2. Image Filtering, Convolution & Gradients

2.1 Goals

The goal of this exercise is to

- Understand and apply a convolutional filter to an image
- Compute image gradients and apply them to real-world images
- Compare the computational complexity of separable and non-separable filters

2.2 Instructions

- To get started, first **download** the corresponding images and code from Moodle.
- Make sure you have the **Matlab Image Processing toolbox** installed.
- To get started with the code, look at the file `main.m` given to you.

2.3 Applying Convolutional Filters

Assume we are given a gray-scale image $I[x, y]$, of size $W \times H$, such that $1 \leq x \leq W$, and $1 \leq y \leq H$.

We want to apply a filter $F[i, j]$ to image I. The filter F is of size $(2N + 1) \times (2M + 1)$, such that $-N \leq i \leq N$, and $-M \leq j \leq M$.

The result can be computed as

$$R[x, y] = (I * F)[x, y] = \sum_{i=-N}^{N} \sum_{i=-M}^{M} I[x + i, y + j]\, F[i, j] \tag{2.1}$$

Exercise 2.1 Implement a function `R = applyImageFilter(I,F)` that takes a gray-scale image `I` and a filter `F` as inputs, and returns the result of the convolution of the two.

Note: Matlab already has a function to convolve an image with a filter, but in this exercise you are requested to implement your own. This will help your understanding of how convolutional filters work.

- To avoid numerical issues, make sure `I` and `F` are of type `double`. To understand why, think what would happen if you add two unsigned 8-bit numbers when computing the convolution, for example 240 and 80.
- What happens when computing $R[x, y]$ near the border of the image? What would you propose to deal with this issue?
- To visualize the output `R`, try using `imagesc()` instead of `imshow()`. Try to understand the difference between the two functions.

Hint: Pre-allocate the result image `R` before evaluating Eq. (2.1) for each pixel. ■

You should get an output similar to Fig. 2.1.

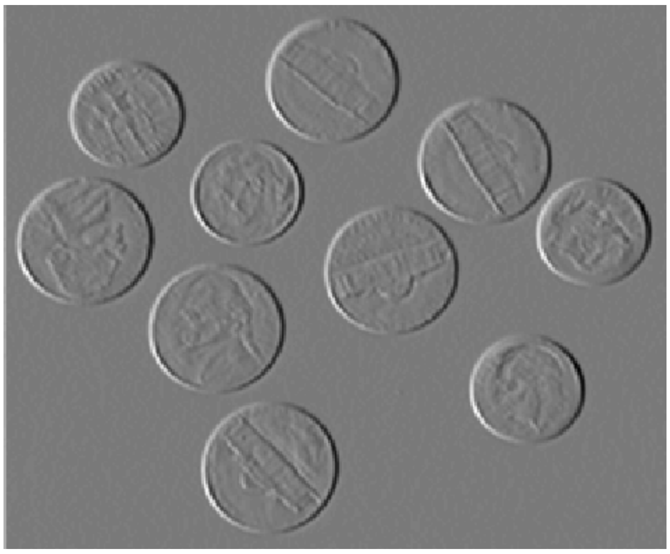

Figure 2.1: Example output of Exercise 2.1.

2.4 Edges and edge detectors

The result you just computed is the convolution of a sample image with a filter called the Sobel Edge Detector. Actually, the Sobel Edge Detector consists of two 3×3 filters,

$$S_x = \frac{1}{8}\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \qquad \text{and} \qquad S_y = \frac{1}{8}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}, \tag{2.2}$$

where S_x computes the partial derivative of the image in the horizontal direction, while S_y does it for the vertical direction.

Exercise 2.2 In the previous exercise you computed the convolution with S_x. Look at the resulting image and understand what it means. Then answer the following questions:

- What is the value of the filtered image when going from a dark area to a bright area, from left to right? You can use `impixelinfo()` and the mouse to show pixel values over an image.
- What is the behavior when going from a bright to a dark area, from left to right?
- Why is there a term $\frac{1}{8}$ in the filters?

■

Mathematically, the goal of the filters S_x and S_y is to approximate the derivatives of the image with respect to the horizontal and vertical directions respectively, such that

$$\nabla I_x(x, y) = (I * S_x)[x, y] \qquad \text{and} \qquad \nabla I_y(x, y) = (I * S_y)[x, y]$$

Therefore, the gradient of an image is a 2D vector

$$\nabla I = \begin{bmatrix} \nabla I_x \\ \nabla I_y \end{bmatrix}. \tag{2.3}$$

This gradient vector can be computed for every pixel. Its magnitude and phase can be computed as

$$\|\nabla I\| = \sqrt{(\nabla I_x)^2 + (\nabla I_y)^2} \tag{2.4}$$

$$\angle \nabla I = \mathrm{atan2}\Big(\nabla I_y,\, \nabla I_x\Big) \tag{2.5}$$

Note: we use `atan2()` instead of `atan()` to be able to determine the right quadrant of the phase.

Exercise 2.3 Now you will write code to compute the gradient magnitude and phase.

- Write the necessary code to convolve the image with S_x and S_y to obtain the horizontal and vertical gradient images.
- With equations (2.4) and (2.5) compute the gradient magnitude and gradient phase images. Show them beside the original image with `subplot()` and `imagesc()`.
- Analyze the resulting images. Where does the gradient maximum get high values? How does the phase change as you move around the border of a coin? Use `impixelinfo()` to investigate this.

Hint: Remember that to do element-wise operations with matrices in MATLAB you have to use the dot-like operators. For example, to square each element in a matrix you write `M.^2`, and *not* `M^2` (the latter is equivalent to matrix multiplication: `M^2 == M * M`). ■

2.5 Image Smoothing

Convolutional filters have many uses. A very common one is to smooth or soften an image. A typical smoothing filter is the Gaussian Filter, which follows the equation of a 2D Gaussian probability distribution.

From now on, we will use MATLAB's `imfilter()` to apply a filter to an image, which is highly optimized for performance.

Exercise 2.4 Image Smoothing

- To create a Gaussian filter, use `fspecial()`:

  ```
  gaussFilt = fspecial('gaussian', fSize, fSigma)
  ```

  - What are `fSize` and `fSigma`? Read the help of `fspecial()` for detailed information.
  - How do `fSize` and `fSigma` affect the filter's shape and size? Use `imagesc()` to visualize different filters with different parameters.
  - If you are given `fSigma`, how would you choose `fSize`?

- To apply the filter to an image, use `imfilter()`:

  ```
  res = imfilter( img, gaussFilt )
  ```

  - Experiment with different values of `fSigma`. How does the amount of smoothing vary with this parameter?

■

2.6 Border Effects

As you saw in the first exercise of this session, applying the filter near the border of the image is not a well-defined problem. It is possible to deal with this issue through different approaches.

Luckily, `imfilter()` implements three different ways of dealing with border effects. Here you will try them out and understand what each of them does.

Exercise 2.5 Border Effects

- Create a Gaussian filter with `fSigma = 10` and `fSize = 25`:

```
gaussFilt = fspecial('gaussian', 25, 10)
```

- To apply the filter to an image, use `imfilter()`:

```
res = imfilter( img, gaussFilt, <borderMethod> )
```

  - To find out what to write in `<borderMethod>`, you can check the help of `imfilter()`.
  - Try out the different options, and see how that affects the output.
  - Make sure you understand what each option does, and think of the situations in which each of them would help.

■

2.7 Separable and Non-separable Filters

As seen in class, certain types of 2D filters can be thought of as the composition of two 1-dimensional filters. These are called Separable Filters, and can be computed more efficiently than those who are non-separable.

For example, the Sobel filter S_x can be decomposed as

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} , \tag{2.6}$$

which means that the convolution of I with the 2D filter S_x can be simplified to two convolutions with 1D filters,

$$\begin{aligned} I * S_x &= \frac{1}{8} I * \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \\ &= \frac{1}{8}\left(I * \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \right) * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} . \end{aligned}$$

For the comparison of the computational complexity of separable and non-separable filters, check out the lecture notes.

Exercise 2.6 Separable and Non-separable Filters

Note: `imfilter()` automatically detects if the filter passed as its second argument if separable. If so, it decomposes it into two 1D filters to speedup computation.

We will compare the performance of separable and non-separable filters.

- Create a separable filter (for example a Gaussian filter) and a non separable filter (for example a random filter) and filter the image of Figure 2.1 with the 2 filters.
- Repeat the test with several increasing filter sizes.
- measure the elapsed time for each filtering. For doing this, you can use the `tic()` and `toc()` functions.
- Plot the results in a graph *(elapsed time)* vs *(filter size)*.
- What is your conclusion ? Is filtering with separable filters always faster than with non-separable ones?

■

2.8 Bonus: Implement Separable Convolution

So far we used Matlab functions to convolve the image with a separable filter. Now you can try for yourself to compute this convolution as the results of two one-dimensional convolutions.

You can reuse the code you wrote for the first exercise of this session.