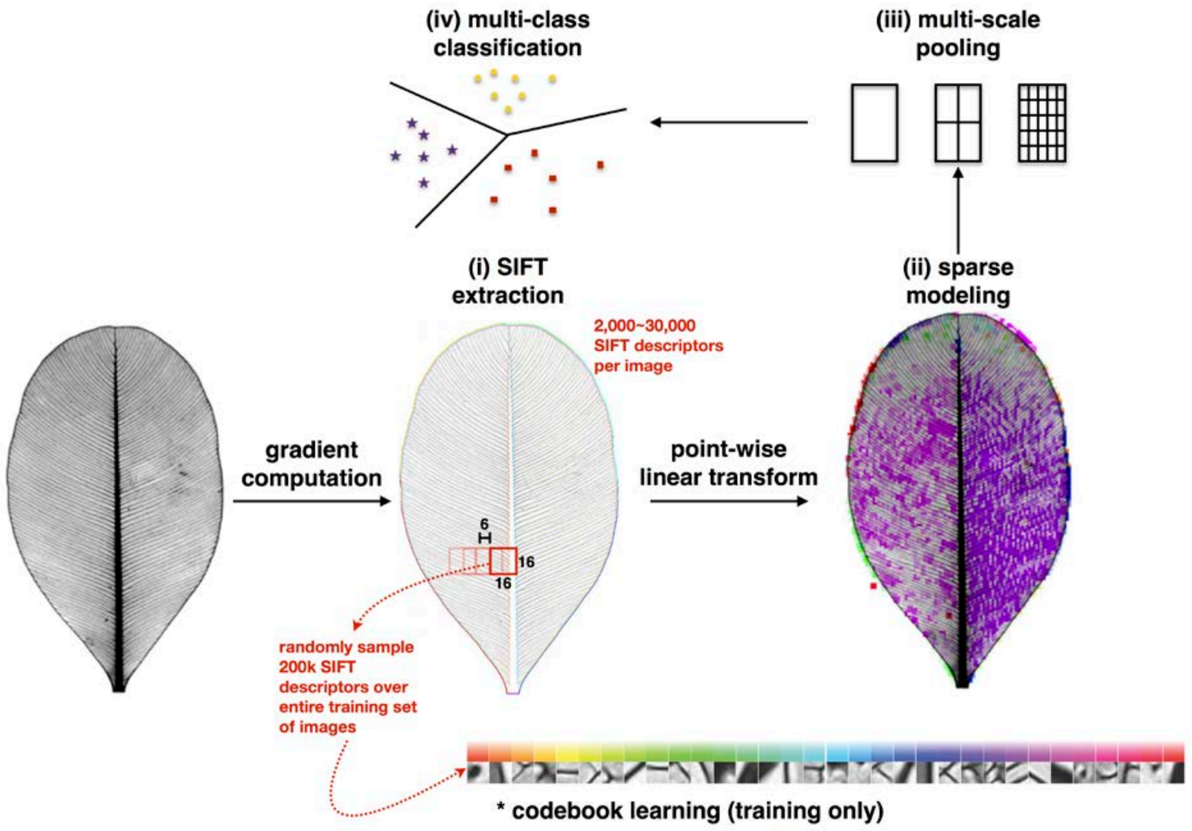
Dataset S2. Archive of computer code developed.

This supplementary material contains the MATLAB code used to produce the results reported in the paper as well as corresponding documentation.

1. System Overview


(iv) multi-class classification
(iii) multi-scale pooling
(i) SIFT extraction
(ii) sparse modeling
2,000~30,000 SIFT descriptors per image
gradient computation
point-wise linear transform
6
16
16
randomly sample 200k SIFT descriptors over entire training set of images
* codebook learning (training only)

The computer vision based leaf recognition system consists of four stages including *SIFT extraction*, *sparse modeling*, *multi-scale pooling* and *multi-class classification*.

SIFT extraction stage

(a) Each leaf image is first resized such that its maximum length or width dimension is 1024 pixels, while maintaining aspect ratio.
(b) Image patches with size of 16x16 in pixel are densely sampled from each resized leaf image with partial overlap.
(c) A 128-dimensional SIFT descriptor is extracted from each image patch.

Sparse modeling

(a) Given a training set of 128-dimensional SIFT descriptors, a sparse coding dictionary with 1024 elements is first learned.
(b) For any 128-dimensional SIFT descriptor extracted from an image patch of a leaf image, a 1024-dimensional sparse coefficient vector is computed.

Multi-scale pooling

(a) From each spatial grid, a 1024-dimensional feature vector can be obtained by max pooling sparse coefficient vectors of image patches inside this grid.

(b) A total of 21 feature vectors are concatenated into a 21504-dimensional feature vector from each leaf image.

Multi-class classification

(a) Given a training set of 21504-dimensional feature vectors computed from leaf images coming from all classes, a multi-class Support Vector Machine (SVMs) model can be learned.

(b) For any query leaf image, the 21504-dimensional feature vector computed from it is fed to the learned SVM model, which then outputs its class label.

2. Installation

The code was written in MATLAB and successfully tested using MATLAB R2014a on Windows, Linux and Mac operating systems.

Before running the code contained in this folder, please first download two public toolboxes from the Internet:

1) Download the LIBSVM toolbox at http://www.csie.ntu.edu.tw/~cjlin/libsvm/, compile it and set up its path in MATLAB.
2) Download the SPAMS toolbox at http://spams-devel.gforge.inria.fr and compile it and set up its path in MATLAB.

You can download the pre-computed SIFT descriptors of all leaf images from http://serre-lab.clps.brown.edu/resources/LeafSIFT.zip and unzip it inside this folder.

3. Run the code

To retrain the models and reproduce the recognition results reported in Table 1 of the manuscript, please first launch MATLAB. Create five individual files with the code included in this pdf and run the command “demo_main.m”.

4. Contact us

If you encounter any problem or if you have questions about the code, please contact us at s.zhang@hit.edu.cn, thomas_serre@brown.edu, or pwilf@psu.edu. We are very interested in hearing how the system and data are used by others, so please let us know if you find this resource useful.

demo_main.m

```
% This code is used to reproduce the recognition results reported in  %
Table 1 of the manuscript. Note that before running this code, please %
make sure you have downloaded the SIFT descritpors and put them into the
% folder 'LeafSIFT'

rand('state', 0);
randn('state', 0);

%% Parameter setting
task                = 'family'; % or 'order'
threshold           = 100;  % or 50
maximalSide         = 1024; % the maximum dimension (length or width) in pixels
split               = 0.5; % the proportion of total leaf images used for training
num_bases           = 1024; % the number of elements of the learned sparse coding dictionary
lc                  = 100;  % SVM parameter

num_patches         = 200000; % the number of samples for sparse coding dictionary learning
param.K             = num_bases;
param.lambda        = 0.15; % regularization parameter for sparse coding
param.numThreads    = 4;
param.batchsize     = 400;
param.iter          = 1000;

pyramid             = [1, 2, 4]; % spatial pyramid parameters
nRounds             = 10; % the number of running
bShow               = 0;

structure_dir = 'data_structure'; % the directory of the data strucure file
sift_dir = 'LeafSIFT'; % the directory of the SIFT descriptors
feature_dir  = 'features'; % the directory of the features
dictionary_dir  = 'dictionary'; % the directory of the dictionary
result_dir  = 'results'; % the directory of the results

%% load the strucutre file for the specific recognition task
data_structure_file = sprintf('%s/%s_%d.mat', structure_dir, task, threshold);
load(data_structure_file);
if strcmp(task, 'family')
    data_structure.class_names = family_names;
```

```
elseif strcmp(task, 'order')
    data_structure.class_names = order_names;
end
data_structure.image_names = image_names;
data_structure.labels = labels;

dim_features = sum(num_bases*pyramid.^2); % the dimension of the final feature vectors fed to SVM
num_images = length(data_structure.image_names);

clabel = unique(data_structure.labels);
nclass = length(clabel);

accuracy = zeros(nRounds, 1);
for r=1: nRounds

    feature_folder = sprintf('%s/Exp1_Task_%s_threshold_%d_Resize_%d_split_%4.2f_numBases_%d_Round_%d', ...
        feature_dir, task, threshold, maximalSide, split, num_bases, r);
    if ~exist(feature_folder, 'dir')
        mkdir(feature_folder);
    end

    dictionary_folder =
sprintf('%s/Exp1_Task_%s_threshold_%d_Resize_%d_split_%4.2f_numBases_%d_Round_%d', ...
        dictionary_dir, task, threshold, maximalSide, split, num_bases, r);
    if ~exist(dictionary_folder, 'dir')
        mkdir(dictionary_folder);
    end

    % choose training samples and test samples for this  round
    samples_file = sprintf('%s/samples_idx.mat', dictionary_folder);
    if ~exist(samples_file, 'file')
        tr_idx = [];
        ts_idx = [];
        for i = 1:nclass,
            idx_label = find(data_structure.labels == clabel(i));
            num = length(idx_label);
            if split < 1
                tr_num = round(num*split);
            else
                tr_num = split;
            end
            idx_rand = randperm(num);

            tr_idx = [tr_idx idx_label(idx_rand(1:tr_num))];
            ts_idx = [ts_idx idx_label(idx_rand(tr_num+1:end))];
        end
        save(samples_file, 'tr_idx', 'ts_idx');
    else
        load(samples_file);
    end

    % learning sparse coding dictionary
    dict_file = [dictionary_folder '/dict.mat'];
    if ~exist(dict_file, 'file')
        [patches, xs, ys] = Collect_SIFT_descriptors(sift_dir, data_structure, tr_idx, num_patches);
        B = mexTrainDL(patches,param);
        save(dict_file, 'B');
    else
        load(dict_file);
    end

    % Compute sparse coding features
    sc_fea_all = zeros(dim_features, num_images);
    sc_label_all = data_structure.labels;
    for i=1: num_images

        [~, fname] = fileparts(data_structure.image_names{i});
        f_sift_path = fullfile(rt_sift_dir, [fname, '_sift.mat']);
        f_sc_fea_path = fullfile(feature_folder, [fname, '.mat']);
        if  ~exist(f_sc_fea_path, 'file')
            fprintf('Compute Sparse coding features for %d/%d image\n', i, num_images);
            load(f_sift_path);
            fea = Compute_Features(feaSet, B, pyramid, param);
            save(f_sc_fea_path, 'fea');
        else
            fprintf('Load Sparse coding features for %d/%d image\n', i, num_images);
            load(f_sc_fea_path);
        end
        sc_fea_all(:, i) = fea;
    end
    % Perform classification
    tr_fea = sc_fea_all(:, tr_idx)';
    tr_label = sc_label_all(tr_idx)';
    ts_fea = sc_fea_all(:, ts_idx)';
    ts_label = sc_label_all(ts_idx)';

    kparam = 1;
    Ktr = Compute_RBF_kernel(tr_fea, tr_fea, kparam);
    Kte = Compute_RBF_kernel(ts_fea, tr_fea, kparam);
    Ktr = double([(1:size(Ktr,1))' Ktr]);
    Kte = double([(1:size(Kte,1))' Kte]);

    option = ['-t 4 -c ' num2str(lc)];
    [C, decmatrix, traintime, testtime] = libsvmova(tr_label, Ktr, ts_label, Kte, lc);

    leaf_frame_acc = mean(C == ts_label);

    confusion_matrix =  genConfus(ts_label,C,data_structure.class_names,bShow);
    accuracy(r) = mean(diag(confusion_matrix));

end

% save the results
save(result_file, 'accuracy');
```

Collect_SIFT_descriptors.m

```
function [descriptors, xs, ys] = Collect_SIFT_descriptors(sift_folder, data_structure, tr_idx,
num_descriptors)
% Functionality:
%       Collect a number of SIFT descriptors from the training leaf images
%       for learning the sparse coding dictionary
% Input:
%       sift_folder     --- the folder where the SIFT descriptors of all leaf
%                           images were saved
%       data_structure  --- the structure variable with fields specifying the
%                           names of leaf images and the corresponding labels
%       tr_idx          --- the indices of training leaf images
%       num_descriptors --- the number of SIFT descriptors to be collected
%                           for dictionary learning
% Output:
%       descriptors     --- a matrix with column vectors being the SIFT
%                           descriptors randomly extracted from training leaf images
%       xs              --- the x coordinates of patches' centers from
%                           whcih the SIFT descriptors were extracted
%       ys              --- the x coordinates of patches' centers from
%                           which the SIFT descriptors were extracted
num_training_images = length(tr_idx);
num_per_img = round(num_descriptors/num_training_images);
num_descriptors = num_per_img*num_training_images;

descriptors = zeros(128, num_descriptors);
xs = zeros(1, num_descriptors);
ys = zeros(1, num_descriptors);
cnt = 0;
for i=1: num_training_images
    fprintf('Extracting training samples for dictionary learning from %d/%d image\n', i,
num_training_images);
    ind = tr_idx(i);
    [~, fname] = fileparts(data_structure.image_names{ind});
    f_sift_path = fullfile(sift_folder, [fname, '_sift.mat']);
    load(f_sift_path);
    num_fea = size(feaSet.feaArr, 2);
    rndidx = randperm(num_fea);
    num_per_img_actural = min(num_fea, num_per_img);
    descriptors(:, cnt+1:cnt+num_per_img_actural) = feaSet.feaArr(:, rndidx(1:num_per_img_actural));
    xs(:, cnt+1:cnt+num_per_img_actural) = feaSet.x(rndidx(1:num_per_img_actural));
    ys(:, cnt+1:cnt+num_per_img_actural) = feaSet.y(rndidx(1:num_per_img_actural));
    cnt = cnt+num_per_img_actural;
end
descriptors = descriptors(:, 1:cnt);
xs = xs(:, 1:cnt);
ys = ys(:, 1:cnt);
```

Compute_Features.m

```
function features = Compute_Features(feaSet, B, pyramid, param)
% Functionality:
```

```
%       Given the SIFT descriptors extracted from a leaf image and the
%       sparse coding dictionary, this function returns the computed
%       features from theleaf image by max pooling the sparse codign
%       responses over the dictoinary
% Input:
%       feaSet      --- SIFT descriptors extracted from a leaf image
%       B           --- Sparse coding dictionary
%       pyramid     --- pyramid scale [0, 1, 2]
%       param       --- Sparse coding parameters
% Output:
%       features    --- the computed features from the input leaf image

dSize = size(B, 2);
img_width = feaSet.width;
img_height = feaSet.height;

sc_codes=mexLasso(double(feaSet.feaArr),B,param);

sc_codes = abs(sc_codes);

% spatial levels
pLevels = length(pyramid);
% spatial bins on each level
pBins = pyramid.^2;
% total spatial bins
tBins = sum(pBins);

features = zeros(dSize, tBins);
features_ind = zeros(dSize, tBins);
beta_sum = zeros(dSize, tBins);
bId = 0;

for iter1 = 1:pLevels,

    nBins = pBins(iter1);

    wUnit = img_width / pyramid(iter1);
    hUnit = img_height / pyramid(iter1);
```

```matlab
    % find to which spatial bin each local descriptor belongs
    xBin = ceil(feaSet.x / wUnit);
    yBin = ceil(feaSet.y / hUnit);
    idxBin = (yBin - 1)*pyramid(iter1) + xBin;

    for iter2 = 1:nBins,
        bId = bId + 1;
        sidxBin = find(idxBin == iter2);
        if isempty(sidxBin),
            continue;
        end
        [features(:, bId), max_ind] = max(sc_codes(:, sidxBin), [], 2);
        features_ind(:, bId) = sidxBin(max_ind);
        beta_sum(:, bId) = sum(sc_codes(:, sidxBin),  2);
    end
end

if bId ~= tBins,
    error('Index number error!');
end

features = features(:);
features = features./sqrt(sum(features.^2));
```

Compute_RBF_kernel.m

```matlab
function K = Compute_RBF_kernel(feaset_1, feaset_2, kparam)
% Functionality:
%       Compute the RBF kernel matrix between two sets of features
% Input:
%       feaset_1     --- feature matrix, each row denotes one sample
%       feaset_2     --- feature matrix, each row denotes one sample
%
% Output:
%       K            --- kernel matrix

if (size(feaset_1,2) ~= size(feaset_2,2))
    error('sample1 and sample2 differ in dimensionality!!');
end
```

```
[L1, dim] = size(feaset_1);
[L2, dim] = size(feaset_2);

% If sigle parammeter, expand it.
if length(kparam) < dim
    a = sum(feaset_1.*feaset_1,2);
    b = sum(feaset_2.*feaset_2,2);
    dist2 = bsxfun(@plus, a, b' ) - 2*feaset_1*feaset_2';
    K = exp(-kparam*dist2);
else
    kparam = kparam(:);
    a = sum(feaset_1.*feaset_1.*repmat(kparam',L1,1),2);
    b = sum(feaset_2.*feaset_2.*repmat(kparam',L2,1),2);
    dist2 = bsxfun(@plus,a,b') - 2*(feaset_1.*repmat(kparam',L1,1))*feaset_2';
    K = exp(-dist2);
end

end
```

genConfus.m

```
function C= genConfus(truth,pred,classes,bDisplay)
% Functionality:
%       Given the ground truth labels of the test samples and the predicted
%       labels by the classifier, this function return the confusion matrix
% Input:
%       truth       --- The ground truth labels of the test samples
%       pred        --- the predicted labels of the test samples
%       classes     --- the names of the multiple classes for classification
%       bDisplay    --- whether display the confusion matrix
% Output:
%       C           --- the confusion matrix

Csize = length(classes);
C = zeros(Csize,Csize);
actionCount = zeros(Csize,1);

for action = 1:Csize
    Tind = find(truth == action);
    Plabel = pred(Tind);
    actionCount(action) = length(Tind);
    Punique = unique(Plabel);

    for i = 1:length(Punique)
        C(action,Punique(i)) = length(find(Plabel == Punique(i))) / length(Tind);
    end
end

d = diag(C); dd = d; dd(find(actionCount == 0 )) = [];
diagAcc = mean(dd);

if bDisplay
    displayConfus(C,actionCount,d,diagAcc,truth,pred,classes);
end

function displayConfus(C,actionCount,d,classes)

    action_truth = cell(1,length(classes));
    action_pred = cell(1,length(classes));
    for i = 1:length(classes)
        action_truth{i} = [classes{i} ' ' num2str(actionCount(i))];
        action_pred{i} = sprintf('%d',round(d(i)*100));
    end

    figure,
    imagesc(C),colorbar, xlabel('prediction'), ylabel('human');
    set(gca,'XTick',[1:length(unique(classes))],'XTickLabel',action_pred,'YTick',...
        [1:length(unique(classes))],'YTickLabel',action_truth)
```