Problem Set 3
6.689 – Advances in Computer Vision

Problem 1

Image A

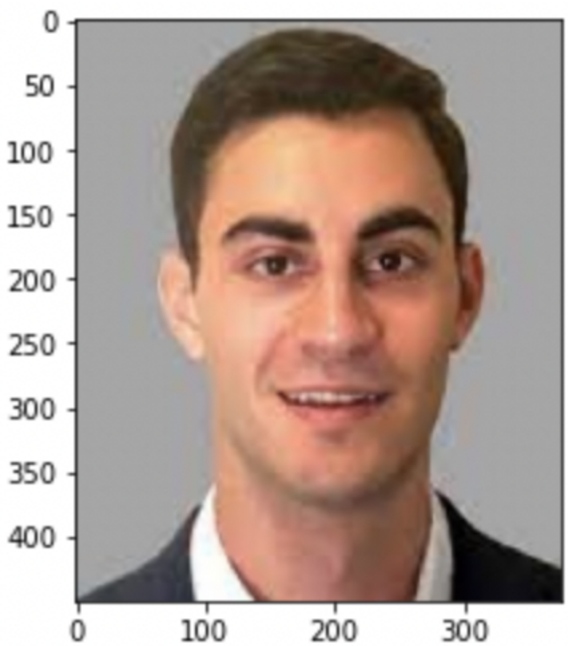
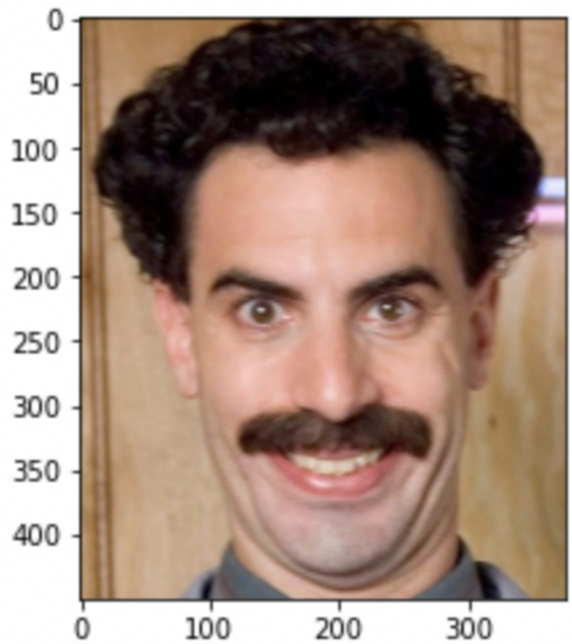
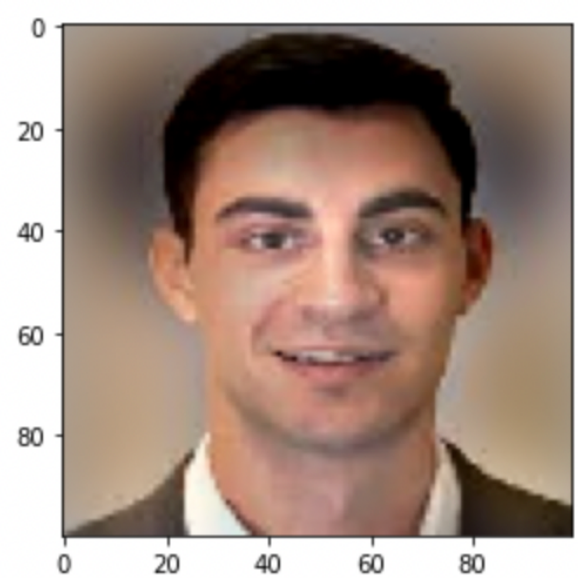


Image B



Original Blend

Downsampled

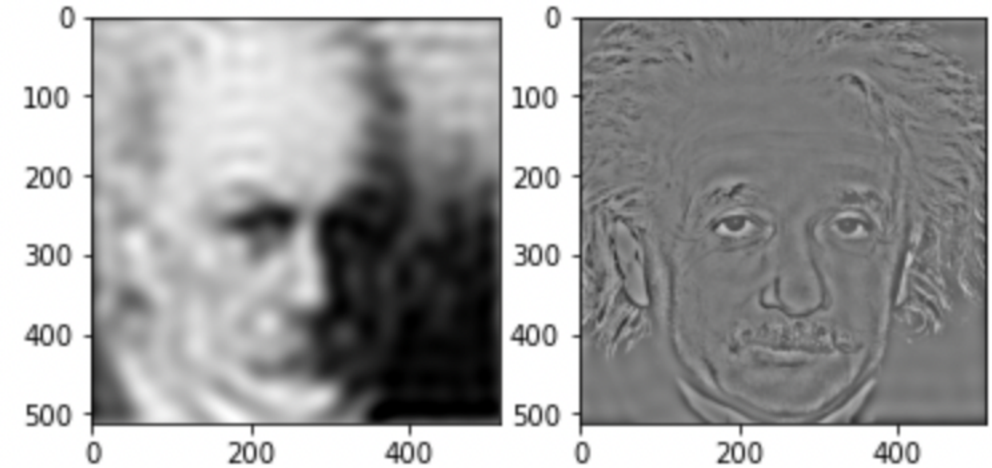


The image above used a Gaussian kernel with dimensions 128x128, mu=64, and sigma=32. The Gaussian seemed to work marginally better, and the value of sigma seemed somewhat irrelevant. The more blurring, the more the lines/edges and high special frequency of image A, and less of B will be prominent.

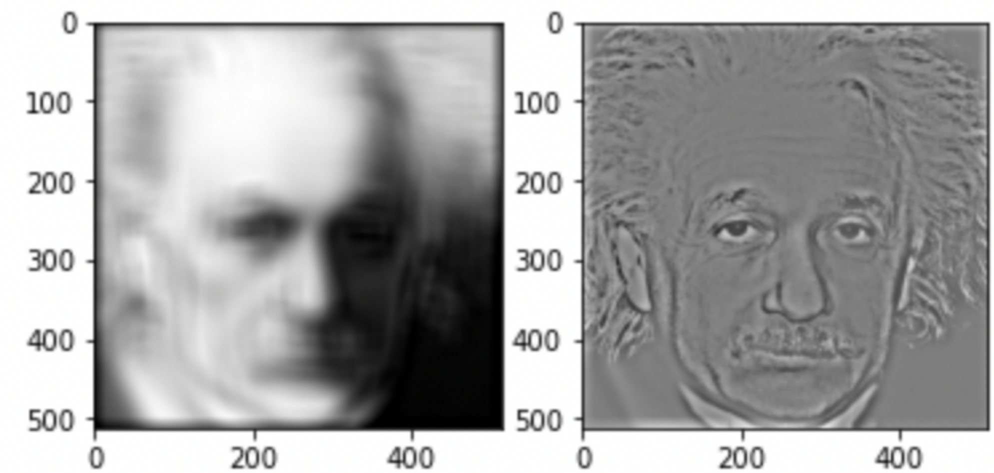
```
#creating a Gaussian filter
def Gauss_kernel_filter(kern_dim, mu, sigma):
    # creating the kernel
    x, y = np.meshgrid(np.arange(kern_dim), np.arange(kern_dim))
    #creating the Gaussian filter with parameters mu, sigma, and coordinates
    Gauss_filt = exp(-1*((x-mu)**2+(y-mu)**2)/(2*sigma**2))
    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:



Using a Box Filter with dimensions 30x30:



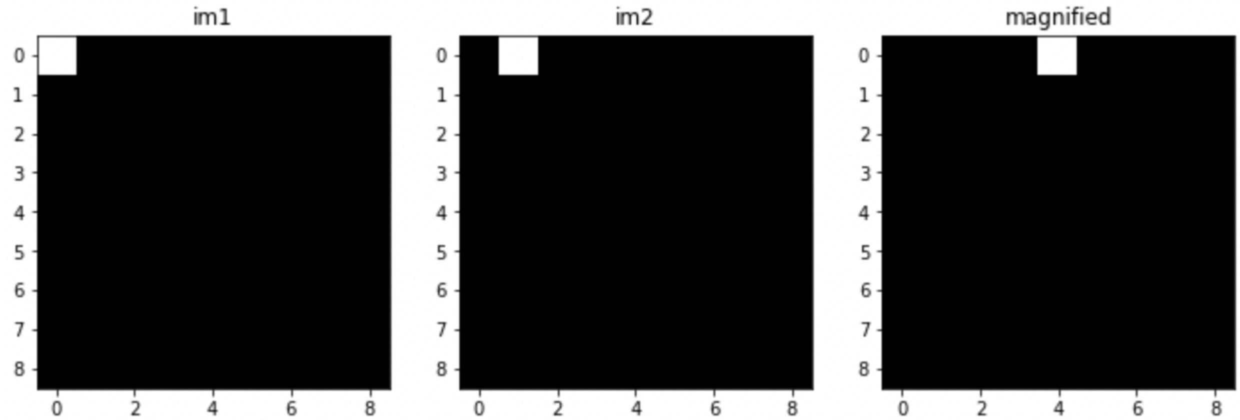
Is the hidden picture Gauss?

```
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

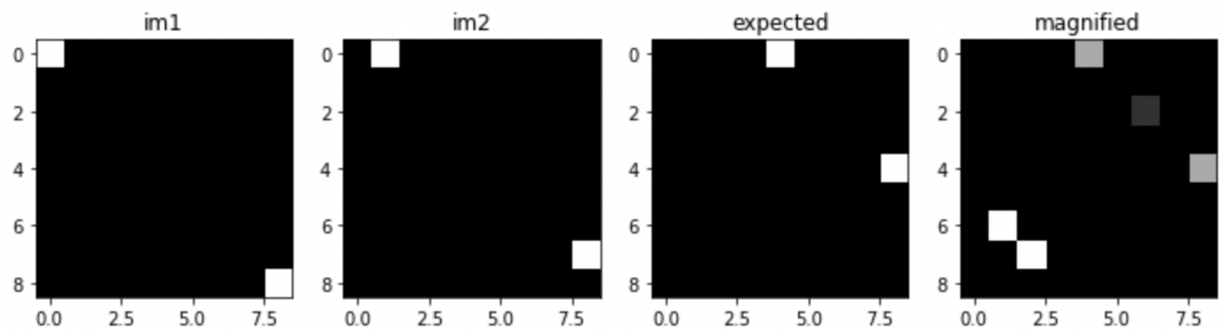
a)



phaseShift = angle(im2Dft)-angle(im1Dft)

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

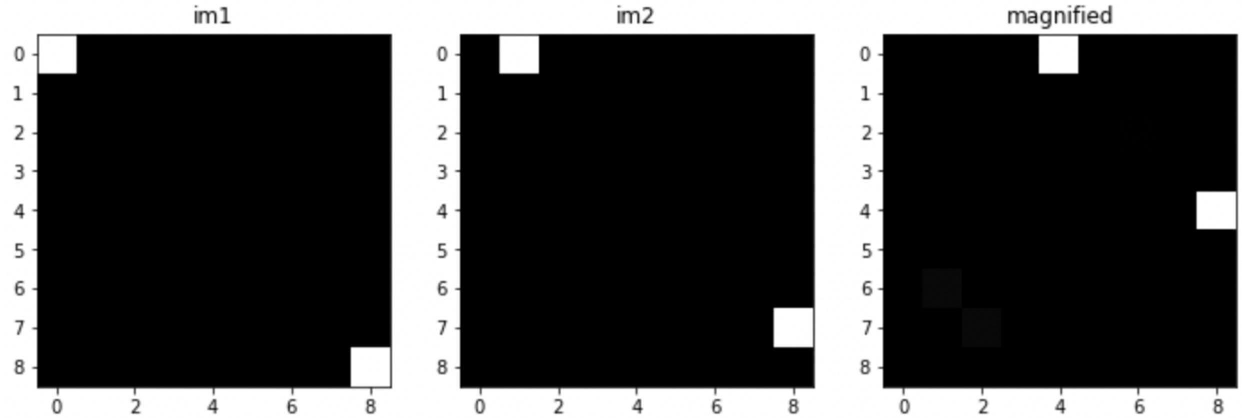
b)
i)



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)



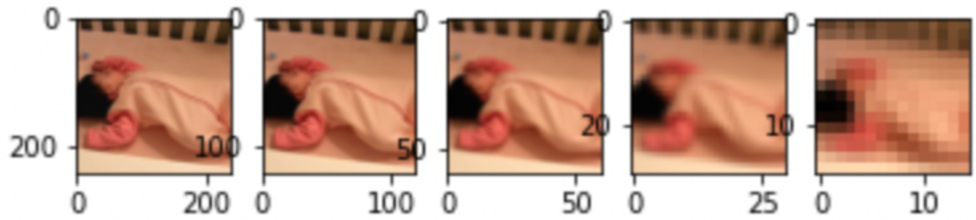

```
gaussianMask = exp(-1*((X-x)**2+(Y-y)**2)/(2*sigma**2))
windowMagnified = magnifyChange(im1*gaussianMask, im2*gaussianMask, magnificationFactor)
magnified = magnified+windowMagnified
```

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))
```

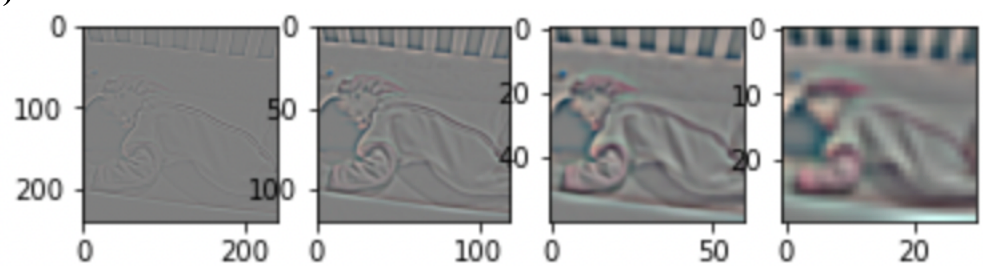
Problem 4)

a)



```
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)



```
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]
```