Computer Vision (600.461/600.661)
Homework 2: Color and Image Processing

Instructor: René Vidal

Due 09/23/2014, 11.59PM Eastern

Student: Gregory Kiar

1. **(20 points) Image filtering, enhancement, and edge detection.**

   (a) Description of MATLAB functions imread, brighten, contrast, histeq, imcontrast and imadjust:

      i. imread: reads in the colour image in the form of a $< width, length, 3 >$ uint8 array. To do more processing on this, it was converted to a grayscale, and double data type.

      ii. brighten: increases the brightness of a grayscale image by shiftingt he values upwards, towards the upper limit of 255. The significance of the shift is determined by the value entered within the function.

      iii. contrast: increases the constrast of the image by adjusting the historgram so that a wider range of the spectrum is covered. The histogram is not shifted, but stretched (or compressed).

      iv. histeq: balances the histogram of the image such that all values that occur in a form which correspond to the inputted histogram, or number of bins given. This is intended to improve contrast.

      v. imcontrast: opens a histogram adjustment tool that allows you to adjust the histogram and view the restults in real time on the image shown above the histogram. This looks like it could potentially be a very useful tool.

      vi. imadjust: remaps the image displayed such that the pixel values with the highest occurances are the extreme intensities (0, 255). This improves the contrast of the image.

   (b) Description of performance of different filters for salt & pepper noise:

      i. Figure 1: normal image

      ii. Figure 2: noisy image with salt & pepper noise

      iii. Figure 3: median filtered image. You can see that the large majority of the noise is completely eliminated, and the rest of the image left intact. Median filtering is seen to be the most effective method of salt and pepper noise reduction in this assignment.

      iv. Figure 4: mean filtered using conv2. You can see that not only is the image noise still present, but the rest of the image is largely blurred. This is a poor solution.

      v. Figure 5: mean filtered using filter2. You can see, as in the previous case, that the noise hasn't been removed but the image has become blurry. This is also a poor solution.

      vi. Figure 6: gaussian filtered with $\sigma = 1.5$. This method, as those completed with averaging, failed to remove the noise and blurred the image. From all of the above, we conclude that median filtering is the best option for salt & pepper noise.

   Description of the performance of different filters for gaussian noise:

      i. Figure 7: noisy image with gausian noise with $\mu = 0, \sigma = 1$

      ii. Figure 8: median filtered image. You can see that there is some removal of the noise, but that the image is still ultimately unclear. This method wouldn't work particularly well for gaussian noise since it often lies within the extremes of the image (not simply black or white).

      iii. Figure 9: mean filtered using conv2. Just like in the case of salt & pepper noise, the averaging mask appears to remove some of the noise effect at the cost of blurring the image.

      iv. Figure 10: mean filtered using filter2. The image appears just like that in Figure 9. The image is blurred, and ultimately still noisy.

      v. Figure 11: gaussian filtered with $\sigma = 1.5$. To my surprise this image still appears very noisy. Where it differs from the mean filtered images is that the contours of the peppers behind the noise remain ultimately intact. I would say that this is the best filtering strategy for gaussian noise, though in practice it is very difficult to know the mean and standard deviation of the noise which makes this filter difficult to apply in automation.

(c) Description of the performance of different edge detection methods as used on 'pepper.png':

i. Sobel: The Sobel operator observes most of the edges in the image, but misses some of the softer transitions between the background and larger foreground image of similar colour (large bell pepper on the left of the image). The operator is also very sensitive to changes within the foreground, including things like glare.

ii. Prewitt: The Prewitt operator appears almost identical to the sobel operator. One difference I noticed was the handling of a small piece of noise above the highest elevated pepper, the prewitt operator detected this false positive slightly less than the sobel had.

iii. Roberts: The Roberts operator appears to be a less sensitive method than the sobel or prewitt methods. Low frequency changes, such as that in the first large pepper mentioned in the sobel comments, remain almost completely undetected. That said, this method is also still very sensitive to false positives in the sense of glare on some of the front lying peppers.

iv. Laplacian of Gaussian: The laplacian of gaussian methos is clearly the most sensitive method of the bunch. This can be advantageous because if more edges are detected than exist, the result can be filtered; in contrast if edges are missed they cannot be recovered as would be the case in the previous 3 methods tested. This method, as implied by the sensitivity, is also sensitive to noise.

v. Canny: The Canny operator seems to be the best middle ground between all of the options tested. This method is very sensitive to edges of the peppers, including low frequency changes, but is less sensitive to noise than the Laplacian of Gaussian method.

2. **(30 points) Color-based face detection.**

(a) Description of MATLAB functions colormap, hsv2rgb, rgb2gray, rgb2hsv, rgb2ntsc, and rgb2ycbcr:

i. colormap: this function either returns the current colormap or allows you to set a new colormap based on the way you wish your image to be displayed, whether it's remapping brightness, or colours themselves.

ii. hsv2rgb: this function can be either used to convert an hsv image to an rgb image, or an hsv colormap to an rgb colormap.

iii. rgb2gray: this function behaves similarly to hsv2rgb, except it either converts or maps from an rgb format to a grayscale image. There happens to be an advanced mode as well for this function I saw in Mathworks documentation that performs this operation on the GPU for better parallel processing.

iv. rgb2hsv: this function is the exact reverse function of hsv2rgb. It takes an rgb image or map and produces the hsv equivalent.

v. rgb2ntsc: this function is similar to those above, where it converts from rgb to ntsc (or yiq) color mapping systems.

vi. rgb2ycbcr: this function is also similar to those above, where it converts from rgb to ycbcr color system. These functions are all valuable depending on what type of color system you're interested in processing images in.

(b) Image visibility:

i. Red: The red image shows the skin colour fairly simiarly to a grayscale version of the entire image. We can clearly see the faces here

ii. Green: The green image shows the face colour intensely as well, and perhaps distinguishes fine differences in the facial colour more than the red image. Also, in this image the face colours are quite distinguished from the other colours.

iii. Blue: The blue image is similar to the green image, though the blue face colours blend into the other colours in the image more, such as Ross' shirt.

iv. Hue: The hue image actually shows a complete absense of faces. This is interesting as it could perhaps be used as a negative to detect where faces are. Unfortunately, the background is also included in this.

v. Saturation: The saturation has the faces showing very differently from the rest of the image. The intensity of them is quite distinct in this image compared to everything else.

vi. Value: The value image actually looks almost identical to the red image mentioned earlier. The features of the image are all easily distinguishable.

(c) The method developed to remove every part of the image except for the face using RGB was created by trial and error, as suggested. It did not completely remove all other components, but I think that though my solution was certainly not optimal, there will most certainly be some left over colours if we are only detecting faces using RGB.

(d) The method developed to remove every part of the image except for the face using HSV was created by trial and error, as suggested. Much like above, it was not perfect, though it was quite successful by thresholding hue and saturation values. I anticipate this being a better method in the general case.

(e) After analyzing the different Friends image with the exact algorithms I developed before, same numbers and all, I realized that my filters no longer worked almost at all. I'm not sure if this was the point: to see that using RGB and HSV colour based algorithms for facial recognition are impractical, or if I just went about solving the challenge in a way that didn't transfer well between images. After playing around with the methods more I did come to the conclusion however that the HSV method was superior to the RGB. I think that this is because the RGB method relies too heavily on the background or remaining portions of the foreground to work effectively. The HSV method, in turn, is able to more reliably observe faces through the saturation value even though the hue value was less useful for this particular image.