assignment 1 - Segmentation of textured regions in an image

Link to the assignment

Introduction

This assignment will use GLCM to try and segment different types of textures. Part A is comprised of visual observations of each of the different textures in ‘mosaic1.png’ and ‘mosaic2.png’. From this visual observations i will try and define parameters to use in part B.

In part B i manually extract each texture from both images. Then i calculate the GLCM for this whole image, not sliding window. The parameters should reflect the observations made in part A. Then the GLCM is plotted with a heatmap. The goal is to have different GLCM for each of the textures.

Moreover, in part C i will try and extract features from the whole image by using a GLCM with a sliding window given a fixed window size. The features are extracted from the GLCM by applying different weights. These are GLCM homogeneity, inertia and cluster shade. These feature maps will then be plotted.

The last part, part D, is segmenting the feature maps created in part C by using a global threshold. The goal is to try and segment the different textures. The threshold value will be different for each of the three GLCM feature maps.

The code for calculating the GLCM, extracting features, threshold and sliding window is taken from the solution to the texture exercises. Source code can be found following this link. The rest of the code is made by me.

The Gray-level Co-Occurrence matrix (GLCM) is basically an estimate of the second order joint probability going from a gray level i to gray level j given a distance d and direction θ. The correlation between pixel intensities in a given direction.

When using GLCM one wants to have a sufficient “average occupancy level” in the image. To achieve this, one could increase the window size or reduce number of gray levels. If the textures have low contrast reducing gray levels could lead to less precise texture description. Also be careful when histogram-equalizing the image due to the risk of changing relationships between pixels.

A. Analyzing the texture

Keywords for the analysis:

- texture direction, frequency, variance, homogeneity, texture element size, texels (texture elements)

Find **what characterizes each texture** and **how the textures differ** for all textures

mosaic1.png

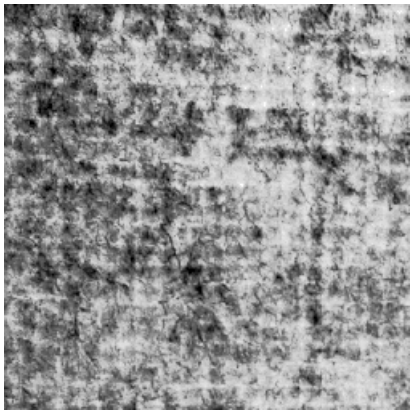
Texture 1 (top left):

Texture 2 (top right):

In this image there is two different textures, first there is a mesh-like texture, with a direction in north, west, east and south. It is repeated, meaning it has a given frequency and the texels are small in size.

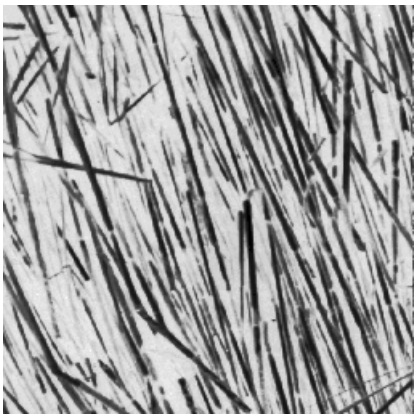
The second texel consists dark spots, varying in size, pixel intensity and position. I cannot determine what direction this texel has. If we define homogeneity as local homogeneity (same pixel intensities in a neighborhood), there are little homogeneity in this image.

Because of the mesh-like texture this does not resemble any of the other three textures



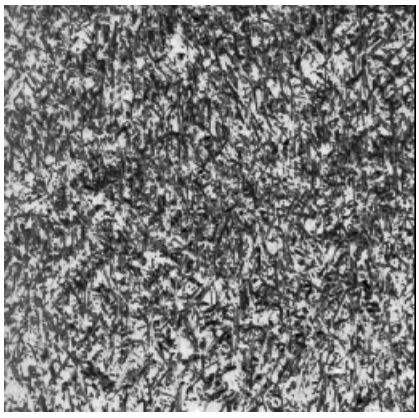
Texture 3 (bottom left):

There is two types of texels in this image. Both have a *north-west* or *south-east* direction. There are areas with some homogeneity, if we use the definition mentioned above. The dark texels are long and thin while the white texels are a bit wider. There is no fixed frequency. These texels are not similar to the other textures at all.



Texture 4 (bottom right):

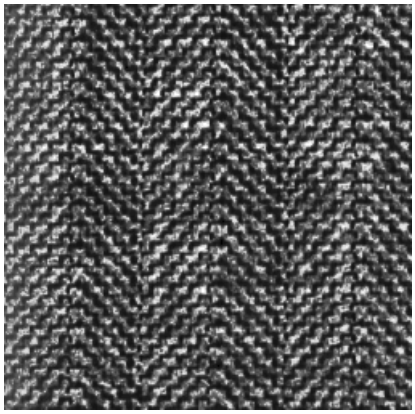
This texture has no clear direction. The texels are very small and varies both in size, position and frequency. It does resemble the texture 1 in some cases, but this is even more noisy. From a global perspective it is homogeneous, but local (pixel neighborhood) it is in-homogeneous.



mosiac2.png

Texture 5 (top left):

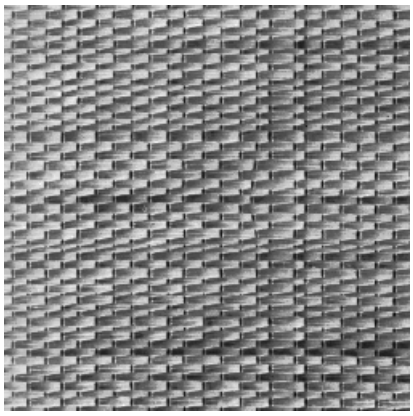
The direction of this texture varies going from left to right. First it starts in a north-*east* then changes to south-*east.* This pattern is repeated about six times, from left to right. The texels are thin short stubs, where the pixel intensity varies. It could look like this is a snippet of a rug. The texels are not homogeneous.



Texture 6 (top right):

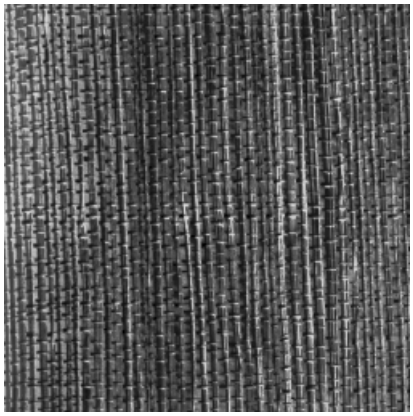
The texels are going in a *east* direction. They are small and rectangular. They are repeated at a fixed frequency and the pixel intensity of each texel are quite homogeneous. Though there are some variance in the intensity throughout the image. Each of the rectangular texels are homogeneous.

In addition to these rectangular texels there are a even smaller texel, going in the north or south direction. These are also repeated at at given frequency.



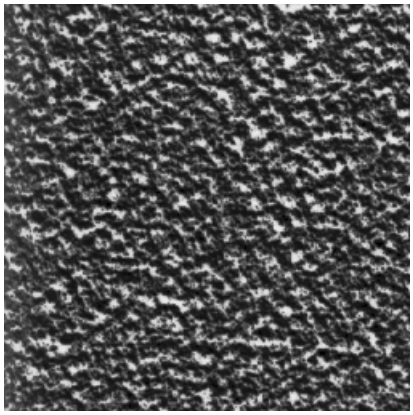
Texture 7 (bottom left):

Tese texels are long and thin with a direction going south or north. The texels are repeated with a fixed frequency and varies in pixel intensities. The texture is homogeneous, because several areas with about the same pixel intensity.



Texture 8 (bottom right):

This texture resembles texture 1 and 4. The texels vary in pixel intensity and have no direction. There are areas with homogeneity. There is no pattern repeated. The texels also vary in size and positon.



B. Visualizing GLCM matrices

Eventual histogram transforms must be discussed:

I've chosen to not do a histogram transform due to finding the re-quantized images sufficient. I did a gray scale transform from 256 gray levels to 16.

GLCM parameters:

Based on the visual observations i have chosen the following list of parameters. All the directions are chosen to reflect the direction of the texture.

Texture 1:

- dx = 1, dy = 1, gray_levels = 16
- These parameters are based upon the texture moving in a *south-east* direction. Also the pixel intensites varies on small distances, therefore the distance is equal 1.

Texture 2:

- distance = 4, isotropic GLCM, gray_levels= 16
- I've chosen to calculate the isotropic GLCM for this image. This is based on the mesh-texture having directions in north, east, south, west.

Texture 3:

- dx = 2, dy = 6, gray_levels = 16
- This texture has long thin homogeneous lines in the diagonal direction, but more towards y-direction than x-direction. Thats why ive scaled dy abit higher.

Texture 4:

- distance 1, isotropic GLCM, gray_levels = 16
- This texture has no clear direction, therefore i've chosen to calculate the isotropic GLCM. The distance is 1, since the pixels change often over short distances.

Texture 5:

- dx = 3, dy = -3, gray_levels = 16

- This texture is alternating between the direction *north-east* and *south-east.* I chose to use the parametere that goes north-east.

Texture 6:

- dx = 2, dy = 0, gray_levels = 16
- The texture is manly going in a east-west direction. Therefore pixel correlation is only evaluated in this direction. In addition the distance is 2, since the pixel intensities changes often.

Texture 7:

- dx = 0, dy = 1, gray_levels = 16
- This texture has a *north-south* direction. Therefore the correlation of this direction is calculated.

Texture 8:

- distance = 1, isotropic GLCM, gray_levels=16
- This texture has no clear direction, therefore i've chosen to calculate the isotropic GLCM. since the pixel intensities varies often, the distance is set to 1.

How the isotropic GLCM is calculated:

I calculate the isotropic GLCM according to the lecture notes:

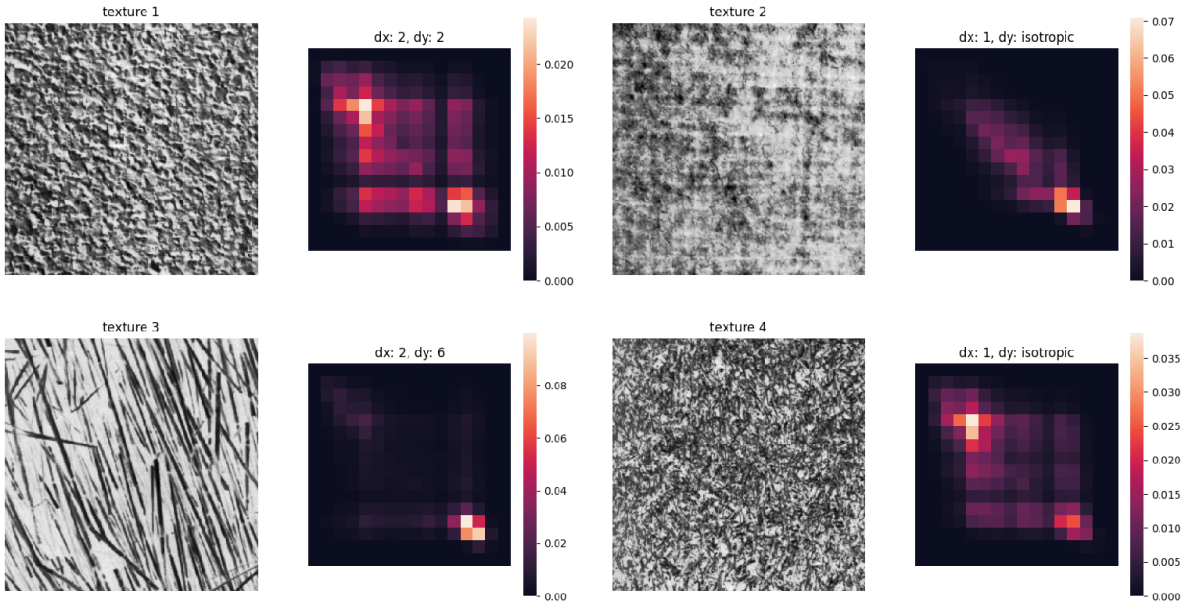
- Isotropic cooccurrence matrix by averaging
  $P(\theta)$, $\theta \in \{0^o, 45^o, 90^o, 135^o\}$

This results in the following code snippet:

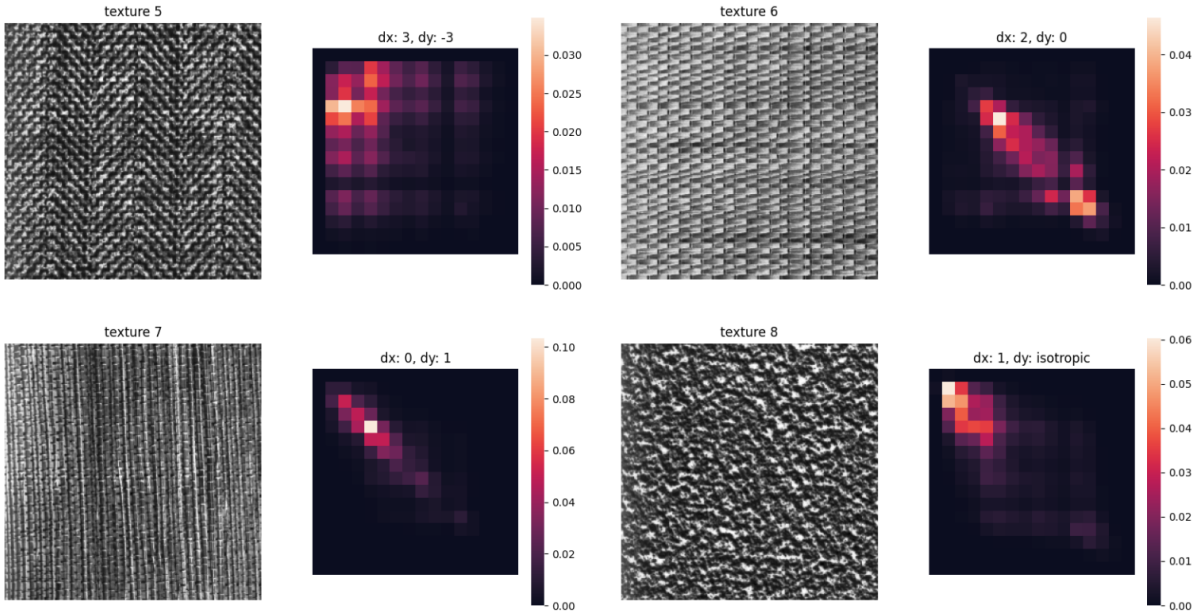
```
def isotropic_glcm(image, gray_levels, distance=1):
    quant_image = requantize(image, gray_levels=gray_levels)
    glcm_0 = glcm(quant_image, dx=distance, dy=0, gray_levels=gray_levels, normalise=True)
    glcm_45 = glcm(quant_image, dx=distance, dy=distance, gray_levels=gray_levels, normalise=True)
    glcm_90 = glcm(quant_image, dx=0, dy=distance, gray_levels=gray_levels, normalise=True)
    glcm_135 = glcm(quant_image, dx=-distance, dy=distance, gray_levels=gray_levels, normalise=True)

    isotropic = (glcm_0+glcm_45+glcm_90+glcm_135) / 4

    return isotropic
```

Include the GLCM matrices:

Mosaic1:



Mosaic2:



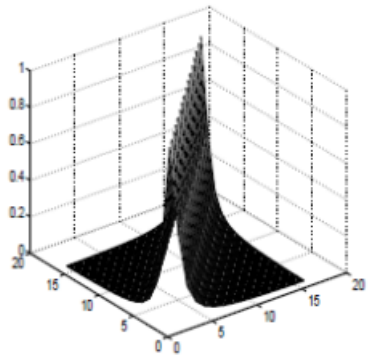
From the GLCM matrices, discuss similarities and differences between the image textures

Looking at the different GLCM matrices for *'mosaic1.png';* the GLCM's for texture 1 og 4 are very similar. This is most likely due to the textures being very similar. While they do not resemble the GLCM's for texture 2 and 3. GLCM for texture 2 and 3 are however also similar. Because we are looking at pixel correlation in an given specific direction in both cases.

The GLCM's for 'mosaic2.png' are all different, which is also the case for the textures. They are all easier to distinguish form each other.

Select on of the GLCM features and discuss which parts of the matrices that seem to be useful for discriminating between textures

I would think that the GLCM homogeneity feature could be useful to extract features that can discriminate between textures. This i due to the weight function. It will have large contributions from the diagonal of the GLCM matrix. Since most of the high values in all of the GLCM's are along the diagonal i would think it this would be a very useful feature extraction.



Weight function of homogeneity feature

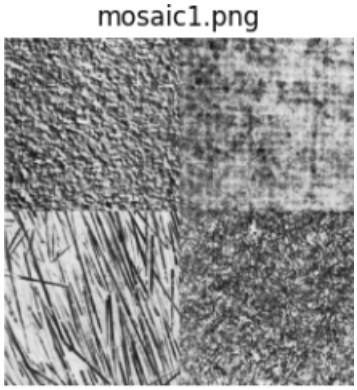
C. Computing GLCM feature images in local windows

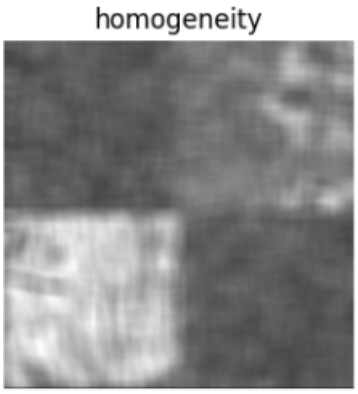
Please give the reasons for your choice of window size:

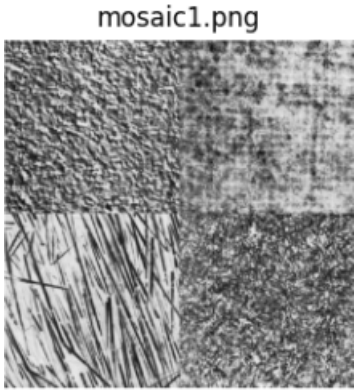
The window be large enough to capture the statistics inside the window. It is recommended to use between 31-51 window size. I tried different sizes in between these values, but concluded with 31 giving the feature map results.

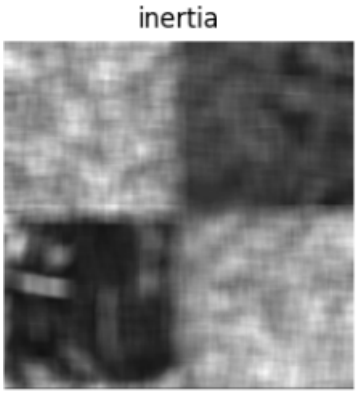
Please include the GLCM feature images in your report:

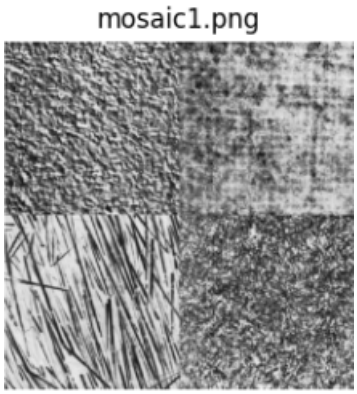
mosaic1.png, dx = 1, dy = 3, gray_levels = 16:

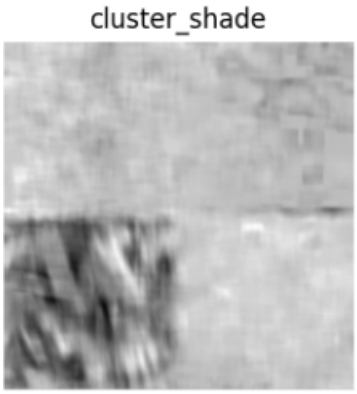




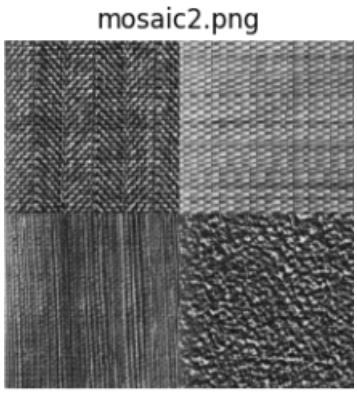


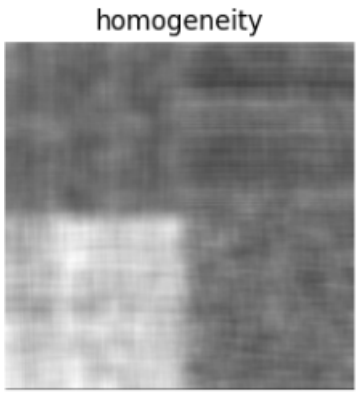


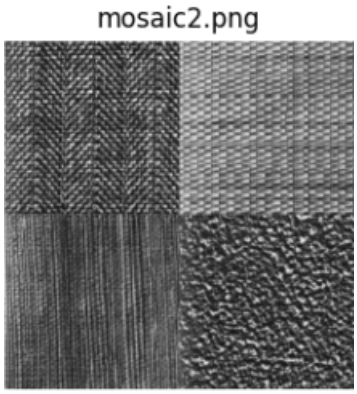


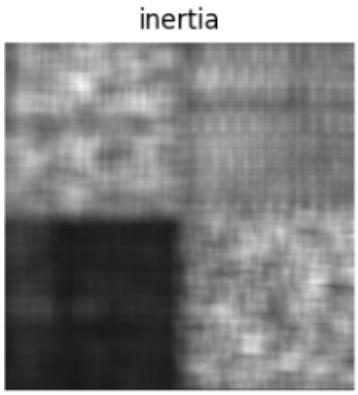


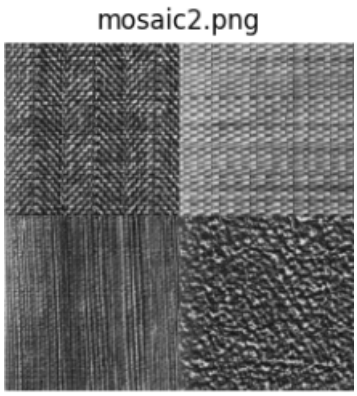
mosaic2.png, dx = 0, dy = 3, gray_levels = 15:

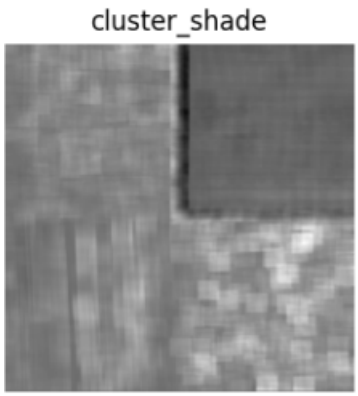












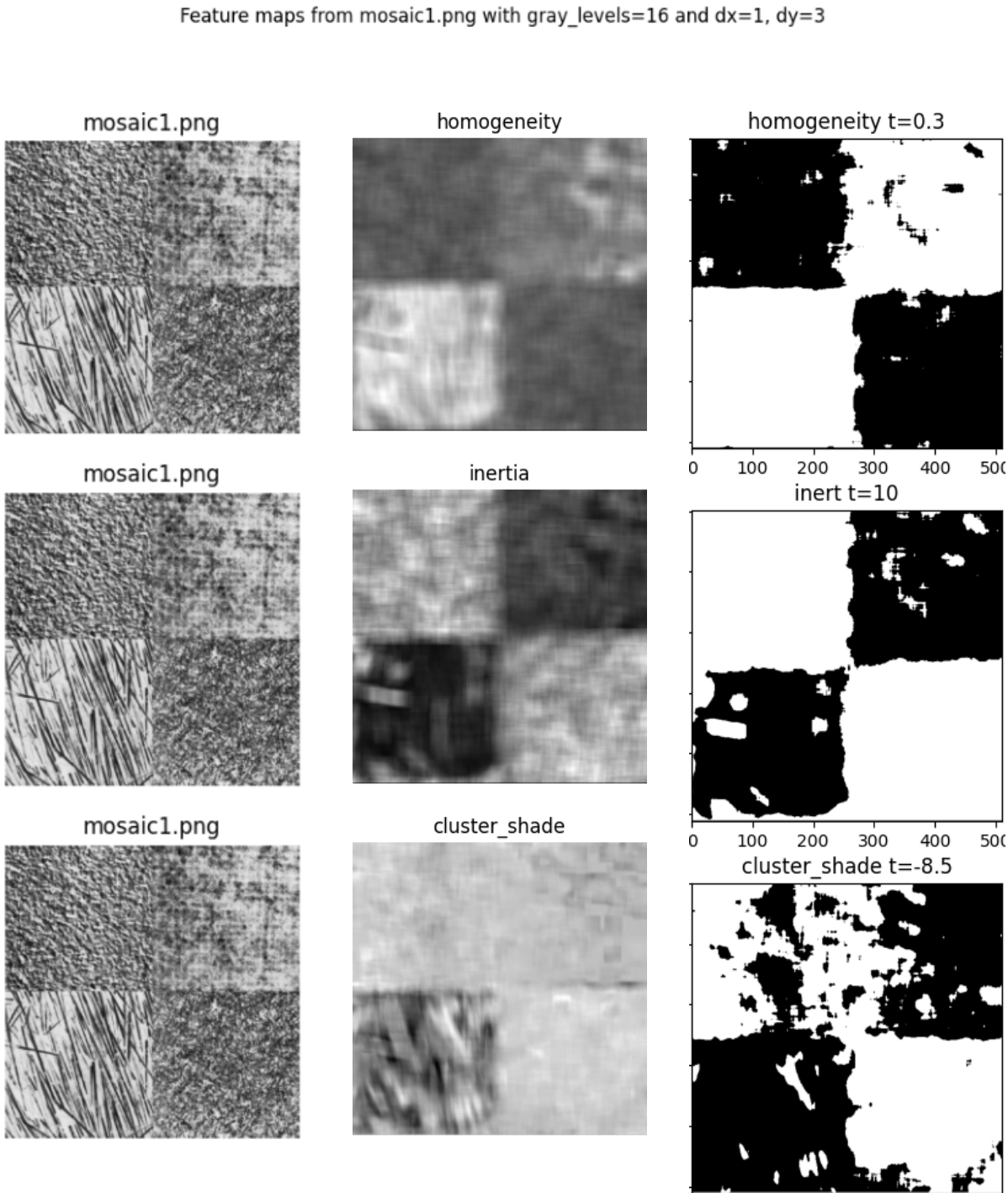
D. Segment the GLCM feature images and describe how well the texture regions are separated

Apply global thresholding to the GLCM feature images

The threshold values have been found by trial and error.

I choose to use dx=1, dy=3 for ‘mosiac1.png’ due to two of the textures having that direction. The two remaining textures have no specific direction.

For ‘mosiac2.png’ i chose dx=0, dy=3. This is mostly due to texture 7 having long textures in that directions. While texture 5 have some textures in this direction (when changing from south-east to north-east) as well as texture 6 having some small textures in this direction. Which i mentioned in the visual observation

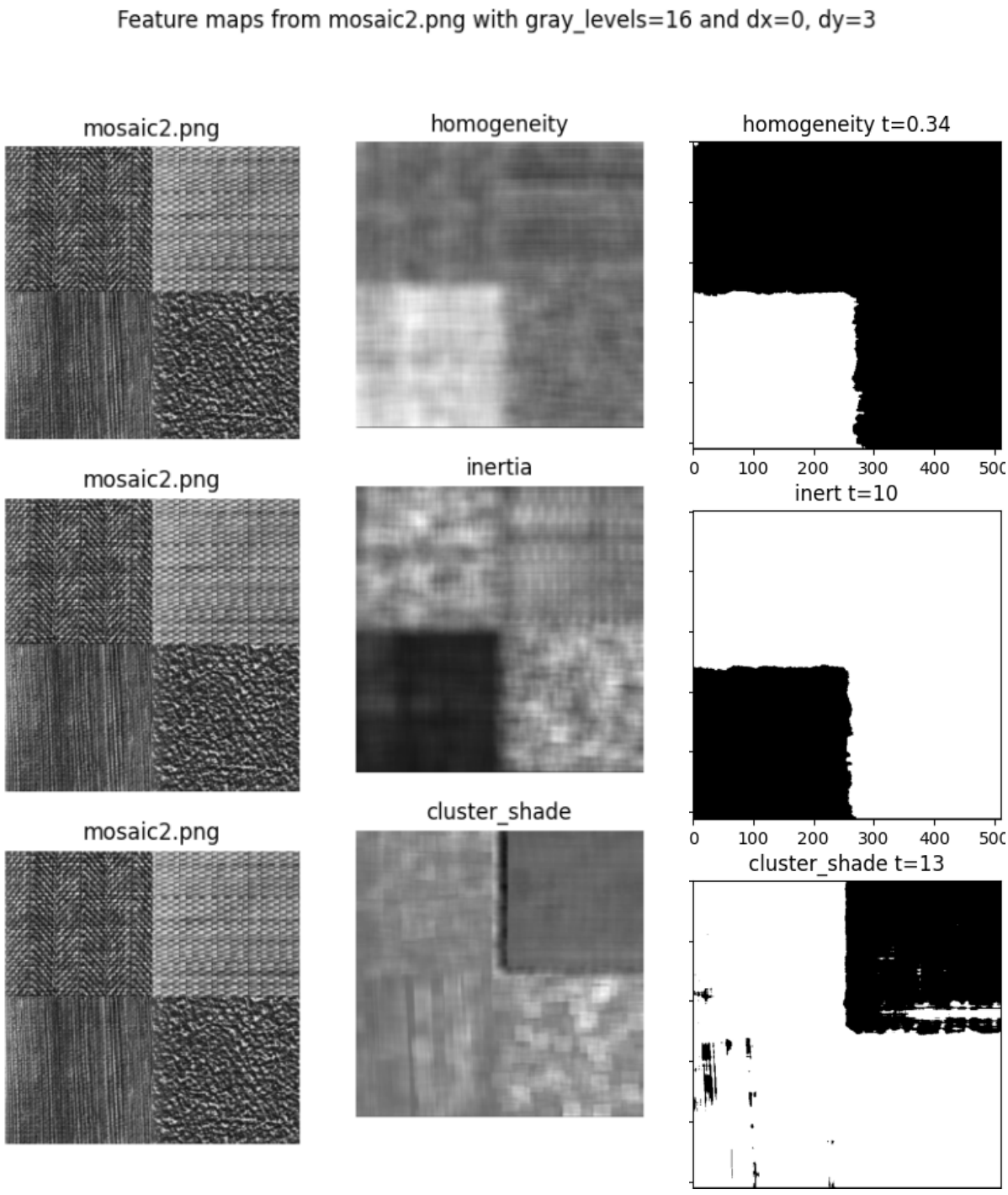


According to the weight function for homogeneity, low values corresponds to in-homogeneous images and higher vales for homogeneous images. Looking at the feature image this is correct, according to local homogeneity when using a sliding window.

The for inertia or contrast feature favors elements that are not in the diagonal line of the GLCM. Looking at the GLCM for texture 1 and 4, those had contribution outside the diagonal, which also could be the reason why inertia segments those textures, while texture 2 and 3 have more contributions along the diagonal.

I am having trouble interpreting the cluster shade function. I found little information about it and am unsure to what the weight actually favors in the GLCM matrix. It might be that it looks at clusters with low pixel

intensities. Which correlates to the result i got,



I expected texture 6 and 7 to be segmented as homogeneous, based on the GLCM done for those textures. But i seems that only texture 7 had higher relative values, meaning that the GLCM's created by the sliding window has a lot of values on the diagonal, while the other 3 dont.

The Inertia is as expected, based on the observations made from task B. Texture 7 had no contributions except along the diagonal, while the other 3 had some contributions outside the diagonal.

Again, i am very uncertain on what the cluster shade does. This result does not resemble what i expected. I thought that clusters with low pixel values would be segmented.

How can segmentation result be combined in order to solve the whole problem

I do not have the results to segment all the 4 different textures for either ‘mosiac1.png’ or ‘mosiac2.png’.

If one were to include different threshold values i think the segmentation would have better results. For example looking at homogeneity for ‘mosiac1.png’ if we had 3 threshold values we could at least extract texture 2 and 3 from texture 1 and 4.

If we inspect the segmentation results from ‘mosiac2.png’ its clear that inertia clearly segments out texture 7 and cluster_shade segments texture 6. If we change the threshold to not be reverse, these segments has a value of 1 and the other textures 0 (i.e. invert of the images diplayed). Then the cluster_shade can be subtracted from inertia result. This would allow us to at least discriminate between texture 6,7 and texture 5,8.

Appendix

Code:

assignment1.py

```
from glcm import glcm, compute_glcm_features, homogeneity, inertia, cluster_shade, isotropic_glcm
from img_utils import requantize, threshhold, save_image, extract_subimages, read_image
import matplotlib.pyplot as plt


def taskB():
    t1 = extract_subimages('mosaic1')
    t2 = extract_subimages('mosaic2')

    #Gray levels
    gray_levels = 16

    #Mosaic1 parameters
    glcm_mosiac1 = []
    dx = [2, 1,           2 , 1]
    dy = [2, "isotropic", 6, "isotropic"]

    #Mosaic1 parameters
    glcm_mosiac2 = []
    # dx_2 = [1, 0, 2, 4]
    # dy_2 = [1, 2, 0, "isotropic"]
    dx_2 = [3, 2, 0, 1]
    dy_2 = [-3, 0, 1, "isotropic"]

    #Mosaic1
    glcm_mosiac1.append(glcm(requantize(t1[0], gray_levels=gray_levels), dx=dx[0], dy=dy[0], gray_levels=gray
_levels))
    glcm_mosiac1.append(isotropic_glcm(t1[1], gray_levels, distance = dx[1]))
    glcm_mosiac1.append(glcm(requantize(t1[2], gray_levels=gray_levels), dx=dx[2], dy=dy[2], gray_levels=gray
_levels))
    glcm_mosiac1.append(isotropic_glcm(t1[3], gray_levels, distance = dx[3]))
    save_image(t1, glcm_mosiac1, gray_levels=gray_levels, dx=dx, dy=dy, filename='mosaic1', texture_start=0)

    #Mosaic2
    glcm_mosiac2.append(glcm(requantize(t2[0], gray_levels=gray_levels), dx=dx_2[0], dy=dy_2[0], gray_levels=
gray_levels))
    glcm_mosiac2.append(glcm(requantize(t2[1], gray_levels=gray_levels), dx=dx_2[1], dy=dy_2[1], gray_levels=
gray_levels))
```

```
    glcm_mosiac2.append(glcm(requantize(t2[2], gray_levels=gray_levels), dx=dx_2[2], dy=dy_2[2], gray_levels=
gray_levels))
    glcm_mosiac2.append(isotropic_glcm(t2[3], gray_levels, distance = dx_2[3]))

    save_image(t2, glcm_mosiac2, gray_levels=gray_levels, dx=dx_2, dy=dy_2, filename='mosaic2', texture_start
=4)


def taskC_D(filename, window_size, dx, dy, gray_levels, threshold):
    img1 = read_image(filename)

    homogen = compute_glcm_features(img1.copy(), window_size=window_size, dx=dx, dy=dy, gray_levels=gray_leve
ls, feature_fn=homogeneity)
    inert = compute_glcm_features(img1.copy(), window_size=window_size, dx=dx, dy=dy, gray_levels=gray_level
s, feature_fn=inertia)
    clust = compute_glcm_features(img1.copy(), window_size=window_size, dx=dx, dy=dy, gray_levels=gray_level
s, feature_fn=cluster_shade)

    homogen_T = threshhold(homogen, threshold[0])
    inert_T = threshhold(inert, threshold[1])
    clust_T = threshhold(clust, threshold[2])

    fig = plt.figure(figsize=(20,10))
    ax1, ax2, ax3 = fig.subplots(3,3)

    for i in range(2):
        ax1[i].axis('off')
        ax1[i].set_aspect(1)

        ax2[i].axis('off')
        ax2[i].set_aspect(1)

        ax3[i].axis('off')
        ax3[i].set_aspect(1)

    ax1[0].imshow(img1, cmap=plt.cm.gray)
    ax1[0].set_title(f'{filename}')

    ax1[1].imshow(homogen, cmap=plt.cm.gray)
    ax1[1].set_title(f'homogeneity')

    ax1[2].imshow(homogen_T, cmap=plt.cm.gray)
    ax1[2].set_title(f'homogeneity t={threshold[0]}')

    ax2[0].imshow(img1, cmap=plt.cm.gray)
    ax2[0].set_title(f'{filename}')

    ax2[1].imshow(inert, cmap=plt.cm.gray)
    ax2[1].set_title(f'inertia')

    ax2[2].imshow(inert_T, cmap=plt.cm.gray)
    ax2[2].set_title(f'inert t={threshold[1]}')

    ax3[0].imshow(img1, cmap=plt.cm.gray)
    ax3[0].set_title(f'{filename}')

    ax3[1].imshow(clust, cmap=plt.cm.gray)
    ax3[1].set_title(f'cluster_shade')

    ax3[2].imshow(clust_T, cmap=plt.cm.gray)
    ax3[2].set_title(f'cluster_shade t={threshold[2]}')

    fig.suptitle(f'Feature maps from {filename} with gray_levels={gray_levels} and dx={dx}, dy={dy}')
    plt.savefig(f"{filename}_featuremaps_dx{dx}_dy{dy}.png")

    plt.show()

def main():
    taskB()

    taskC_D('mosaic1.png', window_size=31, dx=1, dy=3, gray_levels=16, threshold=[0.3,10,-8.5])
    taskC_D('mosaic2.png', window_size=31, dx=0, dy=3, gray_levels=16, threshold=[0.34,10,13])
```

```
main() #23
```

glcm.py (Some functions taken from solution of texture exercises and some are created by me)

```
import matplotlib.pyplot as plt
import numpy as np
from numba import jit
from tqdm import tqdm

from img_utils import requantize
from img_utils import SlidingWindowIter

## CODE TAKEN FROM SOLUTION GIVEN TO EXERCISES ##

@jit(nopython=True)
def glcm(quantized, dy, dx, gray_levels=256, normalise=True, symmetric = True):
    glcm = np.zeros((gray_levels, gray_levels))
    for y in range(quantized.shape[0] - abs(dy)):
        for x in range(quantized.shape[1] - abs(dx)):
            px1 = quantized[y, x]
            px2 = quantized[y + dy, x + dx]
            glcm[px1, px2] += 1

    if normalise and symmetric:
        glcm += np.transpose(glcm) #Added this to the code

        return glcm * (1 / np.sum(glcm))

    if normalise:
        return glcm * (1 / np.sum(glcm))

    return glcm

def compute_glcm_features(image, window_size, dy, dx, gray_levels, feature_fn):
    feature_image = np.zeros(image.shape)
    quantized = requantize(image, gray_levels)

    for x, y, window in tqdm(SlidingWindowIter(quantized, window_size)):
        matrix = glcm(window, dy, dx, gray_levels)
        feature_image[y, x] = feature_fn(matrix)
    return feature_image


## CODE CREATED FOR THIS ASSIGNMENT ##

@jit(nopython=True)
def inertia(matrix):
    inr = 0
    for i in range(matrix.shape[0]):
        for j in range(matrix.shape[1]):
            inr += matrix[i, j] * (i - j)**2
    return inr

@jit (nopython=True)
def homogeneity(matrix):
    idm = 0
    for i in range(matrix.shape[0]):
        for j in range(matrix.shape[1]):
            idm += matrix[i, j] * (1 / (1+(i-j)**2))
    return idm

@jit (nopython = True)
def px(i, matrix):
    P = 0
    for j in range(matrix.shape[0]):
```

```
        P += matrix[i, j]
    return P

@jit (nopython = True)
def py(i, matrix):
    P = 0
    for j in range(matrix.shape[0]):
        P += matrix[j, i]
    return P

@jit (nopython = True)
def ux(matrix):
    u = 0
    for i in range(matrix.shape[0]):
        u += i * px(i, matrix)
    return u

@jit (nopython = True)
def uy(matrix):
    u = 0
    for i in range(matrix.shape[0]):
        u += i * py(i, matrix)
    return u

@jit (nopython = True)
def cluster_shade(matrix):
    u_x = ux(matrix)
    u_y = uy(matrix)

    shd = 0
    for i in range(matrix.shape[0]):
        for j in range(matrix.shape[1]):
            shd += matrix[i, j] * (i+j-u_x-u_y)**3
    return shd

def isotropic_glcm(image, gray_levels, distance=1):
    quant_image = requantize(image, gray_levels=gray_levels)
    glcm_0 = glcm(quant_image, dx=distance, dy=0, gray_levels=gray_levels, normalise=True)
    glcm_45 = glcm(quant_image, dx=distance, dy=distance, gray_levels=gray_levels, normalise=True)
    glcm_90 = glcm(quant_image, dx=0, dy=distance, gray_levels=gray_levels, normalise=True)
    glcm_135 = glcm(quant_image, dx=-distance, dy=distance, gray_levels=gray_levels, normalise=True)

    isotropic = (glcm_0+glcm_45+glcm_90+glcm_135) / 4

    return isotropic
```

img_utils.py (some code taken from solution of exercises and some are made by me)

```
import numpy as np
from seaborn import heatmap
import matplotlib.pyplot as plt
from PIL import Image


## CODE TAKEN FROM SOLUTION GIVEN TO EXERCISES ##

class SlidingWindowIter:
    def __init__(self, image, window_size):
        self.wsize = window_size
        self.x = 0
        self.y = 0
        self.count = 0
        self.pad = (window_size - 1) // 2
        self.img_dims = image.shape
        self.padded = np.pad(
            image,
            ((self.pad, self.pad), (self.pad, self.pad)),
            mode="reflect",
        )
```

```
    def __iter__(self):
        return self

    def __next__(self):
        if self.y >= self.img_dims[0] + self.pad - 1:
            raise StopIteration
        else:
            self.x = (self.count) % self.img_dims[1] + self.pad
            self.y = self.count // self.img_dims[0] + self.pad
            self.count += 1
            return (
                self.x - self.pad,
                self.y - self.pad,
                self.padded[
                    self.y - self.pad : self.y + self.pad + 1,
                    self.x - self.pad : self.x + self.pad + 1,
                ],
            )

def requantize(img, gray_levels):
    return np.uint8(np.round(img * ((gray_levels - 1) / 255)))

def threshhold(img, T, reverse=True):
    """
    Simple global threshhold
    """

    thresholded = np.zeros(img.shape, dtype=np.uint8)
    for index, val in np.ndenumerate(img):
        if not reverse:
            if val < T:
                thresholded[index[0], index[1]] = 1
        else:
            if val > T:
                thresholded[index[0], index[1]] = 1
    return thresholded

## CODE CREATED FOR THIS ASSIGNMENT ##

def save_image(image, glcm_image, gray_levels, dx, dy, filename, texture_start):
    fig = plt.figure(figsize=(20,10))
    ax1, ax2 = fig.subplots(2,4)

    for i in range(4):
        ax1[i].axis('off')
        ax1[i].set_aspect(1)

        ax2[i].axis('off')
        ax2[i].set_aspect(1)

    ax1[0].imshow(image[0], cmap=plt.cm.gray)
    ax1[0].set_title(f'texture {texture_start+1}')

    heatmap(glcm_image[0],ax=ax1[1])
    ax1[1].set_title(f'dx: {dx[0]}, dy: {dy[0]}')

    ax1[2].imshow(image[1], cmap=plt.cm.gray)
    ax1[2].set_title(f'texture {texture_start+2}')

    heatmap(glcm_image[1],ax=ax1[3])
    ax1[3].set_title(f'dx: {dx[1]}, dy: {dy[1]}')

    ax2[0].imshow(image[2],cmap=plt.cm.gray)
    ax2[0].set_title(f'texture {texture_start+3}')

    heatmap(glcm_image[2],ax=ax2[1])
    ax2[1].set_title(f'dx: {dx[2]}, dy: {dy[2]}')

    ax2[2].imshow(image[3],cmap=plt.cm.gray)
    ax2[2].set_title(f'texture {texture_start+4}')
```

```
    heatmap(glcm_image[3],ax=ax2[3])
    ax2[3].set_title(f'dx: {dx[3]}, dy: {dy[3]}')

    fig.suptitle(f'GLCM of {filename} with gray_levels={gray_levels}')
    plt.savefig(f"{filename}_glcms.png")

def read_image(filename):
    f = np.asarray(Image.open(filename))
    return f

def sub_images(image):
    return np.array([image[:256, :256], image[:256,256:], image[256:,:256], image[256:,256:]],dtype=np.uint8)

def save_images(images,filename):
    for i in range(len(images)):
        Image.fromarray(images[i]).save(filename+"_texture"+str(i+1)+".png")

def extract_subimages(filename):
    img = read_image(filename+'.png')
    sub = sub_images(img)
    return sub
```