1. Texture Classification

1.1 Goals

In this exercise, you will learn how to:

- Create Gabor filters bank and use them to extract features from textures;
- Create FFT texture filters;
- Create texture filters based on co-occurrence matrices
- Classify features using Gabor, FFT and co-occurrence based features.

1.2 Exercise 1

Gabor filter is usually used for edge detection and as such it was found to be very useful for texture representation and discrimination. Frequency and orientation components make Gabor filters similar to the receptive field of cells in human visual system. They can be thought of as filters with Gaussian kernel modulated by a sinusoid.

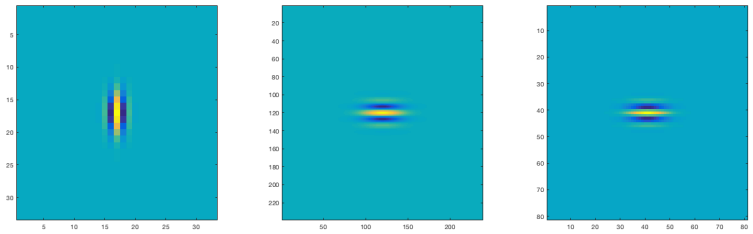

Figure 1.1: Examples of Gabor filters of different frequency and orientation.

In this example, we will see how to classify textures using Gabor filter banks. In order to do that, we have to filter the images using the real parts of different kernels. We will then use mean and variance of the resulting images to construct features necessary for classification.

Exercise 1.1 Creating filter bank

Implement the function `filters = CreateFilterBank(wavelengths, ... angles)`, that takes as inputs 2 vectors of wavelengths and corresponding angles for each Gabor filter. The output filters should be an array of *real* parts of 2D Gabor filters. *Hint:* You can use Matlab function `gabor` and its `SpatialKernel` property and function `real`. The wavelengths and the angles are specified in the base code.

At this point, you should be able to visualize the filters like in Fig. 1. ■

Once we created a set of filters, we can use them to extract features from different images.

Exercise 1.2 Extracting features

Create a function `features = ComputeFeatures(image, kernels)`, that takes as inputs a grayscale image and a set of kernels. It should convolve the image with each filter, compute mean and variance of the resulting image and store it in a

vector `features`. ■

Finally, we can find the closest neighbour of each testing image in the template set.

Exercise 1.3 Texture matching
Create a function `label = MatchTextures(imageFeatures, templateFeatures)` that takes as inputs extracted features of the testing image and features of template images. It should assign the testing image to the class to which it is the closest in feature space, as measured by the Euclidean distance. ■

1.3 Exercise 2

In this exercise you will use properties of the Discrete Fourier Transform (DFT) to characterize texture. It is known that the magnitude of the DFT captures the main orientations in the image. Furthermore, angular and radial bins in the Fourier domain capture the directionality and rapidity of fluctuation of an image texture.

In this example, we will see how to classify textures using the DFT magnitude of an image. In order to do that, we have to apply the DFT on the images. We will use the resulting magnitude of the DFT as features for texture classification.

Exercise 1.4 Computing DFT magnitude
Implement a simple function `imageFeatures = computeDFTMag(image)` that computes the magnitudes of the DFT of an input grayscale image. The 2-d DFT can be computed by calling the built-in MATLAB function `fft2`. ■

The magnitudes computed from the previous exercise will serve as features for texture classification.

Exercise 1.5 Texture matching
Use the previously created function
`label = MatchTextures(imageFeatures, templateFeatures)`
that takes as inputs extracted features (in this exercise, DFT magnitudes) of the testing image and features of template images. It should assign the testing image to the closest template class in feature space, as measured by the Euclidean distance. ■

1.4 Exercise 3

In this exercise you will use second-order gray-level statistics to characterize texture. We will see how to classify textures using the histogram of the co-occurrence of intensity values in the image. The co-occurence matrix of a gray-scale image is the distribution of co-occurring values at a given offset. Mathematically, a co-occurrence matrix $\mathbf{C}$ is defined over an $n \times m$ image $\mathbf{I}$, parameterized by an offset $(\Delta x, \Delta y)$, as:

$$C(i, j, \Delta x, \Delta y) = \sum_{p=1}^{n} \sum_{q=1}^{m} \begin{cases} 1, & I(p, q) = i \text{ and } I(p + \Delta x, q + \Delta y) = j \\ 0, & \text{else} \end{cases} \tag{1.1}$$

From the co-occurrence matrix we can compute several features for texture classification. In this exercise you will implement the following features:

1. Contrast: $\sum_{i,j} C_{i,j}(i - j)^2$

2. Dissimilarity: $\sum_{i,j} C_{i,j}|i-j|$
3. Homogeneity: $\sum_{i,j} \frac{C_{i,j}}{1+(i-j)^2}$
4. Energy: $\sum_{i,j} C_{i,j}^2$
5. Entropy: $-\sum_{i,j} C_{i,j} \log C_{i,j}$

Exercise 1.6 Computing co-occurrence matrix

Implement a function `cooccurFeats = computeCooccurFeatures(image,offset)` that computes the co-occurence matrix features of an input grayscale image with the given offset. You can use the built-in MATLAB function `graycomatrix` to compute the co-occurrence matrix of a gray-scale image. Investigate the different input arguments of the function. *Hint* Investigate the effects of different offsets. ■

The above computed features will be used for texture matching.

Exercise 1.7 Texture matching

Use the previously created function `label = MatchTexturesCoOccur(imageFeatures, templateFeatures)` that takes as inputs extracted features (in this exercise, co-occurrence matrix features) of the testing image and features of template images. It should assign the testing image to the closest template class in feature space, as measured e.g. by Euclidean distance. ■