1 Linear Filters and Frequency Analysis

This post is written as part of AIHelsinki study group on natural image statistics. This text mostly covers material from Natural Image Statistics book chapter 2 available here `http://www.naturalimagestatistics.net/`. The format of this post is something between learning diary and a summary but I tried to include some original examples with reproduction instructions in the text.

The topics covered in this post are mostly the same as in the textbook chapter 2 including linear filtering, frequency analysis, and space-frequency analysis. Some material has been left out and there may be inaccuracies so please consider this post supplemental to the textbook chapter.

1.1 Linear Filtering, Convolution and Impulse Response

Linear filtering is an image processing technique where a filter is applied to an input image to produce an output. Inputs and outputs of the linear filtering operation are images. The filter itself can be thought of as an image as well. The filtering operation corresponds to sliding the filter on the input pixel by pixel and computing the output.

The output $O(x, y)$ is given by following equation

$$O(x,y) = \sum_{x_*=-K}^{K} \sum_{y_*=-K}^{K} W(x_*, y_*) I(x + x_*, y + y_*) \tag{1}$$

where W is the filter of size $(2K + 1) * (2K + 1)$, I is the input image and O is the output image. For each position (x, y) the output is given by the linear correlation of the filter with the sub-area in the same position in the input image. The output pixel at (x, y) is computed by taking the sum over the products of the filter values at positions (x_*, y_*) and the input values centered at (x, y) offset by (x_*, y_*).

Convolution is an operation closely related to linear correlation. The term convolution might be familiar to people following news on neural networks and deep learning. We are talking about the convolution in Convolutional Neural Networks. Convolution is defined as

$$I_1(x, y) * I_2(x, y) = \sum_{x_*=-\infty}^{\infty} \sum_{y_*=-\infty}^{\infty} I_1(x - x_*, y - y_*) I_2(x_*, y_*) \tag{2}$$

where I_1 and I_2 are images. As can be seen here the main difference between linear correlation and convolution is the sign of the terms x_* and y_*. Convolution and linear filtering are related through what is called the impulse response.

Impulse response $H(x, y)$ is the response of a filter to a short impulse. Short impulse is defined as either Dirac delta function for continuous systems or Kronecker delta function for discrete systems such as 2D images. The Kronecker delta function is given by

$$\delta(x,y) = \begin{cases} 1, & \text{if x = y} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

The impulse response $H(x,y)$ of a filter $W(x,y)$ can be computed by plugging the delta function above to the equation of linear filtering 1 defined at the top of the page. By doing so you can observe that $H(x,y) = W(-x,-y)$. This defines the relationship between convolution and linear correlation. The filter used in convolution is thus a ”mirror image“ of the weights used in linear correlation.

Why would anyone bother mirroring their weight matrix to compute convolutions instead of linear correlations you may ask? There are two reasons. In the context of natural image statistics this is done because the impulse response has desired properties in the frequency based analysis of linear filtering, which we will come back to in the next section. The other reason is that the convolution operation offers mathematical convenience as it is a commutative operation `http://www.deeplearningbook.org/contents/convnets.html`.

With convolution in our toolbox we can express the filtering operation using the impulse response as

$$O(x,y) = I(x,y) * H(x,y) \tag{4}$$

1.2 Frequency-based representation

By looking at the values of a 2D filter it is often hard to tell what its effect is on the input image. Frequency-based representation is an useful tool for the analysis of such linear filters. Frequency-based representation is a way of representing signal in terms of sinusoidal components.

The regular way to represent images is called the spatial representation. The spatial representation of an image is a set of numbers which represent the intensity of the pixel in that channel (in grayscale there's one channel, in RGB there are three etc.). In the frequency based representation the image is given as a set of numbers that describe the *amplitudes* and *phases* of the sinusoidal components of the image. The sinusoidal signal is of the form $A\cos(\omega x + \Psi)$ where A is the amplitude, ω is the frequency and Ψ is the phase.

The frequency-based representation of a one dimensional signal is given by

$$I(x) = \sum_{u=0}^{U-1} A_u \cos(\omega_u x + \Psi_u) \tag{5}$$

where $U = (N-1)/2$ if the original signal contains N real-valued numbers. For images and other two-dimensional data the signal becomes a bit more complicated to account for the extra dimension. Two dimensional frequency-based representation of a signal is given by $A\cos(\omega_x x + \omega_y y + \Psi)$, where ω_x and ω_y

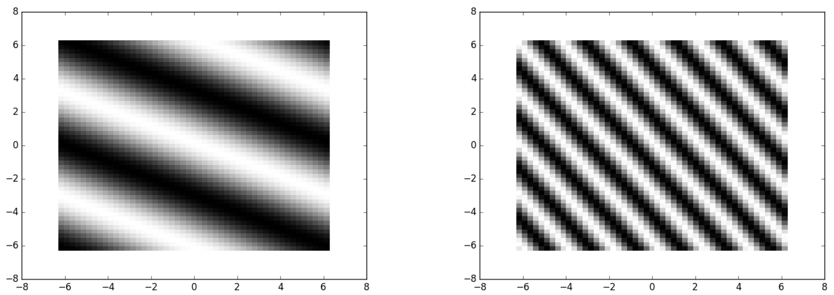

Figure 1: Two examples of sinusoidal waves with different x and y frequencies

stand for frequencies in corresponding directions. The two dimensional sinusoidal components are basically just cosine waves on a plane with different frequencies, amplitudes and phases. The figure 1 present examples of such waves. The transformation from the spatial domain to the frequency domain can be done using discrete Fourier transformation. The theory of discrete Fourier transformation is quite involved and will not be reviewed here. The textbook chapter 20 provides a mathematical treatment of the theory of Fourier analysis. Details on how DFT is used for acquiring the frequency based representation are provided in the textbook chapter 2. For now we are going to focus on the uses of the frequency-based representation.

Linear filters and sinusoidals have a special relationship. The output of linear filtering on a sinusoidal wave is a sinusoidal wave with same frequency but possibly different phase and amplitude. If the input signal has the following frequency-based representation

$$I(x,y) = \sum_{\omega_x} \sum_{\omega_y} A_{\omega_x,\omega_y} \cos(\omega_x x + \omega_y y + \Psi_{\omega_x,\omega_y}) \tag{6}$$

the response of the linear filter has the following frequency-based representation

$$\begin{aligned} O(x,y) &= H(x,y) * I(x,y) = \\ &\sum_{\omega_x} \sum_{\omega_y} |\tilde{H}(\omega_x,\omega_y)| A_{\omega_x,\omega_y} \cos(\omega_x x + \omega_y y + \Psi_{\omega_x,\omega_y} + \angle\tilde{H}(\omega_x,\omega_y)) \end{aligned} \tag{7}$$

where $|\tilde{H}(\omega_x,\omega_y)|$ denotes the amplitude magnification factor of the linear filter and $\angle\tilde{H}(\omega_x,\omega_y)$ denotes the phase shift of the filter. The $|\tilde{H}(\omega_x,\omega_y)|A_{\omega_x,\omega_y}$ part of the equation is called the amplitude response.

The use of this property of the frequency-based representation makes analysis of the linear filters straightforward. It is difficult to describe the behavior of a filter based solely on its spatial representation but if its amplitude response is available you can tell what the filter does for different frequencies.

1.2.1 Example of Linear Filtering

The concepts described above are demonstrated in the following code snippet. NumPy is used throughout the example and the library name abbreviated as np.

First a gaussian kernel is extracted from the SciPy gaussian filtering operation using the properties of impulse response. Next the filter is applied to an input image using 2d convolution. The input image is just a 2d NumPy array containing the grayscale input image. The image can be read using PIL, SciPy, OpenCV or similar. And finally the frequency-based representation of the filter is obtained using Fast-Fourier Transformation.

```
# Here we make use of the properties of the impulse
# response to get a gaussian kernel using the SciPy
# gaussian filter. Create an array with all zeros
# except for one element in the center.
impulse = np.zeros((21,21))
impulse[10,10] = 1.

# Apply the gaussian filter to the impulse to obtain
# the gaussian kernel. gaussian_filter is
# scipy.ndimage.filters.gaussian_filter
gaussian_kernel = gaussian_filter(impulse, 3)

# Convolve the kernel with the example image
filtered_input = signal.convolve2d(img, gaussian_kernel)

# Compute the frequency-based representation of the
# filter fftshift is a convenience method for
# organizing the fft-output
f_b_representation = np.abs(
    np.fft.fftshift(np.fft.fft2(gaussian_kernel)))
```

The gaussian kernel extracted from the SciPy filtering operation and its amplitude response are shown in the figure 2. The input image for the convolution and the results of convolution with Gaussian are presented in the figure 3. (source: `https://commons.wikimedia.org/wiki/File:Carl_Friedrich_Gauss_1840_by_Jensen.jpg`)

The amplitude response of the filter kernel is shown in figure 2. In the picture dark pixels correspond to small amplitude and light pixels correspond to high amplitude. The amplitude of the 0 frequency is shown in the center of the picture. The pixels away from the center have higher frequencies. The direction where the pixel is from the center corresponds to the direction of the wave.

The Gaussian filter does not have any surprising properties in the frequency domain. In the frequency-based representation only the components very close to the origin remain, which means the high-frequency components have been

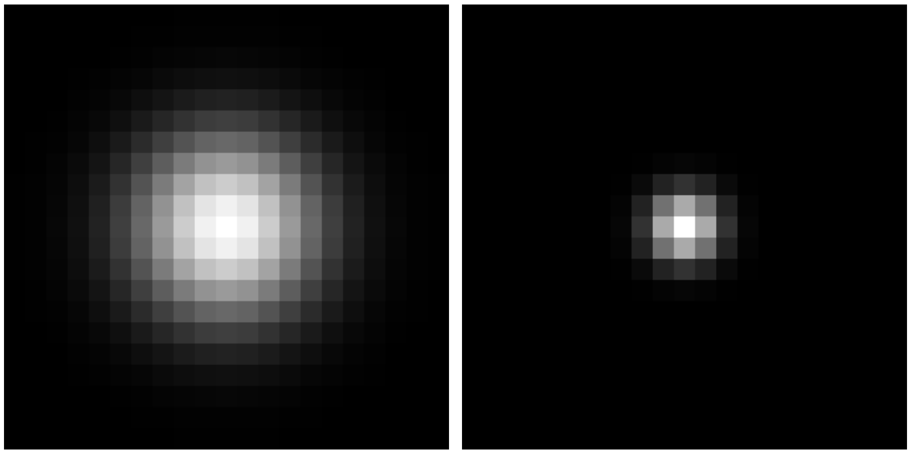
Figure 2: Gaussian kernel on the left with its amplitude response on the right.

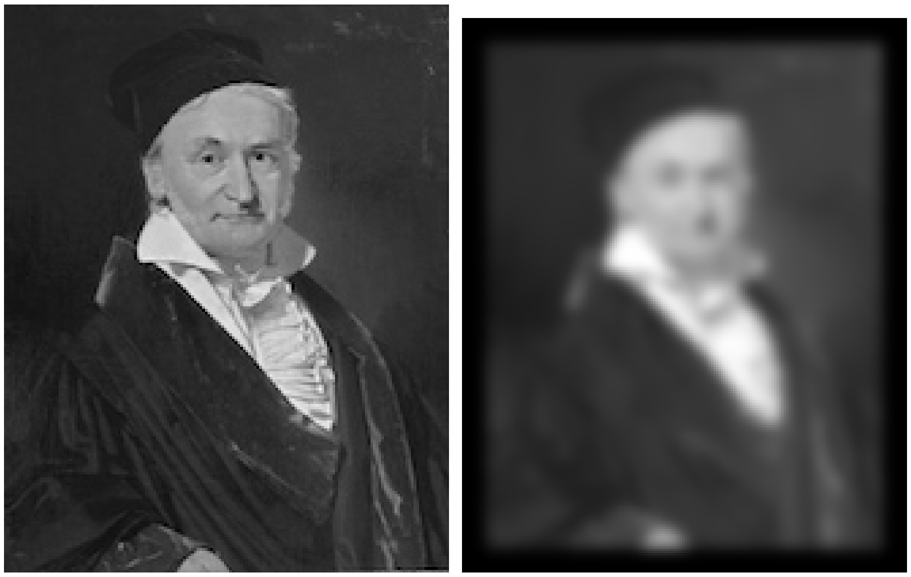
Figure 3: Input and output images of the Gaussian filter.

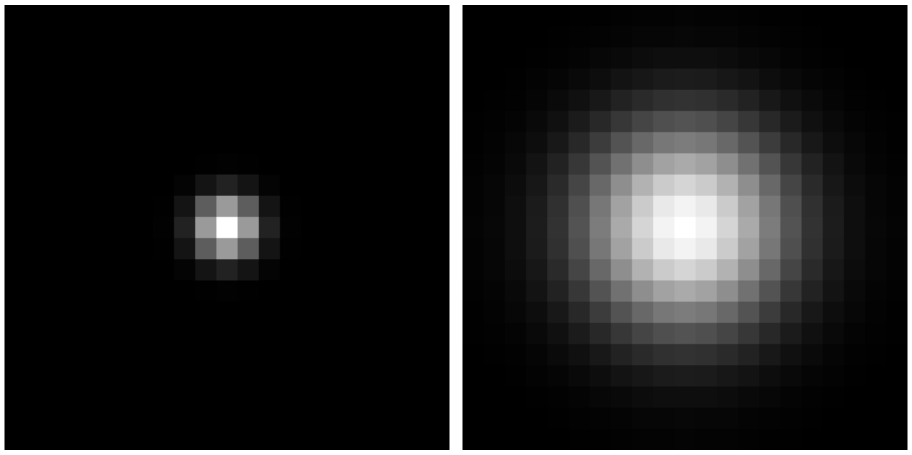

Figure 4: Smaller σ kernel and its amplitude response.

removed. The final image is blurred as a result.

With a Gaussian kernel computed with smaller standard deviation more of the high frequency components are retained. Figure 4 presents the new kernel and its amplitude response. The corresponding amplitude response shows that there are more high frequency components retained (bright pixels away from origin). Filtering with such kernel will result to less blurred image compared to the kernel in figure 2.

1.3 Frequency-based Representation as a Basis

The frequency-based representation can be viewed as defining a linear basis and finding coefficients in that basis. The idea is that any signal of the form $A\cos(\omega x + \Psi)$ can be represented in the form $C\cos(\omega x) + S\sin(\omega x)$ where S and C are some coefficients in our basis. This is useful because finding S and C in a linear basis are linear operations and it makes the calculation of amplitude A and phase Ψ simple. The formulas for A and Ψ in terms of S and C are given in the following equations 8, 9.

$$A = \sqrt{C^2 + S^2} \tag{8}$$

$$\Psi = -\arctan\frac{S}{C} \tag{9}$$

This rather superficial treatment is augmented in the textbook section 2.3.

1.4 Space-frequency Analysis

The tools described above are useful for analyzing linear filters but run into trouble if we want to gain understanding about larger images. Running DFT on a large image will give us a very high-level view on what sort of components the image is made of, but it completely hides the spatial information in the original image. Next we are going to look at methods that combine spatial and frequential information.

Space-frequency analysis can be conducted by filtering the image with two localized sinusoidal filters and computing the amplitudes of the filters. The localized sinusoidal filters are called Gabor filters. On a high-level their idea is that the sinusoidal plane wave is weighted by a windowing function, making it very localized. The formula for coefficients C becomes

$$C(x,y) = \sum_x \sum_y W(x-x_0, y-y_0) \cos(\omega_x(x-x_0) + \omega_y(y-y_0)) \tag{10}$$

The formula for coefficients S is similiar. With the coefficients C and S at hand, the amplitudes can be calculated using the formula 8 given in the previous section.

The windowing function $W(x,y)$ in the above formula for Gabor filters is the Gaussian function. The width of the window can be controlled by changing the σ parameter of the Gaussian function. In 2D case there are separate standard deviation parameters for both axes. In practical computation truncated windows are used as the gaussian function falls off close to zero quickly.

1.4.1 Example of Gabor Filtering

In the following example Gabor filters are applied to an image to obtain a spatial amplitude map of a vertical frequency component. The amplitude is computed from the equation $A = \sqrt{C^2 + S^2}$. The Gabor filters are presented in figure 5. The filters have a phase difference of $\pi/2$. The output of the amplitude calculation is presented in figure 6. As expected from the filter images the figure shows that the selected filters resulted in high output where vertical features are present in the input image.

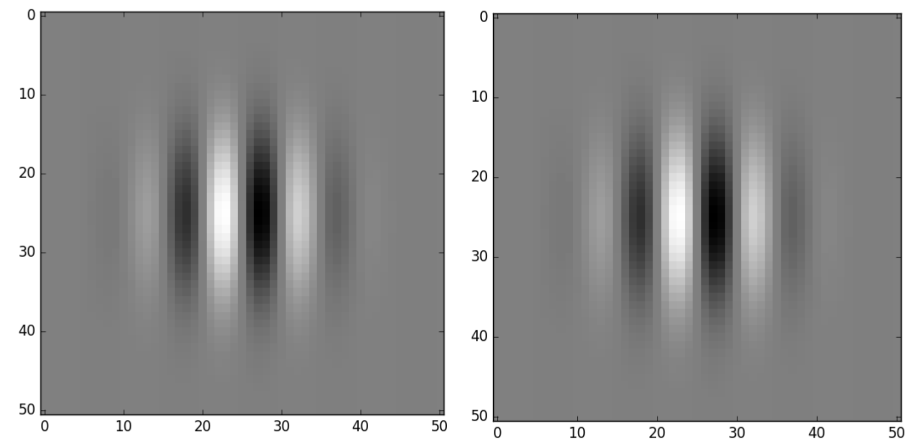

Figure 5: The Gabor kernels to be convolved with the input.

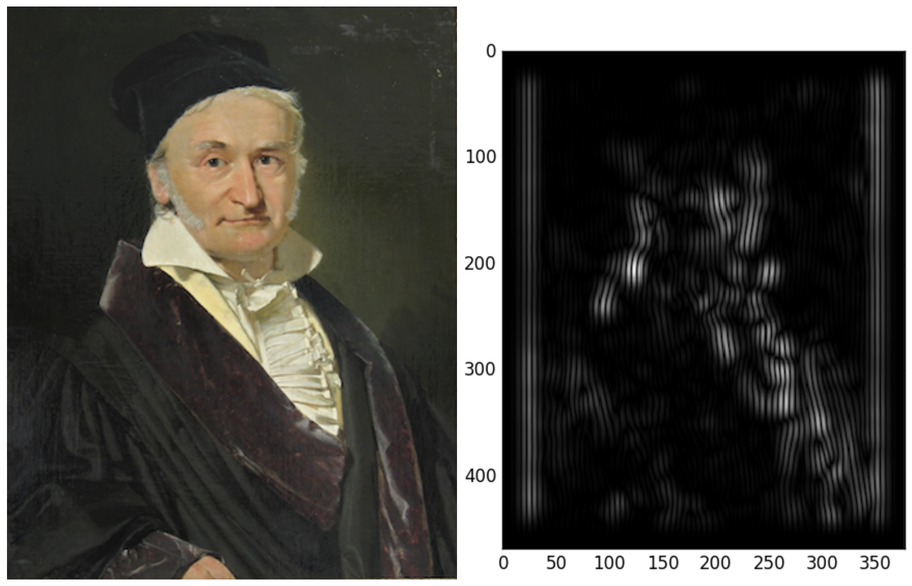

Figure 6: The input image and corresponding spatial amplitude map computed by convolving the gabor kernels with the image and computing the amplitude according to equation 8.