Signal Processing with Matlab

Namuna Panday & Dibakar Sigdel

March 20, 2017

1 Week-2-Project

```
% this I has each pixcel value from 0 to 256. this is an array
I = imread('digital-images-week2_quizzes-lena.gif');
imshow(I);
imwrite(I,'new.png')

%read or crope specific part of an array
Iread = I(:,:);
Icrop = I(80:250,70:100);

% to scale the data within (0 to 1)
idouble = im2double(I);

%create filter of size 3 by 3 which
%takes avarage of selected window

h3 = fspecial('average', [3,3]);
h5 = fspecial('average', [5,5]);

% apply above defined filter for image array in double
new3 = imfilter(idouble, h3, 'replicate');
new5 = imfilter(idouble, h5, 'replicate');



squarederror = (idouble - new3).^2;
mse3 = sum(sum(squarederror))/(256*256);
psnr3 = 10*log10(1/mse3);


squarederror = (idouble - new5).^2;
mse5 = sum(sum(squarederror))/(256*256);
psnr5 = 10*log10(1/mse5);
```

2 Week-3-Project

```
% step 1
    :-------------------------------------------------------------
% The original image is an 8-bit gray-scale image of
% width 479 and height 359 pixels. Convert the original
% image from type 'uint8' (8-bit integer) to 'double' (real number)

I = imread('digital-images-week3_quizzes-original_quiz.jpg');
idouble = im2double(I);
```

```
% step 2: LPF ------------------------------------------------
% Create a 33 low-pass filter with all
% coefficients equal to 1/9. Perform low-pass filtering
% with this filter using the MATLAB function "imfilter"
% with 'replicate' as the third argument

h3 = [1/9,1/9,1/9;1/9,1/9,1/9;1/9,1/9,1/9];
resultLowPass = imfilter(idouble, h3, 'replicate');
```

```
% step 3: Downsampling--------------------------------
% Obtain the down-sampled image by removing every other row
% and column from the filtered image, that is, removing the 2nd,
% 4th, all the way to the 358th row, and then removing the 2nd,
    4th,
% all the way to the 478th column. The resulting image should be of
% width 240 and height 180 pixles. This completes the procedure
% for image down-sampling. In the next steps, you will up-sample
% this low-resolution image to the original resolution
% via spatial domain processing.
```

```
downsampled = resultLowPass(1:2:end, 1:2:end);



% step 4: Upsampling----------------------------
% Create an all-zero MATLAB array of width 479 and height 359.

padding(1:359,1:479) = 0.0;
% For every odd-valued i[1,359] and odd-valued j[1,479],
% set the value of the newly created array at (i,j) equal
% to the value of the low-resolution image at ((i+1)/2,(j+1)/2).
% After this step you have inserted zeros into the low-resolution
    image.

padding(1:2:end,1:2:end) = downsampled(1:end,1:end);



% step 5: Convolution ------------------------------------
% Convolve the result obtained from step (4) with a filter
% with coefficients [0.25,0.5,0.25;0.5,1,0.5;0.25,0.5,0.25]
% using the MATLAB function "imfilter". In this step you should
% only provide "imfilter" with two arguments instead of three,
% that was the case in step (1). The two arguments are the
% result from step (4) and the filter specified in this step.
% This step essentially performs bilinear interpolation to
% obtain the up-sampled image.

filter = [0.25,0.5,0.25;0.5,1,0.5;0.25,0.5,0.25];
upsampled = imfilter(padding, filter);



% step 6:-------------------------------------------------------
% Compute the PSNR between the upsampled image obtained from
% step (5) and the original image. For more information about
% PSNR, refer to the programming problem in the homework of module
    2
```

```
squarederror = (idouble - upsampled).^2;
mse = sum(sum(squarederror))/(359*479);
psnr = 10*log10(1/mse);



disp(mse);
disp(psnr);
% psnr = 28.1753
```

---

3 Week-4-Project

```
im1 = imread('digital-images-week4_quizzes-frame_1.jpg');
I1 = im2double(im1);
im2 = imread('digital-images-week4_quizzes-frame_2.jpg');
I2 = im2double(im2);

block1 = I1(65:96,81:112);
imshow(block1);

min = 999;
row = 100;
col = 100;

totalrow = 288/2;
totalcol = 352/2;

for i = 1:totalrow-31
    for j = 1:totalcol-31
        block2 = I2(i:i+31,j:j+31);
        s = mae(block1, block2);
        if (s<min)
           min = s;
           row = i;
           col = j;
        end
    end
end

display(row);
display(col);
display(min*255);
imshow(I2(row:row+31,col:col+31));
```

4 Week-5-Project

```
im1 = imread('digital-images-week4_quizzes-frame_1.jpg');
I1 = im2double(im1);
im2 = imread('digital-images-week4_quizzes-frame_2.jpg');
I2 = im2double(im2);

block1 = I1(65:96,81:112);
imshow(block1);

min = 999;
row = 100;
col = 100;

totalrow = 288/2;
totalcol = 352/2;

for i = 1:totalrow-31
    for j = 1:totalcol-31

        block2 = I2(i:i+31,j:j+31);
        s = mae(block1, block2);

        if (s<min)
            min = s;
            row = i;
            col = j;
        end
    end
end

display(row);
display(col);
display(min*255);
imshow(I2(row:row+31,col:col+31));;
```

5 Week-6-Project

```
% inverse filter with thresholding

clear all
close all
clc

% specify the threshold T
T = 1e-1;

%% read in the original, sharp and noise-free image
original = im2double(rgb2gray((imread('original_cameraman.jpg'))));
[H, W] = size(original);

%% generate the blurred and noise-corrupted image for experiment
motion_kernel = ones(1, 9) / 9; % 1-D motion blur
motion_freq = fft2(motion_kernel, 1024, 1024); % frequency
    response of motion blur
original_freq = fft2(original, 1024, 1024);
blurred_freq = original_freq .* motion_freq; % spectrum of blurred
    image
blurred = ifft2(blurred_freq);
blurred = blurred(1 : H, 1 : W);
blurred(blurred < 0) = 0;
blurred(blurred > 1) = 1;
noisy = imnoise(blurred, 'gaussian', 0, 1e-4);


%% Restoration from blurred and noise-corrupted image
% generate restoration filter in the frequency domain
inverse_freq = zeros(size(motion_freq));
inverse_freq(abs(motion_freq) < T) = 0;
inverse_freq(abs(motion_freq) >= T) = 1 ./
    motion_freq(abs(motion_freq) >= T);
% spectrum of blurred and noisy-corrupted image (the input to
    restoration)
noisy_freq = fft2(noisy, 1024, 1024);
```

```
% restoration
restored_freq = noisy_freq .* inverse_freq;
restored = ifft2(restored_freq);
restored = restored(1 : H, 1 : W);
restored(restored < 0) = 0;
restored(restored > 1) = 1;

%% analysis of result
noisy_psnr = 10 * log10(1 / (norm(original - noisy, 'fro') ^ 2 / H
    / W));
restored_psnr = 10 * log10(1 / (norm(original - restored, 'fro') ^
    2 / H / W));


%% visualization
figure; imshow(original, 'border', 'tight');
figure; imshow(blurred, 'border', 'tight');
figure; imshow(noisy, 'border', 'tight');
figure; imshow(restored, 'border', 'tight');
figure; plot(abs(fftshift(motion_freq(1, :)))); title('spectrum of
    motion blur'); xlim([0 1024]);
figure; plot(abs(fftshift(inverse_freq(1, :)))); title('spectrum
    of inverse filter'); xlim([0 1024]);
```

6 Week-7-Project

6.1 cls restoration

```
function image_restored = cls_restoration(image_noisy, psf, alpha)

%% find proper dimension for frequency-domain processing
[image_height, image_width] = size(image_noisy);
[psf_height, psf_width] = size(psf);
dim = max([image_width, image_height, psf_width, psf_height]);
dim = next2pow(dim);

%% frequency-domain representation of degradation
psf = padarray(psf, [dim - psf_height, dim - psf_width], 'post');
psf = circshift(psf, [-(psf_height - 1) / 2, -(psf_width - 1) /
    2]);
H = fft2(psf, dim, dim);

%% frequency-domain representation of Laplace operator
Laplace = [0, -0.25, 0; -0.25, 1, -0.25; 0, -0.25, 0];
Laplace = padarray(Laplace, [dim - 3, dim - 3], 'post');
Laplace = circshift(Laplace, [-1, -1]);
C = fft2(Laplace, dim, dim);

%% Frequency response of the CLS filter
% Refer to the lecture for frequency response of CLS filter
% Complete the implementation of the CLS filter by uncommenting the
% following line and adding appropriate content
R = conj(H) ./ ((H .* conj(H)) + alpha * (C .* conj(C)));

%% CLS filtering
Y = fft2(image_noisy, dim, dim);
image_restored_frequency = R .* Y;
image_restored = ifft2(image_restored_frequency);
image_restored = image_restored(1 : image_height, 1 : image_width);
```

6.2 insr

```
function ratio = isnr(image_original, image_noisy, image_restored)
%ISNR Summary of this function goes here
%   Detailed explanation goes here

  ratio = 10 * log10((norm(image_original - image_noisy, 'fro') ^
      2) / ( norm(image_original - image_restored, 'fro') ^ 2) );
end
```

6.3 next2pow

```
function result = next2pow(input)
if input <= 0
    fprintf('Error: input must be positive!\n');
    result = -1;
else
    index = 0;
    while 2 ^ index < input
        index = index + 1;
    end
    result = 2 ^ index;
end
```

6.4 wrapper

```
clear all
close all


%% Simulate 1-D blur and noise
image_original = im2double(imread('Cameraman256.bmp', 'bmp'));
[H, W] = size(image_original);
blur_impulse = fspecial('motion', 7, 0);
image_blurred = imfilter(image_original, blur_impulse, 'conv',
    'circular');
noise_power = 1e-4;
randn('seed', 1);
noise = sqrt(noise_power) * randn(H, W);
image_noisy = image_blurred + noise;

figure; imshow(image_original, 'border', 'tight');
figure; imshow(image_blurred, 'border', 'tight');
figure; imshow(image_noisy, 'border', 'tight');

%% CLS restoration
alpha = 1;
values = [0.1];

for alpha=values
  image_cls_restored = cls_restoration(image_noisy, blur_impulse,
      alpha);

  ISNR = isnr(image_original, image_noisy, image_cls_restored);
  disp(['alpha = ' num2str(alpha), ' - ISNR = ' num2str(ISNR)]);

  lbl = strcat('ALPHA: ', num2str(alpha), ' ISNR:', num2str(ISNR));
  figure('Name', lbl); imshow(image_cls_restored, 'border',
      'tight');
end
```

7 Week-8-Project

```
I = imread('Cameraman256.bmp');
idouble = im2double(I);
imshow(idouble);
e = entropy(idouble);
display(e);
```

8 Week-9-Project

```
I = imread('Cameraman256.bmp');
idouble = im2double(I);
%imshow(idouble);

imwrite(idouble,'compressed.jpg','jpg', 'quality', 75);
idouble2 = im2double(imread('compressed.jpg'));
%imshow(idouble2);
squarederror = (idouble - idouble2).^2;
mse = sum(sum(squarederror))/(256*256);
psnr = 10*log10(1/mse);
display(psnr);

imwrite(idouble,'compressed.jpg','jpg', 'quality', 10);
idouble2 = im2double(imread('compressed.jpg'));
%imshow(idouble2);
squarederror = (idouble - idouble2).^2;
mse = sum(sum(squarederror))/(256*256);
psnr = 10*log10(1/mse);
display(psnr);
```