

Pattern Recognition

EE5532 Module 5 Assignment

Matthew Spencer

03/28/2021

Contents

Introduction	2
Theory	2
Materials	2
Minimum Distance	3
Brute-Force	5
Results	6
Conclusions	7
References	8
Appendix	9

Introduction

This module explores pattern recognition within digital image processing. A classification method is used on a training data set of simplistic binary images containing five different shapes per image; a square, rectangle, circle, oval, and a random gaussian blob. Each shape is parsed and has a chosen feature set extracted from it to be used towards finding similar patterns within a testing data set of similar shapes randomly scaled and rotated.

The classification method chosen for this report's pattern recognition algorithm is the minimum-distance classification, where the feature set uses its mean feature vectors of each shape to find the minimum distance subtracted from the unknown shape's features, along with applying the euclidean norm to find the minimum distance. The classical minimum distance classification method is compared to a brute-force minimum distance classifier devised during experimentation.

The end of this report will then wrap up with an analysis regarding each method by observing the confusion matrices of successful found shapes from the testing data, and what could be done to improve each method's algorithms for future research.

Theory

By finding the Hu-moment features of each parsed shape from the training data set, the minimum-distance classifier will then have enough information to make a higher percentage of successful guesses regarding each unknown shape in the testing data set using the mean feature vector found after calculations. The brute-force minimum classification should be more accurate, as it finds the minimum distance by comparing it to each feature vector found in the training set, but as a result it should be fairly more computationally expensive.

Materials

- Windows 10 PC
 - 10400 i5 Intel Processor
 - 16 GB Ram
- MATLAB R2020a

Minimum Distance

The minimum distance classifier is a simple and common matching method [1] which finds the distance between an unknown pattern vector and a desired pattern's feature vector. It may not be the most complex or powerful choice when choosing a classification method, but considering that the patterns within the test and training images that are being parsed in this report are simple binary shapes, the method seemed sufficient to use for experimentation.

Each training and testing image contains one of each of the following shapes; a square, circle, oval, rectangle, and blob. A sample training image can be seen below in Fig. 1, along with the parsed cropped shapes in Fig. 2.

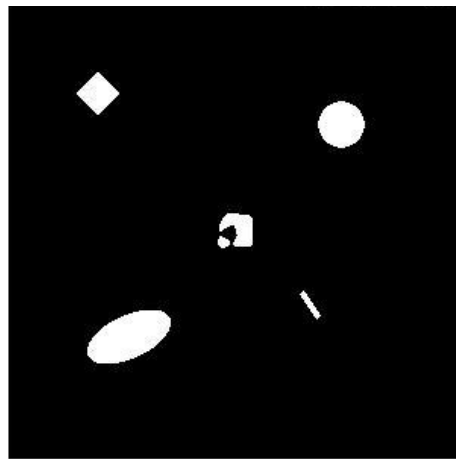


Fig. 1: Training Image



Fig. 2: Parsed Training Patterns from Fig. 1

Each shape from Fig. 2 then has sixteen Hu-moments features extracted from it using an installed MATLAB function `SI_Moments()` [5], which are invariant moments that are immune to orientation variances of shifting patterns within an image space [2]. These features m_j are summed totally over one-thousand training images, and then averaged into one mean feature vector for each shape pattern called a class [Eq. 1].

$$m_j = \frac{1}{n_j} \sum_{x \in C_j}^{N_j} x \quad j = 1, 2, 3, \dots, N_c$$

Eq. 1: Class Mean Vectors [1]

The distance D_j of the unknown pattern's feature x is then found for each class using the Euclidean norm, observed below in Eq. 2.

$$D_j(x) = ||x - m_j|| \quad j = 1, 2, 3, \dots, N_c$$

Eq. 2: Normalized Minimum Distance [1]

An algorithmic implementation of the main guts of Eq. 2 within MATLAB is shown in Fig. 3.

```
% Get current estimates for each cropped shape
for j = 1:5
    sqEst(:,1,j) = data(:,1,j) - sqData;
    circEst(:,1,j) = data(:,1,j) - circData;
    ovalEst(:,1,j) = data(:,1,j) - ovalData;
    rectEst(:,1,j) = data(:,1,j) - rectData;
    blobEst(:,1,j) = data(:,1,j) - blobData;
end

% Find normalized distances for each est
sqDist = zeros(1,5);
circDist = zeros(1,5);
ovalDist = zeros(1,5);
rectDist = zeros(1,5);
blobDist = zeros(1,5);
for j = 1:5
    sqDist(1,j) = norm(sqEst(:,1,j));
    circDist(1,j) = norm(circEst(:,1,j));
    ovalDist(1,j) = norm(ovalEst(:,1,j));
    rectDist(1,j) = norm(rectEst(:,1,j));
    blobDist(1,j) = norm(blobEst(:,1,j));
end

% Make sure we have the minimal estimates
for j = 1:5
    % Get current distances
    distances = [sqDist(1,j), circDist(1,j), ovalDist(1,j), ...
                rectDist(1,j), blobDist(1,j)];

    % Find the minimal distance
    min = 99999;
    est = 5;
    for p = 1:5
        if (distances(p) < min)
            min = distances(p);
            est = p;
        end
    end
end
```

Fig. 3: Minimum Distance MATLAB Eq. 2 Implementation

The class that has the least distance found between all the normalized feature sets is then assigned to the unknown pattern. Found patterns are displayed in a confusion matrix seen in Table 1 within the Results section later on in this report.

Brute-Force

This method is an alteration to the above minimum distance calculation using the normalized distance, as the algorithm keeps each feature vector in memory for comparison with the unknown feature vector.

Every feature vector from each class extracted from the training data is subtracted from the unknown feature vector, absolute valued, and then summed into a single numerical number. The number found that is closest to zero and its corresponding class is then selected to be assigned towards the unknown pattern.

Each class has its minimum distance found using the unknown feature vector x and the individual feature vectors m_i of the corresponding class j seen in Eq. 3 below.

$$D_j(x) = \min \left(\left| \sum_{i=1}^{N_i} x - m_i \right| \right) \quad j = 1, 2, 3, \dots, N_c$$

Eq. 3: Brute-Force Minimum Distance Method

The MATLAB implementation of Eq. 3 is shown below in Fig. 4, where each class and all of its features are subtracted from the unknown feature vector and the class with the smallest valued result is chosen.

```
% Get current estimates using moment data
for j = 1:5
    sqE = sum(sum(abs(sqMom-data(:,j))));
    circE = sum(sum(abs(circMom-data(:,j))));
    ovalE = sum(sum(abs(ovalMom-data(:,j))));
    rectE = sum(sum(abs(rectMom-data(:,j))));
    blobE = sum(sum(abs(blobMom-data(:,j))));

    % Check to see if it is the minimum
    if (sqE < sqEst(j))
        sqEst(j) = sqE;
    end
    if (circE < circEst(j))
        circEst(j) = circE;
    end
    if (ovalE < ovalEst(j))
        ovalEst(j) = ovalE;
    end
    if (rectE < rectEst(j))
        rectEst(j) = rectE;
    end
    if (blobE < blobEst(j))
        blobEst(j) = blobE;
    end
end
end
```

Fig. 4: Brute-Force Eq. 3 MATLAB Implementation

This method has a higher computational overhead, as each feature vector is used in comparison with the unknown pattern feature unlike the previous method, which only used the mean feature vector of each class. A confusion matrix was also constructed using found patterns seen in Table 2.

Results

Tables 1 and 2 below display the confusion matrices populated with the assigned classes of 100 parsed testing images similar to Fig. 1 after parsing 1000 images within the training data set. Each image contains one of each class shape, and is randomly orientated and scaled according to the code provided on the EE5532 class Canvas resource page [3]. Time taken to execute the script was measured using the MATLAB functions `tic` and `toc` [4].

Execution Time for Minimum Distance = 97.4574 seconds

Total % Correct Class Assignments = 69%

	Square	Circle	Oval	Rectangle	Blob	% Correct
Square	100	0	0	0	0	100
Circle	0	100	0	0	0	100
Oval	8	0	83	1	8	83
Rectangle	27	1	16	36	20	36
Blob	14	0	36	22	26	26

Table 1: Minimum Distance Confusion Matrix

Execution Time for Brute Force = 100.7366 seconds

Total % Correct Class Assignments = 94.2%

	Square	Circle	Oval	Rectangle	Blob	% Correct
Square	99	0	0	1	0	99
Circle	0	99	0	0	1	99
Oval	0	0	99	1	0	99
Rectangle	3	0	6	88	3	88
Blob	2	0	0	12	86	86

Table 2: Brute Force Minimum Distance Confusion Matrix

Conclusions

The first minimum distance method using the mean feature vectors of each class seen in Table 1 was successfully able to detect all squares, circles, and most oval shapes, but struggled to detect rectangles and random gaussian blobs. These results were unexpected towards the algorithm finding the rectangles, as it would be assumed to have some bleed-over with false detections towards squares, which was its highest missed guess, but it still is quite abysmal performance. Maybe including a Bayes classification method towards probabilities on-top of this method would improve overall correct pattern recognition rate, but it is something to be researched.

The brute-force method had more favorable results, with strong recognition towards squares, circles, and ovals at 99%, and drastically improved detection accuracy towards rectangles and blobs. As expected, the rectangle had a little bit of bleed over with squares and ovals, but could be further improved by adding more training data images or including a Bayes probability model as discussed previously. Surprisingly, there was not a great deal of computational timing performance between the two algorithms, which the `SI_Moments()` external MATLAB function is believed to be the main time-sink.

Overall, the minimum distance classification method is a very flexible and simple pattern recognition algorithm that can be easily modified and built upon to improve pattern recognition success rate in certain controlled testing environments as shown in this report between the standard and brute-force methods.

References

1. R. Gonzalez, R. Woods, “Digital Image Processing 4th Edition”, *Pearson*, New York, NY, 2018, Ch. 13, pp. 1056 - 1058.
2. M. Hu, “Visual Pattern Recognition by Moment Invariants”, *IRE Transaction on Information Theory*, Feb. 1962, Vo. 8, Issue 2, pp. 179-187, [Online] Available: <https://ieeexplore.ieee.org/document/1057692>, [Accessed: Mar. 28, 2021]
3. M. Roggemann, “make_test_image.m”, *Michigan Technological University*, Houghton, MI, 2021, [Online] Available: https://mtu.instructure.com/courses/1347065/pages/module-5-lesson-4?module_item_id=17403062, [Accessed: Mar. 28, 2021]
4. MATLAB, “Tic”, *MathWorks*, R2021a, 2021, [Online] Available: <https://www.mathworks.com/help/matlab/ref/tic.html>, [Accessed: Mar. 28, 2021]
5. V. Muralidharan, “SI_Moment.m”, *University of Alabama in Huntsville*, [Online] Available: https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/52259/versions/1/previews/Hu%27s%20Moments/SI_Moment.m/index.html, [Accessed: Mar. 28, 2021]

Appendix

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Pgrm   : Module 5
3  % Author : Matthew Spencer
4  % Class  : EE5367
5  % Date   : 03/26/2020
6  % Desc   : Pattern recognition of rectangles, squares, circles, ovals, and
7  %          blobs using a pre-made training data set. Implements a minimum
8  %          distance classification method and Hu Moments as features.
9  % Input   : None. Auto-load .mat file from project page.
10 % Output  : Confusion matrix.
11 % Notes   : None.
12 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
13 function MinDistance()
14 %----- Initializations
15 % Clear all images and cmd window
16 close all;
17 clear;
18 clc;
19
20 % Start timer
21 tStart = tic;
22
23 % Load the training data set
24 tData = load("PR_training_set2.mat");
25 tTest = load("testing_set.mat");
26
27 % Init number of training images to parse
28 N      = 1000;
29
30 % Init data vectors
31 sqData = zeros(16,1);
32 circData = zeros(16,1);
33 ovalData = zeros(16,1);
34 rectData = zeros(16,1);
35 blobData = zeros(16,1);
36
37 % Init image crops
38 sqCrop = [10, 10, 99, 99];
39 circCrop = [175, 35, 99, 99];
40 ovalCrop = [20, 165, 99, 99];
41 rectCrop = [170, 170, 99, 99];
42 blobCrop = [95, 85, 99, 99];
43
44 % Running tallies for taking the mean
45 s = 0;
46 c = 0;
47 o = 0;
48 r = 0;
49 b = 0;
50
51 % Init confusion matrix
52 confusion = zeros(5,5);
53
```

Appendix 1: MinDistance.m Part 1

```

54 %----- Calculations
55 disp("Gathering moment features from training data...");
56 % Get test figure and its moments
57 % Loop through training data
58 for i = 1:N
59     % Get next training image
60     img = mat2gray(tData.img_stack(:,:,i));
61
62     % Acquire crops of each shape
63     square = imcrop(img, sqCrop);
64     circle = imcrop(img, circCrop);
65     oval = imcrop(img, ovalCrop);
66     rect = imcrop(img, rectCrop);
67     blob = imcrop(img, blobCrop);
68
69     % Find Hu moment features of each shape, make sure it is not NaN
70     x = SI_Moment(square);
71     if (~isnan(x(1,3)))
72         sqData = sqData + x(:);
73         s = s + 1;
74     end
75     x = SI_Moment(circle);
76     if (~isnan(x(1,3)))
77         circData = circData + x(:);
78         c = c + 1;
79     end
80     x = SI_Moment(oval);
81     if (~isnan(x(1,3)))
82         ovalData = ovalData + x(:);
83         o = o + 1;
84     end
85     x = SI_Moment(rect);
86     if (~isnan(x(1,3)))
87         rectData = rectData + x(:);
88         r = r + 1;
89     end
90     x = SI_Moment(blob);
91     if (~isnan(x(1,3)))
92         blobData = blobData + x(:);
93         b = b + 1;
94     end
95 end
96
97 % Find the mean of each vector feature
98 sqData = sqData./s;
99 circData = circData./c;
100 ovalData = ovalData./o;
101 rectData = rectData./r;
102 blobData = blobData./b;
103
104 % Loop through testing data set images
105 disp("Parsing shapes from test images...")
106 for k = 1:100
107     fprintf("Image %d\n", k);
108     % Get next test image

```

Appendix 2: MinDistance.m Part 2

```

% Get next test image
currImg = tTest.img_stack(:,:,k);%testImgs(:,:,k);

% Get Each shape from test image
imgCrop1      = imcrop(currImg, sqCrop);%crop1);
imgCrop2      = imcrop(currImg, circCrop);%crop2);
imgCrop3      = imcrop(currImg, ovalCrop);%crop3);
imgCrop4      = imcrop(currImg, rectCrop);%crop3);
imgCrop5      = imcrop(currImg, blobCrop);%crop3);
testImgCrop    = zeros(100,100,5);
testImgCrop(:,:,1) = imgCrop1;
testImgCrop(:,:,2) = imgCrop2;
testImgCrop(:,:,3) = imgCrop3;
testImgCrop(:,:,4) = imgCrop4;
testImgCrop(:,:,5) = imgCrop5;

% Get moments of each shape
data = zeros(16,1,5);
x = SI_Moment(testImgCrop(:,:,1));
if (~isnan(x(1,3)))
    data(:,1,1) = x(:);
end
x = SI_Moment(testImgCrop(:,:,2));
if (~isnan(x(1,3)))
    data(:,1,2) = x(:);
end
x = SI_Moment(testImgCrop(:,:,3));
if (~isnan(x(1,3)))
    data(:,1,3) = x(:);
end
x = SI_Moment(testImgCrop(:,:,4));
if (~isnan(x(1,3)))
    data(:,1,4) = x(:);
end
x = SI_Moment(testImgCrop(:,:,5));
if (~isnan(x(1,3)))
    data(:,1,5) = x(:);
end

% Init estimate variables
sqEst    = zeros(16,1,5);
circEst  = zeros(16,1,5);
ovalEst  = zeros(16,1,5);
rectEst  = zeros(16,1,5);
blobEst  = zeros(16,1,5);

% Get current estimates for each cropped shape
for j = 1:5
    sqEst(:,1,j)    = data(:,1,j) - sqData;
    circEst(:,1,j)  = data(:,1,j) - circData;
    ovalEst(:,1,j)  = data(:,1,j) - ovalData;
    rectEst(:,1,j)  = data(:,1,j) - rectData;
    blobEst(:,1,j)  = data(:,1,j) - blobData;
end

```

```

162
163 % Find normalized distances for each est
164 sqDist = zeros(1,5);
165 circDist = zeros(1,5);
166 ovalDist = zeros(1,5);
167 rectDist = zeros(1,5);
168 blobDist = zeros(1,5);
169 for j = 1:5
170     sqDist(1,j) = norm(sqEst(:,1,j));
171     circDist(1,j) = norm(circEst(:,1,j));
172     ovalDist(1,j) = norm(ovalEst(:,1,j));
173     rectDist(1,j) = norm(rectEst(:,1,j));
174     blobDist(1,j) = norm(blobEst(:,1,j));
175 end
176
177 % Make sure we have the minimal estimates
178 for j = 1:5
179     % Get current distances
180     distances = [sqDist(1,j), circDist(1,j), ovalDist(1,j),...
181                 rectDist(1,j), blobDist(1,j)];
182
183     % Find the minimal distance
184     min = 99999;
185     est = 5;
186     for p = 1:5
187         if (distances(p) < min)
188             min = distances(p);
189             est = p;
190         end
191     end
192
193     % Create confusion matrix to store estimate
194     % Square
195     if (est == 1 && j == 1)
196         confusion(1,1) = confusion(1,1)+1;
197     end
198     if (est == 2 && j == 1)
199         confusion(1,2) = confusion(1,2)+1;
200     end
201     if (est == 3 && j == 1)
202         confusion(1,3) = confusion(1,3)+1;
203     end
204     if (est == 4 && j == 1)
205         confusion(1,4) = confusion(1,4)+1;
206     end
207     if (est == 5 && j == 1)
208         confusion(1,5) = confusion(1,5)+1;
209     end
210
211     % Circle
212     if (est == 1 && j == 2)
213         confusion(2,1) = confusion(2,1)+1;
214     end
215     if (est == 2 && j == 2)
216         confusion(2,2) = confusion(2,2)+1;

```

Appendix 3: MinDistance.m Part 3

```

215     if (est == 2 && j == 2)
216         confusion(2,2) = confusion(2,2)+1;
217     end
218     if (est == 3 && j == 2)
219         confusion(2,3) = confusion(2,3)+1;
220     end
221     if (est == 4 && j == 2)
222         confusion(2,4) = confusion(2,4)+1;
223     end
224     if (est == 5 && j == 2)
225         confusion(2,5) = confusion(2,5)+1;
226     end
227
228     % Oval
229     if (est == 1 && j == 3)
230         confusion(3,1) = confusion(3,1)+1;
231     end
232     if (est == 2 && j == 3)
233         confusion(3,2) = confusion(3,2)+1;
234     end
235     if (est == 3 && j == 3)
236         confusion(3,3) = confusion(3,3)+1;
237     end
238     if (est == 4 && j == 3)
239         confusion(3,4) = confusion(3,4)+1;
240     end
241     if (est == 5 && j == 3)
242         confusion(3,5) = confusion(3,5)+1;
243     end
244
245     % Rectangle
246     if (est == 1 && j == 4)
247         confusion(4,1) = confusion(4,1)+1;
248     end
249     if (est == 2 && j == 4)
250         confusion(4,2) = confusion(4,2)+1;
251     end
252     if (est == 3 && j == 4)
253         confusion(4,3) = confusion(4,3)+1;
254     end
255     if (est == 4 && j == 4)
256         confusion(4,4) = confusion(4,4)+1;
257     end
258     if (est == 5 && j == 4)
259         confusion(4,5) = confusion(4,5)+1;
260     end
261
262     % Blob
263     if (est == 1 && j == 5)
264         confusion(5,1) = confusion(5,1)+1;
265     end
266     if (est == 2 && j == 5)
267         confusion(5,2) = confusion(5,2)+1;

```

Appendix 4: MinDistance.m Part 4

```

266     if (est == 2 && j == 5)
267         confusion(5,2) = confusion(5,2)+1;
268     end
269     if (est == 3 && j == 5)
270         confusion(5,3) = confusion(5,3)+1;
271     end
272     if (est == 4 && j == 5)
273         confusion(5,4) = confusion(5,4)+1;
274     end
275     if (est == 5 && j == 5)
276         confusion(5,5) = confusion(5,5)+1;
277     end
278 end
279 end
280
281 % Print true numbers compared to estimated numbers
282 disp(confusion);
283 disp("Finished!");
284 tEnd = toc(tStart)
285 end
286 % EOF =====

```

Appendix 5: MinDistance.m Part 5

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Pgrm   : Module 5
3  % Author : Matthew Spencer
4  % Class  : EE5367
5  % Date   : 03/26/2020
6  % Desc   : Pattern recognition of rectangles, squares, circles, ovals, and
7  %           blobs using a pre-made training data set. Implements a minimum
8  %           distance classification method and Hu Moments as features,
9  %           changes the standard algorithm and compares against every
10 %           training image for each class, thus "brute forcing" a min
11 %           distance.
12 % Input   : None. Auto-load .mat file from project page.
13 % Output  : Confusion matrix.
14 % Notes   : None.
15 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
16 function MinDistance_BruteForce()
17 %----- Initializations
18 % Clear all images and cmd window
19 close all;
20 clear;
21 clc;
22
23 % Start timer
24 tStart = tic;
25
26 % Load the training data set
27 tData = load("PR_training_set2.mat");
28 tTest = load("testing_set.mat");
29
30 % Init parameters
31 N      = 1000;
32
33 % Init data vectors
34 sqData  = zeros(4,4,N);
35 circData = zeros(4,4,N);
36 ovalData = zeros(4,4,N);
37 rectData = zeros(4,4,N);
38 blobData = zeros(4,4,N);
39
40 % Init image crops
41 sqCrop  = [10, 10, 99, 99]; % 100
42 circCrop = [175, 35, 99, 99]; % 100
43 ovalCrop = [20, 165, 99, 99]; % 110
44 rectCrop = [170, 170, 99, 99]; % 60
45 blobCrop = [95, 85, 99, 99]; % 80
46
47 % Init confusion matrix
48 confusion = zeros(5,5);

```

Appendix 6: BruteForce_MinDistance.m Part 1


```

49 %----- Calculations
50 disp("Gathering moments from training data...");
51 % Get test figure and its moments
52 % Loop through training data
53 for i = 1:1000
54     % Get next training image
55     img = mat2gray(tData.img_stack(:,:,i));
56
57     % Acquire crops of each shape
58     square = imcrop(img, sqCrop);
59     circle = imcrop(img, circCrop);
60     oval = imcrop(img, ovalCrop);
61     rect = imcrop(img, rectCrop);
62     blob = imcrop(img, blobCrop);
63
64     % Find Hu moment features of each shape, make sure it is not NaN
65     x = SI_Moment(square);
66     if (~isnan(x(1,3)))
67         sqData(:,:,i) = x;
68     end
69     x = SI_Moment(circle);
70     if (~isnan(x(1,3)))
71         circData(:,:,i) = x;
72     end
73     x = SI_Moment(oval);
74     if (~isnan(x(1,3)))
75         ovalData(:,:,i) = x;
76     end
77     x = SI_Moment(rect);
78     if (~isnan(x(1,3)))
79         rectData(:,:,i) = x;
80     end
81     x = SI_Moment(blob);
82     if (~isnan(x(1,3)))
83         blobData(:,:,i) = x;
84     end
85 end
86
87 % Loop through testing data set images
88 disp("Parsing shapes from test images...")
89 for k = 1:100
90     fprintf("Image %d\n", k);
91     % Get next test image
92     currImg = tTest.img_stack(:,:,k);
93
94     % Get Each shape from test image
95     imgCrop1 = imcrop(currImg, sqCrop);%crop1;
96     imgCrop2 = imcrop(currImg, circCrop);%crop2;
97     imgCrop3 = imcrop(currImg, ovalCrop);%crop3;
98     imgCrop4 = imcrop(currImg, rectCrop);%crop3;
99     imgCrop5 = imcrop(currImg, blobCrop);%crop3;
100     testImgCrop = zeros(100,100,5);
101     testImgCrop(:,:,1) = imgCrop1;
102     testImgCrop(:,:,2) = imgCrop2;
103     testImgCrop(:,:,3) = imgCrop3;
104     testImgCrop(:,:,4) = imgCrop4;
105     testImgCrop(:,:,5) = imgCrop5;

```

Appendix 7: BruteForce_MinDistance.m Part 2

```

107 % Get moments of each shape
108 data = zeros(4,4,5);
109 data(:,:,1) = SI_Moment(testImgCrop(:,:,1));
110 data(:,:,2) = SI_Moment(testImgCrop(:,:,2));
111 data(:,:,3) = SI_Moment(testImgCrop(:,:,3));
112 data(:,:,4) = SI_Moment(testImgCrop(:,:,4));
113 data(:,:,5) = SI_Moment(testImgCrop(:,:,5));
114
115 % Init estimate variables
116 max = 999999;
117 % Init estimate variables
118 sqEst = [max,max,max,max,max];
119 circEst = [max,max,max,max,max];
120 ovalEst = [max,max,max,max,max];
121 rectEst = [max,max,max,max,max];
122 blobEst = [max,max,max,max,max];
123
124 % Loop through data sets
125 for i = 1:N
126     % Get current moment data
127     sqMom = sqData(:,:,i);
128     circMom = circData(:,:,i);
129     ovalMom = ovalData(:,:,i);
130     rectMom = rectData(:,:,i);
131     blobMom = blobData(:,:,i);
132
133     % Check to make sure the moment data is valid
134     check = sum(sum(abs(sqMom)));
135     if (check == 0 || isnan(check))
136         sqMom = [max,max,max,max;max,max,max,max;...
137                 max,max,max,max;max,max,max,max];
138     end
139     check = sum(sum(abs(circMom)));
140     if (check == 0 || isnan(check))
141         circMom = [max,max,max,max;max,max,max,max;...
142                   max,max,max,max;max,max,max,max];
143     end
144     check = sum(sum(abs(ovalMom)));
145     if (check == 0 || isnan(check))
146         ovalMom = [max,max,max,max;max,max,max,max;...
147                   max,max,max,max;max,max,max,max];
148     end
149     check = sum(sum(abs(rectMom)));
150     if (check == 0 || isnan(check))
151         rectMom = [max,max,max,max;max,max,max,max;...
152                   max,max,max,max;max,max,max,max];
153     end
154     check = sum(sum(abs(blobMom)));
155     if (check == 0 || isnan(check))
156         blobMom = [max,max,max,max;max,max,max,max;...
157                   max,max,max,max;max,max,max,max];
158     end
159
160     % Get current estimates using moment data
161     for j = 1:5
162         sqE = sum(sum(abs(sqMom-data(:,:,j))));

```

Appendix 8: BruteForce_MinDistance.m Part 3

```

163         circE = sum(sum(abs(circMom-data(:,:,j))));
164         ovalE = sum(sum(abs(ovalMom-data(:,:,j))));
165         rectE = sum(sum(abs(rectMom-data(:,:,j))));
166         blobE = sum(sum(abs(blobMom-data(:,:,j))));
167
168         % Check to see if it is the minimum
169         if (sqE < sqEst(j))
170             sqEst(j) = sqE;
171         end
172         if (circE < circEst(j))
173             circEst(j) = circE;
174         end
175         if (ovalE < ovalEst(j))
176             ovalEst(j) = ovalE;
177         end
178         if (rectE < rectEst(j))
179             rectEst(j) = rectE;
180         end
181         if (blobE < blobEst(j))
182             blobEst(j) = blobE;
183         end
184     end
185 end
186
187 % Make sure we have the minimal estimates
188 for j = 1:5
189     % Get current distances
190     distances = [sqEst(1,j), circEst(1,j), ovalEst(1,j),...
191                 rectEst(1,j), blobEst(1,j)];
192
193     % Find the minimal distance
194     min = 99999;
195     est = 5;
196     for p = 1:5
197         if (distances(p) < min)
198             min = distances(p);
199             est = p;
200         end
201     end
202
203     % Create confusion matrix to store estimate
204     % Square
205     if (est == 1 && j == 1)
206         confusion(1,1) = confusion(1,1)+1;
207     end
208     if (est == 2 && j == 1)
209         confusion(1,2) = confusion(1,2)+1;
210     end
211     if (est == 3 && j == 1)
212         confusion(1,3) = confusion(1,3)+1;
213     end
214     if (est == 4 && j == 1)
215         confusion(1,4) = confusion(1,4)+1;
216     end

```

Appendix 9: BruteForce_MinDistance.m Part 4

```

216 end
217 if (est == 5 && j == 1)
218     confusion(1,5) = confusion(1,5)+1;
219 end
220
221 % Circle
222 if (est == 1 && j == 2)
223     confusion(2,1) = confusion(2,1)+1;
224 end
225 if (est == 2 && j == 2)
226     confusion(2,2) = confusion(2,2)+1;
227 end
228 if (est == 3 && j == 2)
229     confusion(2,3) = confusion(2,3)+1;
230 end
231 if (est == 4 && j == 2)
232     confusion(2,4) = confusion(2,4)+1;
233 end
234 if (est == 5 && j == 2)
235     confusion(2,5) = confusion(2,5)+1;
236 end
237
238 % Oval
239 if (est == 1 && j == 3)
240     confusion(3,1) = confusion(3,1)+1;
241 end
242 if (est == 2 && j == 3)
243     confusion(3,2) = confusion(3,2)+1;
244 end
245 if (est == 3 && j == 3)
246     confusion(3,3) = confusion(3,3)+1;
247 end
248 if (est == 4 && j == 3)
249     confusion(3,4) = confusion(3,4)+1;
250 end
251 if (est == 5 && j == 3)
252     confusion(3,5) = confusion(3,5)+1;
253 end
254
255 % Rectangle
256 if (est == 1 && j == 4)
257     confusion(4,1) = confusion(4,1)+1;
258 end
259 if (est == 2 && j == 4)
260     confusion(4,2) = confusion(4,2)+1;
261 end
262 if (est == 3 && j == 4)
263     confusion(4,3) = confusion(4,3)+1;
264 end
265 if (est == 4 && j == 4)
266     confusion(4,4) = confusion(4,4)+1;
267 end
268 if (est == 5 && j == 4)
269     confusion(4,5) = confusion(4,5)+1;
270 end

```

Appendix 10: BruteForce_MinDistance.m Part 5

```

271
272         % Blob
273         if (est == 1 && j == 5)
274             confusion(5,1) = confusion(5,1)+1;
275         end
276         if (est == 2 && j == 5)
277             confusion(5,2) = confusion(5,2)+1;
278         end
279         if (est == 3 && j == 5)
280             confusion(5,3) = confusion(5,3)+1;
281         end
282         if (est == 4 && j == 5)
283             confusion(5,4) = confusion(5,4)+1;
284         end
285         if (est == 5 && j == 5)
286             confusion(5,5) = confusion(5,5)+1;
287         end
288     end
289 end
290
291 % Print true numbers compared to estimated numbers
292 disp(confusion);
293 disp("Finished!");
294 tEnd = toc(tStart)
295 end
296 % EOF =====

```

Appendix 11: BruteForce_MinDistance.m Part 6