6.003: Signal Processing

Deconvolution

29 April 2021

Review: Implementing Convolution with DFT

Assume that $F[k]$ is the product of $F_a[k]$ times $F_b[k]$.
Find an expression for $f[n]$ in terms of $f_a[n]$ and $f_b[n]$.

$$f[n] = \sum_{k=0}^{N-1} F[k] e^{j\frac{2\pi k}{N}n}$$

$$= \sum_{k=0}^{N-1} F_a[k]F_b[k] e^{j\frac{2\pi k}{N}n}$$

$$= \sum_{k=0}^{N-1} F_a[k] \Big(\frac{1}{N} \sum_{m=0}^{N-1} f_b[m] e^{-j\frac{2\pi k}{N}m} \Big) e^{j\frac{2\pi k}{N}n}$$

$$= \frac{1}{N} \sum_{m=0}^{N-1} f_b[m] \underbrace{\sum_{k=0}^{N-1} F_a[k] e^{j\frac{2\pi k}{N}(n-m)}}_{\text{would be } f_a[n-m] \text{ if } 0 \le n-m < N}$$

$$= \frac{1}{N} \sum_{m=0}^{N-1} f_b[m] f_a[(n-m) \bmod N] \equiv \frac{1}{N} \Big(f_a \circledast f_b \Big)[n]$$

Deconvolving as Inverse Filtering

Since (circular) convolution by $h[r, c]$ is equivalent to multiplying by $H[k_r, k_c]$, we can deconvolve by inverse filtering.

For example, if $h[r, c]$ represents blurring by circular convolution

$$f_o[r, c] = (f_i \circledast h)[r, c]$$

then this blurring can be equivalently thought of as multiplication in the frequency domain:

$$F_o[k_r, k_c] = F_i[k_r, k_c]H[k_r, k_c]$$

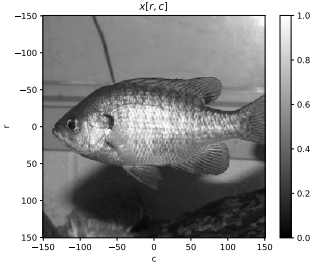
This suggests that if we knew the blurring function, we could recover the original image from the blurred image by inverse filtering

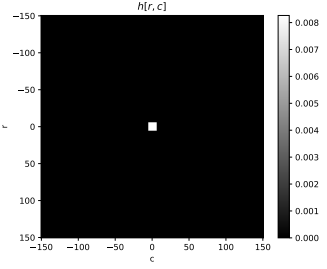
$$F_i[k_r, k_c] = F_o[k_r, k_c]/H[k_r, k_c]$$

provided $H[k_r, k_c] \neq 0$!

Deblurring by Inverse Filtering

Consider blurring an image by circular convolution:





Inverse Filtering

Now, consider reconstructing the original image by inverse filtering:

```
bluegill = png_read('bluegill.png')
kernel = png_read('kernel.png')

B = fft2(bluegill)
K = fft2(kernel)

C = B * K

convolved = ifft2(C)

deconved = ifft2(C / K)
show_image(deconved)
```

What will the resulting image look like?

Inverse Filtering

Now consider a slightly different version of the inverse filtering:

```
bluegill = png_read('bluegill.png')
kernel = png_read('kernel.png')

B = fft2(bluegill)
K = fft2(kernel)

C = B * K
png_write(ifft2(C), 'blurry.png')

blurry = png_read('blurry.png')

deconved = ifft2(fft2(blurry) / K)
show_image(deconved)
```

What will this image look like?

What Happened?

Deconvolution using the `convolved` array directly worked great. So why was the result from the last slide *so bad*, when we loaded the blurry fish from an image rather than using the result of the convolution directly?

Quantization

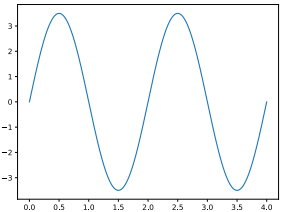
The data we get from loading `blurry.png` are actually different from the data in the `convolved` array, by a very small amount.

This difference arises from the fact that the data are **quantized** when we save our PNG image.

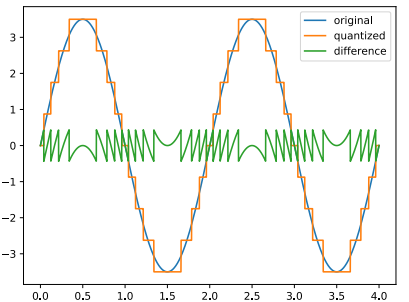
In particular, each pixel's brightness value is represented by an 8-bit number when saving the image; so in the resulting image, there are 256 possibilities for each pixel (and input values are rounded to the nearest of those values).

Quantization

To illustrate this phenomenon, consider the following sine wave:



Quantization introduces some amount of error:



Effect of Error on Deconvolution

Originally, viewed y as being made by circularly convolving some input x with some kernel h:

$$y[r, c] = (x \circledast h)[r, c]$$
$$Y[k_r, k_c] = X[k_r, k_c]H[k_r, k_c]$$

Deconvolution:

$$X[k_r, k_c] = \frac{Y[k_r, k_c]}{H[k_r, k_c]}$$

Now instead, let $\tilde{y}$ be the quantized output:

$$\tilde{y}[r, c] = (x \circledast h)[r, c] + e[r, c]$$
$$\tilde{Y}[k_r, k_c] = X[k_r, k_c]H[k_r, k_c] + E[k_r, k_c]$$

Where $e[\cdot, \cdot]$ represents the error introduced by quantizing our output when saving to PNG.

Deconvolution

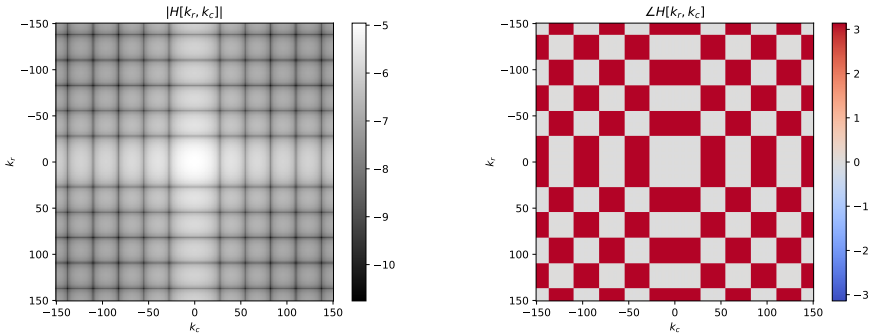
In the presence of this kind of error, what we are actually looking at as the result of our deconvolution is an approximation:

$$X[k_r, k_c] \approx \frac{\tilde{Y}[k_r, k_c]}{H[k_r, k_c]} = \frac{Y[k_r, k_c] + E[k_r, k_c]}{H[k_r, k_c]} = X[k_r, k_c] + \frac{E[k_r, k_c]}{H[k_r, k_c]}$$

Deconvolution

$$\tilde{X}[k_r, k_c] = X[k_r, k_c] + \frac{E[k_r, k_c]}{H[k_r, k_c]}$$

Let's look at $H[\cdot, \cdot]$ (magnitudes on log scale):



The last term in the equation above comes about as a direct result of the quantization error. For what values of $H[\cdot, \cdot]$ is this term maximized?

Deconvolution Strategies

Let's try to minimize the effect of that error term on the output.

Since the effect is worst when $|H[k_r, k_c]|$ is smallest, we could try to get a better output by adding a constant m to the denominator before dividing:

$$X[k_r, k_c] \approx \tilde{X}[k_r, k_c] = \frac{\tilde{Y}[k_r, k_c]}{H[k_r, k_c] + m}$$

Goal: when m is much bigger than $|H[k_r, k_c]|$, the denominator will be appoximately m; when the opposite is true, the denominator will be almost what it was before.

Predict: what will the output look like in the limits $m \to 0$ and $m \to \infty$?

Improvements

$$X[k_r, k_c] \approx \tilde{X}[k_r, k_c] = \frac{\tilde{Y}[k_r, k_c]}{H[k_r, k_c] + m}$$

What is still missing from this approach?

Can we design something better?

Sources of Noise

Noise processes can also lead to errors.

Shot (Poisson) noise: The brightness of a conventional image results from the superposition of photons. Statistical variation in the number of photons gives rise to variability in image brightness. This is most important for relatively dark images (as in telescopy). The average brightness of a pixel can be equivalent to that of 1/10 of a photon. However, every instance of that pixel must contain an integer number of photons – usually 0 or 1, but never 1/10.

Gaussian noise: A variety of processes generate additive noise that can have a Gaussian distribution. One example is electronic noise that is associated with amplification of signals (as when the output of a photosensor is amplified prior to digitization).

Quantization: To reduce the number of bits used to transmit and/or store an image, the signals associated with each pixel are often quantized. For example, 8-bit representations are popular for conventional images.

Other Issues

Worse, we often don't actually know the kernel with which we need to deconvolve!

Real Example: Blurry Photograph

Example (Fergus, et al, 2006): try to remove blur from a photograph

Summary

Deconvolution (circular) can be achieved in the frequency domain by dividing through by the DFT of the convolutional kernel.

Small amounts of noise (or other distortions) can cause *big* problems with naïve deconvolution. We can overcome some of these effects by using something like a *Wiener filter* (to mitigate the noise amplification that occurs when dividing by small numbers in the frequency domain).

Additionally, we often don't know the conovlutional kernel to begin with! But work has been done in this area ("blind deconvolution"), for example:

`https://cs.nyu.edu/~fergus/research/deblur.html`