Mandatory Assignement 1
IN5520

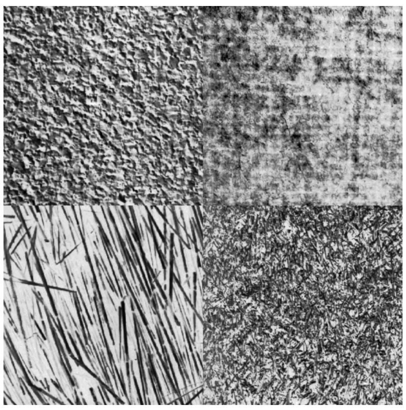
Wonho Lee
wonhol@ifi.uio.no

September 2018

1 Introduction

This assignment is to distinguish different textures in an image with gray level co-occurrence matrix method with various feature functions. The method can be parameterized and therefore give possibility of fine feature engineering to extract and distinguish between different textures. Two example mosaic image is given, each containing four different but featuring some similarity.

2 Texture Analysis


(a) Mosiac 1

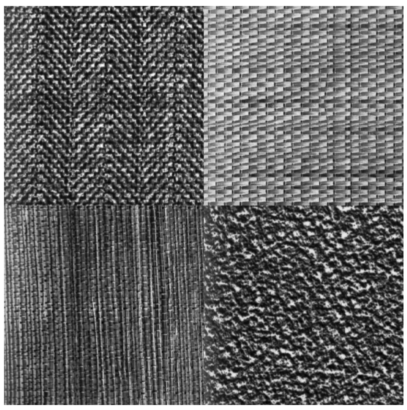

(b) Mosiac 2

Figure 1

* Order of numbering will be from left to right starting on upper left of each mosaics.

2.1 Mosaic 1

1. Direction is not clear but it slightly shows 45 degrees due to strong light source from upper right side. Large texture elements with high contrast, it indicates relatively high variance and low homogeneity for a larger region containing many of its elements.
2. This texture is the smoothest, it will result in low variance but some clustering of particles will give some of homogeneity. A grid like pattern is observed.
3. Almost threshold image like contrast will give highest variance, also these long elements are arranged to point at certain direction.

4. Uniform direction, relatively high variance, low homogeneity is expected. It is similar to the first texture besides size of elements

2.2 Mosaic 2

Strong structures are observed in texture 1,2 and 3. Direction of pixel pairs in GLCM will have great influence in this mosaic.

1. Clear structure is observed in $\pm 45^o$. High contrast with high variance.
2. Strong structure in both x and y direction.
3. Prominent direction along the y direction.
4. Uniform direction, with similar level of contrast and size of elements to the first texture.

3 Visualizing GLCM matrices

3.1 Gray Level Co-occurrence Matrix

In gray level co-occurrence matrix, number of pixel pairs are counted up for given distance between pixels, d and θ, angle between them. and set into a matrix to corresponding index i and j, where they represent gray level of each pixel element. Analysis of GLCM can reveal patterns, structures based on its given parameters, d, θ, size and gray level of image.

GLCM is implemented in two nested for-loops, going through all the pixel pair for its parameters, incrementing corresponding index for its pixel pair. Then it is normalized by number of pixel pairs. Symmetrical GLCM counts for pixel pair (i, j) and (j, i). In computation, it is achieved by calculating a simple GLCM then adding its transposed pair.

Rather than taking continuous value for d at given θ, discrete values, $\{-45°, 0°, 45°, 90°\}$ are chosen. Then for given θ, a pair of Δx and Δy is set such that for 45°, $(\Delta x, \Delta y) = (d, d)$, for 0°, $(\Delta x, \Delta y) = (d, 0)$ and vice versa.

```
img : given image at the size of M x N
w : number of pixels
glcm : zero matrix size of G x G, where G is gray level

for i in M-dx
    for j in N-dy
        glcm(img(i,j), img(i+dx, j+dy)) += 1

glcm = glcm / w
glcm = glcm + glcm.transpose
```

3.2 Parameter selection

Based on the observation on the characteristic of each textures, few set of parameters are tested. Size of texture elements and contrast will take direct effect on different d values, direction of texture will affect choice of θ.

As the goal of the assignment is to segment different textures, parameters should be set so that they reveal fine differences among each pattern.

For Mosaic 1, there are clear distinction between 2 and 3 for its contrast but 1 and 4 is quite similar besides the size of texture elements and direction. In this regard, a proper θ value should be $\pm 45°$. To examine the difference in size of elements between the texture 1 and 4, d is tested for 1,3, and 5. Testing revealed that most distinguishable GLCM matrices are observed for $d = 3$.

For Mosaic 2, the first texture has directions in both $\pm 45°$. This will give opposite GLCM pattern in different region. It is better to choose 0° or 90° which will reveal clear difference between 1,2,and 3. However, it might be wise to compute isotropic GLCM to distinguish texture 1 and 4, as their similarity in element size and contrast. With this observation, GLCM feature is computed for $d = 3$, and $\theta = 90°$ and isotropic GLCM, averaging four GLCM in four each discrete directions.

3.3 Visualisation of GLCM on a Single Patch of each Texture

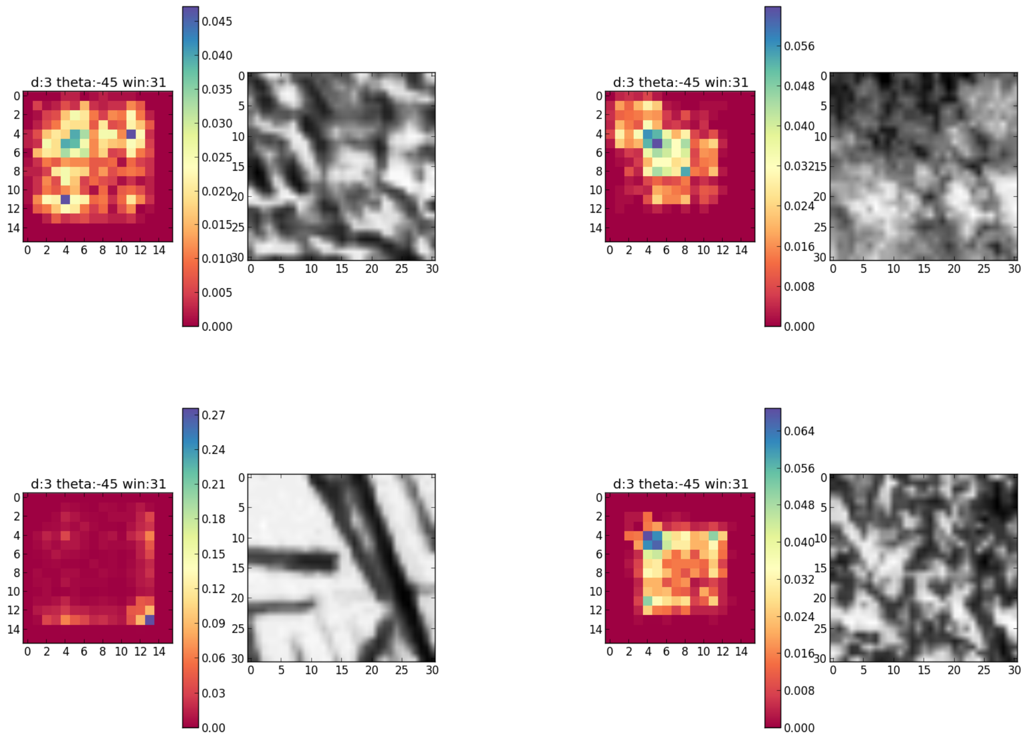


Figure 2: $d = 3$, $\theta = -45$ for Mosaic 1

Figure 2 shows results of a small patch of each texture GLCM calculated from $d = 3$ and $\theta = -45$. From these specific pathces, the resulting GLCM seems to separate them quite well in terms of distributed area and peak values. However as texture 3 has different distribution of texture elements over the image, it is expected that resulting GLCM will vary when they are calculated for all the pixels.

Figure 3 is resulting GLCM from $d = 3$ and $\theta = 90$. Choice of the angle seem to separate the texture 2 and 3 as their prominent directions are orthogonal. However texture 1 and 4 seems to have similar peak and region for the given parameters.

Inverse difference moment has $(i - j)$ expression in the denominator. This indicates that those

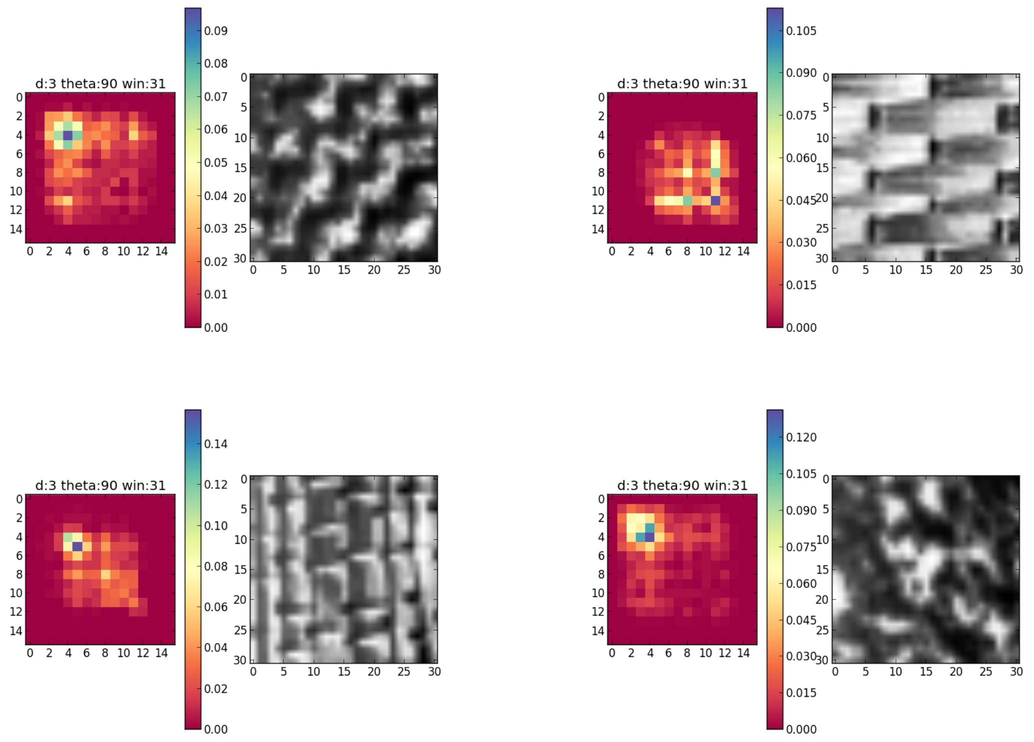


Figure 3: $d = 3$, $\theta = 90$ for Mosaic 2

GLCM which are spread across the matrix will score higher in results, which will successfully separate for example texture 1 and 2 from Mosaic 1. Inertia on the other hand has the same expression as positive weight, it will give an opposite result.

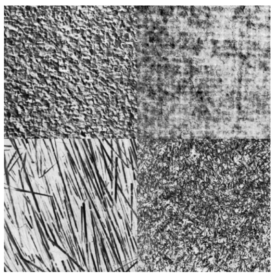
4 Computing GLCM feature images in local windows

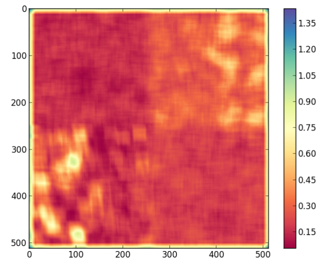
To calculate feature images, original image was padded with zeros around the edges in order to run sliding window from the very first pixel. The GLCM calculated from the local window is then used in feature functions to give a single value for the pixel located in the center of the window.

Following figures 4 and 5 shows the resulting feature map from three functions. Just as the effect of window size on sliding window filtering, the larger window results in smoother image. A moderate size of window should give flat region and clean enough borders for the desired textures to detect. Window size 31 is chosen on the feature maps, which gives moderate edges, and homogeneous area for most of textures.

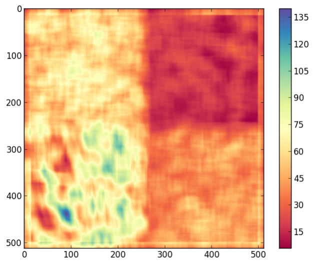
The resulting feature map from Mosaic 1 seems to reveal the differences in the human eyes at a glance. The texture 1 and 4 seems to have similar results as expected, but it still has fine differences in the tone. With fine tuning a range for global threshold will give mask to separate the textures. The texture 3 has most varying feature map. This will make it difficult to extract the texture with global threshold. By increasing window size or decreasing d, it should be able to get smooth feature map.

Isotropic GLCM is computed for the Mosaic 2 with $d = 3$. The goal of isotropic GLCM was to

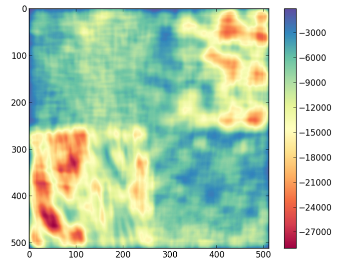

(a) Original image



(b) Inverse difference moment

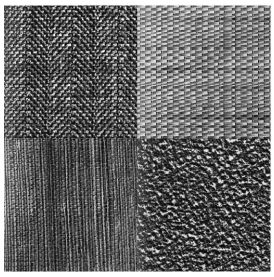


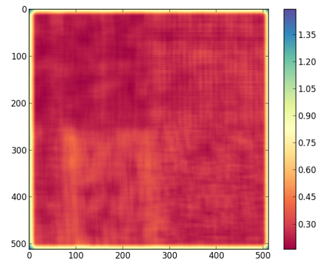
(c) Inertia



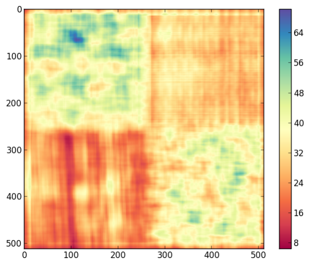
(d) Cluster shade

Figure 4: Feature map for $d = 3$, $\theta = 90$

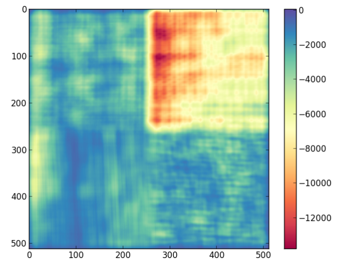

(a) Original image



(b) Inverse difference moment



(c) Inertia



(d) Cluster shade

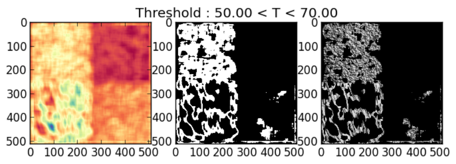
Figure 5: Feature map for $d = 3$, Isotropic GLCM

achieve clear distinction between texture 1 and 4.

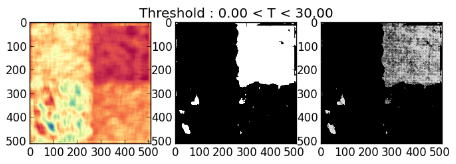
As expected, texture 2 and 3 are easily distinguishable from inertia and cluster shade feature map. However, results between texture 1 and 4 is still not very clear besides minor tone difference shown on the result of inverse difference moment.

5 Segment the GLCM feature images and describe how they separate the textures

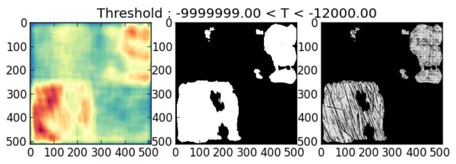
Global threshold values are manually tested to create mask for desired textures. Thresholding was done by setting a range of value on a feature map.



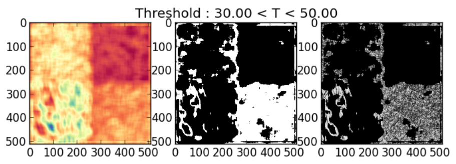
(a) Inertia, $d = 3$, $\theta = -45$



(b) Inertia, $d = 3$, $\theta = -45$



(c) Cluster shade, $d = 1$, $\theta = -45$,$win = 51$

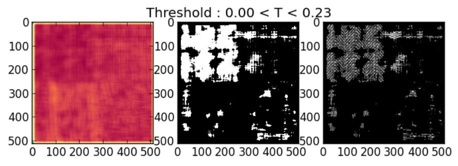


(d) Inertia, $d = 3$, $\theta = -45$

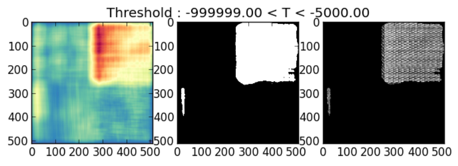
Figure 6: Global thresholding results for Mosiac 1

Figure 6 shows the result of global threshold applied on Mosaic 1. As seen on the figure, texture 1,2 and 4 are easily separated from the feature map of inertia with given parameter, $d = 3$, $\theta = -45°$, and window size of 31. Texture 2 is most clearly separated due to its low contrast. Selection of θ seems to separated texture 1 and 4 as previously discussed on its texture element direction. Despite the most significantly different characteristic of texture 3, it was most difficult to clearly separate as distribution of texture elements were uneven. To overcome this issue, cluster shade feature map was calculated for $d = 1$ and window size 51 to smooth out the feature map. It was capable to detect most area of texture 3, but it also included large area from texture 2 with this approach.

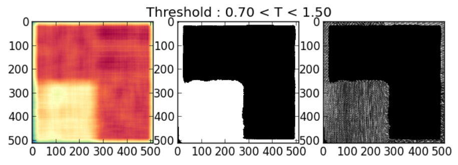
Figure 7 is result of global threshold on Mosaic 2. Texture 2 and 3 are extracted using parameter $d = 3$, $\theta = 90°$, with window size of 51. Larger window size gave clear and smooth feature map for IDM and cluster shade to seprate the textures. The result of isotropic GLCM computation resulted very subtle difference for texture 1 and 4 on IDM feature map. However it was difficult to distinguish between texture 4 and the rest, 2 and 3. IDM feature map witn $\theta = 0°$ gave relatively distinguishable result for texture 4.



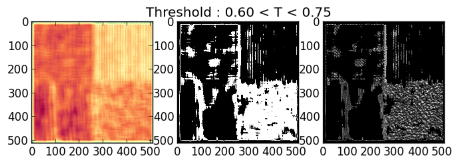
(a) IDM, $d = 3$, isotropic



(b) Cluster shade, $d = 3$, $\theta = 90$, $win = 51$



(c) IDM, $d = 3$, $\theta = 90$, $win = 51$



(d) IDM, $d = 3$, $\theta = 0$

Figure 7: Global thresholding results for Mosiac 2

6 Conclusion

The GLCM texture analysis method discussed in this assignment has shown itself to be effective to distinguish most of textures given in the example mosaic images. However as nature of the method suggests, it poses some limitations. First, it requires fine adjustment of parameters based on observation. It means that for different set of mosaics or different type of texture, whole new parameter set should be tested to separate the textures. Also as given in parameter d and θ, this method is not invariant to size, rotation of textures. Rotating and zooming in/out a texture will result in failure for separating texture with global threshold with predefined values. Isotropic GLCM may resolve this issue with some of textures with strong direction tendencies, unless rotation is done in outside of the discrete θ values.

However, for an well-defined problem with target images obtained from controlled environment, which often is case in industry, this method will provide a reasonable solution considering computation cost and amount of data available for the problem.

Attachments

Code is executed by

```
$ python img_process2.py
```

```
import matplotlib
matplotlib.use('agg')
import sys
import numpy as np
from matplotlib import pyplot as plt
from numba import jit

@jit
def part_texture(img, mat=(2,2)):
    tex = []
    size = img.shape[0]//2
    for i in range(mat[0]):
        for j in range(mat[1]):
            tex.append(img[i*size:i*size+size, j*size:j*size+size])
    return tex

@jit
def quantize(img, lvl=16):
    quantized = img[:]
    quantized = (quantized*(lvl-1)).astype(np.int8)
    return quantized

# zero padding
@jit
def padding(img, winSize=(31,31)):
    padded = np.zeros([(img.shape[0]+winSize[0]-1),
                       (img.shape[1]+winSize[1]-1)],np.uint8)
    padded[winSize[0]//2:winSize[0]//2+img.shape[0],
           winSize[1]//2:winSize[1]//2+img.shape[1]] = img
    return padded

@jit
def slidingWindow(img,d,theta,iso,featFunc,winSize=(31,31)):
    # padding image
    padded = padding(img, winSize)
    res = np.zeros(img.shape)

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            win = padded[i:i+winSize[0],j:j+winSize[1]]
            if iso:
                coMat = isoGLCM(win,d)
            else:
                coMat = GLCM(win,d,theta)
            res[i,j] = featFunc(coMat)
    return res

@jit
def GLCM(win, d, theta='0', grayscale=16):
    assert theta in ['-45','0','45','90'], 'unsupported theta'
    coMat = np.zeros([grayscale,grayscale],np.uint16)

    if theta == '0':
        dx = 1*d
        dy = 0
    elif theta == '45':
        dx = 1*d
        dy = 1*d
    elif theta == '90':
        dx = 0
        dy = 1*d
    else:
        dx = 1*d
        dy = 1*d
        # mirror
        win = np.flip(win[:], axis=1)

    # find pairs with theta = 0 and d = 1
    for i in range(win.shape[0]-dy):
        for j in range(win.shape[1]-dx):
            coMat[win[i][j]][win[i+dy][j+dx]] += 1

    # normalize by number of pixel pair
    w = float((win.shape[0]-dx)*(win.shape[1]-dy))

    # symmetrical glcm
    res = coMat/w + np.transpose(coMat)/w

    return res
@jit
def isoGLCM(win,d,grayscale=16):
    theta = ['-45','0','45','90']
    coMat = np.zeros([grayscale,grayscale],np.float)
    for t in theta:
        coMat += GLCM(win,d,t)
    return coMat/4
@jit
def IDM(coMat):
    res = 0
    for i in range(coMat.shape[0]):
        for j in range(coMat.shape[1]):
            res += 1/(1+(i-j)**2) * coMat[i,j]
    return res
@jit
def inertia(coMat):
    res = 0
    for i in range(coMat.shape[0]):
        for j in range(coMat.shape[1]):
            res += (i-j)**2 * coMat[i,j]
    return res
@jit
def shade(coMat):
    res = 0
    # ux and uy same for symmetrical GLCM
    u = ux(coMat)

    for i in range(coMat.shape[0]):
        for j in range(coMat.shape[1]):
            res += (i + j - 2*u)**3 * coMat[i,j]
    return res
@jit
def ux(img):
    res = 0
    for i in range(img.shape[0]):
        res += i*np.sum(img[i])
    return res

def uy(img):
    img = np.transpose(img[:])
    res = 0
    for i in range(img.shape[0]):
        res += i*np.sum(img[i])
    return res

# Visualize GLCM for a texture patch size of given window size
def visualizeGLCM(filename, d,theta,win,iso=False):
    img = plt.imread(filename)
    q = quantize(img)
    tex = part_texture(img)

    for i in range(4):
        q = quantize(tex[i])
        if iso:
            g = GLCM(q[:win,:win],d,theta)
        else:
            g = isoGLCM(q[:win,:win],d)


        print(i,'IDM : %.02f \t Inertia : %.02f, \t Shade : %.02f'%(IDM(g),inertia(g),
        plt.subplot(1,2,1)
        plt.imshow(g,interpolation='none',cmap=plt.get_cmap('Spectral'))
        plt.title('d:%d theta:%s win:%d'%(d,theta,win))
        plt.colorbar()
        plt.subplot(1,2,2)
        plt.imshow(tex[i][:win,:win],cmap=plt.get_cmap('gray'))
        plt.savefig('../%s%d_d_%d_th_%s_win_%d.png'%(filename,i+1,d,theta,win))
        plt.clf()

# take a numpy file which contains feature map
#   apply global threshold of given range <lower,upper>
#   create mask and outputs image to a file
def globalThreshold(featureFile, imgFile, lower, upper, fromFile=False):
    if fromFile:
        featureMap = np.load(featureFile)
    else:
        featureMap = featureFile
    img = plt.imread(imgFile)
    mask = (featureMap < float(upper))*(featureMap > float(lower))
    plt.subplot(131)
    plt.imshow(featureMap, cmap=plt.get_cmap('Spectral'))
    plt.subplot(132)
    plt.imshow(mask,cmap=plt.get_cmap('gray'))
    title = 'Threshold : %.02f < T < %.02f'%(float(lower),float(upper))
    plt.title(title)
    plt.subplot(133)
    plt.imshow(img*mask,cmap=plt.get_cmap('gray'))
    plt.show()
    plt.savefig('%s%s.png'%(imgFile[:-4],title))

# create feature map for given parameter
# outputs feature map image and numpy file containing the feature map
def createFeatureMap(img, d, theta, win, func, iso):
    filename = 'p%s_%s_d_%d_th_%s_win_%d_iso_%d'%(img[:-4],func,d,theta,win,iso)

    print 'Saving to ', filename

    img = plt.imread(img)
    q = quantize(img)

    featureMap = slidingWindow(q,d,theta,iso,eval(func),(win,win))
    plt.imshow(featureMap, cmap=plt.get_cmap('Spectral'))
    plt.colorbar()
    plt.savefig('%s.png'%filename)
    np.save('%s.npy'%filename, featureMap)

    return featureMap

if __name__ == '__main__':
    if len(sys.argv) < 8:
        print 'Argument\n\t img d theta win featFunc iso lower upper\n'

        print 'Running test example'
        print 'For Mosaic 1:'
        print '\td = 3'
        print '\ttheta = -45'
        print '\twin = 31'
        print '\tfeature function : Inertia'
        print 'Create global threshold mask for range'
        print '\t [30,50]'

        img = 'mosaic1.png'
        d = 3
        theta = '-45'
        win = 31
        func = 'inertia'
        iso = 0
        lower = 30
        upper = 50
```

```
else:

    img = sys.argv[1]
    d = int(sys.argv[2])
    theta = sys.argv[3]
    win = int(sys.argv[4])
    func = sys.argv[5]
    iso = int(sys.argv[6])
    lower = float(sys.argv[7])
    upper = float(sys.argv[8])

featureMap = createFeatureMap(img, d, theta, win, func, iso)
globalThreshold(featureMap, img, lower, upper)
```