IN5520/IN9520 Mandatory term project 2020 – Part I

Segmentation of textured regions in an image

In this mandatory exercise you are going to describe textured regions in an image, compute and visualize GLCM's from each texture, extract GLCM feature images, and segment these images.

We recommend that you use Matlab and/or Python.

The next exercise (Part II) will be about classification, and will not necessarily be based on the same data set as Part I.

Time table

- Exercise Part I available: Wednesday 9th September 2020.
- Deadline Part I: Wednesday 30th September 2020 (3 weeks).

Submission:

Your solution must be submitted as a single PDF file containing the problem description, discussion, and the supporting source code. The files should be compressed (.zip or .tar) in a folder named YOURUIOUSERNAME_PARTI.zip/tar, and uploaded through the devilry system: http://devilry.ifi.uio.no before the deadline above. Questions about submission can be directed to the group teacher Edward Fabian Bull (*edwardfb@math.uio.no*).

Evaluation:

- The two mandatory exercises will be evaluated separately.
- Please note that in order to take the exam, both mandatory exercises must be passed.

Since image processing is a field where solutions often are found by experimenting with different methods, we would like to emphasis the following point:
You should analyze the problem and the input images so that you can select a suitable method and parameters. You should not start testing all available methods that you know of, even if that would be an impressive amount of work. Analysis and logical reasoning about alternatives, and then choosing and eventually comparing just a few approaches is usually a better approach.

How to work:

The exercise is an individual work, and each student should deliver a written report.
Your report should be genuine, in particular we will check that each report provides its own discussion of all method and parameter choices. Include references if you use external sources.

The report should contain a description of the problem, theory, chosen methods, results and algorithms used. You have to document all steps in the algorithms, and listings of your own code should be included as appendix.

The images:

The following two images should be used:
www.uio.no/studier/emner/matnat/ifi/IN5520/h20/undervisningsmateriale/mandatory1/mosaic1.png
www.uio.no/studier/emner/matnat/ifi/IN5520/h20/undervisningsmateriale/mandatory1/mosaic2.png

The four steps:

Part I can be divided into four steps, but must be seen together.

A. Analyzing the textures.

Describe the 8 different textures by words. What characterizes each texture? How do the textures differ? Keywords: texture direction, frequency, variance, homogeneity, texture element size.

B. Visualizing GLCM matrices

- For this step, create one subimage for each texture.
- Eventual histogram transforms must be discussed.
- **Instead of testing a massive list of GLCM parameter combinations, you will be rewarded for giving good reasons for a shorter list of parameters, so you should explain your choice of parameter values.** This should be related to your analysis in A.
- Requantize the image and compute the normalized, symmetric GLCM matrix for the chosen parameter values for each texture. If you use isotropic GLCM, you should give details on how it is computed. You should discuss whether directional or isotropic GLCM is beneficial for this specific problem. Again, your decision here should reflect what you observed during your analysis in A.
- We suggest that you try to find just a few values of the parameters (d, θ) that can be used for all features, knowing that the chosen parameters should help you differentiate between the textures in the mosaic.
- Include the GLCM matrices as images in the report, and include a color bar on the figures.
- From the GLCM matrices, discuss similarities and differences between the image textures. Select one of the GLCM features below, and discuss which parts of the matrices that seem to be useful for discriminating between the textures.

C. Computing GLCM feature images in local windows

Compute feature images from the GLCM features:

- GLCM homogeneity, also called Inverse Difference Moment (IDM)

$$IDM = \sum_{i=0}^{G-1} \sum_{j=0}^{G-1} \frac{1}{1+(i-j)^2} P(i,j)$$

- GLCM inertia (called Contrast in G&W)

$$INR = \sum_{i=0}^{G-1} \sum_{j=0}^{G-1} \{i-j\}^2 \times P(i,j)$$

- GLCM cluster shade

$$SHD = \sum_{i=0}^{G-1} \sum_{j=0}^{G-1} \{i+j-\mu_x-\mu_y\}^3 \times P(i,j)$$

To calculate a feature image, you let a local window glide over the entire image of the four textures and for each window position you get one GLCM matrix which – when multiplied with the proper feature weight function - will give you one scalar value that position for the chosen feature. Please give the reasons for your choice of window size.

Please include the GLCM feature images in your report. Make sure to scale them so they show proper contrast in your PDF file.

D. Segment the GLCM feature images and describe how they separate the textures.

Apply global thresholding to the GLCM feature images. You can experiment with the global threshold to manually find a threshold value for each feature image that gives the best separation of the textured regions.

Include the tresholded feature images in your report, and discuss which image textures that are best separated in each GLCM texture feature image, both in terms of correct boundaries between textures and errors within the textured regions.

You will see that no single thresholded feature image can distinguish between all textures. Discuss how segmentation results can be combined in order to solve the whole problem.

General remarks:

- You may use libraries and pre-programmed functions, as long as you cite your source(s).
- If you want to make absolutely sure that you understand the GLCM method, it may be wise to implement it.
- You should not test a massive list of GLCM feature/parameter combinations.
- You will be rewarded for giving a good discussion based on the image textures and the given GLCM features, leading up to a shorter list of features/parameters.
- You should try to find combinations of features/parameters that are not correlated, but rather complementary to each other.

Mandatory excersice in5520

A

Mosaic1 upper left corner

A lot of sharp edges with high kontrast. Unstructured, Seems very unorganized and random. No structure in any direction. The image is very grained, not smooth

Mosaic1 upper right corner

Soft transitions between contrasts and the contrast is not too big. There is some structure we can see, in both theta=90 degrees and theta =0 degrees there are white lines along those degrees. Quite large areas of homogeneity

Mosaic1 lower left corner

Here there is also quite a lot of sharp edges with high contrast. The image is fairly structured where one can see a lot of black lines going in the same direction of approximatly theta=120 degrees. There are som areas of homogeneity but these areas are quite small. The image is quite smooth.

Mosaic1 lower right corner

This picture resembles the first picture quite a bit, however this picture is even more grained. There are a lot of edges but the contrast is not too big. There is absolutely no structure and it seems very random. Very grained and very unhomogenous.

Mosaic2 upper left corner

This image is quite structured in many different angles. We see structures in 45 degrees and 135 degrees, as well as some structure at 90 degrees. There is little contrast and the image is quite dark but not so homogenous

Mosaic2 upper right corner

A very structured picture with little contrast. A lot of sharp edges. Here we see that the lines of the structures is in the direction of 90 degrees, 0 degrees and approximatly 30 degrees.

Mosaic2 lower left corner

Quite homogenous picture with a lot of dark pixels and little contrast. There is a clear structure at 90 degrees.

Mosaic2 lower right corner

Here we see a lot of edges again, however not so sharp edges. There is some contrast and the picture is very unhomogenous with no structure in any angles.

B

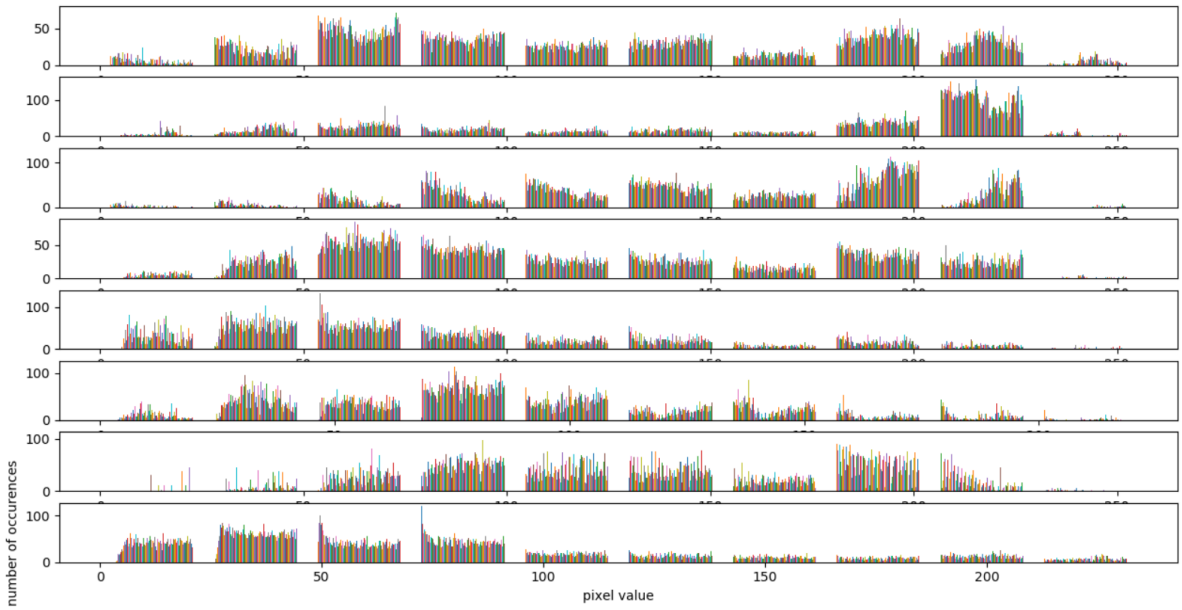


Fig 1: Histogram over all the pixel values for each texture.

As we can see in the histogram of all the pictures, there are a few of them that stand out a little. Especially texture 2 have a quite narrow distribution most values are centered in a few pixel values far to the right. We also see that texture 5 and 8 have an equal distribution just to the left. This could be used to differentiate between textures 2 and 5,8, however the differentiation would not be very good, since there are still quite some pixels distributed over all pixel values in these textures.

From the histogram I have also calculated mean, variance, skewness and kurtosis. Again some of the pictures vary a little in the values of these metrics, but it is not a clear difference and would hardly be useful in this problem to distinguish between the textures. The values for mean, variance, skewness

and kurtosis will be written out to terminal when calling the make_histograms()-function in the script Test1.py.

```
D:\in5520\Oblig1>python test1.py 0 0 0 0 0 0 0 0
[7.2656097412109375, 9.3892822265625, 8.775222778320312, 6.8530426025390625, 4.800689697265625, 4.810821533203125, 7.841888427734375, 4.376922607421875] mean
[14.812279224162921, 18.232828244566917, 9.186489971121773, 13.480158284073696, 12.383950940333307, 7.189126058481634, 9.796503416262567, 12.274677394889295] variance
[0.12907665383095676, -0.6538988781313918, -0.43483677360873757, 0.29494448901251974, 0.8294316436234226, 0.6674900417061735, 0.03615794391374345, 0.9908423219128742] skewness
[1.7897724120933356, 1.912146780629418, 2.3347903024728227, 1.858652427169274, 2.8084406901929397, 2.9716242074519528, 1.9820102013183487, 3.055664799350506] kurtosis
[0.93675801 0.94800557 0.90183076 0.93093998 0.92528365 0.87788685
 0.90737742 0.9246686 ] smoothness
```

Fig 1.2: Values for mean, variance, skewness and kurtosis for the 8 different textures.

Based on the analysis of the pictures in task A, there are a lot of the pictures that have some structure in the angles theta=0,30,45,90, 120, 135. When it comes to the distance d, it is a little hard to say what is the best d, since it can be difficoult to say how many pixels a structure is in the images, so i have tested and tried different d-values. I started with using low values, from 1-5. After a while I started testing with d-values of 7, 10 and even 15, which also gave som good results.

In this task I have not used Isotropic GLCM since a lot of the structures are not equal in any direction, that is, a lot of the images have structures/patterns that go in one direction in particular, so using Isotropic GLCM didn't seem too useful.

GLCM

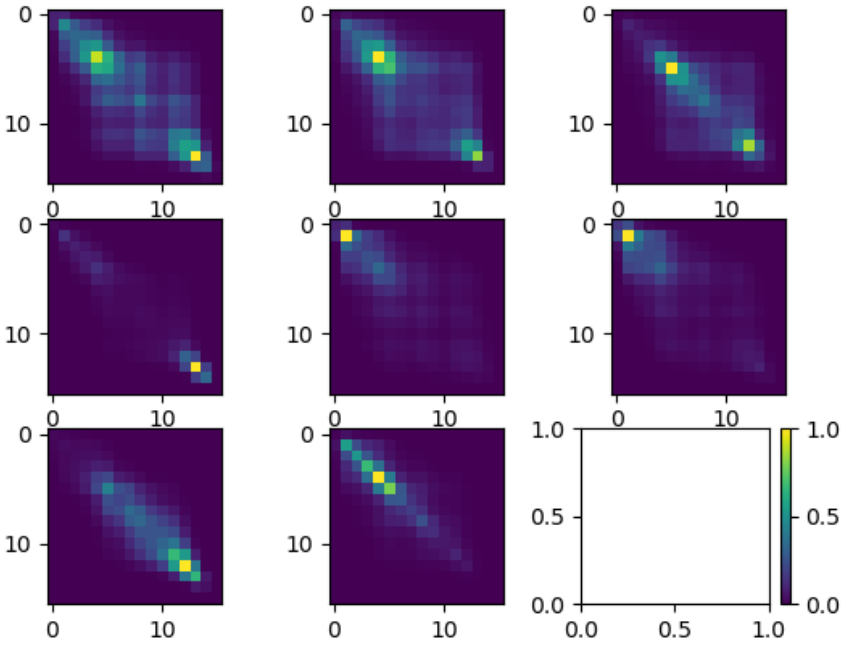


Fig 2: GLCM plot for d=2 theta=20

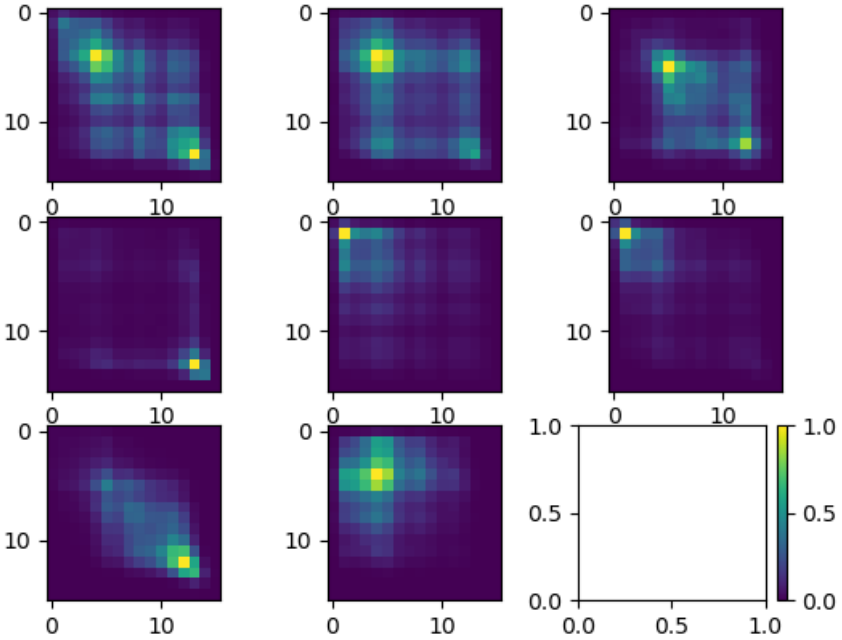


Fig 3: GLCM plot for d=3 theta=60

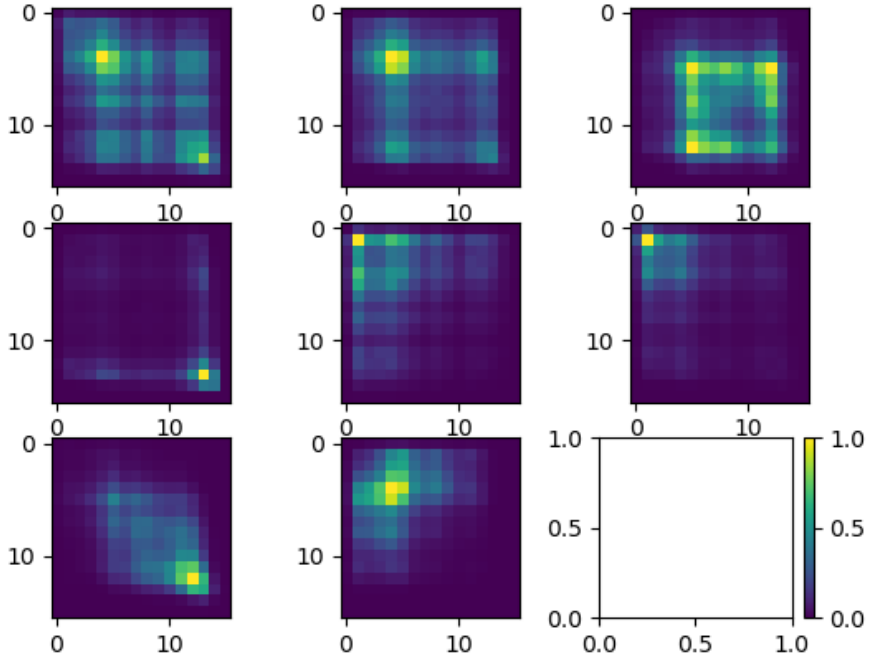


Fig 4: GLCM plot for d=4 theta=50

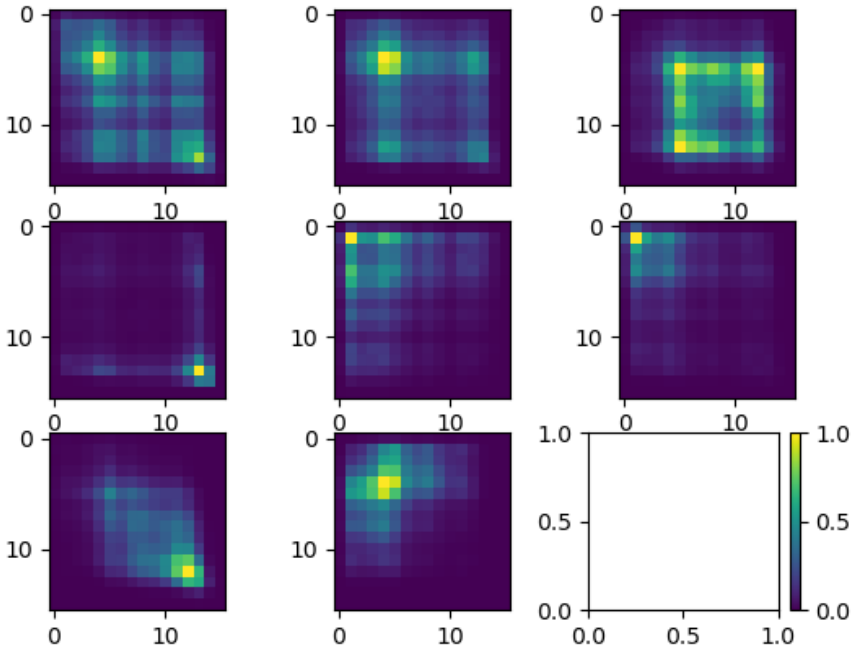


Fig 5: GLCM plot for d=4 theta=60

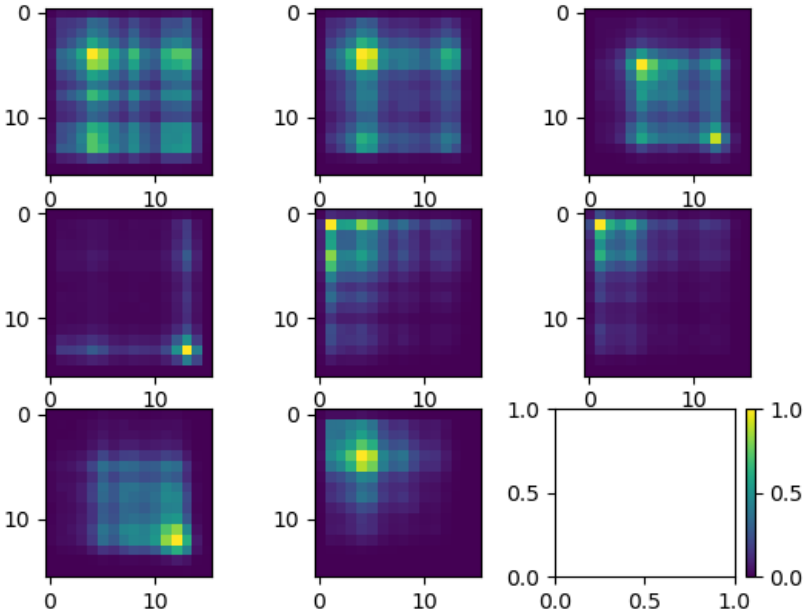


Fig 6: GLCM plot for d=15 theta=120

These are the GLCMs I have used for generating the feature-maps in Task C. I tried using only the angles and distances that seemed reasonable from the analysis in A in the beginning and then I started experimenting with parameters to get the best feature-maps. We see that the choices of parameters I ended up with differs a little from what I thought would give the best GLCM's according

to the analysis in A, however we do have 120, 20 (which is quite close to 30), and 60 (which is quite close to 45). Also angles 0 and 90 gave quite good feature-maps, however other angles gave even better results.

In the GLCM plots with the eight GLCM-matrices, we start counting in the upper left corner which corresponds to the upper left corner of mosaic1.png, we call that texture 1. Texture 2 is the GLCM-matrix below texture 1, and corresponds to the lower left corner of mosaic1. The texture below Texture 2 in the GLCM-plot is Texture 3 and corresponds to the upper right corner of mosaic1. So in both the GLCM-plot and in the mosaic-images we start at the upper left corner and goes downward from there.

Texture 1 in the GLCM-plot is, no matter what the parameters are, quite widespread. This makes Sense since the picture is very unhomogenous and no structure in any particular angle. Texture 5 and 8 always is very similar to each other no matter what the parameters are, which makes sense, since the upper left and lower right picture in mosaic2 resembles each other quite a bit, even though texture 5 is more structured than texture 8. Texture 7 (upper right corner of mosaic2) is the one that changes the most with changes in parameters, which also makes sense, since it has a lot of structure in many different angles and distances, so the GLCM looks quite different with changes in the parameters, specially the angles. We can also see that the GLCM with d=2, looks very much more homogenous than the other GLCM-plots, which is reasonable, since the probability of finding completely different pixel values increase with the distance. However we can see that for texture 1, the distribution of the pixels is not that different from d=3 and d=15, which means that the image is very random even at d=3 and very little homogenous.

If we look at this through the IDM function to create feature-maps, what can we say about the feature-map from the GLCM? IDM is called the homogeneity function, because it gives the values who lie on the i=j-line high values, while the ones who lie off from this line gets lower values. Thus we see that the GLCM-matrices with a wide distribution of values, will typically get low pixel-values in the feature-map, and the GLCM matrices with a dot or a line in the diagonal i=j will become very bright parts of the feature-map. So when it comes to the IDM feature-function, it is wheter or not the bright pixels lie at the diagonal i=j or not and how much of it lies far away from this line that makes the difference. It is called the homogeneity-function since high GLCM-values at the diagonal i=j corresponds to very homogenous pictures. Thus we see that widespread GLCM's will get a low colour on the IDM feature-map, and thin GLCM's on the diagonal i=j will become very bright in the IDM feature-map.

Note: Even though the brightest pixels is completely yellow in all of the GLCM-matrices, does not mean that pixels who are yellow have the same value across different plots. For instance the yellow pixel in texture 1 and texture 2 have quite differnt values even though they have the same color in the different plots.

C/D

Below are the feature images from the given GLCM-parameters that gave the biggest difference between the textures. I have used window size 41 for every single feature-map except for the one in figure 10 where I tried using window size of 51. The reason why I have used 41 and 51 is because from the lecture slides it said that one should use a window size of 31-51 when using 16 possible values for each pixel. So I stayed withing that range. The reason I went for 41 for most of them, is because it seemed to be better results from using 41 instead of 51, and when I used a window size of

51 it became worse in differentiating between textures in and around the borders of the textures. So with a window size of 41 it became a little bit more detailed between the textures, while giving good differences between them.

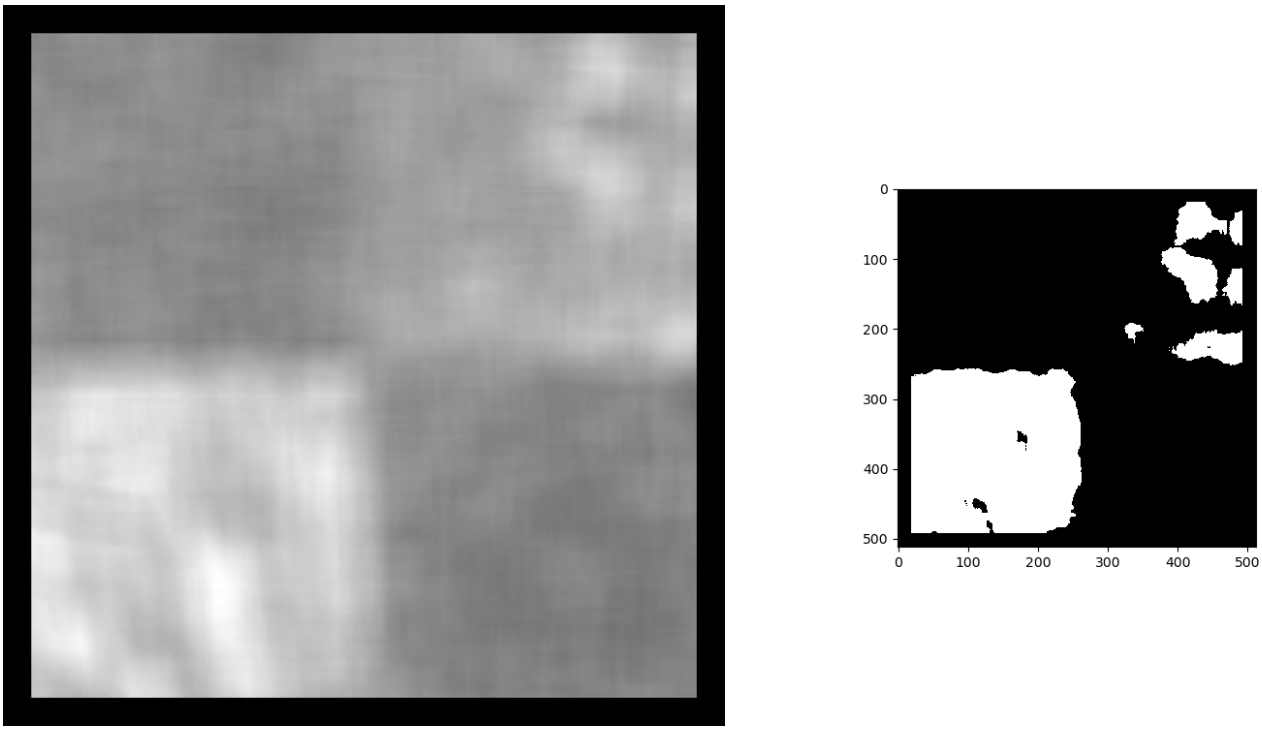


Fig 7: left: Feature-map from d=20 theta=20 using IDM from mosaic1. right: Mask from the feature map trying to distinguish the texture in the lower left corner

Here we see that the texture in the lower left corner is the one that is best differentiated, and we also see in the mask that the texture is very well covered, maybe except for in the boundary to other textures. All the other textures are almost completely gone in the mask, except a bit from the texture in the upper tight corner.

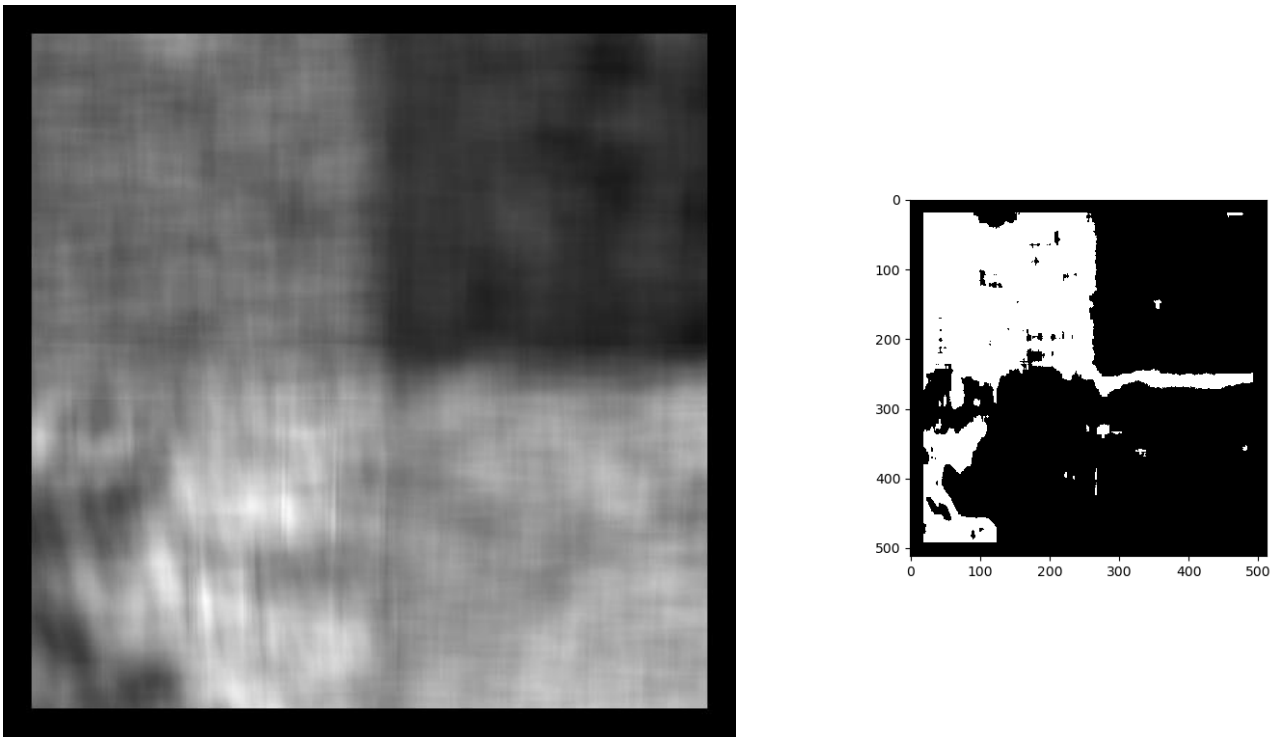
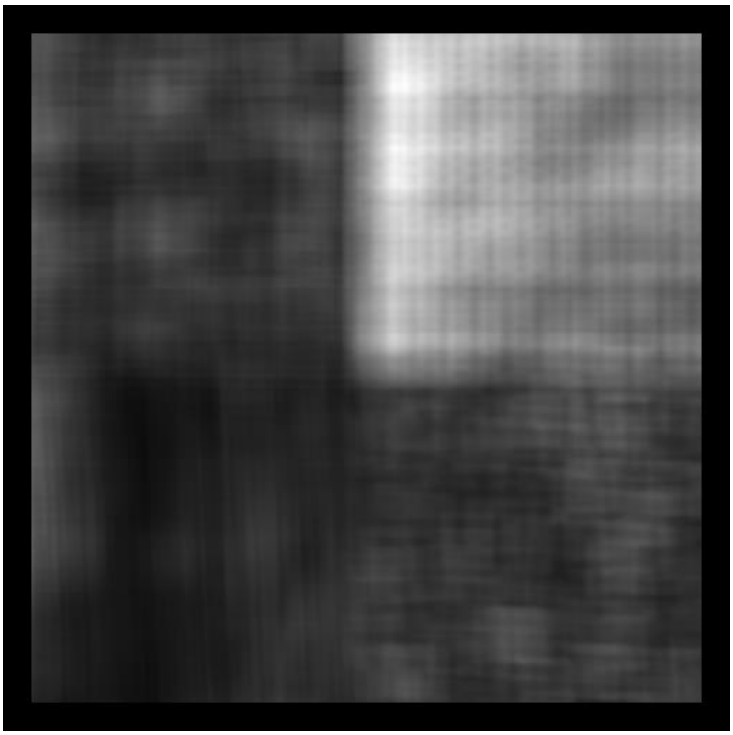


Fig 8: left: Feature-map d=3 theta=60 using inertia from mosaic2. right: Mask from the feature-map trying to distingush the textures in the upper left and upper left corner.

In this feature-map the texture which is best singled out is the one in the upper right corner, which is very well differentiatied from the other textures. However that texture is quite easy to distinguish, and we have to try and distinguish the other ones as well, so we use a lower and upper threshold to detect only the gray texture in the upper left corner, which is quite well detected, however other regions that dont belong to this texture is also detected.



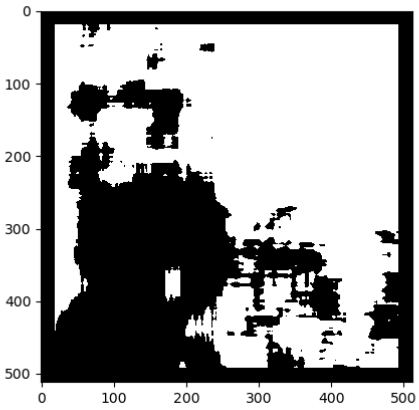
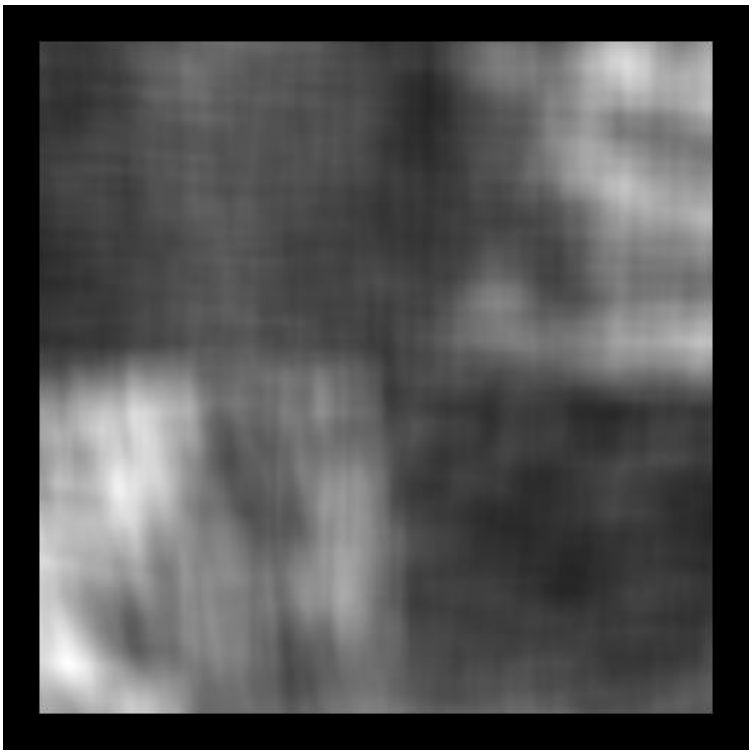


Fig 9: left: feature-map d=3 theta=60 with SHD from mosaic2. right: Mask from the feature-map trying to distingush the textures in the upper left and lower left corner.

Here we see again that the texture in the upper right corner is very well distinguished, but we want to distinguish the very dark corner in the lower left. Here the detection is not so good. The texture is not too well detected with holes and no detection around the edge and it detects other regions that does not belong to the texture as well, so this was not a very good detection, but we have some idea of which texture is detected by looking at the mask.



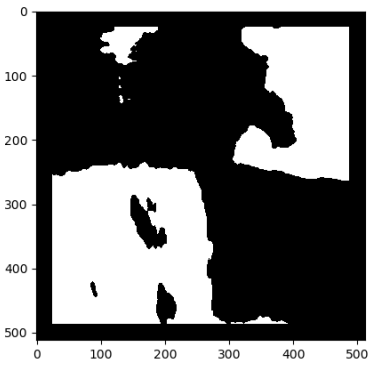
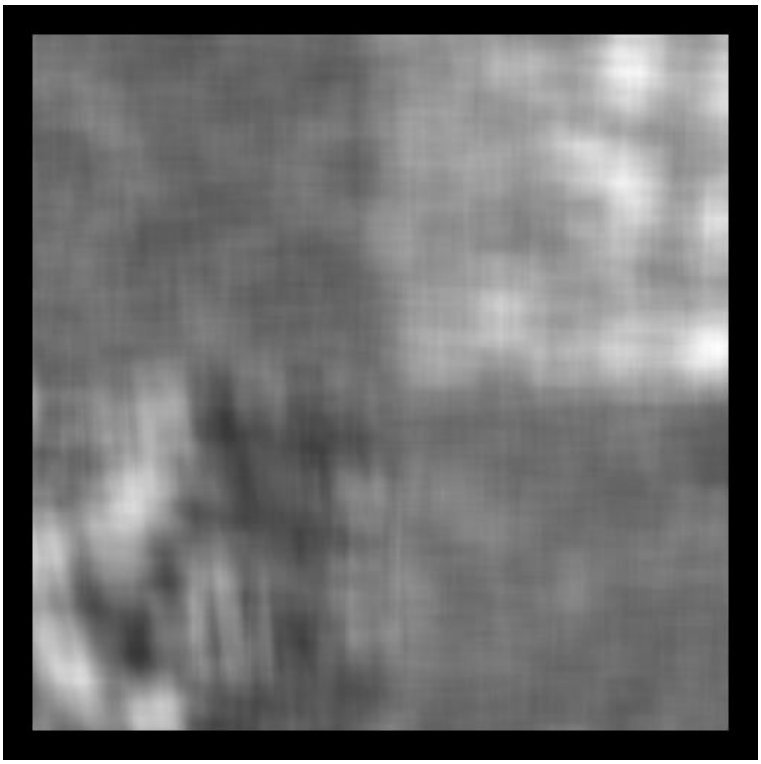


Fig 10: left: feature-map with d=4 theta=50 using SHD with window size=51 from mosaic1. right: Mask from the feature-map trying to distingush the textures in the upper left and lower right corner.

Here we are trying to distinguish two textures from the two others, that is we want to distinguish the lower right corner and the upper left, since they are fairly dark in the feature-map. The detection didn't og too well with holes and not too good detection around the edges, but again we get a good indication for where the two types of textures lie.



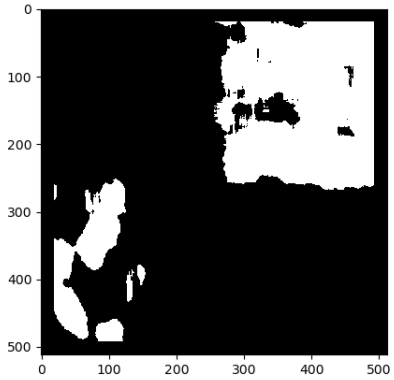
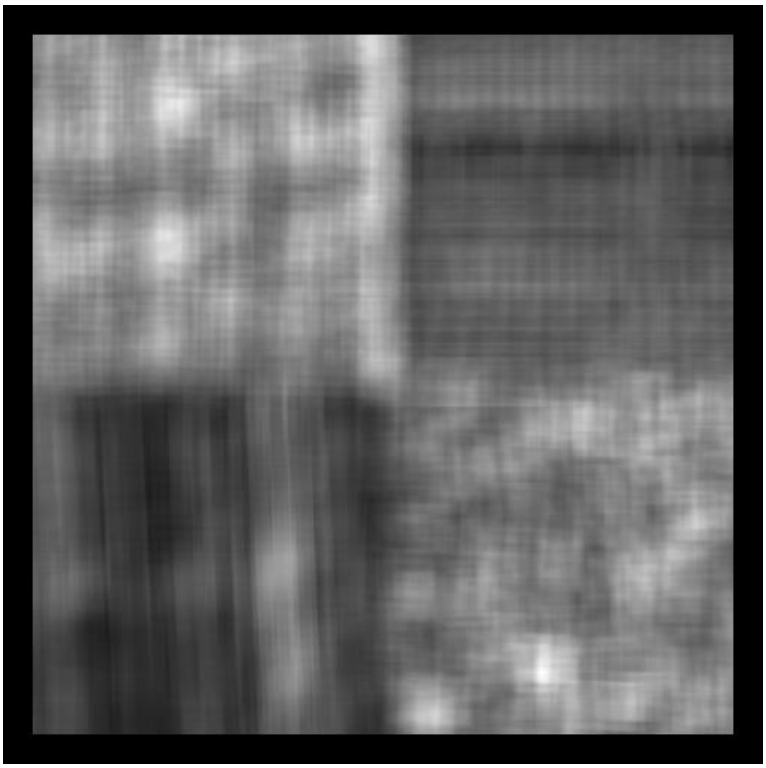


Fig 11: left: feature-map with d=4 theta=60 using IDM from mosaic1. right: Mask from the feature-map trying to distinguish the texture in the upper right corner

Here we try to distinguish the upper right corner. As we see the upper right corner is lighter than everything else. The mask is quite good, with few and small holes, but other things are also detected which should not be detected.



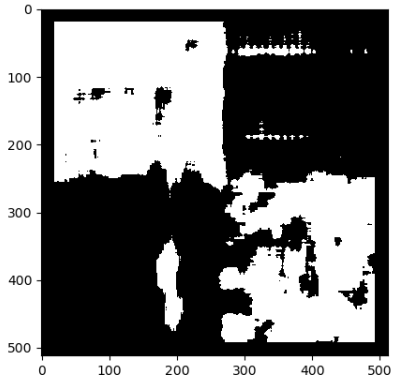
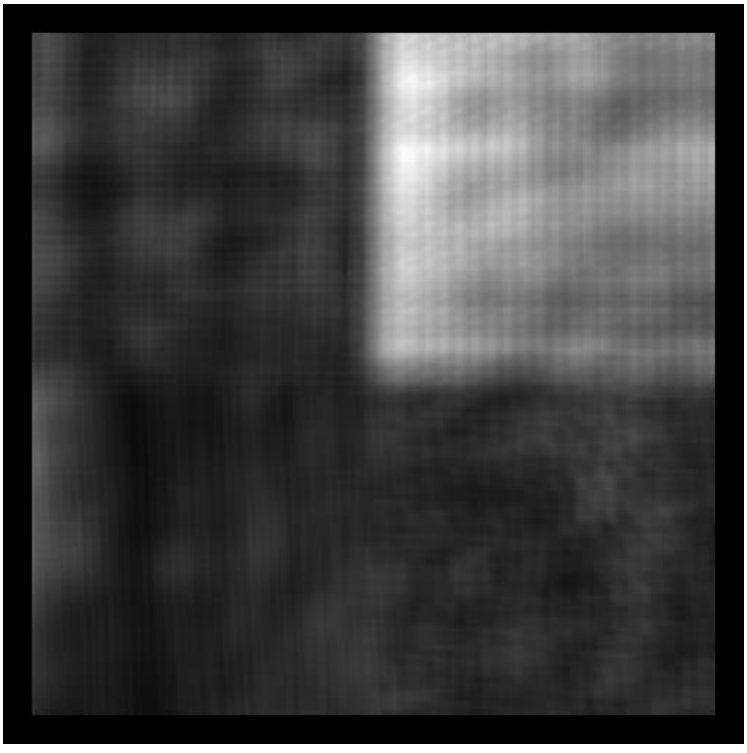


Fig 12: left: d=15 theta=120 using inertia from mosaic2. right: Mask from the feature-map trying to distinguish the textures in the upper left and lower right corner.

Here we try to distinguish again the upper left corner and the lower right corner, and according to the mask this is quite the good detection. We quite clearly see where the two similar types of textures lie, even though the detection is not perfect with holes and additional unwanted detection outside of the textures we wanted.



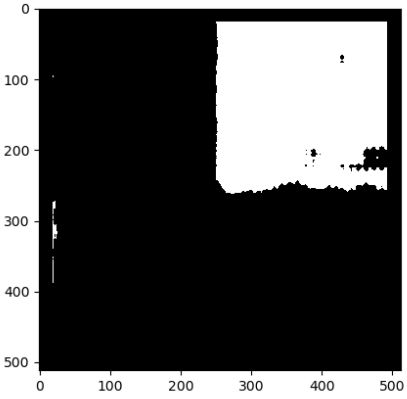


Fig 13: left: d=15 theta=120 using SHD from mosaic2. right: Mask from the feature-map trying to distinguish the textures in the upper right corner.

This is maybe the best detection. We try to find the upper right corner, and it does this very well. Only the texture we are after gets detected and the edges are quite good as well, with a few smaller holes.

Here we could also have combined different feature-maps to distinguish between textures that are very similar. This could be done by for instance adding two or three feature-maps togheter and then normalizing them. Say we have good feature-maps that can detect three regions in total, but one of the textures is always quite similar to the others, then we could add the feature-maps for singling out the three textures, so that we would be able to single out the one texture that wasn't easy to detect from one single feature-map.

Appendix

Test1.py

```
from imageio import imread, imwrite
import matplotlib.pyplot as plt
#from matplotlib.image import imread
import matplotlib.cm as cm
import numpy as np
import sys

#parameters that have proven to be quite effective
"""
#d=5, theta=0
#d=3, theta=315     =45 grader
#d=2, 4, 5, theta=270
#d=4 theta=60
"""

def make_histograms():
    """
    This function is not needed to solve the problem, it is only needed to
get out the histogram so that we can see that the histogram is not much of
help and that histogram transforms will not help much here
    """
    #making histograms
    fig, axs=plt.subplots(len(images))
    for i in range(len(images)):
        axs[i].hist(images[i])
    axs[len(images)-1].set(xlabel='pixel value', ylabel='number of
occurences')
    plt.show()

    #computing mean, variance, skewness, kurtosis
    meanlist=[]
    variancelist=[]
    skewnesslist=[]
    kurtosislist=[]
    for i in range(len(images)):
        image=images[i].ravel()
        mean=sum(image)/len(image)
        variance=sum((image-mean)**2)/len(image)
        skewness=sum((image-mean)**3)/(len(image)*np.sqrt(variance)**3)
        kurtosis=sum((image-mean)**4)/(len(image)*np.sqrt(variance)**4)

        meanlist.append(mean)
```

```
            variancelist.append(variance)
            skewnesslist.append(skewness)
            kurtosislist.append(kurtosis)
        print(meanlist, "mean")
        print(variancelist, "variance")
        print(skewnesslist, "skewness")
        print(kurtosislist, "kurtosis")
        print(1-(1/(1+np.asarray(variancelist))), "smoothness")

    def find_loop_indexes(image, theta_in):
        """
        This is a method that will return the correct indexes to be used in the
    for loop to calcualte the GLCM according to which theta-value one use
        """
        theta=(theta_in/360)*2*np.pi
        if theta_in>=0 and theta_in<=90:
            startindexi=0
            endindexi=image.shape[0]-int(np.cos(theta)*d)
            startindexj=0
            endindexj=image.shape[1]-int(np.sin(theta)*d)
        elif theta_in>90 and theta_in<=180:
            startindexi=abs(int(np.cos(theta)*d))
            endindexi=image.shape[0]
            startindexj=0
            endindexj=image.shape[1]-int(np.sin(theta)*d)
        elif theta_in>180 and theta_in<=270:
            startindexi=abs(int(np.cos(theta)*d))
            endindexi=image.shape[0]
            startindexj=abs(int(np.sin(theta)*d))
            endindexj=image.shape[1]
        elif theta_in>270 and theta_in<=360:
            startindexi=0
            endindexi=image.shape[0]-int(np.cos(theta)*d)
            startindexj=abs(int(np.sin(theta)*d))
            endindexj=image.shape[1]

        return startindexi, endindexi, startindexj, endindexj

    def calculate_and_save_GLCM(image, d, theta, plot=False, write=False,
    filename=None):
        """
        This method calculates the GLCM with the given parameters
        """
        theta_in=theta
        theta=(theta_in/360)*2*np.pi
        GLCM_matrix=np.zeros((num_pixval,num_pixval))
        #find the correct indexes in the for-loop according to the angle and
    distance given
        startindexi, endindexi, startindexj, endindexj=find_loop_indexes(image,
    theta_in)
        for i in range(startindexi, endindexi):
            for j in range(startindexj, endindexj):
                #find the value of the two chosen pixels and put them into the
    GLCM matrix
                pixel1=image[i,j]
                pixel2=image[i+int(np.cos(theta)*d),j+int(np.sin(theta)*d)]
                GLCM_matrix[pixel1,pixel2]+=1

        #make it symmetrical
        GLCM_matrix=(GLCM_matrix+GLCM_matrix.T)
        summ=np.sum(GLCM_matrix)
        #normalize it
        GLCM_matrix=GLCM_matrix/float(summ)

        maxi=np.max(GLCM_matrix)
        mini=np.min(GLCM_matrix)
        #made closer to the 0-255 value to make converting it to image more
    secure(less lossy)
        if write:
            GLCM_matrix_bits=((GLCM_matrix-mini)/(maxi-mini))*255
```

```
        streng=filename
        imwrite(streng,GLCM_matrix_bits)

    if plot:
        plt.imshow(GLCM_matrix)
        plt.show()     #show the plot of all the different GLCM matrices

    return GLCM_matrix


if __name__=="__main__":
    num_images=2
    num_textures=4      #per picture
    num=int(num_textures/2)
    pixels=512

    #reads in the images
    image1=imread("mosaic1.png")
    image2=imread("mosaic2.png")

    #puts the images into list
    imagelist=[image1, image2]

    #images will contain images of only one texture per image
    images=[]

    #divide the original images into the different textures and put them
into images list
    for i in range(num_images):
        image=imagelist[i]
        for j in range(num):
            for k in range(num):

images.append(image[int((pixels/num)*k):int((pixels/num)*(k+1)),int((pixels
/num)*j):int((pixels/num)*(j+1))])


    #rescale so that the images only contains 16 different values
    for i in range(len(images)):
        for j in range(images[i].shape[0]):
            for k in range(images[i].shape[1]):
                images[i][j][k]=int(images[i][j][k]/16)

    #setting some parameters
    num_pixval=16
    """
    make_histograms()
    """
    #taking in the parameters from user
    d=int(sys.argv[1])
    theta=int(sys.argv[2])
    write=bool(int(sys.argv[3]))
    plot=bool(int(sys.argv[4]))
    filename=sys.argv[5]

    fig, axs=plt.subplots(3,3)
    #used to get the subplots at different positions in the main plot
    in_i=0
    in_j=0
    for i in range(len(images)):
        image=images[i]
        GLCM_matrix=calculate_and_save_GLCM(image, d, theta, plot=plot,
write=write, filename=filename+str(i)+".png")
        axs[in_i, in_j].imshow(GLCM_matrix)
        if in_i>=2:
            in_i=0
            in_j+=1
        else:
            in_i+=1
    #save to file and show the plot of all the subplots
```

```
    streng=str(d)+str(theta)+".png"
    fig.colorbar(cm.ScalarMappable())
    plt.savefig(streng)
    plt.show()
```

Test2.py

```
from imageio import imread, imwrite
import matplotlib.pyplot as plt
#from matplotlib.image import imread
import matplotlib.cm as cm
import numpy as np
import sys

def calculate_GLCM(window, d, theta):
    """
    This method calculates the GLCM with the given parameters
    """

    theta_in=theta
    theta=(theta/360)*2*np.pi
    GLCM_matrix=np.zeros((16,16))
```

```
    #find the correct indexes in the for-loop according to the angle and
distance given
    if theta_in>=0 and theta_in<=90:
        startindexi=0
        endindexi=window.shape[0]-int(np.cos(theta)*d)
        startindexj=0
        endindexj=window.shape[1]-int(np.sin(theta)*d)
    elif theta_in>90 and theta_in<=180:
        startindexi=abs(int(np.cos(theta)*d))
        endindexi=window.shape[0]
        startindexj=0
        endindexj=window.shape[1]-int(np.sin(theta)*d)
    elif theta_in>180 and theta_in<=270:
        startindexi=abs(int(np.cos(theta)*d))
        endindexi=window.shape[0]
        startindexj=abs(int(np.sin(theta)*d))
        endindexj=window.shape[1]
    elif theta_in>270 and theta_in<=360:
        startindexi=0
        endindexi=window.shape[0]-int(np.cos(theta)*d)
        startindexj=abs(int(np.sin(theta)*d))
        endindexj=window.shape[1]


    for i in range(startindexi, endindexi):
        for j in range(startindexj, endindexj):
            #find the value of the two chosen pixels and put them into the
GLCM matrix
            pixel1=window[i,j]
            pixel2=window[i+int(np.cos(theta)*d),j+int(np.sin(theta)*d)]
            GLCM_matrix[pixel1,pixel2]+=1
    #make it symmetrical
    GLCM_matrix=(GLCM_matrix+GLCM_matrix.T)
    summ=np.sum(GLCM_matrix)
    #normalize it
    GLCM_matrix=GLCM_matrix/float(summ)
    return GLCM_matrix

def create_feature_map(image, window_size, d, theta):
    #this matrix will be filled with the feature-scalars
    feature_map=np.zeros((image.shape[0], image.shape[1]))

    #have to remove half of the window size at the edges of the image
    index=int(window_size/2)


    for i in range(index, image.shape[0]-index):
        for j in range(index, image.shape[1]-index):
            #create the window according to the window size
            window=image[i-index:i+index, j-index:j+index]
            #calcualte the GLCM
            GLCM=calculate_GLCM(window, d, theta)
            #set in the values to the feature-map
            feature_map[i,j]=IDM(GLCM)

    return feature_map


#implementing the IDM algorithm from a GLCM matrix
def IDM(GLCM):
    s=0
    for i in range(GLCM.shape[0]):
        for j in range(GLCM.shape[1]):
            s=s+(1.0/(1+(i-j)**2))*GLCM[i,j]
    s=s/(GLCM.shape[0]*GLCM.shape[1])
    return s

#implementing the inertia feature function from a GLCM-matrix
def inertia(GLCM):
    s=0
```

```
    for i in range(GLCM.shape[0]):
        for j in range(GLCM.shape[1]):
            s=s+((i-j)**2*GLCM[i,j])
    s=s/(GLCM.shape[0]*GLCM.shape[1])
    return s

#implementing the SHD algorithm from a GLCM-matrix
def SHD(GLCM):
    mux=0
    muy=0
    for i in range(GLCM.shape[0]):
        for j in range(GLCM.shape[1]):
            mux=mux+i*GLCM[i,j]
            muy=muy+j*GLCM[i,j]
    mux=mux/(GLCM.shape[0]*GLCM.shape[1])
    muy=muy/(GLCM.shape[0]*GLCM.shape[1])

    s=0
    for i in range(GLCM.shape[0]):
        for j in range(GLCM.shape[1]):
            s+=((i+j-mux-muy)**3)*GLCM[i,j]
    s=s/(GLCM.shape[0]*GLCM.shape[1])
    return s




if __name__=="__main__":
    #take in parameters from user
    filename=sys.argv[1]
    d=int(sys.argv[1])
    theta=int(sys.argv[2])
    filename=sys.argv[3]
    #read an image to do texture analysis
    image=imread("mosaic2.png")

    #convert the image it to 16 gray-values
    for j in range(image.shape[0]):
        for k in range(image.shape[1]):
            image[j][k]=int(image[j][k]/16)

    window_size=41

    feature_map=create_feature_map(image, window_size, d, theta)

    #scale the feature-map
    mini=np.min(feature_map)
    maxi=np.max(feature_map)
    feature_map=((feature_map-mini)/maxi)*255
    feature_map=feature_map.astype(np.uint8)

    #write the feature-map to file
    imwrite(filename, feature_map)
    """
    from playsound import playsound

    playsound("in5520\\Daft Punk - Instant Crush.mp3")
    """
    plt.imshow(feature_map)
    plt.show()
    """
    plt.hist(feature_map)
    plt.show()
    """
```

```
Test3.py
from imageio import imread, imwrite
import matplotlib.pyplot as plt
#from matplotlib.image import imread
import matplotlib.cm as cm
import numpy as np
import sys



"""
below in comments are filenames for feature-maps and the tested threshold
to give the best mask for the problem
"""
#mosaic 1
"""
threshold=182
```

```
filename="d2t20_feature_map_IDM.png"      #downer left
"""
"""
threshold=145
filename="d4t60_feature_map.png"         #upper right
"""
"""
threshold=95
filename="d4t50_SHD_ws_51.png"               #ish lower right and upper
left
"""


#mosaic 2
"""
threshold=95    #
filename="d15t120_SHD_41_mosaic2.png"   #upper right
"""
"""
threshold=115
filename="d15t120_inertia_41_mosaic2.png"        #upper left and downer
right
"""
"""
threshold1=70
threshold2=125
filename="d3t240_inertia_41_mosaic2.png"  #upper left
"""
"""
threshold=50
filename="d3t240_SHD_41_mosaic2.png"      #ish lower left corner
"""

feature_map=imread(filename)

#loop through the whole feature-map and find pixels  that have a pixel
value below threshold and set them to 0, and pixels above to 255
mask=np.zeros((feature_map.shape))
for i in range(feature_map.shape[0]):
    for j in range(feature_map.shape[1]):
        if feature_map[i,j]>threshold:
            mask[i,j]=255
        else:
            mask[i,j]=0
#show the mask
plt.imshow(mask, cmap=cm.gray)
#and save it to file
filename=filename[:-4]+"mask.png"
plt.savefig(filename)
plt.show()
```