Performing Texture Segmentation Using Gabor Filters

Image segmentation can help you select regions of interest in an image. Image Processing Toolbox™ offers a variety of techniques for image segmentation.

This example shows how to use texture segmentation to identify regions of an image based on their texture—in this case, to segment the dog from the bathroom floor. The segmentation is visually obvious because of the difference in texture between the regular, periodic pattern of the bathroom floor, and the regular, smooth texture of the dog's fur.

From experimentation, it is known that Gabor filters are a reasonable model of simple cells in the mammalian vision system. Because of this, Gabor filters are thought to be a good model of how humans distinguish texture, and are, therefore, a good model to use when designing algorithms to recognize texture. This example uses the basic approach described in "Unsupervised Texture Segmentation Using Gabor Filters" (A. K. Jain and F. Farrokhnia, 1991) to perform texture segmentation.

Reading and Displaying the Input Image

Suppose you have an image of a dog, with the filename kobi.png. You can get started by reading the image into MATLAB® and displaying it. You can use the function `imresize` to shrink the image to make the example run more quickly.

```
A = imread('kobi.png');
A = imresize(A,0.25);
Agray = rgb2gray(A);
figure
imshow(A)
```

Designing an Array of Gabor Filters

You can design an array of Gabor filters that are tuned to different frequencies and orientations. The set of frequencies and orientations is designed to localize different, roughly orthogonal, subsets of frequency and orientation information in the input image. To start, try regularly sampling orientations between 0 and 150 degrees in steps of 30 degrees, and sampling wavelength in increasing powers of two starting from 4/sqrt(2) up to the hypotenuse length of the input image. These combinations of frequency and orientation are taken from the paper cited in the introduction.

```
imageSize = size(A);
numRows = imageSize(1);
numCols = imageSize(2);

wavelengthMin = 4/sqrt(2);
wavelengthMax = hypot(numRows,numCols);
n = floor(log2(wavelengthMax/wavelengthMin));
wavelength = 2.^(0:(n-2)) * wavelengthMin;

deltaTheta = 45;
orientation = 0:deltaTheta:(180-deltaTheta);

g = gabor(wavelength,orientation);
```

Then, you can extract `gabor` magnitude features from the source image. It is common to work with the magnitude response of each Gabor filter. Gabor magnitude response is also sometimes referred to as "Gabor energy." Each MxN Gabor magnitude output image in `gabormag(:,:,ind)` is the output of the corresponding Gabor filter `g(ind)`.

```
gabormag = imgaborfilt(Agray,g);
```

Postprocessing the Gabor Magnitude Images into Gabor Features

To use Gabor magnitude responses as features for use in classification, some postprocessing is required. This postprocessing includes performing Gaussian smoothing, adding additional spatial information to the feature set, reshaping the feature set to the form expected by the `pca` and `kmeans` functions, and normalizing the feature information to a common variance and mean.

Each Gabor magnitude image contains some local variations, even within well-segmented regions of constant texture. These local variations will throw off the segmentation. You can compensate for these variations using simple Gaussian low-pass filtering to smooth the Gabor magnitude information. To do this, choose a sigma that is matched to the Gabor filter that extracted each feature. You can introduce a smoothing term `K` that controls how much smoothing is applied to the Gabor magnitude responses.

```
for i = 1:length(g)
    sigma = 0.5*g(i).Wavelength;
    K = 3;
    gabormag(:,:,i) = imgaussfilt(gabormag(:,:,i),K*sigma);
end
```

When constructing Gabor feature sets for classification, it is useful to add a map of spatial location information in both `X` and `Y`. This additional information allows the classifier to prefer groupings that are close together spatially.

```
X = 1:numCols;
Y = 1:numRows;
[X,Y] = meshgrid(X,Y);
featureSet = cat(3,gabormag,X);
featureSet = cat(3,featureSet,Y);
```

You can then reshape data into a matrix `X` of the form expected by the `kmeans` function. Each pixel in the image grid is a separate data point, and each plane in the variable `featureSet` is a separate feature. In this example, there is a separate feature for each filter in the Gabor filter bank, plus two additional features from the spatial information that was added in the previous step. In total, there are 24 Gabor features and two spatial features for each pixel in the input image.

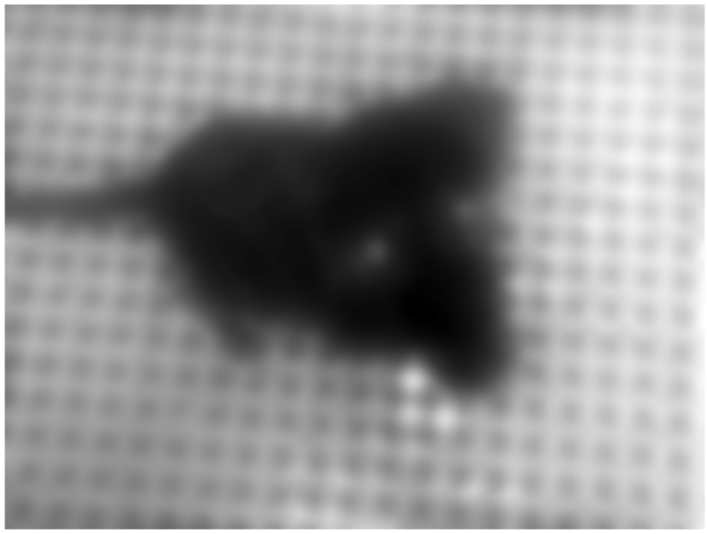
```
numPoints = numRows*numCols;
X = reshape(featureSet,numRows*numCols,[]);
```

Next, you can normalize the features to be zero mean, unit variance.

```
X = bsxfun(@minus, X, mean(X));
X = bsxfun(@rdivide,X,std(X));
```

You can then visualize the feature set. To get a sense of what the Gabor magnitude features look like, you can use principal component analysis to move from a 26D representation of each pixel in the input image into a 1D intensity value for each pixel.

```
coeff = pca(X);
feature2DImage = reshape(X*coeff(:,1),numRows,numCols);
figure
imshow(feature2DImage,[])
```



It is apparent in this visualization that there is sufficient variance in the Gabor feature information to obtain a good segmentation for this image. The dog is very dark compared with the floor because of the texture differences between the dog and the floor.

Classifying Gabor Texture Features Using `kmeans`

You can avoid local minima when searching for means that minimize objective function by repeating k-means clustering five times. The only prior information assumed in this example is how many distinct regions of texture are present in the image being segmented. In this case, there are two distinct regions. This part of the example uses Statistics and Machine Learning Toolbox™.

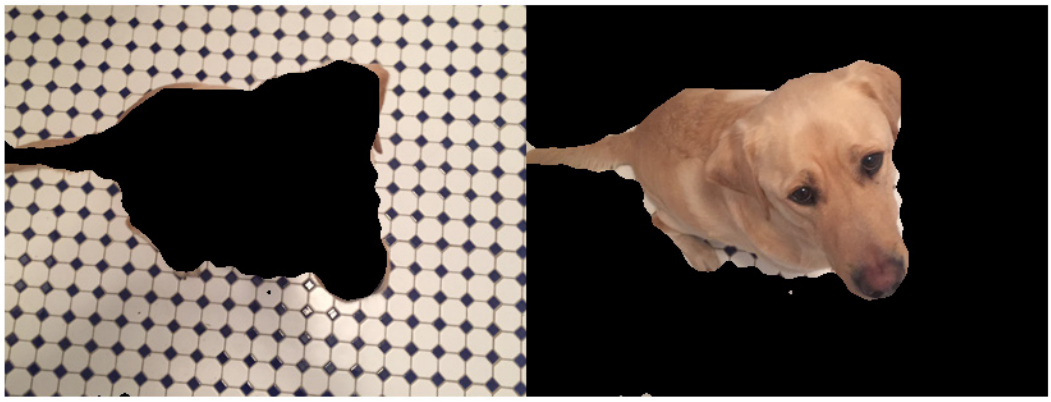
```
L = kmeans(X,2,'Replicates',5);
```

After performing the k-means clustering, you can visualize the segmentation using `label2rgb`.

```
L = reshape(L,[numRows numCols]);
figure
imshow(label2rgb(L))
```

You can then visualize the segmented image using `imshowpair`, and examine the foreground and background images that result from the mask `BW` that is associated with the label matrix `L`.

```
Aseg1 = zeros(size(A),'like',A);
Aseg2 = zeros(size(A),'like',A);
BW = L == 2;
BW = repmat(BW,[1 1 3]);
Aseg1(BW) = A(BW);
Aseg2(~BW) = A(~BW);
figure
imshowpair(Aseg1,Aseg2,'montage');
```



Learn More About Image Processing

- *Computer Vision with MATLAB* 43:32
- *Developing and Deploying Sonar and Echosounder Data Analysis Software* (technical article)
- *Creating Computer Vision and Machine Learning Algorithms that Can Analyze Works of Art* (technical article)
- *Image Processing Toolbox* (product trial)