Image Category Classification Using Bag of Features

Image classification, or object recognition, is the process of identifying a specific object or class of objects in an image or video. Computer Vision System Toolbox™ offers a variety of algorithms, tools, and techniques to create image classification and object recognition systems.

This example shows how to use a "bag of features" approach for image category classification. This technique is also often referred to as *bag of words*. Visual image categorization is a process of assigning a category label to an image under test. Categories may contain images representing just about anything, for example, dogs, cats, trains, or boats.

Downloading the Caltech101 Image Set

To get started with bag-of-features image category classification, you can download a suitable image data set. One of the most widely cited and used data sets is ***Caltech 101***, collected by Fei-Fei Li, Marco Andreetto, and Marc 'Aurelio Ranzato.

```
% Location of the compressed data set
url = 'http://www.vision.caltech.edu/Image _ Datasets/Caltech101/101 _ ...
ObjectCategories.tar.gz';
% Store the output in a temporary folder
outputFolder = fullfile(tempdir, 'caltech101'); % define output folder
```

Note that downloading the 126MB set from the web can take 30 minutes or more 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 126MB Caltech101 data set...');
    untar(url, outputFolder);
end
```

Loading the Image Sets

Instead of operating on the entire Caltech 101 set, which can be time consuming, you can use three categories: airplanes, ferry, and laptop. Note that for the bag-of-features approach to be effective, the majority of each image's area must be occupied by the subject of the category, for example, an object or a type of scene.

```
rootFolder = fullfile(outputFolder, '101_ObjectCategories');
```

Next, you can construct an array of image sets based on the following categories from Caltech 101: `'airplanes'`, `'ferry'`, and `'laptop'`. To manage the data, you can use the `imageSet` class. Since `imageSet` operates on image file locations, and therefore does not load all the images into memory, it is safe to use on large image collections.

```
imgSets = [ imageSet(fullfile(rootFolder, 'airplanes')), ...
            imageSet(fullfile(rootFolder, 'ferry')), ...
            imageSet(fullfile(rootFolder, 'laptop')) ];
```

Each element of the `imgSets` variable now contains images associated with the particular category. You can easily inspect the number of images per category as well as category labels as shown below:

```
{ imgSets.Description } % display all labels on one line
[imgSets.Count]          % show the corresponding count of images
ans =

    'airplanes'    'ferry'    'laptop'

ans =

   800    67    81
```

Note that the labels were derived from directory names used to construct the image sets, but can be customized by manually setting the Description property of the `imageSet` object.

Preparing the Training and Validation Image Sets

`imgSets` contains an unequal number of images per category. You can adjust it to balance the number of images in the training set.

```
minSetCount = min([imgSets.Count]); % determine the smallest amount of
images in a category


% Use partition method to trim the set.
imgSets = partition(imgSets, minSetCount, 'randomize');


% Notice that each set now has exactly the same number of images.
[imgSets.Count]
ans =


     67     67     67
```

You can then separate the sets into training and validation data, using 30% of images from each set for the training data and the remainder, 70%, for the validation data. To avoid biasing the results, you can randomize the split.

```
[trainingSets, validationSets] = partition(imgSets, 0.3, 'randomize');
```

The above call returns two arrays of `imageSet` objects ready for training and validation tasks. Below, you can see example images from the three categories included in the training data.

```
airplanes = read(trainingSets(1),1);
ferry     = read(trainingSets(2),1);
laptop    = read(trainingSets(3),1);


figure
```

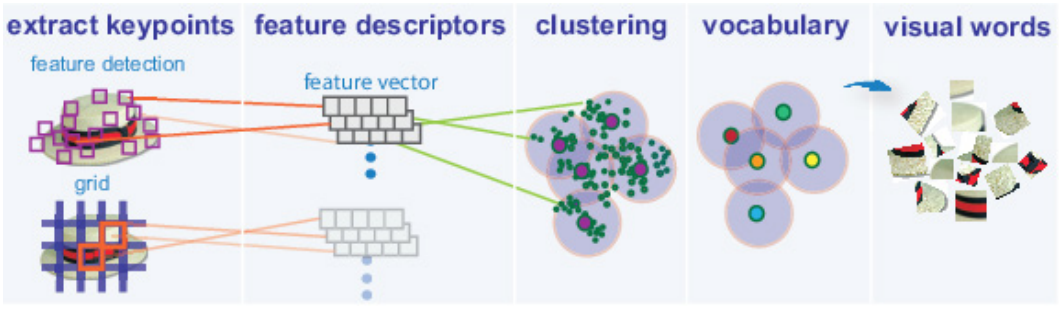
```
subplot(1,3,1);
imshow(airplanes)
subplot(1,3,2);
imshow(ferry)
subplot(1,3,3);
imshow(laptop)
```

Creating a Visual Vocabulary and Training an Image Category Classifier

The bag-of-words technique was adapted to computer vision from the world of natural language processing. Since images do not actually contain discrete words, you first construct a "vocabulary" of visual words by extracting feature descriptors from representative images of each category.

Feature extraction is a type of dimensionality reduction that efficiently represents interesting parts of an image as a compact feature vector. You can use a speeded-up robust features (SURF) detector to find interesting points in the images and encode information about the area around the points as a feature vector.

You can extract features based on a feature detector like SURF, or you can define a regularly spaced grid to extract feature descriptors. The grid method may lose fine-grained scale information, so it is best used onlyfor images that do not contain distinct features, such as an image containing scenery, like the beach. Using a SURF detector also provides greater scale invariance. By default, the algorithm runs the `'grid'` method.



Extracting visual words from training images.

This is accomplished with a single call to the `bagOfFeatures` function, which:

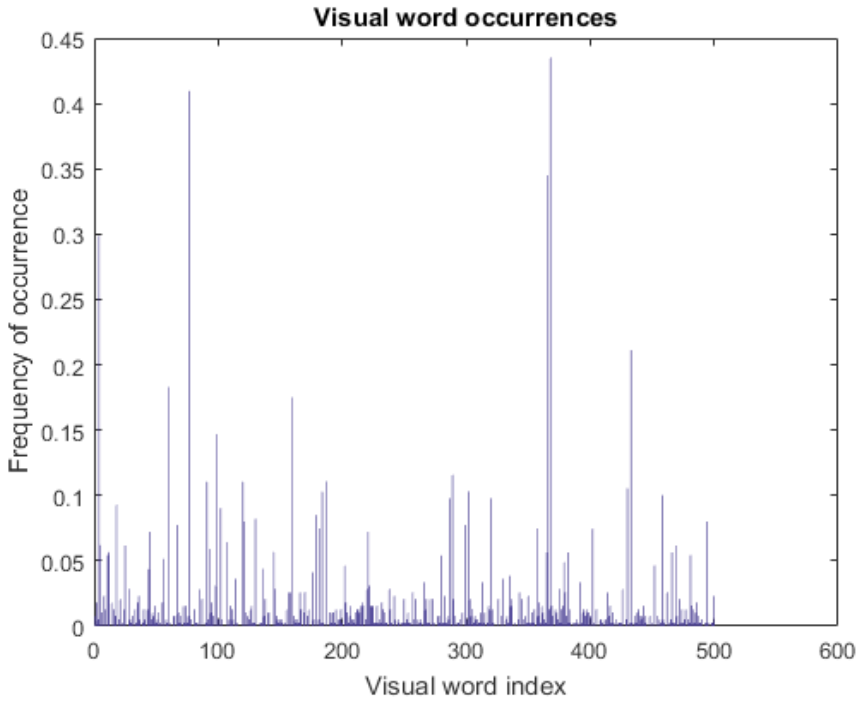
1. Extracts SURF features from all images in all image categories
2. Constructs the visual vocabulary by reducing the number of features through quantization of feature space using K-means clustering

```
bag = bagOfFeatures(trainingSets);
Creating Bag-Of-Features from 3 image sets.
----------------------------------------
* Image set 1: airplanes.
* Image set 2: ferry.
* Image set 3: laptop.

* Selecting feature point locations using the Grid method.
* Extracting SURF features from the selected feature point locations.
** The GridStep is [8 8] and the BlockWidth is [32 64 96 128].

* Extracting features from 20 images in image set 1...done. Extracted
81684 features.
* Extracting features from 20 images in image set 2...done. Extracted
70832 features.
* Extracting features from 20 images in image set 3...done. Extracted
98344 features.

* Keeping 80 percent of the strongest features from each image set.

* Balancing the number of features across all image sets to improve
clustering.
** Image set 2 has the least number of strongest features: 56666.
** Using the strongest 56666 features from each of the other image sets.

* Using K-Means clustering to create a 500 word visual vocabulary.
* Number of features          : 169998
```

```
* Number of clusters (K)       : 500

* Initializing cluster centers...100.00%.
* Clustering...completed 31/100 iterations (~0.42 seconds/iteration)...converged in 31 iterations.

* Finished creating Bag-Of-Features
```

Additionally, the `bagOfFeatures` object provides an encode method for counting the visual word occurrences in an image. It produced a histogram that becomes a new and reduced representation of an image.

```
img = read(imgSets(1), 1);
featureVector = encode(bag, img);

% Plot the histogram of visual word occurrences
figure
bar(featureVector)
title('Visual word occurrences')
xlabel('Visual word index')
ylabel('Frequency of occurrence')
```



This histogram forms the basis for training a classifier and for the actual image classification. In essence, it encodes an image into a feature vector.

Encoded training images from each category are fed into a classifier training process invoked by the `trainImageCategoryClassifier` function. Note that this function relies on the multiclass linear SVM classifier from Statistics and Machine Learning Toolbox™.

```
categoryClassifier = trainImageCategoryClassifier(trainingSets, bag);
Training an image category classifier for 3 categories.
--------------------------------------------------------
* Category 1: airplanes
* Category 2: ferry
* Category 3: laptop

* Encoding features for category 1...done.
* Encoding features for category 2...done.
* Encoding features for category 3...done.
```

```
* Finished training the category classifier. Use evaluate to test the
classifier on a test set.
```

The above function utilizes the encode method of the input bag object to formulate feature vectors representing each image category from the `trainingSets` array of `imageSet` objects.

Evaluating Classifier Performance

Now that you have a trained classifier, `categoryClassifier`, you can evaluate it. First, test it with the training set, which should produce a near-perfect confusion matrix, that is, with ones on the diagonal.

```
confMatrix = evaluate(categoryClassifier, trainingSets);
Evaluating image category classifier for 3 categories.
-------------------------------------------------------


* Category 1: airplanes
* Category 2: ferry
* Category 3: laptop

* Evaluating 20 images from category 1...done.
* Evaluating 20 images from category 2...done.
* Evaluating 20 images from category 3...done.

* Finished evaluating all the test sets.

* The confusion matrix for this test set is:


                          PREDICTED
KNOWN          | airplanes    ferry    laptop
-----------------------------------------------
airplanes      | 0.90          0.10     0.00
ferry          | 0.00          1.00     0.00
laptop         | 0.00          0.00     1.00
```

```
* Average Accuracy is 0.97.
```

Next, you can evaluate the classifier on the `validationSet`, which was not used during the training. By default, the evaluate function returns the confusion matrix, which is a good initial indicator of how well the classifier is performing.

```
confMatrix = evaluate(categoryClassifier, validationSets);

% Compute average accuracy
mean(diag(confMatrix));
Evaluating image category classifier for 3 categories.
-------------------------------------------------------

* Category 1: airplanes
* Category 2: ferry
* Category 3: laptop

* Evaluating 47 images from category 1...done.
* Evaluating 47 images from category 2...done.
* Evaluating 47 images from category 3...done.

* Finished evaluating all the test sets.

* The confusion matrix for this test set is:


                          PREDICTED
KNOWN          | airplanes    ferry    laptop
-----------------------------------------------
airplanes      | 0.77          0.15     0.09
ferry          | 0.04          0.91     0.04
laptop         | 0.09          0.00     0.91
```

```
* Average Accuracy is 0.87.
```

You can derive additional statistics using the rest of the arguments returned by the `evaluate` function. You can tweak the various parameters and continue evaluating the trained classifier until you are satisfied with the results.

Trying the Newly Trained Classifier on Test Images

You can now apply the newly trained classifier to categorize new images.

```
img = imread(fullfile(rootFolder, 'airplanes', 'image_0690.jpg'));
[labelIdx, scores] = predict(categoryClassifier, img);

% Display the string label
categoryClassifier.Labels(labelIdx)
ans =

    'airplanes'
```

Learn More About Image Processing

- *Computer Vision System Toolbox Overview* 2:13
- *Computer Vision Made Easy* 35:43
- *Computer Vision with MATLAB for Object Detection and Tracking* 46:56
- *Computer Vision System Toolbox* (product trial)