CS 280 – Homework #2

Problem 1: Hybrid Images

1. (a) (b) (c) Code: See Appendix 1
2. …

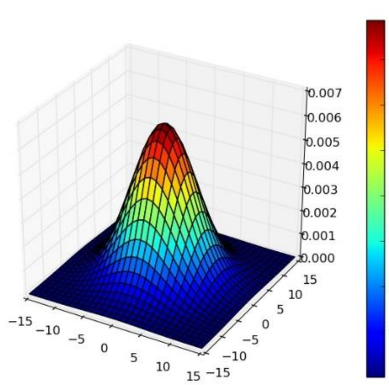
   (a) Algorithm: The algorithm is based on the facts that:
   - The human eye can only see high frequencies from a close distance and cannot see high frequencies from a far distance.
   - The human eye can only see low frequencies from a far distance and cannot see low frequencies from a close distance.

   In brief, human eyes can see high frequencies from a close distance and can see low frequencies from a far distance. So, we low-pass filter image 1 and eliminate its high frequencies so that human eyes can see it from a far distance; we high-pass filter image 2 and eliminate its low frequencies so that human eyes can see it from a close distance. Subsequently, we linearly combine the filtered image 1 and the filtered image 2 to create a hybrid image from which human eyes can see the image 2 from a close distance and the image 1 from a far distance.

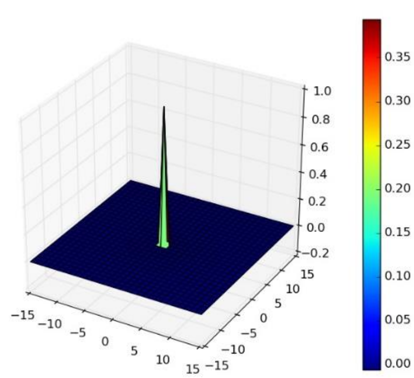
   The parametric from of low-pass and high-pass filters are:

$$\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

   where $\sigma = 5$ for both Gaussian filters, the cutoff frequencies are 10 and 5 for image 1 and image 2, respectively (i.e. image 1 has frequencies less than or equal 10, image 2 has frequencies larger than or equal 5).

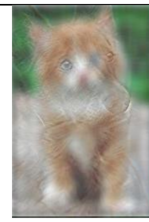


Gaussian low-pass filter

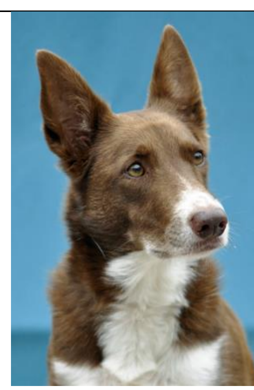


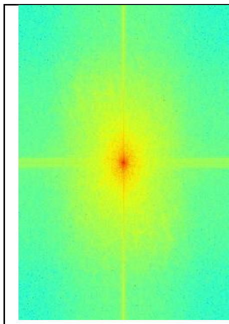
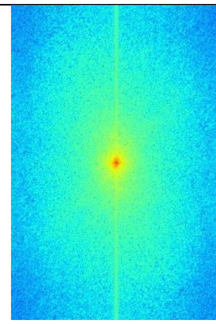
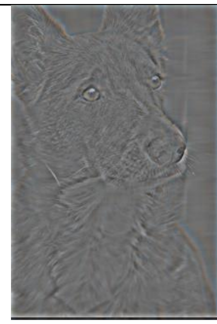
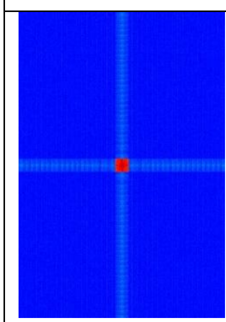
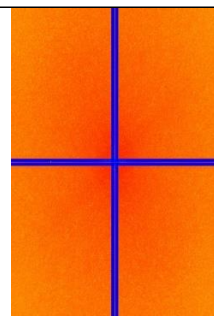
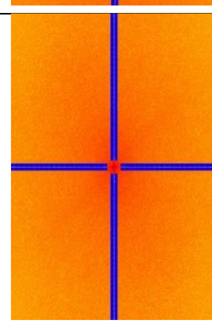
Gaussian high-pass filter

   (b) Favorite result:

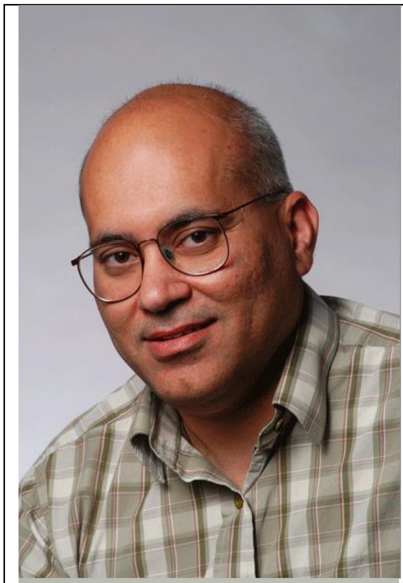
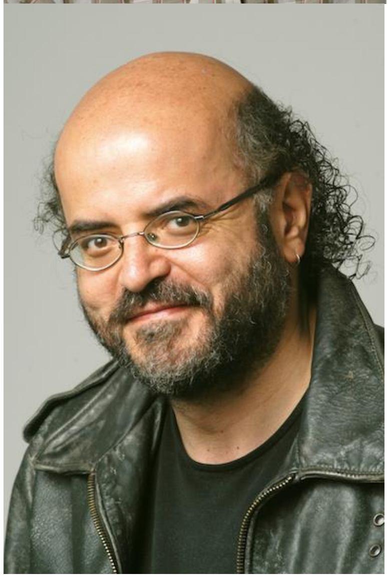
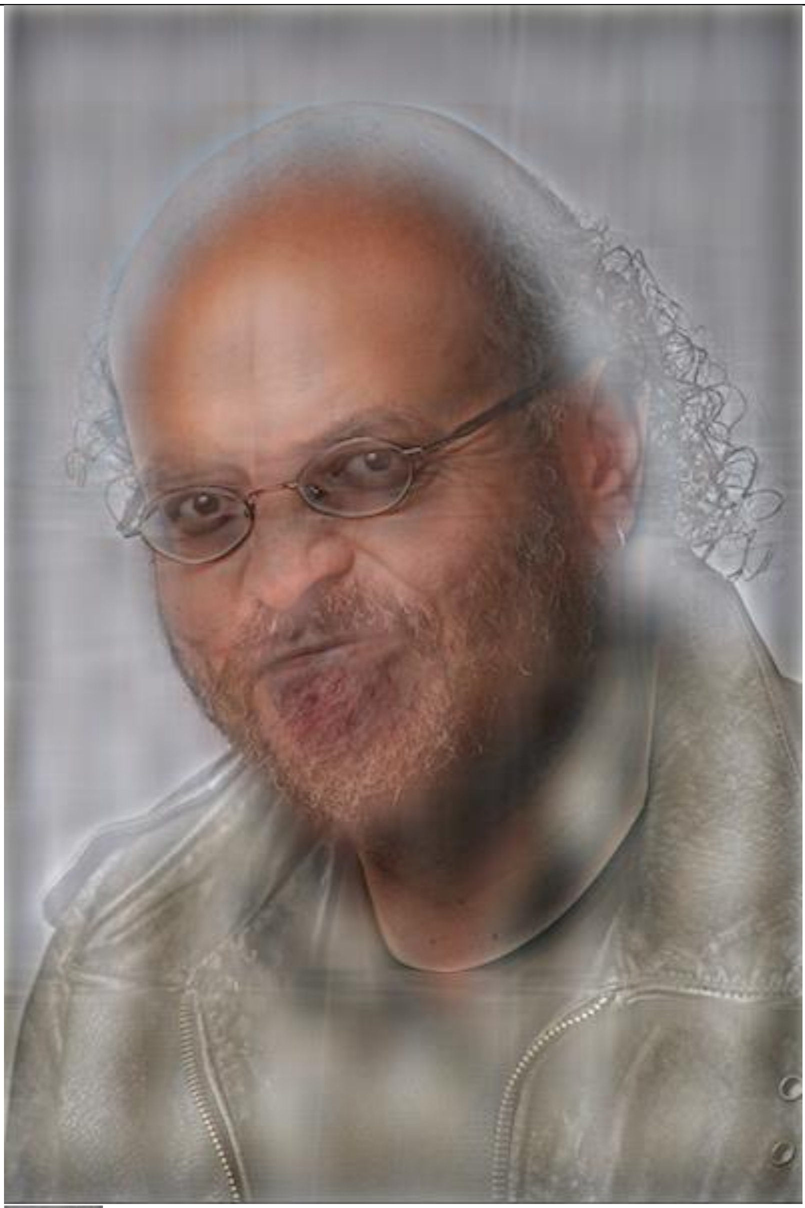
	
Dog from a close distance	Cat from a far distance

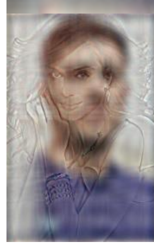
Frequency analysis:

 Ground truth cat	 Ground truth dog

 log of fft of input cat	 log of fft of input dog
 cat filtered (low freqs)	 dog filtered (high freqs)
 log of fft of filtered cat	 log of fft of filtered dog
 hybrid cat + dog	 log of fft of hybrid cat+dog

(c) Two more results:

<table>
<tr><td>

Prof. Malik (above)
Prof. Papadimitriou
(below)</td><td>
 Hybrid Malik+Papadimitriou at close distance (Papa) and at far distance (Malik)</td></tr>
<tr><td></td><td></td></tr>
</table>

  Boy and girl	  Hybrid boy+girl at close distance (girl) and at far distance (boy)

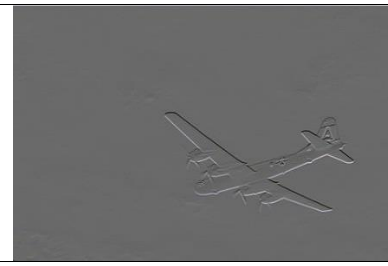
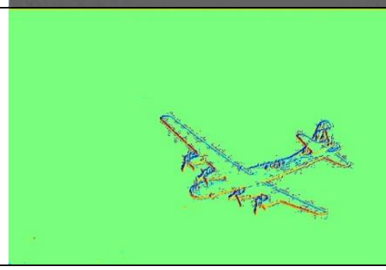
After a lot of trials and errors, I got some tricks to create a good hybrid:

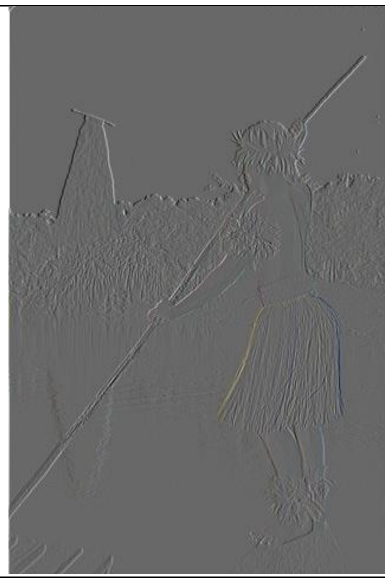
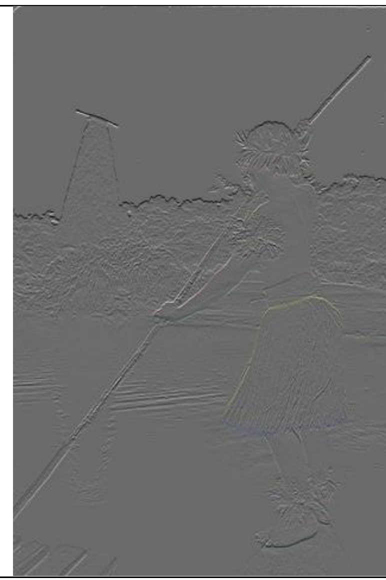
- Cutoff frequencies: should be chosen so that the remaining information of both images should be relatively equal, i.e. image 1 is not too dim compared to image 2 and vice versa. Usually, the one without low frequencies is dimmer than the other, so we should choose its cutoff frequency not too large.
- Alignment: should match eyes and mouth, or pose.
- Linear combination: adjust the weight when adding up two images.
- Selection of images: they should have relatively similar size. Etc.

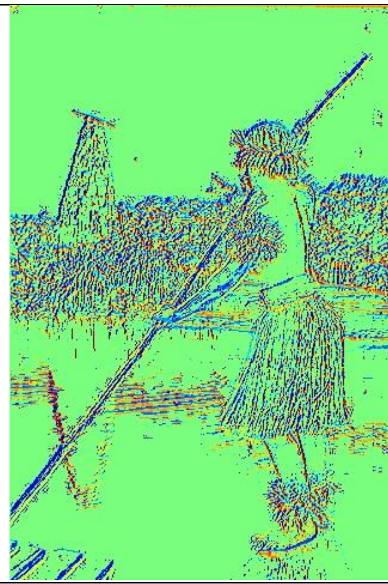
Problem 2: Edge Detection

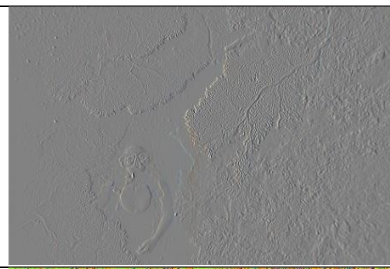
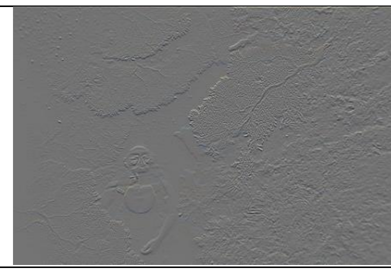
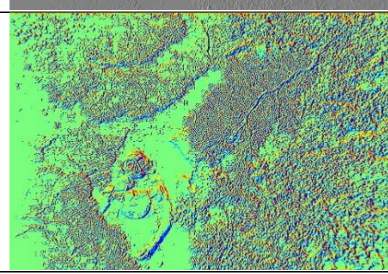
1. Finite operator

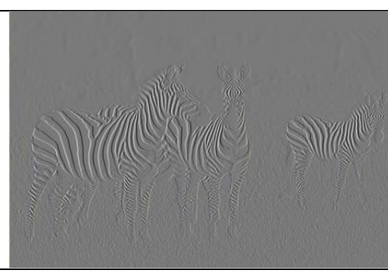
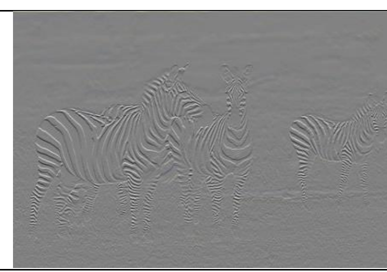
Code `[mag, theta] = difference filter(img)`: see Appendix 2

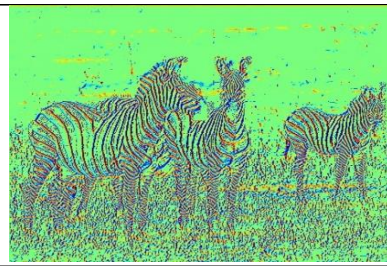
		
		1. Ground truth 2. Filtered by Dx 3. Filtered by Dy 4. Grad magnitude 5. Grad orientation

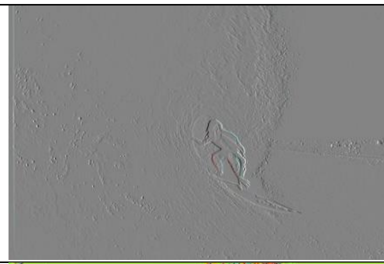
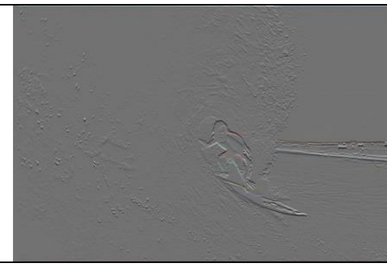
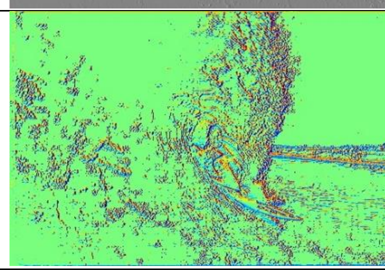
		

		1. Ground truth 2. Filtered by Dx 3. Filtered by Dy 4. Grad magnitude 5. Grad orientation

		
		1. Ground truth 2. Filtered by Dx 3. Filtered by Dy 4. Grad magnitude 5. Grad orientation

		1. Ground truth 2. Filtered by Dx 3. Filtered by Dy 4. Grad magnitude 5. Grad orientation

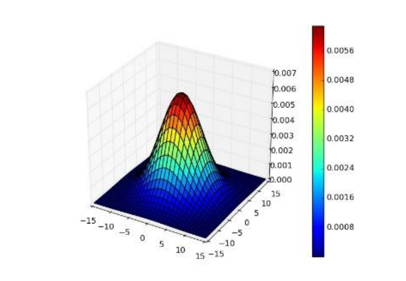
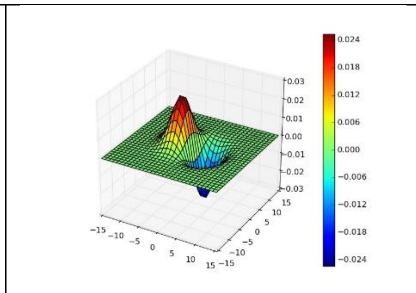
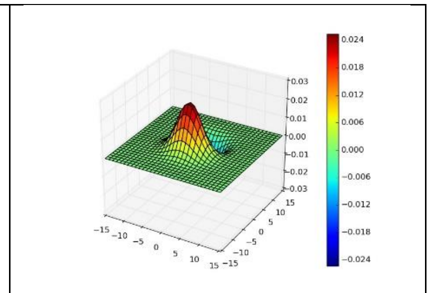
		
		1. Ground truth 2. Filtered by Dx 3. Filtered by Dy 4. Grad magnitude 5. Grad orientation

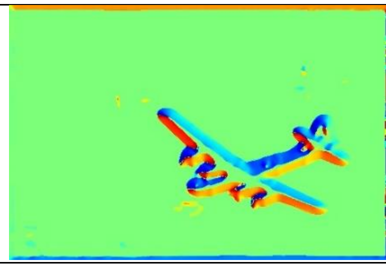
To compute the gradient orientation from x and y filter response: We call the response Rx and Ry, the gradient orientation is given by atan(Ry/Rx).

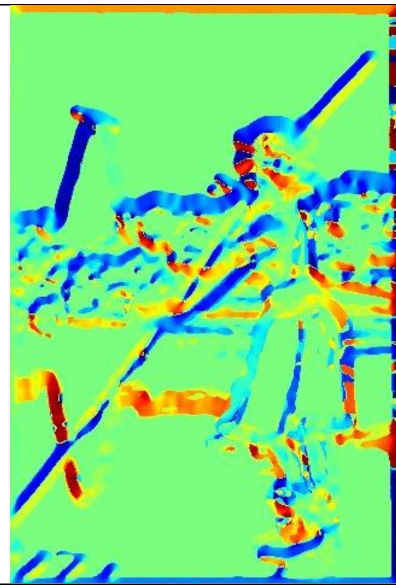
In python, we just simply write them in the complex form (Rx + 1j*Ry) and call the function numpy.angle on this complex array.

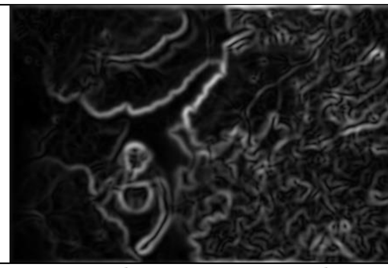
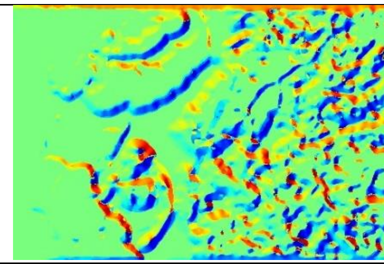
2. Derivative of Gausian

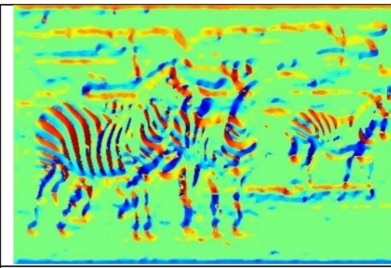
Code `[mag, theta] = derivative gaussian filter(img, sigma)`: See Appendix 2

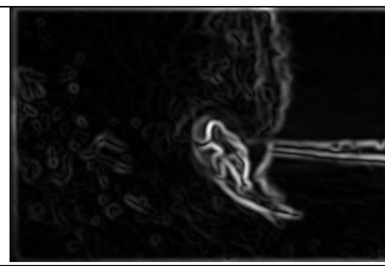
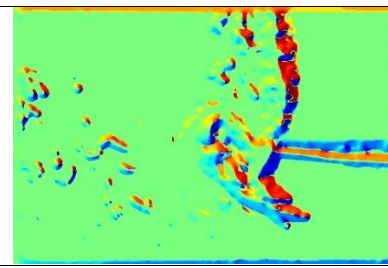
		
Gaussian filter	Derivative over Dx	Derivative over Dy

		
Ground truth	Gradient magnitude	Gradient orientation

		
Ground truth	Gradient magnitude	Gradient orientation

		
Ground truth	Gradient magnitude	Gradient orientation

		
Ground truth	Gradient magnitude	Gradient orientation

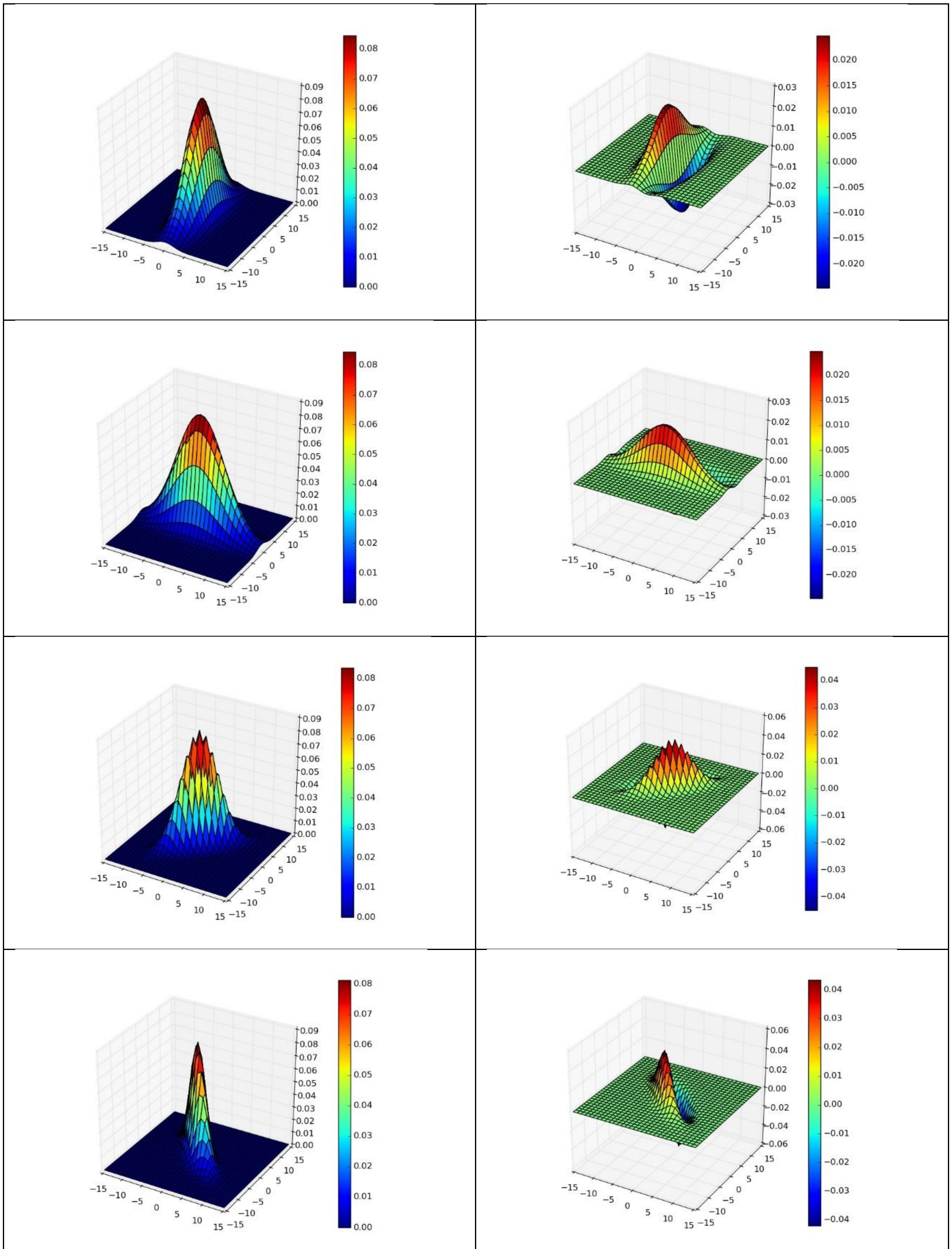
		
Ground truth	Gradient magnitude	Gradient orientation

Observations: compared to the gradient magnitude and orientation of the simple derivative filters, the derivative Gaussian filter produces clearer edge boundary and orientation. The edges are thicker and the orientation has less noises. It is because the image is smoothened before the edges are detected.

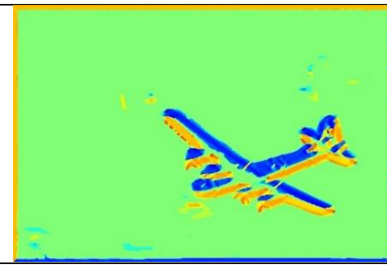
$$G_\sigma * (D_x * I) = D_x * (G_\sigma * I)$$

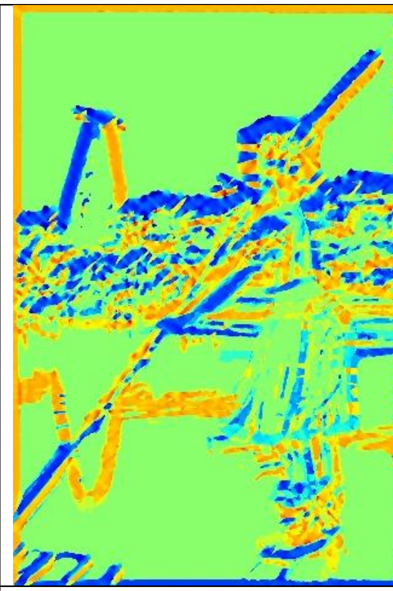
It is because the convolution is the multiplication in frequency domain. The multiplication is commutative so the convolution is commutative in spatial domain. It is important because it allows us to apply as many Gaussian filters as we want (i.e. smoothing image many time) before we take the gradient. By this way, we can combine the Gaussian filters to save computational cost.

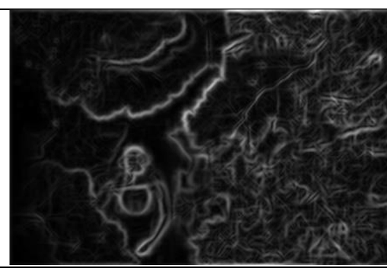
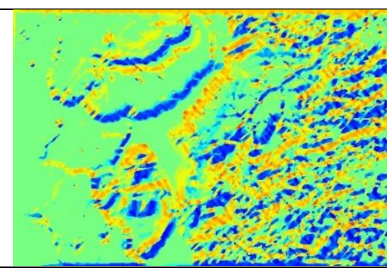
3. Oriented Filters
   Code `[mag,theta] = oriented filter(img)`: See Appendix 2

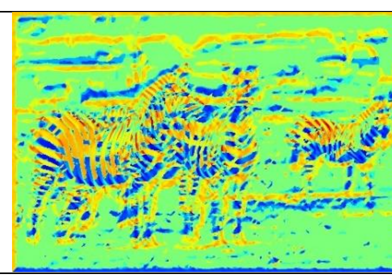


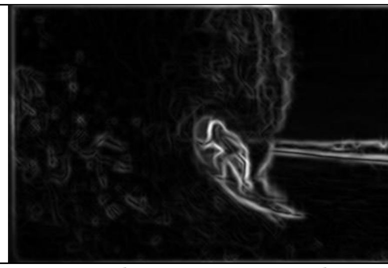
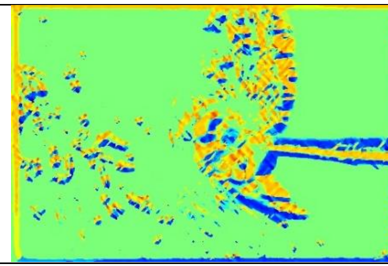
Oriented Gaussian filters	Derivative of oriented Gaussian filters

<table>
<tr><td></td><td></td><td></td></tr>
<tr><td>Ground truth</td><td>Gradient magnitude</td><td>Gradient orientation</td></tr>
</table>

<table>
<tr><td></td><td></td><td></td></tr>
<tr><td>Ground truth</td><td>Gradient magnitude</td><td>Gradient orientation</td></tr>
</table>

<table>
<tr><td></td><td></td><td></td></tr>
<tr><td>Ground truth</td><td>Gradient magnitude</td><td>Gradient orientation</td></tr>
</table>

<table>
<tr><td></td><td></td><td></td></tr>
<tr><td>Ground truth</td><td>Gradient magnitude</td><td>Gradient orientation</td></tr>
</table>

		
Ground truth	Gradient magnitude	Gradient orientation

Explanation of choosing filters: two elongated derivative Gaussian filters Dx and Dy to capture edges in horizontal and vertical directions, two elongated derivative Gaussian filters Dxy and Dyx to capture edges in forward and backward diagonal directions.

To combine the filter responses: we combine them in pairs, two perpendicular filters make a pair. We use L2-norm rule to combine a response pair into magnitude and use complex argument rule to combine the pair into orientation (see problem 2, question 1 answer). Subsequently, we add pairs up equally because we deliberately choose equally distributed directions.

4. Comparison
The result in 3 is close to the 2014 state-of-the-art result from Isola et al. It is not as sharp as the Isola' but if we increase the number of filters in various direction, we can capture more edges and make our result approach the Isola's.
The human annotations have clear boundary at different levels from few to many details.
Our best algorithm does well in detecting edges in various directions.
It does struggle with reducing noise and balancing between the sharpness of detected edge and the smoothening effect of filters.
Some challenges and difficulties with edge detection:
    - Distinguish between noise and edge
    - Some images have many details (ex: hair, grass, etc.)
    - Smoothening makes losing information
    - Computational time increases when we use many oriented filters
    - Etc.

Appendix 1

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from align_image_code import align_images
import numpy as np
from scipy.signal import convolve2d
from skimage import color


def standard_gaussian(X, Y, sigma):
    return 1 / (2 * np.pi * sigma**2) ** 0.5 * np.exp(-(X**2. + Y**2.) / (2 * sigma**2))


def normalize(x):
    return (x - np.min(x)) / (np.max(x) - np.min(x))


def hybrid_image(im1_aligned, im2_aligned, sigma1, sigma2):
    filter_factor = 15
    threshold_1 = sigma1
    threshold_2= sigma2
    std_1 = 5
    std_2 = 5
    x = np.arange(-filter_factor, filter_factor + 1)
    y = np.arange(-filter_factor, filter_factor + 1)
    X, Y = np.meshgrid(x, y)

    low_pass_filter = standard_gaussian(X, Y, std_1)
    low_pass_filter /= np.sum(low_pass_filter)

    impulse_filter = np.zeros((2 * filter_factor + 1, 2 * filter_factor + 1))
    impulse_filter[filter_factor, filter_factor] = 1
    high_pass_filter = standard_gaussian(X, Y, std_2)
    high_pass_filter /= np.sum(high_pass_filter)
    high_pass_filter = impulse_filter - high_pass_filter

    fig = plt.figure()
    ax = fig.gca(projection='3d')
    surf = ax.plot_surface(X, Y, low_pass_filter, rstride=1, cstride=1, cmap=cm.jet)
    fig.colorbar(surf)
    plt.savefig("gaussian_low_pass_filter.jpg")

    fig = plt.figure()
    ax = fig.gca(projection='3d')
    surf = ax.plot_surface(X, Y, high_pass_filter, rstride=1, cstride=1, cmap=cm.jet)
```

```
    fig.colorbar(surf)
    plt.savefig("gaussian_high_pass_filter.jpg")

    im1_filtered = np.zeros(im1_aligned.shape)
    im2_filtered = np.zeros(im2_aligned.shape)

    for i in range(3):
        im1_filtered[:, :, i] = convolve2d(im1_aligned[:, :, i], low_pass_filter,
mode="same")
        im1_filtered_fft = np.fft.fft2(im1_filtered[:, :, i])
        # im1_filtered_fft[(threshold_1 + 1):(-threshold_1), (threshold_1 + 1):(-
threshold_1)] = 0
        im1_filtered_fft[(threshold_1 + 1):(-threshold_1), :] = 0
        im1_filtered_fft[:, (threshold_1 + 1):(-threshold_1)] = 0
        im1_filtered[:, :, i] = np.real(np.fft.ifft2(im1_filtered_fft))

        im2_filtered[:, :, i] = convolve2d(im2_aligned[:, :, i], high_pass_filter,
mode="same")
        im2_filtered_fft = np.fft.fft2(im2_filtered[:, :, i])
        # im2_filtered_fft[1:(threshold_2+1), 1:(threshold_2+1)] = 0
        # im2_filtered_fft[-threshold_2:, -threshold_2:] = 0
        im2_filtered_fft[1:(threshold_2+1), :] = 0
        im2_filtered_fft[-threshold_2:, :] = 0
        im2_filtered_fft[:, 1:(threshold_2+1)] = 0
        im2_filtered_fft[:, -threshold_2:] = 0
        im2_filtered[:, :, i] = np.real(np.fft.ifft2(im2_filtered_fft))

    im1_filtered = normalize(im1_filtered)
    plt.imsave(common_name + "im1_filtered.jpg", im1_filtered, format="jpg")
    plt.figure()
    plt.imshow(im1_filtered)
    plt.show()
    im1_filtered_fft_log =
np.log(np.abs(np.fft.fftshift(np.fft.fft2(color.rgb2gray(im1_filtered)))))
    plt.imsave(common_name + "im1_filtered_fft_log.jpg", im1_filtered_fft_log,
format="jpg")
    plt.imshow(im1_filtered_fft_log)
    plt.show()
    im2_filtered = normalize(im2_filtered)
    plt.imsave(common_name + "im2_filtered.jpg", im2_filtered, format="jpg")
    plt.imshow(im2_filtered)
    plt.show()
    im2_filtered_fft_log =
np.log(np.abs(np.fft.fftshift(np.fft.fft2(color.rgb2gray(im2_filtered)))))
    plt.imsave(common_name + "im2_filtered_fft_log.jpg", im2_filtered_fft_log,
format="jpg")
    plt.imshow(im2_filtered_fft_log)
```

```
    plt.show()
    hybrid = 0.5 * im1_filtered + 0.5 * im2_filtered
    return normalize(hybrid)


if __name__ == "__main__":

    image_pairs = [("malik.png", "papadimitriou.png"), ("cat.png", "dog.png"),
("boy.png", "girl.png")]

    for im1_name, im2_name in image_pairs:

        common_name = im1_name.split('.')[0] + '+' + im2_name.split('.')[0] + '_'

        # First load images

        # high sf
        im1 = plt.imread(im1_name)
        im1 = im1[:,:,:3]
        im1_input_fft_log = np.log(np.abs(np.fft.fftshift(np.fft.fft2(color.rgb2gray(im1)))))
        plt.imsave(common_name + "im1_input_fft_log.jpg", im1_input_fft_log,
format="jpg")
        plt.imshow(im1_input_fft_log)
        plt.show()

        # low sf
        im2 = plt.imread(im2_name)
        im2 = im2[:,:,:3]
        im2_input_fft_log = np.log(np.abs(np.fft.fftshift(np.fft.fft2(color.rgb2gray(im2)))))
        plt.imsave(common_name + "im2_input_fft_log.jpg", im2_input_fft_log,
format="jpg")
        plt.imshow(im2_input_fft_log)
        plt.show()

        # Next align images (this code is provided, but may be improved)
        im1_aligned, im2_aligned = align_images(im1, im2)

        ## You will provide the code below. Sigma1 and sigma2 are arbitrary
        ## cutoff values for the high and low frequencies

        sigma1 = 10
        sigma2 = 5
        hybrid = hybrid_image(im1_aligned, im2_aligned, sigma1, sigma2)

        hybrid_fft_log = np.log(np.abs(np.fft.fftshift(np.fft.fft2(color.rgb2gray(hybrid)))))
        plt.imsave(common_name + "hybrid_fft_log.jpg", hybrid_fft_log, format="jpg")
        plt.imshow(hybrid_fft_log)
```

```
plt.show()
plt.imsave(common_name + "hybrid.jpg", hybrid, format="jpg")
plt.imshow(hybrid)
plt.show()
```

Appendix 2

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from scipy.signal import convolve2d


def normalize(x):
    return (x - np.min(x)) / (np.max(x) - np.min(x))


def standard_gaussian(X, Y, a, b, c, d, sigma_1, sigma_2):
    return 1 / (np.pi * (sigma_1**2 + sigma_2**2)) ** 0.5 * np.exp( -(a * X + b * Y) ** 2. /
(2 * sigma_1**2) - (c * X + d * Y) ** 2. / (2 * sigma_2**2))


def difference_filter(I):
    Dx = np.array([[1, -1]])
    Dy = np.array([[1], [-1]])

    I_filtered_Dx = np.zeros(I.shape)
    I_filtered_Dy = np.zeros(I.shape)
    for i in range(3):
        I_filtered_Dx[:, :, i] = convolve2d(I[:, :, i], Dx, mode="same")
        I_filtered_Dy[:, :, i] = convolve2d(I[:, :, i], Dy, mode="same")

    return I_filtered_Dx, I_filtered_Dy


def derivative_gaussian_filter(I,sigma):
    Dx = np.array([[1, -1]])
    Dy = np.array([[1], [-1]])

    filter_factor = 15
    x = np.arange(-filter_factor, filter_factor + 1)
    y = np.arange(-filter_factor, filter_factor + 1)
    X, Y = np.meshgrid(x, y)

    gaussian_filter = standard_gaussian(X, Y, 1, 0, 0, 1, sigma, sigma)
    gaussian_filter_Dx = convolve2d(gaussian_filter, Dx, mode="same")
    gaussian_filter_Dy = convolve2d(gaussian_filter, Dy, mode="same")

    fig = plt.figure()
    ax = fig.gca(projection='3d')
    surf = ax.plot_surface(X, Y, gaussian_filter_Dx, rstride=1, cstride=1, cmap=cm.jet)
```

```
    fig.colorbar(surf)
    plt.savefig("derivative_gaussian_filter_Dx.jpg")

    fig = plt.figure()
    ax = fig.gca(projection='3d')
    surf = ax.plot_surface(X, Y, gaussian_filter_Dy, rstride=1, cstride=1, cmap=cm.jet)
    fig.colorbar(surf)
    plt.savefig("derivative_gaussian_filter_Dy.jpg")

    I_filtered_Dx = np.zeros(I.shape)
    I_filtered_Dy = np.zeros(I.shape)
    for i in range(3):
        I_filtered_Dx[:, :, i] = convolve2d(I[:, :, i], gaussian_filter_Dx, mode="same")
        I_filtered_Dy[:, :, i] = convolve2d(I[:, :, i], gaussian_filter_Dy, mode="same")

    return I_filtered_Dx, I_filtered_Dy


def oriented_filter(I):
    Dx = np.array([[1, -1]])
    Dy = np.array([[1], [-1]])
    Dxy = np.array([[0, 1], [-1, 0]])
    Dyx = np.array([[1, 0], [0, -1]])

    filter_factor = 15
    x = np.arange(-filter_factor, filter_factor + 1)
    y = np.arange(-filter_factor, filter_factor + 1)
    X, Y = np.meshgrid(x, y)

    gaussian_filter_x = standard_gaussian(X, Y, 1, 0, 0, 1, 2, 6)
    gaussian_filter_Dx = convolve2d(gaussian_filter_x, Dx, mode="same")
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    surf = ax.plot_surface(X, Y, gaussian_filter_x, rstride=1, cstride=1, cmap=cm.jet)
    fig.colorbar(surf)
    plt.savefig("oriented_gaussian_filter_Dx.jpg")
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    surf = ax.plot_surface(X, Y, gaussian_filter_Dx, rstride=1, cstride=1, cmap=cm.jet)
    fig.colorbar(surf)
    plt.savefig("derivative_oriented_gaussian_filter_Dx.jpg")

    gaussian_filter_y = standard_gaussian(X, Y, 1, 0, 0, 1, 6, 2)
    gaussian_filter_Dy = convolve2d(gaussian_filter_y, Dy, mode="same")
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    surf = ax.plot_surface(X, Y, gaussian_filter_y, rstride=1, cstride=1, cmap=cm.jet)
```

```
fig.colorbar(surf)
plt.savefig("oriented_gaussian_filter_Dy.jpg")
fig = plt.figure()
ax = fig.gca(projection='3d')
surf = ax.plot_surface(X, Y, gaussian_filter_Dy, rstride=1, cstride=1, cmap=cm.jet)
fig.colorbar(surf)
plt.savefig("derivative_oriented_gaussian_filter_Dy.jpg")

gaussian_filter_xy = standard_gaussian(X, Y, 1, -1, 1, 1, 2, 6)
gaussian_filter_Dxy = convolve2d(gaussian_filter_xy, Dxy, mode="same")
fig = plt.figure()
ax = fig.gca(projection='3d')
surf = ax.plot_surface(X, Y, gaussian_filter_xy, rstride=1, cstride=1, cmap=cm.jet)
fig.colorbar(surf)
plt.savefig("oriented_gaussian_filter_Dxy.jpg")
fig = plt.figure()
ax = fig.gca(projection='3d')
surf = ax.plot_surface(X, Y, gaussian_filter_Dxy, rstride=1, cstride=1, cmap=cm.jet)
fig.colorbar(surf)
plt.savefig("derivative_oriented_gaussian_filter_Dxy.jpg")

gaussian_filter_yx = standard_gaussian(X, Y, 1, -1, 1, 1, 6, 2)
gaussian_filter_Dyx = convolve2d(gaussian_filter_yx, Dyx, mode="same")
fig = plt.figure()
ax = fig.gca(projection='3d')
surf = ax.plot_surface(X, Y, gaussian_filter_yx, rstride=1, cstride=1, cmap=cm.jet)
fig.colorbar(surf)
plt.savefig("oriented_gaussian_filter_Dyx.jpg")
fig = plt.figure()
ax = fig.gca(projection='3d')
surf = ax.plot_surface(X, Y, gaussian_filter_Dyx, rstride=1, cstride=1, cmap=cm.jet)
fig.colorbar(surf)
plt.savefig("derivative_oriented_gaussian_filter_Dyx.jpg")

I_filtered_Dx = np.zeros(I.shape)
I_filtered_Dy = np.zeros(I.shape)
I_filtered_Dxy = np.zeros(I.shape)
I_filtered_Dyx = np.zeros(I.shape)
for i in range(3):
    I_filtered_Dx[:, :, i] = convolve2d(I[:, :, i], gaussian_filter_Dx, mode="same")
    I_filtered_Dy[:, :, i] = convolve2d(I[:, :, i], gaussian_filter_Dy, mode="same")
    I_filtered_Dxy[:, :, i] = convolve2d(I[:, :, i], gaussian_filter_Dxy, mode="same")
    I_filtered_Dyx[:, :, i] = convolve2d(I[:, :, i], gaussian_filter_Dyx, mode="same")

return I_filtered_Dx, I_filtered_Dy, I_filtered_Dxy, I_filtered_Dyx
```

```
if __name__ == "__main__":

    image_name_list = ["bsds_3096.jpg", "bsds_300091.jpg", "bsds_253027.jpg",
"bsds_156065.jpg", "bsds_101087.jpg"]

    for image_name in image_name_list:
        name = image_name.split('.')[0] + "_"
        im = plt.imread(image_name)

        # Simple derivative filters

        im_filtered_Dx, im_filtered_Dy = difference_filter(im)

        im_filtered_Dx_normalized = normalize(im_filtered_Dx)
        plt.imsave(name + "im_filtered_Dx.jpg", im_filtered_Dx_normalized,
format="jpg")
        plt.imshow(im_filtered_Dx_normalized)
        plt.show()

        im_filtered_Dy_normalized = normalize(im_filtered_Dy)
        plt.imsave(name + "im_filtered_Dy.jpg", im_filtered_Dy_normalized,
format="jpg")
        plt.imshow(im_filtered_Dy_normalized)
        plt.show()

        im_gradient_magnitude = (im_filtered_Dx ** 2 + im_filtered_Dy ** 2) ** 0.5
        im_gradient_magnitude_2d = np.sum(im_gradient_magnitude, axis=2)
        plt.imsave(name + "im_gradient_magnitude.jpg", im_gradient_magnitude_2d,
format="jpg", cmap="gray")
        plt.imshow(im_gradient_magnitude_2d, cmap="gray")
        plt.show()

        im_gradient_orientation = np.angle(im_filtered_Dx + 1j * im_filtered_Dy)
        im_gradient_orientation[np.where(im_gradient_magnitude < 10)] = 0
        im_gradient_orientation_2d = np.sum(im_gradient_orientation, axis=2)
        plt.imsave(name + "im_gradient_orientation.jpg", im_gradient_orientation_2d,
format="jpg")
        plt.imshow(im_gradient_orientation_2d)
        plt.show()

        # Derivative Gaussian filters

        im_gaussian_filtered_Dx, im_gaussian_filtered_Dy =
derivative_gaussian_filter(im, 3)
        im_gaussian_filtered_Dx_normalized = normalize(im_gaussian_filtered_Dx)
        plt.imsave(name + "im_gaussian_filtered_Dx.jpg",
im_gaussian_filtered_Dx_normalized, format="jpg")
```

```
    plt.figure()
    plt.imshow(im_gaussian_filtered_Dx_normalized)
    plt.show()

    im_gaussian_filtered_Dy_normalized = normalize(im_gaussian_filtered_Dy)
    plt.imsave(name + "im_gaussian_filtered_Dy.jpg",
im_gaussian_filtered_Dy_normalized, format="jpg")
    plt.imshow(im_gaussian_filtered_Dy_normalized)
    plt.show()

    im_gaussian_gradient_magnitude = (im_gaussian_filtered_Dx ** 2 +
im_gaussian_filtered_Dy ** 2) ** 0.5
    im_gaussian_gradient_magnitude_2d =
np.sum(im_gaussian_gradient_magnitude, axis=2)
    plt.imsave(name + "im_gaussian_gradient_magnitude.jpg",
im_gaussian_gradient_magnitude_2d, cmap="gray")
    plt.imshow(im_gaussian_gradient_magnitude_2d, cmap="gray")
    plt.show()

    im_gaussian_gradient_orientation = np.angle(im_gaussian_filtered_Dx + 1j *
im_gaussian_filtered_Dy)
    im_gaussian_gradient_orientation[np.where(im_gaussian_gradient_magnitude <
20)] = 0
    im_gaussian_gradient_orientation_2d =
np.sum(im_gaussian_gradient_orientation, axis=2)
    plt.imsave(name + "im_gaussian_gradient_orientation.jpg",
im_gaussian_gradient_orientation_2d)
    plt.imshow(im_gaussian_gradient_magnitude_2d)
    plt.show()

    # Oriented Gaussian filters

    im_oriented_gaussian_filtered_Dx, im_oriented_gaussian_filtered_Dy,
im_oriented_gaussian_filtered_Dxy, im_oriented_gaussian_filtered_Dyx =
oriented_filter(im)
    im_oriented_gaussian_filtered_Dx_normalized =
normalize(im_oriented_gaussian_filtered_Dx)
    plt.imsave(name + "im_oriented_gaussian_filtered_Dx.jpg",
im_oriented_gaussian_filtered_Dx_normalized, format="jpg")
    plt.figure()
    plt.imshow(im_oriented_gaussian_filtered_Dx_normalized)
    plt.show()

    im_oriented_gaussian_filtered_Dy_normalized =
normalize(im_oriented_gaussian_filtered_Dy)
    plt.imsave(name + "im_oriented_gaussian_filtered_Dy.jpg",
im_oriented_gaussian_filtered_Dy_normalized, format="jpg")
```

```
    plt.imshow(im_oriented_gaussian_filtered_Dy_normalized)
    plt.show()

    im_oriented_gaussian_filtered_Dxy_normalized =
normalize(im_oriented_gaussian_filtered_Dxy)
    plt.imsave(name + "im_oriented_gaussian_filtered_Dxy.jpg",
im_oriented_gaussian_filtered_Dxy_normalized, format="jpg")
    plt.imshow(im_oriented_gaussian_filtered_Dxy_normalized)
    plt.show()

    im_oriented_gaussian_filtered_Dyx_normalized =
normalize(im_oriented_gaussian_filtered_Dyx)
    plt.imsave(name + "im_oriented_gaussian_filtered_Dyx.jpg",
im_oriented_gaussian_filtered_Dyx_normalized, format="jpg")
    plt.imshow(im_oriented_gaussian_filtered_Dyx_normalized)
    plt.show()

    im_oriented_gaussian_gradient_magnitude = (im_oriented_gaussian_filtered_Dx
** 2 + im_oriented_gaussian_filtered_Dy ** 2
                                   + im_oriented_gaussian_filtered_Dxy ** 2 +
im_oriented_gaussian_filtered_Dyx ** 2) ** 0.5
    im_oriented_gaussian_gradient_magnitude_2d =
np.sum(im_oriented_gaussian_gradient_magnitude, axis=2)
    plt.imsave(name + "im_oriented_gaussian_gradient_magnitude.jpg",
im_oriented_gaussian_gradient_magnitude_2d, format="jpg", cmap="gray")
    plt.imshow(im_oriented_gaussian_gradient_magnitude_2d, cmap="gray")
    plt.show()

    im_oriented_gaussian_gradient_orientation =
np.angle(im_oriented_gaussian_filtered_Dx + 1j * im_oriented_gaussian_filtered_Dy)
\
                                   + np.angle(im_oriented_gaussian_filtered_Dxy + 1j *
im_oriented_gaussian_filtered_Dyx)

im_oriented_gaussian_gradient_orientation[np.where(im_oriented_gaussian_gradien
t_magnitude < 20)] = 0
    im_oriented_gaussian_gradient_orientation_2d =
np.sum(im_oriented_gaussian_gradient_orientation, axis=2)
    plt.imsave(name + "im_oriented_gaussian_gradient_orientation.jpg",
im_oriented_gaussian_gradient_orientation_2d, format="jpg")
    plt.imshow(im_oriented_gaussian_gradient_orientation_2d)
    plt.show()
```