CS216 : Assignment 2

Reading: Review your class notes, Klette Chapter 1.2-1.3, Szeliski Chapter 2,3 and/or Forsyth and Ponce Chapters 1,4,8,9

For written problems, you can either use Markdown/LaTeX to directly enter your reponse in the provided notebook cell, or if you prefer, write out by hand and embed a photo of your solution. For coding problems, I have provided some suggestion of how to structure your code but feel free to modify or ignore as you see fit.

Please edit the cell below to include your name and student ID #

---

name:

SID:

In [1]:
```
%matplotlib notebook
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from numpy import fft
from scipy import ndimage
```

1. Convolution versus Correlation

Prove that *convolution* is associative, that is $(f \star g) \star h = f \star (g \star h)$

Prove that *correlation* is not associative (best done by showing a counter-example)

You should just show this for discrete 1D signals and assume the signal has been extended periodically as we did in lecture.

Type *Markdown* and LaTeX: α^2

2. Fast Fourier Transform

If we have an image I of dimension HxW and we convolve it with a filter f of size MxN using the spatial domain formula given in class, what will the complexity be (i.e. how many multiplications)? How about if we use the Fast Fourier Transform (https://en.wikipedia.org/wiki/Fast_Fourier_transform) "trick"?

Type *Markdown* and LaTeX: α^2

3. Factorizing Filters

Starting with the formula for 2D convolution, show that if our filter is the product of two functions, $f(x, y) = f_1(x)f_2(y)$, we can compute the convolution more efficiently. Is there a way to use this idea to speed up convlution with a 2D isotropic Gaussian filter $g(x, y) = \frac{1}{2\pi\sigma^2}\exp^{\frac{-(x^2+y^2)}{2\sigma^2}}$?

Type *Markdown* and LaTeX: α^2

4. Experimenting with the DFT

Import the appropriate functions from the numpy fft module (https://numpy.org/doc/stable/reference/routines.fft.html) and carry out some experiments yourself. You will want to use the ***fftshift*** function to get the spectra like we showed in class where 0 is at the center

4.1 Start out in 1D and make a signal which is of length 100 with a single impulse of height 1 in the middle. Compute the DFT of the signal and plot the magnitude of the spectrum.

```
In [ ]:
```

4.2 Now increase the width of the pulse from a single sample to a unit height ``box function'' 5 and 10 samples long and plot resulting the spectrum magnitudes.

```
In [ ]:
```

4.3 Now do the same for a Gaussian function with $\sigma = 1$ and $\sigma = 2$ (the Gaussian has infinite support but you should just analyze a finite chunk centered around the origin).

```
In [ ]:
```

4.4 Figure out how to do a 2D DFT and inverse DFT. Replicate the experiment we showed in class (shown as Figure 7.6 in Forsyth and Ponce) of swapping phase and magnitude by (1) computing the DFT of two images of the same size, (2) computing a new set of coefficents which have the phase from the first image and the mangitude from the second image, (3) taking the inverse DFT of this combined spectrum to produce a new image.

```
In [ ]:
```

5. Edge Detection

Write a gradient based edge detector. Your code should load in an image and convert to grayscale floats. Once you have loaded in the image, you should smooth the image with a Gaussian filter and then compute horizontal and vertical derivatives using the derivative filter described in lecture. The amount of smoothing is determined by the σ of the Gaussian (which should be a parameter of your code). You can use **scipy.ndimage.convolve2d** option to perform the required convolutions. Once you have computed the derivatives in the x and y directions using convolution, compute the gradient magnitude and angle and threshold the magnitude to find the edge pixels. To do non-maximum suppression, you can use **skimage.morphology.thin** in order to get out a 1-pixel-wide version of the detected edges.

5.1 Implementation

```
In [ ]: #
        # suggestion on how to structure your code... feel free to
        # modify or ignore as you see fit
        #

        def image_gradient(I,sigma):
            """
            Compute the gradient orientation and magnitude after
            smoothing I with a Gaussian filter with parameter sigma
            """




        def detect_edge(I,sigma,thresh):
            """
            Detect edges in an image using the given smoothing and threshold
            parameters.
            """


            .
            .
            .
```

5.2 Demonstration

For two images (the synthetic example below and an additional image of your choice), visualize the input image, gradient magnitude, angle and the edge detection output for two different settings of σ. This will yield 16 images in total. You should make sure that the image is low enough resolution and the plots large enough to clearly see the results. For visualizing the orientation, you should use a circular colormap such as "twilight" or "hsv". You should include colorbars on your plots where appropriate so the range of values is visible.

In [ ]:
```
# set the plot figure size
plt.rcParams['figure.figsize'] = (12,12)

# synthetic test image for debugging purposes
[yy,xx] = np.mgrid[-100:100,-100:100]
image = np.minimum(np.maximum(np.array(xx*xx+yy*yy,dtype=float),400),8100)

.
.
.
.
```

6. Template Detection

Implement a simple object detector based on cross-correlation. Load in one of the test images provided and display it on the screen. Clip out a patch of the image containing an object you want to detect. Use this patch as a template in order to try and detect other instances of the object in the image.

You should implement three different variants for computing a match score S between the template T and the image I.

1. Cross-correlation (CC):

   $S(x,y) = \sum_{i,j} T[i,j] I[x+i,y+j]$

2. Normalized Cross-correlation (NCC):

   $S(x,y) = \sum_{i,j} \frac{(T[i,j]-\mu_T)}{\sigma_T} \frac{(I[x+i,y+j]-\mu_{xy})}{\sigma_{xy}}$

   $\mu_{xy} = \frac{1}{N} \sum_{i,j} I[x+i,y+j]$

   $\sigma_{xy} = \sqrt{\frac{1}{N} \sum_{i,j} (I[x+i,y+j]-\mu_{xy})^2}$

3. Sum-of-squared-differences (SSD):

   $S(x,y) = -\sum_{i,j} (T[i,j] - I[x+i,y+j])^2$

Note, I've added a minus sign in front of SSD to make it consistent with the other two so that a larger score S corresponds to a better match.

You can implement all of these quite effeciently using a couple convolutions and vectorized NumPy operations. If you use convolution to perform cross-correlation, don't forget to flip the template left-right and top-bottom. To compute averages, you can use convolution with a filter of all 1s the same size as your template.

You can try thresholding the resulting score map S but you will note that around detected objects there will be many pixels with high values. In order to suppress non-maximal responses, zero out any locations which are smaller than one of their 8 neighbors. You can do this effeciently in NumPy using array indexing, e.g. in 1D this might look like:

```
L = x[1:-1] > x[0:-2]   #bigger than our neighbor to the left?
R = x[1:-1] > x[2:]     #bigger than our neighbor to the right?
T = x[1:-1] > threshold #above detection threshold?
maxima = R & L & T
```

It is also probably a good idea to set an upper limit on the number of detections your function returns (e.g., so if you happen to call it with too low a threshold things don't get overly slow).

6.1 Implementation

In [ ]:
```
#
# suggestion on how to structure your code... feel free to
# ignore or modify as you see fit
#

def detect_template(I,template,thresh,method):
    """
    Detect a object in an image by performing cross-correlation with the
    supplied template

    image : 2D float array of shape HxW
         An array containing pixel brightness values

    template : 2D float array
         Template we wish to match.

    thresh : float
        Score threshold above which we declare a positive match

    method : one of {'cc','ncc','ssd'}
        Method for comparing template and image patch

    Returns
    -------
    detections : a list of tuples of length ndetect
        Each detection is a tuple (x,y,score)

    heatmap : detection score map prior to non-max suppression

    """



    #
    # some example code for finding the top scoring locations
    # stored in some 2D array
    #
    # sort the values in resp in ascending order.
    # val[i] should be ith largest score in resp
    # ind[i] should be the index at which it occurred so that val[i]==resp[ind[i]]
    #
    val = np.sort(resp, axis=None)[::-1] # sorted response values
    ind = np.argsort(resp, axis=None)[::-1]  #corresponding indices

    # recover the x,y coordinates of the ith largest response
    [y,x] = np.unravel_index(ind[i], resp.shape)
```

6.2 Visualization and Discussion

Initially experiment with choosing a relatively small "object" such as Dilbert's nose or a single letter as your template. You can use the provided images but you should also experiment with some image of your own choosing (ideally which contains more than one instance of the object of interest).

Plot the locations of above threshold local maxima on top of the original image using **patches.Rectangle**. Experiment with the threshold to try to find a good tradeoff between detecting all the instances of an object and but not too many background detections. Show a visualization of the template you used, the cross-correlation output using a false colormap (e.g. "jet"), and your final detection results.

Compare outputs of the three different methods (cc,ncc,ssd). You'll need to choose a different threshold for each one. Discuss briefly which one appears to work the best and come up with an explanation as to why you think this is the case.

In [ ]:
```
# visualize some results

#
# example code for visualizing detections
#
plt.imshow(image,cmap=plt.cm.gray)
ax = plt.gca()
w = tsize_pix[1] #size of the template
h = tsize_pix[0]
ct = 0
for (x,y,score) in detections:
    xc = x-(w//2)
    yc = y-(h//2)
    rect = patches.Rectangle((xc,yc),w,h,linewidth=2,edgecolor=np.array([1,0,0]),facecolor='none')
    ax.add_patch(rect)
    ct = ct + 1
plt.show()

.
.
.
.
```

your discussion of results goes here

In [ ]: