function featuresExtraction(myfolder)

% featuresExtraction(folder)

% Calculates all the features for all the images in the folder 'myfolder',

% and saves them in a .mat file called 'features\_myfolder',

% which contains a matrix with the features for all our images 'collected\_features', and a matrix 'names' containing the list of names of the pictures analysed.

%

% /!\ We will need to load the Laplacian images Mp and Ms used with the function edgeDistance

% You first have to calculate those images using generate\_mean and save

% those two matrices in a file called laplacianMean.mat

%list=ls('myfolder') %just used to print a clear list of the image in the file

list=dir(myfolder)

nb\_data=size(list,1)-2;

names=cell(1,nb\_data);

% initialization of the features

fprintf('Initialization of the features \n');

h=zeros(nb\_data,1);

s=zeros(nb\_data,1);

v=zeros(nb\_data,1);

s\_=zeros(nb\_data,1);

l\_=zeros(nb\_data,1);

hist\_distance=zeros(nb\_data,1);

emd\_distance=zeros(nb\_data,1);

freq\_hue=zeros(nb\_data,1);

dev\_color=zeros(nb\_data,1);

nbHue=zeros(nb\_data,1);

hueContrast=zeros(nb\_data,1);

missingHue=zeros(nb\_data,1);

missingContrast=zeros(nb\_data,1);

maxPixel=zeros(nb\_data,1);

hue\_count=zeros(nb\_data,1);

modelDistance=[];

hue\_model=zeros(nb\_data,1);

arithmBrightness=zeros(nb\_data,1);

logarithBrightness=zeros(nb\_data,1);

brightnessContrast=zeros(nb\_data,1);

contrast\_quality=zeros(nb\_data,1);

bounding\_area\_ratio=zeros(nb\_data,1);

edge\_quality=zeros(nb\_data,1);

bounding\_quality=zeros(nb\_data,1);

sum\_edges=zeros(nb\_data,1);

texture\_range=zeros(nb\_data,1);

texture\_deviation=zeros(nb\_data,1);

entropy\_r=zeros(nb\_data,1);

entropy\_g=zeros(nb\_data,1);

entropy\_b=zeros(nb\_data,1);

TextureH1=zeros(nb\_data,1);

TextureH2=zeros(nb\_data,1);

TextureH3=zeros(nb\_data,1);

TextureS1=zeros(nb\_data,1);

TextureS2=zeros(nb\_data,1);

TextureS3=zeros(nb\_data,1);

TextureV1=zeros(nb\_data,1);

TextureV2=zeros(nb\_data,1);

TextureV3=zeros(nb\_data,1);

TextureAvgH=zeros(nb\_data,1);

TextureAvgS=zeros(nb\_data,1);

TextureAvgV=zeros(nb\_data,1);

Low\_DOFH=zeros(nb\_data,1);

Low\_DOFS=zeros(nb\_data,1);

Low\_DOFV=zeros(nb\_data,1);

global\_blur=zeros(nb\_data,1);

h3=zeros(nb\_data,1);

s3=zeros(nb\_data,1);

v3=zeros(nb\_data,1);

focus\_hue=zeros(nb\_data,1);

focus\_saturation=zeros(nb\_data,1);

focus\_lightness=zeros(nb\_data,1);

XY\_100\_=zeros(nb\_data,1);

numb\_conncomp=zeros(nb\_data,1);

AvgH=[];

AvgS=[];

AvgV=[];

SI\_XY\_=[];

Centroid=[];

Color\_spread=zeros(nb\_data,1);

Complem\_colors=zeros(nb\_data,1);

Convexity=zeros(nb\_data,1);

Centroid\_x=[];

Centroid\_y=[];

Shape\_variance=[];

Shape\_skewness=[];

Segment\_brightness=[];

Hue\_contrast=zeros(nb\_data,1);

Saturation\_contrast=zeros(nb\_data,1);

Brightness\_contrast=zeros(nb\_data,1);

Blur\_contrast=zeros(nb\_data,1);

%Loading the Laplacian images Mp and Ms used with the function edgeDistance

%You first have to calculate those images using laplacianMean and save

%those two matrices in a file called laplacianMean.mat

fprintf('Loading laplacianMean');

load laplacianMean

% /!\ the for loop starts at 3 (4 for mac)because list(1).name='.' and list(2).name='..'

fprintf('Feature extraction \n');

for image=3:size(list)

 tic

 i=image-2;

 string=strcat(myfolder,'/',list(image).name)

 names{i}=list(image).name;

 Irgb=imread(string);

 Irgb=im2double(Irgb);

 Ir=Irgb(:,:,1);

 Ig=Irgb(:,:,2);

 Ib=Irgb(:,:,3);

 Ihsv=rgb2hsv(Irgb);

 Ih=Ihsv(:,:,1);

 Is=Ihsv(:,:,2);

 Iv=Ihsv(:,:,3);

 Ihsl=rgb2hsl(Irgb);

 Ih\_=Ihsl(:,:,1);

 Is\_=Ihsl(:,:,2);

 Il\_=Ihsl(:,:,3);

 %average hue, saturation, value (HSV)

 h(i)=mean2(Ih);

 s(i)=mean2(Is);

 v(i)=mean2(Iv);

 %average saturation and lightness (HSL)

 s\_(i)=mean2(Is\_);

 l\_(i)=mean2(Il\_);

 %colorfulness

 [hist\_distance(i),emd\_distance(i)]=rgbCubes(Ir,Ig,Ib);

 %most frequent hue and standard deviation of colorfulness

 Ihb=findReplace(Ih,Ihsl);

 idx=Ihb>0;

 freq\_hue(i)=mode(mode(Ihb(idx)));

 dev\_color(i)=std(var(Ihb));

 %Number of distinct hues present and missing in the image, their

 %contrast and number of pixels that belong to the most frequent hue

 n=20;

 C=0.1;

 c=0.01;

 [~, nbHue(i), hueContrast(i), missingHue(i), missingContrast(i), maxPixel(i)]=hueHistogram(Ihsl,n,C,c);

 %Hue count (re-using hueHistogram with a different color space and C)

 n=20;

 C=0.05;

 c=0.01;

 [~,Hue\_count,~,~,~,~]=hueHistogram(Ihsv,n,C,c);

 hue\_count(i)=n-Hue\_count;

 fprintf('Hue model fitting \n');

 %hue models fitting

 model\_threshold=10;

 modelNormalDistance=zeros(9,1);

 for k=1:9

 g=@(alpha)hueModel(Ih\_,Is\_,alpha,k);

 alpha0=fminbnd(g,0,360);

 [normalizedDistance]=hueModel(Ih\_,Is\_,alpha0,k);

 modelNormalDistance(k,1)=normalizedDistance;

 end

 modelDistance=[modelDistance;transpose(modelNormalDistance)];

 model=modelNormalDistance.\*(modelNormalDistance<model\_threshold);

 if sum(model)>0

 hue\_model(i)=max(find(model));

 else

 [~,hue\_model(i)]=min(modelNormalDistance);

 end

 % arithmetic and logarithmic average brightness + brightness contrast

 [arithmBrightness(i),logarithBrightness(i),brightnessContrast(i)]=brightness(Ir,Ig,Ib);

 %Contrast quality

 [contrast\_quality(i)]=brightnessHistogram(Ir,Ig,Ib);

 fprintf('Getting bounding\_area\_ratio \n');

 %Edge distribution metric

 [bounding\_area\_ratio(i)]=bounding\_box(Ir,Ib,Ig);

 %spatial distribution of the high frequency edges and area of the bounding box

 [edge\_quality(i),bounding\_quality(i)]=edgeDistance(Ir,Ig,Ib,Mp,Ms);

 %sum of edges

 sum\_edges(i)=edgeSum(Ir,Ig,Ib);

 %range of texture

 a=rangefilt(Ihsv);

 texture\_range(i)=sum(mean2(a))/3;

 %standard deviation of texture

 b=stdfilt(Ihsv);

 texture\_deviation(i)=sum(mean2(b))/3;

 %entropy of the red, green and blue matrices

 entropy\_r(i)=entropy(Ir);

 entropy\_g(i)=entropy(Ig);

 entropy\_b(i)=entropy(Ib);

 fprintf('Getting the wavelet related features \n');

 %wavelet related features

 [texture,low\_DOF]=waveletTexture(Ihsl);

 TextureH1(i)=texture(1,1);

 TextureH2(i)=texture(2,1);

 TextureH3(i)=texture(3,1);

 TextureS1(i)=texture(1,2);

 TextureS2(i)=texture(2,2);

 TextureS3(i)=texture(3,2);

 TextureV1(i)=texture(1,3);

 TextureV2(i)=texture(2,3);

 TextureV3(i)=texture(3,3);

 TextureAvgH(i)=texture(4,1);

 TextureAvgS(i)=texture(4,2);

 TextureAvgV(i)=texture(4,3);

 Low\_DOFH(i)=low\_DOF(1);

 Low\_DOFS(i)=low\_DOF(2);

 Low\_DOFV(i)=low\_DOF(3);

 fprintf('Getting the blur feature \n');

 %blur measure

 [global\_blur(i)]=gaussian\_blur(Ir,Ig,Ib);

 %averages from rule of thirds

 margin=0;

 h3(i)=thirdsAvg(Ih,margin);

 s3(i)=thirdsAvg(Is,margin);

 v3(i)=thirdsAvg(Iv,margin);

 %Average hue, saturation and lightness of the focus region

 margin=0.1;

 focus\_hue(i)=thirdsAvg(Ih\_,margin);

 focus\_saturation(i)=thirdsAvg(Is\_,margin);

 focus\_lightness(i)=thirdsAvg((Ir+Ig+Ib)/3,margin);

 fprintf('Getting all the segmentation related features \n');

 %all the region composition features (ie the segmentation related

 %features)

 k=2;

 m=1;

 [nb\_cc,avgH,avgS,avgV,XY\_100,SI\_XY,centroid,color\_spread,complem\_colors,convexity,centroid\_x,centroid\_y,shape\_variance,shape\_skewness,lightness,hue\_contrast,saturation\_contrast,brightness\_contrast,blur\_contrast]=seg1(Irgb,k,m);

 XY\_100\_(i)=XY\_100;

 numb\_conncomp(i)=nb\_cc;

 AvgH=[AvgH; transpose(avgH)];

 AvgS=[AvgS; transpose(avgS)];

 AvgV=[AvgV; transpose(avgV)];

 SI\_XY\_=[SI\_XY\_; transpose(SI\_XY)];

 Color\_spread(i)=color\_spread;

 Complem\_colors(i)=complem\_colors;

 Centroid=[Centroid; transpose(centroid)];

 Convexity(i)=convexity;

 Centroid\_x=[Centroid\_x; transpose(centroid\_x)];

 Centroid\_y=[Centroid\_y; transpose(centroid\_y)];

 Shape\_variance=[Shape\_variance; transpose(shape\_variance)];

 Shape\_skewness=[Shape\_skewness; transpose(shape\_skewness)];

 Segment\_brightness=[Segment\_brightness; transpose(lightness)];

 Hue\_contrast(i)=hue\_contrast;

 Saturation\_contrast(i)=saturation\_contrast;

 Brightness\_contrast(i)=brightness\_contrast;

 Blur\_contrast(i)=blur\_contrast;

 toc

end

collected\_features=[h s v s\_ l\_ hist\_distance emd\_distance freq\_hue dev\_color nbHue missingHue hueContrast missingContrast maxPixel hue\_count modelDistance hue\_model arithmBrightness logarithBrightness brightnessContrast contrast\_quality bounding\_area\_ratio bounding\_quality edge\_quality sum\_edges texture\_range texture\_deviation entropy\_r entropy\_g entropy\_b TextureH1 TextureH2 TextureH3 TextureS1 TextureS2 TextureS3 TextureV1 TextureV2 TextureV3 TextureAvgH TextureAvgS TextureAvgV global\_blur h3 s3 v3 focus\_hue focus\_saturation focus\_lightness numb\_conncomp XY\_100\_ SI\_XY\_ Centroid AvgH AvgS AvgV Segment\_brightness Color\_spread Complem\_colors Centroid\_x Centroid\_y Shape\_variance Shape\_skewness Convexity Hue\_contrast Saturation\_contrast Brightness\_contrast Blur\_contrast Low\_DOFH Low\_DOFS Low\_DOFV];

save(strcat('features\_','Carysfort'),'collected\_features','names');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [bounding\_area\_ratio]=bounding\_box(Ir,Ib,Ig)

% [bounding\_area\_ratio]=bounding\_box(Ir,Ib,Ig)

% Considering a picture whose red, green and blue channels are given in Ir,

% Ig and Ib, edgeDistance computes its laplacian image, and returns the

% ratio of the area of the bounding box containing 81% of the edge energy,

% and the area of the photo

h = fspecial('laplacian', 0.2);

IR=abs(imfilter(Ir,h,'replicate'));

IG=abs(imfilter(Ig,h,'replicate'));

IB=abs(imfilter(Ib,h,'replicate'));

edge\_img=(IR+IG+IB)/3;

percentage=0.9;

[a1,b1]=find\_energy(percentage,sum(edge\_img,2),size(edge\_img,1));

[a2,b2]=find\_energy(percentage,sum(edge\_img,1),size(edge\_img,2));

bounding\_area\_ratio=(b1-a1+1)\*(b2-a2+1)/(size(edge\_img,1)\*size(edge\_img,2));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ arithmBrightness,logarithmBrightness, brightnessContrast ] = brightness( Ir,Ig,Ib )

%[ arithmBrightness,logarithmBrightness, brightnessContrast ] = brightness( Ir,Ig,Ib )

% Calculates the average arithmetic brightness, the average logarithmic

% brightness, and a brightness contrast using a 100 bins histogram

% note: need Ir,Ig,Ib to have values in [0;1]

%arithmetic average brightness

M=(Ir+Ig+Ib)/3;

arithmBrightness=mean2(M);

%logarithmic average brightness + brightness histogram

sum=0;

epsilon=0.001;

B=zeros(100,1);

N=100\*M;

for m=1:size(Ir,1)

 for n=1:size(Ir,2)

 sum = sum + log(epsilon + M(m,n));

 if (N(m,n)~=100)

 B(floor(N(m,n))+1)=B(floor(N(m,n))+1)+1;

 else

 B(100)=B(100)+1;

 end

 end

end

logarithmBrightness=exp(sum/(size(Ir,1)\*size(Ir,2)));

[C,I]=max(B);

area=C;

a=I;

b=I;

nbPixel=size(Ir,1)\*size(Ir,2);

while (area/nbPixel)<0.98

 if a>1

 a=a-1;

 area=area+B(a);

 end

 if b<100

 b=b+1;

 area=area+B(b);

 end

end

brightnessContrast=b-a+1;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [contrast\_quality]=brightnessHistogram(Ir,Ig,Ib)

%[contrast\_quality]=brightnessHistogram(Ir,Ig,Ib)

% Returns the width of the smallest region containing 0.98percent of the

% brightness histogram of the image

Ir=255\*Ir;

Ig=255\*Ig;

Ib=255\*Ib;

Hr=zeros(256,1);

Hg=zeros(256,1);

Hb=zeros(256,1);

for i=1:size(Ir,1)

 for j=1:size(Ir,2)

 Hr(1+floor(Ir(i,j)))=Hr(1+floor(Ir(i,j)))+1;

 Hg(1+floor(Ig(i,j)))=Hg(1+floor(Ig(i,j)))+1;

 Hb(1+floor(Ib(i,j)))=Hb(1+floor(Ib(i,j)))+1;

 end

end

H=(Hr+Hg+Hb)/(size(Ir,1)\*size(Ir,2));

percentage=0.98;

[a,b]=find\_energy(percentage,H,256);

contrast\_quality=b-a;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [IH]=context(Ih,m)

% [IH]=context(Ih,m)

% Performs a uniform blur on Ih, but independantly of its size, with

% ones(m) as a kernel matrix

% Ih must be a 2D image, m an integer >0

 if m>1

 if size(Ih,1)>size(Ih,2)

 m1=floor(m\*size(Ih,1)/3072);

 m2=floor(m\*size(Ih,2)/2304);

 else

 m1=floor(m\*size(Ih,1)/2304);

 m2=floor(m\*size(Ih,2)/3072);

 end

 %sums the i,...,i+m1-1 lines of Ih

 sum\_Ih=[zeros(m1-1,size(Ih,2));Ih];

 for i=1:m1-1

 sum\_Ih=sum\_Ih+[zeros(m1-i-1,size(Ih,2));Ih;zeros(i,size(Ih,2))];

 end

 %sums the i,...,i+m2-1 columns of sum\_Ih

 sum\_Ih2=[zeros(size(Ih,1)+m1-1,m2-1),sum\_Ih];

 for i=1:m2-1

 sum\_Ih2=sum\_Ih2+[zeros(size(Ih,1)+m1-1,m2-i-1) sum\_Ih zeros(size(Ih,1)+m1-1,i)];

 end

 %suppress the additional lines and columns

 for i=1:m1-1

 sum\_Ih2(1,:)=[];

 sum\_Ih2(size(sum\_Ih2,1),:)=[];

 end

 for i=1:m2-1

 sum\_Ih2(:,1)=[];

 sum\_Ih2(:,size(sum\_Ih2,2))=[];

 end

 %we have added the value of m1\*m2 pixels, so to have an average, we have to

 %divide it by m1\*m2

 IH=sum\_Ih2/(m1\*m2);

 else

 IH=Ih;

 end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [edge\_quality,bounding\_quality]=edgeDistance(Ir,Ig,Ib,Mp,Ms)

% [edge\_quality,bounding\_quality]=edgeDistance(Ir,Ig,Ib,Mp,Ms)

%

% Considering a picture whose red, green and blue channels are given in Ir,

% Ig and Ib, edgeDistance computes its normalized 100x100 laplacian image,

% and compares it to Mp and Ms.

% Mp and Ms have to be to be two normalized 100x100 laplacian images also

% (they are supposed to be the mean Laplacian image of the good (resp. bad)

% photos). They can be constructed using the function laplacianMean.

% edge\_quality=ds-dp , where ds (resp dp) is the distance between this

% laplacian image and Ms (resp Mp)

%

% bounding\_quality is 1-the area of the bounding box containing 96.04% of

% the edge energy

%Mp and Ms are the mean Laplacian image of the good and bad photos, respectively

[laplacian\_img]=laplacianImage(Ir,Ig,Ib);

DP=laplacian\_img-Mp;

DS=laplacian\_img-Ms;

dp=norm(DP(:),1);

ds=norm(DS(:),1);

edge\_quality=ds-dp;

percentage=0.98;

[a1,b1]=find\_energy(percentage,sum(laplacian\_img,2),size(laplacian\_img,1));

[a2,b2]=find\_energy(percentage,sum(laplacian\_img,1),size(laplacian\_img,2));

bounding\_quality=1-(b1-a1+1)\*(b2-a2+1)/10000;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ a ] = edgeSum( Ir,Ig,Ib)

%[ a ] = edgeSum( Ir,Ig,Ib)

% x=sum(sum(edge(Ir)))/(size(Ir,1)\*size(Ir,2));

% y=sum(sum(edge(Ig)))/(size(Ig,1)\*size(Ig,2));

% z=sum(sum(edge(Ib)))/(size(Ib,1)\*size(Ib,2));

%

% a=((x+y+z)/3);

x=sum(sum(edge(Ir)))/(size(Ir,1)\*size(Ir,2));

y=sum(sum(edge(Ig)))/(size(Ig,1)\*size(Ig,2));

z=sum(sum(edge(Ib)))/(size(Ib,1)\*size(Ib,2));

a=((x+y+z)/3);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [distance2]=emd(H,D,n)

nbBox=n^3;

nbBox2=nbBox\*nbBox;

A=zeros(2\*nbBox,nbBox2);

f=zeros(1,nbBox2);

for i=1:nbBox

 for j=1:nbBox

 A(i,j+nbBox\*(i-1))=1;

 A(i+nbBox,i+nbBox\*(j-1))=1;

 f(1,j+nbBox\*(i-1))=D(i,j);

 end

end

b=ones(2\*nbBox,1)/nbBox;

for i=1:nbBox

 b(i)=H(i);

end

Aeq=ones(1,nbBox2);

beq=min( norm(H,1),1);

lb=zeros(1,nbBox2);

ub=[];

[~,distance2] = linprog(f,A,b,Aeq,beq,lb,ub);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [a,b]=find\_energy(percentage,vector,n)

%finds the indice a and b, where b-a is the smallest distance with still

%sum\_{k=a}^{b}vector(k)>0.9\*sum\_{k=1}^{size(vector)}vector(k)

%/!\ n is the size of vector

if percentage>1 || percentage<0

 error('percentage must be in [0,1]');

end

total=norm(vector,1);

inverse\_percent=(1-percentage)\*total;

left\_limit=0;

content=0;

while content<inverse\_percent

 left\_limit=left\_limit+1;

 content=content+vector(left\_limit);

end

right\_limit=0;

content=0;

while content<inverse\_percent

 content=content+vector(n-right\_limit);

 right\_limit=right\_limit+1;

end

a=1;

b=n;

percent=percentage\*total;

for i=1:left\_limit

 for j=0:right\_limit-1

 s=0;

 for k=i:n-j

 s=s+vector(k);

 end

 if (s>=percent)&&(n-j-i<b-a)

 a=i;

 b=n-j;

 end

 end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ Ih ] = findReplace( Ih, Ihsl )

%[ Ih ] = findReplace( Ih, Ihsl )

%Returns the matrix Ih, where all the pixels with a saturation lower than

%0.2 or a lightness out of ]0.15,0.95[ have been set to 0.

% if Ihsl(i,j,2)<=0.2 || (Ihsl(i,j,3)>=0.95 || Ihsl(i,j,3)<=0.15)

% Ih(i,j)=0;

% end

for i=1:size(Ih,1)

 for j=1:size(Ih,2)

 if Ihsl(i,j,2)<=0.2 || (Ihsl(i,j,3)>=0.95 || Ihsl(i,j,3)<=0.15)

 Ih(i,j)=0;

 end

 end

end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [blur]=gaussian\_blur(Ir,Ig,Ib)

% [blur]=gaussian\_blur(Ir,Ig,Ib)

% using theta=0.45

% blur is in [-1; 0], -1 being completely blurred, and 0 a very sharp image

I\_blurred=(Ir+Ig+Ib)/3;

M=size(I\_blurred,1);

N=size(I\_blurred,2);

Y=fft2(I\_blurred)/sqrt(M\*N);

theta=0.45;

abs\_Y=abs(Y);

[row,column]=find(abs\_Y>theta);

select\_row=row<M/2;

select\_column=column<N/2;

m=max(row.\*select\_row);

n=max(column.\*select\_column);

blur=max(2\*(m-floor(M/2))/M,2\*(n-floor(N/2))/N);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function rgb=hsl2rgb(hsl\_in)

%Converts Hue-Saturation-Luminance Color value to Red-Green-Blue Color value

%

%Usage

% RGB = hsl2rgb(HSL)

%

% converts HSL, a M [x N] x 3 color matrix with values between 0 and 1

% into RGB, a M [x N] X 3 color matrix with values between 0 and 1

%

%See also rgb2hsl, rgb2hsv, hsv2rgb

% (C) Vladimir Bychkovsky, June 2008

% written using:

% - an implementation by Suresh E Joel, April 26,2003

% - Wikipedia: http://en.wikipedia.org/wiki/HSL\_and\_HSV

hsl=reshape(hsl\_in, [], 3);

H=hsl(:,1);

S=hsl(:,2);

L=hsl(:,3);

lowLidx=L < (1/2);

q=(L .\* (1+S) ).\*lowLidx + (L+S-(L.\*S)).\*(~lowLidx);

p=2\*L - q;

hk=H; % this is already divided by 360

t=zeros([length(H), 3]); % 1=R, 2=B, 3=G

t(:,1)=hk+1/3;

t(:,2)=hk;

t(:,3)=hk-1/3;

underidx=t < 0;

overidx=t > 1;

t=t+underidx - overidx;

range1=t < (1/6);

range2=(t >= (1/6) & t < (1/2));

range3=(t >= (1/2) & t < (2/3));

range4= t >= (2/3);

% replicate matricies (one per color) to make the final expression simpler

P=repmat(p, [1,3]);

Q=repmat(q, [1,3]);

rgb\_c= (P + ((Q-P).\*6.\*t)).\*range1 + ...

 Q.\*range2 + ...

 (P + ((Q-P).\*6.\*(2/3 - t))).\*range3 + ...

 P.\*range4;

rgb\_c=round(rgb\_c.\*10000)./10000;

rgb=reshape(rgb\_c, size(hsl\_in));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [H, nbHue, hueContrast, missingHue, missingContrast, maxPixel]=hueHistogram(Ihsl,n,C,c)

%[H, nbHue, hueContrast, missingHue, missingContrast,maxPixel]=hueHistogram(Ihsl,n,C,c);

%Constructs of the hue histogram H of the image, counts the number of

%significant hues and missing hues, and calculates their contrasts

%Also returns the percentage of pixels belonging to the most frequent hue

%Construction of the histogram

H=zeros(n,1);

nonConsidered=0;

for i=1:size(Ihsl,1)

 for j=1:size(Ihsl,2)

 if ((Ihsl(i,j,2)<=0.2) || (Ihsl(i,j,3)>=0.95 || Ihsl(i,j,3)<=0.15))

 nonConsidered = nonConsidered+1;

 else

 for k=1:n

 if ((Ihsl(i,j,1)>=(k-1)/n) && (Ihsl(i,j,1)<=k/n))

 H(k)=H(k)+1;

 end

 end

 end

 end

end

Q=max(H);

nbHue=0;

missingHue=0;

hueContrast=0;

missingContrast=0;

for k=1:n

 if (H(k)>(C\*Q))

 nbHue=nbHue+1;

 for l=1:n

 if (H(l)>(C\*Q))

 %contrast is the distance between the two hue

 %considered,which are on a wheel

 contrast=min(abs((k-l)/n), 1-abs((k-l)/n));

 hueContrast=max(hueContrast,contrast);

 end

 end

 end

 if (H(k)<(c\*Q))

 missingHue=missingHue+1;

 for l=1:n

 if (H(l)<(c\*Q))

 %contrast is the distance between the two hue

 %considered,which are on a wheel

 contrast=min(abs((k-l)/n), 1-abs((k-l)/n));

 missingContrast=max(missingContrast,contrast);

 end

 end

 end

end

maxPixel=Q/(size(Ihsl,1)\*size(Ihsl,2)-nonConsidered);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [normalizedDistance]=hueModel(Ih, Is, alpha, k)

% [normalizedDistance]=hueModel(Ih, Is, alpha, k)

% /!\ Ih, Is must be defined and their value must be in [0,1]

% /!\ k must be defined in {1;...;9}

%

% Measure how much the image fits in the k-th hue model, rotated with an

% angle of alpha in [0,360]

colorModel=[180 180 180 180 180;

 90 180 270 270 270;

 30 65 145 145 145;

 80 115 145 145 145;

 90 210 240 240 240;

 90 90 90 90 90;

 30 120 150 240 270;

 30 180 210 210 210;

 30 30 30 30 30];

IH=mod(Ih\*360+alpha,360);

distance=0;

IS=0;

for i=1:size(Ih,1)

 for j=1:size(Ih,2)

 if ( IH(i,j)<= colorModel(k,1) )

 nearestBorder=IH(i,j);

 elseif (IH(i,j)>colorModel(k,1))&&(IH(i,j)<colorModel(k,2))

 if IH(i,j)-colorModel(k,1)<=colorModel(k,2)-IH(i,j)

 nearestBorder=colorModel(k,1);

 else

 nearestBorder=colorModel(k,2);

 end

 elseif (IH(i,j)>=colorModel(k,2))&&(IH(i,j)<=colorModel(k,3))

 nearestBorder=IH(i,j);

 elseif (IH(i,j)>colorModel(k,3))&&(IH(i,j)<colorModel(k,4))

 if IH(i,j)-colorModel(k,3)<=colorModel(k,4)-IH(i,j)

 nearestBorder=colorModel(k,3);

 else

 nearestBorder=colorModel(k,4);

 end

 elseif (IH(i,j)>=colorModel(k,4))&&(IH(i,j)<=colorModel(k,5))

 nearestBorder=IH(i,j);

 elseif (IH(i,j)>colorModel(k,5))

 if IH(i,j)-colorModel(k,5)<=360-IH(i,j)

 nearestBorder=colorModel(k,5);

 else

 nearestBorder=360;

 end

 end

 distance = distance + abs( nearestBorder-IH(i,j) )\*Is(i,j);

 IS=IS+Is(i,j);

 end

end

normalizedDistance=distance/IS;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [laplacian\_img]=laplacianImage(Ir,Ig,Ib)

% [laplacian\_img]=laplacianImage(Ir,Ig,Ib)

% Returns the laplacian image of a picture, Ir Ig and Ib being the red,

% green and blue channels of that picture

% This laplacian image size has been resized to 100x100, and the sum of its

% values has been normalized to 1.

h = fspecial('laplacian', 0.2);

IR=abs(imfilter(Ir,h,'replicate'));

IG=abs(imfilter(Ig,h,'replicate'));

IB=abs(imfilter(Ib,h,'replicate'));

edge\_img=(IR+IG+IB)/3;

laplacian\_img=imresize(edge\_img,[100 100]);

laplacian\_img=laplacian\_img/norm(laplacian\_img(:),1);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [M]=laplacianMean(foldername)

% [M]=laplacianMean(foldername)

% Gives the mean across all the 100x100 normalized Laplacian images of the pictures in this folder

%

% /!\ the for loop starts at 3 (should be 4 for mac) because list(1).name='.' and list(2).name='..'

list=dir(foldername);

% /!\ the for loop starts at 3 (should be 4 for mac) because list(1).name='.' and list(2).name='..'

M=zeros(100,100);

for image=3:size(list)

 string=strcat(foldername,'/',list(image).name);

 Irgb=imread(string);

 Irgb=im2double(Irgb);

 Ir=Irgb(:,:,1);

 Ig=Irgb(:,:,2);

 Ib=Irgb(:,:,3);

 [laplacian\_img]=laplacianImage(Ir,Ig,Ib);

 M=M+laplacian\_img;

end

size(list,1)

M=M/(size(list,1)-2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function hsl=rgb2hsl(rgb\_in)

%Converts Red-Green-Blue Color value to Hue-Saturation-Luminance Color value

%

%Usage

% HSL = rgb2hsl(RGB)

%

% converts RGB, a M [x N] x 3 color matrix with values between 0 and 1

% into HSL, a M [x N] X 3 color matrix with values between 0 and 1

%

%See also hsl2rgb, rgb2hsv, hsv2rgb

% (C) Vladimir Bychkovsky, June 2008

% written using:

% - an implementation by Suresh E Joel, April 26,2003

% - Wikipedia: http://en.wikipedia.org/wiki/HSL\_and\_HSV

rgb=reshape(rgb\_in, [], 3);

mx=max(rgb,[],2);%max of the 3 colors

mn=min(rgb,[],2);%min of the 3 colors

L=(mx+mn)/2;%luminance is half of max value + min value

S=zeros(size(L));

% this set of matrix operations can probably be done as an addition...

zeroidx= (mx==mn);

S(zeroidx)=0;

lowlidx=L <= 0.5;

calc=(mx-mn)./(mx+mn);

idx=lowlidx & (~ zeroidx);

S(idx)=calc(idx);

hilidx=L > 0.5;

calc=(mx-mn)./(2-(mx+mn));

idx=hilidx & (~ zeroidx);

S(idx)=calc(idx);

hsv=rgb2hsv(rgb);

H=hsv(:,1);

hsl=[H, S, L];

hsl=round(hsl.\*100000)./100000;

hsl=reshape(hsl, size(rgb\_in));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% RGB2LUV performs RGB to Luv color conversion.

% LUV = RGB2LUV( RGB ) converts the RGB color vectors into LUV. RGB must

% range in (0.0,1.0).

%

% See also LUV2RGB

%

% 6.891 Problem Set 4: Problem 1

% C. Mario Christoudias

function luv = rgb2luv( rgb )

% convert to XYZ

XYZ = [0.4125, 0.3576, 0.1804; ...

 0.2125, 0.7154, 0.0721; ...

 0.0193, 0.1192, 0.9502];

xyz = XYZ \* rgb;

% convert to Luv

luv = xyz;

Yn = 1;

Lt = 0.008856;

Un\_prime = 0.19784977571475;

Vn\_prime = 0.46834507665248;

L0 = xyz(2,:) / Yn;

warning off MATLAB:divideByZero;

constant = xyz(1,:) + 15 \* xyz(2,:) + 3 \* xyz(3,:);

u\_prime = (constant ~= 0) .\* ((4 \* xyz(1,:)) ./ constant) + (constant == 0) \* 4.0;

v\_prime = (constant ~= 0) .\* ((9 \* xyz(2,:)) ./ constant) + (constant == 0) \* 9.0/15.0;

luv(1,:) = (L0 > Lt) .\* (116.0 \* (L0 .^ (1/3)) - 16.0) + (L0 <= Lt) .\* (903.3 \* L0);

luv(2,:) = 13 \* luv(1,:) .\* (u\_prime - Un\_prime);

luv(3,:) = 13 \* luv(1,:) .\* (v\_prime - Vn\_prime);

% be rid of NaNs

luv(find(isnan(luv))) = 0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ hist\_distance, emd\_distance ] = rgbCubes( Ir, Ig, Ib )

%UNTITLED Summary of this function goes here

% Detailed explanation goes here

n=4;

limit=n+1;

n2=n\*n;

nbBox=n^3;

IR=n\*Ir+1;

IG=n\*Ig+1;

IB=n\*Ib+1;

distribution=zeros(n,n,n);

for i=1:size(Ir,1)

 for j=1:size(Ir,2)

 if IR(i,j)==limit

 IR(i,j)=n;

 end

 if IG(i,j)==limit

 IG(i,j)=n;

 end

 if IB(i,j)==limit

 IB(i,j)=n;

 end

 distribution(floor(IR(i,j)),floor(IG(i,j)),floor(IB(i,j)))=distribution(floor(IR(i,j)),floor(IG(i,j)),floor(IB(i,j)))+1;

 end

end

H=zeros(nbBox,1);

Y=ones(nbBox,1)/nbBox;

A=zeros(nbBox,nbBox);

D2=zeros(nbBox,nbBox);

max\_=0;

for i=1:n

 for j=1:n

 for k=1:n

 H(k+n\*(j-1)+n2\*(i-1))=distribution(i,j,k);

 for i2=1:n

 for j2=1:n

 for k2=1:n

 c1=[0.5+(i-1); 0.5+(j-1); 0.5+(k-1)]/n;

 c2=[0.5+(i2-1); 0.5+(j2-1); 0.5+(k2-1)]/n;

 C1=rgb2luv(c1);

 C2=rgb2luv(c2);

 A(k+n\*(j-1)+n2\*(i-1),k2+n\*(j2-1)+n2\*(i2-1))=norm(c1-c2,2);

 D2(k+n\*(j-1)+n2\*(i-1),k2+n\*(j2-1)+n2\*(i2-1))=norm(C1-C2,2);

 max\_=max(max\_,norm(c1-c2,2));

 end

 end

 end

 end

 end

end

H=H/(size(Ir,1)\*size(Ir,2));

A=1-A/max\_;

hist\_distance=sqrt( transpose(H-Y)\*A\*(H-Y) );

emd\_distance=emd(H,D2,n);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function luvim = RGBim2Luv(im)

if size(im,3) ~= 3

 error('im must have three color channels');

end

if ~isa(im,'float')

 im = im2single(im);

end

if (max(im(:)) > 1)

 im = im./255;

end

XYZ = [.4125 .3576 .1804; .2125 .7154 .0721; .0193 .1192 .9502];

Yn = 1.0;

Lt = .008856;

Up = 0.19784977571475;

Vp = 0.46834507665248;

imsiz = size(im);

im = permute(im,[3 1 2]);

im = reshape(im,[3 prod(imsiz(1:2))]);

xyz = reshape((XYZ\*im)',imsiz);

x = xyz(:,:,1);

y = xyz(:,:,2);

z = xyz(:,:,3);

l0 = y./Yn;

l = l0;

l(l0>Lt) = 116.\*(l0(l0>Lt).^(1/3)) - 16;

l(l0<=Lt) = 903.3\*l0(l0<=Lt);

c = x + 15\*y + 3 \* z;

u = 4\*ones(imsiz(1:2),class(im));

v = (9/15)\*ones(imsiz(1:2),class(im));

u(c~=0) = 4\*x(c~=0)./c(c~=0);

v(c~=0) = 9\*y(c~=0)./c(c~=0);

u = 13\*l.\*(u-Up);

v = 13\*l.\*(v-Vp);

luvim = cat(3,l,u,v);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [nb\_cc,avgH,avgS,avgV,XY\_100,SI\_XY,centroid,color\_spread,complem\_colors,convexity,centroid\_x,centroid\_y,shape\_variance,shape\_skewness,brightness,hue\_contrast,saturation\_contrast,brightness\_contrast,blur\_contrast]=seg1(Irgb,k,m)

% [nb\_cc,avgH,avgS,avgV,XY\_100,SI\_XY,centroid,color\_spread,complem\_colors,...

% convexity,centroid\_x,centroid\_y,shape\_variance,shape\_skewness,brightness,...

% hue\_contrast,saturation\_contrast,brightness\_contrast,blur\_contrast]=seg1(Irgb,k,m)

%

% Performs a color-based segmentation process on our image in the LUV color

% space, using kmeans and bnconncomp

% Then, returns all the segmentation related features (in matrices)

%

% Returns an error if the image contains less than 5 'objects'

Ir=Irgb(:,:,1);

Ig=Irgb(:,:,2);

Ib=Irgb(:,:,3);

Ihsv=rgb2hsv(Irgb);

Ih=Ihsv(:,:,1);

Is=Ihsv(:,:,2);

Iv=Ihsv(:,:,3);

Iluv=RGBim2Luv(Irgb);

I(:,:,1)=context(Iluv(:,:,1),m);

I(:,:,2)=context(Iluv(:,:,2),m);

I(:,:,3)=context(Iluv(:,:,3),m);

%we have to reshape Iluv to apply kmeans to it

X=size(I,1);

Y=size(I,2);

XY=X\*Y;

iLuv=reshape(I,[XY 3]);

ID=kmeans(iLuv,k);

%we now have an image where each pixel has an indice i in [1,k],

%corresponding to the cluster it is in

cluster=reshape(ID,[X Y]);

imtool(cluster);

%we now want to find patches of connected components

binary\_cluster=cell(1,k);

connected\_cluster=cell(1,k);

num\_pixels=cell(1,k);

k\_biggest=zeros(2,k);

nb\_cc=0;

for j=1:k

 binary\_cluster{j}=cluster<j+1;

 cluster=cluster+k\*binary\_cluster{j};

 connected\_cluster{j}=bwconncomp(binary\_cluster{j});

 nb\_cc=nb\_cc+connected\_cluster{j}.NumObjects;

 num\_pixels{j}=cellfun(@numel,connected\_cluster{j}.PixelIdxList);

 [k\_biggest(1,j),k\_biggest(2,j)]=max(num\_pixels{j}); %biggest(1,k)=nb of pixel in it, biggest(1,k)= its index number

end

%we want to select the 5-th biggest connected components and

%analyse them

if nb\_cc<5

 error('/!\ ERROR: This image analysis produce less than 5 connected components! It means that that we can"t analyse its 5 biggest connected components, and that some results ARE false.');

end

biggest\_patches=cell(2,5); %1st row for the number of pixels in the patch, 2nd row for the list of the indices of the pixels in this patch

XY\_100=0;

avgH=-1\*ones(5,1);

avgS=-1\*ones(5,1);

avgV=-1\*ones(5,1);

SI\_XY=zeros(5,1);

centroid=zeros(5,1);

lightness=zeros(5,1);

num\_pixels\_=num\_pixels; %we need to modifie num\_pixels in the loop, but we will also need it later

centroid\_x=zeros(3,1);

centroid\_y=zeros(3,1);

shape\_variance=zeros(3,1);

shape\_skewness=zeros(3,1);

brightness=zeros(3,1);

blur\_matrix=zeros(size(Ir));

blur=zeros(5,1);

for i=1:5;

 %finds the new biggest patch and puts its information in biggest\_patches

 [biggest\_patches{1,i},J]=max(k\_biggest(1,:)); %J is the index of the cluster this patch is from (in [1,k])

 IDX=k\_biggest(2,J); %IDX is its index in connected\_cluster{J}.PixelIdxList

 biggest\_patches{2,i}=connected\_cluster{J}.PixelIdxList{IDX};

 %construction of the features linked to the i-th patch

 if biggest\_patches{1,i}>XY/100

 XY\_100=XY\_100+1;

 end

 avgH(i)=mean(Ih(biggest\_patches{2,i}));

 avgS(i)=mean(Is(biggest\_patches{2,i}));

 avgV(i)=mean(Iv(biggest\_patches{2,i}));

 SI\_XY(i)=biggest\_patches{1,i}/XY;

 avg\_x=mean(mod(biggest\_patches{2,i}-1,X)+1);

 avg\_y=mean(floor((biggest\_patches{2,i}-1)/X)+1);

 r=floor(3\*avg\_x/X)+1;

 c=floor(3\*avg\_y/Y)+1;

 centroid(i)=10\*r+c;

 lightness(i)=mean(Ir(biggest\_patches{2,i})+Ig(biggest\_patches{2,i})+Ib(biggest\_patches{2,i}));

 blur\_matrix(biggest\_patches{2,i})=(Ir(biggest\_patches{2,i})+Ig(biggest\_patches{2,i})+Ib(biggest\_patches{2,i}))/3;

 blur(i)=gaussian\_blur(blur\_matrix,blur\_matrix,blur\_matrix);

 num\_pixels\_{J}(IDX)=0;

 [k\_biggest(1,J),k\_biggest(2,J)]=max(num\_pixels\_{J});

 %those features from paper2 only consider the 3 biggest segments

 if i<4

 centroid\_x(i)=avg\_x/X;

 centroid\_y(i)=avg\_y/Y;

 difference\_x=((mod(biggest\_patches{2,i}-1,X)+1)-avg\_x)/X;

 difference\_y=((floor((biggest\_patches{2,i}-1)/X)+1)-avg\_y)/Y;

 for j=1:biggest\_patches{1,i}

 shape\_variance(i)=shape\_variance(i)+difference\_x(j)^2+difference\_y(j)^2;

 shape\_skewness(i)=shape\_skewness(i)+difference\_x(j)^3+difference\_y(j)^3;

 end

 shape\_variance(i)=shape\_variance(i)/biggest\_patches{1,i};

 shape\_skewness(i)=shape\_skewness(i)/biggest\_patches{1,i};

 brightness(i)=lightness(i);

 end

end

%features from paper1

color\_spread=0;

complem\_colors=0;

%features from paper2

hue\_contrast=0;

saturation\_contrast=0;

brightness\_contrast=0;

blur\_contrast=0;

for i=1:5

 for j=1:5

 color\_spread=color\_spread+abs(avgH(i)-avgH(j));

 complem\_colors=complem\_colors+min(abs(avgH(i)-avgH(j)),1-abs(avgH(i)-avgH(j)));

 hue\_contrast=max(hue\_contrast,min(abs(avgH(i)-avgH(j)),1-abs(avgH(i)-avgH(j))));

 saturation\_contrast=max(saturation\_contrast,abs(avgS(i)-avgS(j)));

 brightness\_contrast=max(brightness\_contrast,abs(lightness(i)-lightness(j)));

 blur\_contrast=max(blur\_contrast,abs(blur(i)-blur(j)));

 end

end

%analysis of the convexity of the shapes in the image

convexity=0;

for j=1:k

 %we first need to detect all the shapes which surface is >200/XY

 for i=1:size(num\_pixels{j},2)

 if (num\_pixels{j}(i))>(XY/200)

 pixel\_list=cell2mat(connected\_cluster{j}.PixelIdxList(i));

 shape\_x=mod(pixel\_list-1,X)+1;

 shape\_y=floor((pixel\_list-1)/X)+1;

 [~,conv\_area]=convhull(shape\_x,shape\_y);

 if (num\_pixels{j}(i)/conv\_area)>0.8

 convexity=convexity+num\_pixels{j}(i);

 end

 end

 end

end

convexity=convexity/XY;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ a ] = thirdsAvg( M,margin )

% [ a ] = thirdsAvg( M,margin )

% Returns the average 'a' of the central part matrix with respect to rule of thirds

% This central part is defined as the central ninth of the photo if you

% divide it along the 1/3 and the 2/3 of its height and width, with a

% margin of 'margin' percent around it.

%

%/!\margin must be in [0,1]

v=0;

for x=floor((1-margin)\*size(M,1)/3):floor((1+margin)\*(2\*size(M,1))/3)

 for y=floor((1-margin)\*size(M,2)/3):floor((1+margin)\*(2\*size(M,2))/3)

 v=v+M(x,y);

 end

end

a=v/((floor((1+margin)\*(2\*size(M,1))/3)-floor((1-margin)\*size(M,1)/3))\*(floor((1+margin)\*(2\*size(M,2))/3)-floor((1-margin)\*size(M,2)/3)));

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [cA,cH,cV,cD]=wavelet(I)

% [cA,cH,cV,cD]=wavelet(I)

% Performs a three-level Haar wavelet transform on I, and returns the cells

% cA, cH, cV and cD where forall i in {1,2,3}, cA{i} is the i-th level approximation

% coefficients matrix, and cH{i}, cV{i} and cD{i} are the i-th level

% details coefficients matrices

cA=cell(1,3);

cH=cell(1,3);

cV=cell(1,3);

cD=cell(1,3);

%Performing 3 level Daubechies tranform on I

startImage = I;

for i = 1:3,

 [cA{i},cH{i},cV{i},cD{i}] = dwt2(startImage,'haar');

 startImage = cA{i};

end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [texture,low\_DOF]=waveletTexture(Ihsv)

% [texture,low\_DOF]=waveletTexture(Ihsv)

% Returns the 4x3 matrix 'texture' containing 12 wavelet features to

% analyse the texture of the image (cf paper1), and a 3x1 matrix 'low\_DOF'

% containing a measure of the low depth of field of the image.

%

% In 'texture': lines 1,2,3,4 correspond to the 1st, 2nd and 3rd level and

% to the sum of the average wavelet coefficients of the column, and columns 1,2,3 to h,s,v

% In 'low\_DOF': the 1st row contains the low DOF feature for H, the 2nd row the one for S, and the 3rd row the one for V

X\_4=floor(size(Ihsv,1)/(8\*4));

X3\_4=floor(3\*size(Ihsv,1)/(8\*4));

Y\_4=floor(size(Ihsv,2)/(8\*4));

Y3\_4=floor(3\*size(Ihsv,2)/(8\*4));

%applies a 3-level wavelet transform

[cA,cH,cV,cD]=wavelet(Ihsv);

%The matrix containing the results (lines 1,2,3,4 correspond to the 1st, 2nd and 3rd level and to the sum of the average wavelet coefficients of the column, columns 1,2,3 to h,s,v)

texture=zeros(4,3);

low\_DOF=zeros(3,1); %1st row for H, 2nd for S, 3rd for V

sH=zeros(3,1);

sS=zeros(3,1);

sV=zeros(3,1);

%construction of the k-level features

for k = 1:3

 sumH=cH{k}(:,:,1) + cV{k}(:,:,1) + cD{k}(:,:,1);

 sumS=cH{k}(:,:,2) + cV{k}(:,:,2) + cD{k}(:,:,2);

 sumV=cH{k}(:,:,3) + cV{k}(:,:,3) + cD{k}(:,:,3);

 %L1 norm

 absH=abs(cH{k}(:,:,1)) + abs(cV{k}(:,:,1)) + abs(cD{k}(:,:,1));

 absS=abs(cH{k}(:,:,2)) + abs(cV{k}(:,:,2)) + abs(cD{k}(:,:,2));

 absV=abs(cH{k}(:,:,3)) + abs(cV{k}(:,:,3)) + abs(cD{k}(:,:,3));

 sH(k,1)=norm(absH(:),1);

 sS(k,1)=norm(absS(:),1);

 sV(k,1)=norm(absV(:),1);

 for i = 1:floor(size(Ihsv,1)/(2^k))

 for j = 1:floor(size(Ihsv,2)/(2^k))

 texture(k,1)=texture(k,1) + sumH(i,j);

 texture(k,2)=texture(k,2) + sumS(i,j);

 texture(k,3)=texture(k,3) + sumV(i,j);

 if ((k==3)&&(((i>X\_4)&&(i<X3\_4))&&((j>Y\_4)&&(j<Y3\_4))))

 low\_DOF(1)=low\_DOF(1)+ sumH(i,j);

 low\_DOF(2)=low\_DOF(2)+ sumS(i,j);

 low\_DOF(3)=low\_DOF(3)+ sumV(i,j);

 end

 end

 end

 if k==3

 low\_DOF(1)=low\_DOF(1)/texture(k,1);

 low\_DOF(2)=low\_DOF(2)/texture(k,2);

 low\_DOF(3)=low\_DOF(3)/texture(k,3);

 end

 texture(k,1)=texture(k,1)/sH(k,1);

 texture(k,2)=texture(k,2)/sS(k,1);

 texture(k,3)=texture(k,3)/sV(k,1);

end

texture(4,1)=texture(1,1)+texture(2,1)+texture(3,1);

texture(4,2)=texture(1,2)+texture(2,2)+texture(3,2);

texture(4,3)=texture(1,3)+texture(2,3)+texture(3,3);

end