Computer Vision 1: Exercise Sheet 6

Summary:

1. Discrete Fourier transformations and the frequency domain using `numpy`
2. Filtering images in the frequency domain

1 DFT with NumPy

1. Use `numpy.fft.fft` to calculate the DFT for the signal f from Homework sheet 5, Task 3. Verify that you get the same result as for the homework problem.

2. Read `MaruTaro.jpg` from Moodle. Convert it to grayscale.

Figure 1: Maru Taro the happy dog. Image source: Maru Taro Instagram

   - Use `numpy.fft.fft2` to calculate the DFT of the grayscale image. Print out the value at index $[0, 0]$ of the result. What does this value correspond to? Verify your answer by calculating the value directly from the image. Hint: What do you get from the definition of the 2D DFT $F(u, v)$ when $u = 0$, $v = 0$?
   - Using `matplotlib` subplots, draw side-by-side the 1) Fourier or magnitude spectrum and 2) phase spectrum of the image.

     Hints: `numpy.abs`, `numpy.angle`. If you cannot see the magnitude spectrum in the image, scale it with `numpy.log` before plotting.
   - Create a similar plot as above, but before plotting, call `numpy.fft.fftshift` on the DFT. What is the significance of calling `fftshift`? How do you interpret each image; e.g., where are low frequencies and where are high frequencies?

3. Read the `halftone2.jpg` image from Moodle (Figure 2), and draw its magnitude spectrum. Do you see the strong peaks that appear in the magnitude spectrum (Figure 3 left)? How could you interpret them?

Hints: this image contains black dots that are spaced roughly equidistantly in the spatial domain. As a result, it contains quite a lot of one frequency component.

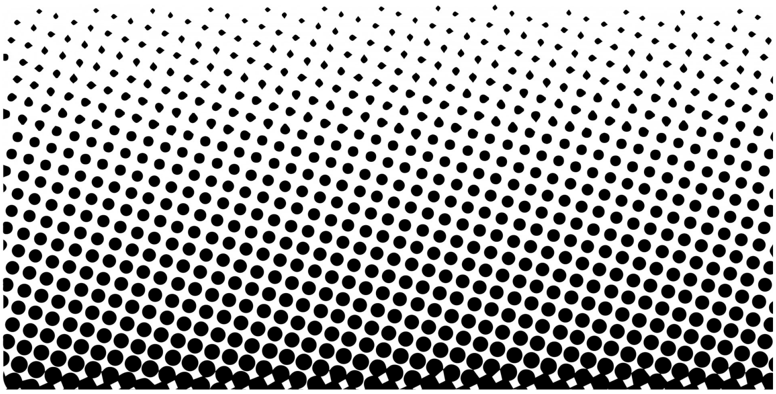

Figure 2: halftone2.jpg

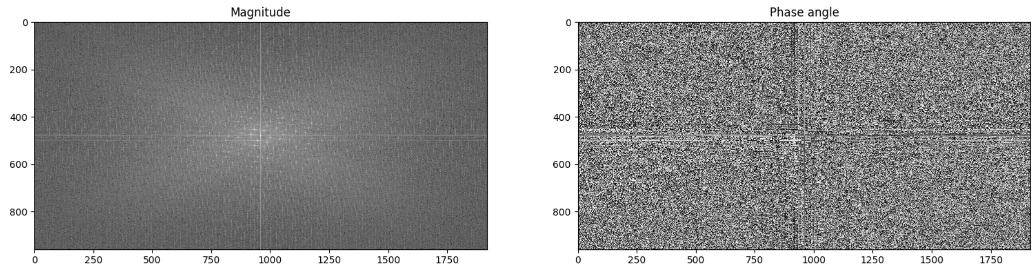

Figure 3: Magnitude and phase spectrum of halftone2.jpg

2 Filtering in the frequency domain

We return to processing `MaruTaro.jpg`. We want to apply the ideal low-pass filter to the image, so that only the lowest frequencies remain in the image. Recall the convolution theorem from the lecture, i.e., that convolution in time domain is equivalent to multiplication in the frequency domain. We will use the procedure for filtering in the frequency domain presented in the Gonzalez & Woods textbook, Section 4.7 but without any padding.

The ideal lowpass filter is specified in the frequency domain by

$$H(u,v) = \begin{cases} 1 & \text{if } D(u,v) \leq D_0 \\ 0 & \text{if } D(u,v) > D_0, \end{cases}$$

where D_0 is the positive constant called the cut-off frequency, and $D(u,v)$ is the distance of point (u,v) in the frequency domain from the center of the P-by-Q frequency rectangle[1]:

$$D(u,v) = \sqrt{(u - P/2)^2 + (v - Q/2)^2}.$$

[1]Python uses 0-based index, so when P or Q is odd, we take the lower bound integer of $P/2$. For example when $P = 3$ take $P/2$ to be 1.

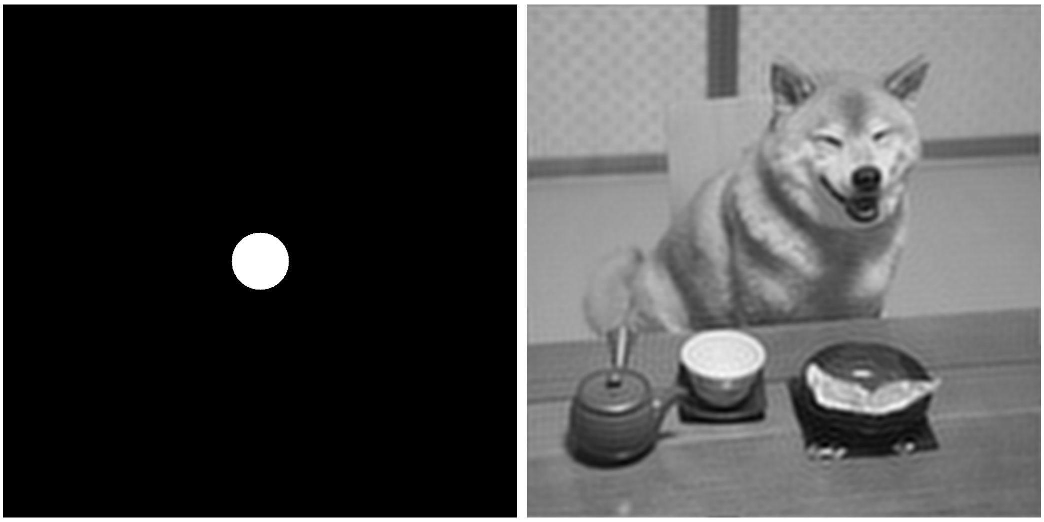

Figure 4: Left: The ideal low-pass filter. Black areas have value zero, and white areas have value 1. Right: Lowpass filtered image.

Here, without any padding, P and Q are equal to the height and width of the original image, respectively.

- Create a filter H with a cut-off frequency of $D_0 = 60.0$ that looks similar to Figure 4 left. Visualize it.
- Apply it according to the aformentioned procedure. The final low-pass filtered image looks similar to Figure 4 right. You should be able to see some ringing artifacts similar to the examples shown on the lecture slides.
- (Optional) The filtering procedure from the Gonzalez & Woods textbook, Section 4.7, uses padding to avoid wrap-around error. Repeat the filtering process above with zero-padding similar to the procedure in the textbook. Note that with padding we have P and Q are *twice* the height and width of the original image, respectively.

Hints: Use `numpy.meshgrid` to create arrays U and V of column and row coordinates. Use the coordinates and D_0 to create a boolean or "mask" index array that looks like H. Depending how you specify P and Q, you might need to use the parameter `indexing='ij'` in `numpy.meshgrid` to make the mask H has the same dimensions as the image.