IN5520 mandatory 1

Erlend Kristensen

11.09.2022

Analysis

To be able to segment a picture, a lot of work has to be put in to it. For example we need to be able to differ between foreground, background and different obstacles in the picture. A good way to segment is to look at the different textures in the picture. A texture can be described as consisting of texels, which is the characteristic object like bark on a tree. Here we can observe the color intensity properties of each texture and use this to segment the picture.

A way we can do this is to use so called Gray Level Occurence Matrices (GLCM). A GLCM is made by taking a region of the picture (or the whole picture), and counting every pair of gray-level intensity we have, and how many we have of each. This is done by choosing a distance d and a direction θ we want to count for. Then we can normalize this GLCM matrix into a so called coocurrance matrix, which gives the probability of changing from a gray level i to j.

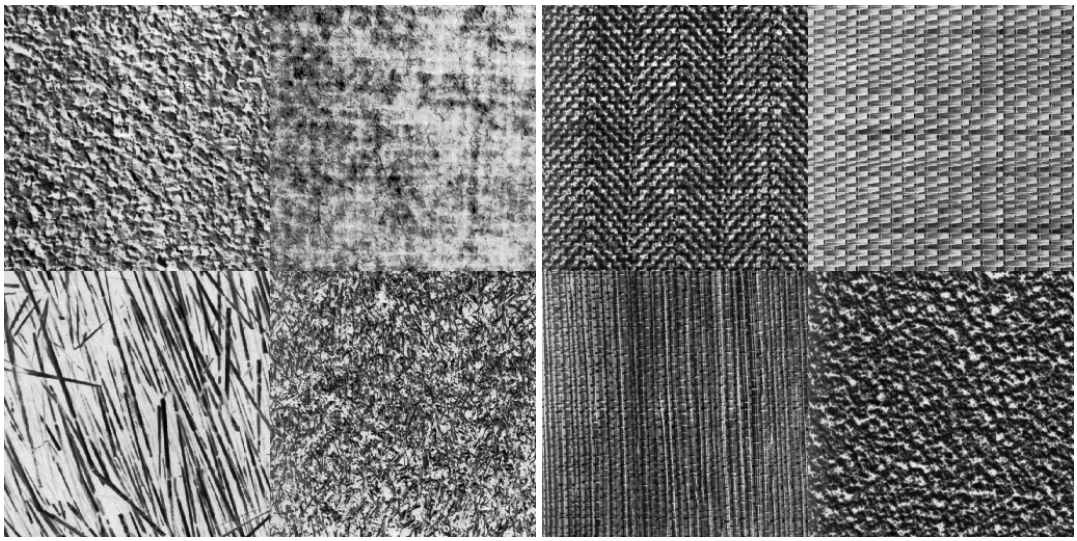
The algorithm:

From one window of size $w \times w$ get one GLCM matrix for a given $(d\theta)$, where the dimensions of the GLCM is $G \times G$ if we have G gray-levels in the image.

We then choose a distance d and a direction θ, and check all pixel pairs with this distance and direction inside the window, and sum these up, so $Q(i, j)$ is the amount of pixels where the first pixel has value 1 and the second pixel has value j.

We could also use something called an Isotropic GLCM, which calculates the four different directions in 180 degrees of the picture, and takes the average of the four.

We have eight different textures we want to analyse. They are shown bellow:



I will denote the textures by number ranging from 1-8, going from top left and rightwards to bottom right texture, or 1-4 further in the repport when talking about one picture at the time.

We can observe that the texture direction of texture 1,2,6,8 are somewhat similar, where there is no strict direction the texture or texels are facing.

We also observe texture 3 having a downright and downleft facing texture, changing between the two directions as we move along the texture.
Texture 5 has a downward, slightly right facing texture, and texture 4 and 7 are simply one directional, where 4 goes to the right and 7 downwards.

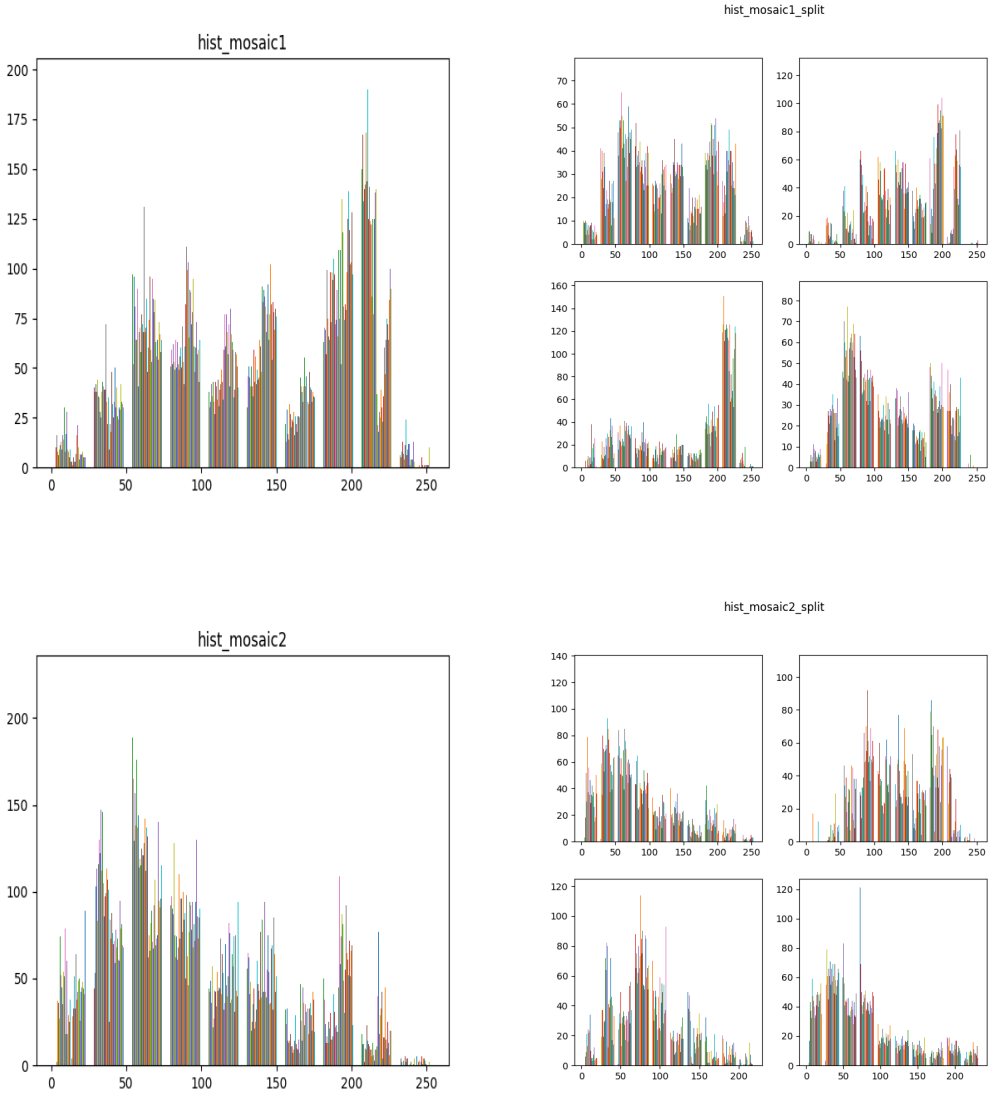
The variance will be greater the more difference there is between the gray levels in the textures. Textures 3,4,7 and 8 have a more dominant gray level, while textures 1,2,6 vary some more. Texture 5 has the highest variance, since it does not have many gray levels, but the few gray levels are very far from each other.

The texture sizes range from 6,7,3,1,8,2,4,5, where texels on 6 being smallest and 5 being largest.

Textures 4 and 7 are the ones with the most homogenity, while texture 3 has homogenity as the pattern repeats. Textures 1,6,8 has some homogenity, as they have a texture that is similar over the whole area. Textures 2 and 5 have the least homogenity, since there is no clear pattern that follows, even though the textures are somewhat the same across the picture.

Visualizing GLCM matrises

For this step, we first split the image into subimages for each texture. When we do this, the histograms corresponding to each subpicture will differ from the histogram of the main picture as seen bellow:

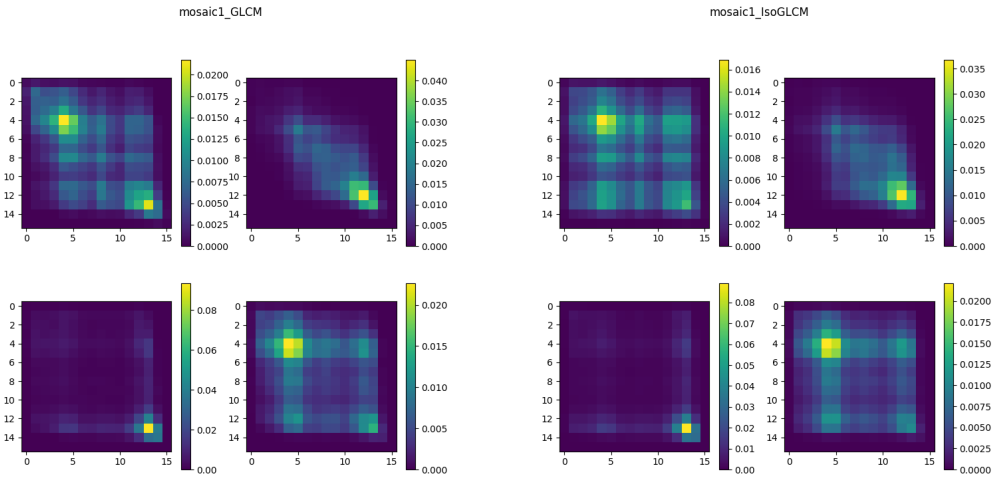


The textures in the split histogram images are the same order as shown in the picture above. Here we can see results as we expected, such that picture 3 has very few low-intensity gray tones, but a lot of high-intensity ones, which differs greatly from the main pictures histogram, which has a lot of diversity.

We now want to use GLCM for the subimages to start analyzing them. To avoid testing for a lot of parameters, we use the information we already got by alanyzing the different textures to find out what would be good parameters. To get a better outcome, we requantize the image, reducing the number of grey levels to 16.

In picture 1 we see that the textures differ quite a bit, but they don't have a clear direction. The only challange here will be textures 1 and 4 (or 1 and 6 as referenced earlier), since they are quite similar. As we will see later, its very hard for our simple algorithm to differ between these, so it won't matter much if we use Isotropic GLCM or directional GLCM. Since the texels are small, i chose a distance $d = 2$, and $\theta = 180$ seems to be a good direction.

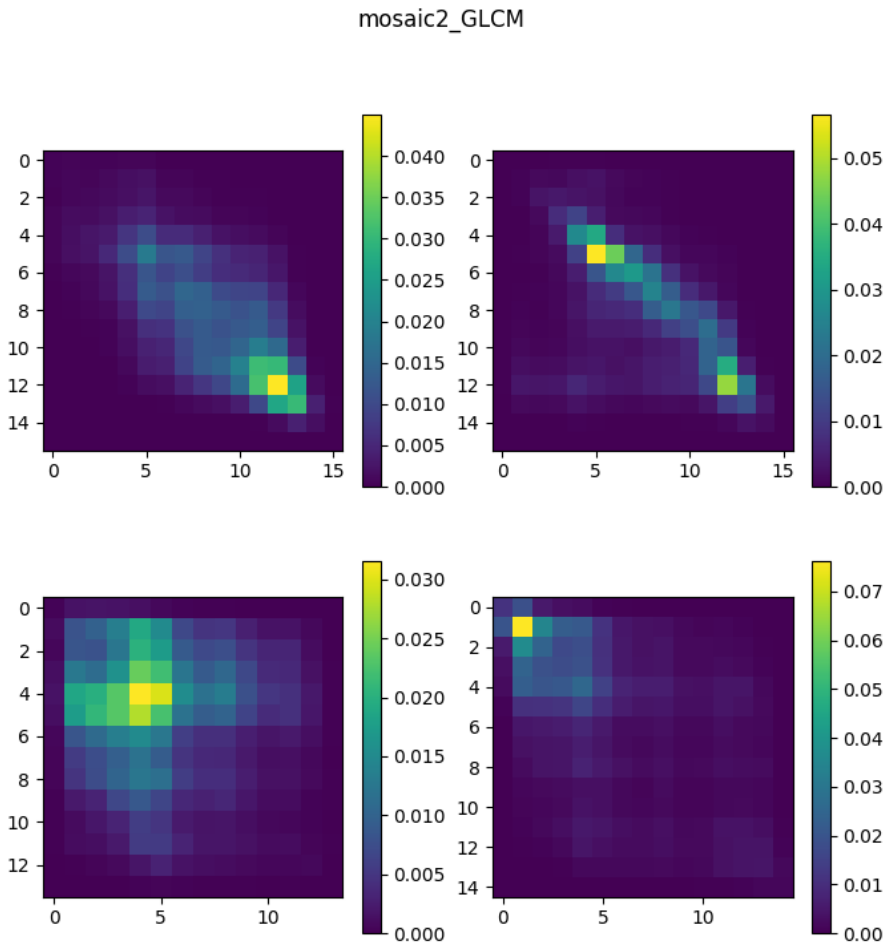
We get the GLCMs:



As we can see, theres not much of a difference between isotropic and

directional, except directional seperates picture 1 and 4 a bit better, so we choose this.
When looking at the GLCM we see what we expected. Picture 3 has a lot of the high-intensity gray tones which we saw in the histogram, while picture 1 and 4 are a lot more noisy and they have a quite similar GLCM.

In picture 2 (mosaic2) we can observe that each texture is very differently directional. Here we can just use a simple directional GLCM as used above. Since each texture differs a lot in the directions, it is easier to seperate between them, and we could just use the same as above so we have GLCM parameters that work for all textures. We get the GLCM:



As we expected, these parameters work well for seperating the specific textures. We also observe texture 4 here having a kinda similar GLCM to texture 1 and 4 in picture above. We also see that the textures are easily seperable, as we expected from our earlier analysis.

Computing GLCM feature images

We now want to use GLCM features to better seperate between the textures. We do this by gliding a window matrix over the image, and calculating the GLCM for each window. We then calculate the feature for each of these GLCMs, and set the pixel value as that feature.

The features we will use:

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

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

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

$Where:$

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

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

Inverse difference momentum (IDM) is influenced by the homogenity in the image, and will therefore give higher values the more homogeneous an image is.

Inertia (INR) weighs contrast, and will therefore give higher values for images with high local contrast.

Cluster Shade (SHD) measures skewness of the matrix. So it measures if there is a wider range of darker or lighter pixels.

We now have to decide on a fitting window size to use for the features. If we choose a large window size, we get precise estimation of features, but errors near edges and an imprecise estimation of location, while a small window size will give the opposite.
To be able to segment well, we chose a window size of 31. This is so we can get a good segmentation, both in finding the objects and retaining the edges as good as possible.

The resulting feature images:

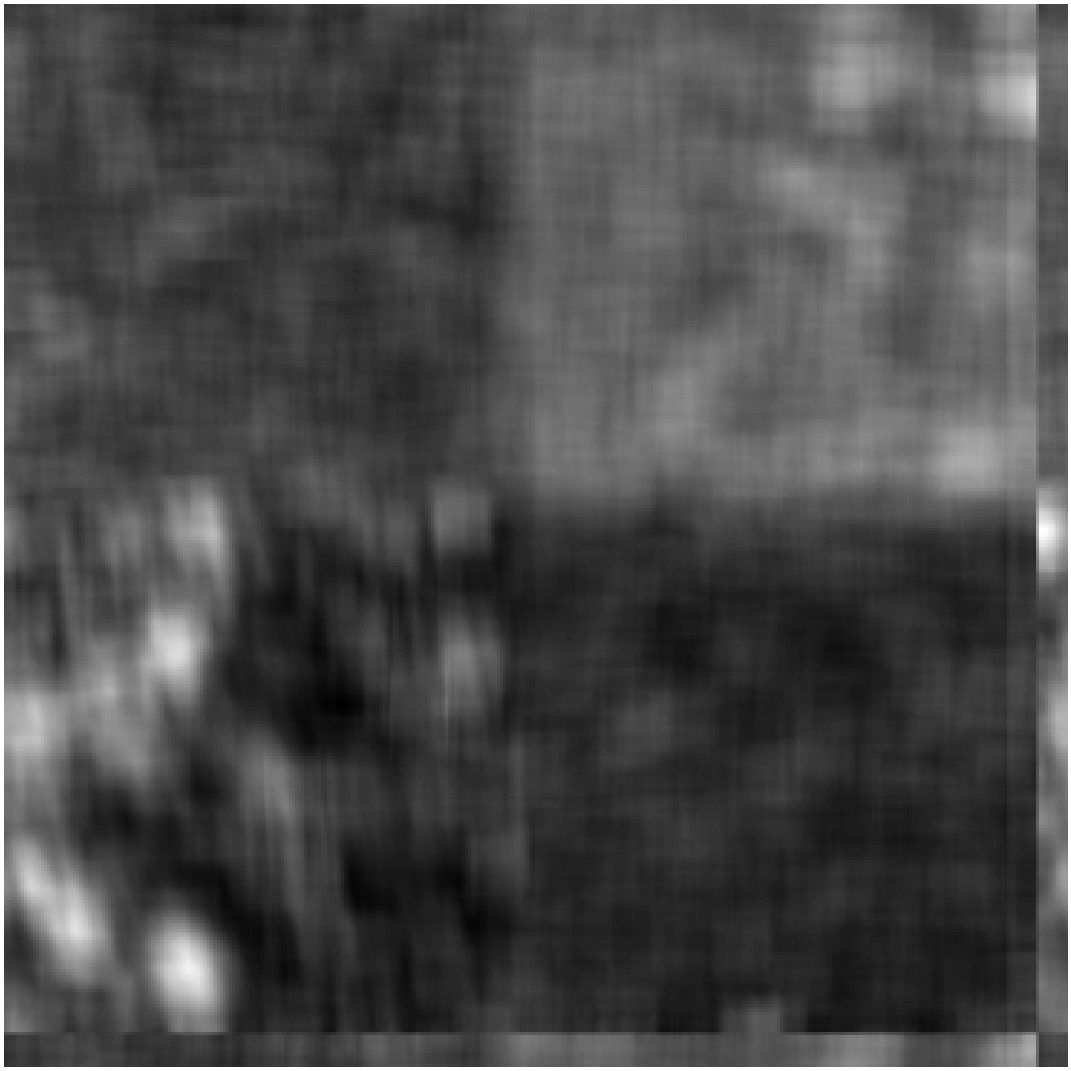


Figure 1: mosaic1 IDM

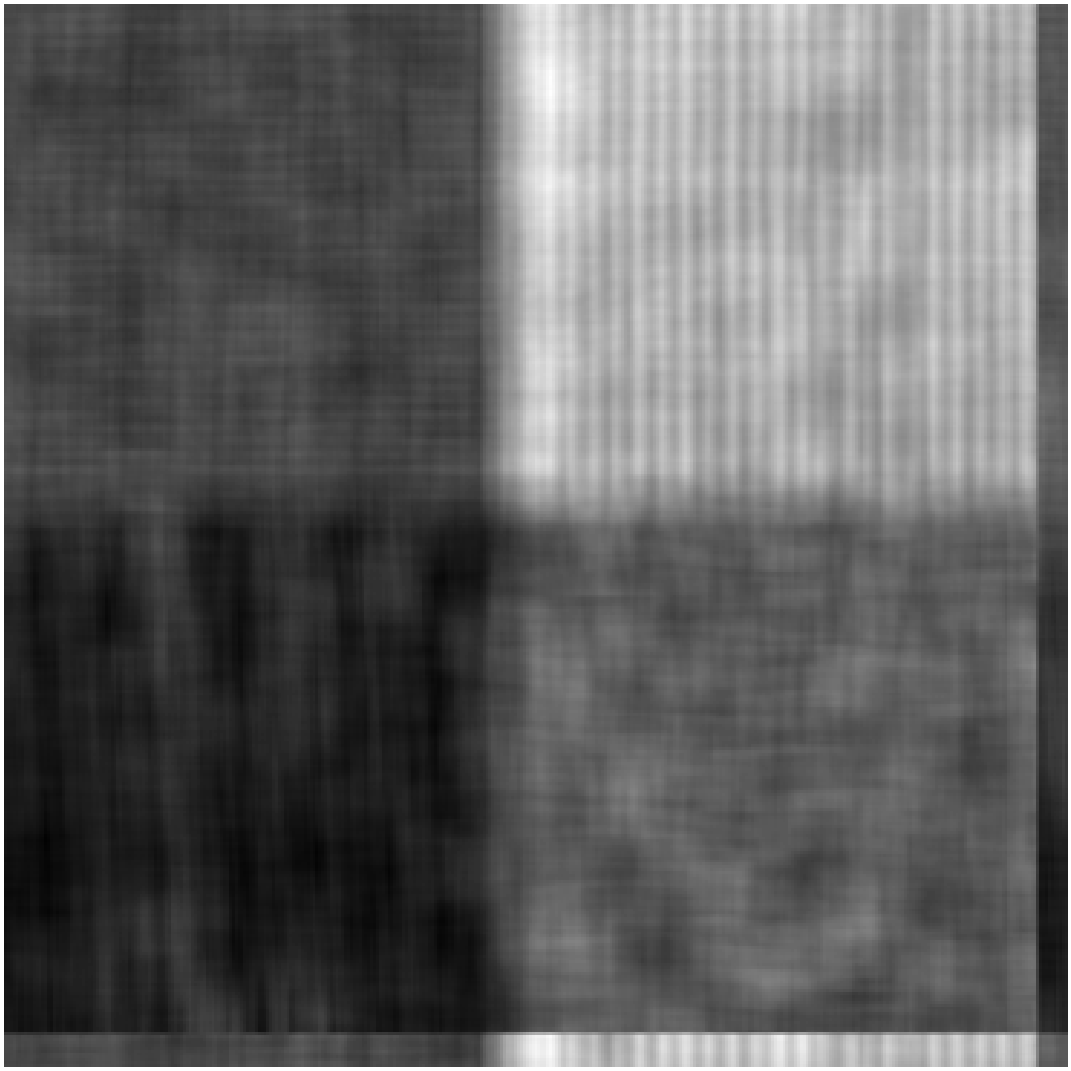


Figure 2: mosaic2 IDM

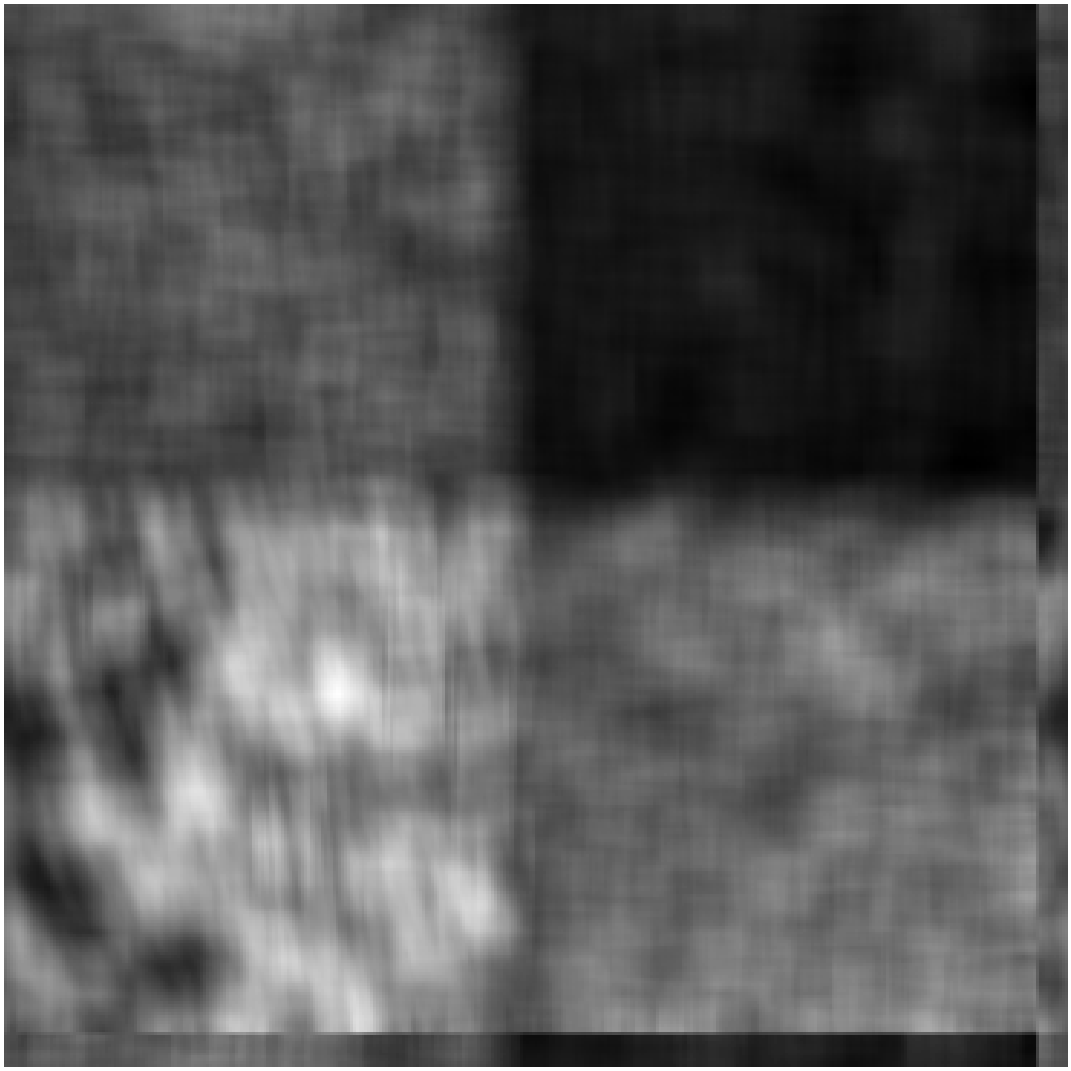


Figure 3: mosaic1 Inertia

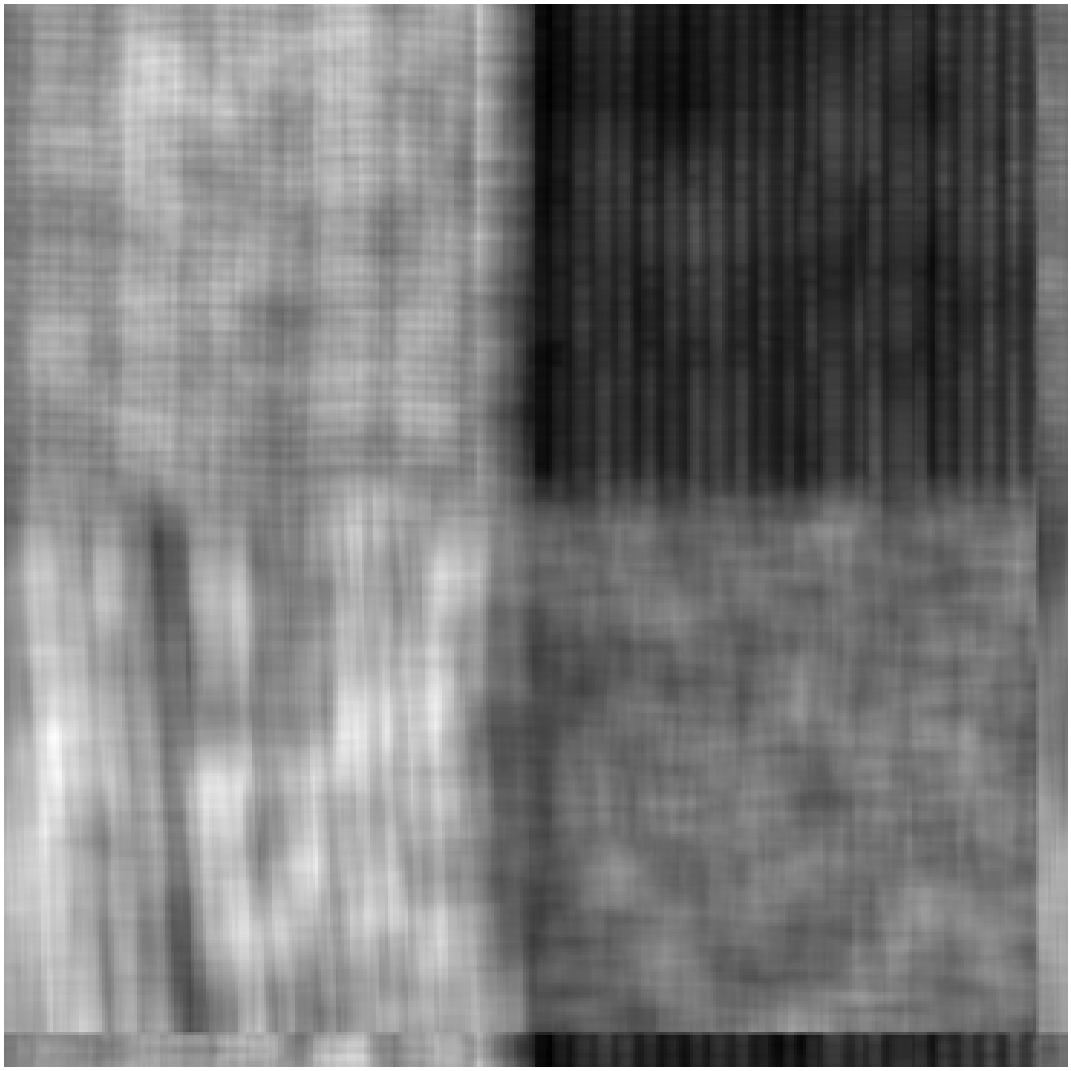


Figure 4: mosaic2 Inertia

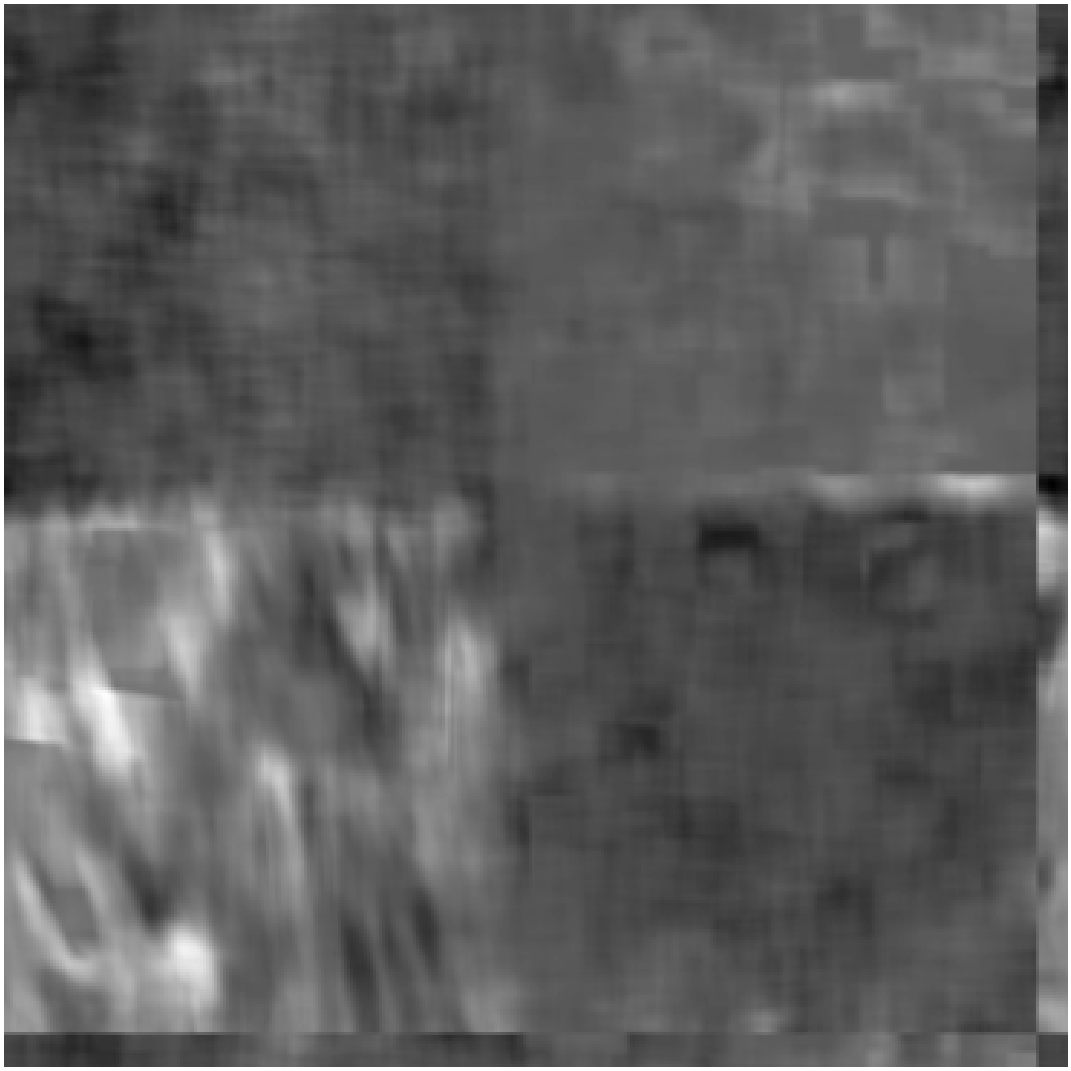


Figure 5: mosaic1 Cluster Shade

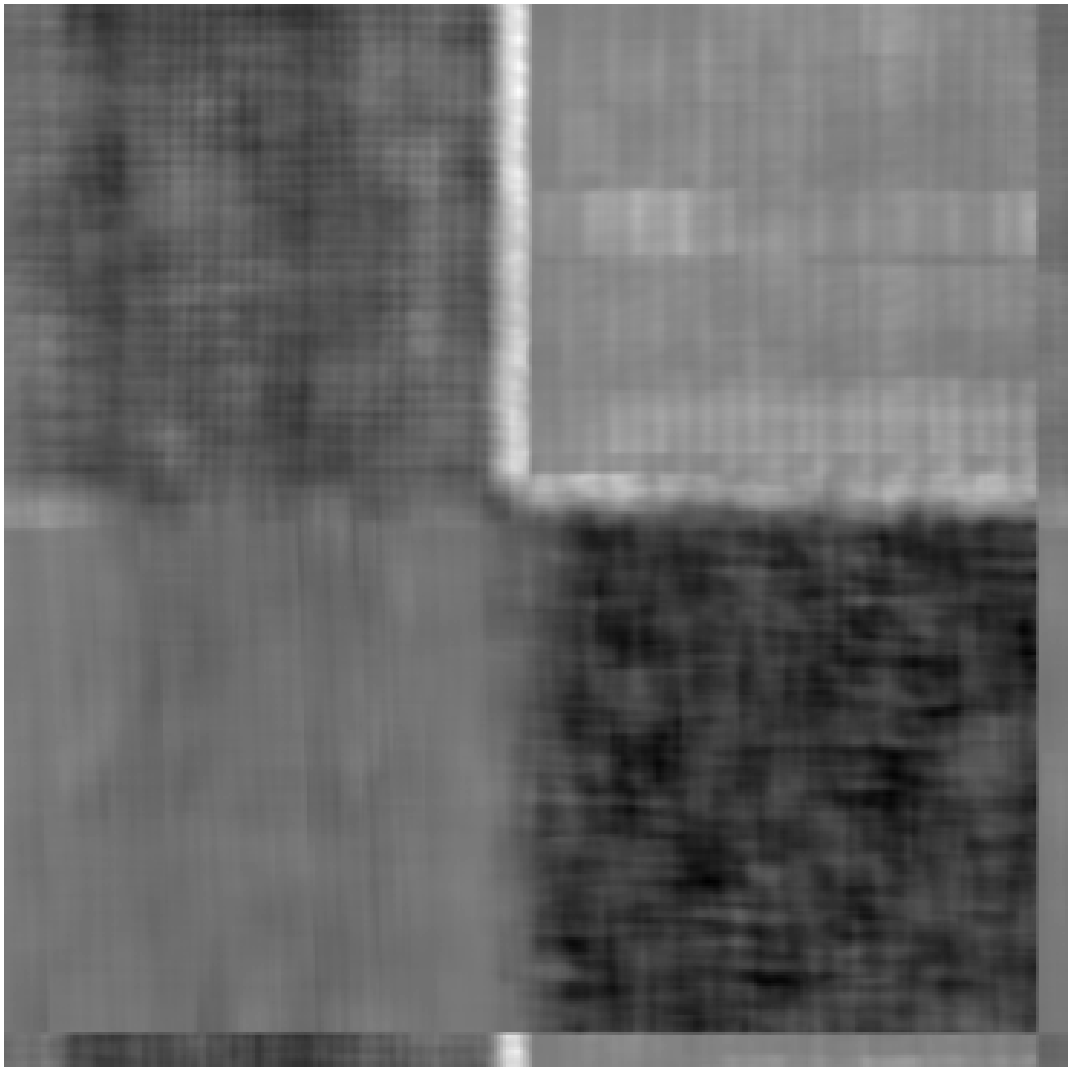


Figure 6: mosaic2 Cluster Shade

Note: I see some obscurity near the right and bottom edges of the picture. This is probably due to some bad coding, but i was not able to figure out what went wrong. This will not hinder the following results.

As we expected, we can see a bit of obscurity near the edges of the textures, but it is not very noticable. We also see that non of the features are really able to seperate well between textures 1 and 4 in mosaic 1.

We also see that IDM struggles a bit more on mosaic1. This is because mosaic2 has textures with different homogenity, while mosaic1 has textures that are a bit more similar.

Inertia does a better job on mosaic1 and 2, but since inertia weighs contrast, some of the textures with similar contrast, like texture 1 and 4 in both images are not that easily seperable.

Cluster shade however, has an easier time seperating between texture 1 and 4 in mosaic2 since these textures vary a lot between darker or lighter pixels, however its not enough to seperate well.

So we can see some of the textures are easy to pick up and seperate from the rest, while some will be a bit of a problem.

Segmenting

We now want to segment the textures by applying a global threshold to the GLCM feature matrices. To do this, we find a threshold that could work on each feature. Here, we just have to test out to find a fitting threshold.
Here are the segmentations i found that worked best:

For Mosaic1 picture:

Figure 7: mosaic1 Inertia feature picture segmented with treshold 0-40

Figure 8: mosaic1 Inertia feature picture segmented with treshold 130-255

Figure 9: mosaic1 IDM feature picture segmented with treshold 85-130

For Mosaic2 picture:

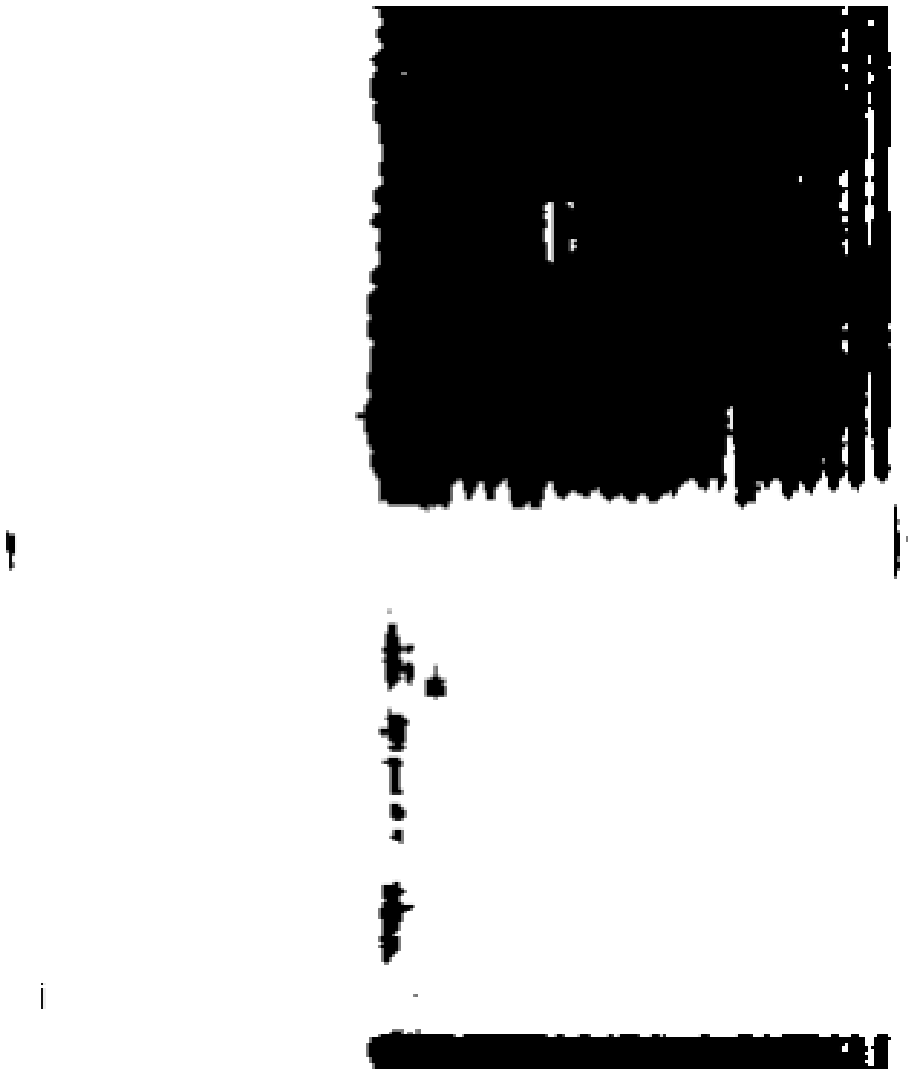


Figure 10: mosaic2 Inertia feature picture segmented with treshold 0-75

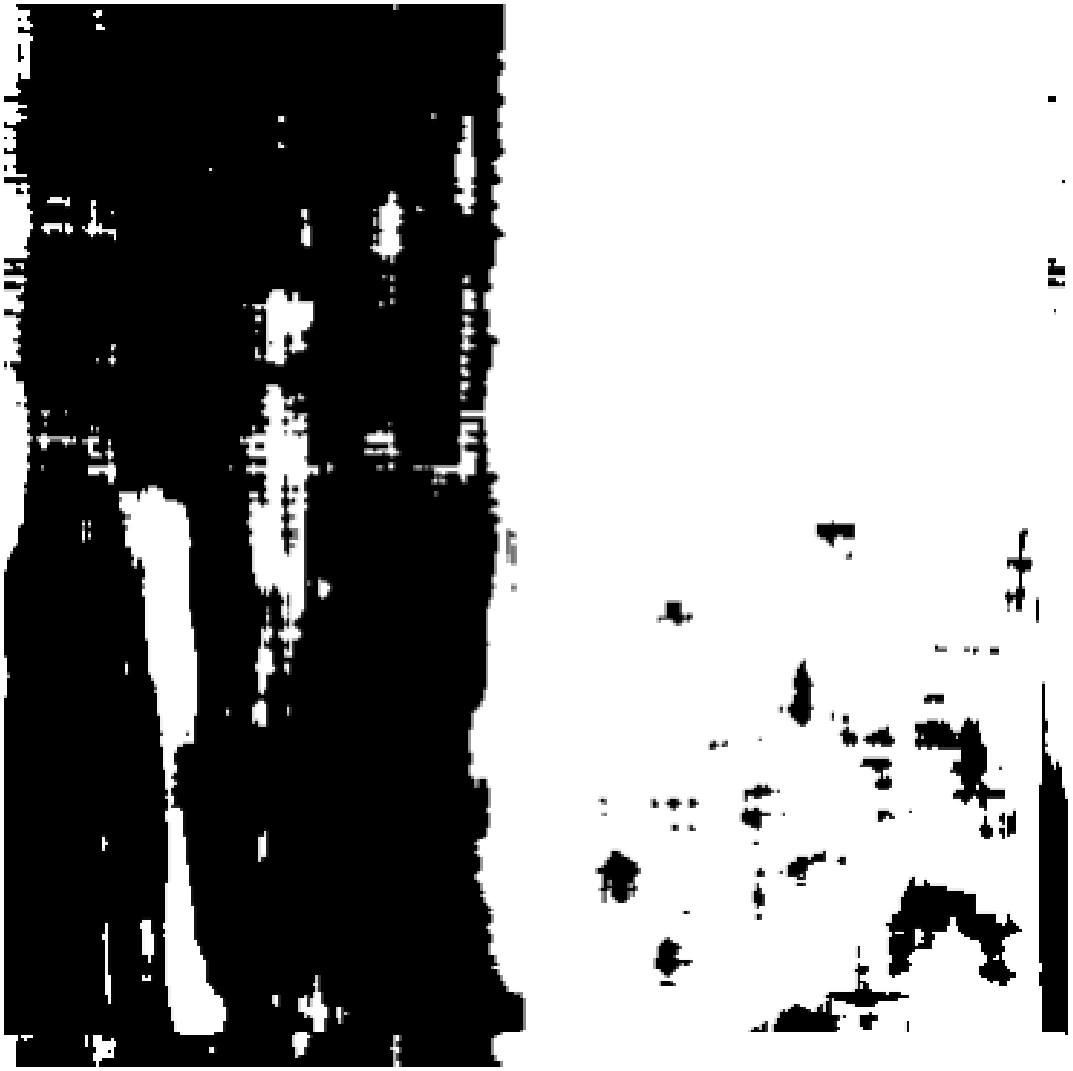


Figure 11: mosaic2 Inertia feature picture segmented with treshold 130-255

Figure 12: mosaic2 IDM feature picture segmented with treshold 130-255

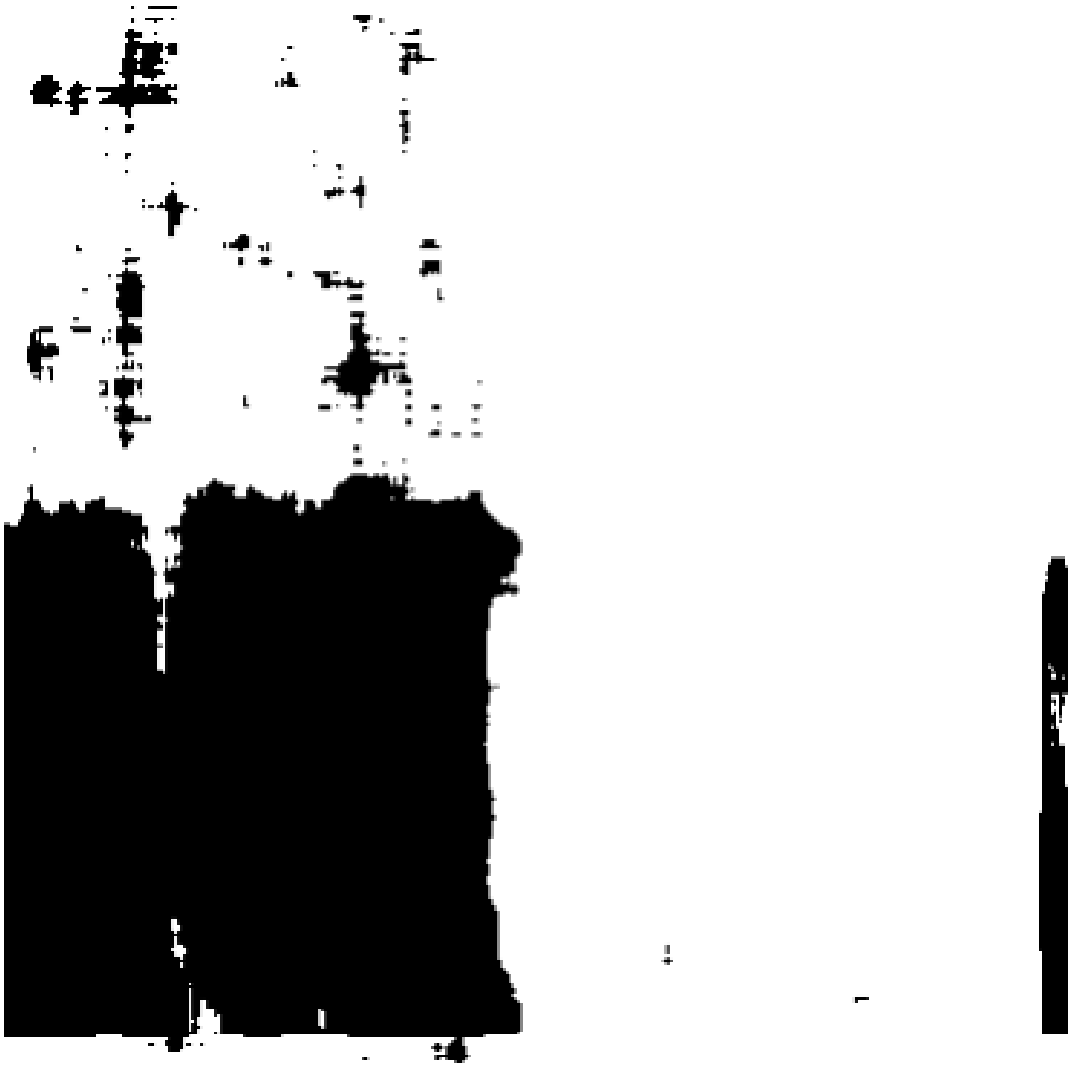


Figure 13: mosaic2 IDM feature picture segmented with treshold 0-60

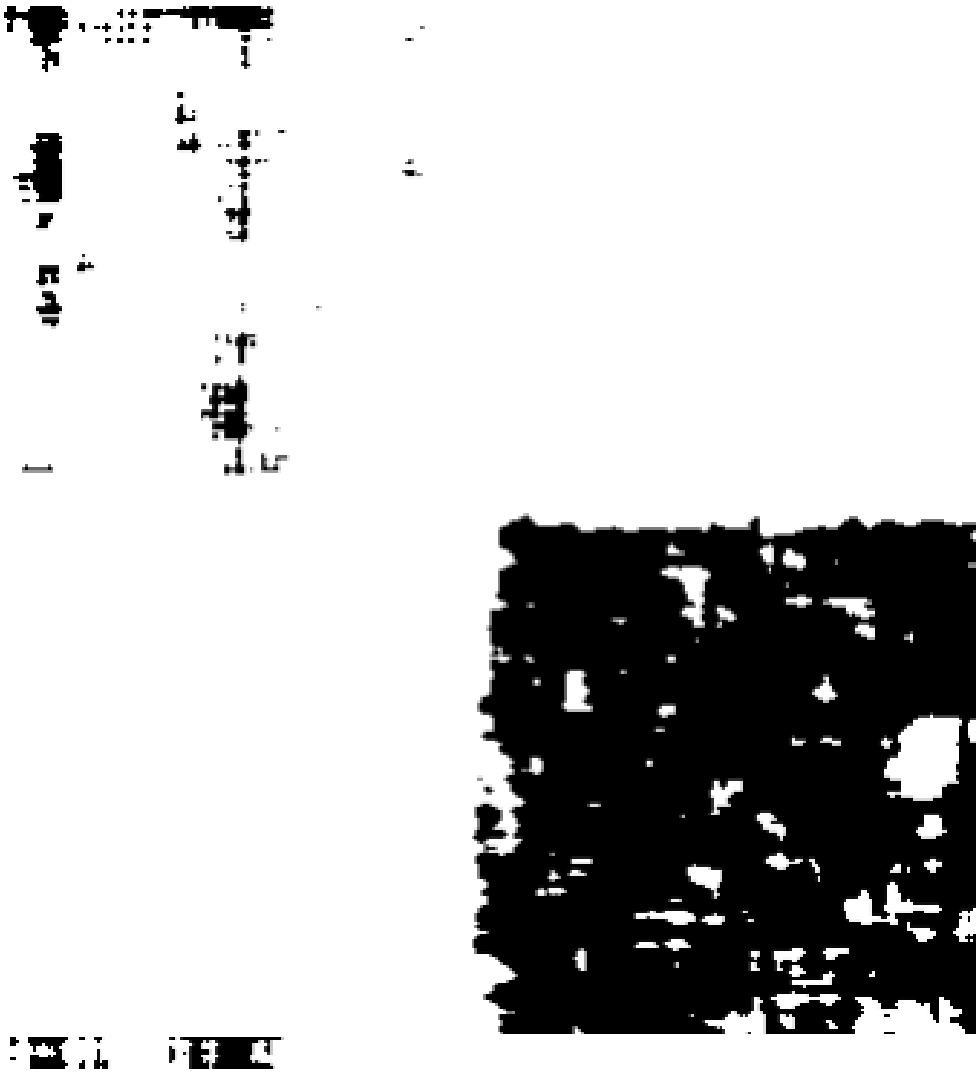


Figure 14: mosaic2 Cluster Shade feature picture segmented with treshold 0-70

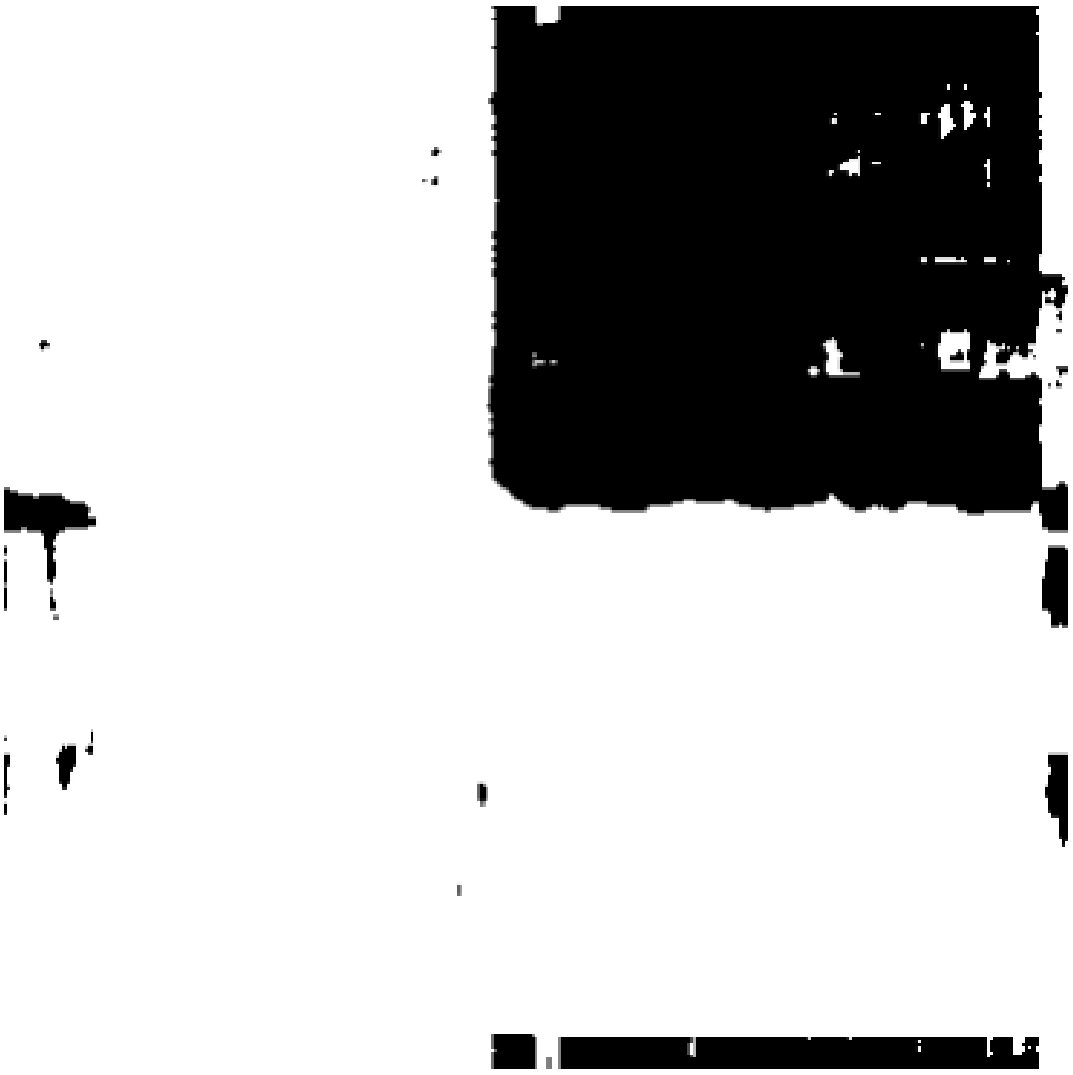


Figure 15: mosaic2 Cluster Shade feature picture segmented with treshold 0-70

As we can see, mosaic2 is easily segmentable. We were able to isolate every single texture except top left (number 1), however, this can be done by combining the thresholds of the others. This means we are able to detect

and segment between each of the textures.

In mosaic1 however, we are only able to really segment top right well, and some on bottom left. We are not able to seperate between texture 1 and 4, which we anticipated.

We can see the overall best result in figure 12, which was IDM used om mosaic2. This really picked up top right texture well.

Conclusion and reflection

All in all our GLCM feature transformation proved that we can in fact seperate between some textures and therefore segment objects in a given picture. Since the parameters for both the GLCM matrices and the feature matrices were found manually, there is a huge room for improvment, which could have resulted in better and more possible segmentations. A larger window size would also allow us to seperate better between the textures, but would give more errors near the edges.

Appendix

The important code used for this report: GLCM and Isotropic GLCM:

```
#Function for making GLCM
def GLCM(im, dist=1, degree=90, L=16):
    #requantize
    if L != None:
        #im = Image.fromarray(im)
        #im = im.quantize(L, method=0)
```

```
    #im = np.array(im)
    im = np.floor(im * (L/255)) * (255/L)

i_jump = 0
j_jump = 0

if degree == 0:
    j_jump = -1
if degree == 45:
    i_jump = 1
    j_jump = -1
if degree == 90:
    i_jump = 1
if degree == 135:
    i_jump = 1
    j_jump = 1
if degree == 180:
    j_jump = 1
if degree == 225:
    i_jump = 1
    i_jump = -1
if degree == 270:
    i_jump = -1
if degree == 315:
    i_jump = -1
    j_jump = -1

i_start = 1 if i_jump == -1 else 0
i_end = -1 if i_jump == 1 else 0
j_start = 1 if j_jump == -1 else 0
j_end = -1 if j_jump == 1 else 0

gray_levels = np.unique(im)
```

```
    map1 = {}
    map2 = {}
    for i in range(len(gray_levels)):
        map1[i] = gray_levels[i]
        map2[gray_levels[i]] = i

    num_gray_levels = len(gray_levels)

    Q = np.zeros((num_gray_levels,num_gray_levels))

    for i in range(i_start*dist, len(im) + i_end*dist):
        for j in range(j_start*dist, len(im[0]) + j_end*dist):
            val1 = im[i,j]
            val2 = im[i+i_jump*dist,j+j_jump*dist]
            Q[map2[val1],map2[val2]] += 1
    P = Q/(sum(sum(Q)))
    return Q,P

#Function for making Isotropic GLCM
def Isotropic_GLCM(im, dist, L):
    _,up = GLCM(im, dist=dist, degree = 0, L=L)
    _,up_right = GLCM(im, dist=dist, degree = 45, L=L)
    _,right = GLCM(im, dist=dist, degree = 90, L=L)
    _,down_right = GLCM(im, dist=dist, degree = 135, L=L)

    return (1/4)*(up + up_right + right + down_right)
```

The feature matrix implementations:

```
#Functions bellow are implementations of IDM, Inertia and Cluster Shade
def IDM(P):
    sum = 0
    for i in range(len(P)):
        for j in range(len(P[0])):
```

```
            sum += P[i,j]/(1 + (i-j)**2)
    return sum

def Inertia(P):
    sum = 0
    for i in range(len(P)):
        for j in range(len(P[0])):
            sum += P[i,j]*((i-j)**2)
    return sum

def Cluster_Shade(P):
    my_i = 0
    my_j = 0
    temp_sum_i = 0

    for i in range(len(P)):
        temp_sum_i = 0
        for j in range(len(P[0])):
            my_j += j * P[i,j]
            temp_sum_i += P[i,j]
        my_i += i*temp_sum_i

    sum = 0
    for i in range(len(P)):
        for j in range(len(P[0])):
            sum += (i + j - my_i - my_j)**3 * P[i,j]

    return sum
```

Method for calculating feature matrices:

```
#Function for creating a feature matrix
def feature_matrix(image, size, L=16 ,iso=False, function=IDM, d = 1, theta = 90):
```

```
    if L != None:
        image = Image.fromarray(image)
        image = image.quantize(L)
        image = np.array(image)
    out = np.ones_like(image, float)
    pad = int(size/2)
    image = np.pad(image, pad, 'reflect')

    for i in range(pad, len(image)-pad):
        for j in range(pad, len(image[0])-pad):
            if iso:
                P = Isotropic_GLCM(image[i-pad:i+pad,j-pad:j+pad], d, None)
            else:
                Q,P = GLCM(image[i-pad:i+pad,j-pad:j+pad], d, theta, L=None)
            feature = function(P)
            out[i-size,j-size] = feature
    return out
```