AVPR - Lab - 2nd Exercise

Carlos García

December 2016

Contents

1 Problem statement

The goal of this exercise is to correctly classify a set of images with textures into 6 classes. We have 24 images corresponding to four image samples of each kind of texture. The problem to solve is to find the features that classify each sample in the correct class: ideally, all samples of the same texture have to be in the same class.

The images to classify are in "Images_Exer2" package on moodle.

To solve the problem you can use any software that implements clustering algorithms, program yourself the classifier or use ImageJ software. In this last case, you can use a K-means clustering plugin. This plugin compute clusters of feature sets using pixel values (integer value) as image features, but also it can use any real value features.

...

2 Basic concepts

Before starting with the project, I think it is very important giving a short review to some key concepts like:

2.1 What does texture means in images?

An image texture is a set of metrics calculated in image processing designed to quantify the perceived texture of an image. Image texture gives us information about the spatial arrangement of color or intensities in an image or selected region of an image. [1]

2.2 What is texture analysis?

Texture analysis refers to the characterization of regions in an image by their texture content. Texture analysis attempts to quantify intuitive qualities described by terms such as rough, smooth, silky, or bumpy as a function of the spatial variation in pixel intensities. In this sense, the roughness or bumpiness refers to variations in the intensity values, or gray levels. Texture analysis is used in a variety of applications, including remote sensing, automated inspection, and medical image processing. Texture analysis can be used to find the texture boundaries, called texture segmentation. Texture analysis can be helpful when objects in an image are more characterized by their texture than by intensity, and traditional thresholding techniques cannot be used effectively. [2]

2.3 What is the co-ocurrence matrix (GLCM)?

A co-occurrence matrix or co-occurrence distribution is a matrix that is defined over an image to be the distribution of co-occurring pixel values (grayscale values, or colors) at a given offset. Whether considering the intensity or grayscale values of the image or various dimensions of color, the co-occurrence matrix can measure the texture of the image. Because co-occurrence matrices are typically large and sparse, various metrics of the matrix are often taken to get a more useful set of features. Features generated using this technique are usually called Haralick features, after Robert Haralick. [4]

2.4 What features can we get from the GLCM?

Haralick described 14 statistics that can be calculated from the co-occurrence matrix with the intent of describing the texture of the image: [3]

Angular Second Moment	$\sum_i \sum_j p(i,j)^2$
Contrast	$\sum_{n=0}^{N_g-1} n^2 \{\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j)\}, \lvert i-j \rvert = n$
Correlation	$\frac{\sum_i \sum_j (ij)p(i,j) - \mu_x \mu_y}{\sigma_x \sigma_y}$ where μ_x , μ_y , σ_x , and σ_y are the means and std. deviations of p_x and p_y , the partial probability density functions
Sum of Squares: Variance	$\sum_i \sum_j (i-\mu)^2 p(i,j)$
Inverse Difference Moment	$\sum_i \sum_j \frac{1}{1+(i-j)^2} p(i,j)$
Sum Average	$\sum_{i=2}^{2N_g} i p_{x+y}(i)$ where x and y are the coordinates (row and column) of an entry in the co-occurrence matrix, and $p_{x+y}(i)$ is the probability of co-occurrence matrix coordinates summing to $x+y$
Sum Variance	$\sum_{i=2}^{2N_g} (i - f_8)^2 p_{x+y}(i)$
Sum Entropy	$-\sum_{i=2}^{2N_g} p_{x+y}(i) \log\{p_{x+y}(i)\} = f_8$
Entropy	$-\sum_i \sum_j p(i,j) log(p(i,j))$
Difference Variance	$\sum_{i=0}^{N_g-1} i^2 p_{x-y}(i)$
Difference Entropy	$-\sum_{i=0}^{N_g-1} p_{x-y}(i) \log\{p_{x-y}(i)\}$
Info. Measure of Correlation 1	$\frac{HXY - HXY1}{\max\{HX, HY\}}$
Info. Measure of Correlation 2	$(1 - \exp[-2(HXY2 - HXY)])^{\frac{1}{2}}$ where $HXY = -\sum_i \sum_j p(i,j) \log(p(i,j))$, HX , HY are the entropies of p_x and p_y , $HXY1 = -\sum_i \sum_j p(i,j) \log\{p_x(i)p_y(j)\} HXY2 = -\sum_i \sum_j p_x(i)p_y(j) \log\{p_x(i)p_y(j)\}$
Max. Correlation Coeff.	Square root of the second largest eigenvalue of $\mathbf{Q}$ where $\mathbf{Q}(i,j) = \sum_k \frac{p(i,k)p(j,k)}{p_x(i)p_y(k)}$

2.5 What is clustering analysis?

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics. [5]

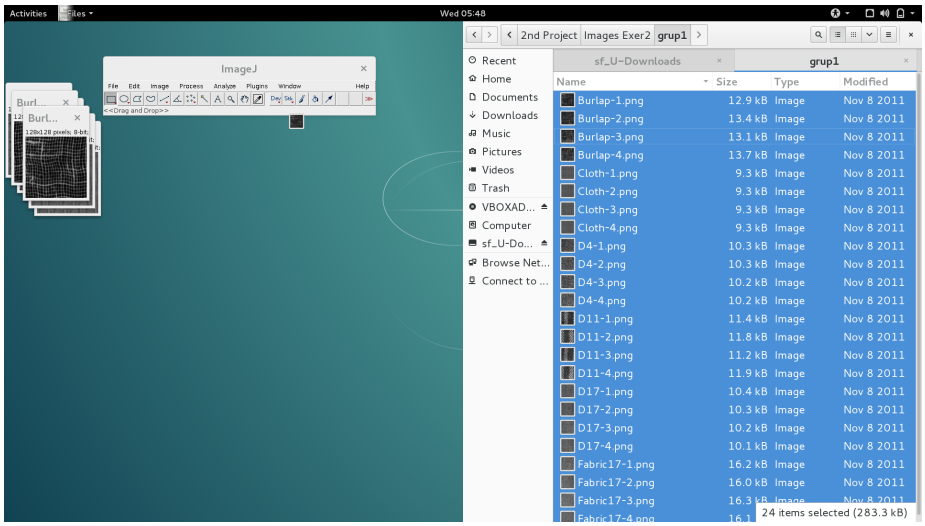
2.6 What is the K-means clustering?

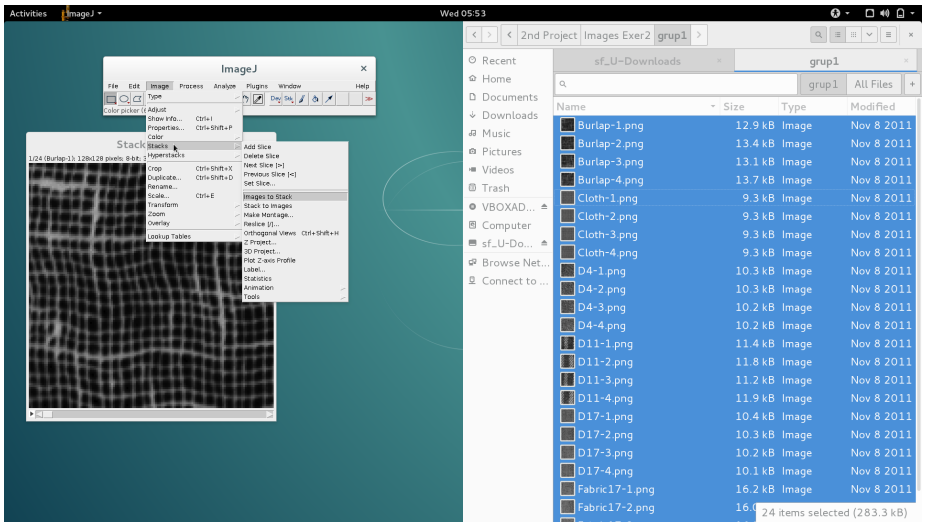
k-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

The problem is computationally difficult (NP-hard); however, there are efficient heuristic algorithms that are commonly employed and converge quickly to a local optimum. These are usually similar to the expectation-maximization algorithm for mixtures of Gaussian distributions via an iterative refinement approach employed by both algorithms. [6]

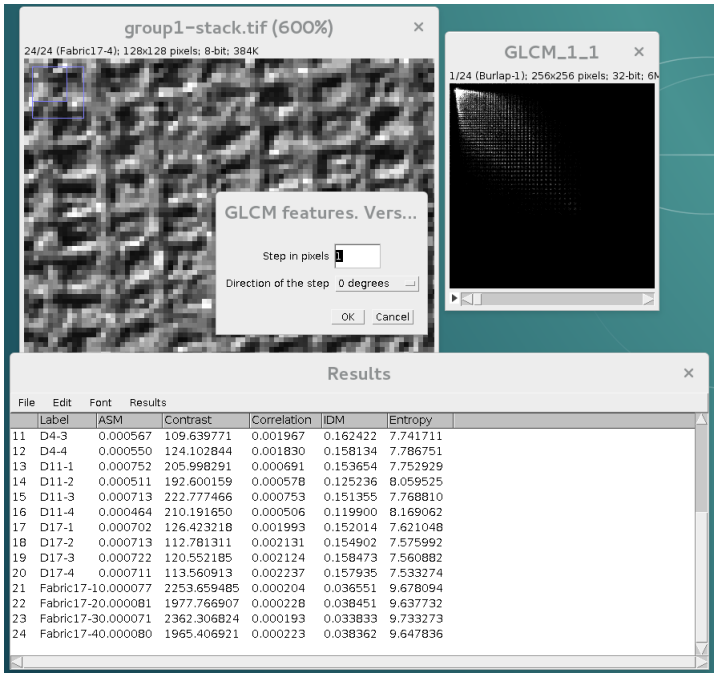
3 Methodology applied to solve the problem

My first approach was trying to make the project using imageJ, then I opened all images and created a stack with them:





Then I got the features from the stack



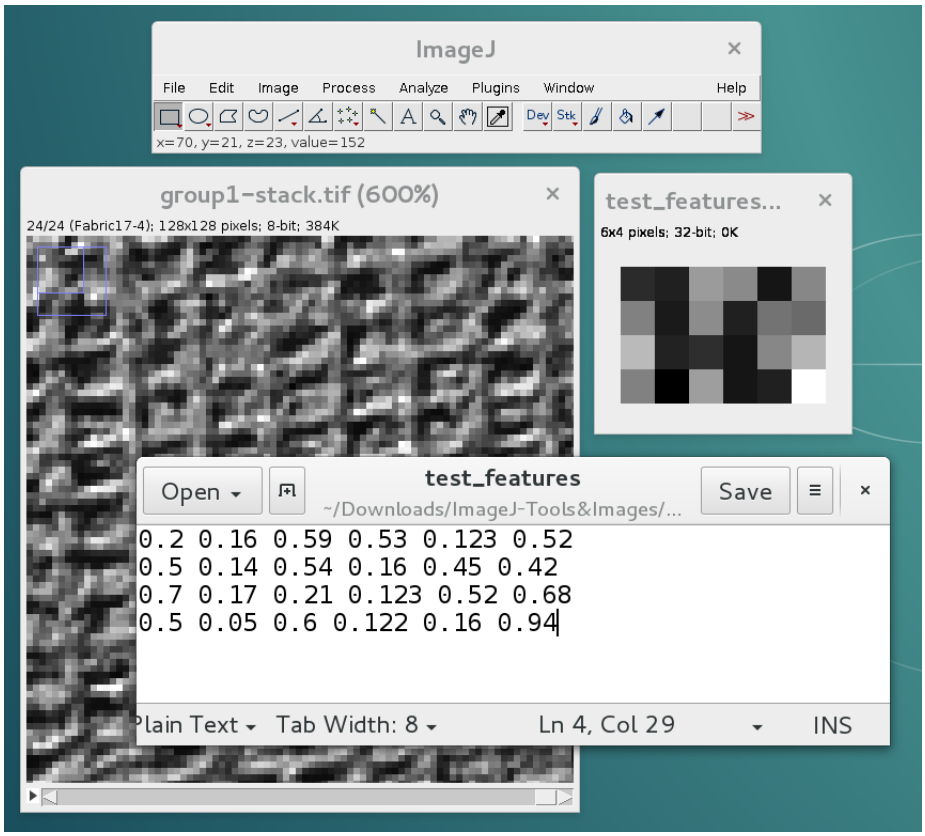
	Label	ASM	Contrast	Correlation	IDM	Entropy
11	D4-3	0.000567	109.639771	0.001967	0.162422	7.741711
12	D4-4	0.000550	124.102844	0.001830	0.158134	7.786751
13	D11-1	0.000752	205.998291	0.000691	0.153654	7.752929
14	D11-2	0.000511	192.600159	0.000578	0.125236	8.059525
15	D11-3	0.000713	222.777466	0.000753	0.151355	7.768810
16	D11-4	0.000464	210.191650	0.000506	0.119900	8.169062
17	D17-1	0.000702	126.423218	0.001993	0.152014	7.621048
18	D17-2	0.000713	112.781311	0.002131	0.154902	7.575992
19	D17-3	0.000722	120.552185	0.002124	0.158473	7.560882
20	D17-4	0.000711	113.560913	0.002237	0.157935	7.533274
21	Fabric17-1	0.000077	2253.659485	0.000204	0.036551	9.678094
22	Fabric17-2	0.000081	1977.766907	0.000228	0.038451	9.637732
23	Fabric17-3	0.000071	2362.306824	0.000193	0.033833	9.733273
24	Fabric17-4	0.000080	1965.406921	0.000223	0.038362	9.647836

Results

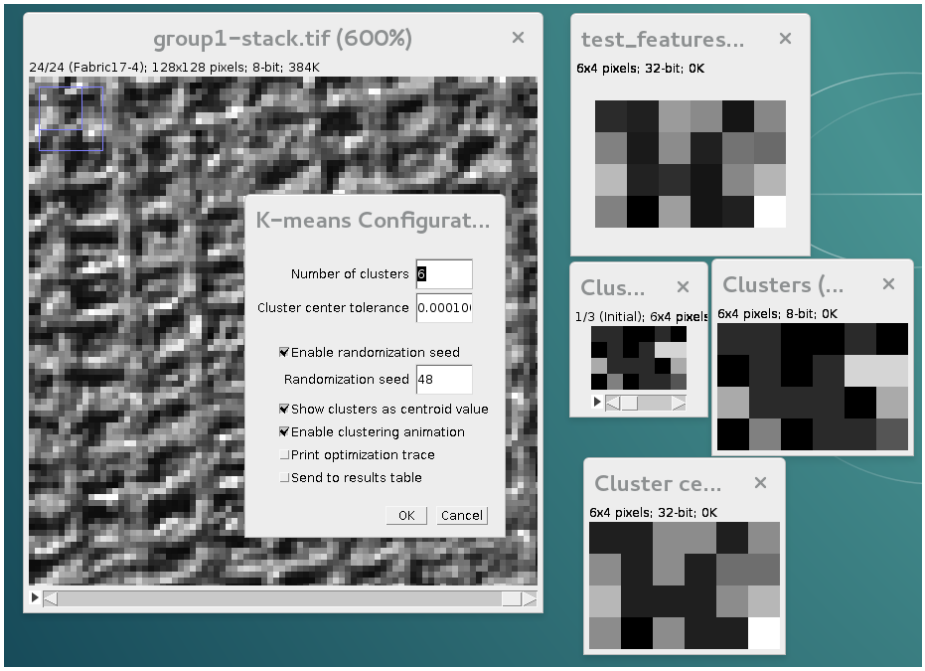
File Edit Font Results

	Label	ASM	Contrast	Correlation	IDM	Entropy
1	Burlap-1	0.000398	738.196960	0.000608	0.069363	8.418401
2	Burlap-2	0.000616	1076.187500	0.000448	0.084955	8.408513
3	Burlap-3	0.000286	591.023193	0.000577	0.065080	8.526521
4	Burlap-4	0.000362	902.548584	0.000458	0.067009	8.533038
5	Cloth-1	0.001202	121.182983	0.003970	0.158775	7.118755
6	Cloth-2	0.001213	123.062866	0.003941	0.156174	7.131342
7	Cloth-3	0.001166	120.165527	0.003956	0.153710	7.159682
8	Cloth-4	0.001187	123.612915	0.003886	0.151839	7.157748
9	D4-1	0.000528	109.268250	0.001862	0.159016	7.793877
10	D4-2	0.000520	108.097290	0.001830	0.154510	7.817680
11	D4-3	0.000567	109.639771	0.001967	0.162422	7.741711
12	D4-4	0.000550	124.102844	0.001830	0.158134	7.786751
13	D11-1	0.000752	205.998291	0.000691	0.153654	7.752929
14	D11-2	0.000511	192.600159	0.000578	0.125236	8.059525
15	D11-3	0.000713	222.777466	0.000753	0.151355	7.768810
16	D11-4	0.000464	210.191650	0.000506	0.119900	8.169062
17	D17-1	0.000702	126.423218	0.001993	0.152014	7.621048
18	D17-2	0.000713	112.781311	0.002131	0.154902	7.575992
19	D17-3	0.000722	120.552185	0.002124	0.158473	7.560882
20	D17-4	0.000711	113.560913	0.002237	0.157935	7.533274
21	Fabric17-1	0.000077	2253.659485	0.000204	0.036551	9.678094
22	Fabric17-2	0.000081	1977.766907	0.000228	0.038451	9.637732
23	Fabric17-3	0.000071	2362.306824	0.000193	0.033833	9.733273
24	Fabric17-4	0.000080	1965.406921	0.000223	0.038362	9.647836

Import sample features file and check clustering using k-means, (this is just for measuring execution time).



Using k-means clustering plugin I got this



This was a sample to compute how long it takes to perform all of this steps. Considering that maybe we need to try this features segmentation using different step sizes and orientations for the GLCM, and having into account that we must organize and normalize all features by ourselves, maybe it is a good idea trying to code an automatic alternative using python, matlab, octave or any other similar tool, besides, if I do it programmatically I'll be able to reuse this same script on future projects.

Let's try using python:

We first must search for a library which helps us to get similar features to those we obtained using ImageJ. A well known library for image processing in python is called "scikit-image", we can find it in:

http://scikit-image.org/

According to the library documentation, this are the features we can get:

greycoprops

`skimage.feature.` **`greycoprops`**`(P, prop='contrast')` **[source]**

Calculate texture properties of a GLCM.

Compute a feature of a grey level co-occurrence matrix to serve as a compact summary of the matrix. The properties are computed as follows:

- 'contrast': $\sum_{i,j=0}^{levels-1} P_{i,j}(i-j)^2$
- 'dissimilarity': $\sum_{i,j=0}^{levels-1} P_{i,j}|i-j|$
- 'homogeneity': $\sum_{i,j=0}^{levels-1} \frac{P_{i,j}}{1+(i-j)^2}$
- 'ASM': $\sum_{i,j=0}^{levels-1} P_{i,j}^2$
- 'energy': $\sqrt{ASM}$
- **'correlation':**

$$\sum_{i,j=0}^{levels-1} P_{i,j}\left[\frac{(i-\mu_i)\ (j-\mu_j)}{\sqrt{(\sigma_i^2)(\sigma_j^2)}}\right]$$

This is how we get the features using the library:

```
import matplotlib.pyplot as plt
import numpy as np
import skimage.io as ski_io
import skimage.feature as ski_feature

props = {}
props['contrast'] = ski_feature.greycoprops(GLCM, 'contrast')[0][0]
props['dissimilarity'] = ski_feature.greycoprops(GLCM, '
    dissimilarity')[0 ][0]
props['homogeneity'] = ski_feature.greycoprops(GLCM, 'homogeneity')
    [0][0]
props['energy'] = ski_feature.greycoprops(GLCM, 'energy')[0][0]
props['correlation'] = ski_feature.greycoprops(GLCM, 'correlation')
    [0][0]
props['ASM'] = ski_feature.greycoprops(GLCM, 'ASM')[0][0]
```

Now we must apply some clustering technique. Searching on the Internet, one of the most used library for clustering is called "scipy.cluster" and it includes and implementation of k-means clustering algorithm.

First we must transform our features array from python-array to numpy-array using the method numpy.asarray(). This way our data is better ordered and can be easily printed, then we must normalize our features data using numpy library.

This is a sample using all features from the library:

```
['energy', 'homogeneity', 'contrast', 'correlation', 'dissimilarity', 'ASM']
[[    0.02008     0.06989   713.89924     0.6691    20.97195     0.0004 ]
 [    0.025       0.08561  1048.55795     0.60302   24.09941     0.00063]
 [    0.01699     0.06556   569.31865     0.78897   19.00824     0.00029]
 [    0.01916     0.06753   852.9359      0.72294   22.77731     0.00037]
 [    0.03493     0.16002    64.27891     0.73784    6.2094      0.00122]
 [    0.03508     0.1574     65.43473     0.72899    6.23136     0.00123]
 [    0.0344      0.15492    66.42188     0.73799    6.34473     0.00118]
 [    0.03471     0.15303    67.27393     0.73367    6.35488     0.0012 ]
 [    0.02763     0.15486   119.297       0.95473    7.72638     0.00076]
 [    0.02276     0.12622   154.26452     0.95205    9.09326     0.00052]
 [    0.0269      0.15255   103.49975     0.95615    7.35039     0.00072]
 [    0.02169     0.12084   165.59972     0.95505    9.49342     0.00047]
 [    0.02668     0.15321    64.84793     0.91814    6.36393     0.00071]
 [    0.0269      0.15612    58.3443      0.92227    6.05309     0.00072]
 [    0.02708     0.15972    58.17797     0.92014    6.01655     0.00073]
 [    0.02686     0.15918    53.34996     0.9229     5.82991     0.00072]
 [    0.02315     0.16027    57.79189     0.93667    5.95971     0.00054]
 [    0.02297     0.15572    60.33372     0.93463    6.11411     0.00053]
 [    0.024       0.1637     54.97263     0.93483    5.83212     0.00058]
 [    0.02363     0.15938    56.48585     0.93662    5.94488     0.00056]
 [    0.00884     0.03684  2141.56668     0.65746   35.30278     0.00008]
 [    0.00903     0.03875  1893.08274     0.66703   33.15336     0.00008]
 [    0.00849     0.03409  2306.52215     0.6538    36.90133     0.00007]
 [    0.009       0.03866  1899.84726     0.683     33.28439     0.00008]]
[[ 0.5723   0.42697  0.30951  0.69978  0.56833  0.32753]
 [ 0.71272  0.52295  0.45461  0.63068  0.65308  0.50797]
 [ 0.48423  0.4005   0.24683  0.82515  0.51511  0.23448]
 [ 0.54622  0.41253  0.36979  0.7561   0.61725  0.29835]
 [ 0.99572  0.97755  0.02787  0.77168  0.16827  0.99145]
 [ 1.       0.96154  0.02837  0.76242  0.16887  1.     ]
 [ 0.9806   0.94637  0.0288   0.77184  0.17194  0.96157]
 [ 0.98928  0.93485  0.02917  0.76732  0.17221  0.97868]
 [ 0.78772  0.94602  0.05172  0.99852  0.20938  0.62051]
 [ 0.6489   0.77105  0.06688  0.99572  0.24642  0.42107]
 [ 0.76674  0.93187  0.04487  1.       0.19919  0.58788]
```

```
 [ 0.61829  0.7382   0.0718   0.99886  0.25726  0.38229]
 [ 0.76058  0.93592  0.02812  0.96026  0.17246  0.57849]
 [ 0.76671  0.95371  0.0253   0.96457  0.16403  0.58784]
 [ 0.77179  0.97569  0.02522  0.96234  0.16304  0.59566]
 [ 0.76567  0.97238  0.02313  0.96523  0.15799  0.58625]
 [ 0.65983  0.97903  0.02506  0.97963  0.1615   0.43538]
 [ 0.65467  0.95129  0.02616  0.9775   0.16569  0.4286 ]
 [ 0.68401  1.       0.02383  0.97771  0.15805  0.46786]
 [ 0.67364  0.9736   0.02449  0.97958  0.1611   0.4538 ]
 [ 0.25194  0.22503  0.92848  0.68762  0.95668  0.06348]
 [ 0.25729  0.23672  0.82075  0.69762  0.89843  0.0662 ]
 [ 0.242    0.20826  1.       0.68379  1.       0.05857]
 [ 0.25649  0.23616  0.82368  0.71432  0.90198  0.06579]]
k-means----------------------------------------
[[ 0.25193  0.22654  0.89323  0.69584  0.93927  0.06351]
 [ 0.9914   0.95508  0.02855  0.76832  0.17032  0.98293]
 [ 0.66804  0.97598  0.02488  0.9786   0.16159  0.44641]
 [ 0.6336   0.75462  0.06934  0.99729  0.25184  0.40168]
 [ 0.76987  0.9526   0.03306  0.97515  0.17768  0.59277]
 [ 0.57887  0.44074  0.34519  0.72793  0.58844  0.34208]]
[5 5 5 5 1 1 1 1 4 3 4 3 4 4 4 4 2 2 2 2 0 0 0 0] End of script
```

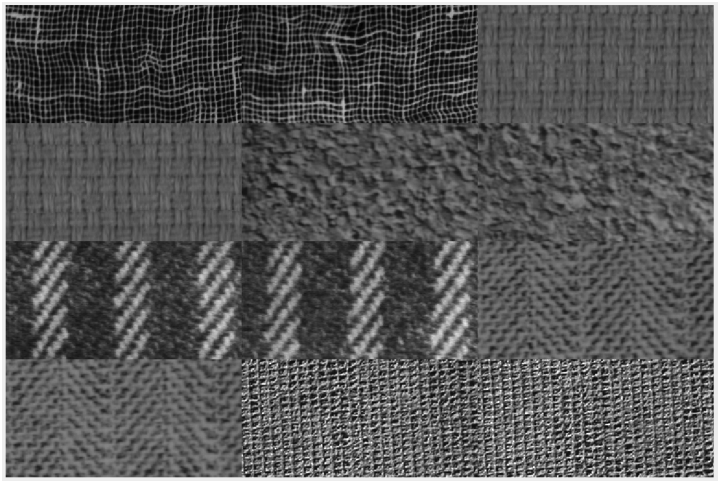
According to this results, it seems to be that most useful features in GLCM for this case are contrast, correlation and ASM, then lets iterate using this features and changing for several step values in the GLCM. The results are as following:

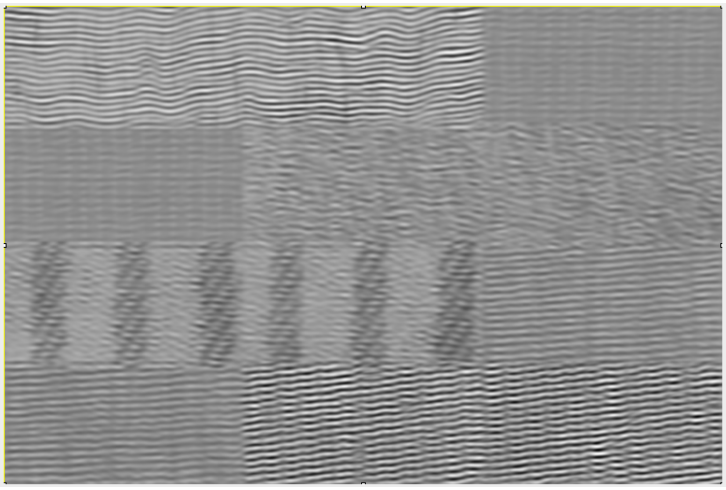
Using step= 1 [3 3 5 5 0 0 0 0 2 4 2 4 3 3 3 3 4 4 4 4 1 1 1 1]
hlUsing step= 2 [0 0 1 1 3 3 3 3 2 2 2 2 4 4 4 4 4 4 4 4 5 5 5 5]
Using step= 3 [2 0 0 0 4 4 4 4 1 1 1 1 2 2 2 2 3 3 3 3 5 5 5 5]
Using step= 4 [2 2 2 4 1 1 1 1 3 0 3 0 4 4 4 4 2 2 2 2 5 5 5 5]
Using step= 5 [0 0 0 0 4 4 4 4 1 1 1 1 2 2 2 2 3 3 3 3 5 5 5 5]
Using step= 6 [0 0 0 0 2 2 2 2 0 0 0 0 3 3 3 3 5 5 5 5 4 4 1 4]
Using step= 7 [0 4 0 0 4 4 4 4 3 3 3 3 2 2 2 2 2 2 2 2 1 5 1 5]
Using step= 8 [4 4 4 4 5 0 5 5 4 4 4 4 1 1 1 1 2 2 2 2 3 3 3 3]
Using step= 9 [3 3 3 3 4 0 1 1 3 3 3 3 2 2 2 2 5 5 5 5 3 3 3 3]
Using step= 10 [1 1 4 4 0 0 0 0 2 1 2 2 5 5 5 5 4 4 4 4 3 3 3 3]
Using step= 11 [0 0 0 5 3 3 3 3 0 0 4 0 1 1 1 1 5 5 5 5 2 2 2 2]
Using step= 12 [0 0 0 0 1 1 1 1 3 3 0 3 2 2 2 2 3 2 3 3 5 4 5 5]
Using step= 13 [0 0 0 0 2 2 2 2 5 5 5 5 3 3 3 3 3 3 3 3 4 1 4 4]
Using step= 14 [5 5 5 5 3 3 3 3 0 1 0 1 5 5 5 5 2 2 2 2 4 4 4 4]
Using step= 15 [5 5 5 5 3 3 3 3 5 5 5 5 4 4 4 4 2 2 2 2 1 0 0 0]
Using step= 16 [4 4 4 4 3 3 3 3 4 4 4 4 0 0 2 2 5 5 5 5 1 1 1 1]
Using step= 17 [4 4 4 4 3 3 3 3 0 1 0 1 3 3 3 3 5 5 5 5 2 2 2 2]
Using step= 18 [2 2 2 2 0 0 0 0 1 1 1 1 5 5 5 5 2 2 2 3 4 4 4 4]
Using step= 19 [1 1 0 1 3 3 3 3 4 0 4 0 1 4 4 4 5 5 5 5 2 2 2 2]

This results make clear that best step values for computing GLCM are between 2 and 8 pixels, but still this results are not always accurate.

Even when results until this moment are quite good, I'd like to give it a try using frequency domain tools to make it a bit more accurate.

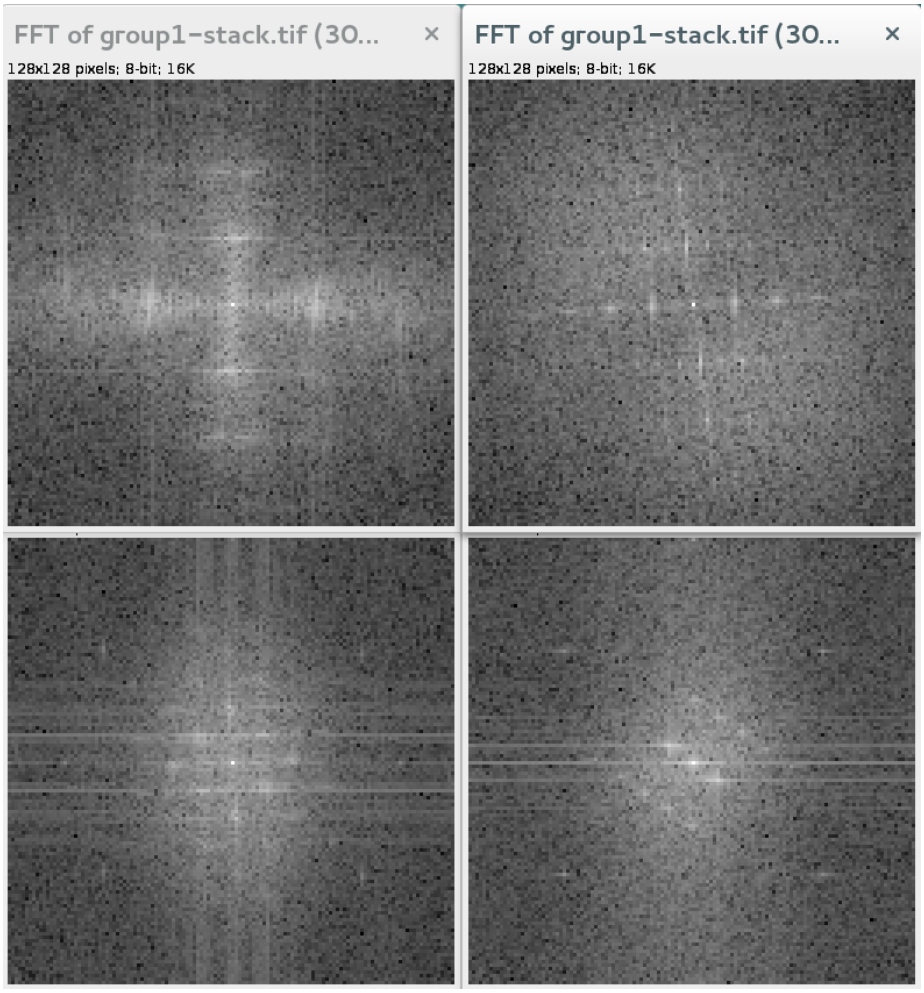
First I thought about applying Gabor filters, so I tried first in ImageJ getting this as result using orientation 0 and scale 4:





The problem with this Gabor filters is that filtered images look even more similar than original ones. Maybe it could be used using a bank of Gabor filters, but I rather giving a try to Fast Fourier Transform.

Let's try now using FFT first with ImageJ so we can have an idea of results:



This are FFT for different textures, it looks a bit more promising, now I need a way to include this features into k-means data, then I will get the mean and standard deviation from every image.

In order to get this new features (and with the intention of using a different library), I will install python-opencv and get this features like this:

```
import cv2
meanStdDevFFT = cv2.meanStdDev(img_gray)
meanFFT, stdsFFT = meanStdDevFFT[0][0][0], meanStdDevFFT[1][0][0]
```

Using this new features we get this stack:

['stdsFFT', 'stds', 'correlation', 'meanFFT', 'contrast', 'mean']

And iterating for different values of GLCM-step, I got the following results after k-means clustering:

Using step= 1
[0 0 0 0 2 2 2 2 1 4 1 4 3 3 3 3 1 1 1 3 5 5 5 5]
Using step= 2
[2 2 2 2 5 5 5 5 1 1 1 1 4 4 4 4 0 0 0 4 3 3 3 3]
Using step= 3
[5 3 5 5 2 2 2 2 4 4 4 4 1 1 1 1 1 1 1 1 0 0 0 0]
Using step= 4
[0 0 0 0 5 5 5 5 0 0 0 0 3 3 4 4 2 2 2 2 1 1 1 1]
Using step= 5
[1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 3 4 3 2 5 5 5 5]
Using step= 6
[3 5 3 5 1 1 1 1 2 2 2 2 4 4 4 4 1 1 1 1 0 0 0 0]
Using step= 7
[0 0 0 0 2 2 2 2 4 3 4 3 1 1 1 1 2 1 2 2 5 5 5 5]
Using step= 8
[3 1 3 1 0 0 0 0 3 3 3 3 2 2 2 2 4 2 4 4 5 5 5 5]
Using step= 9
[0 3 0 3 5 5 5 5 2 2 2 2 5 5 5 5 5 5 5 5 4 4 1 4]

It's important to say that for this last trials I increased the number of kmeans iterations from 100 to 1000, this is one default parameter into the scipy.cluster.kmeans library.

```
from scipy.cluster.vq import kmeans2
kmeansRes = kmeans2(features\_norm,6, minit='points', iter=1000)
```

Where 6 is the number of classes to classify and minit='points' is the way how the library is going to place the random initial positions for clusters.

Finally using glcm step with value 6 and running same script 10 times I got the following results:

Using step= 6
[5 5 5 5 2 2 2 2 2 1 1 1 1 0 0 0 0 4 4 4 4 4 3 3 3 3]
[0 0 0 0 4 4 4 4 4 1 1 1 1 5 5 5 5 2 2 2 2 2 3 3 3 3]
[4 4 4 4 5 5 5 5 5 3 3 3 3 0 0 0 0 2 2 2 2 2 1 1 1 1]
[1 1 1 1 2 2 2 2 2 0 0 0 0 3 3 3 3 3 3 3 3 3 5 4 5 4]
[4 4 4 4 5 5 5 5 5 3 3 3 3 0 0 0 0 2 2 2 2 2 1 1 1 1]
[5 5 5 5 3 3 3 3 3 4 4 4 4 1 1 1 1 0 0 0 0 0 2 2 2 2]
[5 5 5 5 1 1 1 1 1 0 0 0 0 2 2 2 2 2 2 2 2 2 4 3 4 3]
[4 4 4 4 1 1 1 1 1 0 0 0 0 3 3 3 3 5 5 5 5 5 2 2 2 2]
[1 1 1 1 2 2 2 2 2 0 0 0 0 3 3 3 3 3 3 3 3 3 5 4 5 4]
[5 5 5 5 2 2 2 2 2 1 1 1 1 0 0 0 0 4 4 4 4 4 3 3 3 3]

If we compute the classification ratio for this examples, considering 10 trials it means $10 * 24 = 240$ and counting errors manually $errors = 18$, then the classification ratio is:

$$classRatio = 100 - (18/240) * 100 = 100–7.5 = 92.5\%$$

4 Python script Code

```
1
2  import matplotlib.pyplot as plt
3  import numpy as np
4  import skimage.io as ski_io
5  import skimage.feature as ski_feature
6  from skimage import color
7  import os
8  from scipy.cluster.vq import vq, kmeans, whiten, kmeans2
9  import cv2
10
11 # Directory for reading
12 inputDir = "Images Exer2/grup1/"
13 listFiles = os.listdir(inputDir)
14 listFiles.sort()
15
16 # Declare step value for GLCM
17 glcmStep = 6
18 print "Using step= %d" % glcmStep
19
20 # Go over all images into directory for 10 times
21 for i in range(1,10):
22     features = []
23     for fileName in listFiles:
24         I = ski_io.imread(inputDir + fileName);
25         Icv = cv2.imread(inputDir + fileName);
26
27         meanStdDev = cv2.meanStdDev(Icv)
28         mean, stds = meanStdDev[0][0][0], meanStdDev[1][0][0]
29
30         # Get GLCM matrix
31         GLCM = ski_feature.greycomatrix(I, [glcmStep], [0], levels
       =256, symmetric=True, normed=True)
32
33         # Get fft in gray values
34         f = np.fft.fft2(I)
35         fshift = np.fft.fftshift(f)
36         imgAbs = np.abs(fshift)
37         img_gray = color.rgb2gray(imgAbs)
38
39         # Get mean and std dev from fft gray image
40         meanStdDevFFT = cv2.meanStdDev(img_gray)
41         meanFFT, stdsFFT = meanStdDevFFT[0][0][0], meanStdDevFFT
       [1][0][0]
42
43         # Join properties from GLCM and FFT
44         props = {}
45         props['meanFFT'] = meanFFT
46         props['contrast'] = ski_feature.greycoprops(GLCM, 'contrast'
       )[0][0]
47
48         features.append(props.values())
49
50     # Transform features to numpy array
51     features = np.asarray(features)
52
```

```
53      # Normalize features
54      features_norm = features / features.max(axis=0)
55
56      # Change python print options and print features
57      np.set_printoptions(precision=4, suppress=True)
58      print props.keys(), "\n", features
59
60      # Apply k-means clustering
61      kmeansRes = kmeans2(features_norm,6, minit='points', iter
        =100000)
62      # Print kmeans grouping
63      # Note: kmeansRes[0] holds all final Centroids
64      print kmeansRes[1]
```

5 Comments and conclusions

Before starting with this project I had no idea of how important textures were into the world of image processing, now I have some key concepts that make me realize how useful textures can be for classify and segment images into pieces in order to know what is this image about. Using textures, the computer can tell if a part of an image is a piece of wall, grass, sky, road, or even a person or an animal, this knowledge has an impressive utility, and gives the opportunity to make decisions in real time according to what the camera is seeing. A good example of what computer vision segmentation is capable of, is the new self-driving cars technology.

References

[1] Linda g. shapiro and george c. stockman, computer vision, upper saddle river: Prentice–hall, 2001.

[2] Mathworks, texture analysis.

[3] Murphylab, haralick texture features.

[4] Robert m haralick; k shanmugam; its'hak dinstein (1973).

[5] Wikipedia, cluster analysis.

[6] Wikipedia, k-means clustering.