filtering assignment

by Mayank Singhal

Submission date: 20-Aug-2018 11:52PM (UTC+0800)
Submission ID: 991554956
File name: allthecode.m (13.2K)
Word count: 1640
Character count: 10962

```
function [FI ] = LinearContrastStretching( I ,M )

 % m is the number of rows, n is the # of columns and ch is the # of channels

%FI =zeros(size(I));

[m , n, ch] = size(I);

for i=1:ch

        mx = max(max(I(:,:,i)));

        mn = min(min(I(:,:,i)));

        FI(:,:,i) = (I(:,:,i)-mn)*(1/(mx- mn));

end

end

%% UNSHARP MASKING OF IMAGES

% * Performing Unsharp Masking on the given two images.

% * Convolve the blurred image with a gaussian mask.
```

```
% * Then, add the negative of this image with the original image.

% Scale this "unsharp mask" and add it back to the original image.

%% PARAMETERS FOR THE FUNCTION

% * Standard deviation of the gaussian = std

% * Size of the gaussian filter = fs

% * Scaling factor = s

%% Image - lionCrop.mat

% * Standard deviation of the gaussian = std = 7

% * Size of the gaussian filter = fs = 5 (5 X 5 matrix)

% * Scaling factor = s = 2

s1 = 2;

std1 = 7;

fs1 = 5;
```

```
x = load('../data/lionCrop.mat');

x1 = x.imageOrig;

myUnsharpMasking(s1,std1,x1,fs1);

%% Image - superMoonCrop.mat

% * Standard deviation of the gaussian = std = 8

% * Size of the gaussian filter = fs = 8 (8 X 8 matrix)

% * Scaling factor = s = 2

s2 = 2;

std2 = 8;

fs2 = 8;

y = load('../data/superMoonCrop.mat');

y1 = y.imageOrig;
```

```
myUnsharpMasking(s2,std2,y1,fs2);function [  ] = myUnsharpMasking(s, std_dev,I,
filter_size )
% scaling factor = s
% dimensions of the gaussian filter = filter_size
g_filter = fspecial('gaussian',filter_size,std_dev); % creates the gaussian mask of size 5
conv_img = imfilter(I,g_filter,'conv'); % performs a conolution of the image with the
gaussian mask
DOG = I - conv_img;
scaled_DOG = s*DOG; % now to scale by s
out = I + scaled_DOG;
% contrast stretch the images to range [0,1] with the function from the
% previous assignment.
out = LinearContrastStretching(out,[]);
```

```
I = LinearContrastStretching(I,[]);

figure(1);

subplot(1,2,1);

imshow(I);

title('Original')

subplot(1,2,2);

imshow(out)

title('After Unsharp Masking')

end

function [mykernel,noisyImage,outputImage] = myBilateralFiltering(inputImage, hs, hi)

    % constants

    WINDOW_SIZE = 15;

    pad = floor(WINDOW_SIZE/2);
```

```
% create Spatial gaussian kernel

mykernel = fspecial('gaussian', WINDOW_SIZE, hs);

kernel = mykernel(:);

% find intensity range in input

inputSize = size(inputImage);

minIntensity = min(inputImage(:));

maxIntensity = max(inputImage(:));

rangeIntensity = maxIntensity - minIntensity;

% add gaussian noise to input

noisyImage = inputImage + 0.05 * rangeIntensity * randn(inputSize);

paddednoisyImage = padarray(noisyImage, [pad, pad], nan, 'both');

% pre-allocate output matrix

outputImage = zeros(inputSize);
```

```
% run filtering over all pixels

for y=1:inputSize(1)

    for x=1:inputSize(2)

        % get pixel intensity

        pixelIntensity = noisyImage(y, x);

        % Get window

        window = getWindow(paddednoisyImage, WINDOW_SIZE, x+pad , y+pad);

        % Calculations

        windowArray = window(:);

        windowArray = windowArray - pixelIntensity;

        windowArray = windowArray.^2;

        windowArray = windowArray./(hi^2);

        windowArray = exp(-windowArray);
```

```
            windowArray = windowArray.*kernel;

            denominator = nansum(windowArray);

            windowArray = window(:).*windowArray;

            numerator = nansum(windowArray);

            % Update output image

            outputImage(y,x) = numerator/denominator;

        end

    end

end

function window = getWindow(I,wsize,x,y)

    isize = size(I);

    % calculate window boundaries

    left = max(1,x-floor(wsize/2));
```

```
    right = min(isize(2),x+floor(wsize/2));

    top = max(1,y-floor(wsize/2));

    bottom = min(isize(1),y+floor(wsize/2));

    % create window

    window = I(top:bottom,left:right);

endtic;

%% Assignment 2: Question 2:

% This script performs bilateral filtering on some input images

% and automatically publishes the results in a formatted HTML document.

%% a) Barbara

I = load('../data/barbara.mat');

I = I.imageOrig;

hs = 1.58;
```

```
hi = 13.76;

[k,noisy,im] = myBilateralFiltering(I, hs, hi);

rmsd0 = myRMSD(I,im);

%%

imagesc(I);

title('Original Image');

colormap gray;

colorbar;

%%

imagesc(noisy);

title('Corrupted Image');

colormap gray;

colorbar;
```

```
%%
imagesc(im);
title('Filtered Image');
colormap gray;
colorbar;
%%
imagesc(k);
title('Mask for spatial Gaussian');
colormap jet;
colorbar;
%%
% Optimal Values:
```

```
fprintf('%s%f₩n%s%f₩n%s%f₩n','Spatial gaussian standard deviation: ',hs,'Intensity
gaussian standard deviation: ', hi, 'Corresponding value of RMSD is ',rmsd0);

[~,~,im] = myBilateralFiltering(I, 0.9*hs, hi);

rmsd1 = myRMSD(I, im);

[~,~,im] = myBilateralFiltering(I, 1.1*hs, hi);

rmsd2 = myRMSD(I,im);

[~,~,im] = myBilateralFiltering(I, hs, 0.9*hi);

rmsd3 = myRMSD(I,im);

[~,~,im] = myBilateralFiltering(I, hs, 1.1*hi);

rmsd4 = myRMSD(I,im);

%%

% Other Readings:

fprintf('%s₩t%s%f₩n', 'For Case (i)  :', 'the value of RMSD is ', rmsd1);
```

```
fprintf('%s₩t%s%f₩n', 'For Case (ii) :', 'the value of RMSD is ', rmsd2);

fprintf('%s₩t%s%f₩n', 'For Case (iii):', 'the value of RMSD is ', rmsd3);

fprintf('%s₩t%s%f₩n', 'For Case (iv) :', 'the value of RMSD is ', rmsd4);

%% b) grass

I = im2double(imread('../data/grass.png'));

hs = 0.62;

hi = 0.71;

[k,noisy,im] = myBilateralFiltering(I, hs, hi);

rmsd0 = myRMSD(I,im);

%%

imagesc(I);

title('Original Image');

colormap gray;
```

```
colorbar;

%%

imagesc(noisy);

title('Corrupted Image');

colormap gray;

colorbar;

%%

imagesc(im);

title('Filtered Image');

colormap gray;

colorbar;

%%

imagesc(k);
```

```
title('Mask for spatial Gaussian');

colormap jet;

colorbar;

%%

% Optimal Values:

fprintf('%s%f₩n%s%f₩n%s%f₩n','Spatial gaussian standard deviation: ',hs,'Intensity
gaussian standard deviation: ', hi, 'Corresponding value of RMSD is ',rmsd0);

[~,~,im] = myBilateralFiltering(I, 0.9*hs, hi);

rmsd1 = myRMSD(I, im);

[~,~,im] = myBilateralFiltering(I, 1.1*hs, hi);

rmsd2 = myRMSD(I,im);

[~,~,im] = myBilateralFiltering(I, hs, 0.9*hi);

rmsd3 = myRMSD(I,im);
```

```
[~,~,im] = myBilateralFiltering(I, hs, 1.1*hi);

rmsd4 = myRMSD(I,im);

%%

% Other Readings:

fprintf('%s₩t%s%f₩n', 'For Case (i)  :', 'the value of RMSD is ', rmsd1);

fprintf('%s₩t%s%f₩n', 'For Case (ii) :', 'the value of RMSD is ', rmsd2);

fprintf('%s₩t%s%f₩n', 'For Case (iii):', 'the value of RMSD is ', rmsd3);

fprintf('%s₩t%s%f₩n', 'For Case (iv) :', 'the value of RMSD is ', rmsd4);

%% c) honeyComb

I = im2double(imread('../data/honeyCombReal.png'));

hs = 0.787;

hi = 0.263;

[k,noisy,im] = myBilateralFiltering(I, hs, hi);
```

```
rmsd0 = myRMSD(I,im);

%%

imagesc(I);

title('Original Image');

colormap gray;

colorbar;

%%

imagesc(noisy);

title('Corrupted Image');

colormap gray;

colorbar;

%%

imagesc(im);
```

```
title('Filtered Image');

colormap gray;

colorbar;

%%

imagesc(k);

title('Mask for spatial Gaussian');

colormap jet;

colorbar;

%%

% Optimal Values:

fprintf('%s%f₩n%s%f₩n%s%f₩n','Spatial gaussian standard deviation: ',hs,'Intensity

gaussian standard deviation: ', hi, 'Corresponding value of RMSD is ',rmsd0);

[~,~,im] = myBilateralFiltering(I, 0.9*hs, hi);
```

```
rmsd1 = myRMSD(I, im);

[~,~,im] = myBilateralFiltering(I, 1.1*hs, hi);

rmsd2 = myRMSD(I,im);

[~,~,im] = myBilateralFiltering(I, hs, 0.9*hi);

rmsd3 = myRMSD(I,im);

[~,~,im] = myBilateralFiltering(I, hs, 1.1*hi);

rmsd4 = myRMSD(I,im);

%%

% Other Readings:

fprintf('%s₩t%s%f₩n', 'For Case (i)  :', 'the value of RMSD is ', rmsd1);

fprintf('%s₩t%s%f₩n', 'For Case (ii) :', 'the value of RMSD is ', rmsd2);

fprintf('%s₩t%s%f₩n', 'For Case (iii):', 'the value of RMSD is ', rmsd3);

fprintf('%s₩t%s%f₩n', 'For Case (iv) :', 'the value of RMSD is ', rmsd4);
```

```
%%

toc;function r = myRMSD(I,J)

    diff = I - J;

    squareDiff = diff.^2;

    squareDiff = squareDiff(:);

    squareDiff(isnan(squareDiff)) = [];

    N = length(squareDiff);

    avrg = sum(squareDiff) / N;

    r = sqrt(avrg);

end%% Assignment-II Report for CS663 - Fundamentals of Digital Image Processing

% This script performs patch-based filtering on some input images

% and automatically publishes the results in a formatted HTML document.

%
```

```
% If images are not displayed side by side, zoom out until they are.

I1 = load('../data/barbara.mat');

I2 = im2double(imread('../data/grass.png'));

I3 = im2double(imread('../data/honeyCombReal.png'));

h1 = 5.94;

h2 = 0.0538;

h3 = 0.0621;

%% Barbara

[k1,noisy1,im1] = myPatchBasedFiltering(I1.imageOrig,h1,1.43);

r11 = myRMSD(I1.imageOrig,im1);

%%

imagesc(I1.imageOrig);

title('Original');
```

```
colormap gray;

colorbar;

%%

imagesc(noisy1);

title('Corrupted');

colormap gray;

colorbar;

%%

imagesc(im1);

title('Filtered');

colormap gray;

colorbar;

%%
```

```
imagesc(k1);

title('Gaussian kernel used on patches');

colormap jet;

colorbar;

%%

fprintf('%s%f%s%f₩n','Optimal value of h for this image is ',h1,' and corresponding value
of RMSD is ',r11);

[~,~,im1] = myPatchBasedFiltering(I1.imageOrig,0.9*h1,1.43);

r12 = myRMSD(I1.imageOrig,im1);

[~,~,im1] = myPatchBasedFiltering(I1.imageOrig,1.1*h1,1.43);

r13 = myRMSD(I1.imageOrig,im1);

fprintf('%s%f₩n','For 90% of optimal value of h, the value of RMSD is ',r12);

fprintf('%s%f₩n','For 110% of optimal value of h, the value of RMSD is ',r13);
```

```
%% Grass

[k2,noisy2,im2] = myPatchBasedFiltering(I2,h2,0.91);

r21 = myRMSD(I2,im2);

%%

imagesc(I2);

title('Original');

colormap gray;

colorbar;

%%

imagesc(noisy2);

title('Corrupted');

colormap gray;

colorbar;
```

```
%%

imagesc(im2);

title('Filtered');

colormap gray;

colorbar;

%%

imagesc(k2);

title('Gaussian kernel used on patches');

colormap jet;

colorbar;

%%

fprintf('%s%f%s%f₩n','Optimal value of h for this image is ',h2,' and corresponding
value of RMSD is ',r21);
```

```
[~,~,im2] = myPatchBasedFiltering(I2,0.9*h2,0.78);

r22 = myRMSD(I2,im2);

[~,~,im2] = myPatchBasedFiltering(I2,1.1*h2,0.78);

r23 = myRMSD(I2,im2);

fprintf('%s%f₩n','For 90% of optimal value of h, the value of RMSD is ',r22);

fprintf('%s%f₩n','For 110% of optimal value of h, the value of RMSD is ',r23);

%% Honey Comb

[k3,noisy3,im3] = myPatchBasedFiltering(I3,h3,0.95);

r31 = myRMSD(I3,im3);

%%

imagesc(I3);

title('Original');

colormap gray;
```

```
colorbar;

%%

imagesc(noisy3);

title('Corrupted');

colormap gray;

colorbar;

%%

imagesc(im3);

title('Filtered');

colormap gray;

colorbar;

%%

imagesc(k3);
```

```
title('Gaussian kernel used on patches');

colormap jet;

colorbar;

%%

fprintf('%s%f%s%f₩n','Optimal value of h for this image is ',h3,' and corresponding
value of RMSD is ',r31);

[~,~,im3] = myPatchBasedFiltering(I3,0.9*h3,0.95);

r32 = myRMSD(I3,im3);

[~,~,im3] = myPatchBasedFiltering(I3,1.1*h3,0.95);

r33 = myRMSD(I3,im3);

fprintf('%s%f₩n','For 90% of optimal value of h, the value of RMSD is ',r32);

fprintf('%s%f₩n','For 110% of optimal value of h, the value of RMSD is ',r33);function
[mykernel,noisyImage,outputImage] = myPatchBasedFiltering(inputImage,h,a)
```

```
% constants

PATCH_SIZE = 9;

WINDOW_SIZE = 29;

% create gaussian kernel

mykernel = fspecial('gaussian',9,a);

kernel = mykernel(:);

% find intensity range in input

inputSize = size(inputImage);

minIntensity = min(inputImage(:));

maxIntensity = max(inputImage(:));

rangeIntensity = maxIntensity - minIntensity;

% add gaussian noise to input

noisyImage = inputImage + 0.05 * rangeIntensity * randn(inputSize);
```

```
% pre-allocate output matrix

outputImage = zeros(inputSize);

% run filtering over all pixels

for y=1:inputSize(1)

    for x=1:inputSize(2)

        % get center patch

        centerPatch = getPatch(noisyImage,PATCH_SIZE,x,y);

        % get window

        window = getWindow(noisyImage,WINDOW_SIZE,x,y);

        % get all patches from window

        patches = im2col(window,[PATCH_SIZE PATCH_SIZE],'sliding');

        % get center pixels from all patches

        centers = patches(ceil(PATCH_SIZE*PATCH_SIZE/2),:);
```

```
        % find square differences between patches

        patches = patches - centerPatch;

        patches = patches.^2;

        % apply gaussian kernel

        patches = patches .* kernel;

        % sum for gaussian weighted euclidean distance

        distances = nansum(patches);

        % calculate weights

        weights = exp(-distances/h^2);

        % calculate output pixel intensity

        outputImage(y,x) = sum(weights .* centers) / sum(weights);

    end

end
```

```
end

function patch = getPatch(I,psize,x,y)

    isize = size(I);

    patch = nan(9);

    % calculate pad values

    leftPad = max(0,5-x);

    rightPad = max(0,x-isize(2)+4);

    topPad = max(0,5-y);

    bottomPad = max(0,y-isize(1)+4);

    % calculate patch boundaries

    left = max(1,x-floor(psize/2));

    right = min(isize(2),x+floor(psize/2));

    top = max(1,y-floor(psize/2));
```

```
    bottom = min(isize(1),y+floor(psize/2));

    % create patch

    patch(topPad+1:9-bottomPad,leftPad+1:9-rightPad) = I(top:bottom,left:right);

    patch = patch(:);

end

function window = getWindow(I,wsize,x,y)

    isize = size(I);

    % calculate window boundaries

    left = max(1,x-floor(wsize/2));

    right = min(isize(2),x+floor(wsize/2));

    top = max(1,y-floor(wsize/2));

    bottom = min(isize(1),y+floor(wsize/2));

    % create window
```

```
    window = I(top:bottom,left:right);

endfunction r = myRMSD(I,J)

    diff = I - J;

    squareDiff = diff.^2;

    squareDiff = squareDiff(:);

    squareDiff(isnan(squareDiff)) = [];

    N = length(squareDiff);

    avrg = sum(squareDiff) / N;

    r = sqrt(avrg);

end
```

filtering assignment

ORIGINALITY REPORT

8%	2%	4%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Marquette University Student Paper	3%
2	Zhong Wan. "Parameterized integer programming models for multifactors representatives apportionment", The Social Science Journal, 2006 Publication	2%
3	gamow.ist.utl.pt Internet Source	1%
4	Submitted to University of Sheffield Student Paper	1%
5	Qingyang Song. "Network selection in an integrated wireless LAN and UMTS environment using mathematical modeling and computing techniques", IEEE Wireless Communications, 6/2005 Publication	1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		

filtering assignment

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34