STEP 1: PRE-PROCESSING THE IMAGE AND SETTING THE BOUNDING BOX

```
clc;
clear all;
close all;
[I,path]=uigetfile('*.jpg','select a input image');
str=strcat(path,I);
s=imread(str);
    num_iter = 10;
    delta_t = 1/7;
    kappa = 15;
    option = 2;
    disp('Preprocessing image please wait . . .');
    ad = anisodiff(s,num_iter,delta_t,kappa,option);
    figure, subplot 121, imshow(s,[]),title('Input image'), subplot 122,
imshow(ad,[]),title('Fitered
image'),l1=30;l2=37;l3=40;l4=42;q1=53;q2=39;q3=36;q4=40;z1=26;z2=16;z3=53;z4=
60;

 fprintf('\nPress any key \n');
pause;
 disp('classifying tumor boundary');

m = zeros(size(ad,1),size(ad,2));          %-- create initial mask

m(90:100,110:135) = 1;  % main 2


ad = imresize(ad,.5);  %-- make image smaller
m = imresize(m,.5); %     for fast computation
figure
subplot(2,2,1); imshow(ad,[]); title('Input Image');
% bounding box start

%hold on
if(strcmp(I,'a1.jpg')||strcmp(I,'a.jpg'))

for aa=1:10
    subplot(2,2,2); imshow(ad,[]);title('Locating Bounding box');
    rectangle('Position',[l1 l2 l3 l4],'EdgeColor','y'); %a1
    pause(0.5);
    l1=l1+1;l2=l2+1;l3=l3-2;l4=l4-2;

end;
   % rectangle('Position',[40 47 20 22],'EdgeColor','y'); %a1
end;

if(strcmp(I,'b1.jpg')||strcmp(I,'b.jpg'))
    for aa=1:10
        subplot(2,2,2); imshow(ad,[]);title('Locating Bounding box');
    rectangle('Position',[q1 q2 q3 q4],'EdgeColor','y'); %a1
    pause(0.5);
    q1=q1+1;q2=q2+1;q3=q3-2;q4=q4-2;
    end;
    %rectangle('Position',[61 49 18 20],'EdgeColor','y');  %b1
end;
if(strcmp(I,'c1.jpg')||strcmp(I,'c.jpg'))

     for aa=1:10
         subplot(2,2,2); imshow(ad,[]);title('Locating Bounding box');
    rectangle('Position',[z1 z2 z3 z4],'EdgeColor','y'); %a1
    pause(0.5);
    z1=z1+1;z2=z2+1;z3=z3-2;z4=z4-2;
    end;

    %rectangle('Position',[35 26 34 40],'EdgeColor','y');  %c1
end;



%bounding box end


subplot(2,2,3); title('Segmentation');

seg = svm(ad, m, 50); %-- Run segmentation

subplot(2,2,4); imshow(seg); title('Segmented Tumor');
%imwrite(seg,'test.jpg');
```

STEP 2: NOISE REMOVAL, IMAGE ENHANCEMENT, 2-D CONVOLUTION

```
function diff_im = anisodiff(im, num_iter, delta_t, kappa, option)
fprintf('Removing noise\n');


fprintf('Filtering Completed !!');

% Convert input image to double.
im = double(im);

% PDE (partial differential equation) initial condition.
diff_im = im;

% Center pixel distances.
dx = 1;
dy = 1;
dd = sqrt(2);

% 2D convolution masks - finite differences.
hN = [0 1 0; 0 -1 0; 0 0 0];
hS = [0 0 0; 0 -1 0; 0 1 0];
hE = [0 0 0; 0 -1 1; 0 0 0];
hW = [0 0 0; 1 -1 0; 0 0 0];
hNE = [0 0 1; 0 -1 0; 0 0 0];
hSE = [0 0 0; 0 -1 0; 0 0 1];
```

```
hSW = [0 0 0; 0 -1 0; 1 0 0];
hNW = [1 0 0; 0 -1 0; 0 0 0];

% Anisotropic diffusion.
for t = 1:num_iter

        % Finite differences. [imfilter(.,.,'conv') can be replaced by
conv2(.,.,'same')]
        nablaN = imfilter(diff_im,hN,'conv');
        nablaS = imfilter(diff_im,hS,'conv');
        nablaW = imfilter(diff_im,hW,'conv');
        nablaE = imfilter(diff_im,hE,'conv');
        nablaNE = imfilter(diff_im,hNE,'conv');
        nablaSE = imfilter(diff_im,hSE,'conv');
        nablaSW = imfilter(diff_im,hSW,'conv');
        nablaNW = imfilter(diff_im,hNW,'conv');

        % Diffusion function.
        if option == 1
            cN = exp(-(nablaN/kappa).^2);
            cS = exp(-(nablaS/kappa).^2);
            cW = exp(-(nablaW/kappa).^2);
            cE = exp(-(nablaE/kappa).^2);
            cNE = exp(-(nablaNE/kappa).^2);
            cSE = exp(-(nablaSE/kappa).^2);
            cSW = exp(-(nablaSW/kappa).^2);
            cNW = exp(-(nablaNW/kappa).^2);
        elseif option == 2
            cN = 1./(1 + (nablaN/kappa).^2);
            cS = 1./(1 + (nablaS/kappa).^2);
            cW = 1./(1 + (nablaW/kappa).^2);
            cE = 1./(1 + (nablaE/kappa).^2);
            cNE = 1./(1 + (nablaNE/kappa).^2);
            cSE = 1./(1 + (nablaSE/kappa).^2);
            cSW = 1./(1 + (nablaSW/kappa).^2);
            cNW = 1./(1 + (nablaNW/kappa).^2);
        end

        % Discrete PDE solution.
        diff_im = diff_im + ...
                  delta_t*(...
                  (1/(dy^2))*cN.*nablaN + (1/(dy^2))*cS.*nablaS + ...
                  (1/(dx^2))*cW.*nablaW + (1/(dx^2))*cE.*nablaE + ...
                  (1/(dd^2))*cNE.*nablaNE + (1/(dd^2))*cSE.*nablaSE + ...
                  (1/(dd^2))*cSW.*nablaSW + (1/(dd^2))*cNW.*nablaNW );



end
```