Lecture 8
Basic Concepts and Analysis of Images

Jason J. Bramburger

For the next three lectures we are going to discuss image processing and analysis. This is a large area of research inside and outside academia and we will only scratch the surface of possible image analysis techniques. Let's start by talking about applications that use image analysis, then the tasks that are common in image analysis, and finally move to the mathematical techniques that are used.

Applications

1. Digital Photography: I'm guessing you thought of this one immediately when you saw the title of this lecture. When we take pictures, we often want to do things like remove burring from out-of-focus shots, remove red-eye, change the lighting or colouring, or maybe just apply some filters before posting on Instagram.
2. Medical Imaging: This includes things like X-rays, Ultrasound, CAT scans, and MRI.
3. Tomography Outside of Medicine: The basic principles for medical imaging are also used for imaging of other opaque objects. You can find applications of tomography in the geophysical sciences where we wish to detect what is going on underground. These types of imaging may use seismic waves or ultrasound.
4. Infrared/Thermal Imaging: Anything with a temperature above absolute zero emits some radiation. This can be measured to create images in situations when there is insufficient light.
5. Radar and Sonar Imaging: Radar uses radio waves for the detection and ranging of targets. We discussed an application of this at the beginning of this course. Sonar works in a similar fashion, but with underwater acoustic waves.
6. Computer Graphics: There are challenges related to creating life-like computer graphics.

This section will discuss 'image analysis' but you should understand that from a computer's point-of-view an image is just a matrix with some information in it. Therefore, the techniques we discuss here can also be applied to any 2D data. For example, you might have data related to sea surface temperatures. These data might have the same types of problems as images, so some of the solutions are the same. The question then becomes: what are the things you might need to do with an image (or general 2D data)?

Tasks

1. Image Contrast Enhancement: If you take a photo and there isn't much light, it may be difficult to distinguish between different objects. Contrast enhancement is a method that tries to promote or enhance differences in an image in order to achieve sharper contrasts.
2. Edge Detection: In many biomedical and military applications it is very important to detect the edges of objects to see where one object begins and another ends.
3. Image Denoising: This is going to be our main application for this section of the course. Although we will use digital photos for our applications, denoising really becomes important in applications like medical imaging and tomography. Any data that is collected from the real world is likely to have some noise - especially if this data is collected in an indirect way.
4. Image Deblurring: With our application of digital photos in mind, we might be less concerned with noise and more concerned with blurring that comes from being out of focus or from a camera shake. You should note that this is fundamentally different from white noise which affects all frequencies. That is, taking a photo with shaky hands typically has some sort of signature in frequency space based on how your hands are shaking.

5. Inpainting: Sometimes you may have an image file that gets corrupted and you no longer know some of the pixel values. Or maybe you're gathering temperature data at different locations in space and you have a sensor malfunction, thus corrupting your data. We need a way to recover those values based on the values near them. For the temperature data this might be as simple as averaging the temperatures near the bad sensor. For images this may be trickier. If you lose a pixel in a blue sky, it might be easy. But what if you want to crop someone out of a photo and you need to reconstruct what was behind them? Here you would need to continue shapes and edges that go into the reconstructed region.

Mathematical Approaches

1. Morphological Approach: In this approach one can think of the entire 2D domain of an image as a set of subdomains.

2. Fourier Analysis: Just like we did for 1D data, the key idea is to decompose the image into its Fourier components. As we saw, this can help with denoising. It can also help to identify certain features of the image. It can also be useful for image compression. For most images, if you take a Fourier transform, most values will be very close to zero. So that means you can throw away those values and still be able to reconstruct the image. This is the compression technique used in JPEG (actually, it is the discrete cosine transform, which is the real version of the DFT).

   One drawback of the Fourier transform is that if your image has sharp edges in it, the transformed function will look like a sinc function that decays in the Fourier modes like $1/k$. This slow decay means that many Fourier modes won't be that small, meaning you will need to keep a lot of them to accurately reconstruct the image.

3. Wavelets: We already saw that you can use wavelets for image compression and denoising. Wavelets allow for exceptional localization in both the spatial and the wavenumber domains. This means that wide wavelets can be used to represent large slowly-changing regions, while narrow wavelets can represent steep edges. This typically results in an even sparser representation than Fourier analysis gives. The JPEG2000 standard, which was created as an improvement for JPEG, uses wavelets.

4. PDEs and Diffusion Some of the theory of partial differential equations can be used for denoising. We will come back to this later (Section 14.3 of the textbook).

5. Machine Learning: MATLAB has a built-in convolutional neural network that was trained to denoise images. It is also a big area right now to use neural networks to achieve superior resolution. This might include adding more pixels to your image to get greater resolution that the original.

Working with Images in MATLAB

Let's now go over the basics of how MATLAB handles images so that we can apply some techniques in the following lectures. MATLAB has a number of images included with the image processing toolbox. You can read it in using the command imread(). Then you can show it with imshow().

```matlab
I = imread('sherlock.jpg');
imshow(I)
```

Notice that MATLAB represents the image as a $640 \times 960 \times 3$ array. That means the image has 640 pixels in the vertical direction and 960 pixels in the horizontal direction. The 3 is because it is a colour photo so each pixel has a red, green, and blue component. If we want to do analysis on the image, like taking the Fourier transform, we don't really want 3 values for each pixel. Ideally we will have just one. So for this class we will often work with grayscale images instead.

```matlab
Ig = rgb2gray(I);
imshow(Ig)
```

Our particular focus moving forward will be on denoising images. So, let's go ahead and add some noise to our image. We will add a normally distributed random number to each pixel.

```
noiseg = randn(640,960);
Ign = Ig + noiseg;
```

Notice that we get an error. This happens because of the way MATLAB represents images. In particular, the values for each pixel are integers from 0 to 255 - of type uint8. Our noise matrix has values of type double - they are decimals. Since we normally work with doubles in MATLAB, we need to convert the values in the image to double precision. After doing this, we can then add noise in successfully. Note that the numbers get scaled to the interval $[0, 1]$.

```
I = im2double(I);
Ig = im2double(Ig);
Ign = Ig + noiseg;
```

If we want to add noise to the colour image, we need to add noise to each colour component.

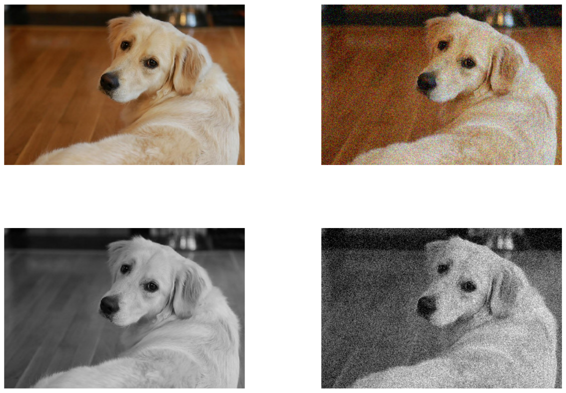
```
noise = randn(640,960,3);
In = I + noise;
```

There is an even easier way to do this though. MATLAB has a built-in routine that adds noise to an image and will even take care of figuring out if it is colour or grayscale. This command is as follows:

```
Ign = imnoise(Ig,'gaussian');
In = imnoise(I,'gaussian');

subplot(2,2,1)
```

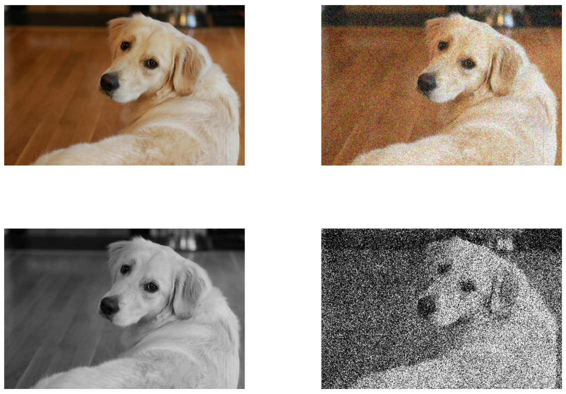
```
imshow(I)
subplot(2,2,2)
imshow(In)
subplot(2,2,3)
imshow(Ig)
subplot(2,2,4)
imshow(Ign)
```



If you want to add more noise, you can specify the mean and variance of the Gaussian distribution. The default values are 0 and 0.01 for the mean and variance, respectively.

```
Ign = imnoise(Ig,'gaussian',0,0.1);
In = imnoise(I,'gaussian',0.1,0.01);

subplot(2,2,1)
imshow(I)
subplot(2,2,2)
imshow(In)
subplot(2,2,3)
imshow(Ig)
subplot(2,2,4)
imshow(Ign)
```



There is another type of noise included with the imnoise() function that might be more realistic for images. It is called *salt and pepper* noise. It randomly chooses some pixels and turns them black and other pixels it turns white. There is a parameter associated with this noise which we will take to be 0.1 (the default is 0.05), so that approximately 10% of the pixels will be corrupted with about 5% black and 5% white.

```
Ign = imnoise(Ig,'salt & pepper',0.1);
In = imnoise(I,'salt & pepper',0.1);

subplot(2,2,1)
imshow(I)
subplot(2,2,2)
imshow(In)
subplot(2,2,3)
imshow(Ig)
subplot(2,2,4)
imshow(Ign)
```