Lecture 8: Image filtering & compression

At this point we are familiar with two widely used signal processing tools DFT and DWT. We also saw extensions of both methods to images & possibly higher dim data.

Now we will look at two important applications in the context of image processing:

- Filtering with DFT
- Compression with DWT.

- Scipy has an extensive image processing suit which should be enough for 99% your applications.
- Wavelet transform is very recent & as such has significantly less support in major software like Scipy. In otherwords, you need to get your hands dirty.
- You might think DWT is "better/stronger/more flexible" than DFT. But always ask yourself whether DFT is actually insufficient for your application!

8.1 Filtering with DFT & Scipy

One of the most common issues you encounter in dealing with real data is noise!

we've already seen an example of this in L7.

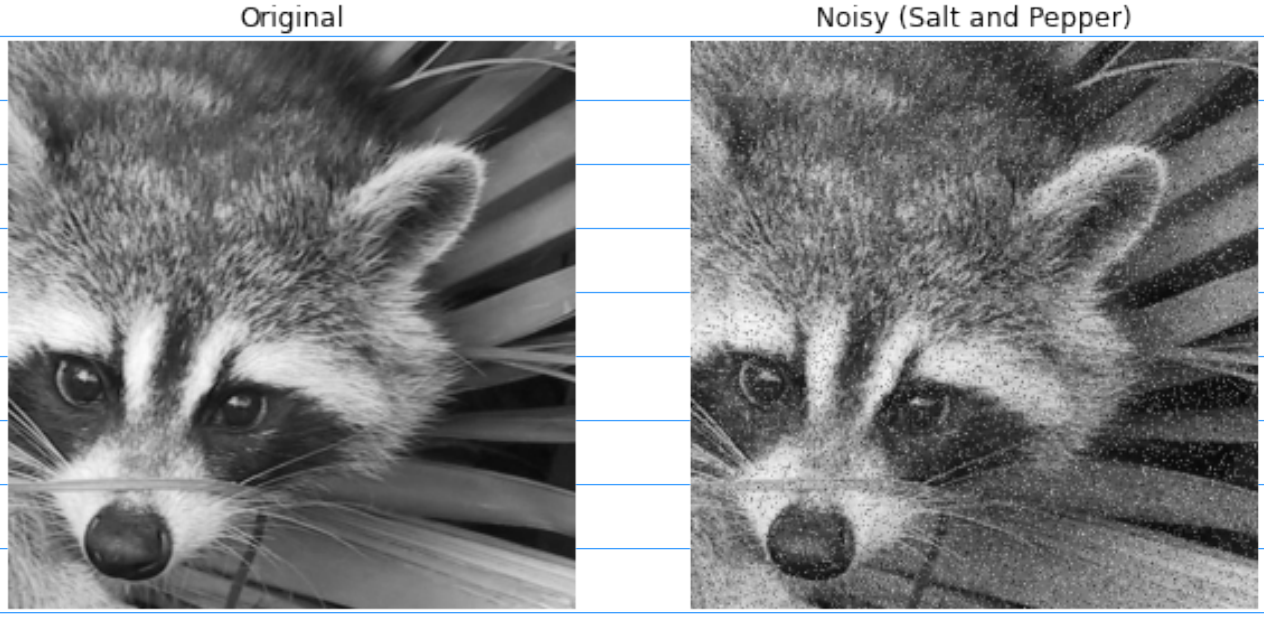
original

OG + Gaussian noise (std 0.5)

OG + more Gaussian noise (std 0.75)

Here noise is added simply by summing the image with a random gaussian matrix of a given standard deviation (amplitude).

Gaussian noise is perhaps the most common model. But it is not the only one. Another common model is the salt & pepper noise where pixels are set to white or black with a given probability.



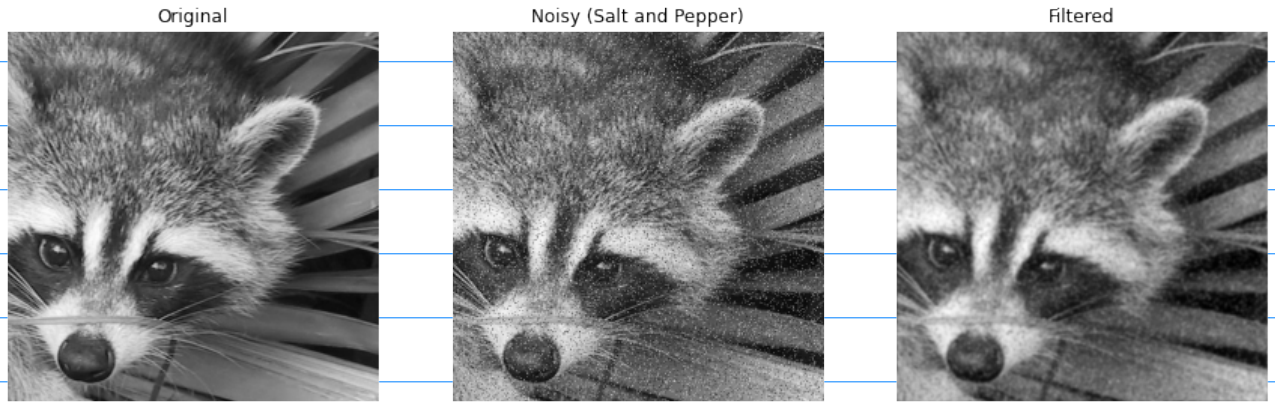
* a quick google search reveals many snippets exist for adding noise.

* Primary use of these functions is to create data for filtering algorithms to be tested on.

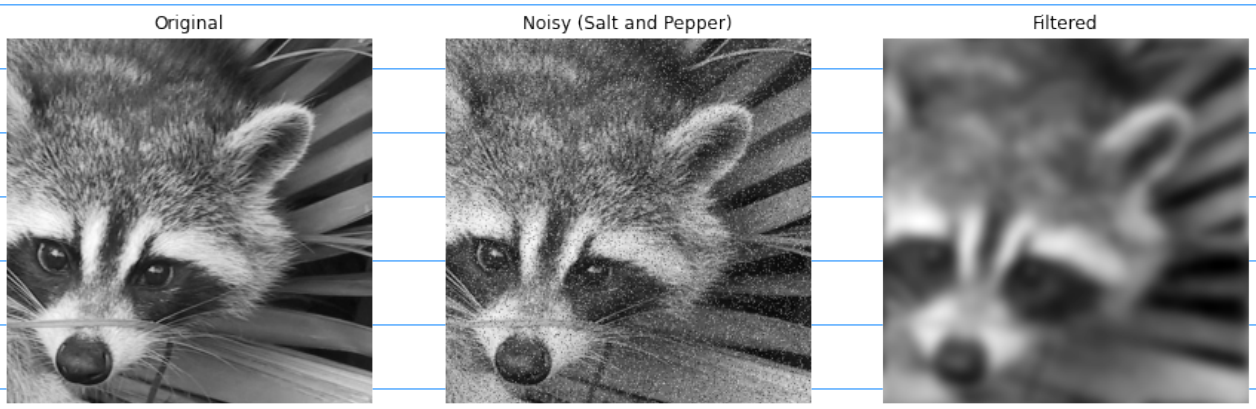
As mentioned earlier, realistic data is noisy so we often need to filter it to get rid of the noise. We already saw an example of this last lecture.

σ=200

σ=50



σ=10



Observe, Gaussian filter is a blurring filter! If we don't dampen high freq modes enough (large σ) then noise is still present. If we over dampen higher freq (small σ) then we blur the image!

* The more noise you have, the more you need to blur.

The notebook shows you how easy it is to code up filters on your own. But, you don't even need to since Scipy already has an extensive set of filtering functions for image processing.

Including:

Gaussian filter

Laplace

Median etc.

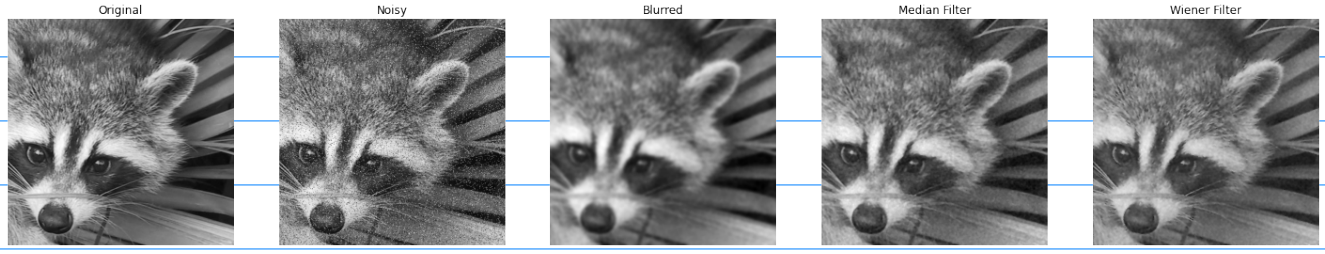
Multidimensional image processing (`scipy.ndimage`)

This package contains various functions for multidimensional image processing.

Filters

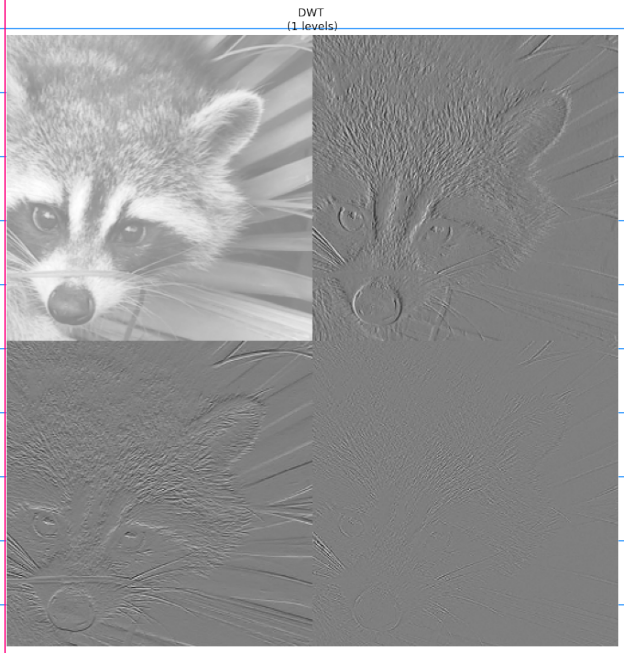
`convolve`(input, weights[, output, mode, ...])	Multidimensional convolution.
`convolve1d`(input, weights[, axis, output, ...])	Calculate a 1-D convolution along the given axis.
`correlate`(input, weights[, output, mode, ...])	Multidimensional correlation.
`correlate1d`(input, weights[, axis, output, ...])	Calculate a 1-D correlation along the given axis.
`gaussian_filter`(input, sigma[, order, ...])	Multidimensional Gaussian filter.
`gaussian_filter1d`(input, sigma[, axis, ...])	1-D Gaussian filter.
`gaussian_gradient_magnitude`(input, sigma[, ...])	Multidimensional gradient magnitude using Gaussian derivatives.
`gaussian_laplace`(input, sigma[, output, ...])	Multidimensional Laplace filter using Gaussian second derivatives.
`generic_filter`(input, function[, size, ...])	Calculate a multidimensional filter using the given function.

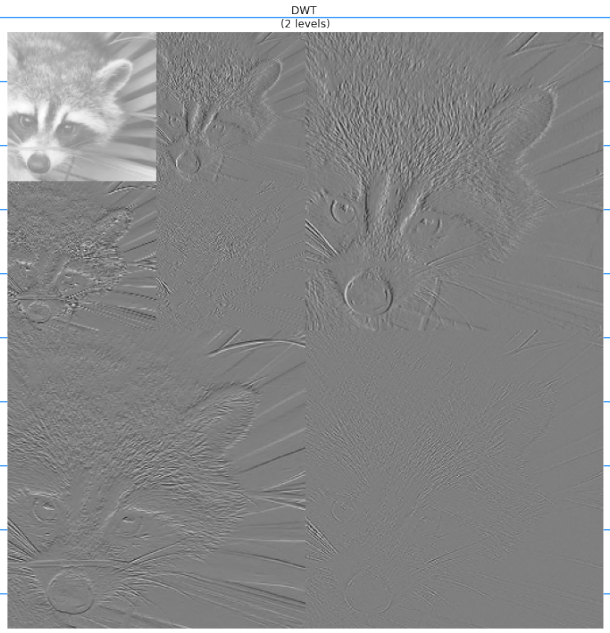
Scipy.Signal also has some more filters.

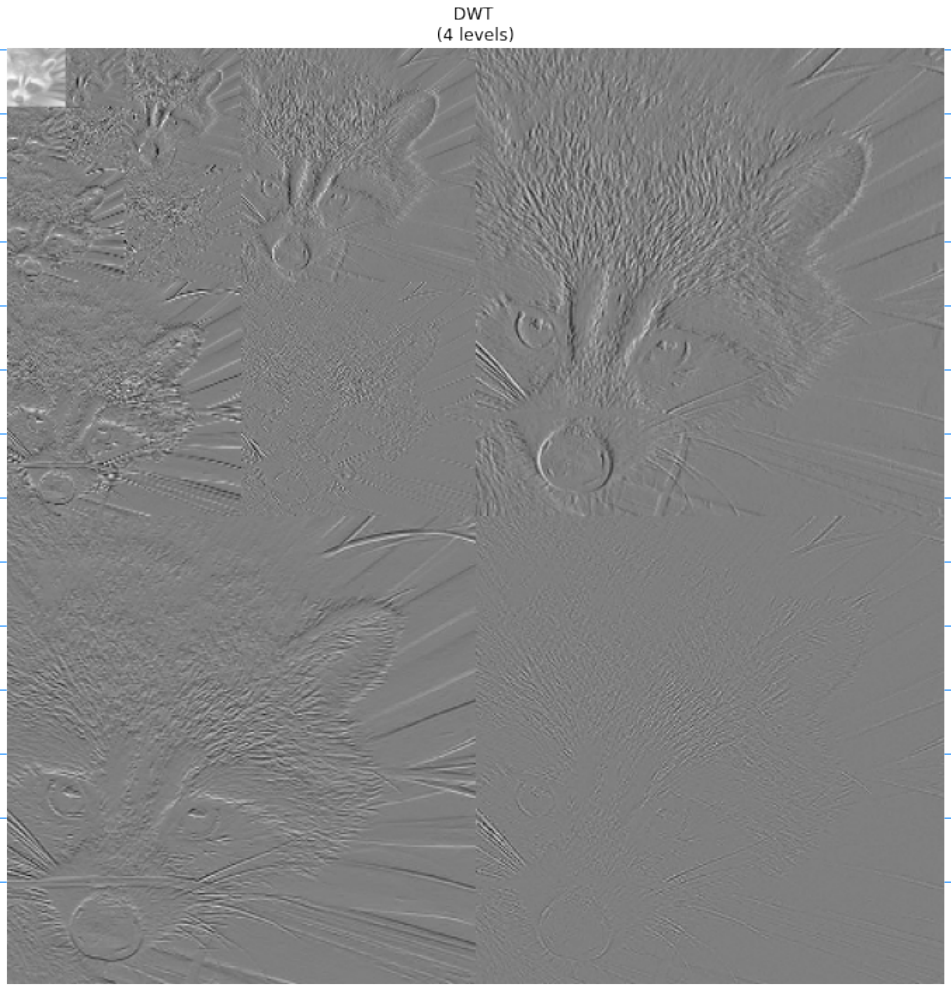


8.2 Image Compression with DWT

Let us consider our example image again & compute its DWT at various levels.



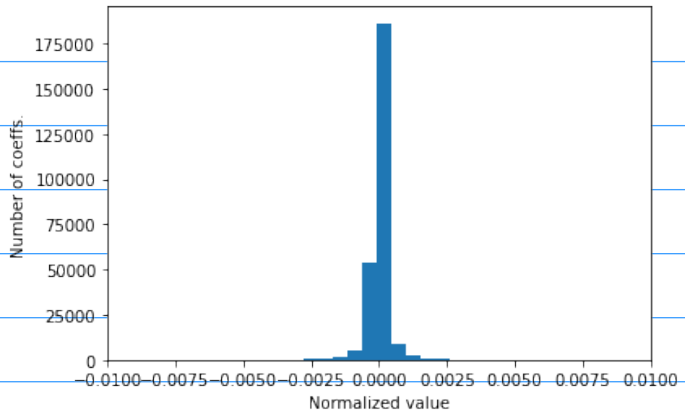




*gray means zero or small coeff.

We can see from the above that most of the detail coeff. are in fact zero or small. We can actually look at the histogram of the full transform.

Observe that the overwhelming majority of coeffs. are almost zero!



This means, we don't really need to keep all of these coeffs. & there is room for significant compression of the image!

A simple approach to this is to only keep a certain percentage of the largest DWT coefficients. This is called image compression (ex JPEG-2000) & all your digital devices use this nowadays!

Idea is very simple, first compute the DWT of your image, image $\xrightarrow{\text{DWT}}$ $\{c_{m,n}\}$

Then only keep the coeffs. $|c_{m,n}| \geq$ threshold.

PyWavelets already has a function for thresholding. all you need to do is give it the cut-off.

more compression

higher quality