Team: Bardh Rushiti , Mohd Junaid Shaikh , Saksham Bansal

Abstract

Seasoned hikers and children alike often have a hard time recognizing poison ivy in their environment. Our group has devised a semi-supervised image recognition algorithm in matlab to inform users if there is a poison ivy in a given picture. We used a combination of feature detection methods like CNN, corners and blob detection for classification. In this paper, we explain our process and detail the problems we faced along with suggesting improvements in our approach in order to perfect the detection methodology.

Introduction

We have grown up being taught phrases such as, "Leaves of three, let it be," to help us identify whether a given plant was poison ivy or not. However, a lot of harmless plants such as raspberry and virginia creeper plants have three leaves, and are often mistaken for poison ivy.

In this paper, we explain the semi-supervised algorithm we devised using matlab that detects the presence of poison ivy in an image. This algorithm assumes that the plant of interest will always be placed in the center of the image. The algorithm utilizes K-means for segmentation, morphological operations for noise reduction, watershed segmentation for leaf separation, convolutional neural networks (CNN) and automatic corner detectors for classification.

Methodology

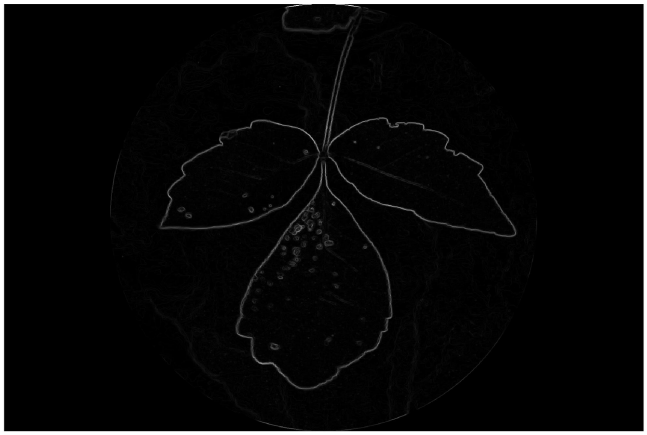
Pre-Processing: Initial Morphology and Noise Reduction

We ran several preprocessing steps. First, we utilized a semi-supervised learning algorithm, of which the initial assumption was that the image of the plant will be centered. We began by finding the ratio of the image dimensions. If the ratio was less than zero, we know the image height was larger than image width, and thus used an equation to calculate the radius from the center of the image. Using that radius, we generated a circular mask and removed the area outside this circle, as a result we avoided a lot of background noise in the image.

Second, we used image quantization where we bucketed the pixels of the image into 16 parts to further remove noise and enable less processing and more consistent results for later parts of the algorithm.

Processing: Strong Edges

Next, we worked on finding the strongest edges for the image, which we found to be one of the most important attributes that helps k-means produce better results. Strong edges were particularly useful in cases where leaves are touching each other and/or the background is indistinguishable. To find the best edges we used a sobel edge detector that detects edges in vertical and horizontal directions. Finally, we identified the edges with a magnitude above the 95th percentile to achieve the strongest edges. The result of which was shown down below.

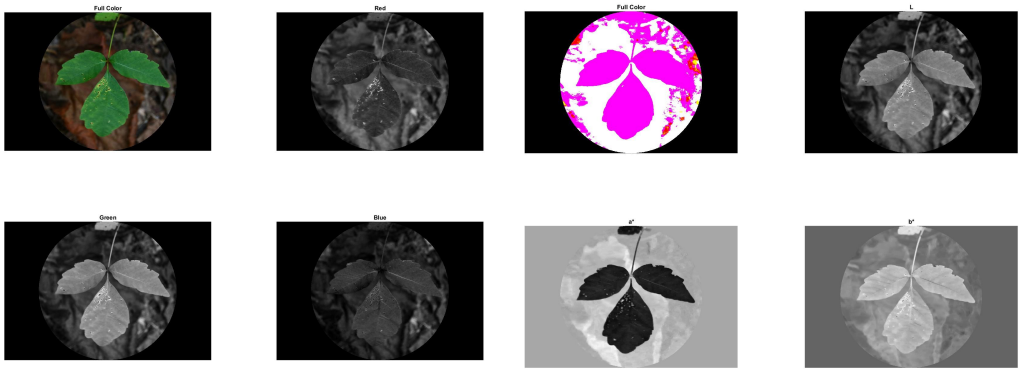


Processing: K-Means and Noise Removal

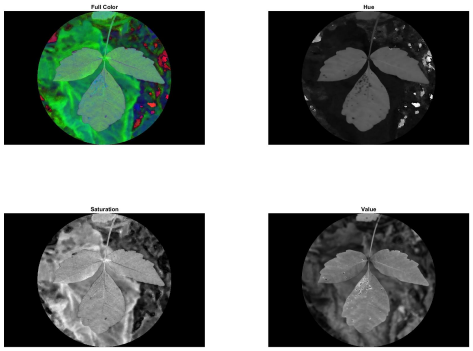
During exploration of color spaces, multiple color channels were inputted into K-means, but not all channels were equally valuable when making the segmentation. As seen below, there were only a handful of channels that were useful. For example, green channel from RGB and a* channels from L*a*b look significant for creating the segmentation of the leaf. During this phase, multiple other adjustments were tried, however, we had the most success using 5% weight for rows and columns, 200% weight for edge magnitudes, and a* from L*a*b. These adjustments proved to be useful for overlapping leaves. It was found that the green channel was susceptible to changes in the lighting conditions across different images while the a* channel remained consistent. After experimenting with different clusters for K-means, they were set to nine.

Team: Bardh Rushiti , Mohd Junaid Shaikh , Saksham Bansal

k =9, data = [col(:) * 0.05, edgeMag(:) * 2, row(:) * 0.05, lab_a(:) * 2];

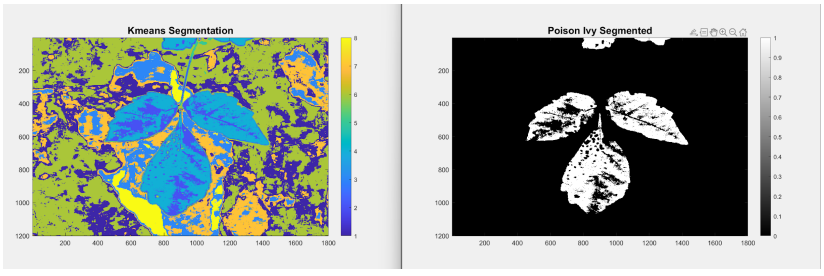


RGB colorspace and its respectful channels CIEL*a*b colorspace and its respectful channels

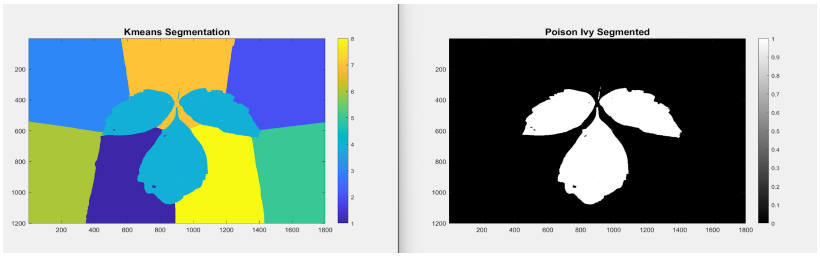


HSV colorspace and its respectful channels

Moreover, during feature space exploration, we discovered that rows and columns were also helpful for segmenting out the leaves. The difference between using and omitting the row/column information can be observed down below - and the distinction was clear.

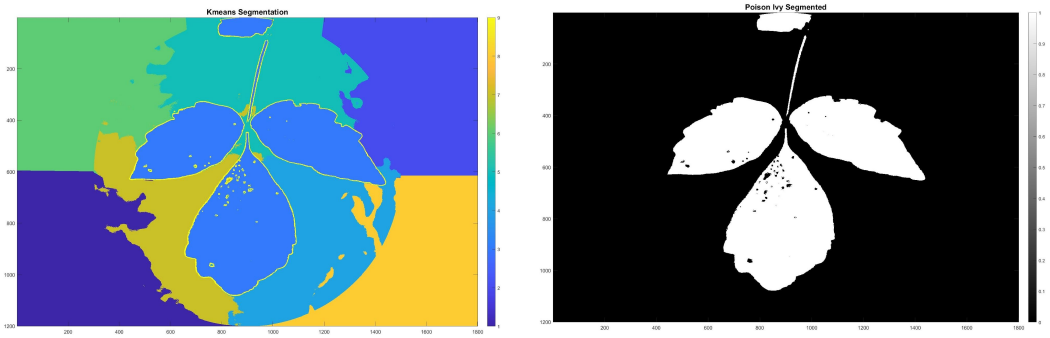


K-means without row and column information

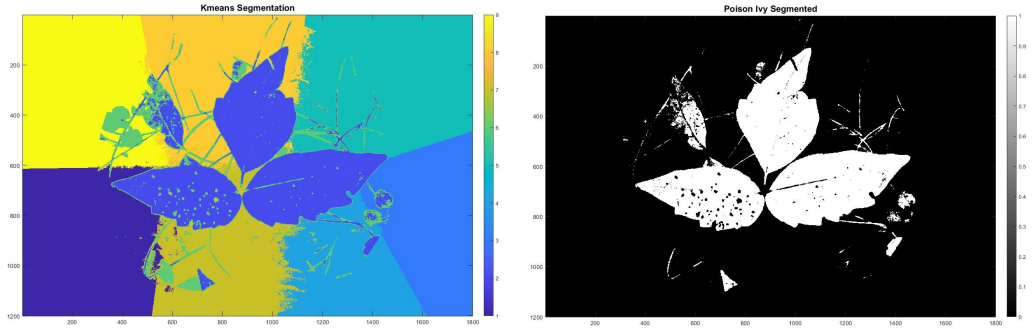


Team: Bardh Rushiti , Mohd Junaid Shaikh , Saksham Bansal

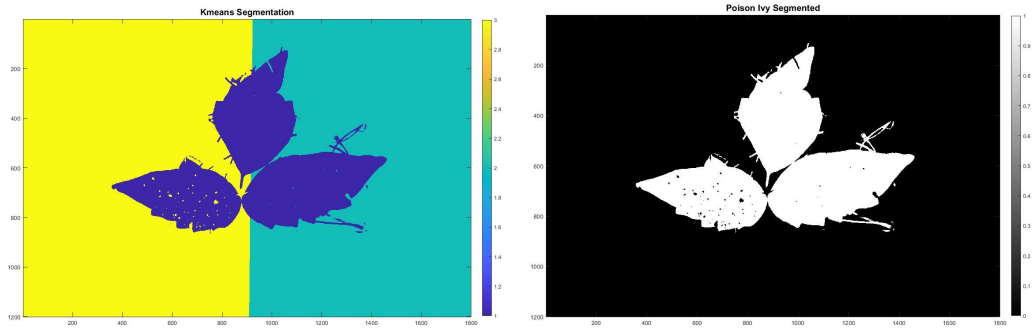
The final result of K-means was presented down below. For detecting the index of the leaves, a square region of interest was generated in the middle of the image. Given the assumption that the leaf was always in the center of the image, we simply calculated the mode (most common value) in the region of interest and the output was presented down below.



Even though using the circle and quantizing the image removes a lot of the noise, those steps were not sufficient. Therefore, we used K-means twice, first, we utilized K-mean, where k=9 to get a vague idea where the leaf was and remove noise in the circle. We applied K-means a second time where k=3, to segment the leaf efficiently and remove the noise with great precision. The figure below shows the output for one such image after using K-means with 9 clusters and then running K-means on the new masked image with 3 clusters.

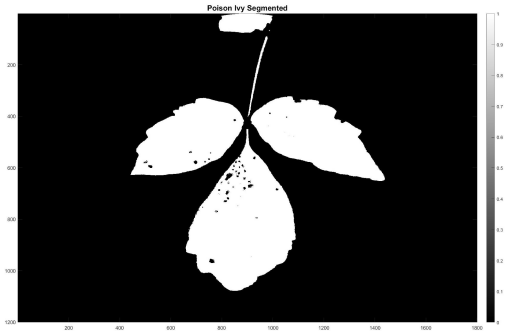


K-means on original image with 9 clusters

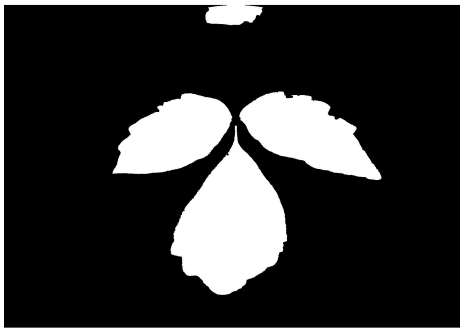


K-means on masked image with 3 clusters

After achieving the segmentation through K-means, we observed some holes inside the leaves and still have some lingering redundant blob noise. This noise can cause issues during feature detection and classification.



To fill those holes we used imfill() with ‘holes’ argument to remove any holes inside the leaves. Even though not shown here, white residue from the first K-means still remains in the image, we used bwareaopen() and removed anything that was less than 30000 pixels area. Next, we used erosion with a structuring element of 1, and then on top of it we removed the potential white blobs generated by erosion of area 10000 pixels. The resulting image was much cleaner.

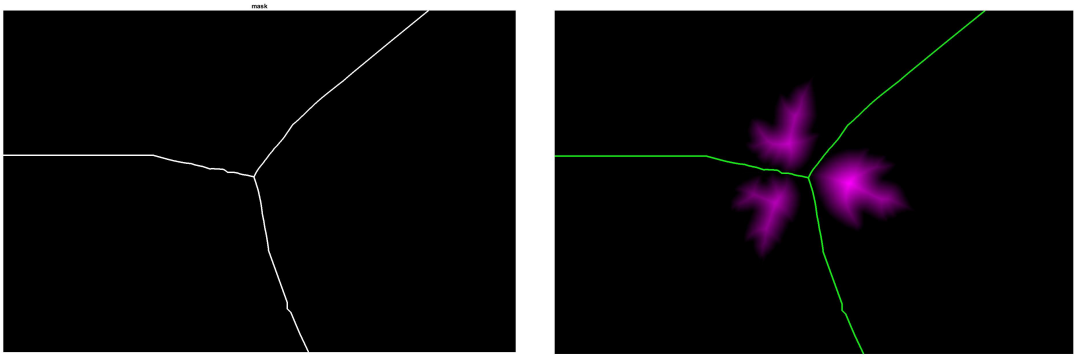


Post-Processing: Watershed (Leaf Separation)

There were a few examples of one gigantic component being segmented, with that the algorithm was having a hard time detecting the leaves and corners (next stage). In this case, we used watershed technique to separate the leaf. Below was an example of watershed at play, where there was a big blob being connected by a few pixels. This was handled by bwdist() which computes the Euclidean distance transform of the binary image. Next, we calculated imextendedmin() which provided the extended-minima transform, which was the regional minima of the H-minima transform with the inverse of the result from the previous step.

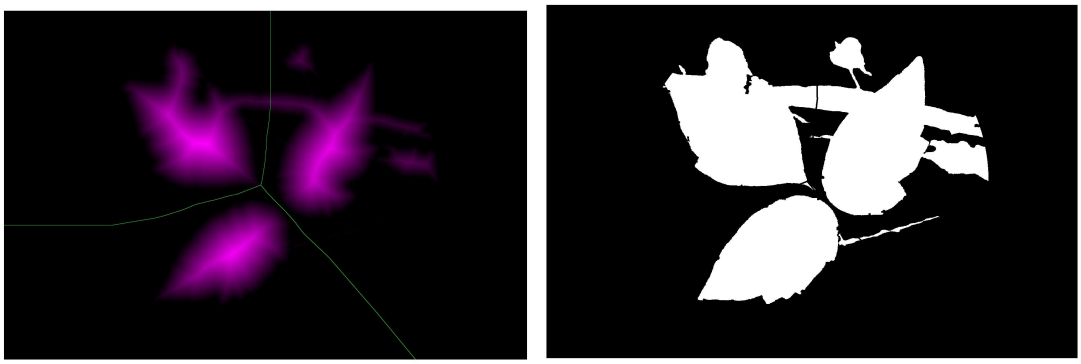
Team: Bardh Rushiti , Mohd Junaid Shaikh , Saksham Bansal

The Watershed algorithm regards the black values as holes - calculated via the Euclidean distance from the pixel with the nearest 1 value. Now, if those regional holes were filled slowly with water until a certain level, the connected components of pixels can be separated with great accuracy; with a boundary that separated these 3 dice. There exists a function that does this, with imextendedmin() and a level of 70 and this result can be achieved as seen down below.



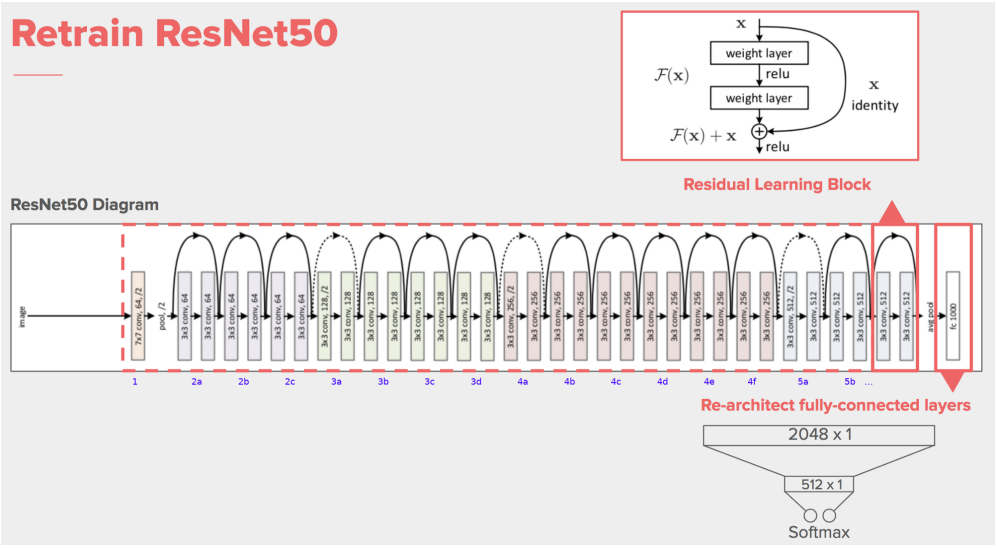
However, this method would not work well if there was noise (in forms of blobs not deleted, dust noise, or anything of that nature) as shown below. In order to solve this, we must either
a) have better morphology to clear the noise
b) find a better method for separating the leaves in the presence of noise.

Team: Bardh Rushiti , Mohd Junaid Shaikh , Saksham Bansal



Neural Network Classifier: ResNet50

ResNet stands for Residual Network. It was an innovative neural network that was first introduced by He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian in 2016 computer vision research paper titled 'Deep Residual Learning for Image Recognition'. ResNet has many variants that run on the same concept but have different numbers of convolution layers. We decided to go with Resnet50, which was used to denote the variant that works with 50 layers of convolutions. Unlike other architectures, the innovation in this architecture was the residual (skip) connections. They targeted a big problem in the neural network training which is vanishing gradients - the skip connections were an alternate shortcut for the gradient to pass through.

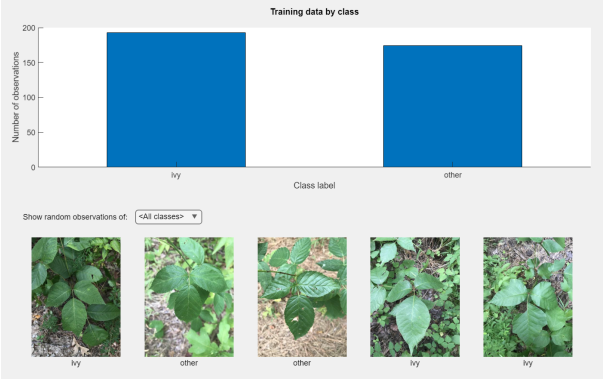


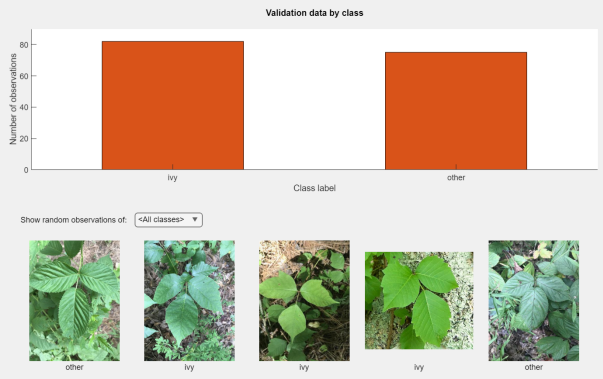
After many iterations and training, this architecture proved to have enough variance in order to learn the problem, and produced low bias results.

Team: Bardh Rushiti , Mohd Junaid Shaikh , Saksham Bansal

Data:

- The model was trained on a dataset of more than 500 images: ~250 of which were poison ivy, and ~250 were non-poison ivy (other)
- Distribution of Data:





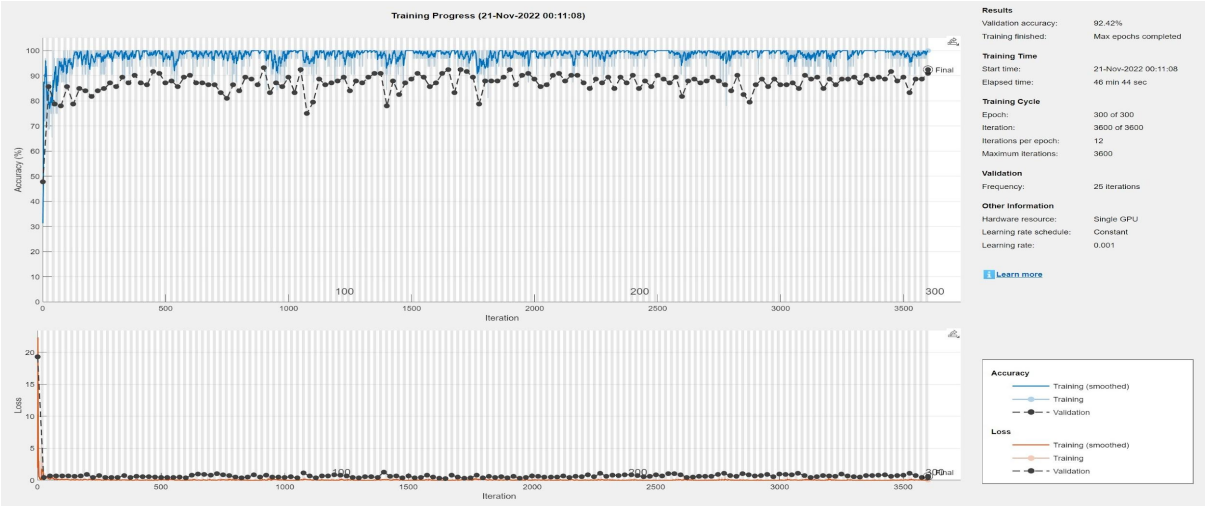
- Big help for building the dataset was from Vasily Kerov.

Procedure:

- Input: ResNet50's input is 224x224x3
- The network was rearchitected to fit the needs of this assignment, which had 2 final classes: Poison Ivy and Non-Poison Ivy, hence the output layer had 2 classes
- Pretrained layers were used, with an additional fully-connected layer on top of ResNet50, to take into account the classes. The whole network was trained for 100 epochs.
- Train/ Validation Split was 70/30 % respectively.

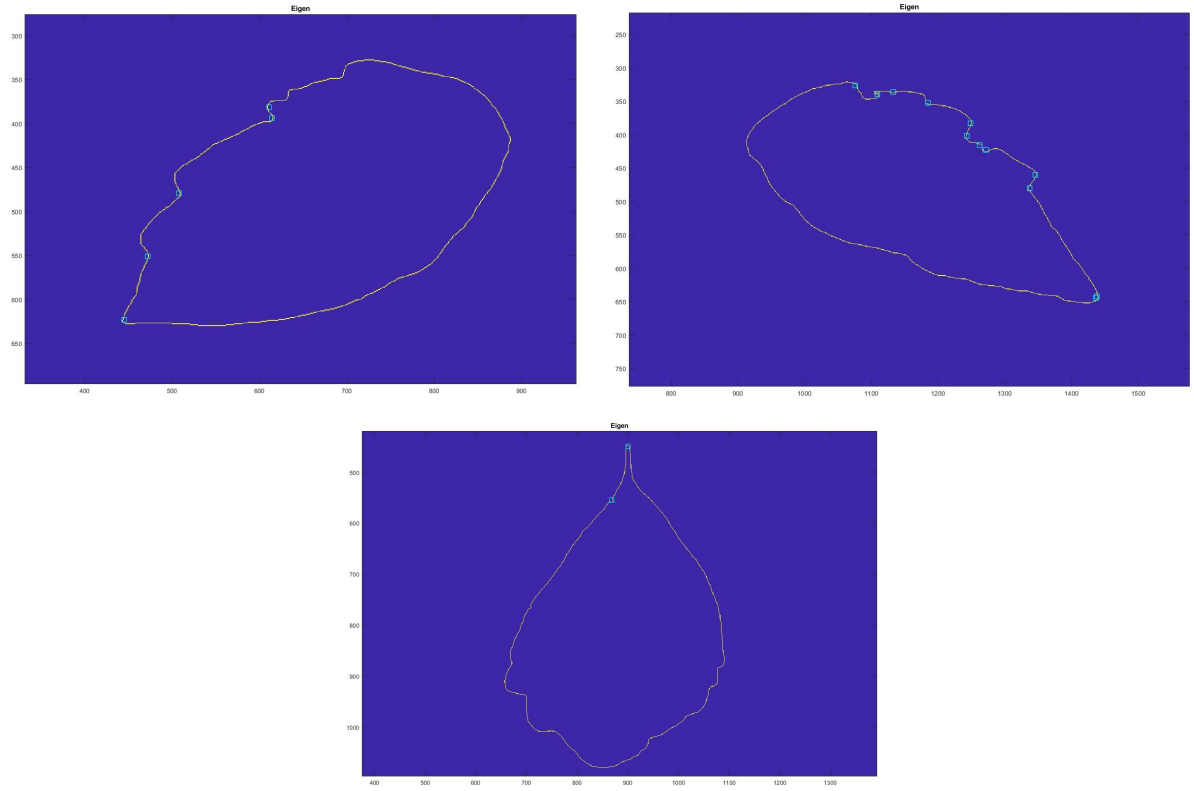
- During training, we augmented dataset with these operations:
  - Rotation 0 - 359°
  - Horizontal and vertical flip

Final Accuracy is **92.42%**



Post-Processing: Feature Detection with EigenCorners

Finally, for classifying and determining if an image was poison ivy we used the eigenvalue corners. To achieve better direction change for the corner detection algorithm we used the outline of the leaf computed by the sobel edge detector. We had to tweak corner detection to achieve the best results. We ended up using “MinQuality = 0.65” and “FilterSize = 15” to get the corners along the thumbs of the leaves. We found that eigenvalue corners were detected properly when the algorithm was run on each leaf separately. The corners found in each leaf were summed up and it was observed that a poison ivy has eigenvalue corners between 15 and 25.



Classification Score

In order to classify the leaf as a poison ivy, we came up with a metric of our own, a classification score. Initially this score was set to 3 and the image was classified as poison ivy . We decrement this score every step of the way whenever a leaf would fail the “expected poison ivy” feature check. First, we checked the classification output from the trained neural network. Second, we confirmed whether the image has exactly 3 leaves or not. Lastly, we counted the number of corners around the leaf. This helped us to make sure that we only classify the leaf as a poison ivy when all three conditions were met. For images that do not meet any conditions we classify them as non-poison ivy. Anything in between we classify as “can be poison ivy,” because our algorithm was unclear on that image. We justified this distinction in our algorithm because it was safer to misclassify a plant that was not poison ivy as poison ivy, rather than misclassifying poison ivy when our algorithm was unsure. This helped to enforce the idea that it is better to classify a leaf that is not a poison ivy as a poison ivy instead of the other way around.

Results And Discussions

Using the above methodology we were able to find if an image was a poison ivy or not fairly accurately. However, with the variable nature of K-means and varying image quality there were cases where we ended up with distorted results. Hence, to increase the accuracy of our classification we added a convolution neural network as a supplementary tool. The output from both feature extraction from K-means and CNN was taken into consideration in a scoring system as part of risk analysis.

Final Output

The images below show the final output of the program.

```
INPUT Filename:     IMG_3127.JPG
K-Means 1 - Running...
K-Means 2 - Running...
Neural Network Running...
Counting number of leaves...
Finding corners...
Final Decision: POISON IVY
>>
```

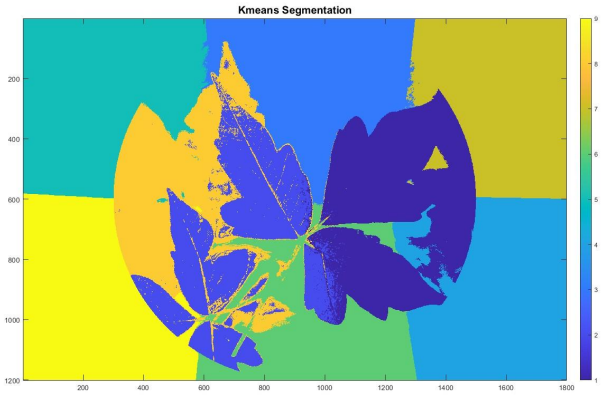
```
INPUT Filename:     IMG_3224_2000_x_3000.JPG
K-Means 1 - Running...
K-Means 2 - Running...
Neural Network Running...
Counting number of leaves...
Finding corners...
Final Decision: NOT A POISON IVY
>>
```

```
INPUT Filename:     IMG_3242_2000_x_3000.JPG
K-Means 1 - Running...
K-Means 2 - Running...
Neural Network Running...
Counting number of leaves...
Finding corners...
Final Decision: Can be a Poison Ivy
>>
```

Special Case

As mentioned earlier because of variable behavior of K-means and images being blurry or having exposure issues, our algorithm sometimes resulted in bad segmentation

Input Poison Ivy Image



K-means segmentation

Choosing modal cluster from the center

Conclusion

This project reinforced all the concepts that we have learned in this course so far. Working on our algorithm enabled us to think critically when it came to making decisions and choosing tools to solve the problem at hand.

We took advantage of the fact that the plant of interest will always be in the center of the image, created a circular mask, and added a layer of noise removal by quantizing the image into 16 buckets. Leveraging knowledge from homework that was presented in class, we found strong edges in the image using sobel filter and edge magnitudes. When it came to segmenting the leaf, we experimented with K-means and used color channels from HSV, RGB and LAB color spaces with a combination of weights as attributes. This allowed us to better understand how each component affects the segmentation produced by K-means. It was interesting to see how well K-means was able to segment the image when strong edge information was provided. In certain

cases where the variable nature of K-means was prominent, we found it useful to use K-means twice which helped in reducing the noise and improved the segmentation further. When it came to detecting corners automatically, we worked with Harris Corners, SURF corners and Eigen Corners. Doing so allowed us to choose the detector that worked best for our use case i.e., Eigen Corners.

We are proud of how we utilized watershed segmentation for the cases where K-means segmentation was not able to segment the leaves individually. When we tried to add Artificial Neural Network as a component along with Classical Computer Vision methodologies, we discovered Deep Network Designer provided by MATLAB, in their Deep Learning Toolbox, allowed us to design, analyze, and train neural networks of various architectures. Since risk analysis was being performed and a false negative classification can have a severe impact on the user; we used a classification score which made sure that our program classified the leaf as not poison ivy only when all the feature extraction components fail to classify the leaf as poison ivy.

References

- Poison Ivy Dataset - https://github.com/bazilione/poison_ivy
- ResNet50 Architecture Description - https://viso.ai/deep-learning/resnet-residual-neural-network/
- Various Resources - https://www.mathworks.com/