Fourier Descriptors Algorithm for Texture Recognition and Classification

C. Emiliano Solórzano Espíndola, carlosemiliano04@gmail.com

ETSEIB- Universitat Politècnica de Catalunya

***Abstract*— The present works evaluates an approach for texture analysis using Fourier Transform for image filtering and reconstruction using frequency bandpass filters. A features vector is generated for each image and is used for a supervised training of a classifier. After the training phase, the new classifier can predict the original class based in a sub-image of the corresponding class original image.**

I. Methodology

The present algorithm can be divided in different steps, which are:

- Fourier Transform and preprocessing of the image.
- Features extraction.
- Training.
- Evaluation.

In the first step the image is processed for better data acquisition using two Sobel directional filters for 0° and 90°, after that the spectrum in the frequency domain is obtained.

The second step consist in creating a feature vector that can represent accurately the image, for this frequency bandpass filters are used.

In the third step 200 random selected sub-images are created for each class in order to train a classifier that can distinguish among the different classes using the feature vector. The Mahalanobis distance for each observation to each class is computed for a visual representation.

In the fourth step a set of 20 random selected sub-images is used for evaluating the accuracy of the newly created classifier. At the end the Mahalanobis distance is again computed and the error percentage is obtained.

II. spect1(): Fourier transform function and preprocessing of the image

This function applies a Sobel filter for 0° and 90° , after that both are transformed to the frequency domain and the outputs are the resulting images

A. Function declaration

In this part we declare the image is the input for the function and as result the output are two images

```
function [fh, fv]=spect1(ima)
```

```
ima=imresize(ima,.5);
[fil,col,capa]=size(ima);
```

B. Sobel filtering

If the image is in RGB converts to grayscale

```
if capa==3
    grey(ima);
end

imaexp=zeros(fil+2,col+2);
[fil,col]=size(imaexp);
for i=2:fil-1
    for j=2:col-1
        imaexp(i,j)=double(ima(i-1,j-1));
    end
end
```

Apply the convolution matrix for horizontal filtering

```
CM=[-1, 0, 1;-2, 0, 2; -1, 0, 1]
```

```
CM =

    -1     0     1
```

```
    -2     0     2
    -1     0     1
```

```
for i=2:fil-1
    for j=2:col-1
        Hh(i-1,j-1)=(1*imaexp(i-1,j-1)+
0*imaexp(i-1,j)+(-1)*imaexp(i-
1,j+1)+2*imaexp(i,j-1)+(0)*imaexp(i,j)-
2*imaexp(i,j+1)+(1)*imaexp(i+1,j-
1)+(0)*imaexp(i+1,j)+(0-
1)*imaexp(i+1,j+1));
    end
end
```

Apply the convolution matrix for vertical filtering

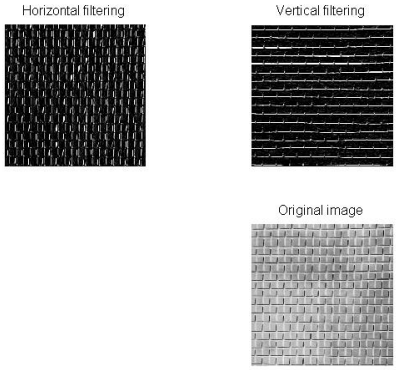
```
CM=[1,2,1;0,0,0;-1,-2,-1]
```

```
CM =

     1     2     1
     0     0     0
    -1    -2    -1
```

```
for i=2:fil-1
    for j=2:col-1
        Hv(i-1,j-1)=((-1)*imaexp(i-1,j-1)-
2*imaexp(i-1,j)+(-1)*imaexp(i-
1,j+1)+0*imaexp(i,j-
1)+(0)*imaexp(i,j)+(0)*imaexp(i,j+1)+1*ima
exp(i+1,j-
1)+2*imaexp(i+1,j)+(1)*imaexp(i+1,j+1));
    end
end

% imaf=Hh+Hv;
```

C. Show modified images

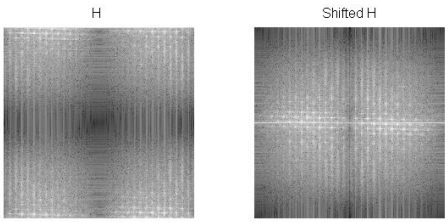
```
figure
subplot(221),
imshow(uint8(Hh)),title('Horizontal
filtering');
subplot(222),
imshow(uint8(Hv)),title('Vertical
filtering');
subplot(224),
imshow(uint8(ima)),title('Original
image');
```



D. Transformation to the frequency domain

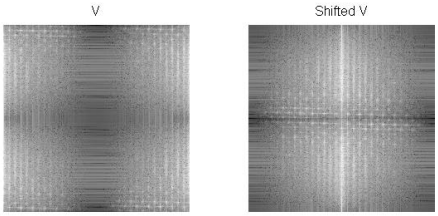
Here the program uses the Fast Fourier Transform (fft2) to obtain the spectrum of both images

```
%Create the horizontal filtered spectrum
fo=fft2(Hh);
fh=fo;
fr=fftshift(fo);
fos=log(1+abs(fo));
fis=log(1+abs(fr));
S=abs(fr);
figure,subplot(121),
imshow(fos,[]),title('H ');
subplot(122),
imshow(fis,[]),title('Shifted H');
```




```
%Create the vertical filtered spectrum
fo=fft2(Hv);
fv=fo;
fr=fftshift(fo);
fos=log(1+abs(fo));
fis=log(1+abs(fr));
```

```
S=abs(fr);
figure, subplot(121),
imshow(fos,[]),title('V');
subplot(122),
imshow(fis,[]),title('Shifted V');
```



III. CAR2(): FEATURES EXTRACTION FUNCTION

This function creates a set of ideal filters and applies them to the input images. After that, the energy is calculated for each resulting filtered image .The output is a features vector with the same size as the rings number.

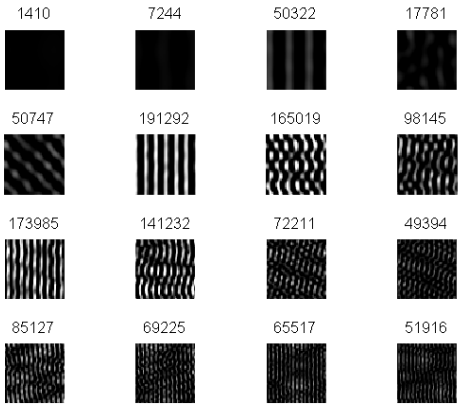
```
function features=car2(ima)
```

Randomly select a pixel within the image range and create a new subimage *cort.*

```
l=randi(500);
r=randi(300);
cort=ima((l:l+99),(r:r+99));


[b c]=spect1(cort); % Calls the _spect1()_
function
[x,y]=size(b);

n=50; % Rings number
if x>=y
h=x/n;

end
```

Calculate energy in horizontal spectrum rings and display the reconstructed image after the applied filter.

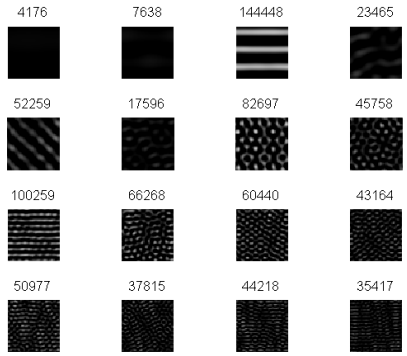
```
Eh=[];
for i=1:n/2
    f=0;
    g=lpfilter('ideal',x,y,h*i);
    g2=1-lpfilter('ideal',x,y,h*(i-1));
    fi=b.*g.*g2;
    d=uint8(ifft2(fi));
    e=sum(d(:));
    Eh(i)=e;
  if i<=16
figure(4),subplot(4,4,i),imshow(d),title(e
);
    end

end
```



Calculate energy in vertical spectrum rings and display the reconstructed image after the applied filter.

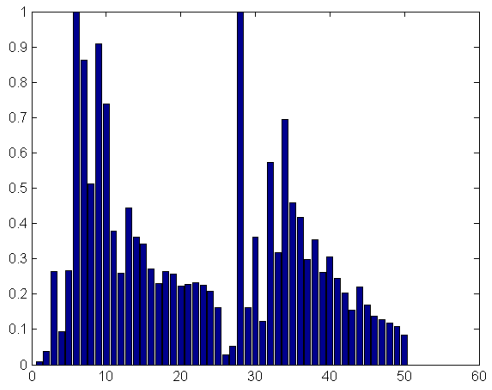
```
Ev=[];
for i=1:n/2

    g=lpfilter('ideal',x,y,h*i);
    g2=1-lpfilter('ideal',x,y,h*(i-1));
    fi=c.*g.*g2;

    d=uint8(ifft2(fi));
    e=sum(d(:));
  if i<=16
figure(4),subplot(4,4,i),imshow(d),title(e
);
    end

    Ev(i)=e;
end
```

```
Eh=Eh/max(Eh);
Ev=Ev/max(Ev);
```



Features vector and output of the function.

```
features(1:25)=Eh;
features(26:50)=Ev;
figure,bar(features)
```



IV. Training

This script's main function is to create a new classifier trained with data obtained from subimages of six different pictures. the resulting classifier has the tags with the names of the classes in the order the images were loaded.

```
clc, close all, clear all
```

A. 1st Part: Feature extraction

In this script a vector of 1200 obervations is created with 200 samples of each feature vector from random subimages of the original picture.

First image for classification

```
im1='D101.gif';
car4=zeros(1200,50);
for i=1:200
    car4(i,:)=car2(im1);
    close all
end
```

Second image for classification

```
im1='D1.gif';
for i=1:200
    car4(i+200,:)=car2(im1);
    close all
end
```

Third image for classification

```
im1='D52.gif';
for i=1:200
    car4(i+400,:)=car2(im1);
    close all
end
```

Fourth image for classification

```
im1='D49.gif';
for i=1:200
    car4(i+600,:)=car2(im1);
    close all
end
```

Fifth image for classification

```
im1='D20.gif';
for i=1:200
    car4(i+800,:)=car2(im1);
    close all
end
```

Sixth image for classification

```
im1='D3.gif';


for i=1:200
    car4(i+1000,:)=car2(im1);
    close all
end
```

B. *2nd part: Training process*

Here a cell array of 'tags' is created with the corresponding names for each observation and it's corresponding class. After that a classifier *X* is created

```
% Creates an array of type 'cell' for the
classes labels
C=cell(1200,1);
C(1:200)={'class 1, D101'};
C(201:400)={'class 2, D1'};
C(401:600)={'class 3, D52'};
C(601:800)={'class 4, D49'};
C(801:1000)={'class 5, D20'};
C(1001:1200)={'class 6, D3'};
```

The new classifier uses the pseudoLinear type since the observation matrix is not a square matrix. For computing the covariance uses the pseudoinverse matrix

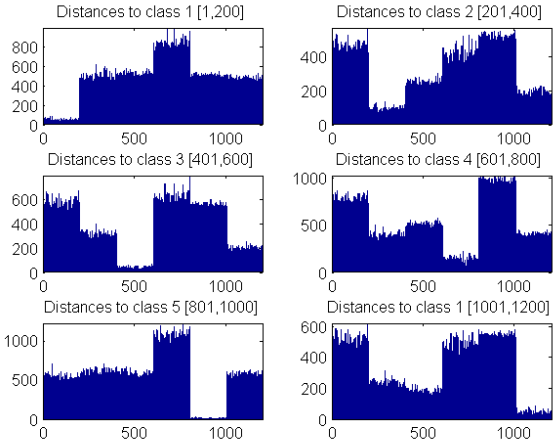
```
X=fitcdiscr(car4,C,'discrimType','pseudoLi
near')
```

```
X =

  ClassificationDiscriminant
     PredictorNames: {1x50 cell}
       ResponseName: 'Y'
         ClassNames: {1x6 cell}
     ScoreTransform: 'none'
    NumObservations: 1200
        DiscrimType: 'pseudoLinear'
                 Mu: [6x50 double]
             Coeffs: [6x6 struct]
```

C. *Evaluation*

Compute the Mahalanobis distance from each observation to each class

```
dist=zeros(1200,6);
for i=1:1200
    dist(i,:)=mahal(X,car4(i,:));
end

figure,
subplot(321),bar(dist(:,1)),title('Distanc
es to class 1 [1,200]')
axis([0 1200 0 inf])
subplot(322),bar(dist(:,2)),title('Distanc
es to class 2 [201,400]')
axis([0 1200 0 inf])
subplot(323),bar(dist(:,3)),title('Distanc
es to class 3 [401,600]')
axis([0 1200 0 inf])
subplot(324),bar(dist(:,4)),title('Distanc
es to class 4 [601,800]')
axis([0 1200 0 inf])
subplot(325),bar(dist(:,5)),title('Distanc
es to class 5 [801,1000]')
axis([0 1200 0 inf])
subplot(326),bar(dist(:,6)),title('Distanc
es to class 1 [1001,1200]')
axis([0 1200 0 inf])
```



V. CLASSIFICATION TEST

This program evaluates the classifier by using the same function for taking samples of an image of a fixed size and evaluating the features vector

A. *Initialization*

Here the program clears all before running the program, also loads the previous programed classifier.

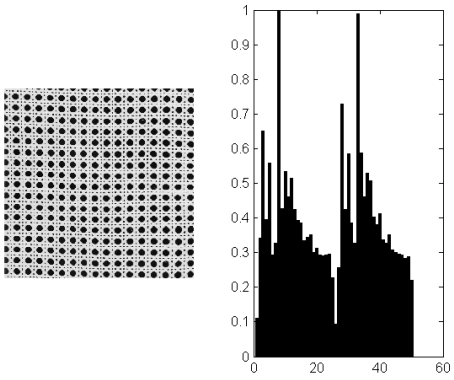
```
clc, close all, clear all
load('classif2.mat');
```

B. *First image for classification*

Loads the images and evaluates the *car2()* function in 20 samples of every image which will be the feature vector of 100*1 size , also displays the averaged feature vector for the samples.

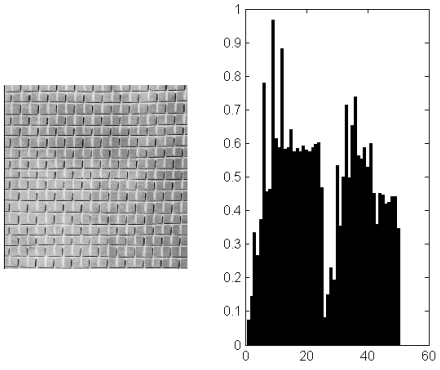
```
im1='D101.gif';

car4=zeros(120,50);
```

```
for i=1:20
    car4(i,:)=car2(im1);
    close all
end
figure,subplot(121),imshow(imread(im1))
subplot(122),bar(mean(car4(1:20,:)))
```



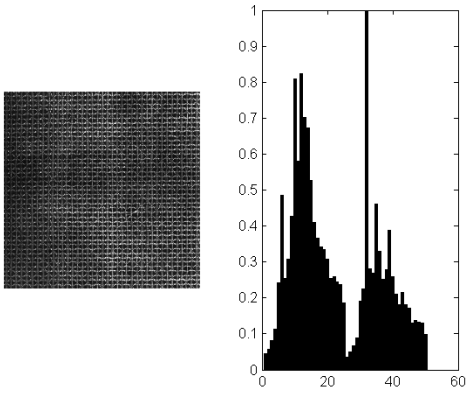
C. Second image for classification

```
im1='D1.gif';

for i=1:20
    car4(i+20,:)=car2(im1);
    close all
end
figure,subplot(121),imshow(imread(im1))
subplot(122),bar(mean(car4(21:40,:)))
```



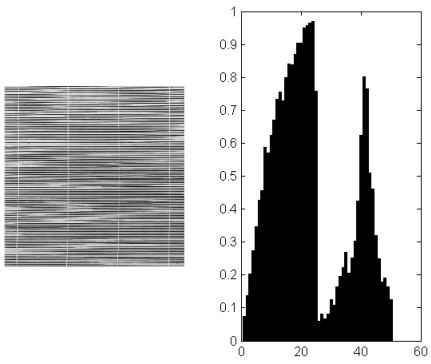
D. Third image for classification

```
im1='D52.gif';

for i=1:20
    car4(i+40,:)=car2(im1);
    close all
end
figure,subplot(121),imshow(imread(im1))
subplot(122),bar(mean(car4(41:60,:)))
```



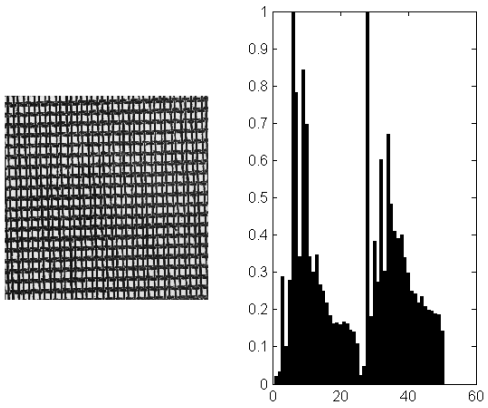
E. Fourth image for classification

```
im1='D49.gif';

for i=1:20
    car4(i+60,:)=car2(im1);
    close all
end
figure,subplot(121),imshow(imread(im1))
subplot(122),bar(mean(car4(61:80,:)))
```



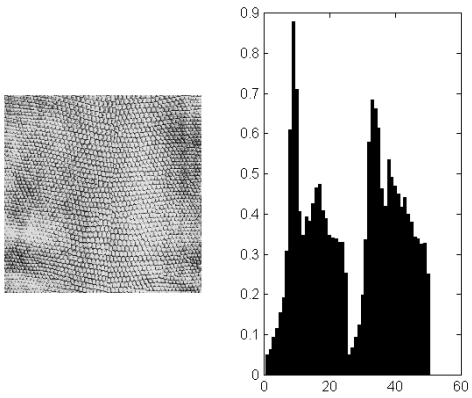
F. Fifth image for classification

```
im1='D20.gif';

for i=1:20
    car4(i+80,:)=car2(im1);
    close all
end
figure,subplot(121),imshow(imread(im1))
subplot(122),bar(mean(car4(81:100,:)))
```



G. Sixth image for classification

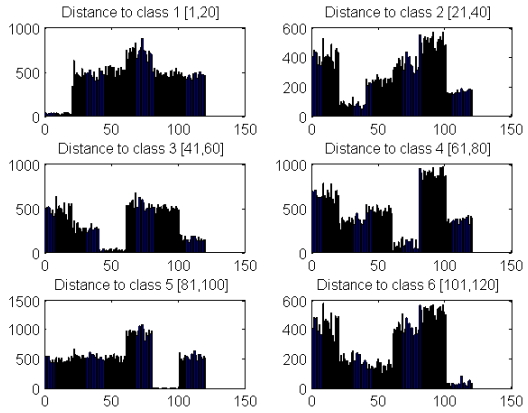
```
im1='D3.gif';

for i=1:20
    car4(i+100,:)=car2(im1);
    close all
end
figure,subplot(121),imshow(imread(im1))
subplot(122),bar(mean(car4(101:120,:)))
```



H. Mahalanobis distance to classes

Here the program evaluates the *Mahalanobis* distance for each observation to each class

```
dist=zeros(120,6);
for i=1:120
    dist(i,:)=mahal(X,car4(i,:));
end

figure(3),
subplot(321),bar(dist(:,1)),title('Distanc
e to class 1 [1,20]')
subplot(322),bar(dist(:,2)),title('Distanc
e to class 2 [21,40]')
subplot(323),bar(dist(:,3)),title('Distanc
e to class 3 [41,60]')
subplot(324),bar(dist(:,4)),title('Distanc
e to class 4 [61,80]')
subplot(325),bar(dist(:,5)),title('Distanc
e to class 5 [81,100]')
subplot(326),bar(dist(:,6)),title('Distanc
e to class 6 [101,120]')
```



I. Evaluation algorithm

Compares the tagged vector from the predictor to the real tag that it should have depending on the class, each time an error occurs the *eva* vector gets a value of one in the corresponding observation

```
comp=cell(120,3);
eva=zeros(120,1);
for i=1:120
    comp(i)={predict(X,car4(i,:))};
end

 for i=1:20
     if strcmp(comp{i},'class 1, D101')==0
         eva(i)=1;
     end
 end

 for i=1:20
     if strcmp(comp{i+20},'class 2,
D1')==0
         eva(i+20)=1;
     end
 end

  for i=1:20
     if strcmp(comp{i+40},'class 3,
D52')==0
         eva(i+40)=1;
     end
  end
```

```
   for i=1:20
     if strcmp(comp{i+60},'class 4,
D49')==0
          eva(i+60)=1;
     end
   end

   for i=1:20
     if strcmp(comp{i+80},'class 5,
D20')==0
          eva(i+80)=1;
     end
   end

    for i=1:20
     if strcmp(comp{i+100},'class 6,
D3')==0
          eva(i+100)=1;
     end
    end
```

J. Result

Error Percentage

```
t=sum(eva)/1.2
```

```
t =

     0
```

VI. Results

Running the test script 6 times the following errors percentages appear:

	Error percentage
1.	1.667
2.	0
3.	0
4.	0
5.	1.667
6.	0