**Problem Set 3**

**6.689 – Advances in Computer Vision**

**Problem 1**

**Image A Image B**

Graphical user interface, application

Description automatically generatedA picture containing application

Description automatically generated

**Original Blend Downsampled**

Graphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generated

**The image above used a Gaussian kernel with dimensions 128x128, mu=64, and sigma=32. The Gaussian seemed to work marginally better, and the value of sigma seemed somewhat irrelevant. The more blurring, the more the lines/edges and high special frequency of image A, and less of B will be prominent.**

#creating a Gaussian filter

def Gauss\_kernel\_filter(kern\_dim, mu, sigma):

# creating the kernel

x, y = np.meshgrid(np.arange(kern\_dim), np.arange(kern\_dim))

#creating the Gaussian filter with parameters mu, sigma, and coordinates

Gauss\_filt = exp(-1\*((x-mu)\*\*2+(y-mu)\*\*2)/(2\*sigma\*\*2))

return Gauss\_filt

def box\_kernel\_filter(kern\_xdim, kern\_ydim):

return np.ones((kern\_ydim, kern\_xdim))/(kern\_xdim\*kern\_ydim)

#blurring

def blurring(filter):

blur = filter/np.sum(filter)

return blur

#Convolving with the color channels

def color\_convolve(blur\_w\_filter, img):

new\_img = img.copy()

for color in range(3):

new\_img[:,:,color] = conv2d(img[:,:,color], Gauss\_blur, mode='same')

new\_img = new\_img.astype('int')

return new\_img

GF = Gauss\_kernel\_filter(128,64,640)

#BF = box\_kernel\_filter(3,3)

#Gauss\_blur = blurring(GF)

#Box\_blur = blurring(BF)

def hybrid(img1, img2, Filter):

FB = blurring(Filter)

blurry\_img1, blurry\_img2 = color\_convolve(FB, img1), color\_convolve(FB, img2)

first\_term = blurry\_img2

second\_term = img1-blurry\_img1

hybrid\_img = first\_term+second\_term

return hybrid\_img

test = hybrid(iggy, borat, GF)

plt.imshow(test)

**Problem 2:**

Using a circle mask with radius 13:

Chart

Description automatically generated

Using a Box Filter with dimensions 30x30:A picture containing chart

Description automatically generated

**Is the hidden picture Gauss?**

def circular\_mask(dimx, dimy, radius):

x, y = np.meshgrid(np.arange(dimx), np.arange(dimy))

mask = (sqrt((x-dimx/2)\*\*2+(y-dimy/2)\*\*2)<=radius)\*1.0

return mask

#circle\_mask = circular\_mask(stein.shape[0], stein.shape[1], 13)

def dehybrid(img, mask):

low\_freq = fftshift(fft2(stein))\*mask

high\_freq = fftshift(fft2(stein))-low\_freq

img1 = intensityscale(real(ifft2(ifftshift(low\_freq))))

img2 = intensityscale(real(ifft2(ifftshift(high\_freq))))

plt.subplot(1, 2, 1)

imshow(img1)

plt.subplot(1, 2, 2)

imshow(img2)

for i in range(2,100):

circle\_mask = circular\_mask(stein.shape[0], stein.shape[1], i)

plt.figure()

dehybrid(stein, circle\_mask)

for i in range(5,50,5):

low\_freq = conv2d(stein, box\_kernel\_filter(i,i), mode='same')

high\_freq = stein - conv2d(stein, box\_kernel\_filter(i,i), mode='same')

plt.figure()

plt.subplot(1, 2, 1)

imshow(intensityscale(low\_freq))

plt.subplot(1, 2, 2)

imshow(intensityscale(high\_freq))

**Problem 3**

**a)**

Chart

Description automatically generated

phaseShift = angle(im2Dft)-angle(im1Dft)

magnifiedDft = exp((magnificationFactor\*phaseShift+angle(im1Dft))\*1j)\*abs(im2Dft)

**b)**

**i)**Chart, histogram

Description automatically generated

expected = np.zeros([imSize, imSize])

expected[0,1\*magnificationFactor] = 1

expected[1\*magnificationFactor,8] = 1

**ii) The key differences in the expected and magnified two squares in the bottom left and the darkest square around (x=6, y=2). The erroneous two white squares are caused by the cyclic nature of a Fourier. The square at (x=1, y=6) is caused by the top left square in image 1 moving vertically and wrapping around into the bottom of the second column. The square at (x=2, y=7) is a similar phenomenon but with the bottom right square moving horizontally and wrapping into the left side of the second to bottom row.**

**c)**Chart

Description automatically generated

gaussianMask = exp(-1\*((X-x)\*\*2+(Y-y)\*\*2)/(2\*sigma\*\*2))

windowMagnified = magnifyChange(im1\*gaussianMask, im2\*gaussianMask, magnificationFactor)

magnified = magnified+windowMagnified

**d)**

# create windowed frames #TODO

gaussianMask = exp(-1\*((X-x)\*\*2+(Y-y)\*\*2)/(2\*sigma\*\*2))

windowedFrames = gaussianMask \* frames[frameIndex,:,:,channelIndex]

# initialize moving average of phase for current window/channel

if frameIndex == 0:

windowAveragePhase = angle(fft2(windowedFrames))

windowDft = fft2(windowedFrames)

# compute phase shift and constrain to [-pi, pi] since

# angle space wraps around

windowPhaseShift = angle(windowDft) - windowAveragePhase

windowPhaseShift[windowPhaseShift > pi] = windowPhaseShift[windowPhaseShift > pi] - 2 \* pi

windowPhaseShift[windowPhaseShift < -pi] = windowPhaseShift[windowPhaseShift < -pi] + 2 \* pi

# magnify phase shift # TODO

windowMagnifiedPhase = (magnificationFactor\*windowPhaseShift)+windowAveragePhase

# go back to image space # TODO

windowMagnifiedDft = exp(windowMagnifiedPhase\*1j)\*abs(windowDft)

windowMagnified = abs(ifft2(windowMagnifiedDft))

**Problem 4)**

**a)**

A picture containing diagram

Description automatically generated

def create\_gaussian\_pyramid(vid, num\_levels=4):

### TODO: ENTER YOUR CODE BELOW

### return a list with the gaussian pyramid of the video.

### consider using the cv2.pyrDown function to create each level of the pyramid.

pyr\_list = []

pyr\_list.append(vid)

for i in range(num\_levels):

pyr\_list.append(np.array([cv2.pyrDown(pyr\_list[-1][j]) for j in range(frames.shape[0])]))

return pyr\_list

**b)**

Diagram

Description automatically generated with medium confidence

def create\_laplacian\_pyramid(gaussian\_pyramid):

### TODO: ENTER YOUR CODE BELOW

### use the gaussian pyramid to create the laplacian pyramid for the video.

### You might find cv2.pyrUp function useful.

lap\_pyr = []

for i in range(len(gaussian\_pyramid)-1):

prev, next1 = gaussian\_pyramid[i], gaussian\_pyramid[i+1]

next1 = np.array([cv2.pyrUp(j) for j in next1])

diff = prev-next1

lap\_pyr.append(diff)

return lap\_pyr

**c)**

b, a = signal.butter(filter\_order, [low, high], btype='band')

# filter the laplcian of the video using the signal.lfilter

y = signal.lfilter(b, a, laplace\_video, axis=0)

**d)**

bandpass\_filtered\_copy = bandpass\_filtered.copy()

for i in range(1,len(bandpass\_filtered)):

lvl = bandpass\_filtered[-i]

for j in range(lvl.shape[0]):

bandpass\_filtered\_copy[-i-1][j,:,:,:]+=cv2.pyrUp(lvl[j,:,:,:])

baby\_euler\_magnification = frames+bandpass\_filtered\_copy[0]