

EECS 504 PS1: Filtering

Please provide the following information (e.g. Andrew Owens, ahowens):

[Aohan] [Mei], [rogermei]

Introduction

We'll provide you with starter code, like this, in a Jupyter notebook for most problem sets. Please fill in the code to complete the assignment, and submit your notebook to Canvas as a .ipynb file. You can, of course, initially write your code offline in an editor like Emacs or Vim -- we'd just like the final output to be in a notebook format to make grading more consistent. You can also host your notebook locally.

Please note that *we won't run your code*. The notebook you submit should already contain all of the results we ask for. In particular, the visualizations of edge responses and blurred images should be computed *before you submit*. Also, please do not include long, unnecessary outputs (a few print statements and visualizations are fine, but pages of debugging messages make grading difficult).

If you'd like to preview what your notebook will look like when we grade it, follow the directions [here](https://stackoverflow.com/questions/15998491/how-to-convert-ipython-notebooks-to-pdf-and-html) (<https://stackoverflow.com/questions/15998491/how-to-convert-ipython-notebooks-to-pdf-and-html>) for converting the .ipynb notebook into an HTML file. We'll be running a command similar to:

```
ipython nbconvert --to html ps1.ipynb
```

First steps

Run the following code to import the modules that you'll need, and to download the images. You are welcome to modify this code (e.g. by importing more modules).

```
In [0]: import numpy as np, matplotlib as mpl, matplotlib.pyplot as plt, urllib,
os
import scipy.ndimage # For image filtering
import imageio # For loading images

# Download the images that you'll need
base_url = 'https://web.eecs.umich.edu/~ahowens/eecs504/w20/psets/ps1/im
s'
for name in ['dog-1.jpg', 'dog-2.jpg', 'apple.jpg']:
    with open(name, 'wb') as out:
        url = os.path.join(base_url, name)
        out.write(urllib.request.urlopen(url).read())
```

Problem 1.2: Pet edge detection (a)

```

In [25]: # You can upload images yourself or load them from URLs
im = imageio.imread('dog-1.jpg')
# Convert to grayscale. We'll use floats in [0, 1].
im = im.mean(2)/255.
dx = [1, -1]
dy = np.transpose([1,-1])
# Your code here!
def convolve(im,h):
    if (h == dy).all():
        im = np.transpose(im)
        h = np.transpose(h)
        I_grad = np.zeros(np.shape(im))
        for i in range(np.shape(im)[1]):
            vec = im[:,i]
            for j in range(np.shape(im)[0]):
                for k in range(2):
                    if j-k >= 0:
                        I_grad[j,i] = I_grad[j,i] + h[k]*vec[j-k]
    if (h == dy).all():
        return np.transpose(I_grad)
    else:
        return I_grad

# Convolve the image with horizontal and vertical gradient filters
Ix = convolve(im, dx)
Iy = convolve(im, dy)
edges = Ix**2. + Iy**2.

# Visualize edge maps using matplotlib
plt.figure()
plt.title('Input image')
plt.axis('off')
plt.imshow(im, cmap = 'gray', vmin = 0, vmax = 1)

plt.figure()
plt.axis('off')
plt.title('Ix')
plt.imshow(Ix)

plt.figure()
plt.title('Iy')
plt.axis('off')
plt.imshow(Iy)

plt.figure()
plt.title('Edges')
plt.axis('off')
# Please visualize edge responses using this range of values.
plt.imshow(edges, vmin = 0., vmax = np.percentile(edges, 99))

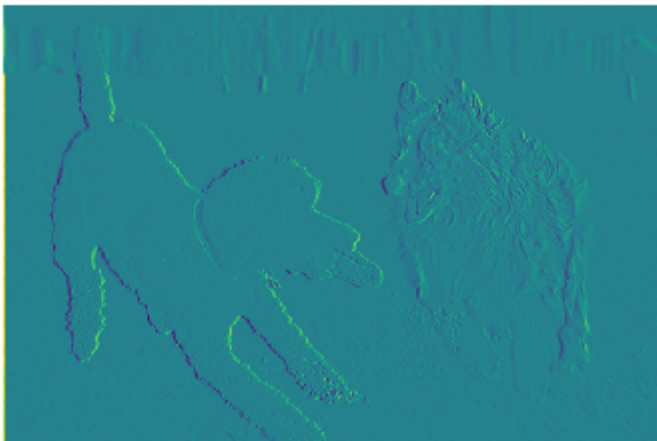
```

```
Out[25]: <matplotlib.image.AxesImage at 0x7f219202b898>
```

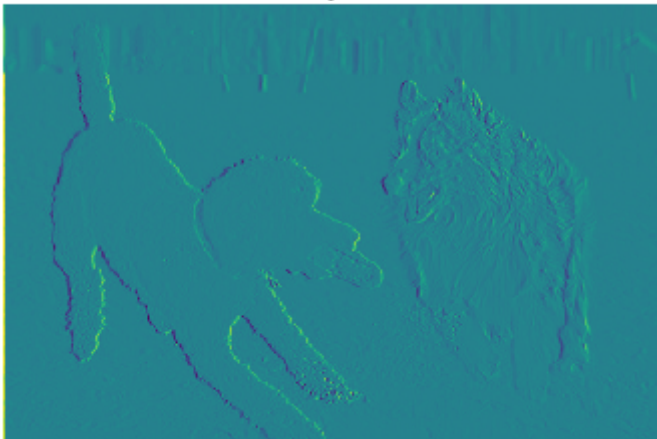
Input image

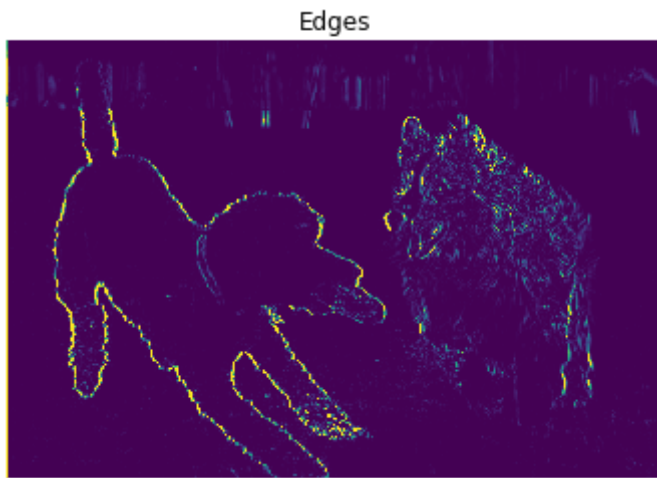


lx



ly





Problem 1.2: Pet edge detection (c)

Please visualize the images and edges with matplotlib, using code similar to what we provided in (a).

```

In [14]: im = imageio.imread('dog-2.jpg').mean(2)/255.
from scipy import ndimage
# Your code here!
dx = np.array([[1,-1]])
dy = np.transpose(dx)
Ix = ndimage.convolve(im,dx)
Iy = ndimage.convolve(im,dy)
plt.figure()
plt.title('original figure')
plt.axis('off')
original_edge = Ix**2. + Iy**2.
plt.imshow(original_edge,vmin = 0., vmax = np.percentile(original_edge,99))

def Gaussian_kernel(dimx,dimy,sigma):
    gridx = np.linspace(-(dimx-1)/2,(dimx-1)/2,11)
    gridy = np.linspace(-(dimy-1)/2,(dimy-1)/2,11)
    [h1,h2] = np.meshgrid(gridx,gridy)
    hg = np.exp(-(h1**2+h2**2)/(2*sigma))
    return hg
Gaussian_filter = Gaussian_kernel(11,11,2)
im_g = ndimage.convolve(im,Gaussian_filter)
Ix = ndimage.convolve(im_g,dx)
Iy = ndimage.convolve(im_g,dy)
edges_g = Ix**2. + Iy**2.

plt.figure()
plt.title('Gaussian blurred image')
plt.axis('off')
plt.imshow(im_g, vmin = 0., vmax = np.percentile(im_g,99))

plt.figure()
plt.title('Gaussian blurred Edges')
plt.axis('off')
plt.imshow(edges_g, vmin = 0., vmax = np.percentile(edges_g, 99))

box_filter = np.ones([11,11])/11**2
im_b = ndimage.convolve(im,box_filter)
Ix = ndimage.convolve(im_b,dx)
Iy = ndimage.convolve(im_b,dy)
box_edge = Ix**2.+Iy**2.

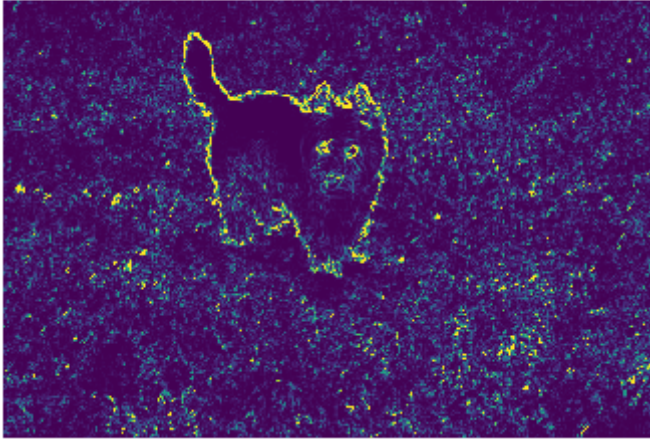
plt.figure()
plt.title('box filtered image')
plt.axis('off')
plt.imshow(im_b, vmin = 0., vmax = np.percentile(im_b,99))

plt.figure()
plt.title('box filtered edges')
plt.axis('off')
plt.imshow(box_edge, vmin = 0., vmax = np.percentile(box_edge,99))

```

```
Out[14]: <matplotlib.image.AxesImage at 0x7f219260d9e8>
```

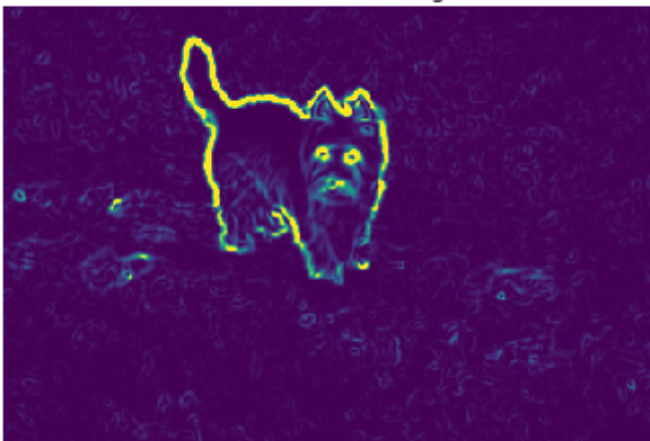
original figure



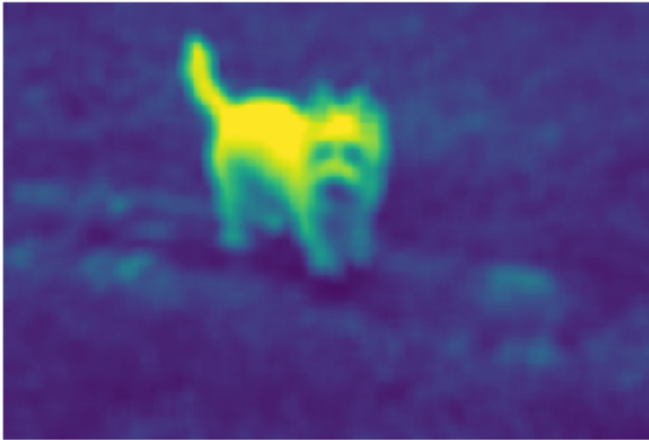
Gaussian blurred image



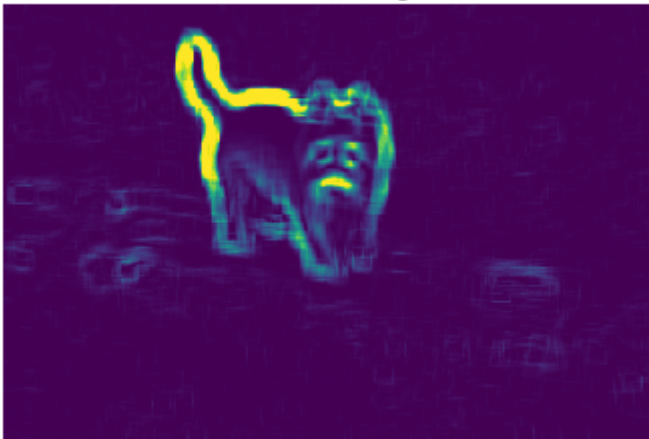
Gaussian blurred Edges



box filtered image



box filtered edges

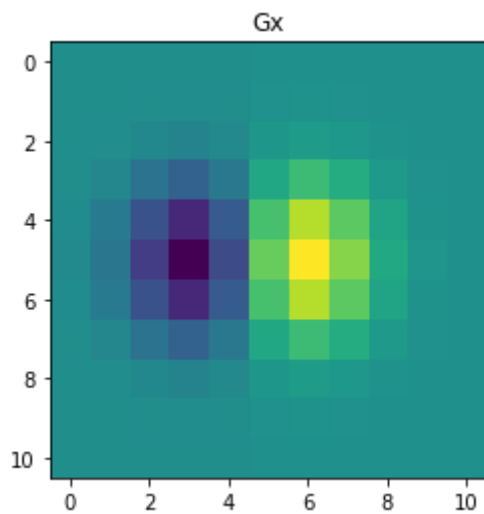


Problem 1.2: Pet edge detection (d)

Please use the visualization code to display the filter you construct.


```
In [16]: # Your code here!
def Gaussian_Ix_filter():
    dimx = 11
    dimy = 11
    sigma = 2
    gridx = np.linspace(-(dimx-1)/2, (dimx-1)/2, 11)
    gridy = np.linspace(-(dimy-1)/2, (dimy-1)/2, 11)
    [h1,h2] = np.meshgrid(gridx,gridy)
    hg = np.exp(-(h1**2+h2**2)/(2*sigma))
    dx = np.array([[-1,1]])
    return ndimage.convolve(hg,dx)
Gx = Gaussian_Ix_filter()
plt.figure()
plt.title('Gx')
plt.imshow(Gx)
```

Out[16]: <matplotlib.image.AxesImage at 0x7f21924773c8>



Problem 1.2 Pet edge detection (e)

Please visualize the results using matplotlib.

```

In [23]: im = imageio.imread('apple.jpg').mean(2)/255.

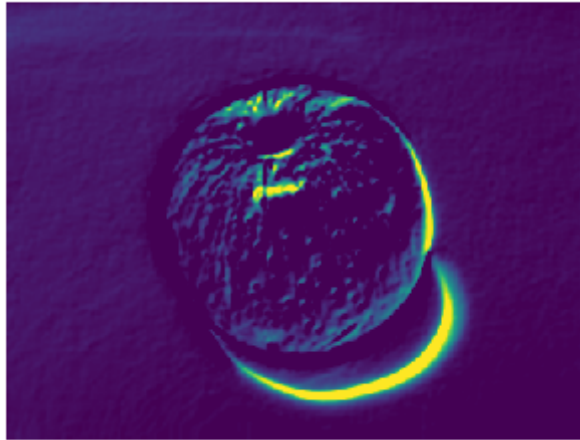
# Your code here!
def oriented_grad(Ix,Iy,theta):
    return Ix*np.cos(theta) + Iy*np.sin(theta)
theta_set = [1/4*np.pi,1/2*np.pi,3/4*np.pi]
def Gaussian_kernel(dimx,dimy,sigma):
    gridx = np.linspace(-(dimx-1)/2,(dimx-1)/2,11)
    gridy = np.linspace(-(dimy-1)/2,(dimy-1)/2,11)
    [h1,h2] = np.meshgrid(gridx,gridy)
    hg = np.exp(-(h1**2+h2**2)/(2*sigma))
    return hg
Gaussian_filter = Gaussian_kernel(11,11,2)
im_g = ndimage.convolve(im,Gaussian_filter)
Ix = ndimage.convolve(im_g,dx)
Iy = ndimage.convolve(im_g,dy)
for theta in theta_set:
    edges_g = oriented_grad(Ix,Iy,theta)
    plt.figure()
    plt.title('Gaussian blurred Edges with theta: {}'.format(theta))
    plt.axis('off')
    plt.imshow(edges_g, vmin = 0., vmax = np.percentile(edges_g, 99))

box_filter = np.ones([11,11])/11**2
im_b = ndimage.convolve(im,box_filter)
Ix = ndimage.convolve(im_b,dx)
Iy = ndimage.convolve(im_b,dy)

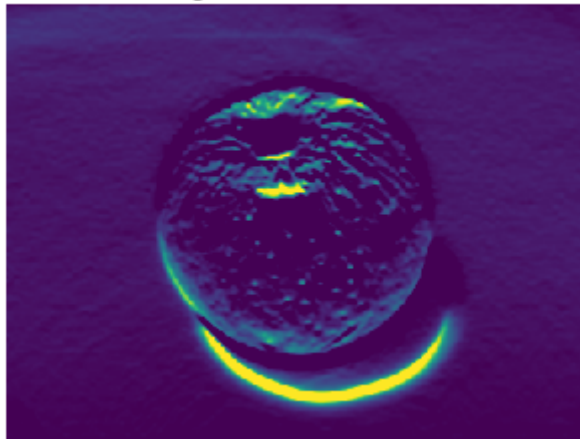
for theta in theta_set:
    box_edge = oriented_grad(Ix,Iy,theta)
    plt.figure()
    plt.title('box filtered edges with theta: {}'.format(theta))
    plt.axis('off')
    plt.imshow(box_edge, vmin = 0., vmax = np.percentile(box_edge,99))

```

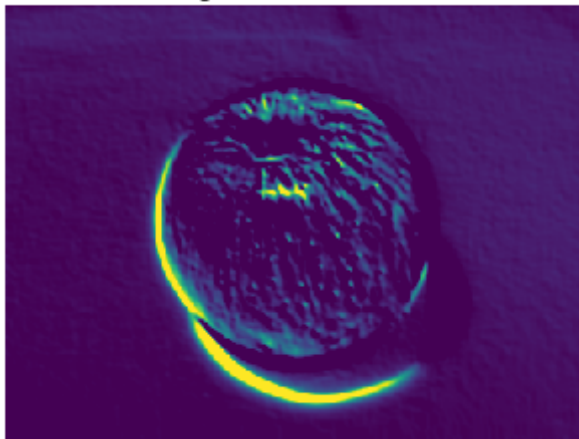
Gaussian blurred Edges with theta: 0.7853981633974483



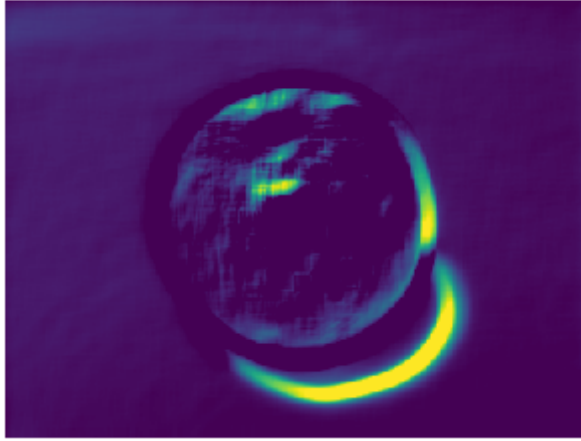
Gaussian blurred Edges with theta: 1.5707963267948966



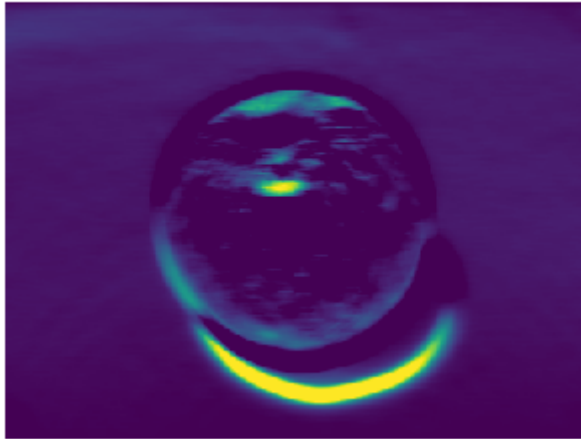
Gaussian blurred Edges with theta: 2.356194490192345



box filtered edges with theta: 0.7853981633974483



box filtered edges with theta: 1.5707963267948966



box filtered edges with theta: 2.356194490192345

