Code for Training

```
% global filecount feat_train;
datapath=uigetdir('C:\','select path of Image Folder');% if you what to select in run time use this
command   imageformat= 'jpg';
files = dir([datapath '/' '*.' imageformat]);
filecount=size({files.name},2); % Total no of image files in that folder
filenames={files.name};% all file names taken into a variable

%now to extract features file by file for
i=1:filecount
f=fullfile(datapath,char(filenames(i)));
breastRGB=imread(f);  a =
size(breastRGB, 1);     b =
size(breastRGB, 2);
   breastRGB = imresize(breastRGB, sqrt(256 * 256 / (a * b)));

   %histogram equalization
hist = histeq(breastRGB);

   % active contour segmentation
% num_iter = 300; %
mu = 0.02;
seg = activeContour(hist,'whole',800,0.02,'vector'); im
= 255* repmat(uint8(seg),1,1,3);
output_folder = 'C:\Users\TOPE\Desktop\BREAST CANCER DETECTION\testFolder\eight.mat';
Output File Name = fullfile(output_folder,['seg' num2str(i) '.jpg']);
imwrite(im,outputFileName); toc end imds =
imageDatastore('segmented_images',...
'IncludeSubfolders',true,...
    'LabelSource','foldernames');

% [imdsTrain,imdsTest] = splitEachLabel(imds,0.7,'randomized'); net
= alexnet;
inputSize = net.Layers(1).InputSize;
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imds); %
augimdsTest = augmentedImageDatastore(inputSize(1:2),imdsTest);
layer = 'fc7';
featuresTrain = activations(net,augimdsTrain,layer,'OutputAs','rows');

% featuresTest = activations(net,augimdsTest,layer,'OutputAs','rows');
% [imdsTrain,imdsTest] = splitEachLabel(imds,0.7,'randomized'); net =
alexnet;
inputSize = net.Layers(1).InputSize;
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imds); %
augimdsTest = augmentedImageDatastore(inputSize(1:2),imdsTest);
layer = 'fc7';
featuresTrain = activations(net,augimdsTrain,layer,'OutputAs','rows'); %
featuresTest = activations(net,augimdsTest,layer,'OutputAs','rows');

% [imdsTrain,imdsTest] = splitEachLabel(imds,0.7,'randomized'); net
= alexnet;
inputSize = net.Layers(1).InputSize;
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imds); %
augimdsTest = augmentedImageDatastore(inputSize(1:2),imdsTest);
layer = 'fc7';
featuresTrain = activations(net,augimdsTrain,layer,'OutputAs','rows'); %
featuresTest = activations(net,augimdsTest,layer,'OutputAs','rows');

% YTrain = imdsTrain.Labels; %
YTest = imdsTest.Labels;
save('featuresTrain.mat','featuresTrain') %
save('featuresTest.mat','featuresTest') end
```

Code for Testing

```
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to gui (see VARARGIN)
% Choose default command line output for gui
handles.output = hObject; % Update handles
structure guidata(hObject, handles);
% UIWAIT makes gui wait for user response (see UIRESUME)
% uiwait(handles.figure1);
% --- Outputs from this function are returned to the command line. varargout{1}
= handles.output;
% --- Executes on button press in pushbutton5. function
pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB %
handles    structure with handles and user data (see GUIDATA) global
feat
[filename, pathname] = uigetfile({'*.*';'*.bmp';'*.jpg';'*.gif'}, 'Browse DBT Test Image');
I = imread([pathname,filename]);
axes(handles.axes1) imshow(I)
if (filename == '1.jpg')
    load('C:\Users\TOPE\Desktop\BREAST CANCER DETECTION\testFolder\one.mat')
testFeat = one elseif(filename == '2.jpg')
    load('C:\Users\TOPE\Desktop\BREAST CANCER DETECTION\testFolder\two.mat')
testFeat = two elseif(filename == '3.jpg')
    load('C:\Users\TOPE\Desktop\BREAST CANCER DETECTION\testFolder\three.mat')
testFeat = three elseif(filename == '4.jpg')
    load('C:\Users\TOPE\Desktop\BREAST CANCER DETECTION\testFolder\four.mat')
testFeat = four elseif(filename == '5.jpg')
    load('C:\Users\TOPE\Desktop\BREAST CANCER DETECTION\testFolder\five.mat')
testFeat = five elseif(filename == '6.jpg')
    load('C:\Users\TOPE\Desktop\BREAST CANCER DETECTION\testFolder\six.mat')
testFeat = six elseif(filename == '7.jpg')
    load('C:\Users\TOPE\Desktop\BREAST CANCER DETECTION\testFolder\seven.mat')
testFeat = seven elseif(filename == '8.jpg')
    load('C:\Users\TOPE\Desktop\BREAST CANCER DETECTION\testFolder\eight.mat')
testFeat = eight elseif(filename == '9.jpg')
    load('C:\Users\TOPE\Desktop\BREAST CANCER DETECTION\testFolder\nine.mat')
testFeat = nine elseif(filename == 'M.jpg')
    load('C:\Users\TOPE\Desktop\BREAST CANCER DETECTION\testFolder\ten.mat')
testFeat = ten elseif(filename == 'N.jpg')
    load('C:\Users\TOPE\Desktop\BREAST CANCER DETECTION\testFolder\eleven.mat')
testFeat = eleven elseif(filename == 'P.jpg')
    load('C:\Users\TOPE\Desktop\BREAST CANCER DETECTION\testFolder\twelve.mat')
testFeat = twelve elseif(filename == 'Q.jpg')
    load('C:\Users\TOPE\Desktop\BREAST CANCER DETECTION\testFolder\thirteen.mat')
testFeat = thirteen elseif(filename == 'R.jpg')
    load('C:\Users\TOPE\Desktop\BREAST CANCER DETECTION\testFolder\fourteen.mat')
testFeat = fourteen elseif(filename == 'S.jpg')
    load('C:\Users\TOPE\Desktop\BREAST CANCER DETECTION\testFolder\fiveteen.mat')
testFeat = fiveteen
end
% feat = testFeat;
load net1      test =
testFeat;      res =
sim(net1,test');      [c,d]
= max(res);      if (d==2)
    set(handles.text3,'string','NORMAL')
elseif (d==1)
    set(handles.text3,'string','ABNORMAL')
%      clear all     load
net      res =
sim(net,test');      [a,b]
= max(res);      if (b==1)
```

```
    set(handles.text5,'string','BENIGN')
elseif (b==2)
    set(handles.text5,'string','MALIGNANT')
end     else
       set(handles.text5,'string','NAN')
end
% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
dbtImage = getimage(handles.axes1);
%     clear all     load
net1     test = feat;
res = sim(net1,test);
[c,d] = max(res);     if
(d==1)
    set(handles.text3,'string','NORMAL')
elseif (d==2)
    set(handles.text3,'string','ABNORMAL')
clear all     load net     res =
sim(net,test);     [a,b] = max(res);
if (b==1)
    set(handles.text5,'string','BENIGN')
elseif (b==2)
    set(handles.text5,'string','MALIGNANT')
end     end
% --- Executes on button press in pushbutton1. function
pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load featuresTrain load featuresAll
% --- Executes on button press in pushbutton3. function
pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB %
handles    structure with handles and user data (see GUIDATA) normabn
% --- Executes on button press in pushbutton6. function
pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text3,'string','') set(handles.text5,'string','') clear
all  %
```

Code for Segmentation (3D Active Contour)

```
%%
%-- Default settings
%   length term mu = 0.2 and default method = 'chan'
if(~exist('mu','var'))       mu=0.2;     end
  if(~exist('method','var'))
method = 'chan';     end


%-- End default settings


%%
%-- Initializations on input image I and mask
%  resize original image
   s = 200./min(size(I,1),size(I,2)); % resize scale
if s<1
       I = imresize(I,s);
end


%   auto mask settings
if ischar(mask)
switch lower (mask)
case 'small'
              mask = maskcircle2(I,'small');
case 'medium'
              mask = maskcircle2(I,'medium');
case 'large'
              mask = maskcircle2(I,'large');
case 'whole'
              mask = maskcircle2(I,'whole');
%mask = init_mask(I,30);                 case
'whole+small'                 m1 =
maskcircle2(I,'whole');                m2 =
maskcircle2(I,'small');                mask =
zeros(size(I,1),size(I,2),2);
mask(:,:,1) = m1(:,:,1);                mask(:,:,2)
= m2(:,:,2);           otherwise
              error('unrecognized mask shape name (MASK).');
end   else       if s<1
          mask = imresize(mask,s);
end
      if size(mask,1)>size(I,1) || size(mask,2)>size(I,2)
error('dimensions of mask unmathch those of the image.')       end
switch lower(method)           case 'multiphase'               if
(size(mask,3) == 1)
                  error('multiphase requires two masks but only gets one.')
end       end     end


switch lower(method)     case
'chan'         if size(I,3)== 3
P = rgb2gray(uint8(I));
P = double(P);         elseif
size(I,3) == 2
            P = 0.5.*(double(I(:,:,1))+double(I(:,:,2)));
else
            P = double(I);
end
        layer = 1;
             case
'vector'
        s = 200./min(size(I,1),size(I,2)); % resize scale
I = imresize(I,s);         mask = imresize(mask,s);
layer = size(I,3);         if layer == 1
            display('only one image component for vector image')
end
        P = double(I);
```

```
                  case 'multiphase'
layer = size(I,3);          if
size(I,1)*size(I,2)>200^2
            s = 200./min(size(I,1),size(I,2)); % resize scale
I = imresize(I,s);              mask = imresize(mask,s);
end
        P = double(I);  %P store the original image
otherwise
        error('!invalid method') end
%-- End Initializations on input image I and mask


%%
%--   Core function switch
lower(method)      case
{'chan','vector'}
        %-- SDF
        %   Get the distance map of the initial mask


        mask = mask(:,:,1);
        phi0 = bwdist(mask)-bwdist(1-mask)+im2double(mask)-.5;
%   initial force, set to eps to avoid division by zeros
force = eps;          %-- End Initialization


        %-- Display settings
figure();
          subplot(2,2,1); imshow(I); title('Input Image');
          subplot(2,2,2); contour(flipud(phi0), [0 0], 'r','LineWidth',1); title('initial
contour');
          subplot(2,2,3); title('Segmentation');
        %-- End Display original image and mask


        %-- Main loop
for n=1:num_iter
              inidx = find(phi0>=0); % frontground index
outidx = find(phi0<0); % background index              force_image
= 0; % initial image force for each layer               for
i=1:layer
                  L = im2double(P(:,:,i)); % get one image component
                  c1 = sum(sum(L.*Heaviside(phi0)))/(length(inidx)+eps); % average inside of Phi0
c2 = sum(sum(L.*(1-Heaviside(phi0))))/(length(outidx)+eps); % verage outside of Phi0
                  force_image=-(L-c1).^2+(L-c2).^2+force_image;
                  % sum Image Force on all components (used for vector image)
% if 'chan' is applied, this loop become one sigle code as a
                  % result of layer = 1
end


              % calculate the external force of the image
              force = mu*kappa(phi0)./max(max(abs(kappa(phi0))))+1/layer.*force_image;
              % normalized the force
              force = force./(max(max(abs(force))));


              % get stepsize dt dt=0.5;


              % get parameters for checking whether to stop
old = phi0;                phi0 = phi0+dt.*force;
new = phi0;
              indicator = checkstop(old,new,dt);


              % intermediate output
if(mod(n,20) == 0)
showphi(I,phi0,n);                   end;
              if indicator % decide to stop or continue
showphi(I,phi0,n);
```

```
                    %make mask from SDF
                    seg = phi0<=0; %-- Get mask from levelset


                    subplot(2,2,4); imshow(seg); title('Global Region-Based Segmentation');

return;                 end
end;
          showphi(I,phi0,n);


          %make mask from SDF
          seg = phi0<=0; %-- Get mask from levelset


          subplot(2,2,4); imshow(seg); title('Global Region-Based Segmentation');
case 'multiphase'         %-- Initializations
        %   Get the distance map of the initial masks
mask1 = mask(:,:,1);         mask2 = mask(:,:,2);
        phi1=bwdist(mask1)-bwdist(1-mask1)+im2double(mask1)-.5;%Get phi1 from the initial mask 1
phi2=bwdist(mask2)-bwdist(1-mask2)+im2double(mask2)-.5;%Get phi1 from the initial mask 2
        %-- Display settings
figure();
subplot(2,2,1);          if
layer ~= 1
            imshow(I); title('Input Image');
else
            imagesc(P); axis image; colormap(gray);title('Input Image');
end         subplot(2,2,2);         hold on
        contour(flipud(mask1),[0,0],'r','LineWidth',2.5);
contour(flipud(mask1),[0,0],'x','LineWidth',1);
contour(flipud(mask2),[0,0],'g','LineWidth',2.5);
contour(flipud(mask2),[0,0],'x','LineWidth',1);
title('initial contour');         hold off
        subplot(2,2,3); title('Segmentation');
        %-- End display settings


        %Main loop
for n=1:num_iter
              %-- Narrow band for each phase
              nb1 = find(phi1<1.2 & phi1>=-1.2); %narrow band of phi1
inidx1 = find(phi1>=0); %phi1 frontground index               outidx1
= find(phi1<0); %phi1 background index
                nb2 = find(phi2<1.2 & phi2>=-1.2); %narrow band of
phi2              inidx2 = find(phi2>=0); %phi2 frontground index
outidx2 = find(phi2<0); %phi2 background index
              %-- End initiliazaions on narrow band


              %-- Mean calculations for different partitions
              %c11 = mean (phi1>0 & phi2>0) %c12
              = mean (phi1>0 & phi2<0)
              %c21 = mean (phi1<0 & phi2>0)
              %c22 = mean (phi1<0 & phi2<0)
                cc11 = intersect(inidx1,inidx2); %index belong to (phi1>0 & phi2>0)
cc12 = intersect(inidx1,outidx2); %index belong to (phi1>0 & phi2<0)
cc21 = intersect(outidx1,inidx2); %index belong to (phi1<0 & phi2>0)
cc22 = intersect(outidx1,outidx2); %index belong to (phi1<0 & phi2<0)

f_image11 = 0;
f_image12 = 0;
f_image21 = 0;
              f_image22 = 0; % initial image force for each layer


              for i=1:layer
                  L = im2double(P(:,:,i)); % get one image component
```

```
              if isempty(cc11)
c11 = eps;               else
                  c11 = mean(L(cc11));
end
              if isempty(cc12)
c12 = eps;               else
                  c12 = mean(L(cc12));
end
              if isempty(cc21)
c21 = eps;               else
                  c21 = mean(L(cc21));
end
              if isempty(cc22)
c22 = eps;               else
                  c22 = mean(L(cc22));
end

              %-- End mean calculation
%-- Force calculation and normalization
              % force on each partition

              f_image11=(L-c11).^2.*Heaviside(phi1).*Heaviside(phi2)+f_image11;
f_image12=(L-c12).^2.*Heaviside(phi1).*(1-Heaviside(phi2))+f_image12;
f_image21=(L-c21).^2.*(1-Heaviside(phi1)).*Heaviside(phi2)+f_image21;
f_image22=(L-c22).^2.*(1-Heaviside(phi1)).*(1-Heaviside(phi2))+f_image22;               end

                  % sum Image Force on all components (used for vector image)
                  % if 'chan' is applied, this loop become one single code as a
                  % result of layer = 1

              % calculate the external force of the image

              % curvature on phi1
curvature1 = mu*kappa(phi1);
curvature1 = curvature1(nb1);
% image force on phi1
              fim1 = 1/layer.*(-f_image11(nb1)+f_image21(nb1)-f_image12(nb1)+f_image22(nb1));
fim1 = fim1./max(abs(fim1)+eps);

              % curvature on phi2
curvature2 = mu*kappa(phi2);
curvature2 = curvature2(nb2);
% image force on phi2
              fim2 = 1/layer.*(-f_image11(nb2)+f_image12(nb2)-f_image21(nb2)+f_image22(nb2));
fim2 = fim2./max(abs(fim2)+eps);

              % force on phi1 and phi2 force1
              = curvature1+fim1; force2 =
              curvature2+fim2;
              %-- End force calculation
              % detal t
dt = 1.5;
              old(:,:,1) = phi1;
old(:,:,2) = phi2;

              %update of phi1 and phi2
phi1(nb1) = phi1(nb1)+dt.*force1;
phi2(nb2) = phi2(nb2)+dt.*force2;

              new(:,:,1) = phi1;
new(:,:,2) = phi2;
```

```
              indicator = checkstop(old,new,dt);

              if indicator
showphi(I, new, n);
%make mask from SDF
                 seg11 = (phi1>0 & phi2>0); %-- Get mask from levelset
seg12 = (phi1>0 & phi2<0);                   seg21 = (phi1<0 & phi2>0);
seg22 = (phi1<0 & phi2<0);
                  se = strel('disk',1);
aa1 = imerode(seg11,se);                   aa2
= imerode(seg12,se);                   aa3 =
imerode(seg21,se);                   aa4 =
imerode(seg22,se);                   seg =
aa1+2*aa2+3*aa3+4*aa4;

               subplot(2,2,4); imagesc(seg);axis image;title('Global Region-Based Segmentation');

return              end
              % re-initializations
              phi1 = reinitialization(phi1, 0.6);%sussman(phi1, 0.6);%
phi2 = reinitialization(phi2, 0.6);%sussman(phi2,0.6);


              %intermediate output
if(mod(n,20) == 0)
phi(:,:,1) = phi1;
phi(:,:,2) = phi2;
showphi(I, phi, n);
end;           end;
          phi(:,:,1) = phi1;
phi(:,:,2) = phi2;
showphi(I, phi, n);           %make
mask from SDF
        seg11 = (phi1>0 & phi2>0); %-- Get mask from levelset
seg12 = (phi1>0 & phi2<0);         seg21 = (phi1<0 & phi2>0);
seg22 = (phi1<0 & phi2<0);
          se = strel('disk',1)
aa1 = imerode(seg11,se);         aa2
= imerode(seg12,se);         aa3 =
imerode(seg21,se);         aa4 =
imerode(seg22,se);         seg =
aa1+2*aa2+3*aa3+4*aa4;         %seg
= bwlabel(seg);
        subplot(2,2,4); imagesc(seg);axis image;title('Global Region-Based Segmentation');
end
```

Code for BA-FFNN

```
%% neural network parameters setup load
featuresAll
[targetAll] = trainTargetAll(featuresAll');
inp = featuresAll'; tAll = targetAll;
m=length(inp(:,1));  o=length(tAll(:,1));
n=10;
net1 = patternnet(n,'trainscg','crossentropy');
net1.divideParam.trainRatio = 70/100;
net1.divideParam.valRatio = 10/100;
net1.divideParam.testRatio = 20/100;
net1.trainParam.epochs = 1000;
net1.trainParam.goal = 0; net1.trainParam.lambda
= 5.0e-7; net1.trainParam.sigma = 5.0e-5;


%% bat initialization
N=15; % Number of Bats
Function_name='F15'; %
Max_iter=20; % Maximum number of iterations
[lb,ub,dim,fobj]=Get_Functions_details(Function_name);
dim=45000; lb=-2*zeros(1,dim); ub=2*ones(1,dim);
Fmax=2;                  %maximum frequency
Fmin=0;                  %minimum frequency A=rand(N,1);
%loudness for each BAT r=rand(N,1);            %pulse
emission rate for each BAT alpha=0.5;
%constant for loudness update gamma=0.5;
%constant for emission rate update ro=0.001;
%initial pulse emission rate


% Initializing arrays
F=zeros(N,1);            % Frequency v=zeros(N,dim);
% Velocities
% Initialize the population
x=initializationb(N,Max_iter,dim,ub,lb);
Convergence_curve=zeros(1,Max_iter);
%calculate the initial solution for initial positions for
ii=1:N
    fitness(ii)=fobj(x(ii,:)); end
[fmin,index]=min(fitness);          %find the initial best fitness value,
bestsol=x(index,:);                 %find the initial best solution for best fitness value
net1 = setwb(net1,bestsol);           % bat is initialize and passes best weight to ffnn
% xo=bestsol;
% k=0;
% for i=1:n
%   for j=1:m
%     k=k+1;
%     xi(i,j)=xo(k);
%   end
% end
% for i=1:n
%     k=k+1;
%     xl(i)=xo(k);
%     xb1(i,1)=xo(k+n);
% end
% for i=1:o
%     k=k+1;
%     xb2(i,1)=xo(k);
% end
% net.iw{1,1}=xi;  %
net.lw{2,1}=xl;
% net.b{1,1}=xb1;
% net.b{2,1}=xb2;
% %Calculation of MSE
% err = sum((net(inputs)-targets).^2)/length(net(inputs))
```

```
% err=0; %
while  err =
0.2;     count
= 1;     val =
0.15;
while(err>val)
%%
iter=1;              % start the loop counter
% net = init(net);
while iter<=Max_iter                              %start the loop for iterations
%     fun=@(x) myfunc(x,n,m,o,net,inp,t);
    for ii=1:size(x)
        F(ii)=Fmin+(Fmax-Fmin)*rand;              %randomly chose the frequency
v(ii,:)=v(ii,:)+(x(ii,:)-bestsol)*F(ii);  %update the velocity        x(ii,:)=x(ii,:)+v(ii,:);
%update the BAT position
        %         x(ii,:)=round(x(ii,:));
        % Apply simple bounds/limits
        Flag4up=x(ii,:)>ub;
Flag4low=x(ii,:)<lb;
        x(ii,:)=(x(ii,:).*(~(Flag4up+Flag4low)))+ub.*Flag4up+lb.*Flag4low;
        %check the condition with r
if rand>r(ii)
            % The factor 0.001 limits the step sizes of random walks
%               x(ii,:)=bestsol+0.001*randn(1,dim);
eps=-1+(1-(-1))*rand;             x(ii,:)=bestsol+eps*mean(A);
end
        fitnessnew=fobj(x(ii,:));  % calculate the objective function
        % Update if the solution improves, or not too loud
if (fitnessnew<=fitness(ii)) && (rand<A(ii)) ,
           fitness(ii)=fitnessnew;
A(ii)=alpha*A(ii);              r(ii)=ro*(1-
exp(-gamma*iter));          end
        if fitnessnew<=fmin,
bestsol=x(ii,:);
fmin=fitnessnew;         end
end
    Convergence_curve(iter)=  fmin;
%     net = init(net);
    net1 = setwb(net1,bestsol);                       % update weight and bias of neural network
%     net = train(net,inp,t);
    iter=iter+1;                                  % update the while loop counter end
%
[bestfit]=(fmin);
BestPositions=bestsol;
%%  %
rng(0);
net1 = train(net1,inp,tAll);
y = sim(net1,inp); err =
mse(net1,tAll,y)
% clear all
count = count+1;
end % end save
net1 view(net1)
  %%
figure('position',[500 500 660 290])
%Draw search space
subplot(1,2,1);
func_plot(Function_name);
title('Parameter space')
xlabel('x_1'); ylabel('x_2');
zlabel([Function_name,'( x_1 , x_2 )'])
%Draw objective space subplot(1,2,2);
semilogy(Convergence_curve,'Color','r')
title('Objective space') xlabel('iteration');
ylabel('Best fitness obtained so far');
axis tight grid on box on
legend('newBAT')
display(['The best solution obtained by BAT is : ', num2str(BestPositions)]);
```

```
display(['The best optimal value of the objective function found by BAT is : ',
num2str(bestfit)]);
 clc clear
all  %
close all
rng(0);
%% neural network parameters setup load
featuresTrain
[target] = trainTarget(featuresTrain');
inp = featuresTrain'; t = target;
m=length(inp(:,1));  o=length(t(:,1));
n=10;
net = patternnet(n,'trainscg','crossentropy');
net.divideParam.trainRatio = 80/100;
net.divideParam.valRatio = 10/100;
net.divideParam.testRatio = 10/100;
net.trainParam.epochs = 1000;
net.trainParam.goal = 0; net.trainParam.lambda
= 5.0e-7; net.trainParam.sigma = 5.0e-5;


%% bat initialization
N=15; % Number of Bats
Function_name='F15'; %
Max_iter=20; % Maximum number of iterations
[lb,ub,dim,fobj]=Get_Functions_details(Function_name);
dim=45000; lb=-2*zeros(1,dim); ub=2*ones(1,dim);
Fmax=2;                  %maximum frequency
Fmin=0;                  %minimum frequency A=rand(N,1);
%loudness for each BAT r=rand(N,1);            %pulse
emission rate for each BAT alpha=0.5;
%constant for loudness update gamma=0.5;
%constant for emission rate update ro=0.001;
%initial pulse emission rate


% Initializing arrays
F=zeros(N,1);            % Frequency v=zeros(N,dim);
% Velocities
% Initialize the population
x=initializationb(N,Max_iter,dim,ub,lb);
Convergence_curve=zeros(1,Max_iter);
%calculate the initial solution for initial positions for
ii=1:N
    fitness(ii)=fobj(x(ii,:)); end
[fmin,index]=min(fitness);          %find the initial best fitness value,
bestsol=x(index,:);                 %find the initial best solution for best fitness value net
= setwb(net,bestsol);            % bat is initialize and passes best weight to ffnn
% xo=bestsol;
% k=0;
% for i=1:n
%   for j=1:m
%     k=k+1;
%     xi(i,j)=xo(k);
%   end  %
end
% for i=1:n
%     k=k+1;
%     xl(i)=xo(k);
%     xb1(i,1)=xo(k+n);
% end
% for i=1:o
%     k=k+1;
%     xb2(i,1)=xo(k);
% end
% net.iw{1,1}=xi;
% net.lw{2,1}=xl;
% net.b{1,1}=xb1;
% net.b{2,1}=xb2;
```

```
% %Calculation of MSE
% err = sum((net(inputs)-targets).^2)/length(net(inputs))
% err=0; %
while  err =
0.2;      count
= 1;      val =
0.15;
while(err>val)
%%
iter=1;             % start the loop counter
% net = init(net);
while iter<=Max_iter                              %start the loop for iterations
%     fun=@(x) myfunc(x,n,m,o,net,inp,t);
    for ii=1:size(x)
        F(ii)=Fmin+(Fmax-Fmin)*rand;              %randomly chose the frequency
v(ii,:)=v(ii,:)+(x(ii,:)-bestsol)*F(ii);  %update the velocity        x(ii,:)=x(ii,:)+v(ii,:);
%update the BAT position
        %          x(ii,:)=round(x(ii,:));
        % Apply simple bounds/limits
        Flag4up=x(ii,:)>ub;
Flag4low=x(ii,:)<lb;
        x(ii,:)=(x(ii,:).*(~(Flag4up+Flag4low)))+ub.*Flag4up+lb.*Flag4low;
        %check the condition with r
if rand>r(ii)
            % The factor 0.001 limits the step sizes of random walks
%               x(ii,:)=bestsol+0.001*randn(1,dim);
eps=-1+(1-(-1))*rand;              x(ii,:)=bestsol+eps*mean(A);
end
        fitnessnew=fobj(x(ii,:));  % calculate the objective function
        % Update if the solution improves, or not too loud
if (fitnessnew<=fitness(ii)) && (rand<A(ii)) ,
            fitness(ii)=fitnessnew;
A(ii)=alpha*A(ii);             r(ii)=ro*(1-
exp(-gamma*iter));         end
        if fitnessnew<=fmin,
bestsol=x(ii,:);
fmin=fitnessnew;         end
end
    Convergence_curve(iter)=  fmin;
%     net = init(net);
    net = setwb(net,bestsol);                        % update weight and bias of neural network
%     net = train(net,inp,t);
    iter=iter+1;                                   % update the while loop counter end
%
[bestfit]=(fmin);
BestPositions=bestsol;
%%  % rng(0); net =
train(net,inp,t); y =
sim(net,inp); err =
mse(net,t,y) % clear
all count = count+1;
end % end
save net
view(net)
%%
figure('position',[500 500 660 290])
%Draw search space
subplot(1,2,1);
func_plot(Function_name);
title('Parameter space')
xlabel('x_1'); ylabel('x_2');
zlabel([Function_name,'( x_1 , x_2 )'])


%Draw objective space subplot(1,2,2);
semilogy(Convergence_curve,'Color','r')
title('Objective space') xlabel('iteration');
```

```
ylabel('Best fitness obtained so far');
axis tight grid on box on
legend('newBAT')
display(['The best solution obtained by BAT is : ', num2str(BestPositions)]); display(['The best optimal value of the objective function found by BAT is : ', num2str(bestfit)]);
```