Leafsnap: A Computer Vision System for Automatic Plant Species Identification

Neeraj Kumar[1], Peter N. Belhumeur[2], Arijit Biswas[3], David W. Jacobs[3], W. John Kress[4], Ida C. Lopez[4], and João V.B. Soares[3]

[1] University of Washington, Seattle WA
[2] Columbia University, New York NY
[3] University of Maryland, College Park MD
[4] National Museum of Natural History, Smithsonian Institution, Washington DC

Abstract. We describe the first mobile app for identifying plant species using automatic visual recognition. The system – called Leafsnap – identifies tree species from photographs of their leaves. Key to this system are computer vision components for discarding non-leaf images, segmenting the leaf from an untextured background, extracting features representing the curvature of the leaf's contour over multiple scales, and identifying the species from a dataset of the 184 trees in the Northeastern United States. Our system obtains state-of-the-art performance on the real-world images from the new Leafsnap Dataset – the largest of its kind. Throughout the paper, we document many of the practical steps needed to produce a computer vision system such as ours, which currently has nearly a million users.

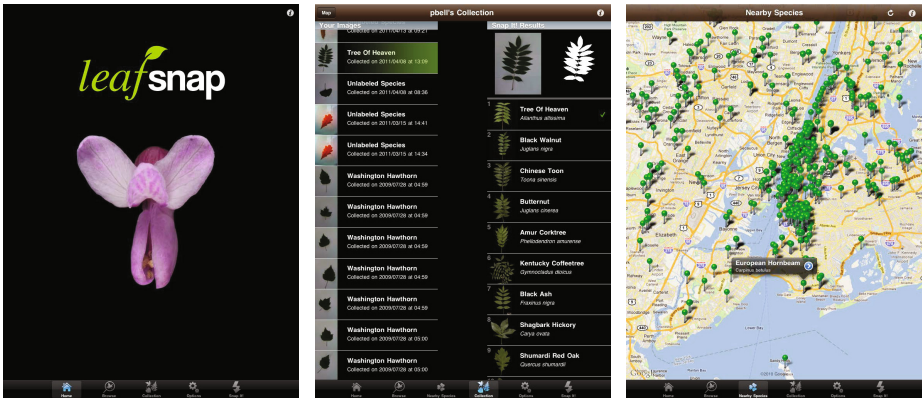
1 Introduction

In this work, we describe a visual recognition system for automatic plant species identification. The system, called Leafsnap, is a mobile app that helps users identify trees from photographs of their leaves (see Fig. 1). The current version of Leafsnap has coverage for all of the 184 tree species of the Northeastern United States. To date, nearly one million copies of Leafsnap have been installed on iPhones and iPads. It is now being used by scientists, ecologists, foresters, urban planners, amateur botanists, gardening clubs, landscape architects, citizen scientists, educators, and even school children in classes across the United States.

Leafsnap was developed to greatly speed up the manual process of plant species identification, collection, and monitoring. Without visual recognition tools such as Leafsnap, a dichotomous key (decision tree) must be manually navigated to search the many branches and seemingly endless nodes of the taxonomic tree. Identifying a single species using this process – by answering dozens of often-ambiguous questions, such as, "are the leaves flat and thin?" – may take several minutes or even hours. This is difficult for experts, and exceedingly so (or even impossible) for amateurs.

In this work, we show how computer vision can be used to significantly simplify the plant species identification problem. Our automatic system requires that a

A. Fitzgibbon et al. (Eds.): ECCV 2012, Part II, LNCS 7573, pp. 502–516, 2012.
© Springer-Verlag Berlin Heidelberg 2012



(a) Home (b) Recognition (c) Nearby Species

Fig. 1. Screenshots of the iPad version of Leafsnap. (a) The Home screen, with an image of a flower from the Redbud tree (*Cercis canadensis*). (b) A user's collection (left) and recognition results for a leaf (right), with the image and corresponding segmentation at the top and ranked species results below. (c) The Nearby Species screen, with pins denoting trees recently labeled by Leafsnap users around New York City.

single leaf specimen is photographed on a solid light-colored background. The recognition process consists of:

Classifying whether the image is of a valid leaf, to decide if it is worth processing further, using a binary classifier applied to gist features [1]. (Section 2)

Segmenting the image to obtain a binary image separating the leaf from the background. We do this by estimating foreground and background color distributions in the saturation-value space of the HSV colorspace. (Section 3)

Extracting curvature features from the binarized image for compactly and discriminatively representing the shape of the leaf. We robustly compute histograms of curvature over multiple scales using integral measures of curvature. (Section 4)

Comparing the features to those from a labeled database of leaf images and returning the species with the closest matches. Due to the discriminative power of the features and the size of our labeled dataset, we use a simple nearest neighbor approach with histogram intersection as the distance metric. (Section 5)

All computation is completed in about 5 seconds, and can be trivially parallelized across many machines. Users are then shown the top matches and make the final identification themselves, by examining additional content present in the app, such as high-quality images of the species and textual descriptions of their characteristics. The complete Leafsnap system is discussed in Sec. 6.

Automatic species identification has been an area of recent but growing interest in computer vision. [2] describes a system that combines human input with computer vision results to assist in the identification of birds. In the plant world, [3] describes a system that can automatically identify plant species using images of flowers. While this system shows impressive results, its concerns

are largely complementary to ours. Identification of species (or varietals) from flowers is of great interest to gardeners and flower enthusiasts. However, flowers are of limited value in systems used in biodiversity studies or for identifying local trees, because flowers are not present throughout most of the year. A summary of recent work on algorithms for plant identification and as well as detailed evaluations are described in the CLEF 2011 plant images classification task [4].

The work described here is most closely related to [5,6], which describe a much earlier version of the current Leafsnap system. (Other related works on plant identification can be found in those references.) Apart from the basic approach of using a color-based segmentation algorithm, all aspects of our work are completely new. In particular: the use of a pre-filter on input images; numerous speedups and additional post-processing within the segmentation algorithm; the use of a simpler and more efficient curvature-based recognition algorithm instead of Inner Distance Shape Context (IDSC) [7]; a much larger dataset of images, including over $5,000$ taken using mobile phones; and the creation and deployment of an interactive system for use by non-expert users.

More generally, our system is an example of fine-grained visual categorization: the discrimination of instances into classes that are more specific than basic level categorization (*e.g.*, leaves or animals) yet not as specific as the identification of individuals (*e.g.*, face recognition). This is a rapidly growing area of computer vision. To spur further related research, we have publicly released our large dataset of real-world leaf images, as well as optimized source code for both the segmentation and feature extraction algorithms[1].

2 Leaf/Non-leaf Classification

Untrained users initially try to take photos of leaves *in-situ*, with multiple leaves present amid clutter, often with severe lighting and blur artifacts, resulting in images that we cannot handle (usually due to segmentation failures). In addition, many users also take photos of objects that are not leaves. We address both of these issues by first running a binary leaf/non-leaf classifier on all input images. If this classifier detects that an input image is not valid – of a single leaf, placed on a light, untextured background with no other clutter – we inform the user of this fact and instruct them on how to take an appropriate image. We found this simple procedure very helpful for training users without the need for long tutorials or help pages, which often go unread. It also greatly reduces the computational load on our server, as images that fail this classification (about 37.1%) are discarded from further processing.

We implement this classifier using gist features [1] computed on the image, which are fed into a Support Vector Machine (SVM) [8] with an RBF kernel as the classification function. To ensure that the gist values are scale-invariant, we resize the input image to 300×400 (rotating it by $90°$ if it has the wrong aspect ratio). We use the libsvm [9] implementation of SVMs and the LEAR [10] implementation of gist. The classifier was trained on 5,972 manually labeled

[1] `http://leafsnap.com/dataset/` and `http://leafsnap.com/code/`

images and takes 1.4 seconds to run per image. We have found the classifier to be extremely effective, with very few invalid images slipping through, and the occasional false negatives addressed by the user simply taking another photo.

3 Color-Based Segmentation

Our system uses the distinctive shapes of leaves as the sole recognition cue. Other features such as the color of the leaf, its venation pattern, or images of the flowers are not suitable for various reasons – they are either too highly variable across different leaves of the same species, undetectable due to the poor imaging capabilities of most mobile phone cameras, or only present at limited times of year. Reliable leaf segmentation is thus crucial in order to obtain shape descriptions that are sufficiently accurate for recognition. We require that users photograph leaves against a light, untextured background; however, even with this requirement, segmentation is challenging due to shadows, blur, fine-scale structures on leaves (such as serrations and thin stems), and specular reflections.

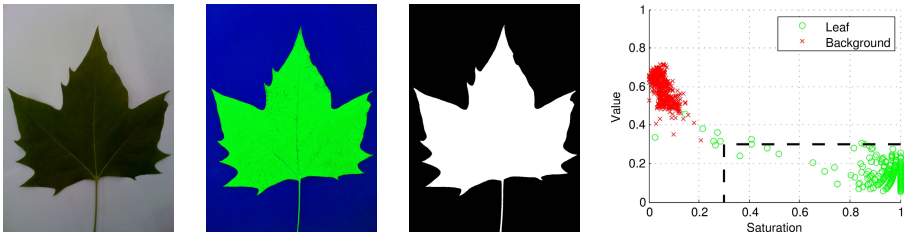
We segment images by estimating foreground and background color distributions and using these to independently classify each pixel. This initial segmentation, solved using Expectation-Maximization (Sec. 3.1), is then post-processed to remove false positive regions (Sec. 3.2) and the leaf stem (Sec. 3.3). Our color-based segmentation has several advantages compared to other approaches. Leaves vary greatly in shape. Some species of leaves are compound (consisting of small leaflets); others are found grouped into clusters (*e.g.*, pine needles). This gives rise to complex segmentation boundaries that are difficult to handle for edge-based methods, or region-based methods that bias towards compact shapes. Our color-based pixelwise approach works much better. In addition, by not making absolute assumptions about the color distributions, our clustering approach is able to adapt to major sources of color variability such as lighting changes and natural variations in leaf color (*i.e.*, although many leaves are a deep green, others have yellowish or brownish hues). Finally, our approach is very fast, suitable for use in an interactive application.

3.1 Initial Segmentation via Expectation-Maximization

We experimented with different color spaces and noted that both the saturation and value of the HSV space were consistently useful to distinguish leaf pixels from the background (see Fig. 2). Hue is not useful because the background often has a greenish tinge due to reflections from the leaf or surrounding foliage. Pixel clustering is thus performed in the saturation-value space.

The probability distribution of a pixel $\boldsymbol{x}$, represented by its saturation and value, is modeled as the sum of two Gaussians:

$$p(\boldsymbol{x}|\boldsymbol{\Theta}) = \sum_{k=1}^{2} \frac{1}{2}\, p(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}), \tag{1}$$



(a) (b) (c) (d)

Fig. 2. A leaf image (a) in original RGB, (b) converted to saturation-value space, and (c) segmented using EM (before any post-processing). (d) The distribution of leaf and background pixels in saturation-value space. The outlined lower-right portion of the space tends to contain leaf pixels. During EM, pixels that fall inside this region are weighted such that their sum matches that of pixels outside it.

where each $p(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$ is a Gaussian with mean $\boldsymbol{\mu}_k$ and a common shared covariance $\boldsymbol{\Sigma}$. The set of model parameters is $\boldsymbol{\Theta} = \{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}\}$. Note that we have assigned each Gaussian an equal weight of 1/2 for now.

We initialize each of the two Gaussians near the expected center of their respective distributions, so that they converge to the corresponding clusters when provided with pixel data from a new image. It is also important that the covariance matrix be initialized to a value near the expected final values to quickly converge to appropriate solutions. We found that a scaled identity matrix worked well. The segmentation is done via EM, by alternating between independently estimating probabilities of each pixel using the current parameters, and updating the parameters using current pixel probabilities.

Straightforward application of the above method works well in general, but has some difficulty with pine leaves, in which the leaf comprises only a tiny fraction of the image. The problem is caused by the model assuming the same weight for each of the two Gaussians, in effect biasing the result towards clusters that are assigned equal numbers of pixels. We resolve this by pixel weighting. We manually define a rectangular region in saturation-value space that tends to contain leaf pixels (see Fig. 2). Then, prior to running EM, we assign different weights to pixels inside and outside this region, such that each set of pixels has equal total weight. This greatly improves segmentation performance on pines without significantly affecting performance for other leaves.

We have optimized our implementation for speed, allowing it to handle reasonably large images. First, the fact that the covariance matrix is shared between the two Gaussians brings a significant speed advantage. In the two-class case, the posterior function defining cluster memberships takes on a linear logistic form [11], which can be efficiently computed. Specifically, if we denote the label of pixel $\boldsymbol{x}$ as $z \in \{1, 2\}$, then we can write

$$p(z = 1|\boldsymbol{x}) = 1/[1 + \exp(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x})], \text{ where} \tag{2}$$

$$\boldsymbol{\beta} \equiv (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1}, \text{ and} \tag{3}$$

$$\beta_0 \equiv -\frac{1}{2}(\boldsymbol{\mu}_2^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_2 - \boldsymbol{\mu}_1^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_1). \tag{4}$$

After computing Equation 2 for each pixel $\boldsymbol{x}$, we can quickly obtain $p(z = 2|\boldsymbol{x}) = 1 - p(z = 1|\boldsymbol{x})$. A second optimization is that it is possible to obtain reliable estimates of the Gaussian parameters using only a fraction of the pixels in an image. We thus use a downsampled version of the original image during EM. Once the procedure has converged, the labels for each pixel can be quickly computed over the complete original image using only Equation 2. Using these optimizations, segmentation runs on average in 0.062 seconds on a 700×525 image using 25% of pixels during EM, converging in 6.6 iterations on average.

3.2 Removing False Positive Regions

After the EM procedure, each pixel is assigned to leaf or background based on its membership to either of the two Gaussians. This results in an initial segmentation that sometimes contains false positive regions, caused by uneven backgrounds, shadows, or extraneous objects in the picture. Another type of false positive region can appear at the outer border of the image. Most users place the leaf on a white sheet of paper when taking the picture. It is common to find that some parts of the image border lay outside the piece of paper, potentially giving rise to falsely detected regions (see Fig. 3, rows 2-3).

Our post-processing procedure aims to remove such false positive regions. We first compute connected components on a dilated version of the segmentation. Any connected component that has a large boundary on the image border (relative to its area) is then excluded, which eliminates regions that have fallen outside the white background. The largest of the remaining components is then taken to be the leaf. This step takes 0.006 seconds for a 700×525 image.

3.3 Removing the Stem

At this point, the stem of the leaf may or may not be present in the segmentation. The original leaf might not have had a stem to begin with, or it might have been lost during segmentation, which may occur when the stem has a lighter color than the leaf. Even when the stem is correctly segmented, it may vary in length depending on how the user picked the leaf. To standardize the shape, we remove the stems from all segmentations through a series of morphological operations.

First, the set of all thin structures that protrude from the leaf is determined. This is done by taking the top-hat transformation of the segmentation [12], using as structuring element a disc with diameter larger than the width of any potential stem. For a binary image B and structuring element s, the top-hat is

$$T_{\text{hat}}(B) = B - B \circ s, \tag{5}$$

where $\circ$ denotes the opening operation: an erosion followed by a dilation.

Next, we determine which of these candidate structures is most likely to be the stem. First, we note that removing the stem should not change the number of connected components in the segmented leaf or in the background; thus, only such candidates are considered as possible stems. Of the possible stem candidates, we only consider those of appropriate size, and finally remove the one

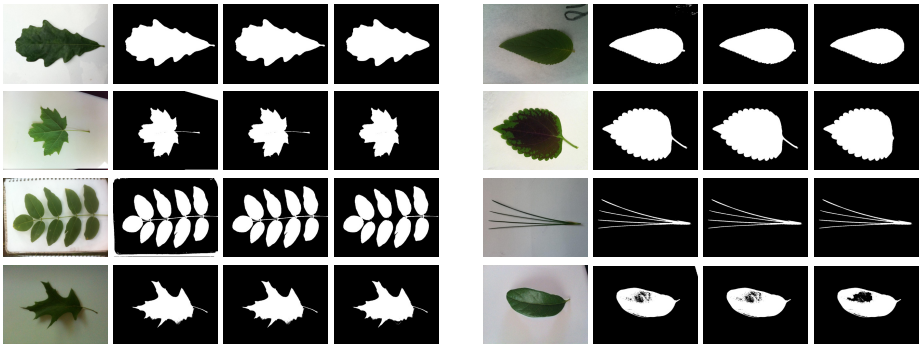


Fig. 3. Segmentation results. (l-r): input image, initial segmentation, results after removing false positive regions, final result after stem removal. The last row shows typical failure cases, due to shadows (left) and specular highlights (right).

that is most elongated (*i.e.*, with the highest ratio of the two principal moments). This step takes 0.036 seconds for a 700×525 image.

3.4 Segmentation Results

Segmentation results are shown in Fig. 3. In general, segmentation succeeds on the vast majority of valid input images (those that pass the initial leaf classifier). We have observed that the most frequent segmentation errors are caused either by shadows cast by the leaf onto the background, which show up as false positives (*e.g.*, bottom row, left), or by specularities present in certain shiny leaves, which show up as false negatives (*e.g.*, bottom row, right). We have also experimented with other, more sophisticated segmentation methods, but found that these are generally too slow, and/or do not perform adequately on more complex images, such as compound leaves or pines. A forthcoming journal article evaluates various segmentation approaches on leaves, showing the efficacy of our method.

4 Curvature-Based Shape Features

Leaf shape can be effectively represented using multiscale curvature measures. Fig. 4a shows a diagrammatic example with segmented images of four different leaf species, and histograms of curvature along the boundary for each shape, computed at two different scales – coarse (large radius) on top, and fine (small radius) on bottom. The pair of images in each row share the same general shape, but the images on the left have a smooth boundary, and the ones on the right have a serrated boundary. Thus, the histograms of coarse-scale values differ for each *row*, while the histograms of fine-scale values differ for each *column*. By using both histograms together, we can distinguish these shapes from each other. We expand upon this basic idea to extract our features.

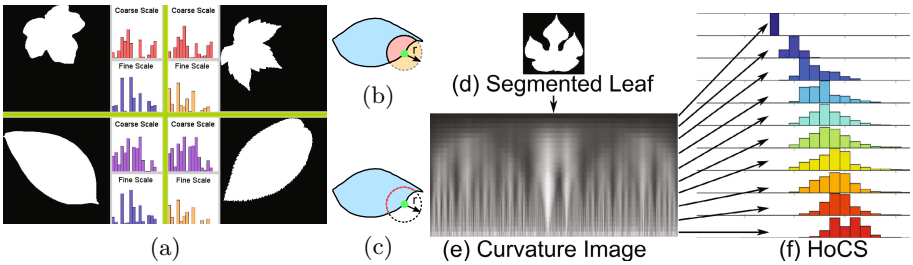


Fig. 4. (a) Histograms of curvature at two different scales are sufficient to distinguish these leaves of four different species. By computing these features over many scales, we can reliably distinguish between hundreds of species. Curvature values can be robustly computed using either (b) the area measure, the fraction of the circle's area contained inside the object (shown in pink); or (c) the arclength measure, the fraction of the circle's perimeter contained inside the object (shown in red). The scale is determined by the radius r of the circle. Using these measures, we take (d) the segmented leaf's contour and extract (e) a curvature image, which shows curvatures for each contour point along the x-axis and at different scales along the y-axis. (f) By taking histograms along each row, we create the *Histograms of Curvature over Scale* (HoCS) feature.

Curvature is a fundamental property of shape and has thus attracted much attention from the vision community; however, when dealing with images on discrete pixel grids, the elegant mathematical definitions of curvature are tricky to implement in a stable manner. Typically, curvatures are computed using differential techniques, which amplify noise, are sensitive to the orientation and scale of captured images, and are difficult to measure at multiple scales. Instead, we can use *integral measures* to compute functions of the curvature at a boundary point [13,14,15]. One such measure in 2D is the area of intersection of a disk centered at a contour point and the inside of the contour (see Fig. 4b). For straight, concave, and convex boundaries, the fraction of the disk intersected will be $=$, $>$, or < 0.5, respectively. Another measure uses the arclength: the fraction of the disk's perimeter inside the contour (Fig. 4c). Furthermore, these representations naturally capture scale by the radius of the disk – perturbations much smaller than the disk radius are ignored. Thus, a serrated boundary will show large, alternating curvatures at a fine scale, but a uniform curvature at a coarse scale. Unlike their differential counterparts, integral measures are fast and easy to compute for images on discrete grids, invariant to rotation, insensitive to small segmentation and discretization errors, independent of the topological complexity, and straightforward to extend to 3D [15].

Given this reliable method for computing curvatures, how should one construct a feature vector suitable for classification? Manay *et al.* [14] concatenated integral measures of curvature along the contour of the object into a single feature vector. But this has several disadvantages in our domain: different contours will have different lengths, making them difficult to compare; contours must be aligned to have the same starting point, otherwise they will not match; minor changes in topology or orientation can cause huge changes in the feature vector;

and there is no straightforward way to handle multiple contours. Instead, we compute histograms of the curvature values at each scale, and concatenate these histograms together to form the Histograms of Curvature over Scale (HoCS) feature. Histograms have the benefit of being compact, simple to represent, not requiring alignment, and fast to compare using metrics such as L_1.

Our approach is very different from the IDSC-based [7] recognition described in [5]. IDSC represents shape implicitly via histograms of distances and angles from sample points on the contour to all other points, along a path inside the leaf shape. This is significantly more expensive to compute ($O(N^3)$ vs $O(N)$ in the number of sample points), requires a complex matching algorithm comparing sets of points ($O(N^2)$) rather than simple histogram intersection ($O(N)$), and harder to reason about than curvature, which is well understood. In addition, IDSC may be more sensitive to segmentation errors.

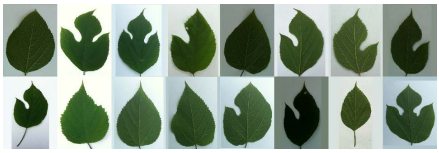
4.1 Computing Histograms of Curvature over Scale

To get the most accurate results, we implement the integral measures of curvature carefully and efficiently. To remove problems caused by "holes" in the shape (*e.g.*, due to segmentation drop-outs), we completely fill-in the contours obtained from the segmented images prior to curvature extraction. For scale invariance, we resize these filled-in images to a common area before extracting curvatures from the boundaries. To reduce histogramming artifacts, we use bilinear interpolation to do soft-binning of curvature values.

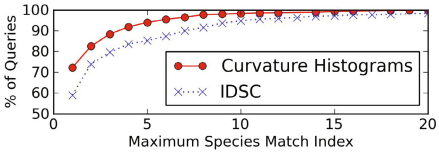
The simplicity of these measures also allows for new speed optimizations. When computing the area measure, the change in intersected area from one contour point to the next can be computed by simply checking the crescent-shaped boundaries of the circle in the direction of the shift. Since the change in position is limited to 4 or 8 different translations (depending on the connectedness of the contour points) and our set of radii are fixed, we precompute the offset coordinates. This results in approximately $2\pi r$ pixels to check for each contour point, *i.e.*, only those that have changed from the last location of the disk, resulting in an order of magnitude improvement over a naive $4r^2$ check over the whole disk. For the arclength measure, we first find the points on the segmented shape contour that intersect the disk edge. The contour is represented as an array of contiguous point locations, and so we search along the contour in both directions away from the current point (the center of the disk), initially jumping r points away from the current point, since the disk has radius r. Once we find the appropriate points, we go along the disk edge (whose coordinates have been precomputed) and simply check each pixel's segmentation value in the image.

From a given contour, we extract a *curvature image* as shown in Fig. 4e. Each row in this image contains the integral measures of curvature values computed along the contour at a given scale; each column contains the curvature measures at all scales for a single contour point. Finally, we compute histograms of curvature values at each scale (*i.e.*, from each row of the curvature image, as in Fig. 4f) and then concatenate these histograms to form the HoCS feature. The complete feature extraction process takes on average 0.11 seconds per image.

(a) Thumbnails of all 184 tree species from the Northeastern United States



(b) Images of *Broussonettia papyrifera*



(c) Recognition accuracy

Fig. 5. (a) A single image from all 184 species in our database. Compare the often small inter-species variation against the (b) large intra-species variation for *Broussonettia papyrifera*. (c) Plot showing the percentage of queries with the correct species appearing in the top N results returned. The correct match is within the top 5 results for 96.8% of queries using curvature histograms, compared to only 85.1% using IDSC [7].

5 Species Identification Using Nearest Neighbors

Identification is done by running a nearest neighbors search using the HoCS feature extracted from the input image as the query. The search database consists of features extracted from 23,915 "clean" lab images of pressed leaves (captured using a high-quality camera) and 5,192 field images taken by mobile devices. The field images contain varying amounts of blur and were photographed with different viewpoints and illumination. A single image from each of the 184 species in our dataset can be seen in Fig. 5a. The difficulty of the problem can be seen, *e.g.*, by comparing the often small inter-species variation against the large intra-species variation shown in Fig. 5b for leaves of the *Broussonettia papyrifera*.

The feature dimensionality is $N = 25$ scales $\times$ $B = 21$ bins per scale $= 525$ values for histograms of the area and arclength measures (each). Histograms are compared using the histogram intersection distance, defined as:

$$d(\mathbf{a}, \mathbf{b}) = N - \sum_{i}^{B} min(a_i, b_i), \tag{6}$$

where the histogram at each scale has been normalized to unit length. (Other metrics – including L_1, L_2, Bhattacharyya distance, and χ^2 – did not perform as well.) Retrieval takes 0.31 seconds using a linear scan of the database. The top 25 results are presented to the user for final identification.

To quantitatively measure recognition performance, we perform leave-one-image-out species identification, using only the field images as queries, matching against all other images in the recognition database. The goal is to have the lowest possible *species match rank*, defined as the rank of the correct *species* (not *image*). This is the most appropriate metric for our application because we show users only the list of matched species, not images. So even if there are, *e.g.*, 20 matches to images of a particular incorrect species, those will count as only a single species error from the user's point of view.

A plot of recognition rates on a subset of our data[2], as a function of the maximum species match rank, is shown in Fig. 5c. Performance for IDSC [7] (used in [5]) is shown as the dotted blue line, and for our curvature histogram features as the solid red line. 96.8% of queries have a species match rank of 5 or lower with our method, *i.e.*, within the top 5 results shown to the user, which is substantially better than IDSC's 85.1%. This vast improvement highlights the effectiveness of our approach for shape retrieval on real-world images.

6 The Leafsnap System

To allow the general public to use the results of this research, we have implemented a complete end-to-end recognition system and packaged it as an electronic field guide called Leafsnap. The recognition engine consists of a backend server that accepts input images from various front-end clients. Currently, we have front-end apps for the iPhone and iPad devices, with work on Android devices in progress. The Leafsnap app contains high-quality photographs (taken by the non-profit group *Finding Species*) and botanist-curated descriptions of all the 184 tree species of the Northeastern United States. This already represents a significant step up from a traditional field guide: as a software application, operations such as browsing, sorting, and textual searching are trivial. Our automatic visual recognition system further improves on this by providing the user with the most likely candidate species for a given query input image. The user makes the final classification decision by visually comparing the actual plant to the high-quality photographs in the app, which span all aspects of the species – the leaf, flower, fruit, petiole, bark, *etc*.

Figures 1 and 6 show screenshots from the iPad and iPhone apps, respectively. For the latter, in scanline order: (a) the home screen, with a randomly-chosen image cycling every few seconds and access to educational games, (b) the browse screen, with a sortable and searchable list of all the species contained in the system, (c) the search functionality for finding particular species by scientific or common name, (d) the detail view for a particular species, showing the different images available for viewing, (e) the Snap It! screen, for performing automatic identification, (f) the returned identification results, in sorted order, (g-i) the manual verification stage as the user explores the images and textual descriptions of one of the results to confirm it as the correct match, (j) labeling the correct

[2] IDSC [7] was too slow to run a test on the entire dataset.

(a) Home (b) Browse (c) Search (d) Detail

(e) Snap It! (f) Results (g) Verification 1 (h) Verification 2

(i) Verification 3 (j) Label (k) Collection (l) Map

Fig. 6. Tour of the iPhone version of Leafsnap. See text for details

match, (k) the addition of that leaf to the user's collection for future reference, and (l) a map view showing where that leaf was collected.

There has been significant public interest in Leafsnap – nearly a million people have downloaded the apps since their launch in May 2011. We received press coverage from major news outlets around the world, including newspapers (The New York Times and Washington Post), wire services (the Associated Press and Reuters), radio (National Public Radio), magazines (Science), television (The Tonight Show) and blogs (Wired and Techcrunch). We have also received requests to use and contribute to the system by urban planners in local governments, educators throughout the U.S. and abroad, not-for-profit institutions working on issues of biodiversity, and citizen scientists eager to map and monitor the flora of their street, backyard, or local park.

The backend server is currently a single Intel Xeon machine with 2 quad-core processors running at 2.33 Ghz each, and 16 GB of RAM. Aside from high-resolution versions of some images, which are served via Amazon's Simple Storage Service (S3), all other operations are handled by our server. The total time for serving a single recognition request is 5.4 seconds (not including the initial image upload) and is easily parallelizable.

7 Discussion and Future Directions

We have built the first computer vision system for automatic plant species identification. Our system relies on computer vision for several key aspects, including classifying images as leaves or not, obtaining fine-scale segmentations of leaves from their backgrounds, efficiently extracting histograms of curvatures along the contour of the leaf at multiple scales, and retrieving the most similar species matches using a nearest neighbors search on a large dataset of labeled images.

All steps of our algorithm have been carefully designed to excel on our task of plant species identification; however, we feel that there are many practical lessons for others in the vision community. The effectiveness of our initial leaf classifier suggests that such simple expedients can greatly improve the overall performance of a public system. Our color-based segmentation outperforms other published segmentation methods, in large part because the constraints for this task – requiring the preservation of fine-scale features such as serrations and thin structures, as well as operating at interactive speeds – are more stringent than commonly assumed in the literature, yet frequent in many application domains. We have shown that curvature histograms are effective shape descriptors, both because they can be robustly computed using integral measures, and because they can be aggregated into histograms at multiple scales, which make for efficient and simple retrieval using nearest neighbors.

The high level of engagement with our Leafsnap system allows for many possible future directions. We are looking at ways of crowd-sourcing the collection of additional species for the system, as well as the verification of labels that users mark in the app. The latter also poses interesting machine learning questions, such as how to automatically determine the trustworthiness of particular users

or labels without requiring (extensive) expert supervision. Since our apps save the GPS coordinates and timestamp of each photo, we also hope to be able to map the biodiversity of a region over space and time. Finally, we would like to further explore the educational aspects of this project. This includes adding more collaborative features in the apps, *i.e.*, for entire classrooms to use together, as well as studying the efficacy of the app in teaching children (and adults) about plants. We believe that the tight feedback loop the app provides via its games and the visual recognition feature are ideal mechanisms for training people to recognize plants themselves and would like to quantify this effect.

The strong response to Leafsnap shows the potential for widespread adoption of computer vision technology, when appropriately packaged and distributed.

Acknowledgments. This work was supported by National Science Foundation grants #0968546, #0325867, and #1116631, as well as funding from Google.

References

1. Oliva, A., Torralba, A.: Modeling the shape of the scene: A holistic representation of the spatial envelope. IJCV 42, 145–175 (2001)
2. Branson, S., Wah, C., Schroff, F., Babenko, B., Welinder, P., Perona, P., Belongie, S.: Visual Recognition with Humans in the Loop. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010, Part IV. LNCS, vol. 6314, pp. 438–451. Springer, Heidelberg (2010)
3. Nilsback, M., Zisserman, A.: Automated flower classification over a large number of classes. In: Indian Conference on Computer Vision, Graphics and Image Processing (2008)
4. Goëau, H., Bonnet, P., Joly, A., Boujemaa, N., Barthelemy, D., Molino, J.F., Birnbaum, P., Mouysset, E., Picard, M.: The clef 2011 plant images classification task. In: CLEF (Notebook Papers/Labs/Workshop) (2011)
5. Belhumeur, P.N., Chen, D., Feiner, S.K., Jacobs, D.W., Kress, W.J., Ling, H., Lopez, I., Ramamoorthi, R., Sheorey, S., White, S., Zhang, L.: Searching the World's Herbaria: A System for Visual Identification of Plant Species. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008, Part IV. LNCS, vol. 5305, pp. 116–129. Springer, Heidelberg (2008)
6. Agarwal, G., Belhumeur, P., Feiner, S., Jacobs, D., Kress, W.J., Ramamoorthi, R., Bourg, N.A., Dixit, N., Ling, H., Mahajan, D., Russell, R., Shirdhonkar, S., Sunkavalli, K., White, S.: First steps toward an electronic field guide for plants. Taxon 55, 597–610 (2006)
7. Ling, H., Jacobs, D.: Shape classification using the inner-distance. TPAMI 29, 286–299 (2007)
8. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning 20 (1995)
9. Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology 2, 27:1–27:27 (2011)
10. Douze, M., Jégou, H., Sandhawalia, H., Amsaleg, L., Schmid, C.: Evaluation of gist descriptors for web-scale image search. In: International Conference on Image and Video Retrieval (2009)
11. Efron, B.: The efficiency of logistic regression compared to normal discriminant analysis. Journal of the American Statistical Association 70, 892–898 (1975)

12. Gonzalez, R., Woods, R.: Digital image processing. Pearson/Prentice Hall (2008)
13. Connolly, M.L.: Measurement of protein surface shape by solid angles. Journal of Molecular Graphics 4, 3–6 (1986)
14. Manay, S., Cremers, D., Hong, B.W., Yezzi, A.J., Soatto, S.: Integral invariants for shape matching. TPAMI 28, 1602–1618 (2006)
15. Pottmann, H., Wallner, J., Huang, Q.X., Yang, Y.L.: Integral invariants for robust geometry processing. Computer Aided Geometric Design 26, 37–60 (2009)