TDT4195: Visual Computing Fundamentals

Digital Image Processing - Assignment 2

October 26, 2017

Aleksander Rognhaugen
Department of Computer Science
Norwegian University of Science and Technology (NTNU)

- **Delivery deadline: November 10, 2017** by 23:00.
- **This assignment counts towards 5 % of your final grade.**
- You can work on your own or in groups of two people.
- Deliver your solution on *Blackboard* before the deadline.
- Please upload your report as a PDF file, and package your code into a single ZIP archive. Not following this format may result in a score deduction.
- The programming tasks may be completed in the programming language of your choice, however, it might be a good idea to select one that supports matrix and image processing operations, e.g. MATLAB or Python with NumPy. *The lab computers at ITS-015 support Python and MATLAB.*
- The delivered code is taken into account with the evaluation. Ensure your code is documented and as readable as possible.
- For each programming task you need to give a brief explanation of what you did, answer any questions in the task text, and show any results, e.g. images, in the report.
- In this assignment, you can use existing implementations of the fast Fourier transform (FFT), such as `numpy.fft.fft2()` from NumPy[1] (Python) and `fft2()` from MATLAB[2].

Objective: Gain a deeper understanding of the convolution operation using the convolution theorem.

[1]FFT in NumPy: `https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.fft.html`
[2]FFT in MATLAB: `https://se.mathworks.com/help/images/fourier-transform.html`

Task 1: Theory Questions [1.5 points]

a) **[0.2 points]** The convolution theorem can be seen in Equation 1, where $\mathcal{F}$ is the Fourier transform, $*$ is the convolution operator, and $\cdot$ is pointwise multiplication. Explain in your own words:

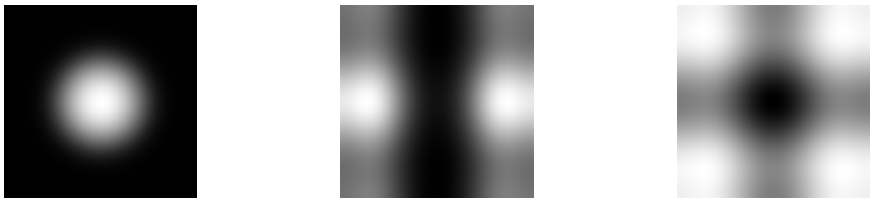
i) What does the convolution theorem entail?

ii) What are the main steps an implementation of frequency filtering requires?

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\} \tag{1}$$

b) **[0.2 points]** Convolution kernels are typically understood in terms of the frequency domain. In your own words, explain what high- and low-pass filters are.

c) **[0.2 points]** The Fourier spectrum $|\mathcal{F}\{f\}|$ of three commonly used convolution kernels can be seen in Figure 1. For each kernel (a, b, and c), write down what kind of kernel it is (high- or low-pass). Explain your reasoning.



(a) (b) (c)

Figure 1: The Fourier spectrum $|\mathcal{F}\{f\}|$ of three convolution kernels that have been transformed by the Fourier transform. The zero-frequency, or DC, component have been shifted to the centre for all images. The logarithmic transformation $\log(1 + |\mathcal{F}\{f\}|)$ was applied to better see the Fourier coefficients.

d) **[0.4 points]** Answer the following questions in your own words:

i) Explain why padding is important when we want to do frequency filtering.

ii) In the context of frequency filtering, we commonly pad images by *appending* zeros to the rows and columns as seen in Figure 2 (a). Let's assume we instead pad the images by *prepending* zeros to the rows and columns. This can be seen in Figure 2 (b). Let the number of zeros we pad by remain the same. Does this change the result of frequency filtering? Explain your reasoning.

(a) (b)

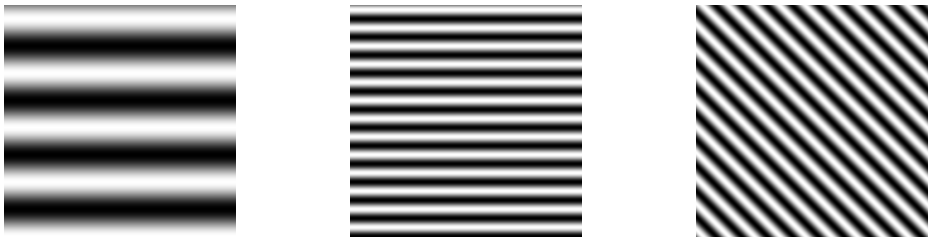
Figure 2: Here we see two different padding schemes for an image called $\boldsymbol{I}$. In (a), the image has been padded by *appending* zeros, while in (b) the image has been padded by *prepending* zeros.

e) **[0.3 points]** Say we have an image like the one in Figure 3 (a). Answer the following questions in your own words:

i) What would the spectrum look like after the image has been transformed to the frequency domain by the Fourier transform?

ii) If we let the frequency of the sinusoidal grating increase as in Figure 3 (b), how does the spectrum change?

iii) How would the spectrum change if the sinusoidal grating is rotated in the XY plane, as seen in Figure 3 (c)?



(a) (b) (c)

Figure 3: Three images of a sinusoidal grating. (a) and (b) have different frequency, while (c) is rotated in the XY plane.

f) **[0.2 points]** There are two ways to avoid aliasing. One of them is to sample the signal, such as an image, with a frequency at least twice the highest frequency in the signal. In your own words, explain how aliasing can be avoided when we cannot increase the sampling rate.

Task 2: Frequency Domain Filtering [1 point]

The Fourier transform is an important signal processing tool that allows us to decompose a signal into its sine and cosine components[3]. A discrete version is used on digital images. It does not contain all frequencies, but the number of frequencies sampled are enough to represent the complete image. A two-dimensional version of the discrete Fourier transform (DFT) can be seen in Equation 2. It transforms an $N \times M$ image in the spatial domain to the frequency, or Fourier, domain. The number of frequencies in the Fourier domain is equal to the number of pixels in the spatial domain.

$$F(u,v) = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f(x,y) e^{-i2\pi(\frac{xu}{N} + \frac{yv}{M})} \qquad \text{where} \qquad f(x,y) \in \mathbb{R}^N \times \mathbb{R}^M \tag{2}$$

The computational complexity of the DFT is $\mathcal{O}(N^3)$, assuming $N = M$.[4] Thus, the DFT is barely runnable on all but the smallest of images. Fortunately, in 1965, Cooley and Tukey published the first of a slew of algorithms commonly referred to as the fast Fourier transform (FFT)[1].[5] The FFT algorithm reduces the computational complexity of the DFT to $\mathcal{O}(N^2 log_2 N)$, assuming $N = M$.

a) **[1 point]** Using an already existing FFT implementation, implement a function that takes a greyscale image and an arbitrary linear convolution kernel and performs frequency filtering.

   Apply the function you just made on an appropriate greyscale image with both a high- and a low-pass convolution kernel. It's up to you to choose which convolution kernels to use. For each of the kernels, you must display the following in your report:

   i) The filtered image(s) in the spatial domain.
   ii) The spectrum $|\mathcal{F}\{f\}|$ of the image(s) before and after the multiplication.
   iii) The spectrum of the convolution kernel(s).

   Make sure to *shift* the zero-frequency, or DC, component to the centre before displaying the spectrum.

   Hint: Fourier spectra typically have a large dynamic range, which make it difficult to see details. As mentioned earlier, this can be alleviated by applying the logarithmic transformation: $\log(1 + |\mathcal{F}\{f\}|)$.

[3] Remember that a complex exponent e^{it} can be rewritten in terms of an imaginary sine part and a real cosine part: $e^{it} = \cos t + i \sin t$ (Euler's formula), where $i^2 = -1$.

[4] You might think that the computational complexity should be $\mathcal{O}(N^4)$, however the DFT is separable. That is, we can get the same result by running 1D DFT along each row of the image to produce an intermediate array. The final result can be assembled by running 1D DFT along the columns of this array.

[5] A similar method was found by Carl Friedrich Gauss around 1800, however, it was left unpublished.

Task 3: Unsharp Masking [1.2 points]

Unsharp masking is a technique for sharpening images. It can be seen summarised in Equation 3. The name, *unsharp*, comes from how an unsharp, or smoothed, version of an image is subtracted from the original to create an edge image. After we have the edge image, it is scaled and added to the original image in order to sharpen it.

$$\boldsymbol{I}_{\text{sharp}} = \boldsymbol{I} + k(\boldsymbol{I} - \boldsymbol{I}_{\text{smooth}}) \qquad \text{where} \qquad k \in [0, 1] \tag{3}$$

a) **[1 point]** Implement a function that performs unsharp masking on a greyscale image using frequency filtering. Instead of implementing Equation 3 directly you **must** compute a *single* kernel, which when used with frequency filtering will yield the desired sharpened image. We recommend that you use Gaussian smoothing when creating the unsharp kernel.

   i) In your own words, explain how you created your *unsharp* kernel.

   ii) Display the Fourier spectrum of your unsharp kernel in your report. If you elected to use Gaussian smoothing, then simply pick an appropriate size and standard deviation.

   Hint: The Dirac delta function will be useful in the creation of your kernel.

b) **[0.2 points]** Apply your unsharp function on either a greyscale or colour image and display the following in your report:

   i) The image in the spatial domain before and after sharpening.

   ii) The spectrum $|\mathcal{F}\{f\}|$ of the image before and after the multiplication.

   Sharpening on colour images can be done channel-wise.
   Hint: If the sharpened image appear grey, then this can be remedied by clipping. For example, if you expect the intensity values to be $\in [0, 1]$*, then values outside of this interval can be clipped to the interval edges.*

Task 4: Selective Filtering [1.3 points]

Selective filtering is a type of frequency filtering where regions of the frequency image are filtered. These types of filters are generally classified as being either band-reject or notch-reject. The former *removes* a band, or range, of frequencies, while the latter *removes* a local neighbourhood of frequencies centred around a single frequency[6]. For notch filters we are often interested in symmetric filters. These are filters where for every notch centred at some location (u_i, v_i) we have a notch centred at $(-u_i, -v_i)$.

[6] This can be visualised as translating a high-pass filter on top of a desired centre frequency.

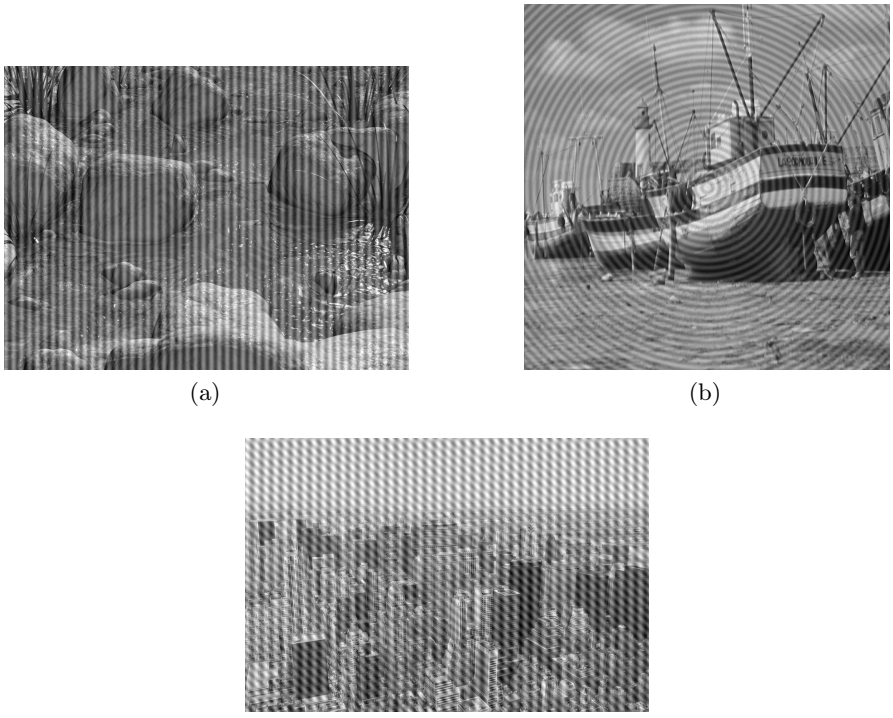
a) **[1.3 points]** Using frequency filtering, attempt to remove the periodic noise from two of the three images in Figure 4. For each of the two images you select, do the following:

i) Identify the periodic noise using the spectrum of the image. Explain your reasoning.

ii) Create a selective filter and display it in your report.

iii) Remove the noise using your filter. Display the filtered image as well as its spectrum in your report.

Hint: It's up to you to come up with a way to create the selective filters. For instance, you may create them by hand using a raster graphics editor, or use built-in functions in your programming language.



(a) (b)

(c)

Figure 4: Three images containing different kinds of periodic noise.

Optional: Implementing the Fast Fourier Transform [0 points]

This task is optional. The intention with this task is for you to gain further insights into the fast Fourier transform. We will consider a common FFT algorithm called radix-2 decimation in time (DIT) FFT. Due to the separability of the Fourier transform we will only be discussing the 1D FFT.

The speed-ups gained by the FFT is due to the fact that the Fourier transform can be computed recursively if the length is a power of two. Namely, a Fourier transform of length N can be computed as the sum of two DFTs of length $\frac{N}{2}$. Consider the 1D DFT in Equation 4:

$$\begin{aligned} F(k) &= \sum_{n=0}^{N-1} f(n) e^{-i\frac{2\pi}{N}nk} \\ &= \sum_{n=0}^{N-1} f(n)\, \boldsymbol{W}_N^{nk} \qquad \text{for} \qquad k = 0 \dots N-1 \end{aligned} \tag{4}$$

where $\boldsymbol{W}_b^a$ is defined as:

$$\boldsymbol{W}_b^a = e^{-i\frac{2\pi}{b}a} \tag{5}$$

The radix-2 decimation in time FFT procedure consists of dividing / decimating the input $f(n)$ of length N into *two* sequences with length $\frac{N}{2}$, recursively. We will call the two sequences $f_e(m)$ and $f_o(m)$, where $m = 0 \dots \frac{N}{2} - 1$. The $f_e(m)$ sequence contains samples with even indices, $f(2m)$; and the $f_o(m)$ sequence contains samples with odd indices, $f(2m+1)$. Let's first modify the DFT to account for this division of samples:

$$\begin{aligned} F(k) &= \sum_{n=0}^{N-1} f(n)\, \boldsymbol{W}_N^{nk} \qquad \text{for} \qquad k = 0 \dots N-1 \\ &= \sum_{m=0}^{\frac{N}{2}-1} f(2m)\, \boldsymbol{W}_N^{2mk} + \sum_{m=0}^{\frac{N}{2}-1} f(2m+1)\, \boldsymbol{W}_N^{(2m+1)k} \\ &= \sum_{m=0}^{\frac{N}{2}-1} f(2m)\, \boldsymbol{W}_N^{2mk} + \boldsymbol{W}_N^k \sum_{m=0}^{\frac{N}{2}-1} f(2m+1)\, \boldsymbol{W}_N^{2mk} \end{aligned} \tag{6}$$

Because $\boldsymbol{W}_N^2 = \boldsymbol{W}_{\frac{N}{2}}$, this can be be expressed as:

$$\begin{aligned} F(k) &= \sum_{m=0}^{\frac{N}{2}-1} f_e(m)\, \boldsymbol{W}_{\frac{N}{2}}^{mk} + \boldsymbol{W}_N^k \sum_{m=0}^{\frac{N}{2}-1} f_o(m)\, \boldsymbol{W}_{\frac{N}{2}}^{mk} \\ &= F_e(k) + \boldsymbol{W}_N^k F_o(k) \qquad \text{for} \qquad k = 0 \dots N-1 \end{aligned} \tag{7}$$

where $F_e(k)$ and $F_o(k)$ are the $\frac{N}{2}$-point DFTs of $f_e(m)$ and $f_o(m)$, respectively. In other words, the N-point DFT can be obtained by taking two $\frac{N}{2}$-point DFTs: one for the even input data, and one for the odd input data. Naturally, we can continue decimating the input, recursively, by, for example, computing the $\frac{N}{2}$-point DFT as two $\frac{N}{4}$-point DFTs. For length N, where N is a power of two, an input sequence can be divided $\log_2 N$ times before reaching $N = 1$. In the cases where N is not a power of two, we can pad the input until it is.

To finish the derivation of the 2-radix DIT FFT we will recognise that the DFT is periodic, and seeing as $F_e(k)$ and $F_o(k)$ are DFTs we have that $F_e(k + \frac{N}{2}) = F_e(k)$ and $F_o(k + \frac{N}{2}) = F_o(k)$. This coupled with the fact that $\boldsymbol{W}_N^{k+\frac{N}{2}} = -\boldsymbol{W}_N^k$ enables us to rephrase Equation 7 as:

$$\begin{aligned} F(k) &= F_e(k) + \boldsymbol{W}_N^k F_o(k) \\ F(k + \frac{N}{2}) &= F_e(k) - \boldsymbol{W}_N^k F_o(k) \qquad \text{for} \qquad k = 0 \dots \frac{N}{2} - 1 \end{aligned} \tag{8}$$

The same line of thinking can be applied for the inverse DFT as well, giving us the inverse FFT. However, to simplify matters it is possible to express the inverse FFT in terms of the FFT by applying the complex conjugate $(\cdot)^*$ operation:

$$\mathcal{F}^{-1}\{\mathbf{x}\} = \frac{1}{N}\mathcal{F}\{\mathbf{x}^*\}^* \tag{9}$$

That is, run the FFT on the complex conjugate of the input, take the complex conjugate of the output, and scale it by dividing by the length of the input.

References

[1] James W. Cooley and John W. Tukey "An algorithm for the machine calculation of complex Fourier series" in: *Mathematics of computation* 19.90 (1965), pp. 297–301 (cit. on p. 4)