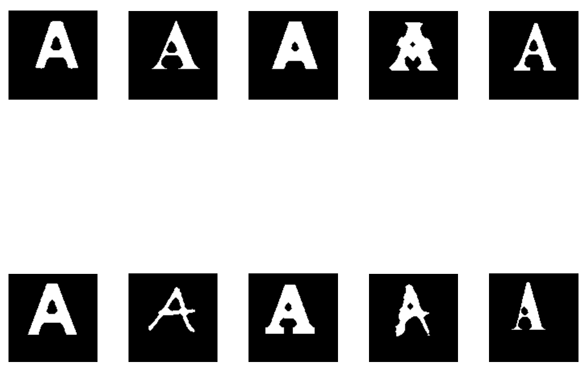
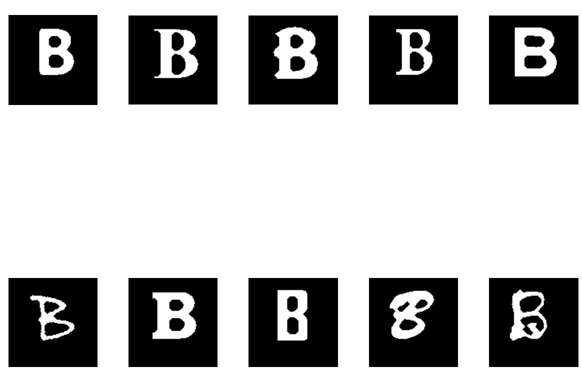
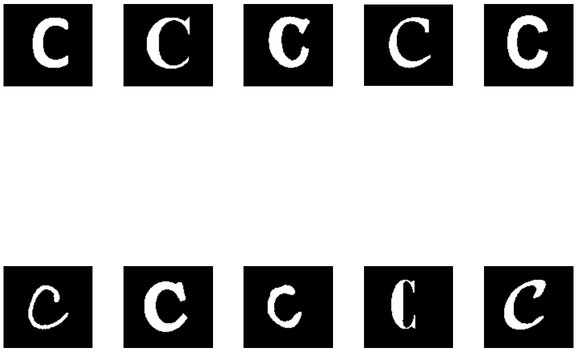
EEE 178 Homework 9

Curtis Muntz

```matlab
% Pre processing
clear all; close all; clc;
%code for my custom functions can be found on
%https://github.com/curtismuntz/machine_vision/
    tree/master/commonFunctions
addpath ../commonFunctions
I1=getIMG('mvHW9A.jpg'); % <- learning set
I1=im2bw(I1);
I1=imclose(I1,strel('diamond',3));
range=[91,37,1317,320];
I1=imcrop(I1, range);
I2=getIMG('mvHW9B.jpg'); % <- testing image
I2=imcomplement(im2bw(I2)); % objects need to
    be white
I2=imclose(I2, strel('diamond',3));
I2=imopen(I2, strel('diamond',2));


cleanI=I2;
rmpath ../commonFunctions

imshow(I1), title('Training Set');
figure
imshow(cleanI), title('Test Set');

% grab data points, parse letters into cells
close all
figure('name','A objects');

x=0; %start of object x=0 (because of cropping)
    x+=134
y=0; %start of object y=0 (because of cropping)
    y+=110
%there are 10 objects
objA={zeros(10)};
Astats={zeros(10)};

for i=1:10
    subplot(2,5,i)
    objRange=[x,y,110,110];
    objA{i}=imcrop(I1, objRange);
    Astats{i}=regionprops(objA{i},'all');
    imshow(objA{i})
    x=x+134;
end

figure('name','B objects');
x=0;
y=y+110;
%there are 10 objects
objB={zeros(10)};
Bstats={zeros(10)};

for i=1:10
    subplot(2,5,i)
    objRange=[x,y,110,110];
    objB{i}=imcrop(I1, objRange);
    Bstats{i}=regionprops(objB{i},'all');
    imshow(objB{i})
    x=x+134;
end

figure('name','C objects');
x=0;
y=y+110;
%there are 10 objects
objC={zeros(10)};
Cstats={zeros(10)};

for i=1:10
    subplot(2,5,i)
    objRange=[x,y,110,110];
    objC{i}=imcrop(I1, objRange);
    Cstats{i}=regionprops(objC{i},'all');
    imshow(objC{i})
    x=x+134;
end
```





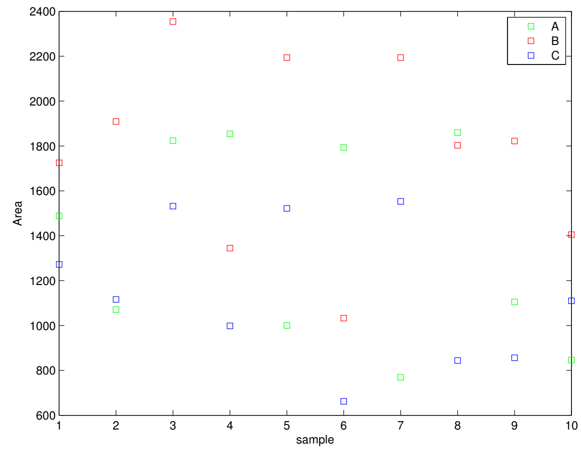


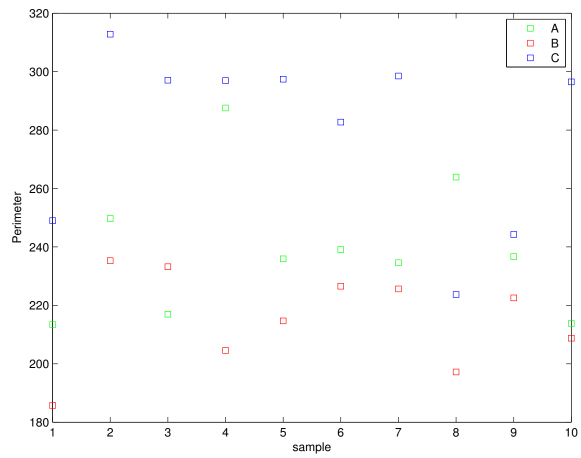
Plotting Regionprop Features

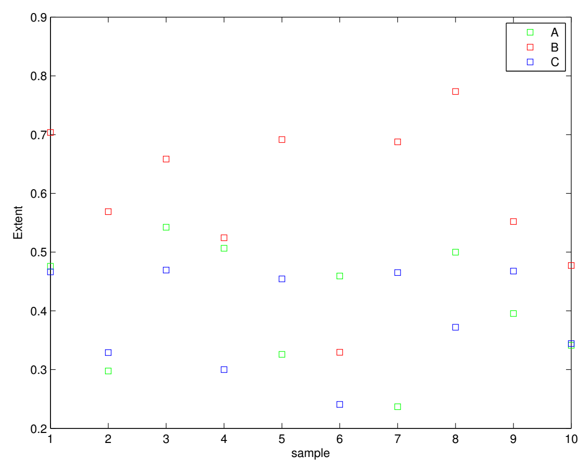
What follows is a bunch of graphs of the distributions of the various features of the letters.

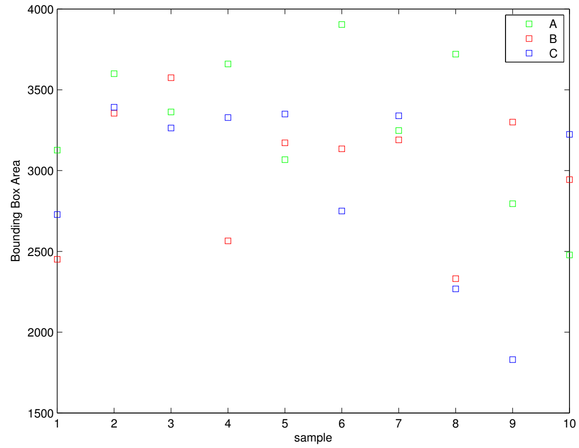
```
close all

x=1:1:10;
figure('name','Area')
A=zeros(1,10);
B=zeros(1,10);
C=zeros(1,10);
for i=1:10
    A(i)=Astats{i}.Area;
    B(i)=Bstats{i}.Area;
    C(i)=Cstats{i}.Area;
end
plot(x,A,'s','color','green'), hold on
plot(x,B, 's','color','red'), hold on
plot(x,C,'s','color','blue')
xlabel('sample')
ylabel('Area')
legend('A','B','C'), hold off


figure('name','Perimiter')
for i=1:10
    A(i)=Astats{i}.Perimeter;
    B(i)=Bstats{i}.Perimeter;
    C(i)=Cstats{i}.Perimeter;
end
plot(x,A,'s','color','green'), hold on
plot(x,B, 's','color','red'), hold on
plot(x,C,'s','color','blue'), hold on
xlabel('sample')
ylabel('Perimeter')
legend('A','B','C'), hold off

figure('name','Extent')
for i=1:10
    A(i)=Astats{i}.Extent;
    B(i)=Bstats{i}.Extent;
    C(i)=Cstats{i}.Extent;
end
plot(x, A, 's','color','green'), hold on
plot(x, B, 's','color','red'), hold on
plot(x, C, 's','color','blue'), hold on
xlabel('sample')
ylabel('Extent')
legend('A','B','C'), hold off


figure('name','BoundingBox Area')
%where bounding box areas are the heights*
    widths (BB(3)*BB(4))
for i=1:10
    A(i)=((Astats{i}.BoundingBox(3))*(Astats{i
    }.BoundingBox(4)));
    B(i)=((Bstats{i}.BoundingBox(3))*(Bstats{i
    }.BoundingBox(4)));
    C(i)=((Cstats{i}.BoundingBox(3))*(Cstats{i
    }.BoundingBox(4)));
end
plot(x, A, 's','color','green'), hold on
plot(x, B, 's','color','red'), hold on
plot(x, C, 's','color','blue'), hold on
xlabel('sample')
ylabel('Bounding Box Area')
legend('A','B','C'), hold off

figure('name','EquivDiameter')
for i=1:10
    A(i)=Astats{i}.EquivDiameter;
    B(i)=Bstats{i}.EquivDiameter;
    C(i)=Cstats{i}.EquivDiameter;
end
plot(x, A, 's','color','green'), hold on
plot(x, B, 's','color','red'), hold on
plot(x, C, 's','color','blue'), hold on
xlabel('sample')
ylabel('EquivDiameter')
legend('A','B','C'), hold off



figure('name','EulerNumber')
for i=1:10
    A(i)=Astats{i}.EulerNumber;
    B(i)=Bstats{i}.EulerNumber;
    C(i)=Cstats{i}.EulerNumber;
end
plot(x, A, 's','color','green'), hold on
plot(x, B, 's','color','red'), hold on
plot(x, C, 's','color','blue'), hold on
xlabel('sample')
ylabel('EulerNumber')
legend('A','B','C'), hold off



figure('name','FilledArea')
for i=1:10
    A(i)=Astats{i}.FilledArea;
    B(i)=Bstats{i}.FilledArea;
    C(i)=Cstats{i}.FilledArea;
end
plot(x, A, 's','color','green'), hold on
plot(x, B, 's','color','red'), hold on
plot(x, C, 's','color','blue'), hold on
xlabel('sample')
ylabel('FilledArea')
legend('A','B','C'), hold off



figure('name','ConvexArea')
for i=1:10
    A(i)=Astats{i}.ConvexArea;
    B(i)=Bstats{i}.ConvexArea;
    C(i)=Cstats{i}.ConvexArea;
end
plot(x, A, 's','color','green'), hold on
plot(x, B, 's','color','red'), hold on
plot(x, C, 's','color','blue'), hold on
xlabel('sample')
```

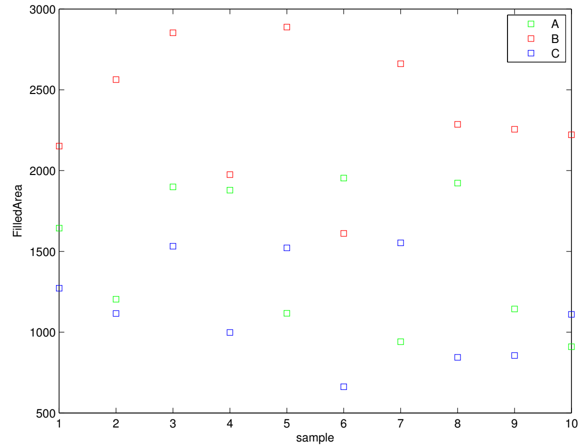
```
ylabel('ConvexArea')
legend('A','B','C'), hold off


figure('name','MinorAxisLength')
for i=1:10
    A(i)=Astats{i}.MinorAxisLength;
    B(i)=Bstats{i}.MinorAxisLength;
    C(i)=Cstats{i}.MinorAxisLength;
end
plot(x, A, 's','color','green'), hold on
plot(x, B, 's','color','red'), hold on
plot(x, C, 's','color','blue'), hold on
xlabel('sample')
ylabel('MinorAxisLength')
legend('A','B','C'), hold off

figure('name','MajorAxisLength')
for i=1:10
    A(i)=Astats{i}.MajorAxisLength;
    B(i)=Bstats{i}.MajorAxisLength;
    C(i)=Cstats{i}.MajorAxisLength;
end
plot(x, A, 's','color','green'), hold on
plot(x, B, 's','color','red'), hold on
plot(x, C, 's','color','blue'), hold on
xlabel('sample')
ylabel('MajorAxisLength')
legend('A','B','C'), hold off


figure('name','Solidity')
for i=1:10
    A(i)=Astats{i}.Solidity;
    B(i)=Bstats{i}.Solidity;
    C(i)=Cstats{i}.Solidity;
end
plot(x, A, 's','color','green'), hold on
plot(x, B, 's','color','red'), hold on
plot(x, C, 's','color','blue'), hold on
xlabel('sample')
ylabel('Solidity')
legend('A','B','C'), hold off



figure('name','Centroid Magnitude')
for i=1:10
    A(i)=sqrt(Astats{i}.Centroid(1)^2 + Astats{
    i}.Centroid(2)^2);
    B(i)=sqrt(Bstats{i}.Centroid(1)^2 + Bstats{
    i}.Centroid(2)^2);
    C(i)=sqrt(Cstats{i}.Centroid(1)^2 + Cstats{
    i}.Centroid(2)^2);
end
plot(x, A, 's','color','green'), hold on
plot(x, B, 's','color','red'), hold on
plot(x, C, 's','color','blue'), hold on
title('Centroid Magnitude')
xlabel('sample')
ylabel('Centroid Magnitude')
legend('A','B','C'), hold off

figure('name','Average Extrema')
for i=1:10
    A(i)=mean(mean(Astats{i}.Extrema));
    B(i)=mean(mean(Bstats{i}.Extrema));
    C(i)=mean(mean(Cstats{i}.Extrema));
end
plot(x, A, 's','color','green'), hold on
plot(x, B, 's','color','red'), hold on
plot(x, C, 's','color','blue'), hold on
xlabel('sample')
ylabel('Average Extrema')
legend('A','B','C'), hold off
```

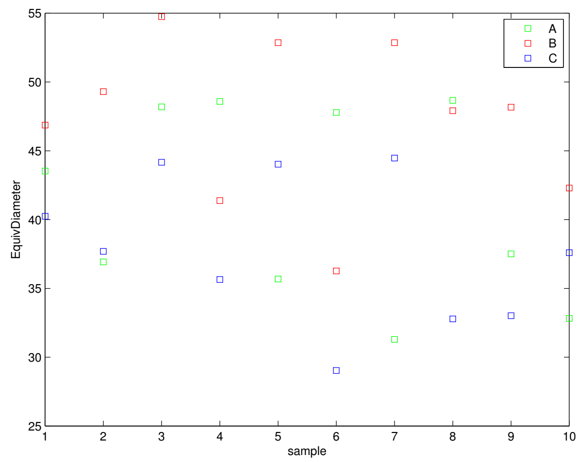


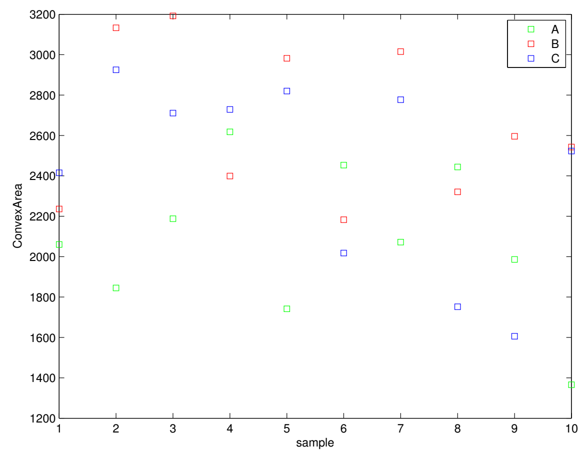


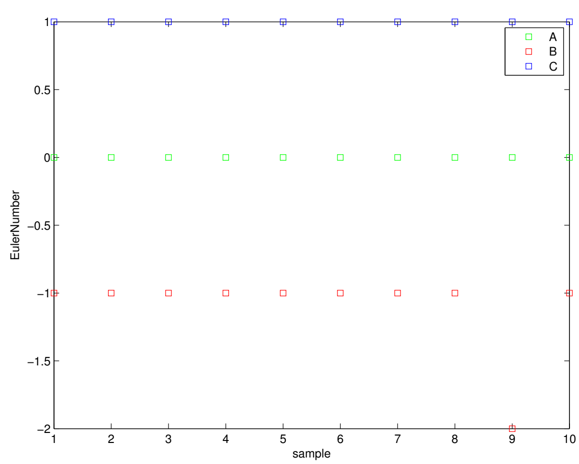


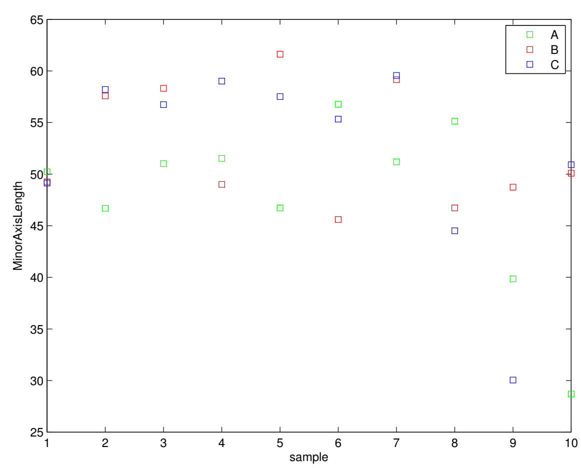

Bounding Box Area
sample
A
B
C

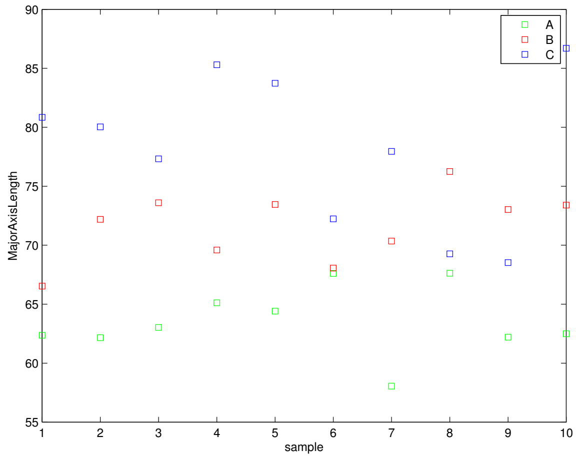

FilledArea
sample
A
B
C

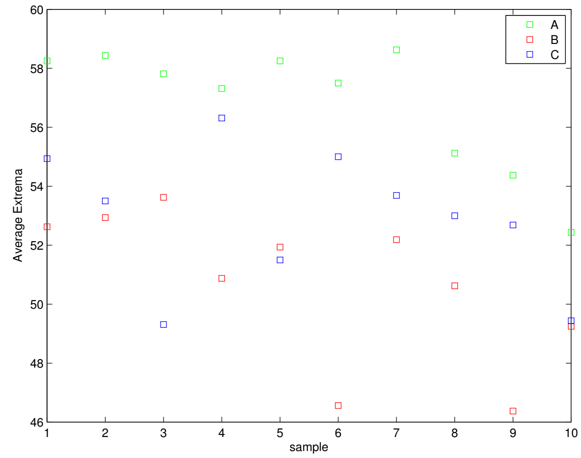

EquivDiameter
sample
A
B
C

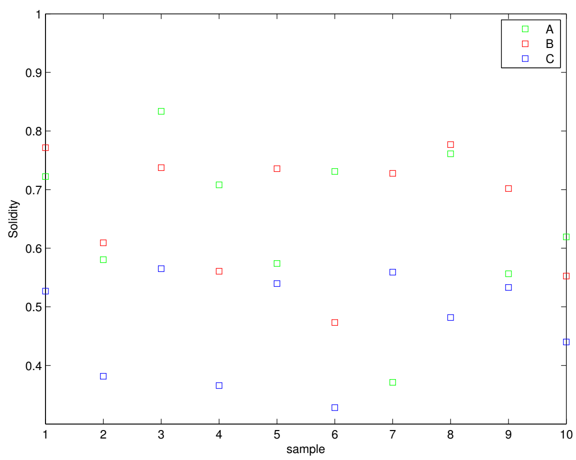

ConvexArea
sample
A
B
C

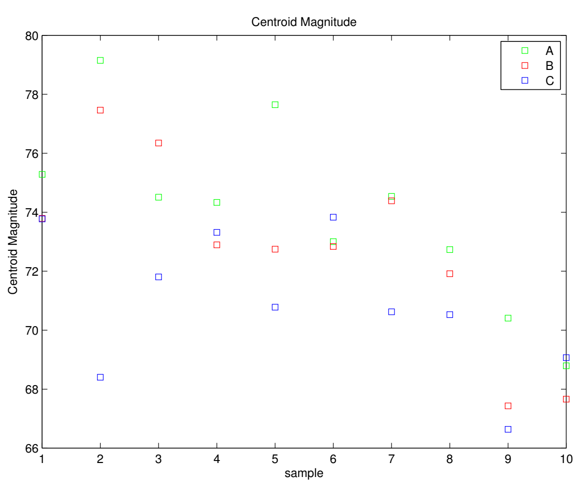

EulerNumber
sample
A
B
C


MinorAxisLength
sample
A
B
C







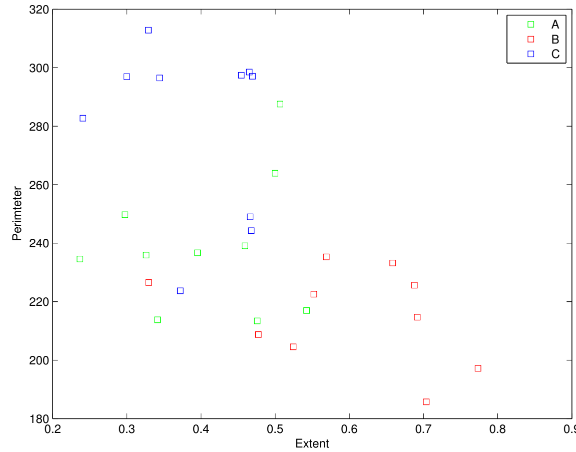


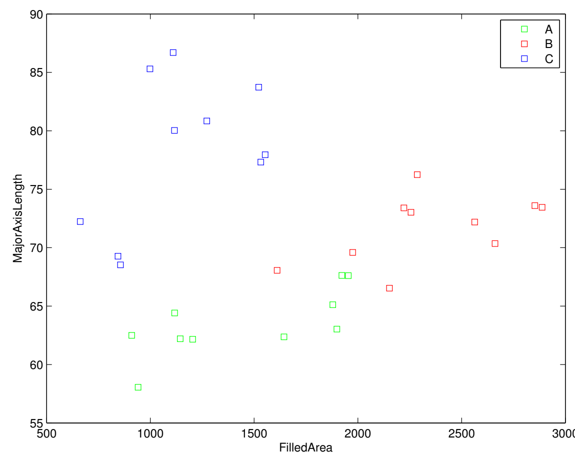
Finding Good Seperators

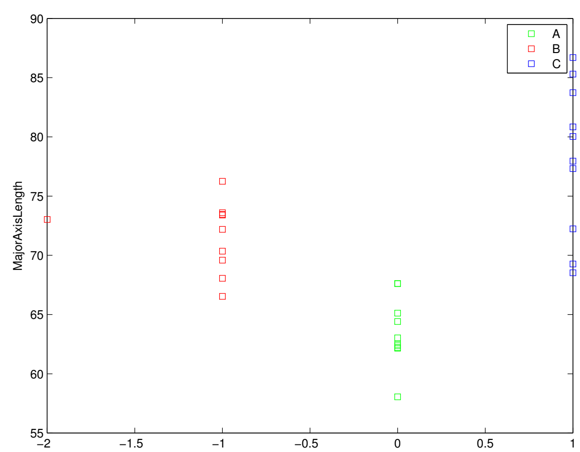
The graphs are analyzed to find good seperators.

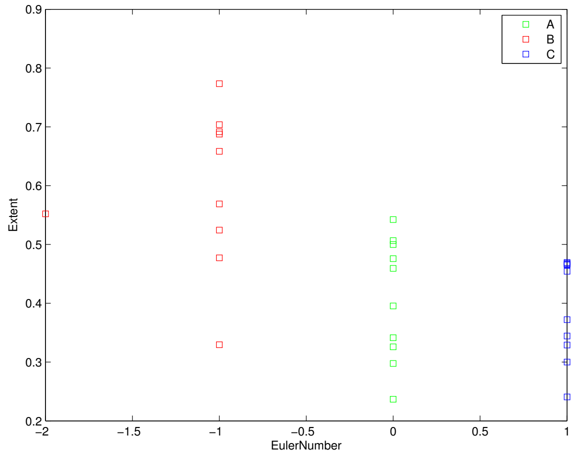
```
A1=zeros(1,10);
A2=A1; A3=A1; B1=A1; B2=A1; B3=A1; C1=A1; C2=A1
    ; C3=A1;

figure('name','Perim vs Extent')
for i=1:10
    A1(i)=Astats{i}.Extent;
    B1(i)=Bstats{i}.Extent;
    C1(i)=Cstats{i}.Extent;
    A2(i)=Astats{i}.Perimeter;
    B2(i)=Bstats{i}.Perimeter;
    C2(i)=Cstats{i}.Perimeter;
end
plot(A1, A2, 's','color','green'), hold on
plot(B1, B2, 's','color','red'), hold on
plot(C1, C2, 's','color','blue'), hold on
xlabel('Extent')
ylabel('Perimteter')
legend('A','B','C'), hold off


figure('name','MajorAxisLength vs FilledArea')
for i=1:10
    A1(i)=Astats{i}.FilledArea;
    B1(i)=Bstats{i}.FilledArea;
    C1(i)=Cstats{i}.FilledArea;
    A2(i)=Astats{i}.MajorAxisLength;
    B2(i)=Bstats{i}.MajorAxisLength;
    C2(i)=Cstats{i}.MajorAxisLength;
end
plot(A1, A2, 's','color','green'), hold on
plot(B1, B2, 's','color','red'), hold on
plot(C1, C2, 's','color','blue'), hold on
xlabel('FilledArea')
ylabel('MajorAxisLength')
legend('A','B','C'), hold off


figure('name','MajorAxisLength vs EulerNumber')
for i=1:10
    A1(i)=Astats{i}.EulerNumber;
    B1(i)=Bstats{i}.EulerNumber;
    C1(i)=Cstats{i}.EulerNumber;
    A2(i)=Astats{i}.MajorAxisLength;
    B2(i)=Bstats{i}.MajorAxisLength;
    C2(i)=Cstats{i}.MajorAxisLength;
```

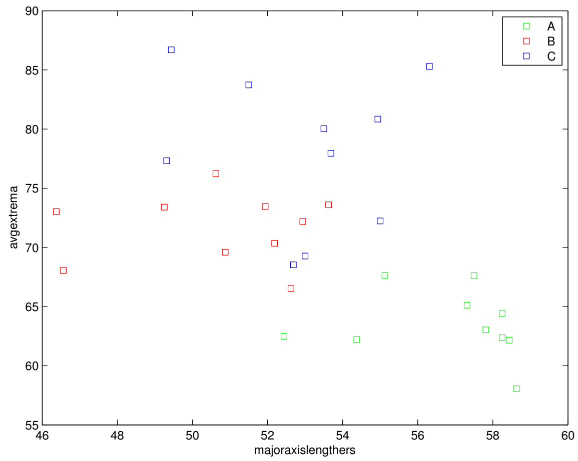
```
end
plot(A1, A2, 's','color','green'), hold on
plot(B1, B2, 's','color','red'), hold on
plot(C1, C2, 's','color','blue'), hold on
xlabel('EulerNumber')
ylabel('MajorAxisLength')
legend('A','B','C'), hold off


figure('name','Extent vs EulerNumber')
for i=1:10
    A1(i)=Astats{i}.EulerNumber;
    B1(i)=Bstats{i}.EulerNumber;
    C1(i)=Cstats{i}.EulerNumber;
    A2(i)=Astats{i}.Extent;
    B2(i)=Bstats{i}.Extent;
    C2(i)=Cstats{i}.Extent;
end
plot(A1, A2, 's','color','green'), hold on
plot(B1, B2, 's','color','red'), hold on
plot(C1, C2, 's','color','blue'), hold on
xlabel('EulerNumber')
ylabel('Extent')
legend('A','B','C'), hold off

figure('name','avg extrema vs major axis length')
for i=1:10
    A1(i)=mean(mean(Astats{i}.Extrema));
    B1(i)=mean(mean(Bstats{i}.Extrema));
    C1(i)=mean(mean(Cstats{i}.Extrema));
    A2(i)=Astats{i}.MajorAxisLength;
    B2(i)=Bstats{i}.MajorAxisLength;
    C2(i)=Cstats{i}.MajorAxisLength;
end
plot(A1, A2, 's','color','green'), hold on
plot(B1, B2, 's','color','red'), hold on
plot(C1, C2, 's','color','blue'), hold on
xlabel('majoraxislengthers')
ylabel('avgextrema')
legend('A','B','C'), hold off


figure('name','MajorAxisLength vs FilledArea vs Extent')
for i=1:10
    A1(i)=Astats{i}.FilledArea;
    B1(i)=Bstats{i}.FilledArea;
    C1(i)=Cstats{i}.FilledArea;
    A2(i)=Astats{i}.MajorAxisLength;
    B2(i)=Bstats{i}.MajorAxisLength;
    C2(i)=Cstats{i}.MajorAxisLength;
    A3(i)=Astats{i}.Extent;
    B3(i)=Bstats{i}.Extent;
    C3(i)=Cstats{i}.Extent;
end

plot3(A1, A2, A3, 's','color','green'), hold on
plot3(B1, B2, B3, 's','color','red'), hold on
plot3(C1, C2, C3, 's','color','blue'), hold on
xlabel('FilledArea')
ylabel('MajorAxisLength')
zlabel('Extent')
legend('A','B','C'), hold off
```

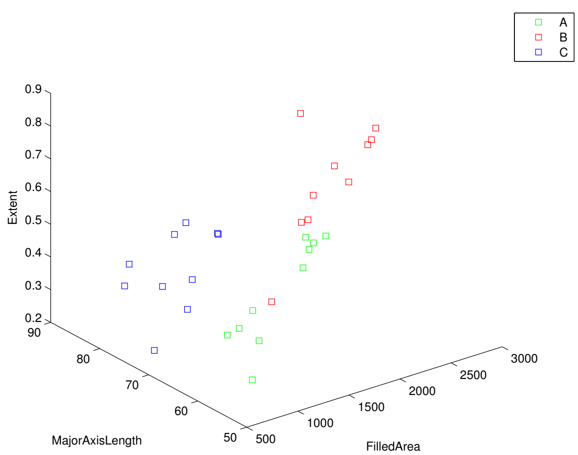










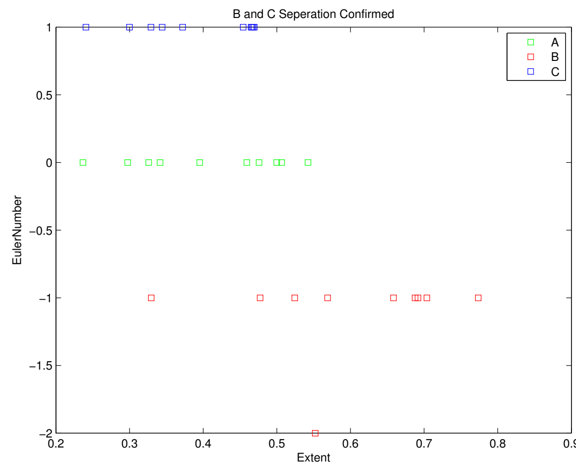


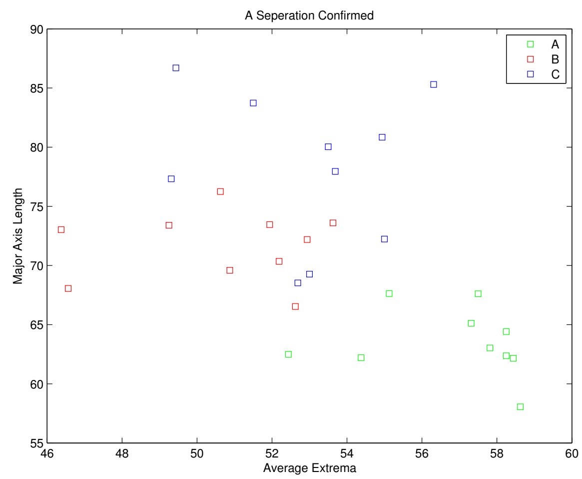
Creating Feature Sets

Now that good seperators can be seen, we can combine these into training sets. Extent and Euler number do a great job seperating the B's and C's, but it fails to address the A's. Even though the A's are still in a very different grouping from the others, there is no way to draw a straight line to seperate them because their distribution is in between the other letters. For this reason, another training set was formed for the A letters, using the average extrema and the major axis length.

```
figure('name', 'combining the feature sets!!!');
trainingSet= zeros(30,2);
resultSetA = zeros(30,1);
resultSetB = zeros(30,1);
resultSetC = zeros(30,1);
for i=1:10
    m=1;
    trainingSet(i,m)   = Astats{i}.Extent;
    trainingSet(i,m+1) = Astats{i}.EulerNumber;
    resultSetA(i)      = 1;
end
for i=11:20
    m=1;
    trainingSet(i,m)   = Bstats{i-10}.Extent;
    trainingSet(i,m+1) = Bstats{i-10}.EulerNumber;
    resultSetB(i)      = 1; % these are A's so we should switch their desired output to 1!!!
end
for i=21:30
    m=1;
    trainingSet(i,m)   = Cstats{i-20}.Extent;
    trainingSet(i,m+1) = Cstats{i-20}.EulerNumber;
    resultSetC(i)      = 1; % these are C's so we should switch their desired output to 1!!!
end

trainingSetA= zeros(30,2);
for i=1:10
    m=1;
    trainingSetA(i,m)   = mean(mean(Astats{i}.Extrema));
    trainingSetA(i,m+1) = Astats{i}.MajorAxisLength;
    resultSetA(i)      = 1;
end
for i=11:20
    m=1;
    trainingSetA(i,m)   = mean(mean(Bstats{i-10}.Extrema));
    trainingSetA(i,m+1) = Bstats{i-10}.MajorAxisLength;
    resultSetB(i)      = 1; % these are A's so we should switch their desired output to 1!!!
end
for i=21:30
    m=1;
    trainingSetA(i,m)   = mean(mean(Cstats{i-20}.Extrema));
    trainingSetA(i,m+1) = Cstats{i-20}.MajorAxisLength;
    resultSetC(i)      = 1; % these are C's so we should switch their desired output to 1!!!
end


avgAExt   = mean(trainingSet(1:10,1));
avgBExt   = mean(trainingSet(11:20,1));
avgCExt   = mean(trainingSet(21:30,1));

avgAEN    = mean(trainingSet(1:10,2));
avgBEN    = mean(trainingSet(11:20,2));
avgCEN    = mean(trainingSet(21:30,2));

avgAXtrm = mean(trainingSetA(1:10,1));
```

```
avgnotAXtrm = mean(trainingSetA(11:30,1));

avgAMAL   = mean(trainingSetA(1:10,2));
avgnotAMAL   = mean(trainingSetA(11:30,2));


figure('name','B and C seperation confirmed')
plot(trainingSet(1:10,1), trainingSet(1:10,2),
    's','color','green'), hold on
plot(trainingSet(11:20,1), trainingSet(11:20,2)
    , 's','color','red'), hold on
plot(trainingSet(21:30,1), trainingSet(21:30,2)
    , 's','color','blue'), hold on
title('B and C Seperation Confirmed')
xlabel('Extent')
ylabel('EulerNumber')
legend('A','B','C'), hold off


figure('name','A Seperation')
plot(trainingSetA(1:10,1), trainingSetA(1:10,2)
    , 's','color','green'), hold on
plot(trainingSetA(11:20,1), trainingSetA
    (11:20,2), 's','color','red'), hold on
plot(trainingSetA(21:30,1), trainingSetA
    (21:30,2), 's','color','blue'), hold on
title('A Seperation Confirmed');
xlabel('Average Extrema')
ylabel('Major Axis Length')
legend('A','B','C'), hold off
```





Mean Distance Classifier

Using the equations defined in class, the following equation was derived in order to find the Mean Distance to Centroid. These equations will be plotted simultaneously with the Neural Network solution in the next part.

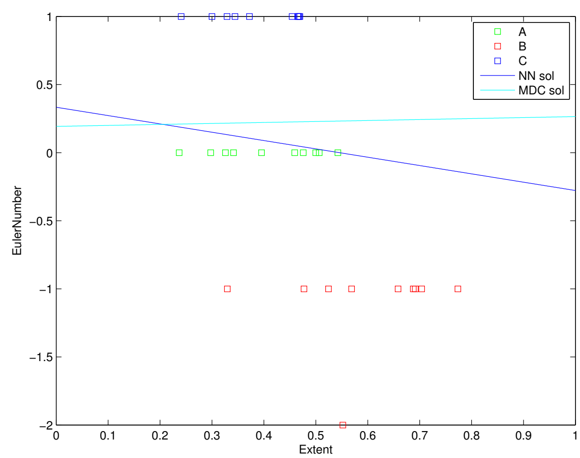
```
%x2=(-2*x1*x1b+(x1b^2)+(x2b^2)+2*x1*a-(a^2)-(b
    ^2))/(2*x1b-2*b)
%MDC C:
x1b=avgCExt; x2b=avgCEN; a=(avgAExt+avgBExt)/2;
     b=(avgAEN + avgBEN)/2; x1=0:0.001:1;
MDC_C=(-2*x1*x1b+(x1b^2)+(x2b^2)+2*x1*a-(a^2)-(
    b^2))/(2*x2b-2*b);
clear x1b x2b a b

%MDC B
x1b=avgBExt; x2b=avgBEN; a=(avgAExt+avgCExt)/2;
     b=(avgAEN + avgCEN)/2; x1=0:0.001:1;
MDC_B=(-2*x1*x1b+(x1b^2)+(x2b^2)+2*x1*a-(a^2)-(
    b^2))/(2*x2b-2*b);
clear x1b x2b a b

%MDC A
x1b=avgAXtrm; x2b=avgAMAL; a=avgnotAXtrm; b=
    avgnotAMAL; x1=45:1:65;
MDC_A=(-2*x1*x1b+(x1b^2)+(x2b^2)+2*x1*a-(a^2)-(
    b^2))/(2*x2b-2*b);
```

Perceptron for C

Because the perceptron learning algorithm is a binary classifier, we have to stage the detections in order to solve for three classes. In this section, the perceptron algorithm is computed for the letter B vs $notB$, and draws the resultant discrimination line for the classification. Because we will need to run the perceptron three times, I created a function [weightVector, bias] = customPerceptron(trainingSet, resultSet) to reduce the amount of code needed. For clarity, the full code is listed in this section.

```
close all; clc;
%build training set

[m, n] = size(trainingSet);
weightVectorC = ones(1,n);
for i=1:n
    weightVectorC(i) = weightVectorC(i)./(5); % initialize to small numbers. 5 guaranteed to be random, it was chosen by rolling a dice.
end

%weightVector = zeros(1,n);
threshold = 0;%threshold to decide if the output is good or bad. usually this is 0
error_count = 1;
biasC = 0.1;
result = 1;
iterationNo = 1;
learningRate = 0.001;
% training phase
while (error_count > 0)
 error_count = 0;
 for i=1:m
        gx=dot(weightVectorC,trainingSet(i,:))+biasC;
        if (gx > threshold)
            result = 1;
        else
            result = 0;
        end

        error = resultSetC(i)-result;
        if (error ~= 0)
            error_count = error_count + 1;
            weightVectorC = weightVectorC + (learningRate*(error))*trainingSet(i,1:n);
            biasC = biasC + learningRate*error;
        end
    end

 if (iterationNo >= 1000)
        disp('Neuron input calculation couldn''t completed in timely fashion.');
        break
    end
 iterationNo = iterationNo +1;
end
disp(['answer converged in ' num2str(iterationNo) ' iterations']);

disp(['weights: ' num2str(weightVectorC)]);
disp(['bias: ' num2str(biasC)]);

t=0:0.001:1;
figure('name','Seperation of class C');
plot(trainingSet(1:10,1), trainingSet(1:10,2),'s','color','green'), hold on
plot(trainingSet(11:20,1), trainingSet(11:20,2),'s','color','red'), hold on
plot(trainingSet(21:30,1), trainingSet(21:30,2),'s','color','blue'), hold on
plot(t,(-weightVectorC(1)*t-biasC)/weightVectorC(2),'color','blue'), hold on
plot(t,MDC_C,'color','cyan'), hold on;
xlabel('Extent')
ylabel('EulerNumber')
legend('A','B','C','NN sol','MDC sol'), hold off
```

```
answer converged in 24 iterations
weights: 0.12655       0.207
bias: -0.069
```



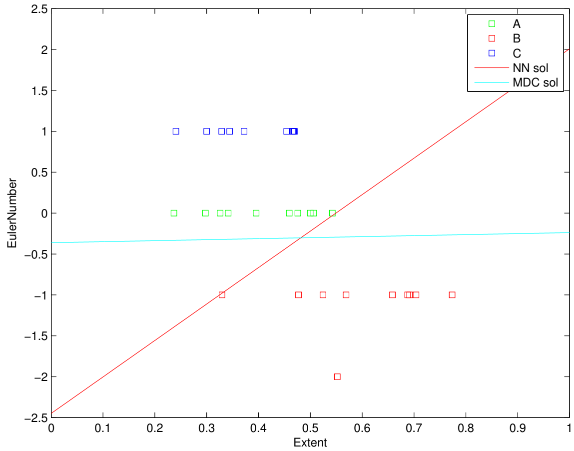
Perceptron for B

This section will solve the perceptron algorithm for the letter B vs $notB$ and draws the resultant discrimination line for the classificaition. the customPerceptron function is implemented using the code from the previous section.

```
addpath ../commonFunctions;
[weightVectorB, biasB] = customPerceptron(trainingSet, resultSetB);
rmpath ../commonFunctions;

disp(['weights: ' num2str(weightVectorB)]);
disp(['bias: ' num2str(biasB)]);

t=0:.001:1;
figure('name','Seperation of class B');
plot(trainingSet(1:10,1), trainingSet(1:10,2),'s','color','green'), hold on
plot(trainingSet(11:20,1), trainingSet(11:20,2),'s','color','red'), hold on
plot(trainingSet(21:30,1), trainingSet(21:30,2),'s','color','blue'), hold on
plot(t,(-weightVectorB(1)*t-biasB)/weightVectorB(2),'color','red'); %NN solution
plot(t,MDC_B,'color','cyan'), hold on; % MDC sol
xlabel('Extent')
ylabel('EulerNumber')
```

```
legend('A','B','C','NN sol','MDC sol'), hold
    off
```

```
answer converged in 47 iterations
weights: 0.12925      -0.029
bias: -0.071
```

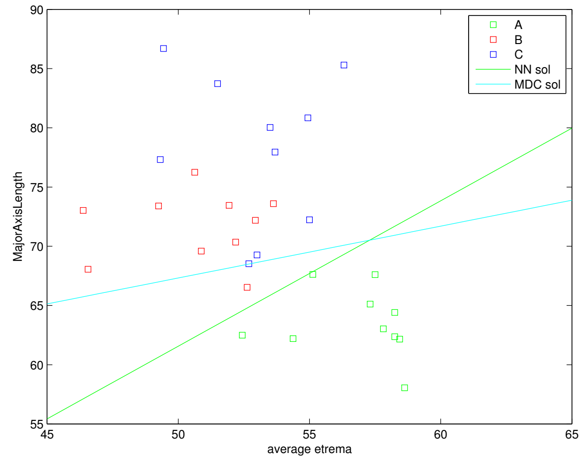


Perceptron for A

This section will solve the perceptron algorithm for the letter A vs $notA$ and draws the resultant discrimination line for the classificaition

```
addpath ../commonFunctions;
[weightVectorA, biasA] = customPerceptron(
    trainingSetA, resultSetA);
rmpath ../commonFunctions;

disp(['weights: ' num2str(weightVectorA)]);
disp(['bias: ' num2str(biasA)]);

t=45:1:65;
figure('name','Seperation of class A');
plot(trainingSetA(1:10,1), trainingSetA(1:10,2)
    , 's','color','green'), hold on
plot(trainingSetA(11:20,1), trainingSetA
    (11:20,2), 's','color','red'), hold on
plot(trainingSetA(21:30,1), trainingSetA
    (21:30,2), 's','color','blue'), hold on
plot(t,(-weightVectorA(1)*t-biasA)/
    weightVectorA(2),'color','green'), hold on
plot(t,MDC_A,'color','cyan'), hold on; % MDC
    sol
xlabel('average etrema')
ylabel('MajorAxisLength')
legend('A','B','C','NN sol','MDC sol'), hold
    off
```

```
answer converged in 109 iterations
weights: 0.55556    -0.45275
bias: 0.096
```



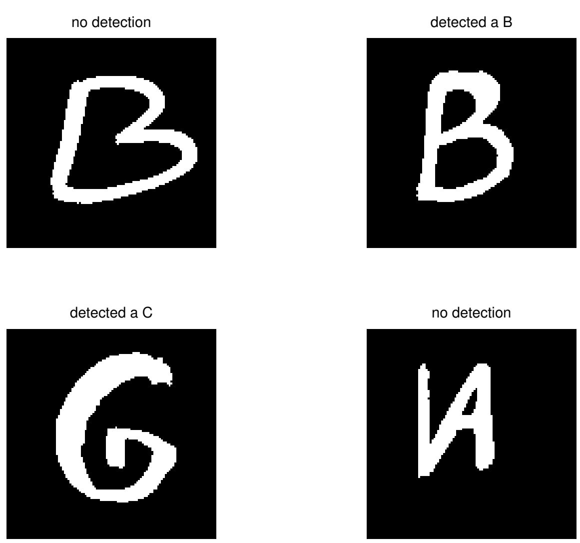
Comparing our unknown set vs the perceptrons

Through a careful manual cropping, four values of the test set were compared against the neural network solution for each letter.

```
clear testDataA testDataB testDataC

figure('name', 'Test Objects')
scalar = 4.1818;
test1=imcrop(I2,[(1472-ceil(25*scalar)),(628-
    ceil(15*scalar)), 600,600]);
test1=imresize(test1, [110,110]);
test2=imcrop(I2,[(875-ceil(25*scalar)), (645-
    ceil(15*scalar)), 600,600]);
test2=imresize(test2, [110,110]);
test3=imcrop(I2,[(858-ceil(25*scalar)),(1878-
    ceil(15*scalar)),600,600]);
test3=imresize(test3, [110,110]);
test4=imcrop(I2,[(792-ceil(25*scalar)),(161-
    ceil(15*scalar)),600,570]);
test4=imresize(test4, [110,110]);

subplot(221), imshow(test1);
stats=regionprops(test1,'all');
test1Stats=[stats.Extent,stats.EulerNumber];
test1StatsA=[mean(mean(stats.Extrema)),stats.
    MajorAxisLength];
if(dot(weightVectorA,test1StatsA)+biasA > 0),
    disp('detected an A');title('detected an A'
    );
elseif((dot(weightVectorB,test1Stats)+biasB) >
    0), disp('detected a B'); title('detected a
     B');
elseif((dot(weightVectorC,test1Stats)+biasC) >
    0), disp('detected a C'); title('detected a
     C');
else disp('-----'),title('no detection'); end


subplot(222), imshow(test2);
stats=regionprops(test2,'all');
test2Stats=[stats.Extent,stats.EulerNumber];
test2StatsA=[mean(mean(stats.Extrema)),stats.
    MajorAxisLength];
if(dot(weightVectorA,test2StatsA)+biasA > 0),
    disp('detected an A');title('detected an A'
    );
elseif(dot(weightVectorB,test2Stats)+biasB > 0)
    , disp('detected a B'); title('detected a B
    ');
```

```
elseif(dot(weightVectorC,test2Stats)+biasC > 0), disp('detected a C'); title('detected a C');
else disp('-----'),title('no detection'); end

subplot(223), imshow(test3);
stats=regionprops(test3,'all');
test3Stats=[stats.Extent,stats.EulerNumber];
test3StatsA=[mean(mean(stats.Extrema)),stats.MajorAxisLength];
if(dot(weightVectorA,test3StatsA)+biasA > 0), disp('detected an A');title('detected an A');
elseif(dot(weightVectorB,test3Stats)+biasB > 0), disp('detected a B'); title('detected a B');
elseif(dot(weightVectorC,test3Stats)+biasC > 0), disp('detected a C'); title('detected a C');
else disp('-----'), title('no detection'); end

subplot(224), imshow(test4);
stats=regionprops(test4,'all');
test4Stats=[stats.Extent,stats.EulerNumber];
test4StatsA=[mean(mean(stats.Extrema)),stats.MajorAxisLength];
if(dot(weightVectorA,test4StatsA)+biasA > 0), disp('detected an A');title('detected an A');
elseif(dot(weightVectorB,test4Stats)+biasB > 0), disp('detected a B'); title('detected a B');
elseif(dot(weightVectorC,test4Stats)+biasC > 0), disp('detected a C'); title('detected a C');
else disp('-----'), title('no detection'); end
```

```
-----
detected a B
detected a C
-----
```



Neural Network Results

The results from the testing were not very successful. By using only two features to create the weight vectors in the neurons, we are limiting the robustness of the solution. Should we want better results, perhaps implementing most of the regionprop data into our calculations will result in better classification.