Image Retrieval Using Customized Bag of Features

Image retrieval systems use the visual content of an input image to find similar or related images in a database, and are used in image search tasks in applications such as online shopping. Computer Vision System Toolbox™ provides algorithms and tools to extract discriminative information from images as well as mechanisms to find similar images in a dataset.

This example shows how to create a content-based image retrieval (CBIR) system using a customized bag-of-features workflow.

Introduction

Content-based image retrieval systems are used to find images that are visually similar to a query image. The application of CBIR systems can be found in web-based product search, surveillance, visual place identification, and other areas. A common technique used to implement a CBIR system is bag of visual words, also known as *bag of features*[1,2]. The bag-of-features technique was adapted to image retrieval from the world of document retrieval. Instead of using actual words as in document retrieval, bag of features uses image features as the visual words that describe an image.

Image features are an important part of CBIR systems. These image features are used to gauge similarity between images and can include global image features such as color, texture, and shape. Image features can also be local image features such as speeded up robust features (SURF), histogram of gradients (HOG), or local binary patterns (LBP). The benefit of the bag-of-features approach is that the type of features used to create the visual word vocabulary can be customized to fit the application.

The speed and efficiency of image search is also important in CBIR systems. For example, it may be acceptable to perform a brute force search in a small collection of images of less than a 100 images, where features from the query image are compared with features from each image in the collection. For larger collections, a brute force search is not feasible and more efficient search techniques must be used. The bag of features provides a concise encoding scheme to represent a large collection of images using a sparse set of visual word histograms. This approach enables compact storage and efficient search through an inverted index data structure.

Computer Vision System Toolbox provides a customizable bag-of-features framework to implement an image retrieval system using the following steps:

1. Select the image features for retrieval
2. Create a bag of features
3. Index the images
4. Search for similar images

In this example, you will see how to create an image retrieval system for searching the 17-Category Flower Dataset[3]. This dataset contains about 1300 images of 17 different types of flowers.

You can get started by downloading this dataset for use in the rest of this example.

```
% Location of the compressed data set
url = 'http://www.robots.ox.ac.uk/~vgg/data/flowers/17/17flowers.tgz';

% Store the output in a temporary folder
outputFolder = fullfile(tempdir, '17Flowers'); % define output folder
```

Note that downloading the dataset from the web can take a very long time depending on your Internet connection. The commands below will block MATLAB® for that period of time. Alternatively, you can use your web browser to first download the set to your local disk. If you choose that route, repoint the 'url' variable above to the file that you downloaded.

```
if ~exist(outputFolder, 'dir') % download only once
    disp('Downloading 17-Category Flower Dataset (58 MB)...');
    untar(url, outputFolder);
end

flowerImageSet = imageSet(fullfile(outputFolder,'jpg'));

% Total number of images in the data set
flowerImageSet.Count
ans =
         1360
```

Step 1: Selecting the Image Features for Retrieval

The type of feature used for retrieval depends on the type of images within the collection. For example, if searching an image collection made up of scenes (beaches, cities, highways), it is preferable to use a global image feature, such as a color histogram that captures the color content of the entire scene. However, if the goal is to find specific objects within the image collections, then local image features extracted around object keypoints are a better choice.

You can start by viewing a few of the images to get an idea of how to approach the problem.

```
% Display a few of the flower images
helperDisplayImageMontage(flowerImageSet.ImageLocation(1:50:1000));
```

In this example, the goal is to search for similar flowers in the dataset using the color information in the query image. The images in the dataset contain one kind of flower in every image. Therefore, a simple image feature based on the spatial layout of color is a good place to start.

The following function describes the algorithm used to extract color features from a given image. This function will be used as a custom feature extractor within `bagOfFeatures` to extract color features.

```
type exampleBagOfFeaturesColorExtractor.m
function [features, metrics] = exampleBagOfFeaturesColorExtractor(I)
% Example color layout feature extractor. Designed for use with
% bagOfFeatures.
```

```
%
% Local color layout features are extracted from truecolor image, I and
% returned in features. The strength of the features are returned in
% metrics.

[~,~,P] = size(I);

isColorImage = P == 3;

if isColorImage

    % Convert RGB images to the L*a*b* colorspace. The L*a*b* colorspace
    % enables you to easily quantify the visual differences between
    % colors. Visually similar colors in the L*a*b* colorspace will have
    % small differences in their L*a*b* values.
    Ilab = rgb2lab(I);

    % Compute the "average" L*a*b* color within 16-by-16 pixel blocks. The
    % average value is used as the color portion of the image feature. An
    % efficient method to approximate this averaging procedure over
    % 16-by-16 pixel blocks is to reduce the size of the image by a factor
    % of 16 using IMRESIZE.
    Ilab = imresize(Ilab, 1/16);

    % Note, the average pixel value in a block can also be computed  using
    % standard block processing or integral images.

    % Reshape L*a*b* image into "number of features"-by-3 matrix.
    [Mr,Nr,~] = size(Ilab);
    colorFeatures = reshape(Ilab, Mr*Nr, []);
```

```
        % L2 normalize color features
        rowNorm = sqrt(sum(colorFeatures.^2,2));
        colorFeatures = bsxfun(@rdivide, colorFeatures, rowNorm + eps);

        % Augment the color feature by appending the [x y] location within the
        % image from which the color feature was extracted. This technique is
        % known as spatial augmentation. Spatial augmentation incorporates the
        % spatial layout of the features within an image as part of the
        % extracted feature vectors. Therefore, for two images to have similar
        % color features, the color and spatial distribution of color must be
        % similar.

        % Normalize pixel coordinates to handle different image sizes.
        xnorm = linspace(-0.5, 0.5, Nr);
        ynorm = linspace(-0.5, 0.5, Mr);
        [x, y] = meshgrid(xnorm, ynorm);

        % Concatenate the spatial locations and color features.
        features = [colorFeatures y(:) x(:)];

        % Use color variance as feature metric.
        metrics  = var(colorFeatures(:,1:3),0,2);
    else

        % Return empty features for non-color images. These features are
        % ignored by bagOfFeatures.
        features = zeros(0,5);
        metrics  = zeros(0,1);
    end
```

Step 2: Creating a Bag Of Features

With the feature type defined, the next step is to learn the visual vocabulary within the `bagOfFeatures` using a set of training images. The code shown below picks a random subset of images from the dataset for training and then trains `bagOfFeatures` using the `'CustomExtractor'` option.

```
% Pick a random subset of the flower images
% trainingSet = partition(flowerImageSet, 0.4, 'randomized');
%
% Create a custom bag of features using the 'CustomExtractor' option
% colorBag = bagOfFeatures(trainingSet, ...
%    'CustomExtractor', @exampleBagOfFeaturesColorExtractor, ...
%    'VocabularySize', 10000);
```

The previous code is commented out because the training process takes several minutes. The rest of the example uses a pretrained `bagOfFeatures` to save time. If you wish to recreate `colorBag` locally, consider enabling parallel computing to reduce processing time.

```
% Load pre-trained bagOfFeatures
load('savedColorBagOfFeatures.mat','colorBag');
```

Step 3: Indexing the Images

Now that the `bagOfFeatures` is created, the entire flower image set can be indexed for search. The indexing procedure extracts features from each image using the custom extractor function from step 1. The extracted features are encoded into a visual word histogram and added into the image index.

```
% Create a search index
% flowerImageIndex = indexImages(flowerImageSet, colorBag, ...
'SaveFeatureLocations', false);
```

Because the indexing step processes thousands of images, the rest of this example uses a saved index to save time. You may recreate the index locally by running the code shown above. Consider enabling parallel computing to reduce processing time.

```
% Load the pre-saved index
load('savedColorBagOfFeatures.mat', 'flowerImageIndex');
```

Step 4: Searching for Similar Images

The final step is to use the retrieveImages function to search for similar images.

```
% Define a query image
queryImage = read(flowerImageSet, 502);

figure
imshow(queryImage)

% Search for the top 20 images with similar color content
[imageIDs, scores] = retrieveImages(queryImage, flowerImageIndex);
```

`retrieveImages` returns the image IDs and the scores of each result. The scores are sorted from best to worst.

```
scores
```

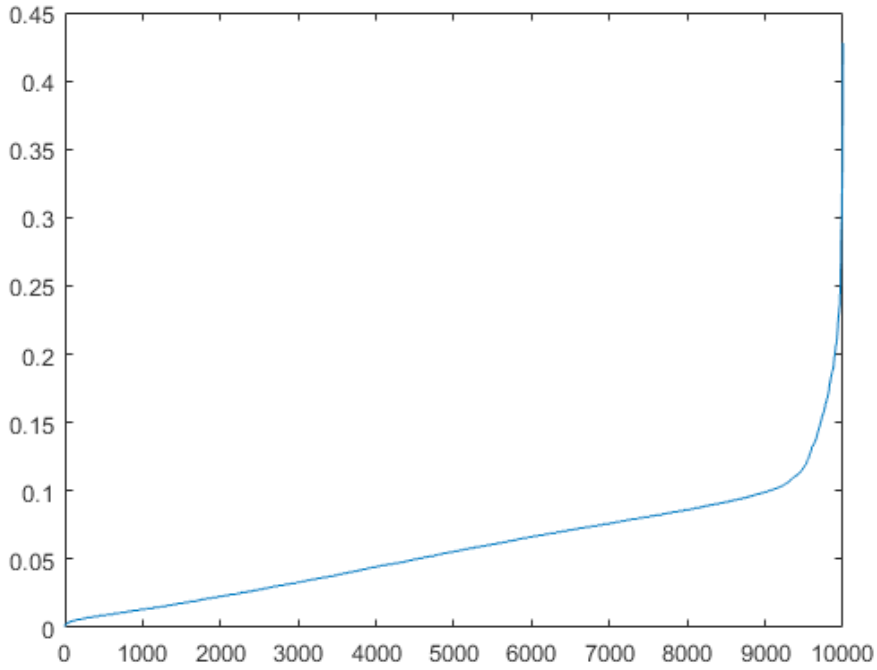
```
scores =

    0.9063
    0.2839
    0.2811
    0.2626
    0.2437
    0.2427
    0.2387
    0.2366
    0.2365
    0.2170
    0.2168
    0.2130
    0.2129
    0.2101
    0.2056
    0.2050
    0.2040
    0.2018
    0.1999
    0.1995
```

The imageIDs correspond to the images within the image set that are similar to the query image.

```
% Display results using montage. Resize images to thumbnails first.
helperDisplayImageMontage(flowerImageSet.ImageLocation(imageIDs))
```

`flowerImageIndex` contains several index statistics that are relevant to search. One of these is the `'WordFrequency'` property, which contains the percentage of images in which each visual word occurs. This property shows you which words are more common and which are rare across the entire dataset. It is often helpful to suppress the most common words as these do not help us reduce the search set when looking for the most relevant images. Conversely, it is also helpful to suppress very rare words as they may be coming from outliers in the image set. You can control how much the upper and lower ends of the visual word distribution affect the search results by tuning the `'WordFrequencyRange'` property. A good way to set this value is to plot the sorted `WordFrequency` values.

```
figure
plot(sort(flowerImageIndex.WordFrequency))
```



The plot shows that the distribution of visual words reaches its peak at around 45%. This means that only a few visual words are in 45% of the images. If the upper part of the distribution is at 80% or 90%, then it's best to exclude those via the `'WordFrequencyRange'` property, and run the search again. For example, try lowering the upper range to 20% and checking the effects on the search results.

```
% Lower WordFrequencyRange
flowerImageIndex.WordFrequencyRange = [0.01 0.2];
% Re-run retrieval
[imageIDs, scores] = retrieveImages(queryImage, flowerImageIndex);
% Show results
helperDisplayImageMontage(flowerImageSet.ImageLocation(imageIDs))
```

In this case, because the upper range is near 45%, a setting of 20% is going to prevent a lot of relevant matches from appearing in the search results. This is confirmed by the poor search results.

Conclusion

This example showed you how to customize the `bagOfFeatures` and how to use `indexImages` and `retrieveImages` to create an image retrieval system based on color features. The techniques shown here may be extended to other feature types by further customizing the features used within `bagOfFeatures`.

References

[1] Sivic, J., Zisserman, A. "Video Google: A text retrieval approach to object matching in videos," IEEE International Conference on Computer Vision, 2003.

[2] Philbin, J., Chum, O., Isard, M., Sivic, J., Zisserman, A. "Object retrieval with large vocabularies and fast spatial matching," IEEE Conference on Computer Vision and Pattern Recognition, 2007.

[3] Nilsback, M-E. and Zisserman, A. "A Visual Vocabulary for Flower Classification," IEEE Conference on Computer Vision and Pattern Recognition, 2006.

Learn More About Image Processing

- *Computer Vision System Toolbox Overview* 2:13
- *Computer Vision System Toolbox* (product trial)