Instruction Manual

Running the Project

This project is split into two sections: feature extraction/descriptor construction and SVM training/classification. The feature extraction and descriptor construction script (`learn_image_vectors.m`) preprocesses the images from all 4 species, extracts the LBP and SIFT features from all the images, performs dictionary learning for the LBP and SIFT features and applies spatial pyramid matching to produce one image vector for each input image. These results are saved (at *intermediate_results/trial_<trial_number> _dictsize_<dictionary_size>_iter_<num_iterations>_lambda_<lambda>*) to minimize the amount of time spent running this code. This script is time consuming (runtime is between 20 and 100 minutes depending on hardware).

Run `classification.m` to (optionally) load image vectors created by `learn_image_vectors.m`, train SVMs and perform classification. The output of `classification.m` is cross-validation scores and final results for: SIFT, LBP and boosting (combines SIFT and LBP).

Some image vector files have been provided for convenience.

Runtime for `classification.m`: about 370 seconds.

Function Descriptions

`preprocess.m`
- INPUT: cell array of image filenames, desired image size
- OUTPUT: cell array of 256x256 greyscale images
- EXAMPLE: `image_list = preprocess(image_filenames, [256, 256])`

Feature extraction - package `feats`:
- `LBP.m`: this function takes an image and outputs cell-structured LBP features
    - INPUT:
        - Required: img -> greyscale image, RxC
        - Optional:
            - 1. cellsize: 2x2 array, size of cells, default = [16,16]
            - 2. filt: parameter for filtering, default = 0 (no filtering)
            - 3. rot: whether to implement rotation invariance default = 0 (false)

    - OUTPUT: (R/cellsize)x(C/cellsize)x59 matrix of LBP features (or RxCx10 for rot. invariant)
    - EXAMPLE: `LBP_feature_matrix = feats.LBP(img)`
    - NOTES: change to 2D matrix: `features = reshape(features, [R*C, 59])`

- `Sift_features.m`: this function takes an image and outputs dense SIFT features
    - INPUT:
        - Gray level image
    - OUTPUT:
        - (R/blocksize)x(C/blocksize)x128 SIFT feature
    - EXAMPLE: `sift_feature_matrix = feats.sift_features(img)`

Dictionary learning - package `dict`:

- `learn_dictionary.m`
    - INPUT:
        - Features: complete set of features extracted from the input images (the feature space)
        - Training_set_size: how many features from the feature space should be used in training the dictionary
        - Dictionary_size: the size of the dictionary
        - Iterations: How many iterations the dictionary optimization will take, at maximum.
        - Lambda: sparsity penalty for the weights
    - PROCESS:
        - Flattens the features, randomly selects training_set_size features, and learns the dictionary using sparse coding. Sparse coding alternately optimizes the weights using `optimize_weights` (which calls `lasso`), and the dictionary using `optimize_dictionary` (which calls `lagrange_dual`).
    - OUTPUT:
        - The learned Dictionary
    - EXAMPLE: `dictionary = dict.learn_dictionary(features, training_set_size, dictionary_size, dictionary_iterations, lambda);`

Sparse coding:

- `spatial_pyramid_matching.m`
    - INPUT:
        - Dictionary: the dictionary learned by `learn_dictionary.m`
        - Feature_descriptors: the set of feature descriptors associated with a single image [RxC x d] where R = rows, C = columns, and d = length of a single feature vector.

        - Lambda: sparsity penalty for the weights
    - PROCESS:
        - At spatial scale 1, weights are calculated for all feature descriptors in the image, then max-pooled (the maximum value for each row of the dictionary is taken, such that the matrix of weights is condensed into one vector).
        - At spatial scale 2, the set of feature descriptors is divided into quarters based on their spatial location in the image they came from, and the process from spatial scale 1 is repeated for each quarter.
        - Spatial scale 3 is the same as 2, but the feature descriptors are divided into sixteenths.
    - OUTPUT:
        - Descriptor
    - EXAMPLE: `image_vector = dict.spatial_pyramid_matching(dictionary, image_features, lambda);`

Adaptive Boosting - package `boost`:

- `ada_prep.m`: formats image vectors to pass in to `ada_train.m` or `ada_predict.m`
    - INPUT: LBP and SIFT image vectors from spatial pyramid matching, class labels
    - OUTPUT: ada_data, cell array of features as below
    - EXAMPLE: `ada_data = boost.ada_prep(LBP_img_vectors, SIFT_image_vectors, Y_train)`
    - NOTES: Pass in zeros(size(LBP_img_vectors, 1),1) as class labels if formatting for `ada_predict.m`

	feature_1	feature_2	...	feature_n	class_labels
sample 1	1xN1	1xN2	...	1xNn	+1/-1
sample 2	1xN1	1xN2	...	1xNn	+1/-1
...	...	...	...	...	...
sample M	1xN1	1xN2	...	1xNn	+1/-1

- `ada_train.m`: trains a classification model using the Adaboost algorithm, with SVMs as weak learners
    - INPUT:
        - training_set: cell array of feature vectors as formatted in ada_prep.m
        - mode: "labels" (1/0) or "scores" (floating pt)
    - OUTPUT:
        - ada_labels: predicted labels for training set, 1/0 for “labels” or probabilities for “scores”
        - Model: final classifier, cell array containing:
            - h_model: intermediate classifiers, set of (N feature types)x(T trials) SVMs

        - h_weights: NxT weights for SVMs in h_model
        - alpha: T weights for intermediate classifiers h_model(:,1:T)
    - EXAMPLE: `[labels, model]= boost.ada_train(ada_data, “labels”)`

- `ada_predict.m`: predict class labels using Adaboost model
    - INPUT: model as from `ada_train.m`
    - OUTPUT: (M samples)x1 column arrays of predicted class labels and probabilities
    - EXAMPLE: `[labels, scores] = boost.ada_predict(model, ada_data)`

- `ada_eval.m`: train and evaluate the performance of an Adaboost classifier, used mostly for testing before integrating with `classification.m`
    - INPUT: training and testing sets for LBP and SIFT, along with class labels for training and testing sets
    - OUTPUT: precision and recall scores for the trained model on the test samples
    - EXAMPLE: `[precision, recall] = boost.ada_eval(LBP_train, LBP_test, SIFT_train, SIFT_test, test_labels, train_labels, mode)`