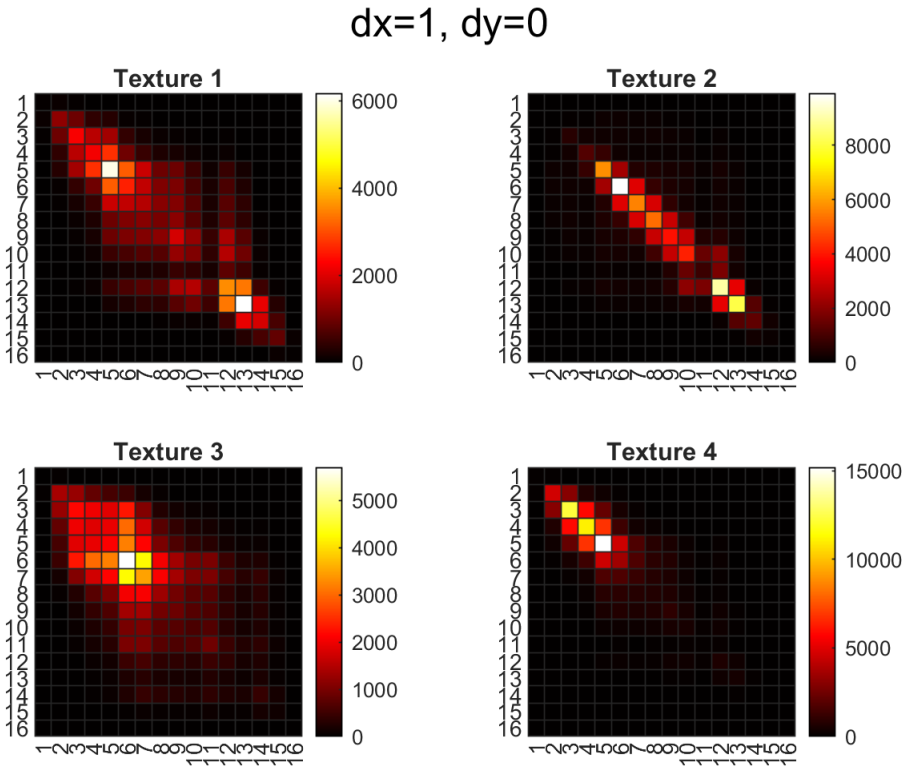
IN5520 Mandatory 2

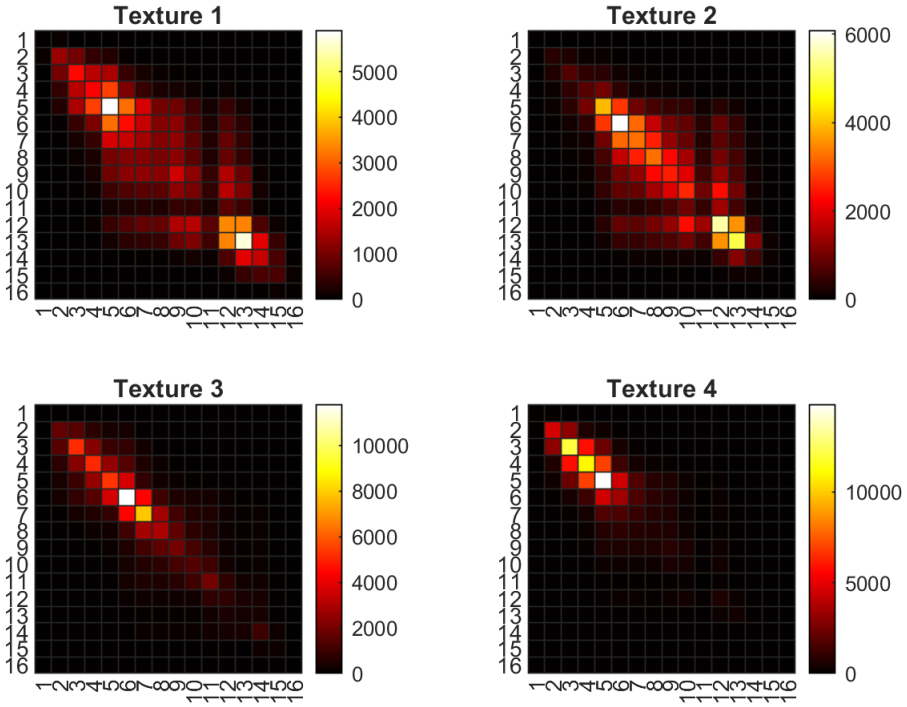
Espen Lønes

November 12, 2021

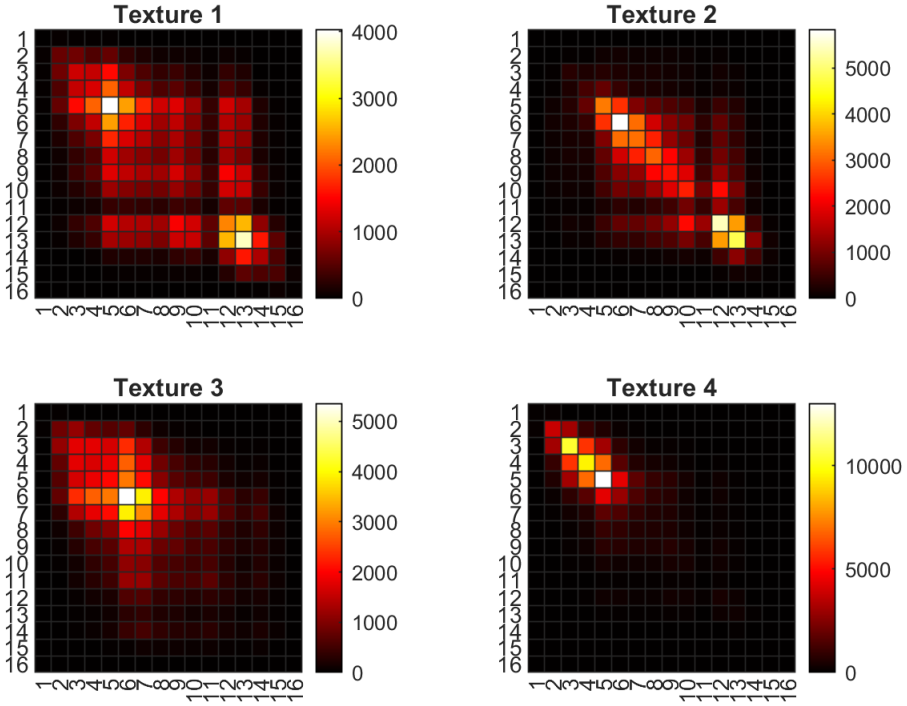
Task 1)
Plots of the glcms of the 4 textures for each direction:

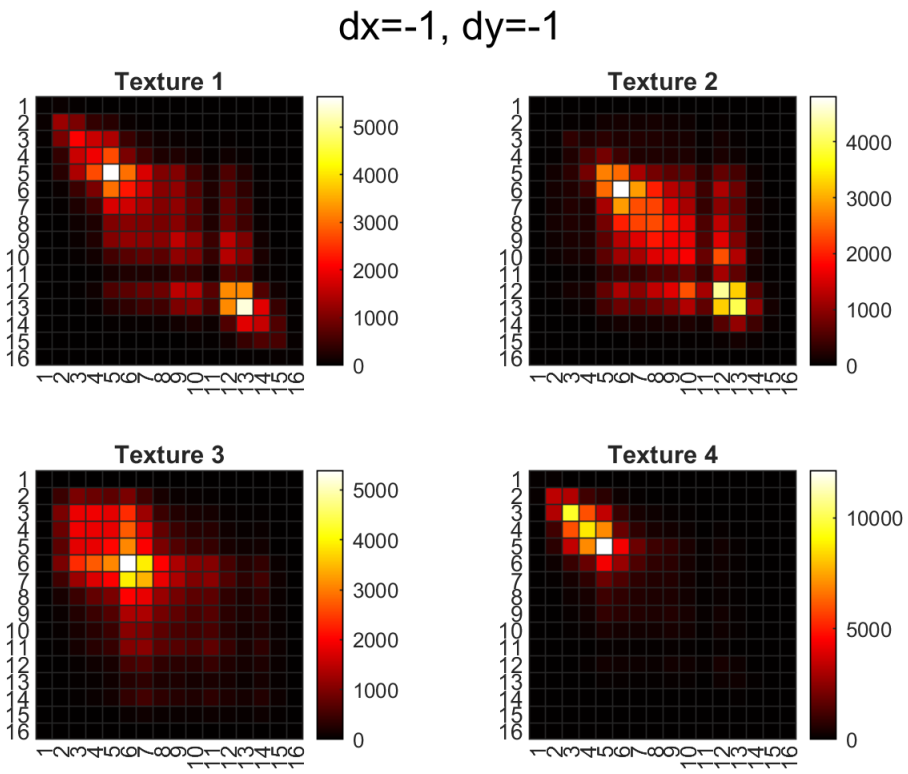


dx=0, dy=-1



dx=1, dy=-1





I don't think one direction will be enough to discriminate between all the textures. Because for all the directions at least two of the textures have very similar glcms.
Therefore i will chose two directions. As for which directions, i think dx=1,dy=0 and dx=0,dy=-1 will give a good chance for differentiating the textures.
Because each pair of textures are different in at least one direction.
(for convenience i will from now on call dx=1,dy=0 for d1. And dx=0,dy=-1 for d2)
e.g. texture 1 and 4 are different in both d1 and d2. But texture 1 and 2 are different in d1 but quite similar in d2.
Also texture 2 and 3 are quite different from themselves when comparing d1 and d2.
These differences between d1 and d2 should give us a lot of good information to give the classifier.

Task 2)

Based on the glcms produced by d1 and d2. We see that texture 1 and 2 has most of their energy split between Q1 and Q4, for d1 and d2.
While texture 3 and 4 has most of theirs in just Q1, again for d1 and d2.
It may therefore be beneficial to subdivide Q1 into 4 new sub quadrants.
I will call these Q5, Q6, Q7 and Q8 each being a quadrant of Q1. Q5 being in the top left and then going in reading direction. (see 'computequadrants.m' if unclear).

Furthermore, texture 1 has some energy in Q2 and Q3 for both d1 and d2.
As for texture 2, is also has some energy in Q2 and Q3 for d2. But for d1 it has almost no energy in Q2 and Q3.

A similar relation is also true for textures 3 and 4. Therefore Q2 and Q3 may be useful in separating 1 from 2 and 3 from 4.
Texture 3 has some energy in Q2, Q3 and Q4 (still most in Q1) for d1. And also a tiny amount for d2 as well. Texture 4 on the other hand appears to have no energy at all for Q2, Q3 and Q4 for both d1 and q2. Thus Q4 may also be useful for separation between 3 and 4. We also see that Q5 should be good for separating 4 from the rest.

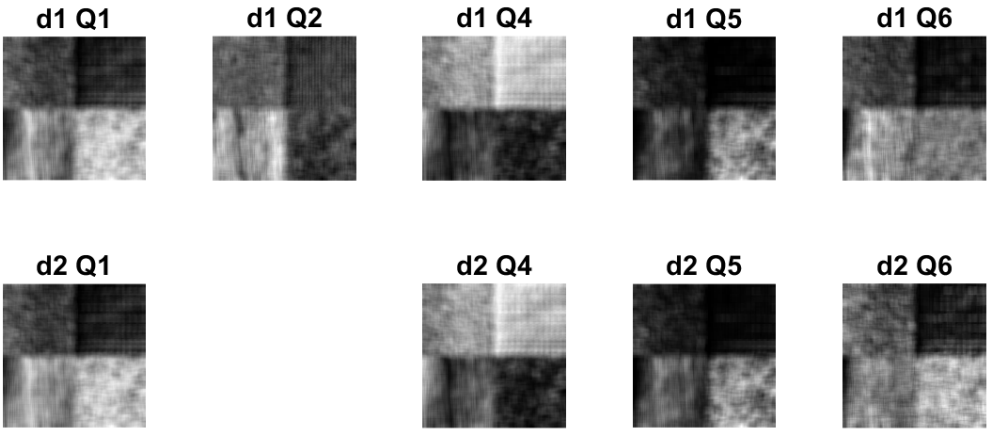
Therefore, we can use Q1 and Q4 for d1 and d2 (+4 quadrants total), to separate texture 1 and 2 from texture 3 and 4.
Then we may use Q2 and Q3 for d1 (+2 quadrant total) to separate texture 1 from 2.
We may then use Q2, Q3 and Q4 for d1, and possibly Q4 for d2
(+0 quadrants total as all are already used) to separate texture 3 from 4.
We can also make the observation that since the glcms are by inherently symmetric, Q2 and Q3 for the same angle will always be equal. We can therfore choose to ignore all Q3 values an use just Q2.
Q5 looks to be useful for separating out texture 4.
Q6 looks like it would bee good for separation as well.

We should therefore look closer at, Q1, Q2, Q4, Q5, Q6 for d1, and Q1, Q4, Q5, Q6 for d2. As features for discriminating between all 4 textures.
We probably wont need all of these for separation between all the textures.

Task 3)

Se 'task3.m' and 'computequadrants.m' for how glcms and quadrants are computed, as well as plotting of the feature images. If you wish to run this code you can skip the computation of the glcms by running the load glcms part instead.

The following feature images was produced:



We see there is strong correlation for anny Q between d1 and d2.
My thought of needing more than one direction was therefore incorrect.
From now on we will therefore only use d1.

Q1 and Q4 looks to be negatively correlated, but good at separating so we will use Q1.
Q2, Q5 and Q5 are also a good separators and fairly independent so they will be used as well.
We thus have the featurevetor: $\mathbb{Q} = [Q1, Q2, Q5, Q6]$.

Taks 4)

Code is below:

First i extract the index values for the different classes in the training mask. I then create feature matrices by using the masking indexes on the relevant feature images.
Once i have these feature matrices i can calculate their (the features) mean value and covariance matrix for each class. This is essentially the training of the classifier. I then create a new set of feature matrices, but these are from entire feature images for the features we use.
I then for each sample/pixel with its chosen features calculate its 'probability' (not real probability but it doesn't mater) and the prediction with the highest value is chosen as the classification of that pixel.

The predictions are done with Gaussian classifier using byes rule.

$$P(y = c|x) = \frac{P(x|y = c)P(y = c)}{p(x)}$$

For our problem $p(x)$ is equal for all classes so it is ignored.
So is $P(y = c)$ since it can also be assumed equal for all classes.
Then the posterior is just equal the class-conditional density.
Which itself is given by:

$$N(x|y = c) = N(x|\mu_c, \Sigma_c) = \frac{1}{(2\pi)^{p/2}|\Sigma_c|^{1/2}} exp[-\frac{1}{2}(x - \mu_c)^T \Sigma_c^{-1}(x - \mu_c)]$$

Which is implemented in the predict function (located in the 'predict.m' file)

```
function prediction = predict(x, cov, mean, numVar)
    exponent = ((x - mean')') * inv(cov) * (x - mean');
    prediction = ((1 / (sqrt((2*pi)^numVar * abs(det(cov))))) * exp(-0.5 * exponent));
end


%% Feature matrices (for training)
[class_1_row, class_1_col] = find(training_mask==1);
[class_2_row, class_2_col] = find(training_mask==2);
[class_3_row, class_3_col] = find(training_mask==3);
[class_4_row, class_4_col] = find(training_mask==4);
cls1 = zeros(length(class_1_row),4); %class 1 (values/observations, feature)
cls2 = zeros(length(class_2_row),4); %class 2
cls3 = zeros(length(class_3_row),4); %class 3
cls4 = zeros(length(class_4_row),4); %class 4
quadrant_images(1,:) = quadrants_mosaic1(1,:,1); % d1 Q1 feature image
quadrant_images(2,:) = quadrants_mosaic1(1,:,2); % d1 Q2 feature image
quadrant_images(3,:) = quadrants_mosaic1(1,:,5); % d1 Q5 feature image
quadrant_images(4,:) = quadrants_mosaic1(1,:,6); % d1 Q6 feature image
for i = 1:4
    quadrant_image = quadrant_images(i,:);
    quadrant_image = reshape(quadrant_image, [512, 512]).'; % transpose because reshape cre

    %Class1
    q = 1;
    for j = [class_1_row'; class_1_col']
        cls1(q,i) = quadrant_image(j(1), j(2));
        q = q + 1;
    end

    %Class2
    q = 1;
    for j = [class_2_row'; class_2_col']
        cls2(q,i) = quadrant_image(j(1), j(2));
        q = q + 1;
    end

    %Class3
    q = 1;
    for j = [class_3_row'; class_3_col']
        cls3(q,i) = quadrant_image(j(1), j(2));
        q = q + 1;
    end
```

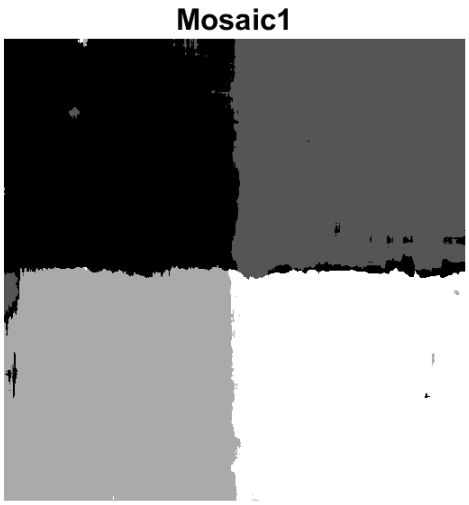
```
    %Class4
    q = 1;
    for j = [class_4_row'; class_4_col']
        cls4(q,i) = quadrant_image(j(1), j(2));
        q = q + 1;
    end
end

%% Covariance matrix and means (training)
cov_class1 = cov(cls1);
cov_class2 = cov(cls2);
cov_class3 = cov(cls3);
cov_class4 = cov(cls4);

means_class1 = mean(cls1);
means_class2 = mean(cls2);
means_class3 = mean(cls3);
means_class4 = mean(cls4);

%% Feature matrices
x1 = [quadrants_mosaic1(1,:,1); quadrants_mosaic1(1,:,2); quadrants_mosaic1(1,:,5); quadra
x2 = [quadrants_mosaic2(1,:,1); quadrants_mosaic2(1,:,2); quadrants_mosaic2(1,:,5); quadra
x3 = [quadrants_mosaic3(1,:,1); quadrants_mosaic3(1,:,2); quadrants_mosaic3(1,:,5); quadra
%% Predicting / Classifying
numVar = 4;

% Mosaic 1
res1 = zeros(512*512,1);
for i = 1:(512*512)
    pixel = x1(:,i);
    p1 = predict(pixel, cov_class1, means_class1, numVar);
    p2 = predict(pixel, cov_class2, means_class2, numVar);
    p3 = predict(pixel, cov_class3, means_class3, numVar);
    p4 = predict(pixel, cov_class4, means_class4, numVar);
    P = [p1, p2, p3, p4];
    [M, I] = max(P);
    res1(i) = I;
end

% Mosaic 2
res2 = zeros(512*512,1);
for i = 1:(512*512)
    pixel = x2(:,i);
    p1 = predict(pixel, cov_class1, means_class1, numVar);
    p2 = predict(pixel, cov_class2, means_class2, numVar);
    p3 = predict(pixel, cov_class3, means_class3, numVar);
    p4 = predict(pixel, cov_class4, means_class4, numVar);
```

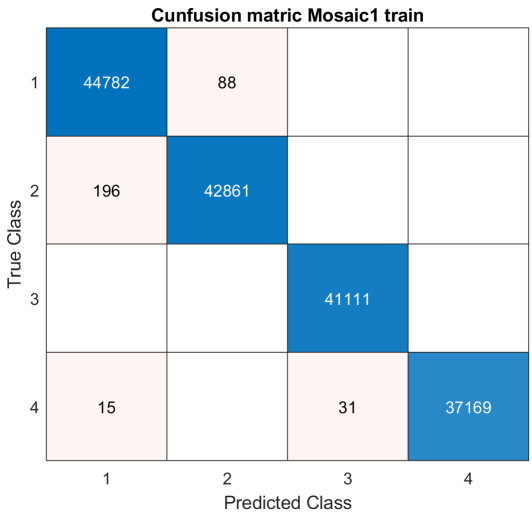
```
    P = [p1, p2, p3, p4];
    [M, I] = max(P);
    res2(i) = I;
end

% Mosaic 3
res3 = zeros(512*512,1);
for i = 1:(512*512)
    pixel = x3(:,i);
    p1 = predict(pixel, cov_class1, means_class1, numVar);
    p2 = predict(pixel, cov_class2, means_class2, numVar);
    p3 = predict(pixel, cov_class3, means_class3, numVar);
    p4 = predict(pixel, cov_class4, means_class4, numVar);
    P = [p1, p2, p3, p4];
    [M, I] = max(P);
    res3(i) = I;
end
```

This code is found in 'predict.m' and 'task4_5_6.m'

Task 5)

Black: class 1; Dark gray: class 2; light gray: class 3; white: class 4;



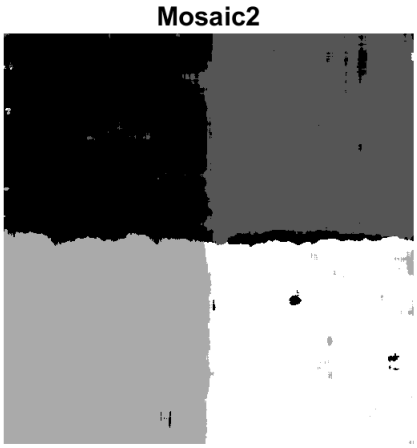


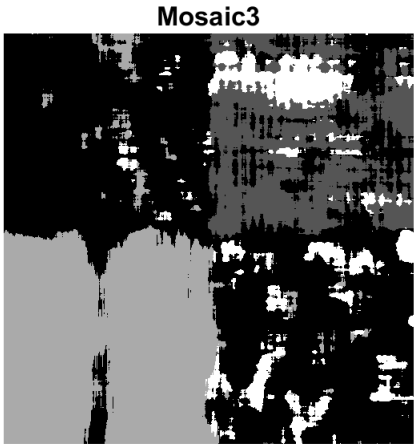
accuracy: 0.9980, 0.9954, 1.0000, 0.9988

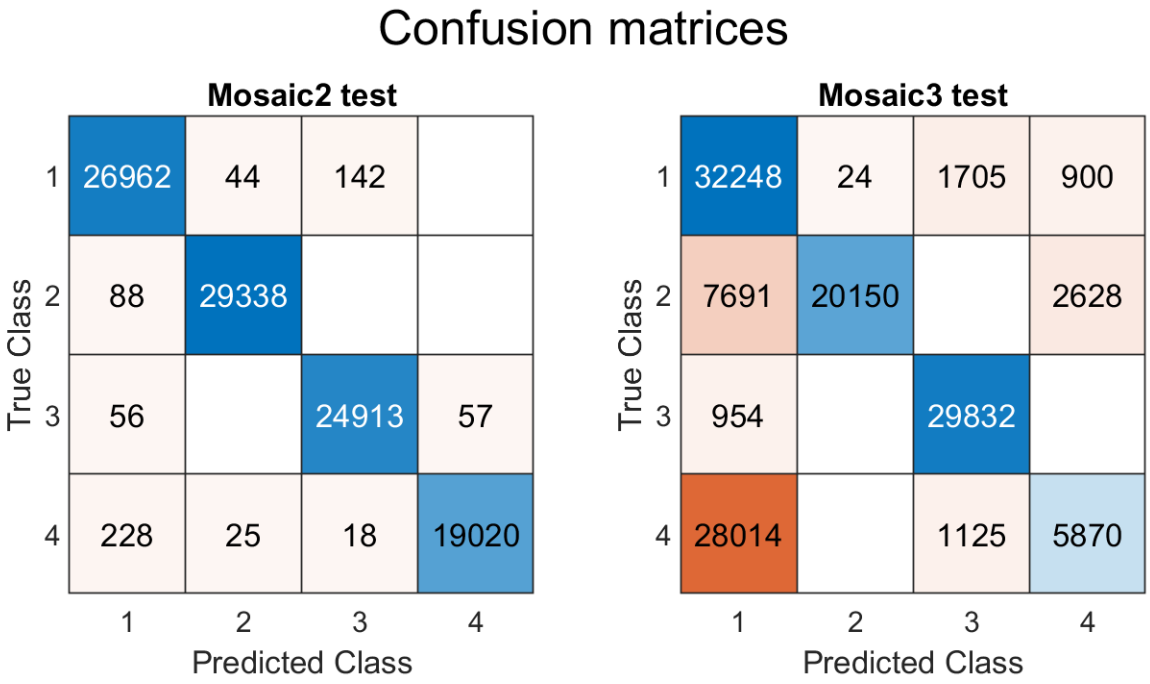
As we see we have very god calcification on the test image.
The accuracy scores are almost perfect with class 3 actuality having perfect accuracy.
From the confusion matrix we see that class 1 had 88 pixels mislabeld as class 2.
class 2 196 mislabeld as class 1.
Showing some difficulty in separating 1 from 2.
Class 4 also had 15 mislabels as class 1 and 31 mislabels as class 3.

Task 6)

Black: class 1; Dark gray: class 2; light gray: class 3; white: class 4;







Accuracy mosaic 2: 0.9931, 0.9970, 0.9955, 0.9860

Accuracy mosaic 3: 0.9246, 0.6613, 0.9690, 0.1677

We see mosaic 2 also has very god results with class 4 having the worst accuracy, but class 1 having most points not part of the class classified as such.

Mosaic 3 on the other hand is not as good.
Class 3 was quite good. So was class 1, but class 1 also had a lot of false positives in the region of class 4. Even more than class 4 itself.
Class 2 was classified better than class 4 but still not great.

Did not do the voluntary extension.