IMGS 682 Spring 2018 - Homework 2: Image Enhancement, Color, Filtering, and Morphological Operators

Nilesh Pandey

Due: See MyCourses

Problem 1 - Image Enhancement

Histogram equalization is a method used for enhancing the contrast of an image. See Section 3.1.4 of Szeliski or look it up on Wikipedia. For these problems, use 256 histogram bins.

Theory

Histogram equalization is used to enhance the contrast of an image, but not necessarily it should always work. Their are cases where it fails miserably and makes it worse. The image with high peak indicates an image with low contrast, histogram is stretched all over range to improve contrast.

Normalized Histogram Normalized Histogram is converting the actual histogram between small range. The values re typically between 0-1

Cumulative Histogram The cumulative histogram is a variation of the histogram in which the vertical axis gives not just the counts for a single bin, but rather gives the counts for that bin plus all bins for smaller values of the response variable

Part 1 (2 points)

Download frog.png display the normalized histogram and cumulative histogram of the pixel values for each image.

Solution:

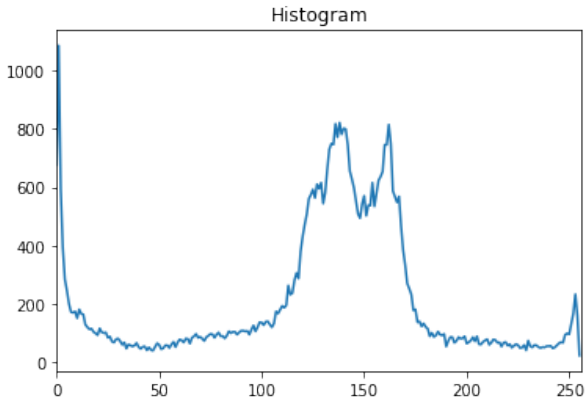

Figure 1: Histogram

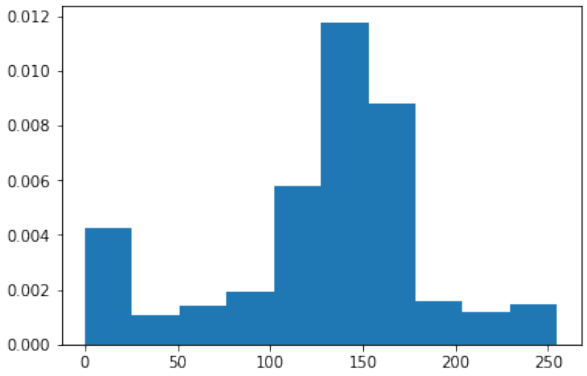

Figure 2: Normalized Histogram

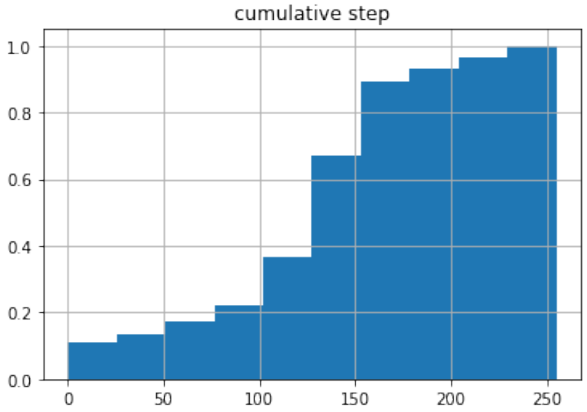

Figure 3: CDF Histogram

Part 2 (3 points)

Implement linear contrast stretching by subtracting the minimum pixel value and dividing by the new maximum pixel value, i.e.,

$$\mathbf{x}' = \frac{\mathbf{x} - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})}.$$

Apply linear contrast stretching to jupiter-gray.png histogram is stretched all over range to improve contrast.. Display the image before and after applying the transformation. Display the normalized cumulative histogram of the pixel values for the image after applying linear contrast stretching.

What happens when you try to apply linear contrast stretching to frog.png instead? Explain.

Solution:

Contrast stretching Contrast Stretching is histogram normalization, It does this by stretching the range of intensities values it conatins to the desired range. It is different from histogram equalization formula is img-min/max-min

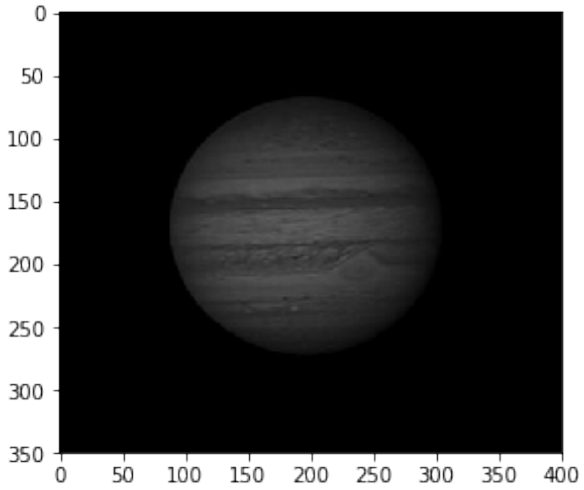

Figure 4: Image

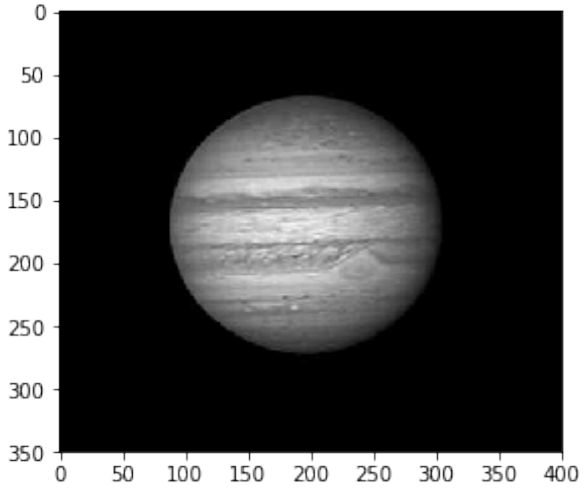

Figure 5: After contrast stretching

We apply contrast stretching to the frog image, and but there is no change in the image contrast, the reason is because the contrast stretching works on mapping of values from the image to all range. Like, if we have min intensity of 84 in the given image, then it would map it to zero.

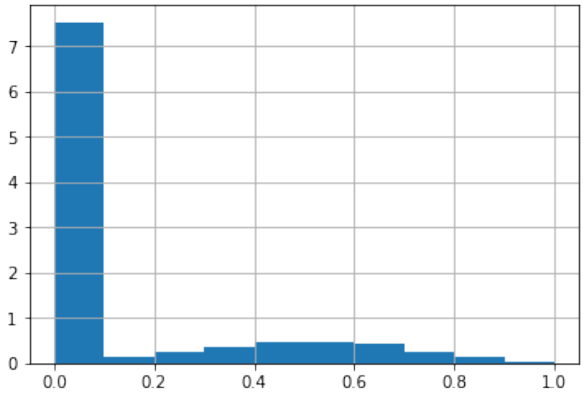

Figure 6: Normalized histogram

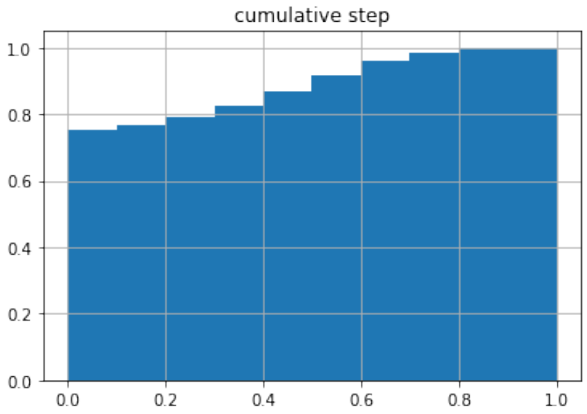

Figure 7: CDF histogram

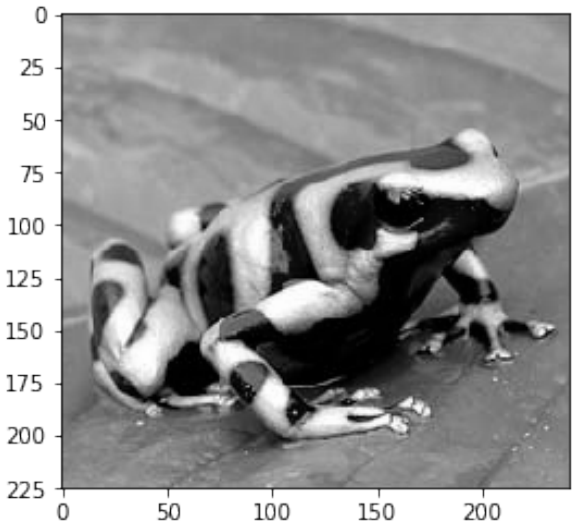

Figure 8: Frog After Contrast Stretching

Image hasn't changed because it is already spread out through whole region, thus using contrast stretching won't make any visible change.

Part 3 (3 points)

One form of image enhancement is to alter the image's gamma value. This can be done by raising an image's pixels to a power called gamma (i.e., γ). For this problem, make sure your pixels are scaled to be between 0 and 1. Apply gamma values of 0.5, 1, and 2 to frog.png. Display the results.

Display the normalized cumulative histogram of the pixel values for the image for the three values of gamma. Use subplots to do this so that they are displayed as a single figure. Make sure to give each subplot a title.

How does changing gamma affect the cumulative histogram?

Solution:

Gamma Correction Gamma correction, or often simply gamma, is the name of a nonlinear operation used to encode and decode luminance. Each pixel in an image has brightness value associated to it called luminance. This value range in between of 0-1, where 0 is complete dark, 1 bright. Different devices don't capture luminance correctly, neither display it correctly. So, we use gamma correction to correct the values. Being non linear function it is just power raised to the image.

For gamma correction the image should have been normalized.

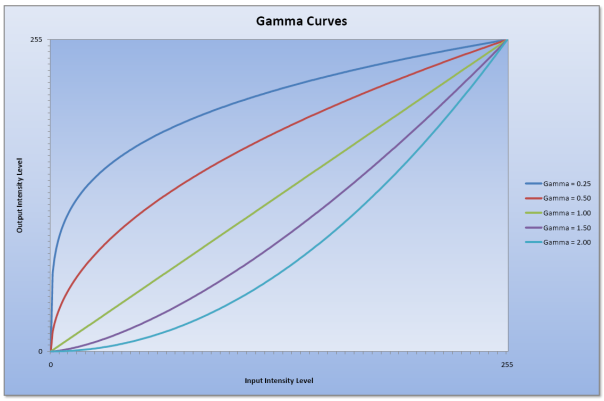

Figure 9: Gamma Nonlinearity

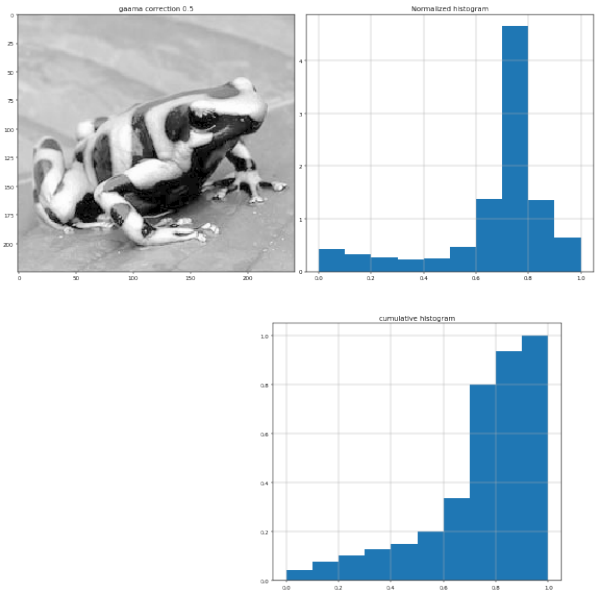

Figure 10: Gamma value of 0.5

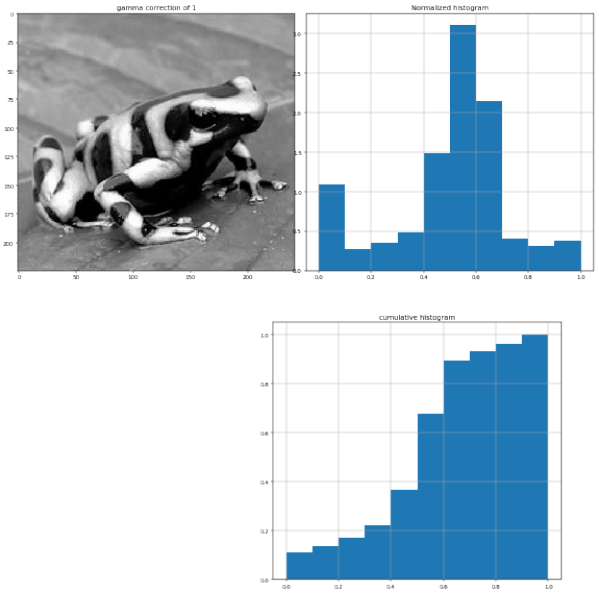

Figure 11: Gamma value of 1

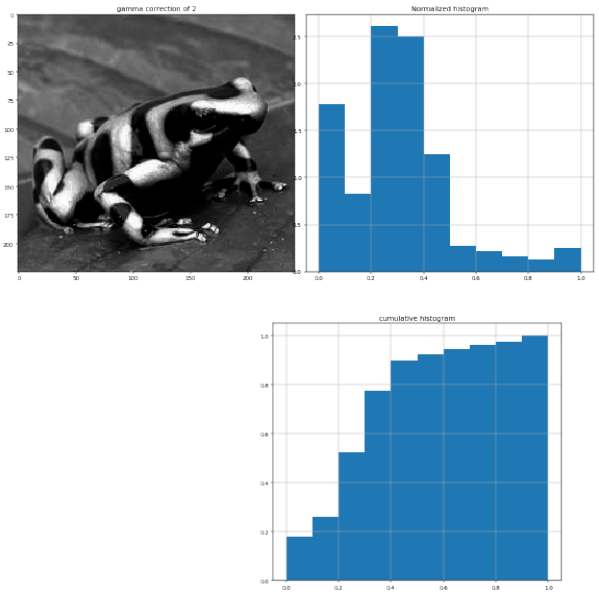

Figure 12: Gamma value of 2

Part 4 (6 points)

Implement a function that takes in a monochrome image, performs histogram equalization on it, and then returns the transformed image. Make sure your pixels are from 0 to 255, and use 256 histogram bins.

Apply the histogram equalization function you wrote to frog.png and show the image that is produced. Also display the normalized cumulative histogram of the transformed image's pixel values.

Solution:

Histogram Equalization images have their pixel values confined to certain range. Example, brighter image will have all pixels confined to high values. A good image should have pixels from all regions of the image.

Histogram Equalization is more sophisticated technique compared to normalization for improving contrast. Histogram equalization is reliable technique compared to contrast stretching.

- Advantage - Histogram Equalization is invertible, we can recover original histogram from the histogram equalization.
- Disadvantage - It may increase the background noise, relatively decreasing usable data from using it.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('/home/n/images/frog.png',0)

def hist_e(img):
    hist,bins = np.histogram(img.flatten(),256,[0,256])

    cdf = hist.cumsum()
    cdf_normalized = cdf * hist.max()/ cdf.max()
```

```
plt.plot(cdf_normalized, color = 'b')
plt.hist(img.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'), loc = 'upper_left')
plt.show()
cdf_m = np.ma.masked_equal(cdf,0)
cdf_m = (cdf_m - cdf_m.min())*255/(cdf_m.max()-cdf_m.min())
cdf = np.ma.filled(cdf_m,0).astype('uint8')
plt.hist(cdf[img].ravel(), bins=10, normed=True, stacked=True)
plt.grid(True)
plt.show()
```

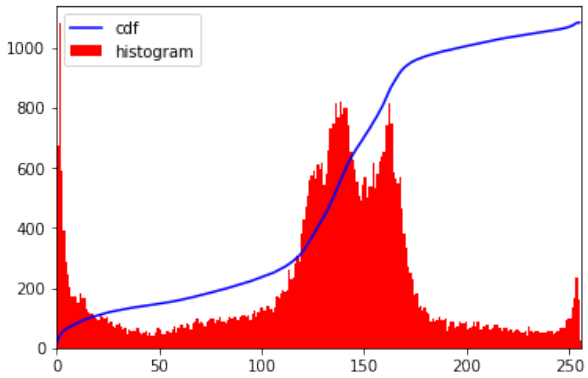

Figure 13: before applying histogram equalization

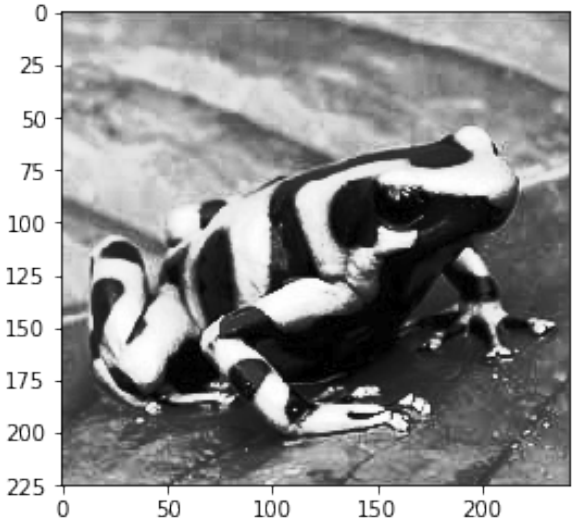

Figure 14: Histogram equalization

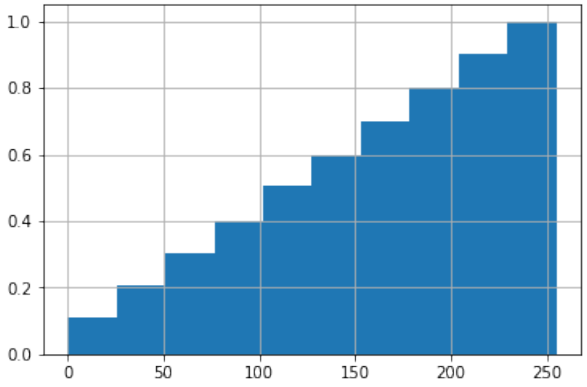

Figure 15: Normalized CDF after his equalization

Problem 2 - Colorspaces

Let $\mathbf{x} = \begin{bmatrix} r & g & b \end{bmatrix}^T$ be a pixel in RGB space. The transformation from RGB space to HSI space is given by (Gonzalez and Woods, 1992)

$$I = \frac{1}{3}(r + g + b)$$

$$H = \begin{cases} \frac{360^\circ - \alpha}{360^\circ} & \text{if } b > g \\ \frac{\alpha}{360^\circ} & \text{otherwise} \end{cases}$$

$$S = 1 - \frac{\min(r, g, b)}{I}$$

where

$$\alpha = \cos^{-1}\left(\frac{0.5\left((r-g) + (r-b)\right)}{\sqrt{(r-g)^2 + (r-b)(g-b)}}\right)$$

Part 1 (3 points)

Make the following substitution for r, g, and b into the equation for computing Hue from RGB pixels and simplify:

$$r \leftarrow (\beta r)^\gamma, g \leftarrow (\beta g)^\gamma, b \leftarrow (\beta b)^\gamma$$

What does this tell you about how hue is affected by changes in brightness and gamma?

Solution:
PUT SOMETHING HERE

We substitute the given values for r,b,g in the following equation

$$I = \frac{1}{3} \times ((\beta r)^\gamma + (\beta g)^\gamma + (\beta b)^\gamma) \tag{1}$$

$$I = \frac{(\beta)^\gamma}{3} \times ((r)^\gamma + (g)^\gamma + (b)^\gamma) \tag{2}$$

$$S = 1 - \frac{min((\beta r)^\gamma, (\beta g)^\gamma, (\beta b)^\gamma))}{I} \tag{3}$$

$$S = 1 - \frac{min((\beta r)^\gamma, (\beta g)^\gamma, (\beta b)^\gamma))}{\frac{(\beta)^\gamma}{3} \times ((r)^\gamma + (g)^\gamma + (b)^\gamma)} \tag{4}$$

$$H = \begin{cases} \frac{360^\circ - \alpha}{360^\circ} & \text{if } b > g \\ \frac{\alpha}{360^\circ} & \text{otherwise} \end{cases}$$

$$\alpha = cos^{-1}\left(\frac{0.5(2(\beta r)^\gamma - (\beta g)^\gamma - (\beta b)^\gamma)}{\sqrt{(\beta r)^{2\gamma} + (\beta g)^{2\gamma} + (\beta b)^{2\gamma} - (\beta r)^\gamma(\beta g)^\gamma - (\beta b)^\gamma(\beta r)^\gamma - (\beta b)^\gamma(\beta g)^\gamma}}\right) \tag{5}$$

$$\alpha = cos^{-1}\left(\frac{0.5(2(r)^\gamma - (g)^\gamma - (b)^\gamma)}{\sqrt{(r)^{2\gamma} + (g)^{2\gamma} + (b)^{2\gamma} - (r)^\gamma(g)^\gamma - (b)^\gamma(r)^\gamma - (b)^\gamma(g)^\gamma}}\right) \tag{6}$$

Part 2 (2 points)

We want to increase the contrast of an image. Download dune.png and apply histogram equalization to each chromatic channel (R, G, and B) independently and show the image side-by-side with the original. What does this procedure do to the image's hue and saturation? Was this a good way to enhance the contrast alone?

Solution:

Solution:

Listing 1: Histogram Equalization

```
def hist_e(img):
    hist,bins = np.histogram(img.flatten(),256,[0,256])

    cdf = hist.cumsum()
    cdf_normalized = cdf * hist.max()/ cdf.max()

#       plt.plot(cdf_normalized, color = 'b')
#       plt.hist(img.flatten(),256,[0,256], color = 'r')
#       plt.xlim([0,256])
#       plt.legend(('cdf','histogram'), loc = 'upper left')
#       plt.show()
    cdf_m = np.ma.masked_equal(cdf,0)
    cdf_m = (cdf_m - cdf_m.min())*255/(cdf_m.max()-cdf_m.min())
    cdf = np.ma.filled(cdf_m,0).astype('uint8')
    return cdf,cdf_normalized

q,_ = hist_e(cv_rgb[:,:,0])
q = q[cv_rgb[:,:,0]]
w,_ = hist_e(cv_rgb[:,:,1])
w = w[cv_rgb[:,:,1]]
e,_ = hist_e(cv_rgb[:,:,2])
e = e[cv_rgb[:,:,2]]
res = np.dstack((q,w,e))
```

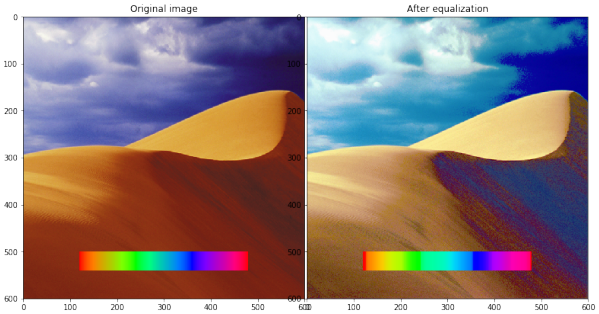

Figure 16: After Applying Equalization

Histogram equalization seems to working perfectly for blue and and shades of sands, but if we notice the corners we can see that the high contrast corners of blue and black, are not perfectly balanced. This has lead to wrong colors in high contrast places.

Part 3 (10 points)

Find a toolbox to convert between RGB and HSI, RGB and HLS, or RGB and HSV. HSV and HLS are a similar colorspaces, but they use different calculations for image brightness. Alternatively, you could implement the transformations from RGB to HSI and the inverse, but it is easy to make errors and I do not recommend this.

Download dune.png and convert it to HSI, HSL, or HSV space, apply histogram equalization to the I, L, or V channels, and then convert the image back to RGB color space. Show the result before and after the transformation side-by-side. Comment on how the image compares to when we normalized each channel individually.

Tip: You may use a toolbox function to convert from RGB to HSV and vice versa.

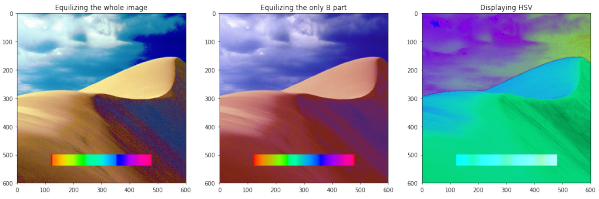

Figure 17: RGB Equalization and Conversion

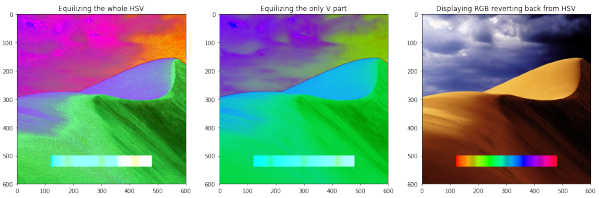

Figure 18: HSV Equalization and Conversion To RGB

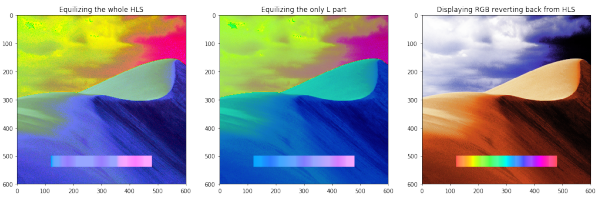

Figure 19: HLS Equalization and Conversion To RGB

On performing different color space conversion, and channel equalization we can see some improvements for example from converting HSV to RGB, we can the image has more contrast, and the color is balanced

but from converting HLS to RGB we can see that the image has more white tone and unbalanced color segregation.

Problem 3 - Color Constancy

Part 1 (3 points)

The Gray World algorithm makes the assumption that the world is gray, and it uses this idea to normalize an image's color channels. Let $\bar{r}$ be the average value for the red channel, $\bar{g}$ be the average value for the green channel, and $\bar{b}$ be the average value for the blue channel. In the variant we will use, then the red channel is normalized by multiplying it by $\bar{g}/\bar{r}$ and the blue channel is normalized by multiplying it by $\bar{g}/\bar{b}$. The green channel is customarily left alone.

Implement Gray World as a function, test it on plate.jpg, and display the image. Show the mean values for the red, green, and blue channels before and after applying Gray World. Before computing the mean, ensure your pixels are between 0 and 1 instead of 0 and 255. Qualitatively, how well did Gray World work?

Solution:

Listing 2: Gray World

```
def grayworld(img):
    r = np.average(img[:,:,0])
    g = np.average(img[:,:,1])
    b = np.average(img[:,:,2])
    av = np.average(img)
    gr = np.minimum((g/r)*img[:,:,0],255)
    gg = (1)*img[:,:,1]
    gb = np.minimum((g/b)*img[:,:,2],255)
    f = np.zeros_like(img)
    f[:,:,0] = gr
    f[:,:,1] = gg
    f[:,:,2] = gb
    img1 = (f - np.min(f))/(np.max(f) - np.min(f))
    return img1
```

Figure 20: Before applying Gray World

We can see a problem in the plate image displayed, that the red channel is too dominant. To improve and balance the color among the channel we use gray world, where there is an underlying assumption that the system is through gray eyes.

Mean Color Values Before Gray World: R=125.64175, G=104.10, B=102.648

Mean Color Values After Gray World: R=0.406348, G=0.406, B=0.406

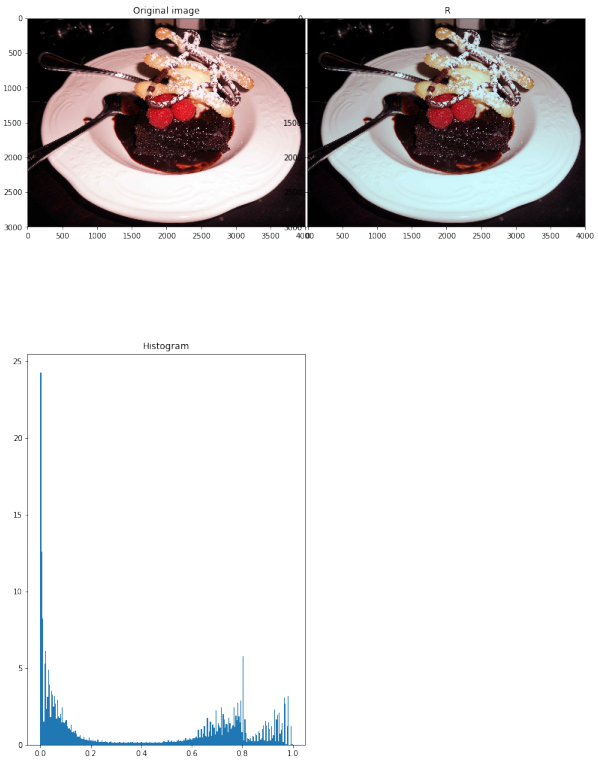

Figure 21: After applying Gray World

Part 2 (4 points)

What are some problems with the Gray World algorithm? What are some situations in which it will fail to correctly normalize the color? Verify your argument by finding an image with a property that will make Gray World fail and display the result before and after applying Gray World. We have modified version of gray world with white balancing for this specific task for color balancing in an image.

Solution:
Gray World fails when it has to deal with the white balancing. Lets analyze the fig.21, we can see after

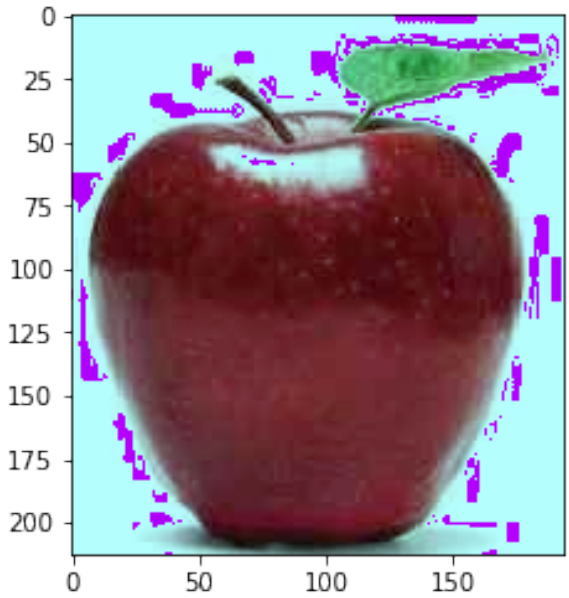

Figure 22: Gray world fails

applying gray world, the whites are blue and purple, and the tower color is also affected. We can say gray world fails when there is white shade in the image.

Problem 4 - Linear Filtering

Part 1 (2 points)

Write a function called rgb2gray that takes as its input an RGB image and outputs a monochrome image simply by taking the mean of the red, green, and blue channels.

Convert empire.jpg to grayscale using your function. Save it to disk and call it empireGray.jpg.

Listing 3: RGB to GRAY

```
def rgbtogray(img):
```

```
        q = cv_rgb[:,:,0]
    # q = q/np.average(q)
    w = cv_rgb[:,:,1]
    # w = w/np.average(w)
    e = cv_rgb[:,:,2]
    # e = e/np.average(e)
    g = np.zeros_like(cv_rgb)
    g[:,:,0] = q
    g[:,:,1] = w
    g[:,:,2] = e

    s = (q+w+e)/3
    return s

cv2.imshow('image',s)
cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.imwrite("empireGray.jpg",rgbtogray(img))
```

Figure 23: Empire Gray is saved as Gray after using floating point image

Part 2 (10 points)

In this problem you will implement two versions of 2-D convolution. Implement a function conv2Spatial that does 2D convolution in the spatial domain and implement a function called conv2FFT that does the convolution in the Fourier domain. Use zero-padding to handle the borders for both functions. Both functions will take a 2D monochrome image and a 2D spatial filter as inputs, and return the image after doing the 2D convolution.

You may wish to read this article: `https://en.wikipedia.org/wiki/Kernel_(image_processing)`

Hints: Make sure you rotate the filter correctly for each version, and I'd debug with a random asymmetric filter. You can verify whether your solution is correct by comparing its output to the output of a built-in 2D convolution function (use OpenCV).

Test your two methods by convolving empireGray.jpg with this filter:

$$\mathbf{D} = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}.$$

Report the mean squared error between the two filtered images.

Display the results side-by-side for both methods after contrast stretching each image.

Solution: PUT A SIDE-BY-SIDE FIGURE HERE
Mean Squared Error: 19452.82

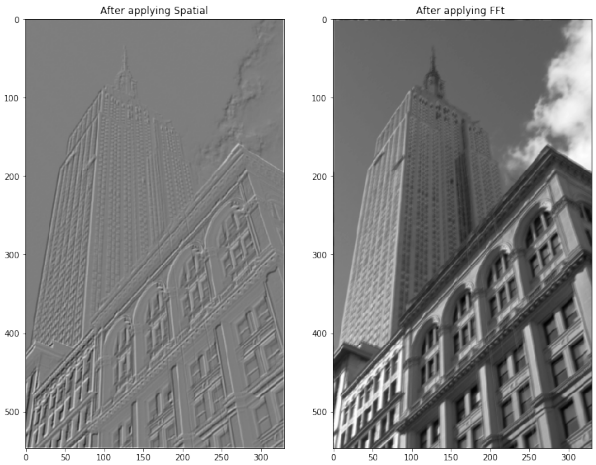

Figure 24: guassian and FFT

Listing 4: Conv2D and FFT

```
from skimage.exposure import rescale_intensity
def spatial_(img,filter_,P):
    HH,WW = filter_.shape
    H,W,C = img.shape
    pad_x = np.pad(img, ((P,), (P,), (0,)), 'constant')
    h = int(1 + (H + 2 * P - HH) / 1)
    wi = int(1 + (W + 2 * P - WW) / 1)
```

```
    out = np.zeros_like(img,dtype="float32")

    # for n in range(C):
    for g in range(h):
        for t in range(wi):
            for n in range(C):
                out[g, t,n] = np.sum(pad_x[g:g + HH, t:t + WW,n] * filter_)
    return out

def confft(img,fi):
    fi = np.flipud(np.fliplr(fi))
    q = np.fft.fft2(img)
    fi = np.fft.fft2(fi)
    out = np.zeros_like(img)
    out[0:3,0:3] = fi
    r = np.fft.fft2(out)
    o = q*r
    t = np.fft.ifft2(o)
    t = np.real(t)

    output = rescale_intensity(t, in_range=(0, 255))
    output = (output * 255).astype("uint8")
    return t
//Reference http://machinelearninguru.com/computer_vision/basics/convolution/image_convolution_1.html
```

Part 3 (3 points)

Time conv2Spatial and conv2FFT using a 9×9 random filter and report how long each of the methods takes on empireGray.jpg. To do this, run each method 10 times and report the mean in seconds. **Hint:** For Python, see this: `http://stackoverflow.com/questions/1557571/how-to-get-time-of-a-python-program-exe`

Solution:
conv2Spatial: 0.8420894145965576 seconds
conv2FFT: 0.398831844329834 seconds

Listing 5: Conv2D and FFT

```
start = time.time()
plt.figure(figsize=(18,19),dpi=70)
plt.imshow(spatial_(cv_rgb[:,:,0],filter_,1),cmap="gray")
end = time.time()
print("Time taken for spatial conv to run is {}".format(end-start))

start = time.time()
plt.figure(figsize=(18,19),dpi=70)
plt.imshow(conf(cv_rgb[:,:,0],filter_),cmap="gray")
end = time.time()
print("Time taken for fft to run is {}".format(end-start))
```

Part 4 (3 points)

Use one of your 2-D convolution functions to convolve empireGray.jpg with the 3×3 Gaussian filter given by

$$\mathbf{G} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix},$$

and also convolve empireGray.jpg with a 3×3 normalized Box (averaging) filter given by

$$\mathbf{B} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Show the results side-by-side. Comment on how the output differs for the two types of 'blur' filters.

Solution:

Figure 25: Gaussian and averaging filter

Gaussian filter is better option if want to reduce noise in frequency domain, and Averaging filter is good at reducing random noise in spatial domain. At the same time Averaging Filter is worst for frequency domain analysis. Mean filter is worst of low pass filters, since it passes high pass filter and even stops low pass filter. Being the simplest of all the filter it is first choice for reducing random noise.

Part 5 (4 points)

Sobel filters can compute the x- and y-gradients of an image. The sobel filter for the x-gradient is given by

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}.$$

The filter for the y-gradient is given by $\mathbf{G}_y = \mathbf{G}_x^T$.

After convolving an image with both of these filters, let the responses at pixel (i, j) be denoted $F_x(i, j)$ for the response from $\mathbf{G}_x$ and let $F_y(i, j)$ be the response from convolving with $\mathbf{G}_y$. The gradient magnitude is given by

$$M(i, j) = \sqrt{(F_x(i, j))^2 + (F_y(i, j))^2}.$$

If you want more information, see this article: `https://en.wikipedia.org/wiki/Sobel_operator`

Apply sobel filters to empireGray.jpg to compute the $x-$gradient, $y-$gradient, and magnitude images. Contrast stretch each of them individually, and then Show the results side-by-side. Qualitatively, explain what each of these images is showing.

Solution:

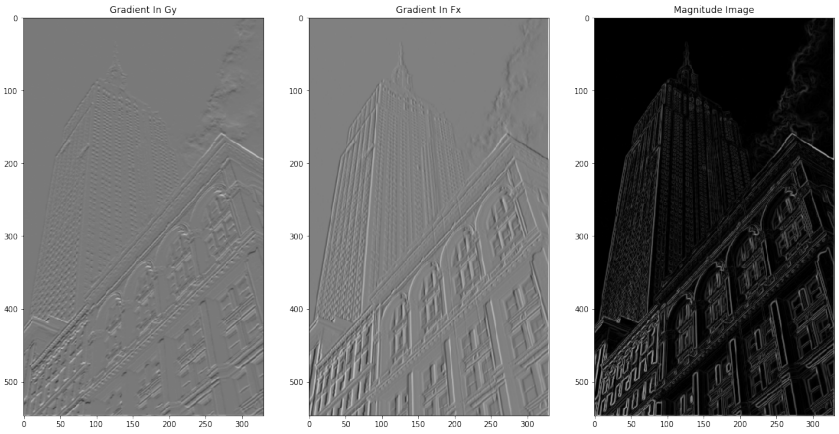

Figure 26: Sobel Edge Detection

Listing 6: Sobel

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

plt.figure(figsize=(18,19),dpi=70) #changing dpi we can zoom it, and change frame size using figsize
Fx = spatial_(cv_rgb[:,:,0],Gx,1)
Fy = spatial_(cv_rgb[:,:,0],Gy,1)
ma = np.zeros(Fx.shape,dtype=np.uint8)
```

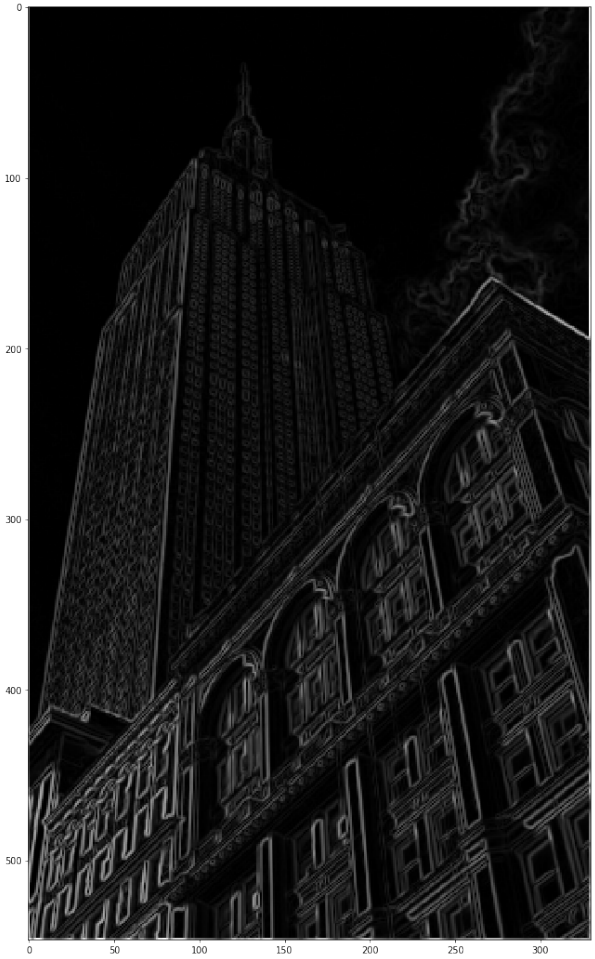

Figure 27: After Contrast Stretch

```
for i in range(Fx.shape[0]):
    for j in range(Fy.shape[1]):
        ma[i,j] = np.sqrt(Fx[i,j]**2 + Fy[i,j]**2)
ma = np.array(ma, dtype=np.uint8)

plt.subplot(131),plt.imshow(Fx,cmap="gray"),plt.title("X")
plt.subplot(132),plt.imshow(Fy,cmap="gray"),plt.title("Y")
plt.subplot(133),plt.imshow(ma,cmap="gray"),plt.title("Final Image")
plt.show()
```

Fx image shows the edge detection better which are vertical to the axis, this we can see from the image. Similarly we also see that Fy calculates the edges along the horizontal with axis. F is like magnitude which is displaying edges across both axis with respect to the image. After doing contrast stretching we can clearly see only edges which was our motive behind using sobel method.

Part 6 (3 points)

One common image processing operation is image sharpening. This is done by applying a linear filter known as an unsharp filter to the image and then adding the result back to the image. Convolve

empireGray.jpg with the unsharp filter

$$\mathbf{S} = \frac{1}{16}\begin{bmatrix} -1 & -2 & -1 \\ -2 & 12 & -2 \\ -1 & -2 & -1 \end{bmatrix},$$

and then add the filter response back to each channel of the color **empire.jpg** image. Show the results side-by-side. See this for more information: `https://en.wikipedia.org/wiki/Unsharp_masking`

Solution:

Solution:
Solution:

Listing 7: Unsharp

```
unsharp_filter = np.array([[-1,-2,-1],[-2,12,-2],[-1,-2,-1]])/16
k = spatial_(cv_rgb,unsharp_filter,P=1)
q = k[:,:,0] + cv_rgb[:,:,0]
w = k[:,:,1] + cv_rgb[:,:,1]
e = k[:,:,2] + cv_rgb[:,:,2]
r = np.zeros_like(cv_rgb)
r[:,:,0] = q
r[:,:,1] = w
r[:,:,2] = e
```

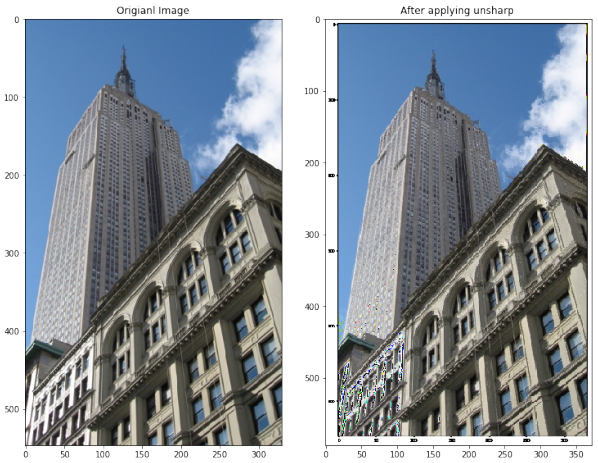

Figure 28: Unsharp original image

Problem 5 - Median Filtering

Part 1 (10 points)

An $n \times n$ median filter computes the median within each 'window' as it 'slides' across the image. Implement a $n \times n$ median filter. Your function should take an image I, the filter size n, and a Boolean flag p as input. It should return an filtered image the same size as the input. Ensure your function supports monochrome and RGB images, but the filtering should be done on each chromatic channel independently. You can assume that n is odd. The Boolean p flag indicates the padding style. If p is true, then the image's border is padded with $floor(n/2)$ zeros. If p is false, instead of padding with zeros to ensure the output is the same size as the input image, n will be reduced to use as much of the image as possible, e.g., if $n = 3$ then for the output of the filter "centered" on pixel $I(1,1)$, the filter would compute the median over $I(1,1)$, $I(1,2)$, $I(2,1)$, and $I(2,2)$. In other words, the size of the filter dynamically shrinks within $floor(n/2)$ pixels from the border.

Apply a 15×15 median filter to noisypic.jpg using each of the padding schemes. Show the two images produced and comment on any border effects.

Note: Unfortunately, this problem is not easy to vectorize, and you will probably need to use two or three **for** loops. You may use a built-in median function. You should be able to just modify the conv2dSpatial function you wrote earlier.

Solution:

Listing 8: Median Filter

```
def med_fil(img,p,n,S):
    try:
        C = img.shape[2]
    except:
        C = 1
    img = img
    if C >1:

        if p:
            P = int(np.floor(n/2))

            print(P)
        else:
            P = 0

        HH,WW = n,n
```

```
    H,W = img.shape[0],img.shape[1]

    pad_x = np.pad(img, ((P,), (P,), (0,)), 'constant')
    h = int(1 + ((H + 2 * P - HH) / S))
    wi = int(1 + ((W + 2 * P - WW) / S))

    out = np.zeros((h, wi,C))
    f = np.zeros((h,wi))
#     o = np.zeros((h,wi,C))

    for c in range(0,C):
        img2 = pad_x[:,:,c]
        for g in range(h):
            for t in range(wi):
                im = img2[g:g + HH, t:t + WW]
                im = np.sort(im)
                median_ = np.median(im)
    #                 filter_ = np.array([[1,1,1],[1,median_,1],[1,1,1]])
    #                / print(median_)
    #                 print(f.shape)
    #                 print(img2.shape)
    #                 print(g,HH,t,WW)
    #                 out[g:g + HH, t:t + WW,c] = pad_x[g:g + HH, t:t + WW,c]
                f[g:g + HH, t:t + WW] = median_
        out[:,:,c] = f
        f = np.zeros((h,wi))
    #                 out[g + P, t + P,c] = median_

    if C==1:
        o = np.zeros_like(img)
        o[:,:] = out[:,:,0]
    else:
        o = np.zeros_like(img)
        o = o[0:h,0:wi]
        o[:,:,0] = out[:,:,0]
        o[:,:,1] = out[:,:,1]
        o[:,:,2] = out[:,:,2]

    return o

else:
    if p:
        P = int(np.floor(n/2))

        print(P)
    else:
        P = 0

    HH,WW = n,n
    H,W = img.shape[0],img.shape[1]

    pad_x = np.pad(img, ((P,), (P,)), 'constant')
    h = int(1 + ((H + 2 * P - HH) / S))
    wi = int(1 + ((W + 2 * P - WW) / S))

    f = np.zeros((h,wi))
#     o = np.zeros((h,wi,C))



    for g in range(h):
        for t in range(wi):
            im = pad_x[g:g + HH, t:t + WW]
            im = np.sort(im)
            median_ = np.median(im)
    #                 filter_ = np.array([[1,1,1],[1,median_,1],[1,1,1]])
    #                / print(median_)
    #                 print(f.shape)
    #                 print(img2.shape)
    #                 print(g,HH,t,WW)
```

```
#                   out[g:g + HH,  t:t + WW, c] = pad_x[g:g + HH,  t:t + WW, c]
            f[g:g + HH,  t:t + WW] = median_

#                   out[g + P,  t + P, c] = median_

if C==1:
    o = np.zeros_like(img)
    o[:,:] = f[:,:]

return o
```

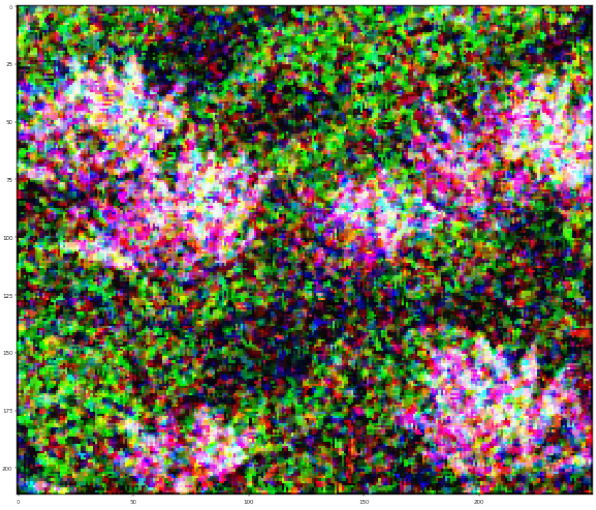

Figure 29: When padding is true and filter is size 3

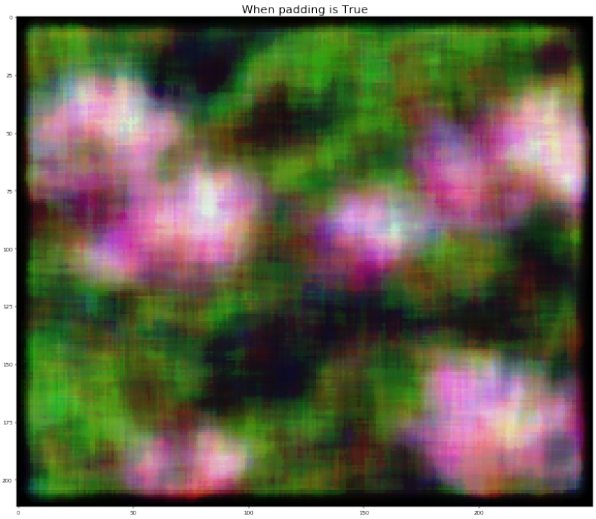

Figure 30: When padding is true

We can see from the above when padding is true, we get black borders, the reason is because when padding is zero there is accumulation 0 around the borders leading to zeros or dark coloration.

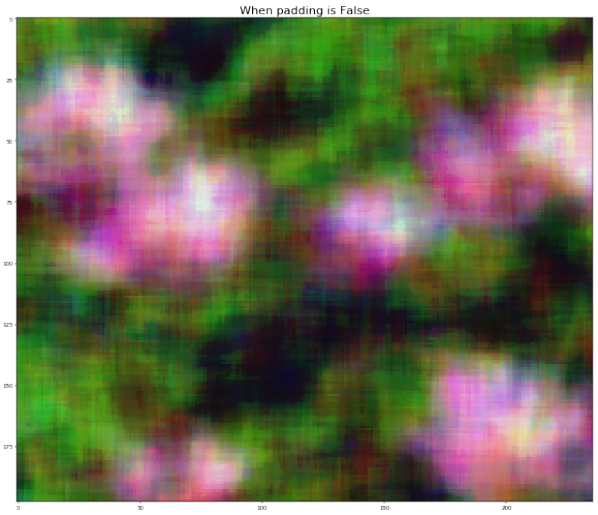

Figure 31: When padding is false

Part 2 (3 points)

While dynamically changing the filter size for our median filter was a reasonable alternative to padding the image, would dynamic changing of the filter size near an image's borders be a sensible strategy when using a linear filter? Explain.

Solution:

Median Filter As we can see from result fig.30 where we have used padding. It has caused the boundaries to go black, which is not needed. We could avoid this by using dynamic filter. Dynamic filter doesn't needs zero on the boundaries so we don't have to deal with black colors on the edge of the image. This is true for median filters.

Linear Filter But linear filter is not affected by presence of zero, in fact we use padding to keep edge information in image, when using linear filter. SO, We need padding in linear filter but not in Median filter.

Part 3 (3 points)

Show that a 3×3 median filter is not a linear filter. It is sufficient to show that a median filter violates one of the principles of linearity for some images. Hint: Find an example using two 3×3 images.

Solution:
We assume a matrix of 6×6 like below

$$M = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & 17 & 18 \\ 19 & 20 & 21 & 22 & 23 & 24 \\ 30 & 29 & 28 & 27 & 26 & 25 \\ 31 & 32 & 33 & 34 & 35 & 36 \end{pmatrix} \tag{7}$$

We choose linear filter type Mean Filter for smoothing purposes, and non linear filter median filter. Both 3×3. For Mean Filter we get following values if we slide the window on first layer as [8,9,10,11,7,4] Here we can deduce that Linear filter will increase or decrease, this is the basic property of Linear filter.

Listing 9: example

```
M = np.arange(1,37).reshape(6,6)

f1 = np.ones((3,3))/9

f2 = np.ones((3,3))/9

f2 = 3*f2

f3 = f1+f2
r1 = spatial_1(M, f1,P = 1)
r2 = spatial_1(M, f2,P=1)
r3 = spatial_1(M, f3,P=1)
```

we find that r3 = r1 + r2

value for r1+r2 =

8.00	13.33	16.00	18.67	21.33	15.11
20.00	32.00	36.00	40.00	44.00	30.67
36.00	56.00	60.00	64.00	68.00	46.67
52.00	80.00	84.00	88.00	92.00	62.67
68.00	104.00	108.00	112.00	116.00	78.67
50.67	77.33	80.00	82.67	85.33	57.78

value for r3 =

8.00	13.33	16.00	18.67	21.33	15.11
20.00	32.00	36.00	40.00	44.00	30.67
36.00	56.00	60.00	64.00	68.00	46.67
52.00	80.00	84.00	88.00	92.00	62.67
68.00	104.00	108.00	112.00	116.00	78.67
50.67	77.33	80.00	82.67	85.33	57.78

this satisfies the condition of linear filters.

Now, we see Median Filter. Median filter does not follow this linearity, since it is taking median from all sorted values. The operation itself is non linear, which concludes that median filter is also non linear. Now we apply Median filter to same problem

Median Filter Value of r1+r2 =

0.00	4.00	11.00	18.00	25.00	0.00
4.00	16.00	23.00	30.00	37.00	32.00
11.00	23.00	30.00	37.00	44.00	39.00
18.00	30.00	37.00	44.00	51.00	46.00
25.00	37.00	44.00	51.00	58.00	53.00
0.00	32.00	39.00	46.00	53.00	0.00

value of r3 =

0.00	9.00	16.00	23.00	30.00	0.00
9.00	16.00	23.00	30.00	37.00	37.00
16.00	23.00	30.00	37.00	44.00	44.00
23.00	30.00	37.00	44.00	51.00	51.00
30.00	37.00	44.00	51.00	58.00	58.00
0.00	37.00	44.00	51.00	58.00	0.00

r3 $\neq$ r1 + r2, so non linear

Part 4 (6 points)

Apply a 5×5 median filter to lady.jpg. Also, convolve lady.jpg with a 5×5 linear box filter. Show the results side-by-side. Which approach worked better? Explain.

Listing 10: example

```
r,c = 4,60
plt.figure(figsize=(18,19),dpi=70) #changing dpi we can zoom it, and change frame size using figsize
mpl.gridspec.GridSpec(r,c)
plt.figure(figsize=(20,20))
plt.subplot2grid((r,c),(0,0),colspan=20,rowspan=2)
# plt.figure(figsize=(18,19),dpi=40)
img = cv2.imread('/home/n/images/lady.jpg')
cv_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
kk = cv2.filter2D(cv_rgb,-1,f)
plt.imshow(kk)
plt.title("Linear FIlter")



plt.subplot2grid((r,c),(0,20),colspan=20,rowspan=2)
img = cv2.imread('/home/n/images/lady.jpg')
cv_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
kk = med_fil(cv_rgb,p=True,n=5,S=1)
plt.imshow(kk)
plt.title("Median Filter")
```

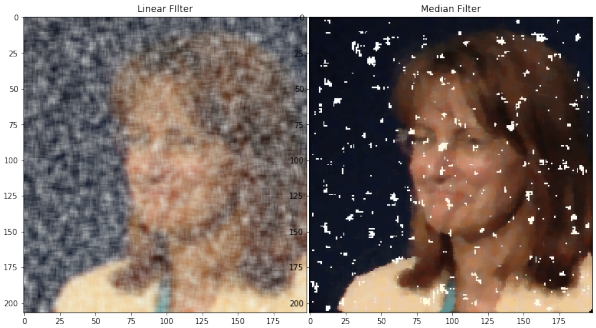

Figure 32: Linear filter v/s non linear filter

we can see that linear filter is not good enough to remove noise from the image, but as we see median filter filter has performed well with the noise, although it has left many white patches in the image. This

white patches are also noise, but even so median filter has helped has to recover more than 60 percent of the information required to recognize the person in the image.

Problem 6 - Binary Images

In this problem we will explore binary images and morphological operators. You will need to use OpenCV. See this: `http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html`

Part 1 (2 points)

Convert cells.jpg to a monochrome image. Find a good threshold to binarize the image, where your choice of threshold isolates the majority of the cells from the background. Finding a good threshold will require some experimentation. You may wish to look at the mean/median pixel values and the image histogram. After finding a good threshold, binarize the image. Ensure that the cells are white (ones) and the background is black (zeros). Show the image and say what value for the threshold you used. Save the image and call it binarycells.png. Note that you are going to be using this image in the next few parts.

Solution:

Listing 11: example

```
def to_binary(img, lower, upper):
    return (lower < img) & (img < upper)

j = to_binary(img,120,255)
```

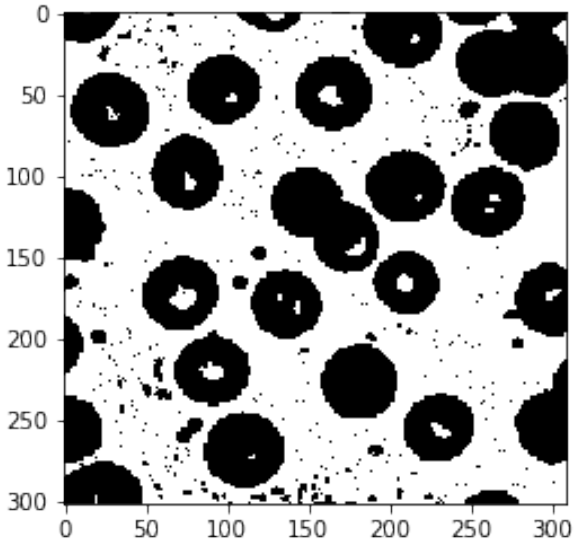

Figure 33: Black and White cell

Part 2 (3 points)

Test the image **erode** operation using a square structuring element (a kernel) on binarycells.png. A square structuring element of size n is given by an $n \times n$ matrix of ones. Apply the erode operation for $n = 3$, $n = 5$, and $n = 7$. Show the images side-by-side.

Solution:

Solution:

Listing 12: example

```
r,c = 4,40
plt.figure(figsize=(18,19),dpi=70) #changing dpi we can zoom it, and change frame size using figsize
mpl.gridspec.GridSpec(r,c)

plt.subplot2grid((r,c),(1,0),colspan=10,rowspan=3)
plt.imshow(img)
plt.title("Original image")



plt.subplot2grid((r,c),(1,10),colspan=10,rowspan=3)
kernel = np.ones((3,3),np.uint8)
erosion = cv2.erode(img,kernel,iterations = 1)
plt.imshow(erosion,)
plt.title("Kernel size 3")

plt.subplot2grid((r,c),(1,20),colspan=10,rowspan=3)
kernel = np.ones((5,5),np.uint8)
erosion = cv2.erode(img,kernel,iterations = 1)
plt.imshow(erosion,)
plt.title("Kernel size 5")



plt.subplot2grid((r,c),(1,30),colspan=10,rowspan=3)
kernel = np.ones((7,7),np.uint8)
erosion = cv2.erode(img,kernel,iterations = 1)
plt.imshow(erosion,)
plt.title("Kernel size 7")


plt.show()
```

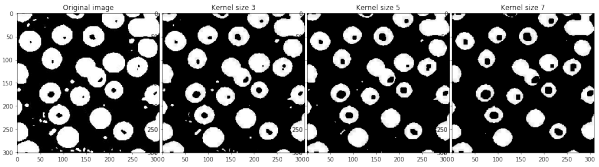

Figure 34: erosion

Part 3 (3 points)

Test the image **dilate** operation using a square structuring element on binarycells.png. Apply the dilate operation for $n = 3$, $n = 5$, and $n = 7$. Show the images side-by-side.

Solution:

Listing 13: example

```
r,c = 4,40
plt.figure(figsize=(18,19),dpi=70) #changing dpi we can zoom it, and change frame size using figsize
mpl.gridspec.GridSpec(r,c)

plt.subplot2grid((r,c),(1,0),colspan=10,rowspan=3)
plt.imshow(img)
plt.title("Original image")



plt.subplot2grid((r,c),(1,10),colspan=10,rowspan=3)
kernel = np.ones((3,3),np.uint8)
dilate = cv2.dilate(img,kernel,iterations = 1)
plt.imshow(dilate,)
plt.title("Kernel size 3")

plt.subplot2grid((r,c),(1,20),colspan=10,rowspan=3)
kernel = np.ones((5,5),np.uint8)
dilate = cv2.dilate(img,kernel,iterations = 1)
plt.imshow(dilate,)
plt.title("Kernel size 5")



plt.subplot2grid((r,c),(1,30),colspan=10,rowspan=3)
kernel = np.ones((7,7),np.uint8)
dilate = cv2.dilate(img,kernel,iterations = 1)
plt.imshow(dilate,)
plt.title("Kernel size 7")


plt.show()
```

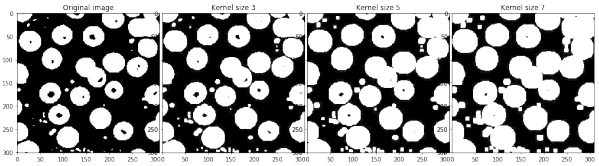

Figure 35: Deliate

Part 4 (3 points)

Test the image **open** operation using a square structuring element on binarycells.png. Apply the open operation for $n = 3$, $n = 5$, and $n = 7$. Show the images side-by-side.

Solution:

Listing 14: example

```
opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)



r,c = 4,40
plt.figure(figsize=(18,19),dpi=70) #changing dpi we can zoom it, and change frame size using figsize
mpl.gridspec.GridSpec(r,c)
```

```
plt.subplot2grid((r,c),(1,0),colspan=10,rowspan=3)
plt.imshow(img)
plt.title("Original image")



plt.subplot2grid((r,c),(1,10),colspan=10,rowspan=3)
kernel = np.ones((3,3),np.uint8)
opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
plt.imshow(opening)
plt.title("Kernel size 3")

plt.subplot2grid((r,c),(1,20),colspan=10,rowspan=3)
kernel = np.ones((5,5),np.uint8)
opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
plt.imshow(opening)
plt.title("Kernel size 5")



plt.subplot2grid((r,c),(1,30),colspan=10,rowspan=3)
kernel = np.ones((7,7),np.uint8)
opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
plt.imshow(opening)
plt.title("Kernel size 7")


plt.show()
```

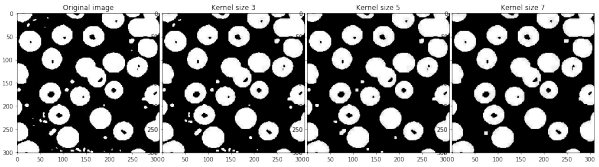

Figure 36: opening

Part 5 (3 points)

Test the image **close** operation using a square structuring element on binarycells.png. Apply the close operation for $n = 3$, $n = 5$, and $n = 7$. Show the images side-by-side.

Solution:

Listing 15: example

```
# opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)



r,c = 4,40
plt.figure(figsize=(18,19),dpi=70) #changing dpi we can zoom it, and change frame size using figsize
mpl.gridspec.GridSpec(r,c)

plt.subplot2grid((r,c),(1,0),colspan=10,rowspan=3)
plt.imshow(img)
plt.title("Original image")
```

```
plt.subplot2grid((r,c),(1,10),colspan=10,rowspan=3)
kernel = np.ones((3,3),np.uint8)
closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
plt.imshow(closing)
plt.title("Kernel size 3")

plt.subplot2grid((r,c),(1,20),colspan=10,rowspan=3)
kernel = np.ones((5,5),np.uint8)
closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
plt.imshow(closing)
plt.title("Kernel size 5")



plt.subplot2grid((r,c),(1,30),colspan=10,rowspan=3)
kernel = np.ones((7,7),np.uint8)
closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
plt.imshow(closing)
plt.title("Kernel size 7")


plt.show()
```

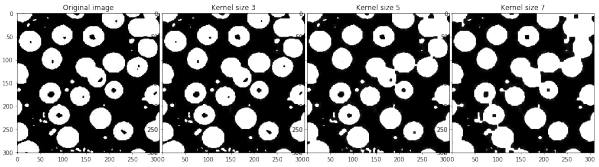

Figure 37: closing

Part 6 (10 points)

Use thresholding and morphological operators to write a script that takes in an image with coins, and outputs the same image with each coin labeled with its type (Penny, Nickel, Dime, Quarter). The title of the image should be the total amount of money in the image. Test your script on coins.jpg. (My solution was 32 lines long)

Hints: See this link for some information about connected components in Python: `http://stackoverflow.com/questions/16937158/extracting-connected-objects-from-an-image-in-python`

Solution:
PUT AN IMAGE HERE PUT CODE HERE

Listing 16: example

```
i1 = np.array(i1, dtype=np.uint8)

kernel = np.ones((3,3),np.uint8)
lol = cv2.dilate(i1, kernel,iterations = 3)
plt.imshow(lol,cmap="gray")


dst = med_fil(lol,p=True,n=3,S=1)

kernel = np.ones((3,3),np.uint8)
lol2 = cv2.dilate(dst, kernel,iterations = 1)
plt.imshow(lol2,cmap="gray")



im2, contours, hierarchy = cv2.findContours(lol2, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

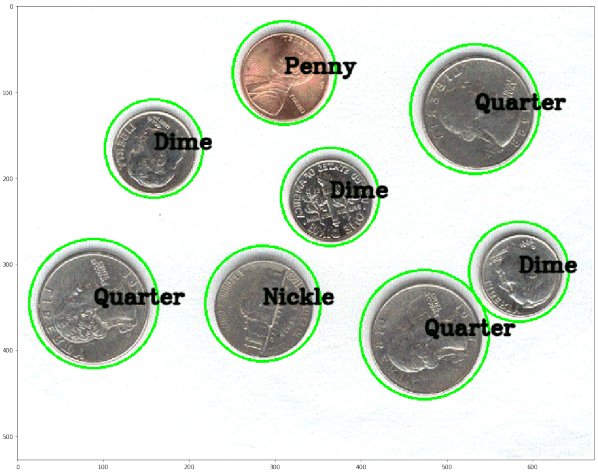

Figure 38: erosion

```
i = 0
while (i < 8):
    cnt = contours[i]
    (x,y),radius = cv2.minEnclosingCircle(cnt)
    center = (int(x),int(y))
    radius = int(radius)
    img = cv2.circle(cv_rgb,center,radius,(0,255,0),2)
    if radius > 55 and radius < 59:
        cv2.putText(img, "Dime", center, cv2.FONT_HERSHEY_COMPLEX, 0.8, (0, 5, 0), 2, cv2.LINE_AA)
    elif radius > 59 and radius < 65:
        cv2.putText(img, "Penny", center, cv2.FONT_HERSHEY_COMPLEX, 0.8, (0, 5, 0), 2, cv2.LINE_AA)
    elif radius > 72 and radius < 77:
        cv2.putText(img, "Quarter", center, cv2.FONT_HERSHEY_COMPLEX, 0.8, (0, 5, 0), 2, cv2.LINE_AA)
    elif radius > 65 and radius < 70:
        cv2.putText(img, "Nickle", center, cv2.FONT_HERSHEY_COMPLEX, 0.8, (0, 5, 0), 2, cv2.LINE_AA)
    i = i+1
    print(radius)
    print(center)
```

Problem 7 - Image Speckle (6 points)

Some cameras can be rather noisy and sometimes fails to capture one of the RGB pixels. You can try to fix this situation by using a median filter, but this will produce unnecessary blurring in regions of the image that do not have the noisy pixels. Describe an algorithm for removing the pixel speckle while only blurring a region if it is absolutely necessary due to the speckle. Apply your method on noisyretina.jpg and show the result. Show the image at full resolution in your PDF.

Solution:

Listing 17: example

```
plt.figure(figsize=(18,19),dpi=40) #changing dpi we can zoom it, and change frame size using figsize

plt.imshow(cv2.fastNlMeansDenoisingColored(cv_rgb,None,10,10,7,21))
```

We know speckle noise is from electro magnetic refraction of wavelength coming from objects. This noise are sometimes useful, but for most are needed to be reduced. Linear filtering doesn't seems to be

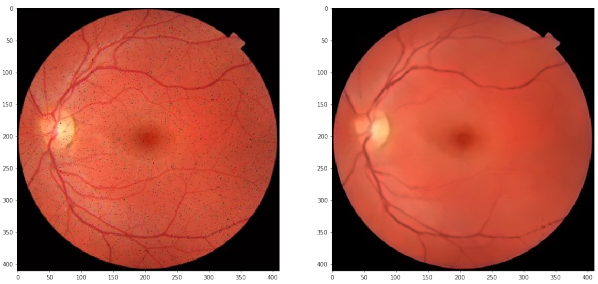

Figure 39: Non local means algorithm

Figure 40: Median Filtering

working on speckle noise.

Non Local Means We can see from the image, that image is more blurred compared to non linear means. The thing to note is that median filter is blurring the image, but its still maintaining detailed features, compared to non linear means which removes detailed parts from the image.

We have two windows, one window size of 21 is used to search the regions, on which the second window of size 7 is used to approximate neighbors.