**ASSIGNMENT - 1**

**IMAGE PROCESSING**

**PRIYANSHU SHARMA**

**15BCE1282**

**QUESTION**

**CODE -**

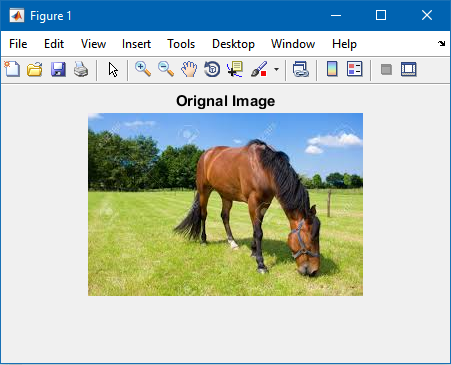
clc

clear all

a=imread('C:\Users\PRIYANSHU SHARMA\Desktop\PRIYANSHU\6 STUDY\5 SEMSTER\CBIR\MATLAB\LAB 6\1.jpg');

figure, imshow(a)

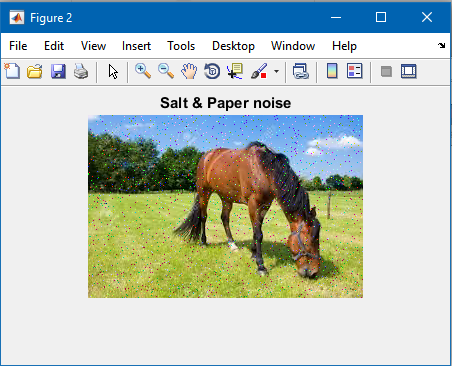
title('Orignal Image');



b= imnoise(a,'salt & pepper',0.02);

figure, imshow(b)

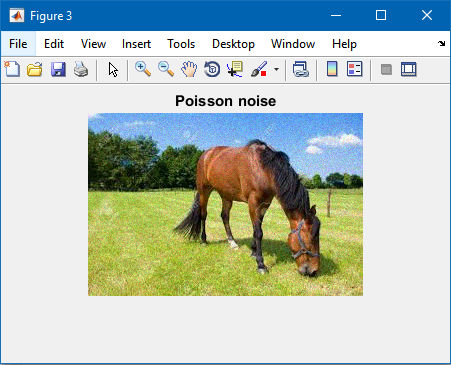
title('Salt & Paper noise');



c=imnoise(a,'poisson');

figure, imshow(c)

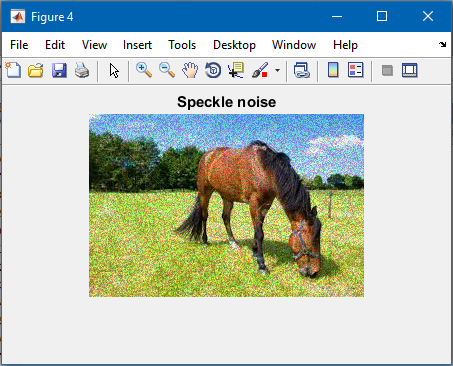
title('Poisson noise');



d = imnoise(a,'speckle',0.04);

figure, imshow(d)

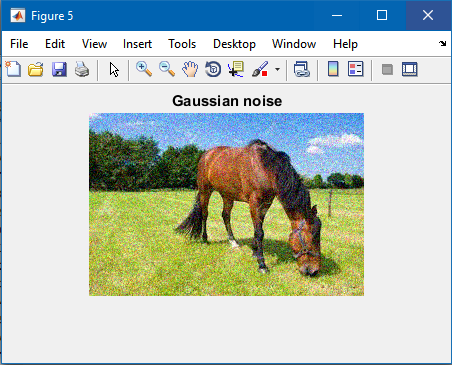
title('Speckle noise');



e = imnoise(a,'gaussian',0,0.01);

figure, imshow(e)

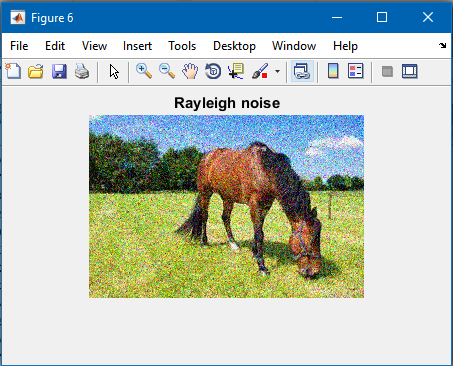
title('Gaussian noise');



f = imnoise(a,'localvar',0.05\*rand(size(a)));

figure, imshow(f)

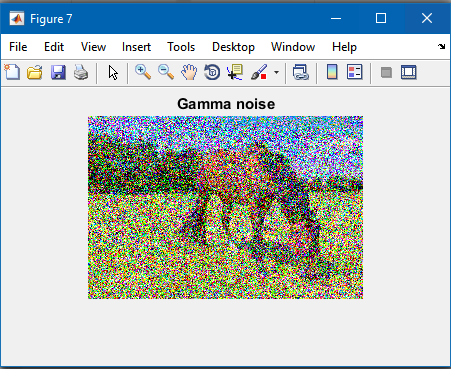
title('Rayleigh noise');



g = imnoise(a,'localvar',rand(1,100),rand(1,100));

figure, imshow(g)

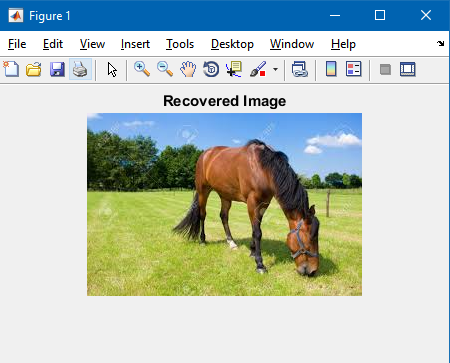
title('Gamma noise');



h=medfilt2(f);

figure, imshow(h)

title('Recovered Image');



**FUNCTION USED**

1. imnoise()

function b = imnoise(varargin)

%IMNOISE Add noise to image.

% J = IMNOISE(I,TYPE,...) Add noise of a given TYPE to the intensity image

% I. TYPE is a string that can have one of these values:

%

% 'gaussian' Gaussian white noise with constant

% mean and variance

%

% 'localvar' Zero-mean Gaussian white noise

% with an intensity-dependent variance

%

% 'poisson' Poisson noise

%

% 'salt & pepper' "On and Off" pixels

%

% 'speckle' Multiplicative noise

%

% Depending on TYPE, you can specify additional parameters to IMNOISE. All

% numerical parameters are normalized; they correspond to operations with

% images with intensities ranging from 0 to 1.

%

% J = IMNOISE(I,'gaussian',M,V) adds Gaussian white noise of mean M and

% variance V to the image I. When unspecified, M and V default to 0 and

% 0.01 respectively.

%

% J = imnoise(I,'localvar',V) adds zero-mean, Gaussian white noise of

% local variance, V, to the image I. V is an array of the same size as I.

%

% J = imnoise(I,'localvar',IMAGE\_INTENSITY,VAR) adds zero-mean, Gaussian

% noise to an image, I, where the local variance of the noise is a

% function of the image intensity values in I. IMAGE\_INTENSITY and VAR

% are vectors of the same size, and PLOT(IMAGE\_INTENSITY,VAR) plots the

% functional relationship between noise variance and image intensity.

% IMAGE\_INTENSITY must contain normalized intensity values ranging from 0

% to 1.

%

% J = IMNOISE(I,'poisson') generates Poisson noise from the data instead

% of adding artificial noise to the data. If I is double precision,

% then input pixel values are interpreted as means of Poisson

% distributions scaled up by 1e12. For example, if an input pixel has

% the value 5.5e-12, then the corresponding output pixel will be

% generated from a Poisson distribution with mean of 5.5 and then scaled

% back down by 1e12. If I is single precision, the scale factor used is

% 1e6. If I is uint8 or uint16, then input pixel values are used

% directly without scaling. For example, if a pixel in a uint8 input

% has the value 10, then the corresponding output pixel will be

% generated from a Poisson distribution with mean 10.

%

% J = IMNOISE(I,'salt & pepper',D) adds "salt and pepper" noise to the

% image I, where D is the noise density. This affects approximately

% D\*numel(I) pixels. The default for D is 0.05.

%

% J = IMNOISE(I,'speckle',V) adds multiplicative noise to the image I,

% using the equation J = I + n\*I, where n is uniformly distributed random

% noise with mean 0 and variance V. The default for V is 0.04.

%

% Note

% ----

% The mean and variance parameters for 'gaussian', 'localvar', and

% 'speckle' noise types are always specified as if for a double image

% in the range [0, 1]. If the input image is of class uint8 or uint16,

% the imnoise function converts the image to double, adds noise

% according to the specified type and parameters, and then converts the

% noisy image back to the same class as the input.

%

% Class Support

% -------------

% For most noise types, I can be uint8, uint16, double, int16, or

% single. For Poisson noise, int16 is not allowed. The output

% image J has the same class as I. If I has more than two dimensions

% it is treated as a multidimensional intensity image and not as an

% RGB image.

%

% Example

% -------

% I = imread('eight.tif');

% J = imnoise(I,'salt & pepper', 0.02);

% figure, imshow(I), figure, imshow(J)

%

% See also RAND, RANDN.

% Copyright 1993-2013 The MathWorks, Inc.

[a, code, classIn, classChanged, p3, p4] = ParseInputs(varargin{:});

clear varargin;

b = images.internal.algimnoise(a, code, classIn, classChanged, p3, p4);

%%%

%%% ParseInputs

%%%

function [a, code, classIn, classChanged, p3, p4, msg] = ParseInputs(varargin)

% Initialization

p3 = [];

p4 = [];

msg = '';

% Check the number of input arguments.

narginchk(1,4);

% Check the input-array type.

a = varargin{1};

validateattributes(a, {'uint8','uint16','double','int16','single'}, {}, mfilename, ...

'I', 1);

% Change class to double

classIn = class(a);

classChanged = 0;

if ~isa(a, 'double')

a = im2double(a);

classChanged = 1;

else

% Clip so a is between 0 and 1.

a = max(min(a,1),0);

end

% Check the noise type.

if nargin > 1

if ~ischar(varargin{2})

error(message('images:imnoise:invalidNoiseType'))

end

% Preprocess noise type string to detect abbreviations.

allStrings = {'gaussian', 'salt & pepper', 'speckle',...

'poisson','localvar'};

idx = find(strncmpi(varargin{2}, allStrings, numel(varargin{2})));

switch length(idx)

case 0

error(message('images:imnoise:unknownNoiseType', varargin{ 2 }))

case 1

code = allStrings{idx};

otherwise

error(message('images:imnoise:ambiguousNoiseType', varargin{ 2 }))

end

else

code = 'gaussian'; % default noise type

end

switch code

case 'poisson'

if nargin > 2

error(message('images:imnoise:tooManyPoissonInputs'))

end

if isa(a, 'int16')

error(message('images:imnoise:badClassForPoisson'));

end

case 'gaussian'

p3 = 0; % default mean

p4 = 0.01; % default variance

if nargin > 2

p3 = varargin{3};

if ~isRealScalar(p3)

error(message('images:imnoise:invalidMean'))

end

end

if nargin > 3

p4 = varargin{4};

if ~isNonnegativeRealScalar(p4)

error(message('images:imnoise:invalidVariance', 'gaussian'))

end

end

case 'salt & pepper'

p3 = 0.05; % default density

if nargin > 2

p3 = varargin{3};

if ~isNonnegativeRealScalar(p3) || (p3 > 1)

error(message('images:imnoise:invalidNoiseDensity'))

end

if nargin > 3

error(message('images:imnoise:tooManySaltAndPepperInputs'))

end

end

case 'speckle'

p3 = 0.05; % default variance

if nargin > 2

p3 = varargin{3};

if ~isNonnegativeRealScalar(p3)

error(message('images:imnoise:invalidVariance', 'speckle'))

end

end

if nargin > 3

error(message('images:imnoise:tooManySpeckleInputs'))

end

case 'localvar'

if nargin < 3

error(message('images:imnoise:toofewLocalVarInputs'))

elseif nargin == 3

% IMNOISE(a,'localvar',v)

code = 'localvar\_1';

p3 = varargin{3};

if ~isNonnegativeReal(p3) || ~isequal(size(p3),size(a))

error(message('images:imnoise:invalidLocalVarianceValueAndSize'))

end

elseif nargin == 4

% IMNOISE(a,'localvar',IMAGE\_INTENSITY,NOISE\_VARIANCE)

code = 'localvar\_2';

p3 = varargin{3};

p4 = varargin{4};

if ~isNonnegativeRealVector(p3) || (any(p3) > 1)

error(message('images:imnoise:invalidImageIntensity'))

end

if ~isNonnegativeRealVector(p4)

error(message('images:imnoise:invalidLocalVariance'))

end

if ~isequal(size(p3),size(p4))

error(message('images:imnoise:invalidSize'))

end

else

error(message('images:imnoise:tooManyLocalVarInputs'))

end

end

%%%

%%% isReal

%%%

function t = isReal(P)

% isReal(P) returns 1 if P contains only real

% numbers and returns 0 otherwise.

%

isFinite = all(isfinite(P(:)));

t = isreal(P) && isFinite && ~isempty(P);

%%%

%%% isNonnegativeReal

%%%

function t = isNonnegativeReal(P)

% isNonnegativeReal(P) returns 1 if P contains only real

% numbers greater than or equal to 0 and returns 0 otherwise.

%

t = isReal(P) && all(P(:)>=0);

%%%

%%% isRealScalar

%%%

function t = isRealScalar(P)

% isRealScalar(P) returns 1 if P is a real,

% scalar number and returns 0 otherwise.

%

t = isReal(P) && (numel(P)==1);

%%%

%%% isNonnegativeRealScalar

%%%

function t = isNonnegativeRealScalar(P)

% isNonnegativeRealScalar(P) returns 1 if P is a real,

% scalar number greater than 0 and returns 0 otherwise.

%

t = isReal(P) && all(P(:)>=0) && (numel(P)==1);

%%%

%%% isVector

%%%

function t = isVector(P)

% isVector(P) returns 1 if P is a vector and returns 0 otherwise.

%

t = ((numel(P) >= 2) && ((size(P,1) == 1) || (size(P,2) == 1)));

%%%

%%% isNonnegativeRealVector

%%%

function t = isNonnegativeRealVector(P)

% isNonnegativeRealVector(P) returns 1 if P is a real,

% vector greater than 0 and returns 0 otherwise.

%

t = isReal(P) && all(P(:)>=0) && isVector(P);

1. medfilt2

function b = medfilt2(varargin)

%MEDFILT2 2-D median filtering.

% B = MEDFILT2(A,[M N]) performs median filtering of the matrix

% A in two dimensions. Each output pixel contains the median

% value in the M-by-N neighborhood around the corresponding

% pixel in the input image. MEDFILT2 pads the image with zeros

% on the edges, so the median values for the points within

% [M N]/2 of the edges may appear distorted.

%

% B = MEDFILT2(A) performs median filtering of the matrix A

% using the default 3-by-3 neighborhood.

%

% B = MEDFILT2(...,PADOPT) controls how the matrix boundaries

% are padded. PADOPT may be 'zeros' (the default),

% 'symmetric', or 'indexed'. If PADOPT is 'zeros', A is padded

% with zeros at the boundaries. If PADOPT is 'symmetric', A is

% symmetrically extended at the boundaries. If PADOPT is

% 'indexed', A is padded with ones if it is double; otherwise

% it is padded with zeros.

%

% Class Support

% -------------

% The input image A can be logical or numeric. The output image B is of

% the same class as A.

%

% Remarks

% -------

% If the input image A is of integer class, all of the output

% values are returned as integers. If the number of

% pixels in the neighborhood (i.e., M\*N) is even, some of the

% median values may not be integers. In these cases, the

% fractional parts are discarded. Logical input is treated

% similarly.

%

% Example

% -------

% I = imread('eight.tif');

% J = imnoise(I,'salt & pepper',0.02);

% K = medfilt2(J);

% figure, imshow(J), figure, imshow(K)

%

% See also FILTER2, ORDFILT2, WIENER2.

% Copyright 1993-2014 The MathWorks, Inc.

narginchk(1,3);

[a, mn, padopt] = parse\_inputs(varargin{:});

if isempty(a)

b = a;

return

end

% switch to IPP iff

% UseIPPL preference is true .AND.

% kernel is odd .AND.

% input data type is single .AND. kernel size is == 3x3

% .OR. input data type is (int16 .OR. uint8 .OR. uint16) .AND. kernel size

% is between 3x3 and 19x19

domain = ones(mn);

if (rem(prod(mn), 2) == 1)

tf = hUseIPPL(a, mn);

if tf

a = hPadImage(a,mn, padopt);

b = medianfiltermex(a, [mn(1) mn(2)]);

else

order = (prod(mn)+1)/2;

b = ordfilt2(a, order, domain, padopt);

end

else

order1 = prod(mn)/2;

order2 = order1+1;

b = ordfilt2(a, order1, domain, padopt);

b2 = ordfilt2(a, order2, domain, padopt);

if islogical(b)

b = b | b2;

else

b = imlincomb(0.5, b, 0.5, b2);

end

end

%%%

%%% Function parse\_inputs

%%%

function [a, mn, padopt] = parse\_inputs(varargin)

% Any syntax in which 'indexed' is followed by other arguments is discouraged.

%

% We have to catch and parse this successfully, so we're going to use a strategy

% that's a little different that usual.

%

% First, scan the input argument list for strings. The

% string 'indexed', 'zeros', or 'symmetric' can appear basically

% anywhere after the first argument.

%

% Second, delete the strings from the argument list.

%

% The remaining argument list can be one of the following:

% MEDFILT2(A)

% MEDFILT2(A,[M N])

a = varargin{1};

% validate that the input is a 2D, real, numeric or logical matrix.

validateattributes(a, ...

{'uint8','uint16','uint32','int8','int16','int32','single','double','logical'},...

{'2d','real','nonsparse'}, mfilename, 'A', 1);

charLocation = [];

for k = 2:nargin

if (ischar(varargin{k}))

charLocation = [charLocation k]; %#ok<AGROW>

end

end

if (length(charLocation) > 1)

% More than one string in input list

error(message('images:medfilt2:tooManyStringInputs'));

elseif isempty(charLocation)

% No string specified

padopt = 'zeros';

else

options = {'indexed', 'zeros', 'symmetric'};

padopt = validatestring(varargin{charLocation}, options, mfilename, ...

'PADOPT', charLocation);

varargin(charLocation) = [];

end

if (strcmp(padopt, 'indexed'))

if (isa(a,'double'))

padopt = 'ones';

else

padopt = 'zeros';

end

end

if length(varargin) == 1,

mn = [3 3];% default

elseif length(varargin) >= 2

mn = varargin{2}(:)';

validateattributes(mn,{'numeric'},{'real','positive','integer','nonempty','size',[1 2]},...

mfilename,'[M N]',2);

if length(varargin) > 2

% Error if more than one [M N] is specified

error(message('images:medfilt2:invalidSyntax'));

end

end

% ------------------------------------------------------------------------

function tf = hUseIPPL(a, mn)

% switch to IPP iff

% UseIPPL preference is true .AND.

% kernel is odd .AND.

% input data type is single .AND. kernel size is == 3x3

% .OR. input data type is uint8 .AND. kernel size is

% 1xn, n<=5

% .OR. nx1, n<=7

% .OR. between 3x3 and 19x19

% .OR. input data type is (int16 .OR. uint16) .AND. kernel size

% is between 3x3 and 19x19

tf = false;

switch class(a)

case 'single'

if all(mn==[3 3])

tf = true;

end

case 'uint8'

if (mn(1)==1 && mn(2)<=5) || (all(mn >= [3 3]) && all(mn <= [19 19])) || (mn(2)==1 && mn(1)<=7)

tf = true;

end

case {'uint16', 'int16'}

if all(mn >= [3 3]) && all(mn <= [19 19])

tf = true;

end

end

tf = tf & iptgetpref('UseIPPL');

% -------------------------------------------------------------------------

function A = hPadImage(A, domainSize, padopt)

% pad the image suitably -

center = floor((domainSize + 1) / 2);

padSize = domainSize-center;

if (strcmp(padopt, 'zeros'))

A = padarray(A, padSize, 0, 'both');

elseif (strcmp(padopt, 'symmetric'))

A = padarray(A, padSize, 'symmetric', 'both');

else

% This block should never be reached.

error(message('images:medfilt2:incorrectPaddingOption'))

end