# A Texture and Curvature Bimodal Leaf Recognition Model for Identification of Costa Rican Plant Species

Mata-Montero, Erick (emata@itcr.ac.cr); Carranza-Rojas, Jose (jcarranza@itcr.ac.cr)
Computer Engineering Department
Costa Rica Institute of Technology
Cartago, Costa Rica

***Abstract*—In the last decade, research in Computer Vision has developed several algorithms to help botanists and non-experts to classify plants based on images of their leaves. LeafSnap is a mobile application that uses a multiscale curvature model of the leaf margin to classify leaf images into species. It has achieved high levels of accuracy on 184 tree species from Northeast US. We extend the research that led to the development of LeafSnap along two lines. First, LeafSnap's underlying algorithms are applied to a set of 66 tree species from Costa Rica. Then, texture is used as an additional criterion to measure the level of improvement achieved in the automatic identification of Costa Rica tree species. A 25.6% improvement was achieved for a Costa Rican clean image dataset and 42.5% for a Costa Rican noisy image dataset. In both cases, our results show this increment as statistically significant.**

***Index Terms*—Biodiversity Informatics, Computer Vision, Image Processing, Leaf Recognition**

## I. Introduction

Plant species identification is fundamental to conduct studies of biodiversity richness of a region, inventories, monitoring of populations of endangered plants and animals, climate change impact on forest coverage, bioliteracy, invasive species distribution modelling, payment for environmental services, and weed control, among many other major challenges for biodiversity conservation. Unfortunately, the traditional approach used by taxonomists to identify species is tedious, inefficient and error-prone [1]. In addition, it seriously limits public access to this knowledge and participation as, for instance, citizen scientists. In spite of enormous progress in the application of computer vision algorithms in other areas such as medical imaging, OCR, and biometrics [2], only recently they have been applied to this context. In the last decade, research in Computer Vision has produced algorithms to help botanists and non-experts classify plants based on images of their leaves [3], [4], [5], [6], [7], [8], [9], [10], [11], [12]. However only a few studies have resulted in efficient systems that are used by the general public, such as [13]. The most popular system to date is LeafSnap [13]. It is considered a state-of-the-art mobile leaf recognition application that uses an efficient multiscale curvature model to classify leaf images into species. LeafSnap was applied to 184 tree species from Northeast USA, resulting in a very high accuracy method for species recognition for that region. It has been downloaded by more than 1 million users [13]. LeafSnap has not been applied to identified trees from tropical countries such as Costa Rica. The challenge of recognizing tree species in biodiversity rich regions is expected to be considerably bigger.

Vein analysis is an important, discriminative element for species recognition that has been used in several studies such as [14], [15], [16], [17], [18]. According to Nelson Zamora, curator of the herbarium at the National Biodiversity Institute (INBio), venation is as important as the curvature of the margin of the leaf when classifying plant species in Costa Rica [19].

This paper focuses on studying the accuracy of a leaf recognition model based not only on the curvature of the leaf margin, but also on its texture (in which veins are visually very important). This is the first attempt to create such model in Costa Rica.

The rest of this manuscript is organized as follows: Section II presents relevant related work. Section III and Section IV cover methodological aspects and experiment design, respectively. Section V describes the results obtained. Section VI presents the conclusions and, finally, Section VII summarizes future work.

## II. Related Work

In LeafSnap [13] the authors create a leaf classification method based on unimodal curvature features and similarity search using k Nearest Neightbors (kNN). This method is tested against an image dataset from North American trees, using 184 species in total. Since their system requires images to have a uniform background, leaf segmentation works by estimating the foreground and background color distributions, and then classifying each pixel at a time into one of those two categories. A conversion to Hue Saturation Value (HSV) color domain is applied before using Expectation-Maximization (EM) [20] for the leaf segmentation. A 96.8% of accuracy is reported by the authors on their dataset with $k = 5$.

Researchers in [21] use Local Binary Patterns (LBP) features to classify medicinal and house plants from Indonesia. They extract LBP descriptors from different sample points and radius, calculate a histogram for each radius length feature set, and concatenate those histograms, similarly to Histogram of Curvature over Scale (HCoS) of LeafSnap [13]. As a classifier, a four layer Probabilistic Neural Network (PNN) is used. Their dataset consists of two subsets; one comprises 1,440 images of 30 species of tropical plants, and the other one has 300 images of 30 house plant species. The image background of the medicinal plants is uniform, while house plant images have non-uniform backgrounds. For medicinal plants the reported precision is 77% and for house plants 86.67%, revealing that using LBP for complex image backgrounds is a suitable technique.

Authors in [22] use Speeded Up Robust Features (SURF) features to develop an Android application for mobile leaf recognition. For the species classification task, SURF features are extracted from the gray scale image of the leaf. The feature set is reduced to histograms in order to reduce dimensionality since the resulting SURF feature vector may be too big. The precision reported is 95.94% on the Flavia dataset [6], which consists of 3,621 leaf images of 32 species.

## III. Methodology

This section describes how the leaf recognition process was set up. Section III-A describes the image datasets used. Section III-B summarizes the techniques used to segment each image into leaf and non-leaf pixels clusters. Section III-C presents several image enhancements conducted, such as cleaning up undesirable artifacts and elements, stem removal, clipping and resizing. Section III-D describes the feature extraction approach for both the curvature and texture model. Finally, Section III-E presents the species classification metrics and algorithms used in this research.

### A. *Image Datasets*

An image dataset of leaves from Costa Rica was created from scratch. To our knowledge, no other suitable Costa Rican datasets existed before. The dataset has both clean and noisy images, in order to identify how the amount of noise affects the algorithms. All images were captured from mainly two places: La Sabana Park, located in San Jose, and INBiopark, located in Santo Domingo, Heredia. In most cases, images for both surfaces of each leaf were taken. The dataset includes endemic species of Costa Rica and threatened species according to [19]. The complete list of species in the dataset can be found in [23]. The dataset consists of the following two subsets:

*a) Clean Subset:* Fresh leaf images were captured during field trips to both La Sabana and INBiopark. If the leaves were not flat enough, a press was used to flatten them for 24 hours. A total of 1468 leaf images were scanned. The images have

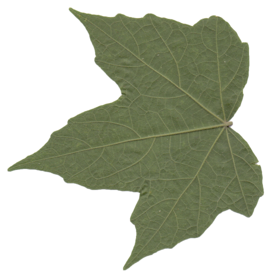

Figure 1: A *Robinsonella lindeniana var. divergens* sample scanned from a leaf sample from INBiopark, using a HP ScanJet 300 scanner, then cleaned using Photoshop CS6.

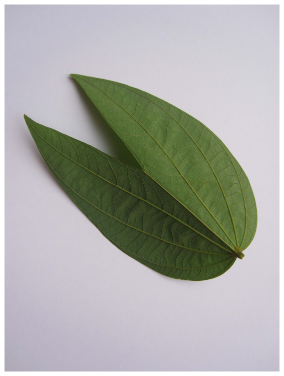

Figure 2: A *Bauhinia ungulata sample* taken using a Canon PowerShot SD780 IS camera at the Sabana Park.

a white uniform background and a size of 2548x3300 pixels, scanned with 300 dpi in JPEG format. Photoshop CS6 was used to remove shadows, dust particles and other undesired artifacts from the background. Figure 1 shows a sample of a cleaned Costa Rican leaf image of this subset. The scanner used was an HP ScanJet 300.

*b) Noisy Subset:* Fresh leaf images were captured during field trips to both La Sabana and INBiopark. No press was used to flatten them. A total of 2345 fresh leaf images were captured. This subset was captured against white uniform backgrounds (normally a sheet of paper). Each image has a 3000x4000 pixel resolution, in JPG format. No artifacts were removed manually. However as explained in Section III-C several automated image enhancements were performed both on the clean subset and the noisy subset. Figure 2 presents a noisy leaf image sample. The camera used is a Canon PowerShot SD780 IS.

### B. *Image Leaf Segmentation*

The first step to process the leaf image is to segment which pixels belong to a leaf and which do not. We used the same approach as LeafSnap by applying color-based segmentation.

*1) HSV Color Domain:* When segmenting with color it is imperative to use the right color domains in order to exclude

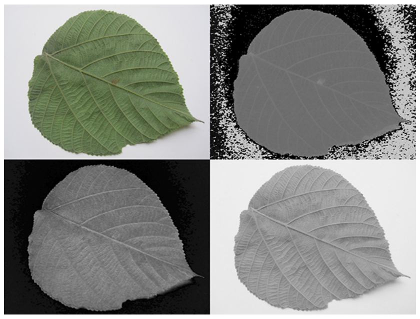

Figure 3: HSV decomposition of a leaf image
The top-left image shows the original sample. The top-right image shows the Hue channel of the image with noticeable noise. The bottom-left image shows the Saturation component and the bottom-right image shows the Value component.

undesired noise. [13] states how, in the HSV domain, Hue had a tendency to contain greenish shadows from the original leaf pictures. Saturation and Value however, had a tendency to be clean. So we also used those two color components for leaf segmentation. Figure 3 shows the noise present in the Hue channel, but also shows how Saturation and Value are cleaner. This was useful for posterior segmentation using Expectation-Maximization (EM). We used OpenCV [24] to convert the original images into the HSV domain. Then, by using NumPy [25] , we extracted the Saturation and Value components, which were fed to the Expectation-Maximization (EM) algorithm.

*2) Expectation-Maximization (EM):* Once images were converted to HSV and the desired channels were extracted, we applied EM to the color domain in order to cluster the pixels into one of 2 possible groups: leaf and non-leaf groups [13]. Figure 4 shows several samples of the final segmentation after applying EM. As shown, EM segments the image into the leaf and non-leaf pixel groups by assigning a 1 to the leaf pixels and a 0 to the non-leaf pixels. This method also works well on both simple and compound leaves. It is important to highlight that we did not assign weights to each cluster manually as the work done by [13], because we wanted to leave the process as automatic as possible. In their work, they improve the segmentation of certain types of leaves, especially skinny ones, by manually assigning different weights to each cluster. Weights play a fundamental role into the segmentation process as reported in [26].

*a) Training:* Algorithm 1 describes the process to train the EM algorithm. We used OpenCV's implementation of EM. First we stacked all the pixels of the image matrix into a single vector. Then we trained the model using a diagonal matrix as a co-variance matrix, and we assigned two clusters to it, which internally were translated into two Gaussian Distributions, one for the leaf cluster and one for the non-leaf cluster. Once trained, we returned the EM object.

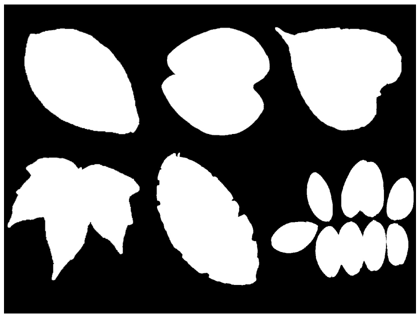

Figure 4: Segmented Samples
After applying EM to different Costa Rican species.

**Algorithm 1** EM Training

```
$stackedPixels \leftarrow \emptyset$
**for all** $pixelRow$ in image **do**
    **for all** $pixel$ in pixelRow **do**
        $stackedPixels \leftarrow stackedPixels \cup pixel$
    **end for**
**end for**
$EM \leftarrow OpenCV.EM(nClusters = 2, covMatType = OpenCV.DIAGONAL)$
$EM.train(stackedPixels)$
**return** $EM$
```

*b) Pixel Prediction:* Algorithm 2 explains how the owning cluster of a single pixel of the image was predicted. Once the EM object was trained, the OpenCV's implementation allowed to compute the probabilities of the pixel belonging to each cluster. However, for more efficiency, we created a dictionary containing each unique $(Saturation, Value)$ pair as key, and the cluster as value. If the key was not found in the dictionary, we then proceeded to predict the probabilities for each cluster, added the key and cluster to the dictionary, and returned the associated cluster with the biggest probability.

**Algorithm 2** EM Pixel Prediction

```
$key \leftarrow hash(pixel[S], pixel[V])$
**if** $hash$ in pixelDictionary **then**
    **return** $pixelDictionary[key]$
**end if**
$probabilities \leftarrow EM.predict(pixel[S], pixel[V])$
$pixelDict[key] = probabilities[0] > probabilities[1]$
**return** $pixelDict[key]$
```

### C. Image Enhancements/Post-Processing

After segmentation of the leaf using EM, some extra work was needed to clean up several false positives areas. We followed the process of LeafSnap [13]. First of all, each image was clipped to the internal leaf size provided by the segmentation. Then the image was resized to a common leaf area, followed by a heuristic applied to delete undesired objects. Finally, the stem was deleted since it added noise to the model of curvature (not that much to the texture model).

*1) Clipping:* Before extracting features, a clipping phase was needed in order to resize the region where the leaf was present to a common size. The clipping algorithm was trivial to implement once the contours were calculated using

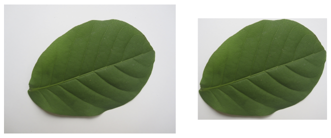

Figure 5: Clipping of a Coccoloba floribunda sample
The left image is the original leaf image, and the right one is clipped to the leaf size.

OpenCV. As shown in Algorithm 3, the minimum and maximum coordinates were calculated for all contour $x$ and $y$ components, followed by a cut of the leaf image matrix to those resulting minimum and maximum coordinates. The $\epsilon$ was used to allow posterior algorithms ignore false positives regions that intersect the border. The results of the Clipping phase can be seen in Figure 5.

**Algorithm 3** Clipping Leaf Portion of the Image

```
$xmin \leftarrow min(contours.xs) - \epsilon$
$ymin \leftarrow min(contours.ys) - \epsilon$
$xmax \leftarrow max(contours.xs) + \epsilon$
$ymax \leftarrow max(contours.ys) + \epsilon$
$clipped \leftarrow image[xmin : xmax, ymin : ymax]$
```

*2) Resizing Leaf Area:* Once the leaf area had been clipped, a resize was applied in order to standardize the leaf areas inside all images. If not, the model of curvature would be affected negatively since the amount of contour pixels varied significantly [13]. Our implementation of the resize was applied to the whole clipped image. Images may end up having different sizes, but the internal leaf areas were the same or almost the same. Algorithm 4 shows how a new width and height were obtained by calculating the ratio between the current leaf area, the desired new leaf area, and the current height and width of the image. Finally, OpenCV was used to resize the clipped image to a constant leaf size of $100,000$ pixels. This approach means that the absolute measures of leafs are lost.

**Algorithm 4** Common Leaf Area Resize

```
$newLeafArea \leftarrow 100000$
$imgArea \leftarrow height \times weight$
$newImgArea \leftarrow (imgArea \times newLeafArea)/leafArea$
$wGrowth \leftarrow weight/height + weight$
$hGrowth \leftarrow height/height + weight$
$a \leftarrow wGrowth \times hGrowth$
$x \leftarrow abs(\sqrt{4 \times a \times newImgArea}/(2 \times a))$
$newWidth = wGrowth \times x$
$newHeight = hGrowth \times x$
return $OpenCV.resize(image, newWidth, newHeight)$
```

*3) Deleting Undesired Objects:* Even when uniform background images were used, initial segmentation turned out not to be enough when the image contained undesired objects, such as dust, shadows, among others. [13] attempted to delete these noisy objects by using the same heuristic we implemented as shown in Algorithm 5. By using Scikit-learn [27] we calculated the connected components of the segmented image. We deleted the ”small” components by area (in pixels). Small components were normally dust, small bugs or pieces of leaves, among other things. Once all small components were deleted, if the remaining was only one then we took that to be the leaf. If more than one component remained, then we calculated for each remaining component how many pixels had intersections with the image margin. We then deleted the component with the biggest number of intersections. The thinking behind this is to get rid of components that were not centered on the image, which tend to be non-leaf objects. Finally, the component with the biggest area from the remaining components was taken as the leaf.

**Algorithm 5** Deleting Undesired Objects Heuristic

```
$n, components \leftarrow connectedComponents(segmentedImage)$
$components \leftarrow deleteSmallComponents(components, kMinimumArea)$
if $size(components) == 1$ then
    return $components[0]$
end if
$inters \leftarrow empty$
$areas \leftarrow empty$
for all $component$ in $components$ do
    $inters \leftarrow inters \cup getImageMarginIntersections(component)$
    $areas \leftarrow areas \cup getComponentArea(component)$
end for
$noisyObject \leftarrow max(inters)$
return $max(areas - noisyObject)$
```

*4) Deleting the stem:* We followed the approach for stem deletion described in [13]. If the stem was left intact, it would add noise to the model of curvature, given all the possible sizes the stem may take. Algorithm 7 shows the procedure. First, a Top Hat transformation was applied to the segmented image in order to leave only possible stem regions, as shown in Figure 6. Then all connected components were calculated from the Top Hat transformed image, and also their quantity. Then we looped over all the components, deleting every single one from the original segmentation and recalculating the new number of connected components. If the original number of recalculated connected components did not change upon deletion, that meant the current component was a good stem candidate (heuristically, a stem does not affect how many original connected components there are). Once all stem candidates were calculated, the one with the biggest area and largest aspect ratio was chosen to be the stem, as described in Algorithm 6.

**Algorithm 6** Calculate Aspect Ratio Combined with Area

```
$width, heigth \leftarrow calculateRectangleAround(component)$
$area \leftarrow calculateArea(component)$
return $width/heigth * area$
```

**Algorithm 7** Deleting the Stem

```
$candidates \leftarrow empty$
$candidatesRatios \leftarrow empty$
$possibleStemsImage \leftarrow topHatTransformation(segmentedImage)$
$n, components \leftarrow connectedComponents(possibleStemsImage)$
for all $component$ in $components$ do
    $tempSegmentation \leftarrow delete(component, segmentedImage)$
    $currentN \leftarrow connectedComponents(tempSegmentation)$
    if $currentN = n$ then
        $candidates \leftarrow candidates \cup component$
        $candidatesRatios \leftarrow candidatesRatios \cup calculateAspectRatio(component)$
    end if
end for
$bestCandidate \leftarrow candidates[max(candidatesRatios).index]$
$segmentedImage \leftarrow delete(bestCandidate, segmentedImage)$
```

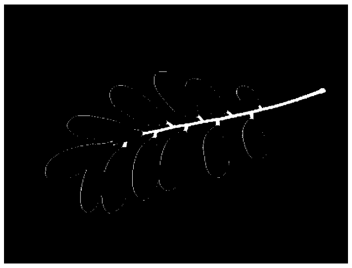

Figure 6: Top Hat Transformation applied to a segmented compound leaf image to detect the stem of the leaf.

### D. *Leaf Feature Extraction*

Feature extraction was designed and implemented considering three main design goals:

- Efficiency: algorithms should be fast enough to support future mobile apps.
- Rotation invariance: the leaf may be rotated by any angle within the image.
- Leaf Size Invariance: datasets contain different sizes of leaves and users can capture images independently of the relative size of leaves.

Two different feature sets were calculated. The first one captures information about the contour of the leaf, while the second one captures information about its texture. Section III-D1 describes how we implemented Histogram of Curvature over Scale (HCoS) [13] to extract contour information. Section III-D2 describes how we implemented Local Binary Pattern Variance (LBPV) to extract texture information. Both models generate histograms that are suitable for distance metric calculations.

*1) Extracting contour information (HCoS):* The model of curvature used by LeafSnap comprises several steps. Previously explained segmentation and post-processing resulted in a mask of leaf and non-leaf pixels. The non-leaf pixels have values of $0$, and the leaf pixels have values of $1$. First, the different contour pixels were found, then 25 different masks with disk shapes were applied on top of each contour point, providing both an area of the intersection and an arc length. Then all calculations at each scale were turned into a histogram, resulting in 25 different histograms per image, one per scale. Finally, the 25 resulting histograms were concatenated, conforming the HCoS.

*a) Contours:* On a binary image (resulted from the previous segmentation), the OpenCV implementation of contour finding worked very well, based on the original algorithm of [28] for contour finding. The algorithm generated in a vector of pairs $(x, y)$ that represent the coordinates where a contour pixel was found. A contour pixel can be defined as a pixel which is surrounded by at least another pixel with the opposite color of it. Figure 7 shows in red the contour pixels detected in the original image, calculated from the segmented mask. Notice how shadows affect the contour algorithm, since they were not segmented perfectly.

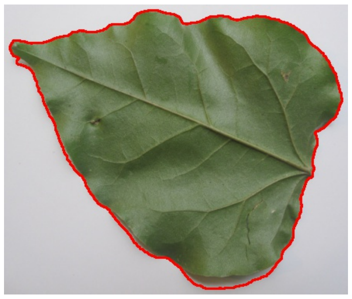

Figure 7: *Croton niveus* contours extracted using OpenCV.

$$\begin{matrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{matrix}$$

Figure 8: A filled disk of *radius*=2 pixels used to calculate area of the intersection with the leaf segmentation.

*b) Scales:* The original algorithm of [13] makes use of 25 different scales, creating one disk per scale. We implemented a discrete version of the disks making use of matrices based on [29], whose code is available in Matlab [1].

The disks used are actually matrices of 1's and 0's. They were applied as masks over specific parts of the segmented leaf image (mostly contour points). The idea was to count how many pixels intersected the segmented image and each disk mask. We created two different types of disks. The first type is filled up with 1's, as shown in Figure 8. It is used to measure the area of intersection. The second type is more like a ring, where 1's are present only in the circumference of the disk (see Figure 9). It is used to determine the arc's length of the intersection of the disk with the leaf, at a given contour point.

Once all disks were created for both area and arc length

[1]https:www.ceremade.dauphine.frp̃eyrenumerical-tourtoursshapes_4_shape_matching

$$\begin{matrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{matrix}$$

Figure 9: An unfilled disk (ring) of *radius*=2 *pixels* used to calculate arc length of the intersection with the leaf segmentation.

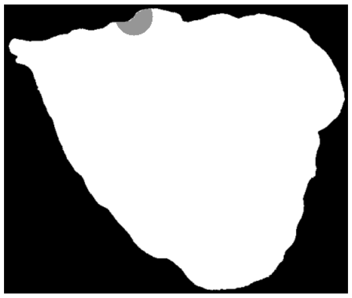

Figure 10: Area disk applied to a *Croton niveus* sample at an specific pixel of the contour, with radius=18.

versions, we applied them to each pixel of the contour vector, as shown by Algorithm 8.

**Algorithm 8** Area and Arc Length Vector Calculation

```
$arcs \leftarrow empty$
$areas \leftarrow empty$
for all $pixel$ of the contour vector do
    for all $areaMask, arcMask = 1$ to $25$ do
        center $areaMask, arcMask$ at current contour $pixel$
        $area \leftarrow count(areaMask \cap segmentation)$
        $areas \leftarrow areas \cup area$
        $arc \leftarrow count(arcMask \cap segmentation)$
        $arcs \leftarrow arcs \cup arc$
    end for
end for
```

Figure 10 shows how one specific area disk was applied to the segmented image, for an specific scale (radius=18 in this case), at a given contour pixel. The gray area shows the intersection of pixels with the leaf segmentation. This procedure was then repeated over all the pixels from the contour vector in the same way.

*c) Histograms:* Using NumPy at each scale, a histogram was created from all the values generated from all contour pixels, as described by Algorithm 8. We used histograms of 21 bins, as [13] did. This means a total of 25 different histograms were created, each with 21 bins, per image. At each scale, each histogram was normalized to unit length. Then, all histograms were concatenated together (both the 25 for area and 25 for arc length), generating what [13] describes as the Histogram of Curvature over Scale (HCoS).

*2) Extracting texture information (Local Binary Pattern Variance (LBPV)):* We aimed at improving the model of curvature by adding texture analysis. We used a Local Binary Pattern Variance (LBPV) implementation called Mahotas [30] that is invariant to rotation, multiscale, and efficient. This implementation of LBPV is based on the algorithm of [31] and makes use of NumPy libraries to represent the image and the resulting histograms. It works on gray images, so we used OpenCV to convert the Red Green Blue (RGB) images to gray scale images. The LBPV approach detects micro structures such as lines, spots, flat areas, and edges [31]. This is useful to detect patterns of the veins, areas between them, reflections, and even roughness. Figure 11 shows what two different LBPV implementations look like. The upper image shows a $radius = 2, pixels = 16$ (R2P16) implementation, and the one below shows a $radius = 1, pixel = 8$ (R1P8) pixel implementation. The different variants of the LBPV used are shown in Table I. In some cases we concatenated two histograms of different scales such as R1P8 & R2P16. It is important to note that we did not use the variant which samples 24 pixels, since it generated too large histograms. We did, however, run some tests in which we noticed the 24 pixels variation didn't add more accuracy, so we decided to ignore this method.

Table I: Variants of LBPV

| Variant | Radius | Pixels |
|---|---|---|
| R1P8 | 1 | 8 |
| R2P16 | 2 | 16 |
| R3P16 | 3 | 16 |
| R1P8 & R2P16 | 1 and 2 | 8 and 16 |
| R1P8 & R3P16 | 1 and 3 | 8 and 16 |
| R3P24 | 3 | 24 |

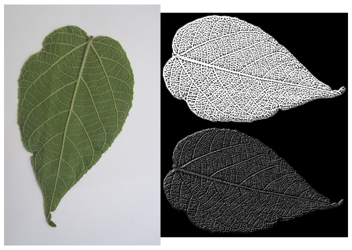

Figure 11: LBPV patterns of a *Croton draco* sample. The upper image corresponds to a $radius = 2, pixels = 16$ (R2P16) and the lower one to a $radius = 1, pixels = 8$ (R1P8) pattern.

Just like the HCoS, LBPV generates histograms that can be used for similarity search. Several histograms were generated at different radius sizes and different circumference pixel sampling, in order to validate which combinations provided the best results. The Mahotas implementation returned a histogram of the feature counts, where position $i$ corresponds the count of pixels in the leaf texture that had code $i$. Also, given that the implementation is a LBPV, non-uniform codes are not used. Thus, the bin number $i$ is the $i-th$ feature, not just the binary code $i$ [30]. Figure 12 describes at a very high level how the process of extracting the local patterns histograms works. First, the image is converted to a gray scale image. Then, for each pixel inside the segmented leaf area, we calculated the local pattern with different radius and circumference using the mahotas implementation. Finally, each pattern was assigned to a bucket in the resulting histogram.

## E. Species Classification based on Leaf Images

Once all histograms were ready and normalized, a machine learning algorithm was used to classify unseen images into species. We implemented the same classification scheme used by LeafSnap. The following paragraphs describe how k Nearest Neightbors (kNN) was implemented.

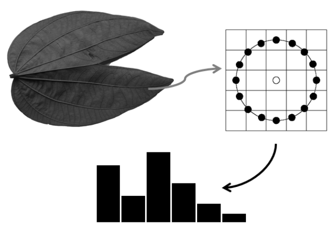

Figure 12: Process of extracting LBPV. Each pixel has a number assigned to it corresponding to a pattern, and the histogram was created using all those numbers from the segmented leaf pixels.

Scikit-learn's kNN implementation was used for leaf species classification. This process was fed with previously generated histograms from both the model of curvature using HCoS and the texture model using LBPV. Additional code was created to take into consideration only the first matching $k$ species, not the first $k$ images, as shown by Algorithm 9. The difference resides in taking into account only the best matching image per species, until completing the first $k$ species [13].

**Algorithm 9** k Species Ranking

```
$neighborImages, distances \leftarrow knnSearch(histogram, k)$
$resultSpecies \leftarrow empty$
while each $neighborImage$ and $k > 0$ do
    if not $neighborImage.species$ in $resultSpecies$ then
        $resultSpecies \leftarrow resultSpecies \cup neighborImage.species$
        $k \leftarrow k - 1$
    end if
end while
```

We used $1 <= k <= 10$ in order to measure how different algorithms behaved as the value of $k$ increased.

## *F. Distance Metric - Histogram Intersection*

We tested the basic Euclidean distance to measure similarity between histograms, however the results were not encouraging. We implemented the histogram intersection shown on Equation 1, where $I(x, y)$ is the histogram intersection between a histograms $x$ and $y$ of same size, $n$ is the number of bins, and $x_i$ and $y_i$ are each a bin in histograms $x$ and $y$, respectively. This distance metric is also normalized to unit length.

$$I(x, y) = \sum_{i=1}^{n} x_i - \sum_{i=1}^{n} min(x_i, y_i) \tag{1}$$

## *G. Accuracy*

Let $E$ be an identification experiment that consists of a model $M$, a set $S$ that contains $n$ images of leaves of $n$ (not necessarily different) unknown tree species to be identified, and an integer value $k$, $k \geq 1$. We define $hit(M, k, x)$ as a boolean function that indicates if model $M$ generates a ranking in which one of the top $k$ candidate species is a correct identification of sample $x$. Equation 2 formally defines $Accuracy(M, S, k)$.

$$Accuracy(M, S, k) = \sum_{x \in S} \frac{hit(M, k, x)}{n} \tag{2}$$

Table II: Models used in the experiments including curvature, variants of texture model, and combination of both.

| Model Name | Description | Type |
|---|---|---|
| HCoS | 25 scales, 21 bins per scale | Curvature |
| R1P8 | $radius = 1, pixels = 8$ | Texture |
| R2P16 | $radius = 2, pixels = 16$ | Texture |
| R3P16 | $radius = 3, pixels = 16$ | Texture |
| R1P8 & R2P16 | $radius = 1, pixels = 8$ & $radius = 2, pixels = 16$ | Texture |
| R1P8 & R3P16 | $radius = 1, pixels = 8$ & $radius = 3, pixels = 16$ | Texture |
| HCoS & R1P8 & R3P16 | Assigned a factor to curvature and texture. Factors summed 1, increasing by 0.10 | Curvature and Texture |

## IV. Experiments

Several model variations were used in the experiments (see Table II).

1) Our implementation of LeafSnap's model of curvature HCoS.
2) Several scales of the texture model based on LBPV.
3) The combination of HCoS and the best LBPV variant, which according to our tests was R1P8 & R3P16. This combination was further disaggregated by assigning different weights to HCoS and the texture model.

*One Versus All:* One approach to test a model is to partition a dataset into two datasets: one for training and one for testing. Another approach is to use One versus All, that is, each image in a dataset with $n$ elements is considered a test image and the remaining $n - 1$ images the training subset. We used both approaches as explained at the end of this section.

*Combining Curvature and Texture:* When combining two different models, we faced the issue of having different scales in the resulting ranking of each model. This was resolved by normalizing the rankings to unit length.

After normalizing the rankings (one per combined algorithm), we assigned a factor to each combined model in order to rank the predicted species into a single ranking. This factor sums 1 in total. However we varied the factor associated with each model to see the behavior across different combinations. We used factors of (0.10, 0.90), (0.20, 0.80), (0.30, 0.70), (0.40, 0.60), (0.50, 0.50), (0.60, 0.40), (0.70, 0.30), (0.80, 0.20), (0.90, 0.10). For example, $(0.50, 0.50)$ means we gave the same level of importance to each model on that combination. Algorithm 10 describes how the merge between two methods was achieved.

**Algorithm 10** Combining Two Rankings

```
$combinedRanking \leftarrow \emptyset$
$FACTORS \leftarrow \{0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90\}$
for all $factor$ in $FACTORS$ do
    $results \leftarrow empty$
    for all $species$ in $allSpecies$ do
        $distance1 \leftarrow resultsAlgorithm1[species]$
        $distance2 \leftarrow resultsAlgorithm2[species]$
        $results[species] \leftarrow (distance1 * factor) + (distance2 * (1 - factor))$
    end for
    $combinedRanking[factor] \leftarrow TakeBestKDistances(results)$
end for
```

*Texture and Curvature Model Experiments*

We ran all models $M$ described in Table II, with $1 \leq k \leq 10$, and the following data sets: Costa Rica clean subset (One versus All, $n = 1468$), Costa Rica noisy subset (One versus All, $n = 2345$), and Costa Rica complete data set (training set with all 1468 clean images and testing set with all 2345 noisy images). In each experiment, $Accuracy(M, S, k)$ was calculated for the corresponding dataset $S$. In addition, for model HCoS & R1P8 & R3P16, Algorithm 10 was used to comprehensively consider different weight combinations for HCoS and the texture model. Table III summarizes the results obtained.

*Processing Times*

To understand the duration of the recognition process, we measured the recognition time for all images from both Costa Rican noisy and clean subsets, as if a back-end received images from a mobile app. The measured time includes image loading, segmentation, stem deletion, normalization, curvature calculations, texture calculations, and similarity search. It does not include network related times. We used a MacBook Pro with an Intel Core i7, 2.8 GHz, and 8 GBs on RAM.

Table III: Accuracy obtained when combining curvature and texture over the clean subset, the noisy subset, and the complete Costa Rican dataset

| | | Clean | | |
|---|---|---|---|---|
| | | HCoS=a, R1P8 & R3P16=b | | |
| k | HCoS | a=0.1 b=0.9 | a=0.5 b=0.5 | a=0.9 b=0.1 |
| 1 | 0.311 | 0.567 | 0.563 | 0.386 |
| 2 | 0.446 | 0.702 | 0.702 | 0.520 |
| 3 | 0.535 | 0.766 | 0.785 | 0.610 |
| 4 | 0.587 | 0.816 | 0.822 | 0.668 |
| 5 | 0.631 | 0.857 | 0.854 | 0.706 |
| 6 | 0.674 | 0.875 | 0.881 | 0.748 |
| 7 | 0.710 | 0.890 | 0.909 | 0.779 |
| 8 | 0.740 | 0.903 | 0.924 | 0.812 |
| 9 | 0.768 | 0.918 | 0.937 | 0.832 |
| 10 | 0.790 | 0.931 | 0.945 | 0.845 |
| | | Noisy | | |
| | | HCoS=a, R1P8 & R3P16=b | | |
| k | HCoS | a=0.1 b=0.9 | a=0.5 b=0.5 | a=0.9 b=0.1 |
| 1 | 0.151 | 0.519 | 0.320 | 0.177 |
| 2 | 0.225 | 0.638 | 0.435 | 0.257 |
| 3 | 0.277 | 0.701 | 0.515 | 0.311 |
| 4 | 0.325 | 0.750 | 0.574 | 0.364 |
| 5 | 0.364 | 0.783 | 0.616 | 0.408 |
| 6 | 0.399 | 0.810 | 0.660 | 0.455 |
| 7 | 0.435 | 0.830 | 0.692 | 0.484 |
| 8 | 0.470 | 0.844 | 0.721 | 0.516 |
| 9 | 0.496 | 0.858 | 0.744 | 0.546 |
| 10 | 0.521 | 0.872 | 0.771 | 0.574 |
| | | All | | |
| | | HCoS=a, R1P8 & R3P16=b | | |
| k | HCoS | a=0.1 b=0.9 | a=0.5 b=0.5 | a=0.9 b=0.1 |
| 1 | 0.070 | 0.145 | 0.120 | 0.084 |
| 2 | 0.119 | 0.209 | 0.178 | 0.133 |
| 3 | 0.148 | 0.252 | 0.216 | 0.165 |
| 4 | 0.176 | 0.295 | 0.251 | 0.201 |
| 5 | 0.204 | 0.326 | 0.277 | 0.224 |
| 6 | 0.228 | 0.350 | 0.304 | 0.249 |
| 7 | 0.253 | 0.377 | 0.328 | 0.277 |
| 8 | 0.273 | 0.400 | 0.353 | 0.299 |
| 9 | 0.295 | 0.417 | 0.371 | 0.320 |
| 10 | 0.318 | 0.439 | 0.393 | 0.336 |

## V. Results

### A. Analysis of Results

*a) Clean Subset:* As shown in Table III, the best results were obtained when $k = 10$ and the model is 0.5 HCoS and 0.5 R1P8 & R3P16. The resulting accuracy is 0.945, in contrast with the accuracy of HCoS which is 0.79. Notice however that 0.5 HCoS and 0.5 R1P8 & R3P16 is also the best for all values of $6 <= k <= 10$. For $1 <= k <= 5$, 0.5 HCoS and 0.5 R1P8 & R3P16 and 0.1 HCoS and 0.9 R1P8 & R3P16 have very similar levels of accuracy. Figure 13 more clearly depicts these comparisons.

*b) Noisy Subset:* Figure 14 clearly shows that 0.1 HCoS and 0.9 R1P8 & R3P16 has the best accuracy for all values of $k$. In addition, the level of accuracy improvement with respect to HCoS is considerably larger, ranging from 35.2% when $k = 10$ to 42.5% when $k = 4$ as shown in Table V.

*c) Complete Dataset:* As Figure 15 shows, the level of accuracy is considerably lower for all models, as compared to the previous two experiments. Even the best model achieves levels of accuracy in a poor $[14.5\%, 43.9\%]$ range.

*Discussion:* These experiments show how, in general, the combination of HCoS and LBPV consistently increases the accuracy of HCoS alone. Accuracy declines as the combination factor assigned to curvature reaches 1. Overall, the best combination seems to be 0.1 HCoS and 0.9 LBPV. It is also important to notice how the accuracy is sensitive to the quality of the dataset. The clean subset has a tendency to improve the recognition accuracy, in contrast with the noisy subset. This reflects the importance of good pre-processing and good segmentation. Shadows, dust, and other artifacts affect the final accuracy results.

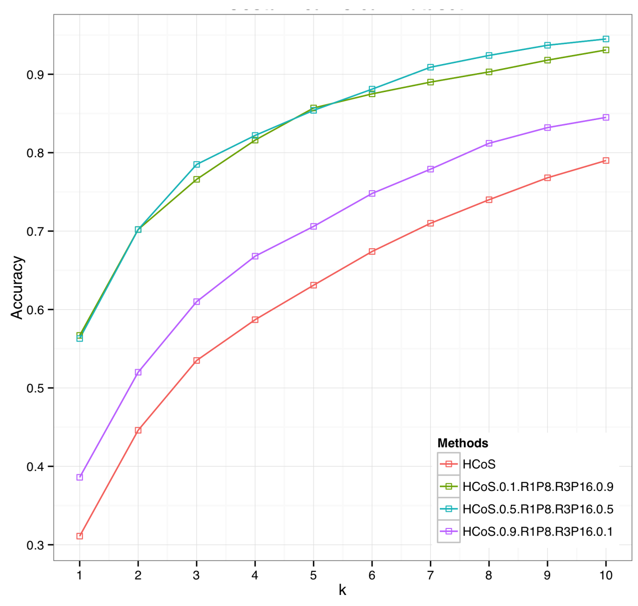

Figure 13: Accuracy of HCoS Vs Combined Methods against Costa Rican clean dataset.

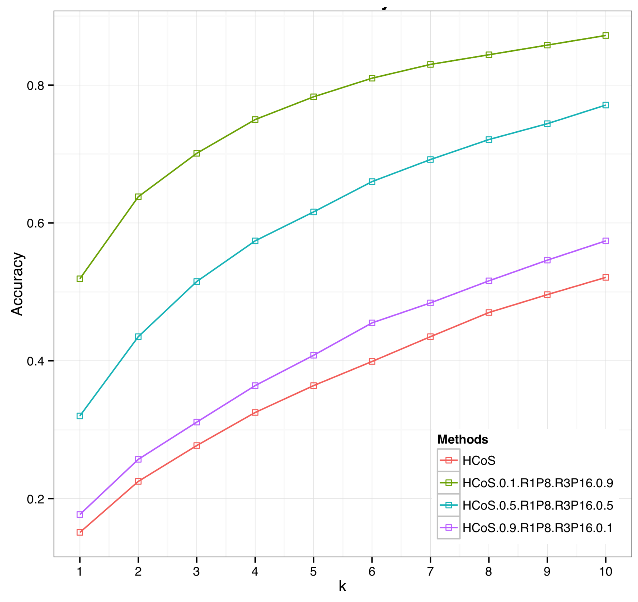

Figure 14: Accuracy of HCoS Vs Combined Methods against Costa Rican noisy dataset.

## B. Measuring Significance of the Accuracy Increase

As shown in the previous section there is an increase in accuracy when texture is added to our implementation HCoS. This, however, may not be statistically significant. We proceeded then to apply a Statistical Proportion Test for Two Samples. Our null hypothesis $H0$ is that the accuracy of the implementation of HCoS equals the ones obtained by combining curvature and texture. In contrast, our alternative hypothesis $H1$ is that the accuracy of the implementation of HCoS is less than the combinations.

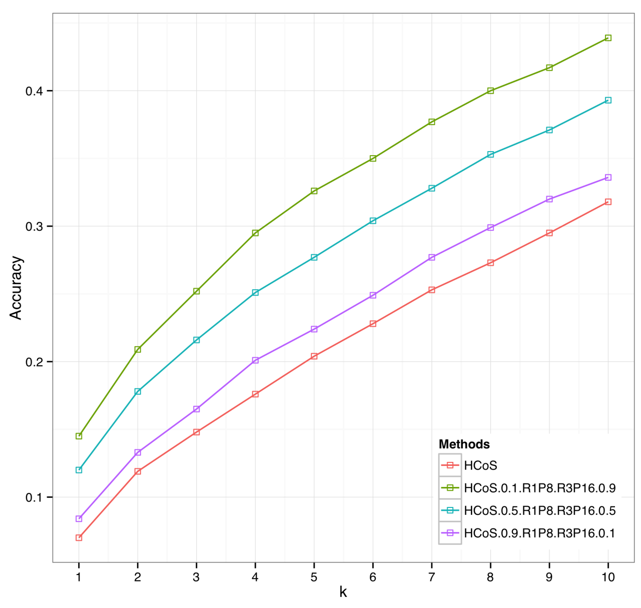

Figure 15: Accuracy of HCoS Vs Combined Methods against the complete Costa Rican dataset. Clean subset used for training and noisy subset for testing.

*d) Proportion Tests on the Clean Subset:* Table IV shows the results obtained for all the proportion tests for the clean subset. Most combinations of HCoS and R1P8 & R3P16 for $1 <= k <= 10$ resulted in very low p-Values, which reject $H0$. However a few accuracy increases from $0.9$ HCoS and $0.1$ R1P8 & R3P16 did fail the test. This means that, as the weight increases for HCoS, it starts getting non-significant accuracy increases, which makes sense since it is almost equal to HCoS alone.

*e) Proportion Tests on the Noisy Subset:* Table V shows the results obtained for all the proportion tests for the noisy subset. All combinations of HCoS and R1P8 & R3P16 resulted in very low p-Values, which reject $H0$.

*f) Proportion Tests on the Complete Dataset:* Table VI shows the results obtained for all the proportion tests on the complete dataset of leaf images from Costa Rica. Almost every single test rejected $H0$. For $k = 1$ the results are not significant.

In all Proportion Tests, by adding texture with a bigger factor the model improves significantly the accuracy. As the factor assigned to texture declines, the improvement becomes statistically insignificant.

## C. Processing Time

As shown in Figure 16, times range from $2.76$ to $12.81$ seconds. However, the median of the elapsed time is $5.70$ seconds for the clean subset and $5.66$ seconds for the noisy subset. These are suitable times even for mobile applications that use the developed back-end.

Table IV: Proportion Test results over the Costa Rican Clean Subset

| Costa Rica Clean Subset<br>Confidence Level=0.95<br>Sample Size=1468<br>H0: HCoS=HCoS & R1P8 & R3P16<br>H1: HCoS<HCoS & R1P8 & R3P16 | | | | | |
|---|---|---|---|---|---|
| k | HCoS | HCoS=0.1, R1P8 & R3P16=0.9 | p-Value | Reject H0? | Accuracy Improvement |
| 1 | 0.311 | 0.567 | 5.65023E-21 | YES | 0.255 |
| 2 | 0.446 | 0.702 | 4.99997E-19 | YES | 0.255 |
| 3 | 0.535 | 0.766 | 2.00608E-15 | YES | 0.231 |
| 4 | 0.587 | 0.816 | 1.21109E-19 | YES | 0.230 |
| 5 | 0.631 | 0.857 | 2.81689E-21 | YES | 0.225 |
| 6 | 0.674 | 0.875 | 9.64321E-21 | YES | 0.201 |
| 7 | 0.710 | 0.890 | 2.70704E-18 | YES | 0.180 |
| 8 | 0.740 | 0.903 | 4.32615E-17 | YES | 0.163 |
| 9 | 0.768 | 0.918 | 6.49779E-16 | YES | 0.151 |
| 10 | 0.790 | 0.931 | 1.14726E-14 | YES | 0.141 |
| k | HCoS | HCoS=0.5, R1P8 & R3P16=0.5 | p-Value | Reject H0? | Accuracy Improvement |
| 1 | 0.311 | 0.563 | 4.32788E-06 | YES | 0.251 |
| 2 | 0.446 | 0.702 | 6.56883E-09 | YES | 0.256 |
| 3 | 0.535 | 0.785 | 1.09341E-11 | YES | 0.251 |
| 4 | 0.587 | 0.822 | 5.88439E-16 | YES | 0.235 |
| 5 | 0.631 | 0.854 | 2.42945E-19 | YES | 0.223 |
| 6 | 0.674 | 0.881 | 4.19306E-23 | YES | 0.207 |
| 7 | 0.710 | 0.909 | 1.18899E-21 | YES | 0.198 |
| 8 | 0.740 | 0.924 | 1.62723E-20 | YES | 0.185 |
| 9 | 0.768 | 0.937 | 7.26426E-20 | YES | 0.170 |
| 10 | 0.790 | 0.945 | 7.84393E-20 | YES | 0.155 |
| k | HCoS | HCoS=0.9, R1P8 & R3P16=0.1 | p-Value | Reject H0? | Accuracy Improvement |
| 1 | 0.311 | 0.386 | 0.976355356 | NO | 0.075 |
| 2 | 0.446 | 0.520 | 0.823819993 | NO | 0.074 |
| 3 | 0.535 | 0.610 | 0.840833982 | NO | 0.075 |
| 4 | 0.587 | 0.668 | 0.26158887 | NO | 0.082 |
| 5 | 0.631 | 0.706 | 0.0201783 | YES | 0.074 |
| 6 | 0.674 | 0.748 | 0.017077481 | YES | 0.074 |
| 7 | 0.710 | 0.779 | 0.002586312 | YES | 0.069 |
| 8 | 0.740 | 0.812 | 0.000201496 | YES | 0.072 |
| 9 | 0.768 | 0.832 | 5.92221E-05 | YES | 0.065 |
| 10 | 0.790 | 0.845 | 3.63353E-06 | YES | 0.055 |

Table V: Proportion Test results over the Costa Rican Noisy Subset

| Costa Rica Noisy Subset<br>Confidence Level=0.95<br>Sample Size=2345<br>H0: HCoS=HCoS & R1P8 & R3P16<br>H1: HCoS<HCoS & R1P8 & R3P16 | | | | | |
|---|---|---|---|---|---|
| k | HCoS | HCoS=0.1, R1P8 & R3P16=0.9 | p-Value | Reject H0? | Accuracy Improvement |
| 1 | 0.151 | 0.519 | 1.7283E-129 | YES | 0.368 |
| 2 | 0.225 | 0.638 | 7.7632E-157 | YES | 0.413 |
| 3 | 0.277 | 0.701 | 1.3354E-165 | YES | 0.424 |
| 4 | 0.325 | 0.750 | 6.4313E-182 | YES | 0.425 |
| 5 | 0.364 | 0.783 | 5.3369E-191 | YES | 0.420 |
| 6 | 0.399 | 0.810 | 2.8814E-194 | YES | 0.411 |
| 7 | 0.435 | 0.830 | 5.1936E-187 | YES | 0.396 |
| 8 | 0.470 | 0.844 | 1.3596E-178 | YES | 0.374 |
| 9 | 0.496 | 0.858 | 2.6133E-177 | YES | 0.362 |
| 10 | 0.521 | 0.872 | 5.9603E-173 | YES | 0.352 |
| k | HCoS | HCoS=0.5, R1P8 & R3P16=0.5 | p-Value | Reject H0? | Accuracy Improvement |
| 1 | 0.151 | 0.320 | 1.2405E-75 | YES | 0.169 |
| 2 | 0.225 | 0.435 | 2.2453E-116 | YES | 0.209 |
| 3 | 0.277 | 0.515 | 1.1237E-149 | YES | 0.238 |
| 4 | 0.325 | 0.574 | 7.6123E-168 | YES | 0.250 |
| 5 | 0.364 | 0.616 | 5.4143E-184 | YES | 0.252 |
| 6 | 0.399 | 0.660 | 1.5749E-202 | YES | 0.261 |
| 7 | 0.435 | 0.692 | 5.0885E-199 | YES | 0.258 |
| 8 | 0.470 | 0.721 | 8.0747E-191 | YES | 0.250 |
| 9 | 0.496 | 0.744 | 1.7097E-191 | YES | 0.248 |
| 10 | 0.521 | 0.771 | 2.0950E-191 | YES | 0.250 |
| k | HCoS | HCoS=0.9, R1P8 & R3P16=0.1 | p-Value | Reject H0? | Accuracy Improvement |
| 1 | 0.151 | 0.177 | 2.4494E-26 | YES | 0.025 |
| 2 | 0.225 | 0.257 | 1.9667E-50 | YES | 0.032 |
| 3 | 0.277 | 0.311 | 1.9949E-63 | YES | 0.035 |
| 4 | 0.325 | 0.364 | 4.4262E-79 | YES | 0.040 |
| 5 | 0.364 | 0.408 | 1.5164E-96 | YES | 0.044 |
| 6 | 0.399 | 0.455 | 6.3080E-102 | YES | 0.055 |
| 7 | 0.435 | 0.484 | 8.9291E-112 | YES | 0.050 |
| 8 | 0.470 | 0.516 | 6.4232E-118 | YES | 0.046 |
| 9 | 0.496 | 0.546 | 4.2650E-125 | YES | 0.049 |
| 10 | 0.521 | 0.574 | 9.9417E-134 | YES | 0.054 |

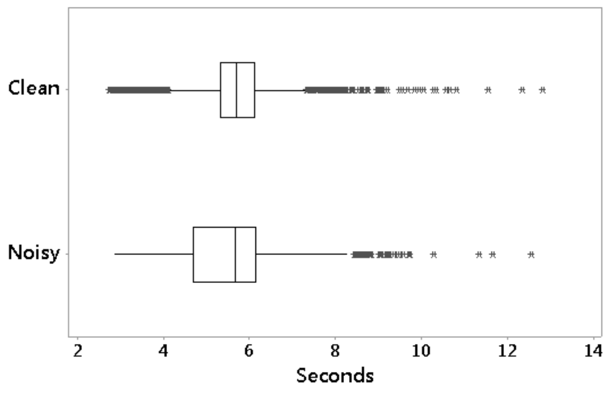

Figure 16: Box Plot of leaf image recognition times simulating a mobile app back-end, for Costa Rican noisy and clean subsets.

## VI. Conclusions

The addition of texture increases significantly the accuracy of our implementation of the HCoS. When comparing HCoS versus the combination of $0.1$ HCoS and $0.9$ R1P8 & R3P16, for the Costa Rican clean subset, the improvement ranges from $14.1\%$ to $25.5\%$, depending on the value of $k$. Similarly, with the noisy subset, the improvement ranges from $35.5\%$ to $42.5\%$. These improvements were proved to be statistically significant in our experiments.

The complete dataset experiments demonstrated that poor accuracy levels are achieved when noisy images are classified against clean images. We speculate that this is due to the many enhancements that leaf images underwent before being added to the clean dataset. First, leaves were pressed for 24 hours in order to flatten them and thus minimize the presence of shadows. Secondly, Photoshop was used to manually remove artifacts. Finally, image enhancement algorithms (e.g., stem removal) were applied. This result has important implications if a mobile application is developed, given that users will take noisy pictures. As a result we are left with two alternatives. The first one is to use a noisy dataset to train the classifier. Alternatively a clean dataset could be used but user images would need to undergo further automated image enhancements comparable to those performed manually with Photoshop.

## VII. Future Work

A natural next step in this research is to develop a mobile app that uses the georeference of photographs of leaves as an additional criterion to classify species. Most modern mobile phones already include excellent cameras and provide the option of automatically georeferencing any picture taken with these cameras. In addition to the reference image dataset such as the one developed for this research, maps of potential distribution of species of Costa Rican trees would be needed. *Atta*, a comprehensive and fully georeferenced database of thousands of species of organisms from Costa Rica developed

Table VI: Proportion Test results over the Costa Rican Complete Dataset

| Costa Rica All Dataset<br>Confidence Level=0.95<br>Sample Size=2345<br>H0: HCoS=HCoS & R1P8 & R3P16<br>H1: HCoS<HCoS & R1P8 & R3P16 | | | | | |
|---|---|---|---|---|---|
| k | HCoS | HCoS=0.1, R1P8 & R3P16=0.9 | p-Value | Reject H0? | Accuracy Improvement |
| 1 | 0.070 | 0.145 | 4.3210E-01 | NO | 0.075 |
| 2 | 0.119 | 0.209 | 6.2947E-06 | YES | 0.090 |
| 3 | 0.148 | 0.252 | 1.8102E-10 | YES | 0.105 |
| 4 | 0.176 | 0.295 | 3.2827E-12 | YES | 0.119 |
| 5 | 0.204 | 0.326 | 1.6580E-12 | YES | 0.122 |
| 6 | 0.228 | 0.350 | 6.0361E-13 | YES | 0.122 |
| 7 | 0.253 | 0.377 | 8.0774E-12 | YES | 0.124 |
| 8 | 0.273 | 0.400 | 4.1983E-11 | YES | 0.126 |
| 9 | 0.295 | 0.417 | 5.8190E-11 | YES | 0.122 |
| 10 | 0.318 | 0.439 | 1.0927E-10 | YES | 0.121 |
| k | HCoS | HCoS=0.5, R1P8 & R3P16=0.5 | p-Value | Reject H0? | Accuracy Improvement |
| 1 | 0.070 | 0.120 | 8.2576E-01 | NO | 0.050 |
| 2 | 0.119 | 0.178 | 6.0228E-03 | YES | 0.059 |
| 3 | 0.148 | 0.216 | 1.3785E-05 | YES | 0.069 |
| 4 | 0.176 | 0.251 | 4.9141E-09 | YES | 0.075 |
| 5 | 0.204 | 0.277 | 2.9011E-10 | YES | 0.072 |
| 6 | 0.228 | 0.304 | 1.0408E-11 | YES | 0.076 |
| 7 | 0.253 | 0.328 | 9.4610E-11 | YES | 0.075 |
| 8 | 0.273 | 0.353 | 8.2167E-12 | YES | 0.080 |
| 9 | 0.295 | 0.371 | 2.6311E-11 | YES | 0.076 |
| 10 | 0.318 | 0.393 | 1.6020E-10 | YES | 0.075 |
| k | HCoS | HCoS=0.9, R1P8 & R3P16=0.1 | p-Value | Reject H0? | Accuracy Improvement |
| 1 | 0.070 | 0.084 | 6.9915E-01 | NO | 0.014 |
| 2 | 0.119 | 0.133 | 4.7461E-03 | YES | 0.014 |
| 3 | 0.148 | 0.165 | 3.6781E-04 | YES | 0.018 |
| 4 | 0.176 | 0.201 | 7.3870E-06 | YES | 0.025 |
| 5 | 0.204 | 0.224 | 4.0212E-06 | YES | 0.020 |
| 6 | 0.228 | 0.249 | 7.8066E-07 | YES | 0.021 |
| 7 | 0.253 | 0.277 | 2.8185E-06 | YES | 0.024 |
| 8 | 0.273 | 0.299 | 2.0626E-07 | YES | 0.026 |
| 9 | 0.295 | 0.320 | 1.0903E-07 | YES | 0.025 |
| 10 | 0.318 | 0.336 | 1.6458E-06 | YES | 0.017 |

by the National Biodiversity Institute (INBio) [2] and GBIF's database [3] are excellent foundations to generate these potential distribution maps of species. In addition to curvature, texture, and georeferencing as discriminating factors, morphological measures of leaves are also frequently used by specialists to identify plant species. Some of these measures are: aspect ratio, which is the ratio of horizontal width to vertical length; form coefficient, which is a numerical value that grades the leaf shape as between circular (shortest perimeter for a given area) and filliform (longest perimeter for a given area); and blade and petiole length. Algorithms to calculate these measures have already been developed (e.g., WinFOLIA). However, they have not been integrated in computer vision systems for automatic identification of plant species.

A crowd sourcing approach could be a very efficient way to increase the size of the image dataset that currently comprises 66 plant species from Costa Rica. In addition, crowdsourcing could also be used to clean noisy pictures as a citizen science project.

Finally, the individual contribution of texture features such as venation, porosity, and reflection in characterizing a plant species has not been formally established. A more elaborate analysis of the leaf texture that disaggregates it into a separate layer for each these features would help understand and quantify their individual contribution.

[2]www.inbio.ac.cr
[3]www.gbif.org

## ACKNOWLEDGEMENT

To the National Biodiversity Institute of Costa Rica (INBio) and Nelson Zamora for their help with the leaf sample recollection and expert feedback during this research.

## REFERENCES

[1] M. de Carvalho, F. Bockmann, D. Amorim, C. Brandão, M. de Vivo, J. de Figueiredo, H. Britski, M. de Pinna, N. Menezes, F. Marques, N. Papavero, E. Cancello, J. Crisci, J. McEachran, R. Schelly, J. Lundberg, A. Gill, R. Britz, Q. Wheeler, M. Stiassny, L. Parenti, L. Page, W. Wheeler, J. Faivovich, R. Vari, L. Grande, C. Humphries, R. DeSalle, M. Ebach, and G. Nelson, "Taxonomic impediment or impediment to taxonomy? a commentary on systematics and the cybertaxonomic-automation paradigm," *Evolutionary Biology*, vol. 34, no. 3-4, pp. 140–143, 2007.

[2] A. Andreopoulos and J. K. Tsotsos, "50 years of object recognition: Directions forward.," *Computer Vision and Image Understanding*, vol. 117, no. 8, pp. 827–891, 2013.

[3] L. R.D and V. S, "Plant classification using leaf recognition," in *Proceedings of the 22nd Annual Symposium of the Pattern Recognition Association of South Africa*, p. 91–95, November 2011.

[4] M. S. K. M. Z. Rashad, B.S.el-Desouky, "Plants images classification based on textural features using combined classifier," *International Journal of Computer Science and Information Technology*, vol. 3, no. 4, 2011.

[5] A. Bhardwaj, M. Kaur, and A. Kumar, "Recognition of plants by leaf image using moment invariant and texture analysis," *International Journal of Innovation and Applied Studies*, vol. 3, no. 1, pp. 237–248, 2013.

[6] S. Wu, F. Bao, E. Xu, Y.-X. Wang, Y.-F. Chang, and Q.-L. Xiang, "A leaf recognition algorithm for plant classification using probabilistic neural network," in *Signal Processing and Information Technology, 2007 IEEE International Symposium on*, pp. 11–16, Dec 2007.

[7] T. Beghin, J. Cope, P. Remagnino, and S. Barman, "Shape and texture based plant leaf classification," in *Advanced Concepts for Intelligent Vision Systems* (J. Blanc-Talon, D. Bone, W. Philips, D. Popescu, and P. Scheunders, eds.), vol. 6475 of *Lecture Notes in Computer Science*, pp. 345–353, Springer Berlin Heidelberg, 2010.

[8] D. Wijesingha and F. Marikar, "Automatic detection system for the identification of plants using herbarium specimen images," *Tropical Agricultural Research*, vol. 23, no. 1, 2012.

[9] C. H. Arun, W. R. S. Emmanuel, and D. C. Durairaj, "Texture feature extraction for identification of medicinal plants and comparison of different classifiers," *International Journal of Computer Applications*, vol. 62, January 2013.

[10] N. Aggarwal and R. K. Agrawal, "First and second order statistics features for classification of magnetic resonance brain images," *Journal of Signal and Information Processing*, vol. 3, no. 2, pp. 146–153, 2012.

[11] Y. Herdiyeni and M. Santoni, "Combination of morphological, local binary pattern variance and color moments features for indonesian medicinal plants identification," in *Advanced Computer Science and Information Systems (ICACSIS), 2012 International Conference*, pp. 255–259, Dec 2012.

[12] A. Kadir, L. E. Nugroho, A. Susanto, and P. I. Santosa, "Leaf classification using shape, color, and texture features," *International Journal of Computer Trends and Technology*, 2011.

[13] N. Kumar, P. Belhumeur, A. Biswas, D. Jacobs, W. Kress, I. Lopez, and J. Soares, "Leafsnap: A computer vision system for automatic plant species identification," in *Computer Vision – ECCV 2012* (A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, eds.), Lecture Notes in Computer Science, pp. 502–516, Springer Berlin Heidelberg, 2012.

[14] J. Clarke, S. Barman, P. Remagnino, K. Bailey, D. Kirkup, S. Mayo, and P. Wilkin, "Venation pattern analysis of leaf images," in *Proceedings of the Second International Conference on Advances in Visual Computing - Volume Part II*, ISVC'06, (Berlin, Heidelberg), pp. 427–436, Springer-Verlag, 2006.

[15] M. G. Larese, R. Namías, R. M. Craviotto, M. R. Arango, C. Gallo, and P. M. Granitto, "Automatic classification of legumes using leaf vein image features," *Pattern Recogn.*, vol. 47, pp. 158–168, Jan. 2014.

[16] K.-B. Lee, K.-W. Chung, and K.-S. Hong, "An implementation of leaf recognition system based on leaf contour and centroid for plant classification," in *Ubiquitous Information Technologies and Applications* (Y.-H. Han, D.-S. Park, W. Jia, and S.-S. Yeo, eds.), vol. 214 of *Lecture Notes in Electrical Engineering*, pp. 109–116, Springer Netherlands, 2013.

[17] K.-B. Lee and K.-S. Hong, "An implementation of leaf recognition system using leaf vein and shape," *International Journal of Bio-Science and Bio-Technology*, pp. 57–66, Apr 2013.

[18] Y. Li, Z. Chi, and D. Feng, "Leaf vein extraction using independent component analysis," in *Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on*, vol. 5, pp. 3890–3894, Oct 2006.

[19] N. Zamora, May 2014. Private Communication, National Biodiversity Institute, Costa Rica.

[20] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, vol. 39, no. 1, pp. 1–38, 1977.

[21] Y. Herdiyeni and I. Kusmana, "Fusion of local binary patterns features for tropical medicinal plants identification," in *Advanced Computer Science and Information Systems (ICACSIS), 2013 International Conference on*, pp. 353–357, Sept 2013.

[22] Q. Nguyen, T. Le, and N. Pham, "Leaf based plant identification system for android using surf features in combination with bag of words model and supervised learning," in *International Conference on Advanced Technologies for Communications (ATC)*, October 2013.

[23] J. Carranza-Rojas, "A Texture and Curvature Bimodal Leaf Recognition Model for Costa Rican Plant Species Identification," Master's thesis, Costa Rica Institute of Technology, Cartago, Costa Rica, 2014.

[24] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[25] T. E. Oliphant, *Guide to NumPy*. Provo, UT, Mar. 2006.

[26] X. Zhu, C. C. Loy, and S. Gong, "Constructing robust affinity graphs for spectral clustering," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pp. 1450–1457, June 2014.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[28] S. Suzuki and K. be, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32 – 46, 1985.

[29] S. Manay, D. Cremers, B.-W. Hong, A. Yezzi, and S. Soatto, "Integral invariants for shape matching," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 28, pp. 1602–1618, Oct 2006.

[30] L. P. Coelho, "Mahotas: Open source software for scriptable computer vision," *Journal of Open Research Software*, vol. 1, 2013.

[31] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, pp. 971–987, Jul 2002.