TEXTURE ANALYSIS AND CLASSIFICATION

Introduction

Texture is one of the important characteristics used in object recognition. Commonly, the textural features are derived from **cooccurrence** matrices, discrete Markov Random Fields, Gabor multi-channel filters, or fractal geometry. In this exercise we study gray level **cooccurrence** matrices and the textural features that can be derived from them. Finally, we apply the textural features to the problem of image classification.

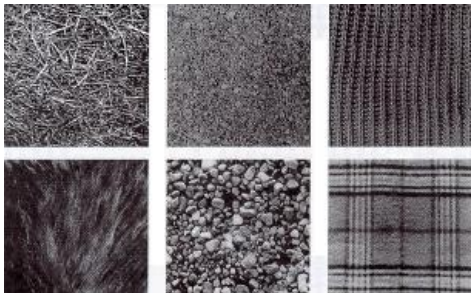


Fig.1: Textures: grass, cork, knitted fabric, dog fur, river pebbles and checkered textile (from Sonka[2], p. 719).

I. Cooccurrence-matrix and textural features.

The Gray Level Cooccurrence Matrix (**GLCM**) is based on the repeated occurrence of gray level in the texture. The GLCM $P_{\varphi,d}(a,b)$, for image I (size $M \times N$) quantized to N_g gray levels, is a matrix of non-normalized frequencies describing how frequently two pixels with gray level a, b are separated by distance d in direction φ. See Sonka [2] for the formal definitions of GCLMs. Example: for angle $\varphi = 0$, $P_{\varphi=0,d}(a,b)$ is defined by :

$$P_{0,d}(a,b) = \#\left\{ \begin{matrix} ((k,l),(m,n)) \in D: \\ k-m=0, \qquad |l-n| = d, \qquad I(k,l) = a, \qquad I(m,n) = b \end{matrix} \right\}$$

$$D = (M \times N) \times (M \times N)$$

Haralick [1] defined 14 features that can be derived from the cooccurrence matrices and subsequently used in texture description and discrimination. In the **Appendix** (a separate pdf-file) you can find definitions for all of the Haralick's features. The most commonly features used in image analysis are Angular Second Moment (ASM), Contrast (CON), Correlation (COR) and Entropy (ENT).

II. Classification of single-texture images.

The classification problem can be formulated as follows: given a number of known images and an unseen image, decide to which class the unseen image belongs.

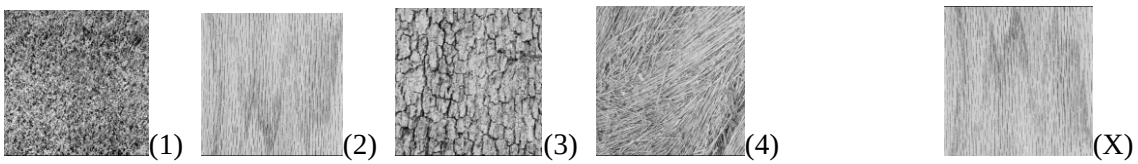


Fig.2: Known images (four classes) and a new image. What class does the image X belong to?

EXERCISES

PART I. Cooccurrence matrix and textural features.

1. Implement a function **`cooc`,** that calculates a cooccurrence matrix for an image, for distance **`d=1`** between pixels.

```
function P = cooc(Im, angle, Ng, normalize)
where
      Im          -     input grayscale image
      angle       -     0,45,90,135 degrees
      Ng          -     number of quantization gray levels
      normalize   -     frequency normalize entries of P
      P           -     cooccurrence matrix
```

Note: The quantization to `Ng` levels, `1..Ng`, should be done <u>before</u> `cooc` is called. Take a look at `quantize.m` function that performs the quantization using Matlab's `histeq` function. Below you can find a test image `Im` and the corresponding cooccurrence matrices. You can use it for testing functions you implement.

```
Im = [ 1     1     2     2        % Im is already quantized to Ng=4 levels
       1     1     2     2
       1     3     3     3
       3     3     4     4 ];

>> P0 = cooc(Im, 0, 4, 0)
P0 =
       4     2     1     0
       2     4     0     0
       1     0     6     1
       0     0     1     2

>> P0n = cooc(Im, 0, 4, 1)              % frequency normalized
P0n =
    0.1667    0.0833    0.0417         0
    0.0833    0.1667         0         0
    0.0417         0    0.2500    0.0417
         0         0    0.0417    0.0833

>> P45 = cooc(Im, 45, 4, 0)
P45 =
     4     1     0     0
     1     2     2     0
     0     2     4     1
     0     0     1     0

>> P90 = cooc(Im, 90, 4, 0)
P90 =
     6     0     2     0
     0     4     2     0
     2     2     2     2
     0     0     2     0

>> P135 = cooc(Im, 135, 4, 0)
P135 =
     2     1     3     0
     1     2     1     0
     3     1     0     2
     0     0     2     0
```

2. Implement at least four of Haralick's textural features (see **Appendix**):

```
function asm(p)              - Angular Second Moment ASM
function contrast(p)         - Contrast CON
function corr(p)             - Correlation COR
function entropy(p)          - Entropy ENT
```

where **p** is a frequency normalized co-occurrence matrix.

$$\boldsymbol{ASM} = \sum_{a=1}^{N_g}\sum_{b=1}^{N_g} p^2(a,b)$$

$$\boldsymbol{CON} = \sum_{k=0}^{N_g-1}\left\{k^2\left\{\sum_{a=1}^{N_g}\sum_{b=1}^{N_g} p(a,b)\right\}\right\} \quad |a-b| = k, \quad k = 0, \ldots, N_g - 1$$

$$\boldsymbol{COR} = \frac{\sum_{a=1}^{Ng}\sum_{b=1}^{Ng} a \cdot b \cdot p(a,b) - \mu_a \cdot \mu_b}{\sigma_a \cdot \sigma_b}$$

$$\boldsymbol{ENT} = -\sum_{a=1}^{N_g}\sum_{b=1}^{N_g} p(a,b) \cdot \ln\left(p(a,b)\right)$$

```
>>  % test your functions on cooccurrence matrices for the image Im
>>  P0n = cooc(Im,0,4,1);
>>  P0n_features =[asm(P0n) contrast(P0n) corr(P0n) entropy(P0n)]

P0n_features =

    0.1458    0.5833    0.7195    2.0947

>> P45n = cooc(Im, 45, 4, 1)
P45n =

    0.2222    0.0556         0         0
    0.0556    0.1111    0.1111         0
         0    0.1111    0.2222    0.0556
         0         0    0.0556         0

>> P45n_features=[asm(P45n) contrast(P45n) corr(P45n) entropy(P45n)]

P45n_features =

    0.1481    0.4444    0.7353    2.0432
```

```
>> P90n = cooc(Im, 90, 4, 1)

P90n =

    0.2500         0    0.0833         0
         0    0.1667    0.0833         0
    0.0833    0.0833    0.0833    0.0833
         0         0    0.0833         0

>> P90n_features =[asm(P90n) contrast(P90n) corr(P90n) entropy(P90n)]

P90n_features =

    0.1389    1.0000    0.4857    2.0947

>> P135n = cooc(Im, 135, 4, 1)

P135n =

    0.1111    0.0556    0.1667         0
    0.0556    0.1111    0.0556         0
    0.1667    0.0556         0    0.1111
         0         0    0.1111         0

>>P135n_features=[asm(P135n) contrast(P135n) corr(P135n) entropy(P135n)]

P135n_features =

    0.1173    1.7778    0.1628    2.2161
```

3. Use your developed functions to calculate the `ASM, CON, COR` and `ENT` features for the images of (1) `wood` (2) `grass` and (3) `random` texture. Use `Ng=8` quantization levels. Complete the Tables 1, 2 and 3 with the calculated ASM, CON, COR and ENT values (Hint: You might use `texturetable.m` for the tables).

```
>> I= imread('wood.tiff '); I= quantize(I,8);
>> P0 = cooc(I,0,8,1); …
```

Images :

```
(1)   wood.tiff       Brodatz - Wood grain (D68)
(2)   grass.tiff      Brodatz - Grass (D9)
(3)   random.png      random image,  rand(512)
```

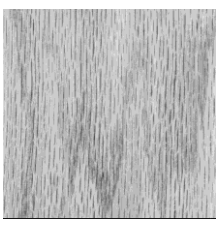
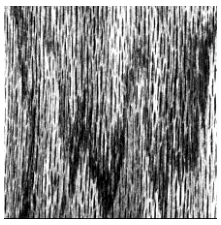
 (a)  (b)

Fig. 3. (a) The original wood image and (b) histogram equalized and quantized image (8 levels).

	ASM	CON	COR	ENT
0°				
45°				
90°				
135°				
Average				

Table 1. ASM, contrast, correlation and entropy for the wood-image.

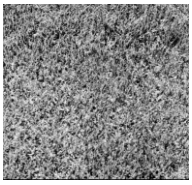
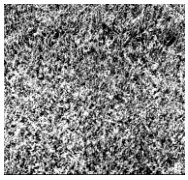
 (a)  (b)

Fig. 4. (a) Grass (b) histogram equalized and quantified (8 levels).

	ASM	CON	COR	ENT
0°				
45°				
90°				
135°				
Average				

Table 2. ASM, contrast, correlation and entropy for the grass-image.

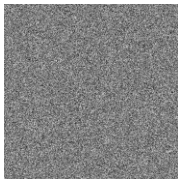


Fig 5: Uniform random texture (generated using Matlab's `rand(512)` function).

	ASM	CON	COR	ENT
0°				
45°				
90°				
135°				
Average				

Table 3. ASM, contrast, correlation and entropy for the random-image.

Query: Which texture (wood, grass, random) has highest ASM, CON , COR, ENT ? Do the feature values coincide with your perception ?

ASM ……..
CON ……..
COR ……..
ENT ……

Query: What happens with the feature values, when we rotate the image 90 degrees (`imrotate(I,90)`) ?

……

PART II. Classification of single-texture images using textural features.

In this part we will investigate a classifier for discrimination between different types of images. As an example we will use the `wood` and `grass` images (two-class problem) from the Part I. The task is to build a classifier capable to distinguish between the `wood` and `grass` images.

Usually the process of the classifier design consists of the following steps:

1. Feature extraction.
2. Feature selection.
3. Classifier training.
4. Classifier testing (performance evaluation).

You will run a number of Matlab programs supplying different input parameters. This exercise does not require any programming, however the programs require the functions you developed in the Part I, i.e. `cooc, asm, contrast, corr` and `entropy` functions.

Before running the experiments you should read the instructions below and study the source code of the Matlab functions:
`textfeat.m` – a function for feature extraction
`testerr.m` – a function for error rate estimation of the classifier

Step 1: Feature extraction.

Feature extraction is a process of calculation of potentially discriminatory variables to be used for the classification task.

As features we will use the textural features that we have derived from the cooccurrence matrices in the Part I. In this way we obtain five features (four different directions and an average) for every textural feature type. In total, our feature vector will consist of 20 features (see Table 4).

Having only one image we obtain only one instance of the feature vector. To obtain a better representation of the texture we may divide the image into a set of smaller images and calculate the features for them. In this way we obtain a higher number of feature vectors.

Using the function `textfeat` the features will be computed from single texture grayscale image on 196 sub-images of size 32x32 pixels. In this way we simulate 196 images. The images are histogram equalized and quantified to $N_g=8$ levels.

It is desirable to normalize the feature values to the same mean and the same variance. In this way we make the features equally important. Otherwise some features could dominate and the classifier might be not optimal.

Table 4. The cooccurrence features.

Feature number	Feature name (distance d=1)
1	asm 0 deg
2	asm 45 deg
3	asm 90 deg
4	asm 135 deg
5	con 0 deg
6	con 45 deg
7	con 90 deg
8	con 135 deg
9	cor 0 deg
10	cor 45 deg
11	cor 90 deg
12	cor 135 deg
13	ent 0 deg
14	ent 45 deg
15	ent 90 deg
16	ent 135 deg
17	asm average
18	con average
19	cor average
20	ent average

* Run the following commands to perform feature extraction and normalization on two images (class 0 == wood, class 1 == grass) :

```
>> % FEATURE EXTRACTION
>> % function textfeat() calculates 20 features for
>> % 196 subimages of size 32x32
>>
display  = 1;      % a flag to display images
woodId   = 0;      % set class label of wood to 0
%
% calculate textural features for wood (20 features)
% using 32x32 subimages
% the feature values are returned in P matrix,
% and class labels in T (target vector)
%
[P0, T0] = textfeat('wood.tiff' , woodId , display);
%
% calculate  textural features for grass
grassId  = 1;  % set class label of grass to 1
[P1, T1] = textfeat('grass.tiff', grassId, display);
%
% collect P0,P1,T0,T1 into training sets
%
Ptr = [P0; P1];   % training set features (parameters)
Ttr = [T0; T1];   % training set labels (target vectors)
%
% normalize the features to zero mean and unit variance, (why ?)
%
[Ptr, Ptr_mean, Ptr_std]  = normalize(Ptr)
% check, mean(Ptr)== 0 ? , std(Ptr) == 1 ???, after normalization
```

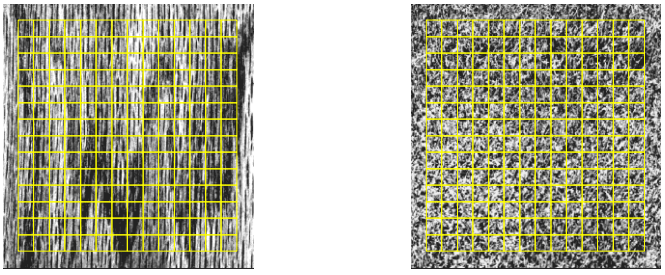


Fig 6. Training images (a) wood (b) grass. The grid squares represent 32x32 sub-images taken as training samples.

Step 2: Feature selection.
Feature selection is a process of finding the best feature subset from the fixed set of the original features. Usually more features leads to the better performance. However, the irrelevant features may result in performance degradation thus we need to select an optimal set of features. To determine the best feature subset one needs to examine all possible subsets of size p. To choose the best subset of size p from set of d thus requires

$$\binom{d}{p} = \frac{d!}{p!\,(d-p)!}$$

examinations.

In this exercise we will evaluate subsets with two elements using 2-D scatterplots. A 2-D scatterplot is a chart displaying values for two variables. When in the 2-D scatterplot the overlapping between samples from different classes is low than we expect that the corresponding variables may be regarded as "good" discriminatory features. For $d = 20$, and $p = 2$, this would require a visual inspection of 190 scatterplots. Instead of visual investigation we might measure the separability of classes using Bhattacharyya distance and use it for feature selection. For multivariate Gaussian distributions the Bhattacharyya D_B distance is expressed as:

$$D_B = \frac{1}{8}(\boldsymbol{m}_1 - \boldsymbol{m}_2)^T \boldsymbol{S}^{-1}(\boldsymbol{m}_1 - \boldsymbol{m}_2) + \frac{1}{2}\ln\left\{\frac{\det(\boldsymbol{S})}{\sqrt{\det(\boldsymbol{S}_1)\det(\boldsymbol{S}_2)}}\right\}$$

where $\boldsymbol{m}$ is the class mean and $\boldsymbol{S}$ is the class covariance matrix, $\boldsymbol{S} = (\boldsymbol{S}_1 + \boldsymbol{S}_2)/2$.

* Run function `scattter2D`, to visually investigate the distribution of the features and the overlapping region between classes:

```
>> % display 2-D scatterplots for visual inspection
f1 =  2; % index of the first feature,  ASM 45 deg
f2 =  4; % index of the second feature, ASM 135 deg
% FEATURE INSPECTION AND SELECTION
scatter2D(Ptr, Ttr, f1, f2);
```

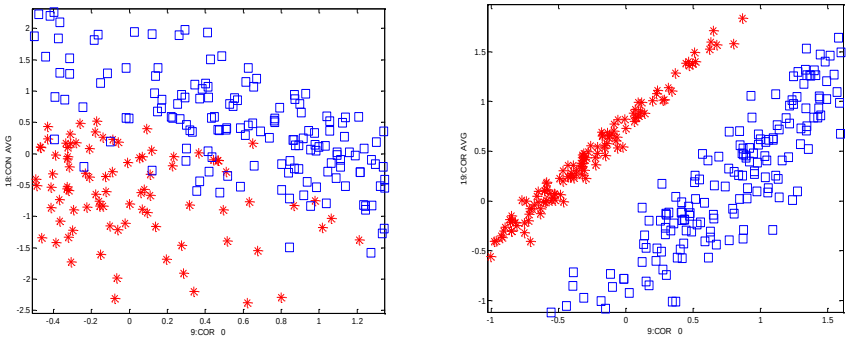


Fig 7. The 2D-scatterplots: in (left) we observe a considerable overlapping between the class samples (class 0 = *, class 1 = ⊡), while in (right) the classes are well separated. Thus we expect that using features from (right) will result in a better classifier.

Query: Investigate 2D scatterplots for some of the feature pairs and suggest which features may be useful for our discrimination problem.

Feature pair No 1: ……
Feature pair No 2: ……
Feature pair No.3 ……

Step 3: Classifier training.

As a classifier we will use the voting kNN (k-Nearest Neighbour) classifier. The voting kNN (k-Nearest Neighbour) classifier assigns an unknown sample to a majority class of its k nearest neighbours:

Given: A training set `Ttr` of N patterns (feature vectors) $\boldsymbol{x}_1, \boldsymbol{x}_2, \ \ldots \ \boldsymbol{x}_N$, labeled by c classes.
A new pattern $\boldsymbol{x}$.

1. Compute for a given $\boldsymbol{x}$ the k nearest neighbours from a whole training set, using the Euclidean pattern distance measure:

$$d(\boldsymbol{x}, \boldsymbol{x}_i) = \sqrt{\sum_{j=1}^{D} (x_j - x_{ij})^2}$$

$$\boldsymbol{x}_i = [x_{i1} \ x_{i2} \ \ldots \ x_{iD}], \quad \boldsymbol{x} = [x_1 \ x_2 \ \ldots \ x_D]$$

   D is dimensionality of the feature vector $\boldsymbol{x}$.

2. Assign $\boldsymbol{x}$ to a majority class of its k nearest neighbours.

For kNN classifier we simply keep all the training samples in a database and every time a new unknown sample has to be classified, we calculate the k nearest neighbours and assign the sample to the class most frequently represented among these neighbours.

Step 4: Classifier testing.

The final step is the classifier's performance evaluation. We need to estimate the classifier's probability of error (error rate), i.e. how many errors (wrong classifications) we expect when using the classifier. The straightforward way to evaluate a classifier is simply counting the number of errors on an independent **test** data set. The estimate of the classifier's error rate is then the ratio:

$$errorRate = N_e/N$$

where N is the number of test samples, N_e is the number of misclassified samples.

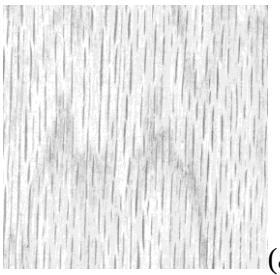
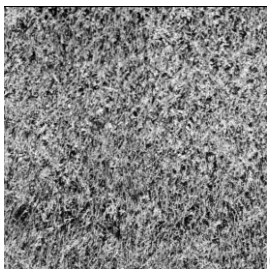

(a)

(b)

Fig.8. Test images: (a) `'testwood.tiff'`, the true class is `wood`.
(b) `'testgrass.tiff'`, the true class is `grass`. Number of 32x32 subimages is 196.

* Run the following commands to estimate error rate of the kNN-classifier (k=1):

```
% ERROR RATE ESTIMATION
% calculate the features for the test images
[Pte0, Tte0] = textfeat('testwood.tiff' ,  0, display);
[Pte1, Tte1] = textfeat('testgrass.tiff',  1, display);
%
% collect test features and labels into a test set
Pte = [Pte0; Pte1];
Tte = [Tte0; Tte1];
%
% transform the features using the normalization transform
sizeTr = size(Ptr,1);
sizeTe = size(Pte,1);
Pte = Pte(:,:) - repmat(Ptr_mean,sizeTe,1);     % substract the mean
Pte = Pte./repmat(Ptr_std, sizeTe,1);           % divide by std dev
%
% select two of the features
f1=9; f2=14;            % f1=index feature #1, f2=index of feature #2
%
% evaluate error rate using features f1 & f2
testErrorRate = testerr(Ptr, Ttr, Pte, Tte, f1, f2);


Nr of test errors = 67 of 392
Test error rate   = 0.17092
Used features     : 9 14
```

* Evaluate error rate using another pair of features (or write a loop that tests a larger number of pairs). Try to find a feature pair that gives error rate below 8%.

```
f1 = 9   % your choice 1 (1..20)
f2 = 18  % your choice 2 (1..20)
testerr(Ptr, Ttr, Pte, Tte, f1, f2)
%
```

Reporting

1. Matlab source code for `cooc`, `asm`, `contrast`, `corr` and `entropy` functions.
2. Answers to the questions.
3. Scatterplot(s) for the best feature pairs, with a low test error rate (below 8%), and the estimates of the error rates.
4. Optional: Calculate the Bhattacharyya distance between wood and grass training distributions (assuming Gaussians), using the selected feature pairs and check whether it can be used for feature selection.

Send your report to Artur: **`artur.chodorowski@chalmers.se`**

References

[1] Haralick R M, Shanmugam K, and Dinstein I, "Textural Features for Image Classification", *IEEE Trans on Systems, Man and Cybernetics.* Vol.3, No. 6, November 1973, pp. 610-621
[2] Sonka M, Hlavac V and Boyle R, "Image Processing, Analysis and Machine Vision", 3rd ed, 2008, pp. 718-725
[3] The USC-SIPI Image Database. Textures: http://sipi.usc.edu/database/