

STUDY OF LEGENDRE MOMENTS OF AN IMAGE, IMAGE RECONSTRUCTION AND CLASSIFICATION

WHAT ARE MOMENTS?

In image processing, computer vision and related fields, an image moment is a certain particular weighted average (moment) of the image pixels' intensities, or a function of such moments, usually chosen to have some attractive property or interpretation.

Image moments are useful to describe objects after segmentation. Simple properties of the image which are found via image moments include area (or total intensity), its centroid, and information about its orientation.

General moment M_{pq}^f of an image $f(x, y)$, where p, q are non-negative integers and $r = p + q$ is called the order of the moment, defined as

$$M_{pq}^{(f)} = \iint_D p_{pq}(x, y) f(x, y) dx dy,$$

where $p_{00}(x, y), p_{10}(x, y), \dots, p_{kj}(x, y), \dots$ are polynomial basis functions defined on D .

Depending on the polynomial basis used, we recognize various systems of moments.

GEOMETRIC AND COMPLEX MOMENTS

The most common choice is a standard power basis $p_{kj}(x, y) = x^k y^j$ that leads to geometric

Moments.

$$M_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x, y) dx dy$$

for $p, q = 0, 1, 2, \dots$ Adapting this to scalar (greyscale) image with pixel intensities $I(x, y)$, raw image moments M_{ij} are calculated by

$$M_{ij} = \sum_x \sum_y x^i y^j I(x, y)$$

ORTHOGONAL MOMENTS

If the polynomial basis $\{p_{kj}(x, y)\}$ is orthogonal, i.e. if its elements satisfy the condition:

$$\iint_{\Omega} p_{pq}(x, y) \cdot p_{mn}(x, y) dx dy = 0$$

Legendre moments are also Orthogonal.

LEGENDRE MOMENTS

Using the Legendre polynomials $P_p(x)$ and $P_q(y)$ in the expression

$$M_{pq}^{(f)} = \iint_D p_{pq}(x, y) f(x, y) dx dy,$$

we get Legendre Moments.

The two-dimensional Legendre moments of order $(p+q)$, with image intensity function $f(x; y)$, are defined as:

$$L_{pq} = \frac{(2p+1)(2q+1)}{4} \int_{-1}^1 \int_{-1}^1 P_p(x) \times P_q(y) f(x, y) dx dy; \quad x, y \in [-1, 1],$$

where Legendre polynomial, $P_p(x)$, of order p is given by:

$$P_p(x) = \sum_{k=0}^p \left\{ (-1)^{\frac{p-k}{2}} \frac{1}{2^p} \frac{(p+k)! x^k}{\left(\frac{p-k}{2}\right)! \left(\frac{p+k}{2}\right)! k!} \right\}_{p-k=\text{even}} \quad (2.2)$$

The recurrence relation of Legendre polynomials, $P_p(x)$, is given as follows:

$$P_p(x) = \frac{(2p-1)xP_{p-1}(x) - (p-1)P_{p-2}(x)}{p},$$

where $P_0(x)=1$, $P_1(x)=x$ and $p>1$. Since the region of definition of Legendre polynomials is the interior of $[-1; 1]$, a square image of $N \times N$ pixels with intensity function $f(i, j)$,

$0 < (i, j) < (N-1)$, is scaled in the region of $-1 < (x, y) < 1$.

In the result of this, equation can now be expressed in discrete form as:

$$L_{pq} = \lambda_{pq} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} P_p(x_i) P_q(y_j) f(i, j),$$

where the normalizing constant,

$$\lambda_{pq} = \frac{(2p+1)(2q+1)}{N^2}.$$

x_i and y_j denote the normalized pixel coordinates in the range of $[-1; 1]$, which are given by

$$x_i = \frac{2i}{N-1} - 1 \quad \text{and} \quad y_j = \frac{2j}{N-1} - 1.$$

TRANSLATION INVARIANTS OF LEGENDRE MOMENTS

The effect resulted from the change of location of an image on moment computation can be cancelled out by considering its translation invariant property. The direct description of translation invariant of 2D Legendre moments can be obtained by evaluating their central moments, which is defined as follows:

$$\varphi_{pq} = \frac{(2p+1)(2q+1)}{4} \int_{-1}^1 \int_{-1}^1 P_p(x-x_0) \times P_q(y-y_0) f(x, y) dx dy,$$

where the centroids of x- and y-coordinate, x_0 and y_0 , respectively, are derived as

$$x_0 = \frac{\sum \sum x f(x, y)}{\sum \sum f(x, y)} \quad \text{and} \quad y_0 = \frac{\sum \sum y f(x, y)}{\sum \sum f(x, y)}.$$

CODE for Legendre Moments:

(Results are stored in form of a matrix $L(i,j)$ for all the orders)

```
%img is the image object formed after reading the image using imread()
function of MATLAB.

function L=Legendre_Cntral_Matrix_c(img)
[nx,ny]=size(img);
img = double(img);

for x = 0 : nx - 1
    for n = 0 : nx-1
        if(n == 0)
            M_LX(n + 1,x + 1) = 1; % p0(x)
        elseif(n == 1)
            M_LX(n + 1,x + 1) = 2*x/(nx-1)-1 ; %p1(x)
        else
            A = (2*n-1)*( 2*x/(nx-1)-1)/n;
            B = (n-1)/n;
            M_LX(n + 1,x + 1) = M_LX(n,x + 1) * A - M_LX(n-1,x + 1) * B;%
        legendre polynomial pp(x)
    end
end

for y = 0 : ny - 1
    for n = 0 : ny-1
        if(n == 0)
            M_LY(n + 1,y + 1) = 1;
        elseif(n == 1)
            M_LY(n + 1,y + 1) = 2*y/(ny-1)-1;
        else
            A = (2*n-1)*( 2*y/(ny-1)-1)/n;
            B = (n-1)/n;
            M_LY(n + 1,y + 1) = M_LY(n,y + 1) * A - M_LY(n-1,y + 1) * B;%
        legendre polynomial pq(y)
    end
end

%storing legendre moments
for m = 0 : nx-1
    for n= 0 : ny-1
        L(m+1,n+1)=0;
        for i = 1 : nx
            for j = 1 : ny
                L(m+1,n+1)=L(m+1,n+1)+(M_LX(m+1,i)*M_LY(n+1,j)*img(i,j));
            end
        end
        L(m+1,n+1)=L(m+1,n+1)* ((2*m+1)*(2*n+1))/(nx*ny);
    end
end
```

RECONSTRUCTION OF IMAGE FROM LEGENDRE MOMENTS

Using the orthogonality property of Legendre moments, the image can be approximately reconstructed from a finite number moments of order up to (M, M) as:

$$f(x, y) \approx \sum_{i=0}^M \sum_{j=0}^M P_i(x) P_j(y) L_{ij}^{(f)}.$$

Using the imshow() functionality of MATLAB we can see the reconstructed image.

(1) BINARY IMAGES

CODE:

```
%CALLING FUNCTION
clc
clear
close
img=imread('7.bmp');
img=imresize(img,[64 64],'nearest');% to run faster
img= double(img)/255;
inLegendre_Cntral_Matrix_c(img);

function Tl=inLegendre_Cntral_Matrix_c(img)%%%%% translation invariant LM
                                                    %%% max -- order

[nx,ny]=size(img);
img = double(img);

for x = 0 : nx - 1
    for n = 0 : nx-1
        if(n == 0)
            M_LX(n + 1,x + 1) = 1; % p0(x)
        elseif(n == 1)
            M_LX(n + 1,x + 1) = 2*x/(nx-1)-1 ; %p1(x)
        else
            A = (2*n-1)*( 2*x/(nx-1)-1)/n;
            B = (n-1)/n;
            M_LX(n + 1,x + 1) = M_LX(n,x + 1) * A - M_LX(n-1,x + 1) * B;
        % legendre polynomial pp(x)
        end
    end
end
```

```

for y = 0 : ny - 1
    for n = 0 : ny-1

        if(n == 0)
            M_LY(n + 1,y + 1) = 1;
        elseif(n == 1)
            M_LY(n + 1,y + 1) = 2*y/(ny-1)-1;
        else
            A = (2*n-1)*( 2*y/(ny-1)-1)/n;
            B = (n-1)/n;
            M_LY(n + 1,y + 1) = M_LY(n,y + 1) * A - M_LY(n-1,y + 1) * B;
        end
    end
end

% legendre polynomial pq(y)
end

%storing legendre moments
for m = 0 : nx-1
    for n= 0 : ny-1
        L(m+1,n+1)=0;
        for i = 1 : nx
            for j = 1 : ny
                L(m+1,n+1)=L(m+1,n+1)+(M_LX(m+1,i)*M_LY(n+1,j)*img(i,j));
            end
        end
        L(m+1,n+1)=L(m+1,n+1)* ((2*m+1)*(2*n+1))/(nx*ny);
    end
end



















%inverse calculation using stored values of Moments
for x = 0 : nx-1
    for y = 0 : ny-1
        T1(x+1,y+1)=0;
        for i = 0 : nx-1
            for j = 0 : ny-1
                T1(x+1,y+1)=T1(x+1,y+1)+
                (M_LX(i+1,x+1)*M_LY(j+1,y+1)*L(i+1,j+1));
            end
        end
    end
end

% Applying the threshold and checking the output:

level = graythresh(ans);
BW = im2bw(ans,level);
figure,imshow(BW)

```

RESULTS:

INPUT IMAGE	RECONSTRUCTED IMAGE
	
	
	
	
	
	
	
	
	

(2) GRAYSCALE IMAGES

CODE:

```
%CALLING FUNCTION
clc
clear
close
img=imread('einstein.jpg');
%img = imresize(img,0.5); %for faster execution
img= double(img)/255;
imshow(img); %Input image to computer moments first
inLegendre_Cntral_Matrix_c(img);

function T1=inLegendre_Cntral_Matrix_c(img)

[nx,ny]=size(img);
img = double(img);

for x = 0 : nx - 1
    for n = 0 : nx-1
        if(n == 0)
            M_LX(n + 1,x + 1) = 1; % p0(x)
        elseif(n == 1)
            M_LX(n + 1,x + 1) = 2*x/(nx-1)-1 ; %p1(x)
        else
            A = (2*n-1)*( 2*x/(nx-1)-1)/n;
            B = (n-1)/n;
            M_LX(n + 1,x + 1) = M_LX(n,x + 1) * A - M_LX(n-1,x + 1) * B;
        % legendre polynomial pp(x)
        end
    end
end

for y = 0 : ny - 1
    for n = 0 : ny-1
        if(n == 0)
            M_LY(n + 1,y + 1) = 1;
        elseif(n == 1)
            M_LY(n + 1,y + 1) = 2*y/(ny-1)-1;
        else
            A = (2*n-1)*( 2*y/(ny-1)-1)/n;
            B = (n-1)/n;
            M_LY(n + 1,y + 1) = M_LY(n,y + 1) * A - M_LY(n-1,y + 1) * B;
        % legendre polynomial pq(y)
        end
    end
end

%Storing legendre moments
for m = 0 : nx-1
    for n= 0 : ny-1
        L(m+1,n+1)=0;
        for i = 1 : nx
```

```

        for j = 1 : ny
            L(m+1,n+1)=L(m+1,n+1)+(M_LX(m+1,i)*M_LY(n+1,j)*img(i,j));
        end
    end
    L(m+1,n+1)=L(m+1,n+1)* ((2*m+1)*(2*n+1))/(nx*ny);
end
end

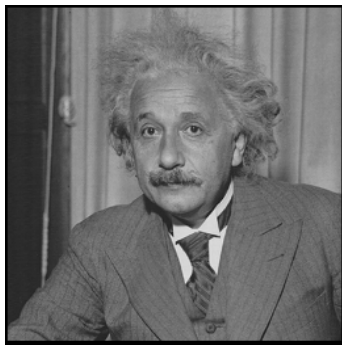
%Using stored values to legendre moments to calculate inverse
for x = 0 : nx-1
    for y = 0 : ny-1
        T1(x+1,y+1)=0;
        for i = 0 : nx-1
            for j = 0 : ny-1
                T1(x+1,y+1)=T1(x+1,y+1)+
(M_LX(i+1,x+1)*M_LY(j+1,y+1)*L(i+1,j+1));
            end
        end
    end
end
end

% Applying the threshold and checking the reconstructed image:

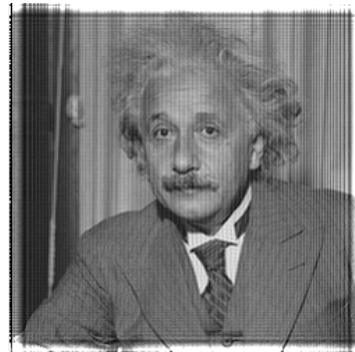
level = graythresh(ans);
BW = im2bw(ans,level);
figure,imshow(BW)

```

RESULTS:



Input image



Reconstructed Image

IMAGE CLASSIFIER

This method applies Euclidean Distance classifier on translated set of Legendre Moments.

Working Principle

The test database comprises of 16 images. Another sample database is created from these 16 images, by translating each of the 16 images by four translating factors. Then the comparison is done with each of the 16 images and set of 64 images. The minimum among 64 values (for each of 16 images) is found, which gives the recognized output. For 1st iteration recognized index should be between 1 and 4, then 5 and 8, then 9 and 12, and so on. This is because (in set of 64 final images), first 4 came from translating the 1st image, next 4 came from 2nd and so on.

CODE:

```
%function to calculate features of an image
function d = feat(img,p,q)
%p,q are translating factor
a = Legendre_trans_fn(2,0,p,q,img);
b = Legendre_trans_fn(0,2,p,q,img);
c = Legendre_trans_fn(2,1,p,q,img);
e = Legendre_trans_fn(1,2,p,q,img);
f = Legendre_trans_fn(3,0,p,q,img);
g = Legendre_trans_fn(1,1,p,q,img);
d = [a b c e f g];
end

%FUNCTION FOR TRANSLATION INVARIANT OF LEGENDRE MOMENTS
function s = Legendre_trans_fn(p,q,a,b,img)
%p,q are order; a,b are translating factor

img=imtranslate(img,[a b]);
[nx,ny]=size(img);
img = double(img);
sum_F = sum(img(:));

sum_Fx = 0;
sum_Fy = 0;

for i = 0 : nx - 1
    for j = 0 : ny - 1
        x=2*i/(nx-1)-1;
        y=2*j/(ny-1)-1;
        sum_Fx = sum_Fx + x * img(i + 1,j + 1);
        sum_Fy = sum_Fy + y * img(i + 1,j + 1);
    end
end

%%% 1st order moment
x_0 = double(sum_Fx / sum_F);
y_0 = double(sum_Fy / sum_F);
```

```

for x = 0 : nx - 1
    for n = 0 : p
        if(n == 0)
            M_LX(n + 1,x + 1) = 1; % p0(x)
        elseif(n == 1)
            M_LX(n + 1,x + 1) = 2*x/(nx-1)-1-x_0 ; %p1(x)
        else
            A = (2*n-1)*( 2*x/(nx-1)-1-x_0)/n;
            B = (n-1)/n;
            M_LX(n + 1,x + 1) = M_LX(n,x + 1) * A - M_LX(n-1,x + 1) * B;
        % legendre polynomial pp(x)
        end
    end
end

for y = 0 : ny - 1
    for n = 0 : q
        if(n == 0)
            M_LY(n + 1,y + 1) = 1;
        elseif(n == 1)
            M_LY(n + 1,y + 1) = 2*y/(ny-1)-1-y_0;
        else
            A = (2*n-1)*( 2*y/(ny-1)-1-y_0)/n;
            B = (n-1)/n;
            M_LY(n + 1,y + 1) = M_LY(n,y + 1) * A - M_LY(n-1,y + 1) * B;
        % legendre polynomial pq(y)
        end
    end
end

T=0;
for i = 1 : nx
    for j = 1 : ny
        T=T+(M_LX(p+1,i)*M_LY(q+1,j)*img(i,j));
    end
end
s=T* ((2*p+1)*(2*q+1))/(nx*ny);

%MAIN CODE
clc;
clear all;
close all;
TestPath = 'D:\winter training\svm classify\mywork\TestDatabase3';
frames = dir(fullfile(TestPath, '*.png'));
c_num = length(frames);
for jj = 1:c_num,
    imgpath = fullfile(TestPath, frames(jj).name);

    I = imread(imgpath);p = 2;q = 1;
    X(jj,:) = feat(I,p,q); %y(jj) = jj;
end
% test part

c=1;

```

```

for jj = 1:c_num,
    imgpath = fullfile(TestPath, frames(jj).name);

    I = imnoise(imread(imgpath), 'salt & pepper', 0.1);
    for a = .5 : .5 : 1
        for b = .5 : .5 : 1
            y(c,:) = feat(I,a,b); %y(jj) = jj;
            c = c+1;
        end
    end
    %double for loop runs 4 times and outer loop runs for 16 times.
end
% c becomes 64 here; y is a vector containing 64 elements (as vector of
features)

for jj = 1 : c_num
    Euc_dist = []; %euc_dist vector made for each of 16 images
    for c = 1 : 64
        temp = 0;
        for i = 1 : 6
            temp = temp + ( X(jj,i) - y(c,i))^2;
        end
        s = sqrt(temp);
        Euc_dist = [Euc_dist s];
    end
    u(jj,:) = Euc_dist;
    [Euc_dist_min, Recognized_index] = min(Euc_dist)
end

```