Editor - I:\code\main.m

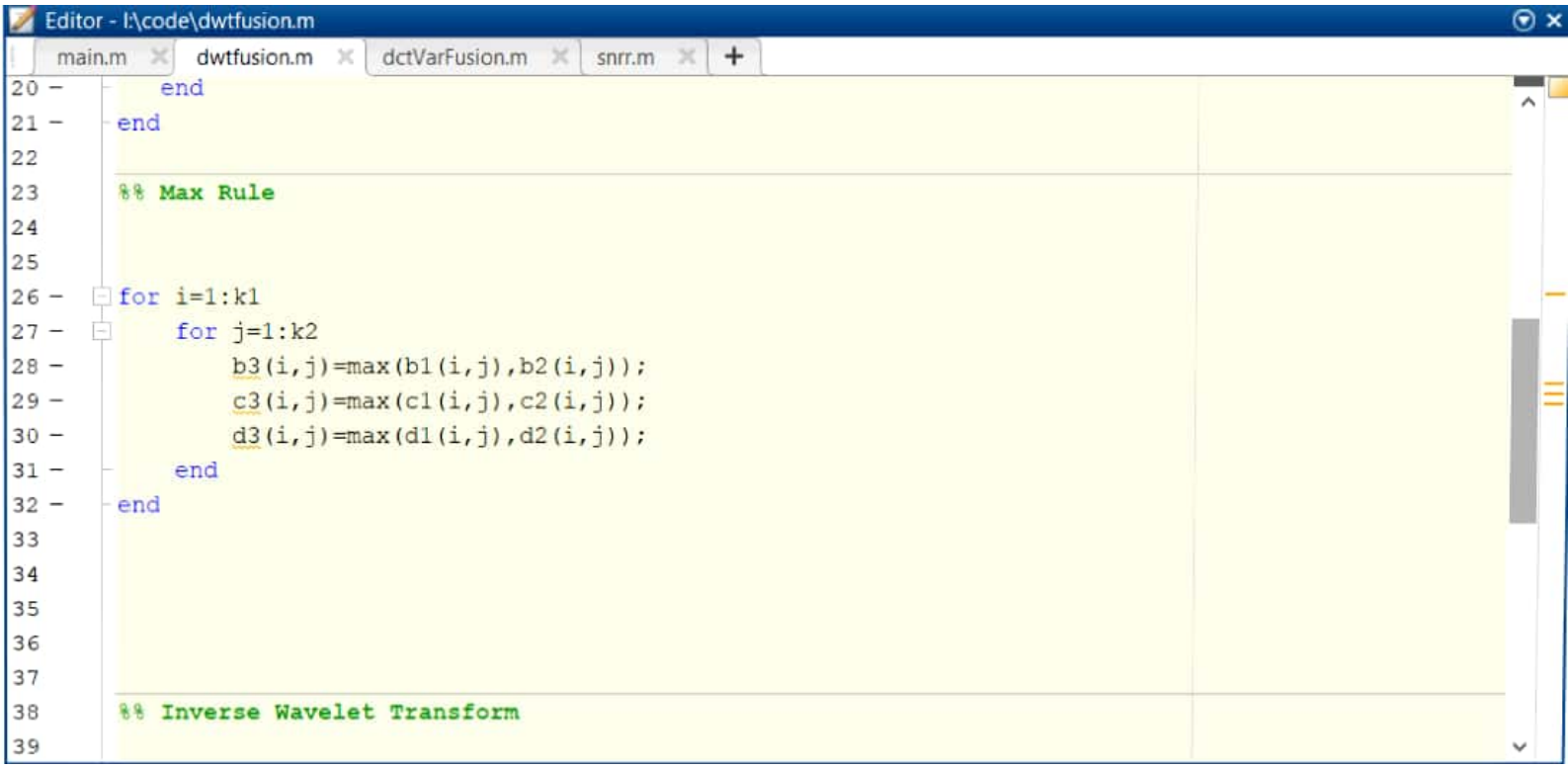
main.m | dwtfusion.m | dctVarFusion.m | snrr.m

```
1 -  close all;
2 -  clear;
3 -  clc;
4    %% Read Images
5
6 -  [file, pathname] = uigetfile('*.jpg','Load Image 1 ');cd(pathname);
7 -  im1=imread(file);
8 -  [file, pathname] = uigetfile('*.jpg','Load Image 2 ');cd(pathname);
9 -  im2=imread(file);
10 - subplot(2,2,1), imshow(im1), title('Source image 1');
11 - subplot(2,2,2), imshow(im2), title('Source image 2');
12
13 - dwtfusion(im1,im2);
14 - dctVarFusion(im1, im2);
15
```

Editor - I:\code\dwtfusion.m

main.m | dwtfusion.m | dctVarFusion.m | snrr.m

```
function im3 = dwtfusion(im1, im2)




%%   Wavelet Transform


[a1,b1,c1,d1]=dwt2(im1,'db2');
[a2,b2,c2,d2]=dwt2(im2,'db2');

[k1,k2]=size(a1);

%% Fusion Rules

%% Average Rule

for i=1:k1
    for j=1:k2
        a3(i,j)=(a1(i,j)+a2(i,j))/2;
   end
```


Editor - I:\code\dwtfusion.m
main.m
dwtfusion.m
dctVarFusion.m
snrr.m
end
end
%% Max Rule
for i=1:k1
for j=1:k2
b3(i,j)=max(b1(i,j),b2(i,j));
c3(i,j)=max(c1(i,j),c2(i,j));
d3(i,j)=max(d1(i,j),d2(i,j));
end
end
%% Inverse Wavelet Transform

Editor - I:\code\dwtfusion.m

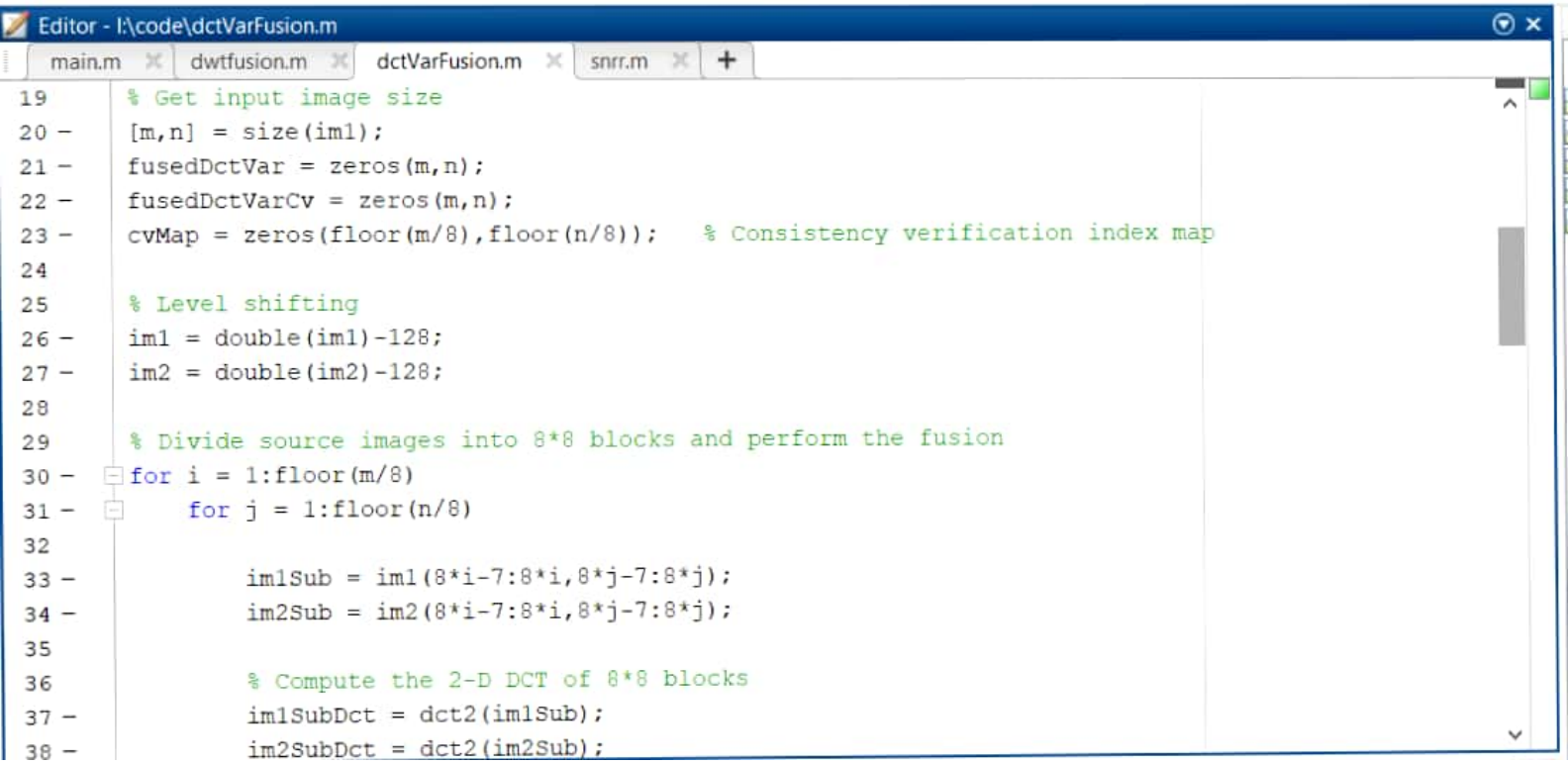
main.m | dwtfusion.m | dctVarFusion.m | snrr.m | +

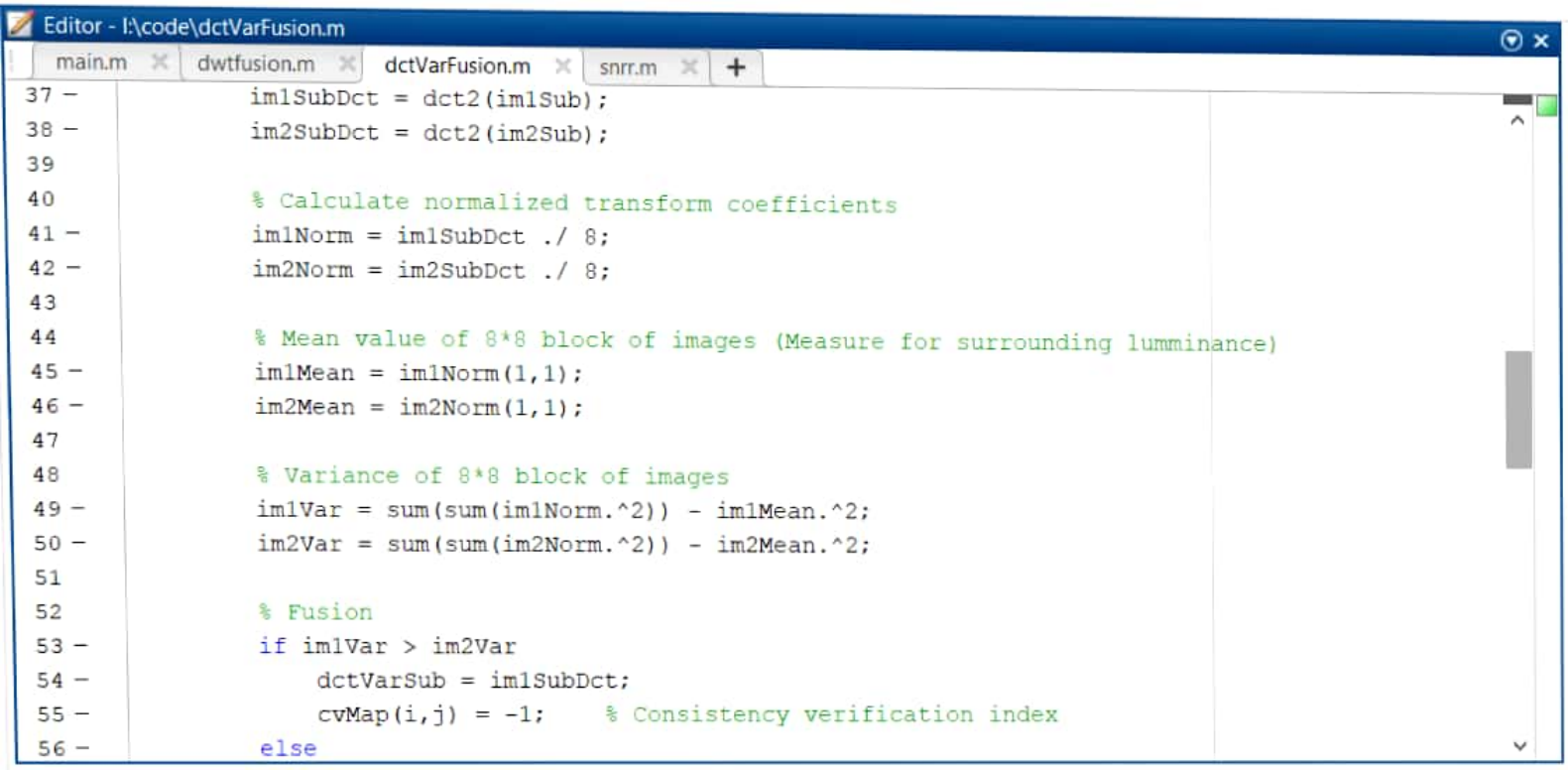
```
38    %% Inverse Wavelet Transform
39
40 -  im3=idwt2(a3,b3,c3,d3,'db2');
41
42 -  subplot(2,2,3), imshow(im3,[]), title('Fused Image using DWT');
43
44
45    %% Performance Criteria
46
47 -  CR1=corr2(im1,im3);
48 -  CR2=corr2(im2,im3);
49 -  S1=snrr(double(im1),double(im3));
50 -  S2=snrr(double(im2),double(im3));
51
52
53 -  fprintf('Correlation between first image and dwt fused image =%f \n\n',CR1);
54 -  fprintf('Correlation between second image and dwt fused image =%f \n\n',CR2);
55 -  fprintf('SNR between first image and dwt fused image =%4.2f db\n\n',S1);
56 -  fprintf('SNR between second image and dwt fused image =%4.2f db \n\n',S2);
57
```

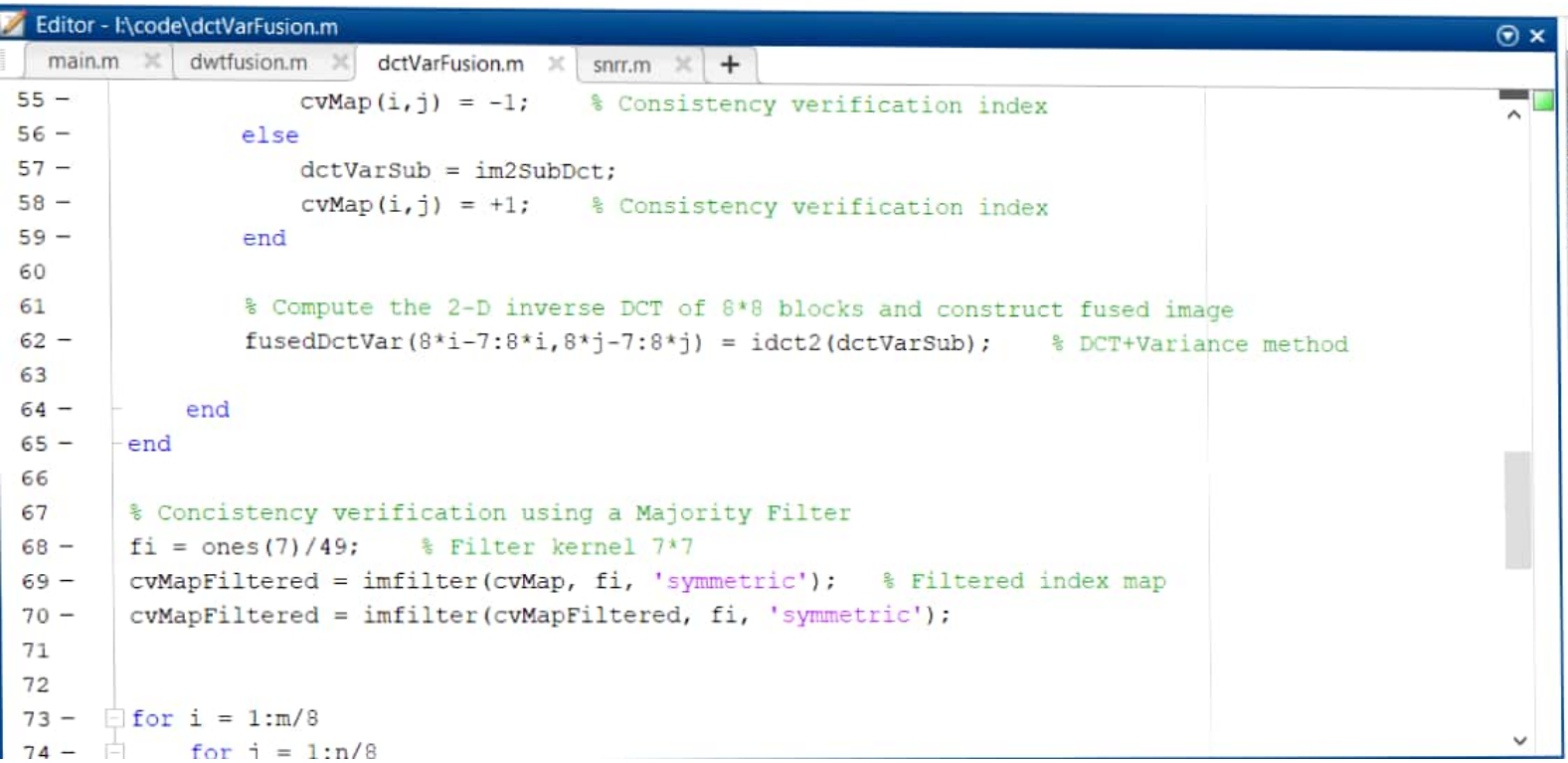
Editor - I:\code\dctVarFusion.m

main.m | dwtfusion.m | dctVarFusion.m | snrr.m

```
1   function [fusedDctVarCv, fusedDctVar] = dctVarFusion(im1, im2)
2
3   if nargin ~= 2  % Check the correct number of arguments
4       error('There should be two input images!')
5   end
6
7   if size(im1,3) == 3     % Check if the images are grayscale
8       im1 = rgb2gray(im1);
9   end
10  if size(im2,3) == 3
11      im2 = rgb2gray(im2);
12  end
13
14  if size(im1) ~= size(im2)   % Check if the input images are of the same size
15      error('Size of the source images must be the same!')
16  end
17
18
19  % Get input image size
20  [m,n] = size(im1);
```


Editor - I:\code\dctVarFusion.m
main.m
dwtfusion.m
dctVarFusion.m
snrr.m
19 % Get input image size
20 - [m,n] = size(im1);
21 - fusedDctVar = zeros(m,n);
22 - fusedDctVarCv = zeros(m,n);
23 - cvMap = zeros(floor(m/8),floor(n/8)); % Consistency verification index map
24
25 % Level shifting
26 - im1 = double(im1)-128;
27 - im2 = double(im2)-128;
28
29 % Divide source images into 8*8 blocks and perform the fusion
30 - for i = 1:floor(m/8)
31 - for j = 1:floor(n/8)
32
33 - im1Sub = im1(8*i-7:8*i,8*j-7:8*j);
34 - im2Sub = im2(8*i-7:8*i,8*j-7:8*j);
35
36 % Compute the 2-D DCT of 8*8 blocks
37 - im1SubDct = dct2(im1Sub);
38 - im2SubDct = dct2(im2Sub);


Editor - I:\code\dctVarFusion.m
main.m
dwtfusion.m
dctVarFusion.m
snrr.m
im1SubDct = dct2(im1Sub);
im2SubDct = dct2(im2Sub);
% Calculate normalized transform coefficients
im1Norm = im1SubDct ./ 8;
im2Norm = im2SubDct ./ 8;
% Mean value of 8*8 block of images (Measure for surrounding lumminance)
im1Mean = im1Norm(1,1);
im2Mean = im2Norm(1,1);
% Variance of 8*8 block of images
im1Var = sum(sum(im1Norm.^2)) - im1Mean.^2;
im2Var = sum(sum(im2Norm.^2)) - im2Mean.^2;
% Fusion
if im1Var > im2Var
dctVarSub = im1SubDct;
cvMap(i,j) = -1;
% Consistency verification index
else


Editor - I:\code\dctVarFusion.m
main.m
dwtfusion.m
dctVarFusion.m
snrr.m
55 - cvMap(i,j) = -1; % Consistency verification index
56 - else
57 - dctVarSub = im2SubDct;
58 - cvMap(i,j) = +1; % Consistency verification index
59 - end
60
61 % Compute the 2-D inverse DCT of 8*8 blocks and construct fused image
62 - fusedDctVar(8*i-7:8*i,8*j-7:8*j) = idct2(dctVarSub); % DCT+Variance method
63
64 - end
65 - end
66
67 % Concistency verification using a Majority Filter
68 - fi = ones(7)/49; % Filter kernel 7*7
69 - cvMapFiltered = imfilter(cvMap, fi, 'symmetric'); % Filtered index map
70 - cvMapFiltered = imfilter(cvMapFiltered, fi, 'symmetric');
71
72
73 - for i = 1:m/8
74 - for j = 1:n/8

Editor - I:\code\dctVarFusion.m

main.m | dwtfusion.m | dctVarFusion.m | snrr.m | +

```
73 -    for i = 1:m/8
74 -        for j = 1:n/8
75              % DCT+Variance+CV method
76 -            if cvMapFiltered(i,j) <= 0
77 -                fusedDctVarCv(8*i-7:8*i,8*j-7:8*j) = im1(8*i-7:8*i,8*j-7:8*j);
78 -            else
79 -                fusedDctVarCv(8*i-7:8*i,8*j-7:8*j) = im2(8*i-7:8*i,8*j-7:8*j);
80 -            end
81
82 -        end
83 -    end
84
85
86 -    fusedDctVar = uint8(double(fusedDctVar)+128);
87 -    fusedDctVarCv = uint8(double(fusedDctVarCv)+128);
88 -    subplot(2,2,4), imshow(fusedDctVarCv), title('"DCT+Variance+CV"  fusion result');
89
90 -    CR3 = corr2(im1,fusedDctVarCv);
91 -    CR4 = corr2(im2,fusedDctVarCv);
92 -    S3 = snrr(double(im1),double(fusedDctVarCv));
```

Editor - I:\code\dctVarFusion.m

main.m | dwtfusion.m | dctVarFusion.m | snrr.m

```
81
82 -        end
83 -    end
84
85
86 -    fusedDctVar = uint8(double(fusedDctVar)+128);
87 -    fusedDctVarCv = uint8(double(fusedDctVarCv)+128);
88 -    subplot(2,2,4), imshow(fusedDctVarCv), title('"DCT+Variance+CV"  fusion result');
89
90 -    CR3 = corr2(im1,fusedDctVarCv);
91 -    CR4 = corr2(im2,fusedDctVarCv);
92 -    S3 = snrr(double(im1),double(fusedDctVarCv));
93 -    S4 = snrr(double(im2),double(fusedDctVarCv));
94
95
96 -    fprintf('Correlation between first image and dct fused image =%f \n\n',CR3);
97 -    fprintf('Correlation between second image and dct fused image =%f \n\n',CR4);
98 -    fprintf('SNR between first image and dct fused image =%4.2f db\n\n',S3);
99 -    fprintf('SNR between second image and dct fused image =%4.2f db \n\n',S4);
100
```

Editor - I:\code\snrr.m

main.m | dwtfusion.m | dctVarFusion.m | snrr.m

```
1     function r = snrr(in, est)
2
3
4 -   error = in - est;
5 -   r = 10 * log10((255^2)/ mean(error(:).^2));
```