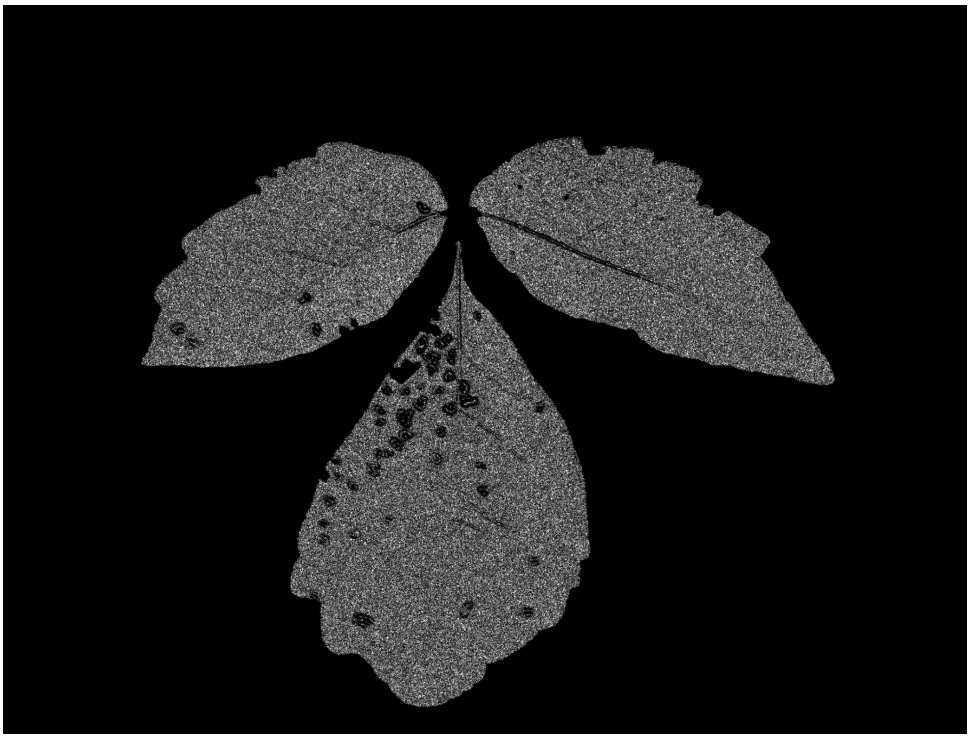
b. Once we get both the edges from the image, we take the square root of the sum of the square of the individual edge magnitudes and store that in a variable. This variable essentially contains the edge magnitude information of the image. That means all the details which were present in the original image can be retained and we can distinguish between them (and the objects to which the details belong) by comparing the magnitudes using this variable as it has stored the details as magnitudes.

c. The program writes the edge magnitude image to disk. For reference, what it looks like follows for one of the input images:

The edge magnitudes are very faint and we need to look at the image very carefully to be able to see it.

d. Then, by trial and error we find a threshold value for the edge magnitudes. For this program we chose the threshold to be the 45th percentile magnitude value because it gave us a good balance between sharp edges and actual fine details from the image.

e. Using the **edge-magnitudes image** and the **threshold value**, and the **segmented image** we generate a new image for which the pixel value at each coordinate is equal to the pixel value at the same location in the thresholded image if that location in the segmented image is present on the leaf. Else the value is 1. This manipulation gives us a segmented image which also has the details of the original image present on it in the form of fine edges.
The image after this manipulation looks like this:



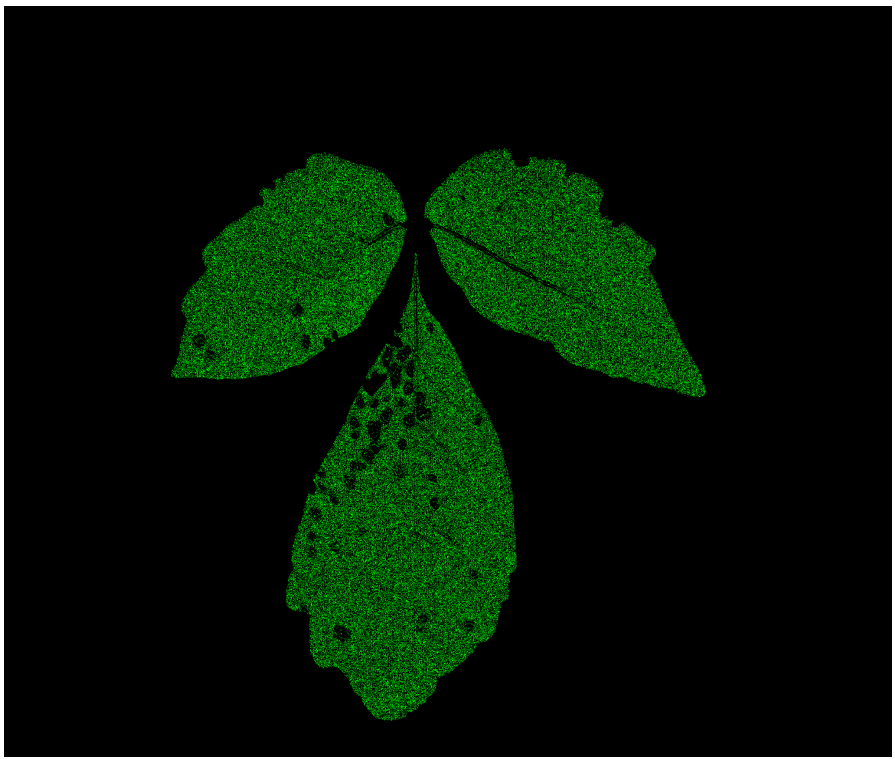
f. As we can see from the output images above, certain details of the leaf like the vein network pattern and locations of the spots are not lost. While not primarily important like the area or the shape of the leaf which we get from the basic segmented image, the details from this image could help us in situations when our classifiers have a split decision about the result class of the leaf.
g. Especially if we use a classifier like CNN which takes images as input and creates features on its own based on the hyper parameters and the details in the image, such images could be more effective than a basic segmented image like the one displayed earlier.

10. After having both the basic segmented image and the segmented images with details, we are ready to extract attributes from both to build and train a classifier using those attributes either directly or after creating features from them. In the interest of simplicity, we have chosen to work on the basic segmented image only for now.

11. We are using the **regionprops** function from matlab to get image attributes. Specifically, we are extracting the following attributes for every image:
    a. Area: This is the area of the leaf in the picture. This attribute is not very reliable because the area of the leaf depends upon the focal length of the lens used to click the picture. If we make a naive assumption of standardized images, this attribute could be useful.
    b. MinorAxisLength: As the name suggests, this is the length of the smaller axis of the leaf. Again, like area, this attribute is not very reliable on its own for the same reasons.
    c. MajorAxisLength: As the name suggests, this is the length of the larger axis of the leaf. Again, like area, this attribute is not very reliable on its own for the same reasons.
    d. Circularity: This is a measure of how circular the leaf is. Since the Ivy leaf has a very unique shape, this is a very useful attribute for us. If used with the number of leaves it can have a heavy weight in the classification formula.

12. Attributes b and c from above can be used to create a very useful feature **minorAxis/majorAxis**. Both attributes are not reliable on their own but if we use this feature we eliminate the uncertainty caused by the parameters like the focal length, crop factor, etc.

13. Below, you can see the attributes extracted from the following image using the above mentioned method:



Area	MajorAxisLength	MinorAxisLength	Circularity	Perimeter
9.4322e+05	1535	790.1	0.48663	4935.3
1.6856e+06	1725.7	1285.5	0.42942	7023.3
1.0539e+06	1781.2	781.19	0.47077	5303.9

14. We can see that the three leaves are discovered correctly for this image as there are only three entries in the details table. If due to improper segmentation we get more than three objects, we can ignore the ones which do not match the area requirements.
    E.g.

Area	MajorAxisLength	MinorAxisLength	Circularity	Perimeter
9.6566e+05	1397.5	920.05	0.32946	6069
9.5388e+05	1511.9	856.19	0.2843	6493.3
1.1304e+06	1573.3	948.82	0.27339	7208.4
16038	247.99	143.24	0.067791	1724.2
15451	381.81	58.464	0.068186	1687.5

    We can see that we got 2 extra objects here. But we can easily discard the bottom 2 entries as the values of all the parameters suggest that these are just stray objects in the image which are not big enough to be considered but are left in the

image due to segmentation issues.

15. Also, if we compare these values to a non ivy segmented image, we can clearly see the difference in the values. And so, we can be sure that when we feed these attributes and features to a classifier we will get sufficiently accurate results.
    E.g.

Area	MajorAxisLength	MinorAxisLength	Circularity	Perimeter
82297	1743.6	236.2	0.018031	7573.4
3.2597e+05	1068	537.17	0.12573	5708
2.8098e+05	677.59	541.09	0.28447	3523.1
2.8612e+05	720.28	513.34	0.33973	3253.3

    The minor to major axis ratio is closer to 1 for non ivy leaves. The area for non ivy leaves is also multiple times smaller than the ivy leaves area. The vein network pattern is also clearly different for this leaf.

16. This attribute data is stored in a matrix form which the code prints out for reference and can be the input for our classifier building code. In the interest of time, we are skipping the classifier building code, but this information that we have extracted from the input images can be used to train different types of classifiers like decision trees, SVMs, Neural Networks, etc. The bigger our dataset becomes, the better the accuracy of our model will be.

Discussion and Conclusion

This project, overall, was a telling undertaking in terms of how powerful a set of images can be and how much wealth of information is hidden in the visual form. If we devise the right tools and techniques to mine and analyze this information, we can uncover fascinating insights which would stay hidden otherwise.

Initially, when we saw the set of raw input images, it was difficult to imagine how those images would be used to classify. Through the checkpoints, as we went through the course work and learned about various image processing techniques in class, the picture became clearer.

We faced problems in segmentation of some images because they were underexposed or had poor contrast due to bad lighting. After we were taught how contrast enhancement works, we studied the nuances of it and found a way to apply one of the techniques learned in class to increase the contrast of problematic input images. We did color space manipulation, histogram equalization to achieve the desired result.

Next, we spent a lot of time getting the segmentation part right. We tried various techniques. We started with using simple color space manipulation and image thresholding to convert the image into a binary image based on an intensity threshold. It worked for some images but was not general enough. It failed terribly in images where there were a lot of fillers like grass, insects, etc with the same intensity values. So, we tried k-means clustering next. It also worked well for some images but did not generalize well. The initial centroid positions needed to vary to be able to cluster and segment well. After a few more iterations of such trials, we settled with what we have done now. I was surprised by how morphology came in hand often throughout the project when we needed to fix minor problems with our intermittent images. We were also surprised by how difficult it is to build robust and resilient code because we were able to break our own algorithm just by changing the way we selected points for the ginput method.

Getting the texture details of the leaves and then extracting attributes and creating parameters from those was also a process full of finding things out and understanding the concepts first. This process of trying things out, reading through the documentation and texts describing how the concepts and technique come together taught us how to approach and tackle a big task. Along the way we also discovered and learned very powerful tools and techniques in matlab which reinforced our understanding of the concepts taught in class.

We had so much fun learning and implementing the concepts like segmentation that we plan on working further in our free time on the classification part to check how accurately the model we build works. We also plan on trying to use matlab code for editing the personal photos for instagram to see how it compares with softwares like lightroom and snapseed.