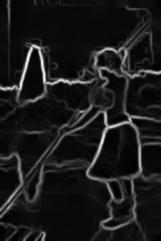
**2 Edge Detection**

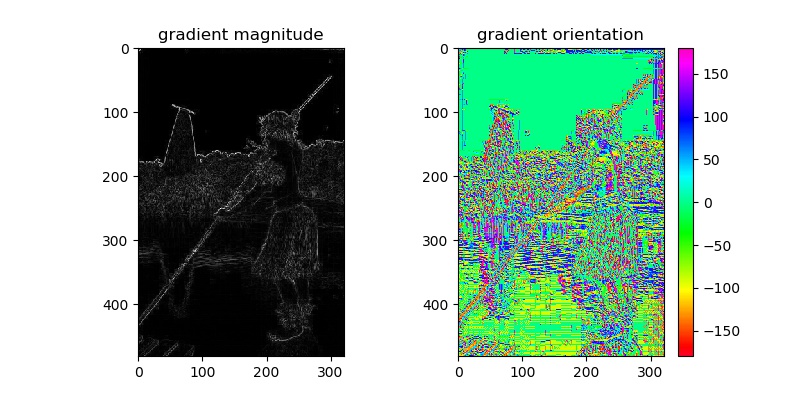
One of the state-of-art edge detector examples is shown below as benchmark:

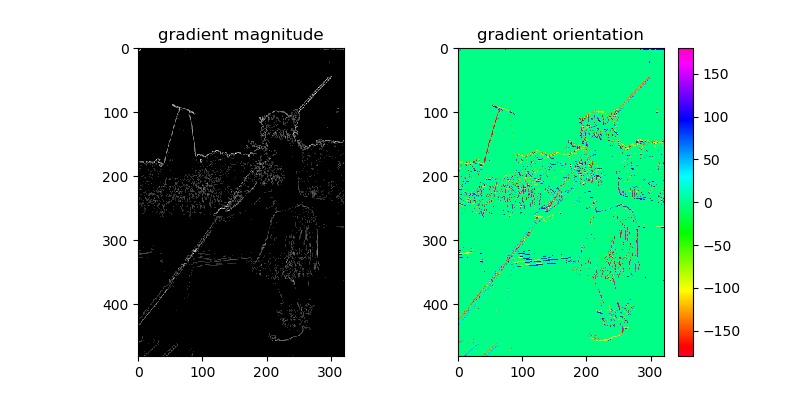
**** 

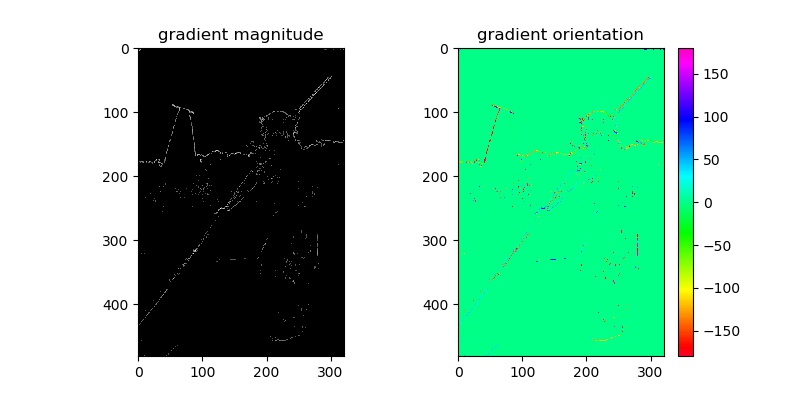
**1\ Finite operator**

I use arctan2 function to calculate the angle along each RGB channel and then take an average of the three channels. We cannot use norm here, since norm are positive, but angles can be positive or negative from -180 to 180 degree. Average makes more sense than norm for angles.

Below are gradient magnitude and gradient orientation mappings for several different thresholds, for an example image:



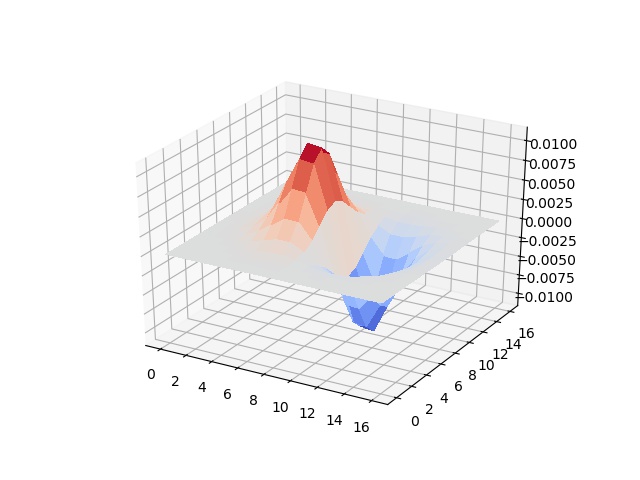
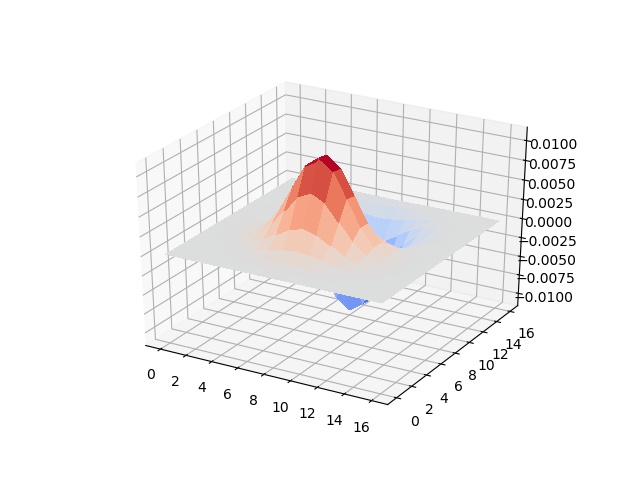




**2\ Derivative of Gaussian**

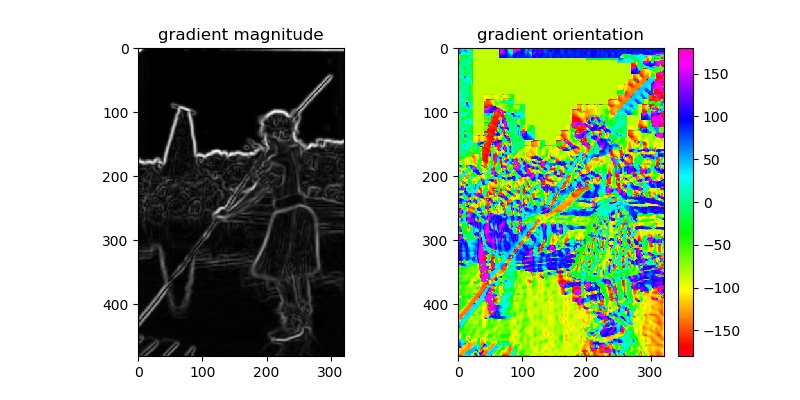
Differentiation can be calculated as convolution, and since convolution are commutative and associative, we can convolve any two of the matrices as a first step. But we want to begin with small matrices, and leave the large image matrix as last operator, so that we don’t have to deal with large matrix multiple times. In this way, the speed of calculation is much faster.

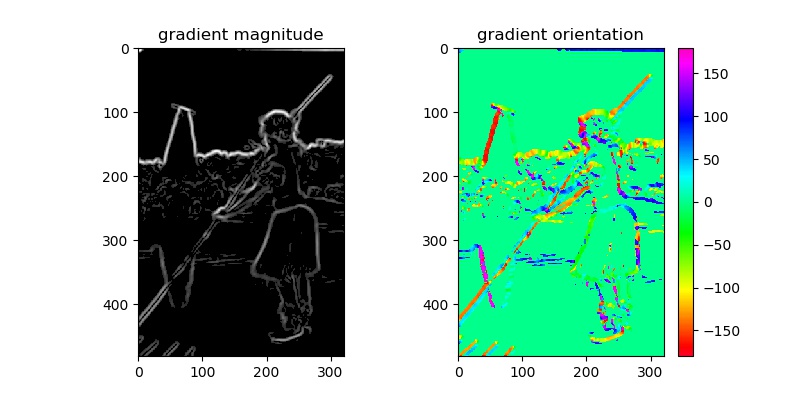
Below are the plots for Derivative of Gaussian along x and y direction, respectively.

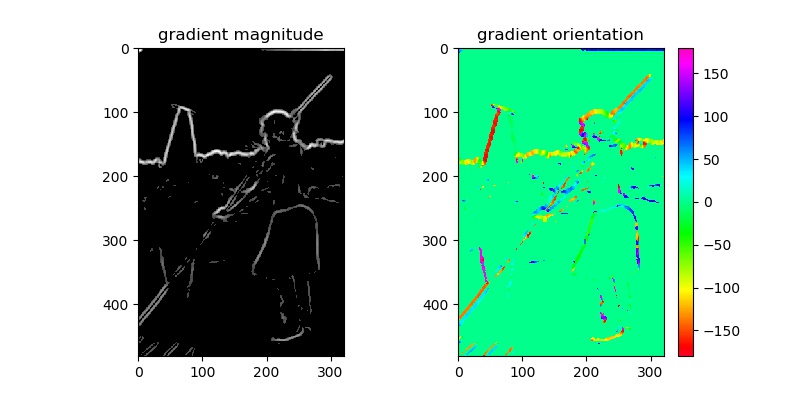


Below are gradient magnitude and gradient orientation mappings for several different thresholds, for an example image:

We can see that using derivative of gaussians, the resulting filtered images are less noisy with clear edges. But we don’t want the details in some of the textures detected as edges, for example, the details in the trees at riverside. By using a higher threshold, we somehow remove the details in the textures, but at the same time remove some weaker edges as well.





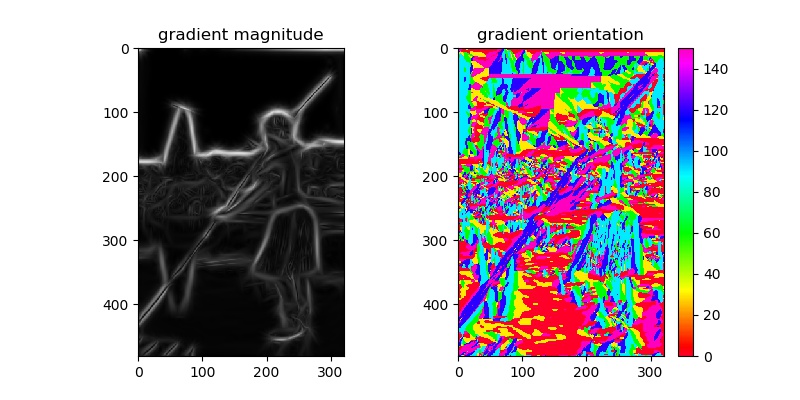


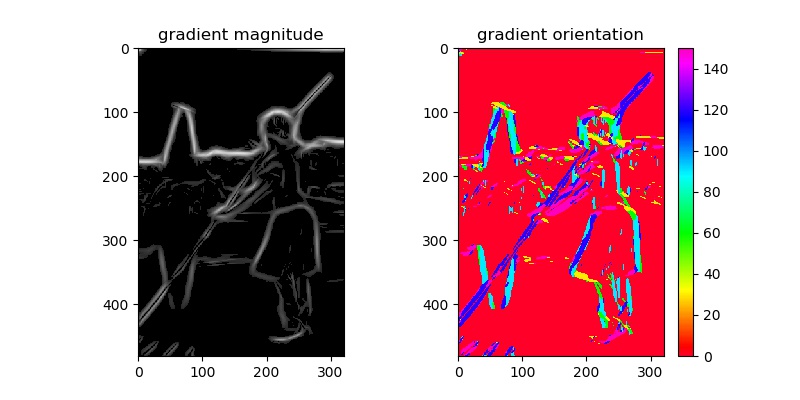
**3\ Oriented Filters**

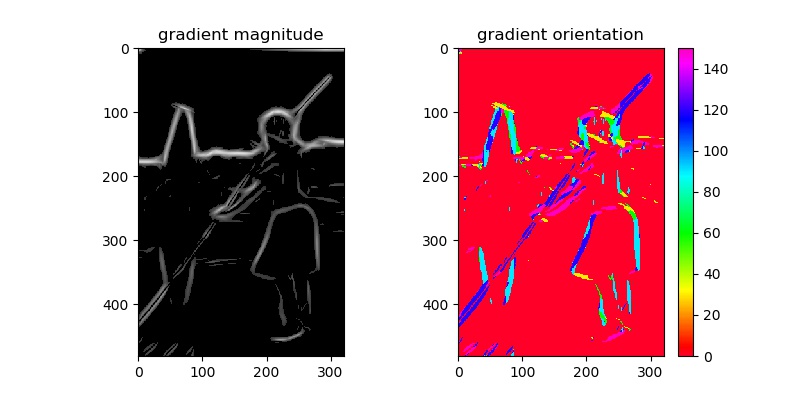
I use 18 Elongated Gaussian Derivative filter along 6 different directions(0, 30, 60, 90, 120, 150 degree) and 3 different sizes(sigma2 = sqrt(2), 2\* sqrt(2), 3\* sqrt(2), sigma1 = 3\*sigma2). These are the 18 elongated Gaussian 1st derivative used in Leung-Malik Filter Bank.

Then at each pixel, I choose the largest response from each filter as element in the gradient magnitude matrix, and I use the angle of the oriented filter with the largest response as element in the gradient orientation matrix.

The mapping are shown below for several threshold. We can observe that Oriented Filters are better at smoothing the details in textures than the DoG filters. For example, in the filtered image without using threshold, there are still some details in the leaves on his arm, but the details are somewhat smoothed.



****

****

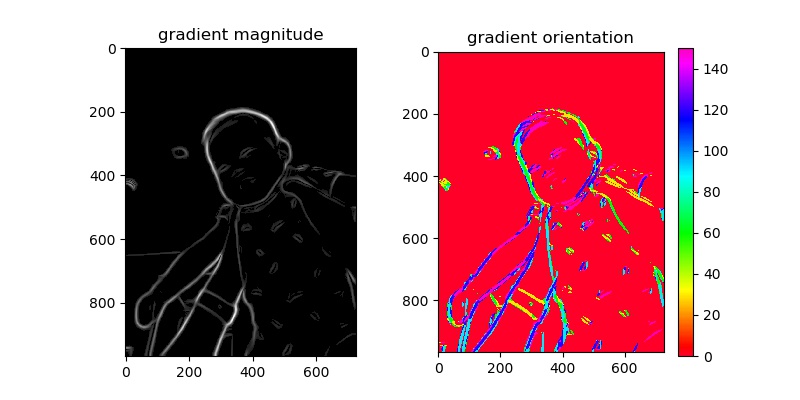
**4\ Compare with state-of-art result and human annotation**

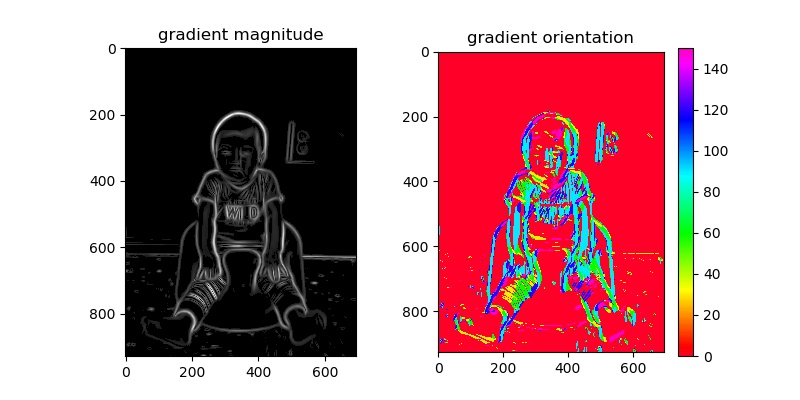
Human labelers will ignore any texture. The start-of-art results are better at ignoring textures than my best algorithm. My best algorithm is good at finding edges of high contrast, but struggles with ignoring the details in textures. Human labelers will probably ignore the shadows on clothes, but my algorithm are not able to ignore them since sometimes they have large contrast. Human will probably ignore stripes on clothes or on zebras, but these are all high contrast and hard to ignore by my algorithms.

We may need some algorithm that can recognize textures and and remove them from edge response. I think this is the biggest challenge in edge detection.

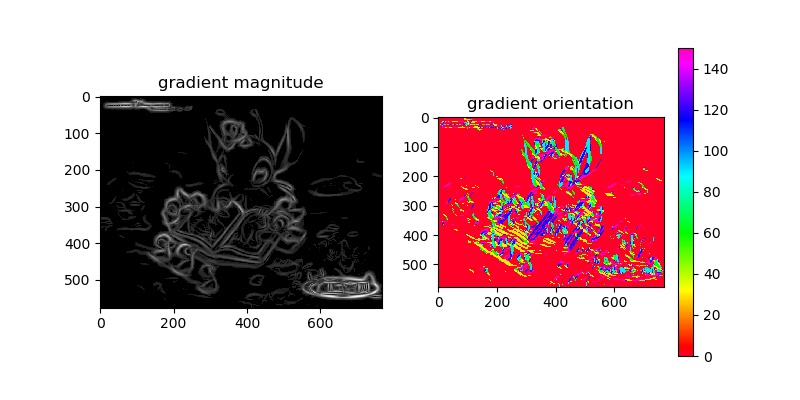
Below I will attach some images and their filtered images with best performance.



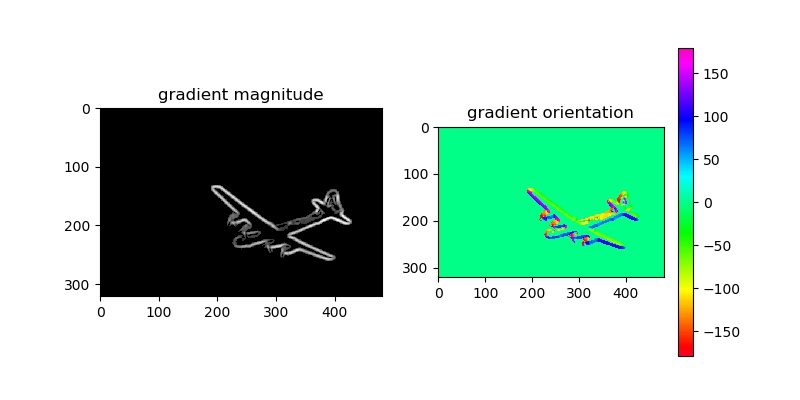




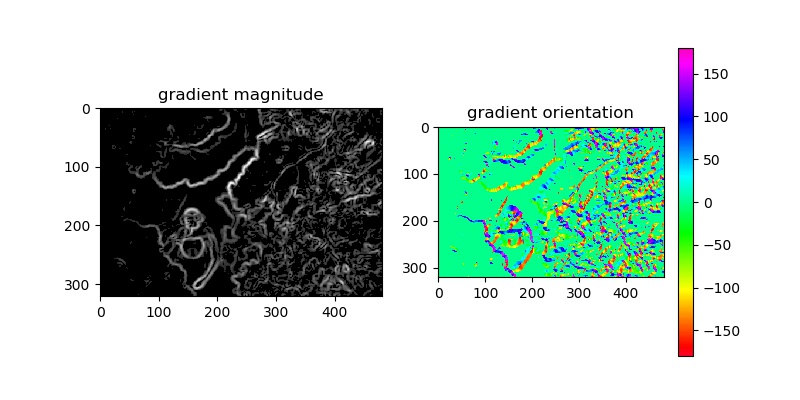




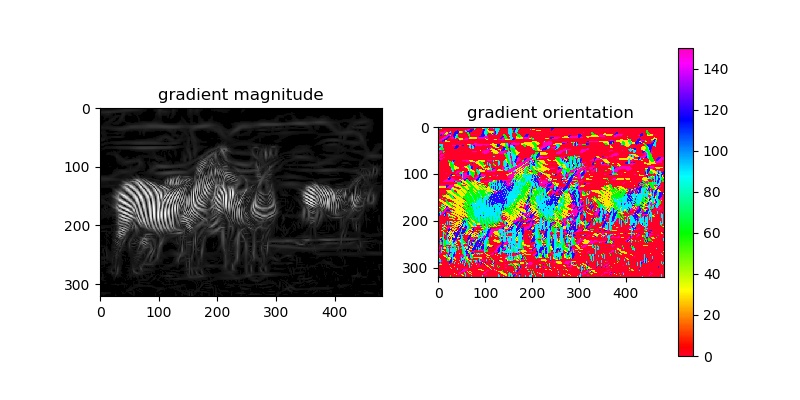




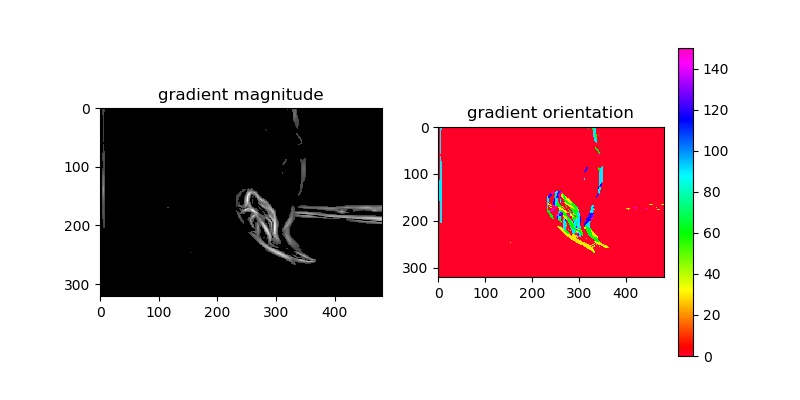












import numpy as np

from scipy import signal

import matplotlib.pyplot as plt

import os

from numpy.linalg import norm, inv

from matplotlib import cm

from math import pi, sqrt, exp

from mpl\_toolkits.mplot3d import Axes3D

from math import cos, sin

**def gaussian\_kernel(sigma, truncate=4.0):**

n = int(2\*sigma\*truncate + 1)

gaussian = np.zeros((n, n))

mean = np.array([n//2, n//2])

for i in range(n):

for j in range(n):

gaussian[i, j] = 1/(2\*pi\*sigma\*\*2)\*exp(-((i-mean[0])\*\*2+(j- mean[1])\*\*2)/2/sigma\*\*2)

return gaussian/gaussian.sum()

**def elongated\_gaussian\_derivative(sigma1, sigma2, theta, truncate=4.0):**

n = int(sigma1\*truncate)

x, y = np.meshgrid(np.arange(-n, n+1), np.arange(-n, n+1))

orgpts = np.vstack((x.reshape(-1), y.reshape(-1)))

theta = theta/180\*pi

rotation = np.array([[cos(theta), sin(theta)], [-sin(theta), cos(theta)]])

rotpts = rotation @ orgpts

gx = np.exp(-1/2\*(rotpts[0, :])\*\*2/sigma1\*\*2)

gy = np.exp(-1/2\*(rotpts[1, :])\*\*2/sigma2\*\*2) \* (-(rotpts[1, :])/sigma2\*\*2)

egd = (gx \* gy).reshape(2\*n+1, 2\*n+1)

egd\_plus = egd \* (egd>0)

return egd/egd\_plus.sum()

**def difference\_filter(I):**

dx = np.array([[1, -1]])

dy = np.array([[1], [-1]])

return get\_gradient\_images(I, dx, dy)

**def derivative\_gaussian\_filter(I, sigma):**

dx = np.array([[1, -1]])

dy = np.array([[1], [-1]])

dogx = signal.convolve2d(gaussian\_kernel(sigma), dx, boundary='symm', mode='same')

dogy = signal.convolve2d(gaussian\_kernel(sigma), dy, boundary='symm', mode='same')

plot\_filter(dogx, 'derivative\_gaussian\_x.jpg')

plot\_filter(dogy, 'derivative\_gaussian\_y.jpg')

return get\_gradient\_images(I, dogx, dogy)

**def get\_gradient\_images(I, filterx, filtery):**

img\_x = np.zeros(I.shape)

img\_y = np.zeros(I.shape)

for i in range(3):

img\_x[:, :, i] = signal.convolve2d(I[:, :, i], filterx, boundary='symm', mode='same')

img\_y[:, :, i] = signal.convolve2d(I[:, :, i], filtery, boundary='symm', mode='same')

img\_x\_norm = norm(img\_x, axis = 2)

img\_y\_norm = norm(img\_y, axis = 2)

img\_mag = np.concatenate((img\_x, img\_y), axis = 2)

img\_mag = norm(img\_mag, axis = 2)

img\_orient = np.arctan2(img\_y, img\_x).mean(axis=2)/pi\*180

return img\_x\_norm, img\_y\_norm, img\_mag, img\_orient

**def oriented\_filter(I):**

thetas = np.array([0, 30, 60, 90, 120, 150])

sigma1s = 3\*sqrt(2) \*\* np.array([1, 2, 3])

theta, sigma1 = np.meshgrid(thetas, sigma1s)

theta = theta.reshape(-1)

sigma1 = sigma1.reshape(-1)

img\_mag = np.zeros((I.shape[0], I.shape[1], len(theta)))

img\_orient = np.zeros((I.shape[0], I.shape[1], len(theta)))

for i in range(len(theta)):

egd = elongated\_gaussian\_derivative(sigma1[i], sigma1[i]/3, theta[i])

img\_mag[:, :, i] = convolve3channel(I, egd)

img\_orient[:, :, i] = theta[i]

img\_max\_repeat= np.repeat(img\_mag.max(axis=2)[:, :, np.newaxis], len(theta), axis=2)

indexes = np.equal(img\_mag, img\_max\_repeat)

img\_orient = (img\_orient \* indexes).max(axis=2)

img\_mag = img\_mag.max(axis=2)

return img\_mag, img\_orient

**def convolve3channel(I, filter\_):**

filtered = np.zeros(I.shape)

for i in range(3):

filtered[:, :, i] = signal.convolve2d(I[:, :, i], filter\_, boundary='symm', mode='same')

filtered = norm(filtered, axis = 2)

return filtered