APSC 1001, George Washington University

APSC 1001 Image Processing

Randy Schur

October 20, 2015

1 Introduction

Image processing is a type of signal processing using an image as input. The output will either be another image, or statistics and parameters describing the image. Image processing is used by everyone - Photoshop and Instagram are two common examples, but engineers use it for tasks like medical diagnoses, object identification, and part inspection.

In order to capture an image, we use a camera. A camera is a sensor that can measure the intensity (and sometimes the wavelength) of light over an area. Often, this area is broken up into sections which we call pixels. A grayscale image has an intensity value for each pixel which determines how light or dark it is. Color images are often in RGB format, and have three intensity values at each pixel - one for red, green, and blue wavelengths.

In MATLAB, these intensities are stored in matrices. Each pixel has an intensity value ranging from 0 - 255, where 0 is completely dark and 255 is completely light. In a grayscale image, these correspond to black and white. To import an image, run the following commands:

```
img = imread('starry_night.jpg');
img = imresize(img, .25);
figure
imshow(img)
```

2 Image Manipulation

2.1 Grayscale Images

Notice that the image has a size of $n \times m \times 3$. Since this is an RGB (color) image, it has values for red, green, and blue at each pixel. We can do several manipulations with this image. The first is to turn it into grayscale, which may make it easier to run further image processing. Now, the image has size $n \times m$, because there is only a single intensity value at each pixel. The next step we are going to take is converting the image to black and white using a process called threshholding. Here, we go pixel by pixel through the grayscale image. If the pixel value is below a certain threshhold, we turn it black, otherwise we turn it white. Once reason for converting an image to black and white is that edges become easier to detect.

```
gray_img = rgb2gray(img);
figure
imshow(gray_img)
bw_img = im2bw(gray_img, 0.15);
figure
imshow(bw_img)
```

2.2 RGB Images

Another manipulation that we can do is pulling out the red, green, and blue content of the image. Why might we do this? An analysis of the color content (frequency) of the image can give us a lot of information about the objects in the image. This strategy is particularly used in astronomy. In terms of aesthetics, this can allow us to modify only certain colors within an image. One way we can visualize this frequency content is to use a histogram, which tells us how many pixels there are that have a given intensity.

An RGB image can be thought of as three $n \times m$ matrices stacked up. Our strategy will be to take the first matrix from that stack and separate it out, which gives us all of the red intensities in the image. In order to display just the red parts of the image we will combine, or concatenate, the red intensities with zeros for the green and blue intensities. We can do the same for the other colors.

```
%% colors
red = img(:,:,1);
green = img(:,:,2);
blue = img(:,:,3);
blank = zeros(size(img(:,:,1)));
red_img = cat(3, red, blank, blank);
figure
imshow(red_img)
green_img = cat(3, blank, green, blank);
figure
imshow(green_img)
blue_img = cat(3, blank, blank, blue);
figure
imshow(blue_img)

%% histograms
% imhist(gray_img)
figure
imhist(red)
title('red histogram')
figure
imhist(green)
title('green histogram')
figure
imhist(blue)
title('blue histogram')
```

2.3 Cropping

Sometimes we want to look at only a section of the image rather than the entire image. Since images in MATLAB are stored as matrices, this is extremely simple to do! All we need is to take a subsection of the matrix, which is something we already have practice with using the : operator.

```
%% Cropping
img_section = img(100:200, 1:100, :);
figure
imshow(img_section)
```

3 Filters

Another type of image processing is called filtering. When you talk about adding an Instagram filter, what you are actually doing is manipulating the values for each pixel. When you watch a TV show and someone tells the computer geek to 'zoom in on that and sharpen it up,' they are again using image filtering.

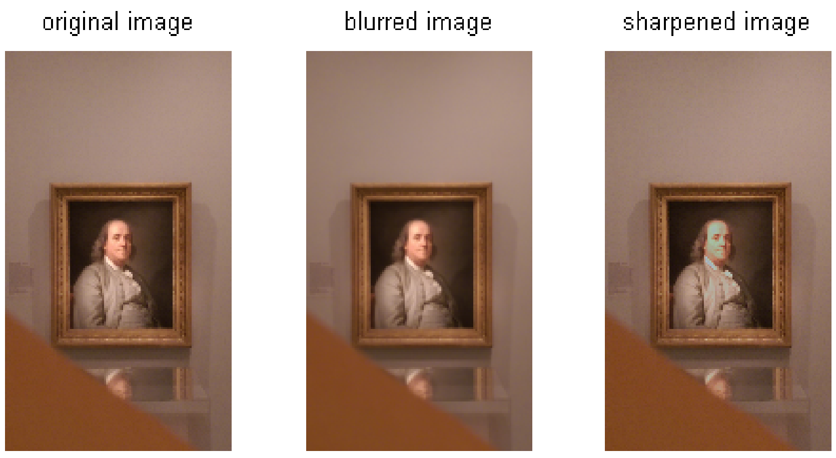
3.1 Unsharpen Filter

One of the most popular filters is called the Unsharpen filter, which does exactly the opposite of its name and sharpens up an image.

Let's assume that we have an image that is a little blurry. Either the camera was out of focus, or it was moved a bit while the image was taken. We are also going to assume that the distortion in the image is Gaussian, which means it follows a specific statistical distribution. This may or may not be an accurate assumption, but often in can be close to true.

The unsharpen filter takes the original image and blurs it using that Gaussian distribution. Then, we subtract the blurred image from the original image, in theory getting rid of some of the blur. Test out the code below to see if you get similar results.

```
%% filters - image sharpening
close all
img2 = imread('bf.jpg');
img2 = imresize(img2, .5);
figure
subplot(1,3,1), subimage(img2)
title('original image')
axis off
psf = fspecial('gaussian', 7, 2);
blurred = imfilter(img2, psf, 'symmetric', 'conv');
subplot(1, 3, 2), subimage(blurred)
title('blurred image')
axis off
img_new = 1.35*img2 - 0.35*blurred;
subplot(1,3,3), subimage(img_new)
title('sharpened image')
axis off
```



It should be noted that applying filters to an image causes us to *lose* information. However, we can often extract insights into the image at the expense of this lost information.

References

[1] MATLAB Documentation, 2015.

[2] Image Processing Presentation, APSC 1001. Matthew Wilkins, 2014.