Pattern Recognition

EE5532 Module 5 Assignment

Matthew Spencer

03/28/2021

Contents

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.

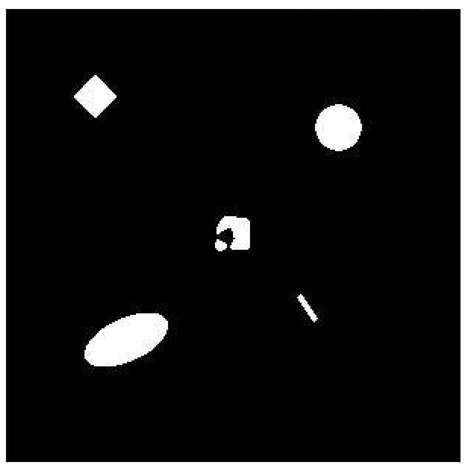

Fig. 1: Training Image

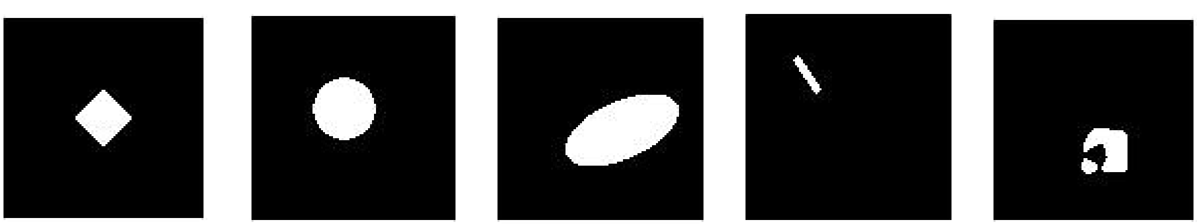

Fig. 2: Parsed Training Patterns from Fig. 1

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].

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

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.

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

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
```

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.

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

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
```

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

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pgrm   : Module 5
% Author : Matthew Spencer
% Class  : EE5367
% Date   : 03/26/2020
% Desc   : Pattern recognition of rectangles, squares, circles, ovals, and
%          blobs using a pre-made training data set. Implements a minimum
%          distance classification method and Hu Moments as features.
% Input  : None. Auto-load .mat file from project page.
% Output : Confustion matrix.
% Notes  : None.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function MinDistance()
%----------------------------------------------------------- Initializations
    % Clear all images and cmd window
    close all;
    clear;
    clc;

    % Start timer
    tStart = tic;

    % Load the training data set
    tData = load("PR_training_set2.mat");
    tTest = load("testing_set.mat");

    % Init number of training images to parse
    N     = 1000;

    % Init data vectors
    sqData   = zeros(16,1);
    circData = zeros(16,1);
    ovalData = zeros(16,1);
    rectData = zeros(16,1);
    blobData = zeros(16,1);

    % Init image crops
    sqCrop   = [10,  10,  99, 99];
    circCrop = [175, 35,  99, 99];
    ovalCrop = [20,  165, 99, 99];
    rectCrop = [170, 170, 99, 99];
    blobCrop = [95, 85, 99, 99];

    % Running tallies for taking the mean
    s = 0;
    c = 0;
    o = 0;
    r = 0;
    b = 0;

    % Init confustion matrix
    confusion = zeros(5,5);

```

Appendix 1: MinDistance.m Part 1

```matlab
%------------------------------------------------------------ Calculations
    disp("Gathering moment features from training data...");
    % Get test figure and its moments
    % Loop through training data
    for i = 1:N
        % Get next training image
        img = mat2gray(tData.img_stack(:,:,i));

        % Acquire crops of each shape
        square = imcrop(img, sqCrop);
        circle = imcrop(img, circCrop);
        oval   = imcrop(img, ovalCrop);
        rect   = imcrop(img, rectCrop);
        blob   = imcrop(img, blobCrop);

        % Find Hu moment features of each shape, make sure it is not NaN
        x = SI_Moment(square);
        if (~isnan(x(1,3)))
            sqData = sqData + x(:);
            s = s + 1;
        end
        x = SI_Moment(circle);
        if (~isnan(x(1,3)))
            circData = circData + x(:);
            c = c + 1;
        end
        x = SI_Moment(oval);
        if (~isnan(x(1,3)))
            ovalData = ovalData + x(:);
            o = o + 1;
        end
        x = SI_Moment(rect);
        if (~isnan(x(1,3)))
            rectData = rectData + x(:);
            r = r + 1;
        end
        x = SI_Moment(blob);
        if (~isnan(x(1,3)))
            blobData = blobData + x(:);
            b = b + 1;
        end
    end

    % Find the mean of each vector feature
    sqData   = sqData./s;
    circData = circData./c;
    ovalData = ovalData./o;
    rectData = rectData./r;
    blobData = blobData./b;

    % Loop through testing data set images
    disp("Parsing shapes from test images...")
    for k = 1:100
        %fprintf("Image %d\n", k);
        % 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
```

```
        % 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

            % Create confusion matrix to store estimate
            % Square
            if (est == 1 && j == 1)
               confusion(1,1) = confusion(1,1)+1;
            end
            if (est == 2 && j == 1)
               confusion(1,2) = confusion(1,2)+1;
            end
            if (est == 3 && j == 1)
               confusion(1,3) = confusion(1,3)+1;
            end
            if (est == 4 && j == 1)
               confusion(1,4) = confusion(1,4)+1;
            end
            if (est == 5 && j == 1)
               confusion(1,5) = confusion(1,5)+1;
            end

            % Circle
            if (est == 1 && j == 2)
               confusion(2,1) = confusion(2,1)+1;
            end
            if (est == 2 && j == 2)
               confusion(2,2) = confusion(2,2)+1;
```

Appendix 3: MinDistance.m Part 3

```
            if (est == 2 && j == 2)
                confusion(2,2) = confusion(2,2)+1;
            end
            if (est == 3 && j == 2)
                confusion(2,3) = confusion(2,3)+1;
            end
            if (est == 4 && j == 2)
                confusion(2,4) = confusion(2,4)+1;
            end
            if (est == 5 && j == 2)
                confusion(2,5) = confusion(2,5)+1;
            end

            % Oval
            if (est == 1 && j == 3)
                confusion(3,1) = confusion(3,1)+1;
            end
            if (est == 2 && j == 3)
                confusion(3,2) = confusion(3,2)+1;
            end
            if (est == 3 && j == 3)
                confusion(3,3) = confusion(3,3)+1;
            end
            if (est == 4 && j == 3)
                confusion(3,4) = confusion(3,4)+1;
            end
            if (est == 5 && j == 3)
                confusion(3,5) = confusion(3,5)+1;
            end

            % Rectangle
            if (est == 1 && j == 4)
                confusion(4,1) = confusion(4,1)+1;
            end
            if (est == 2 && j == 4)
                confusion(4,2) = confusion(4,2)+1;
            end
            if (est == 3 && j == 4)
                confusion(4,3) = confusion(4,3)+1;
            end
            if (est == 4 && j == 4)
                confusion(4,4) = confusion(4,4)+1;
            end
            if (est == 5 && j == 4)
                confusion(4,5) = confusion(4,5)+1;
            end

            % Blob
            if (est == 1 && j == 5)
                confusion(5,1) = confusion(5,1)+1;
            end
            if (est == 2 && j == 5)
                confusion(5,2) = confusion(5,2)+1;
```

Appendix 4: MinDistance.m Part 4

```
            if (est == 2 && j == 5)
                confusion(5,2) = confusion(5,2)+1;
            end
            if (est == 3 && j == 5)
                confusion(5,3) = confusion(5,3)+1;
            end
            if (est == 4 && j == 5)
                confusion(5,4) = confusion(5,4)+1;
            end
            if (est == 5 && j == 5)
                confusion(5,5) = confusion(5,5)+1;
            end
        end
    end

    % Print true numbers compared to estimated numbers
    disp(confusion);
    disp("Finished!");
    tEnd = toc(tStart)
end
% EOF ======================================================================
```

Appendix 5: MinDistance.m Part 5

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pgrm   : Module 5
% Author : Matthew Spencer
% Class  : EE5367
% Date   : 03/26/2020
% Desc   : Pattern recognition of rectangles, squares, circles, ovals, and
%          blobs using a pre-made training data set. Implements a minimum
%          distance classification method and Hu Moments as features,
%          changes the standard algorithm and compares against every
%          training image for each class, thus "brute forcing" a min
%          distance.
% Input  : None. Auto-load .mat file from project page.
% Output : Confustion matrix.
% Notes  : None.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function MinDistance_BruteForce()
%---------------------------------------------------------- Initializations
    % Clear all images and cmd window
    close all;
    clear;
    clc;

    % Start timer
    tStart = tic;

    % Load the training data set
    tData = load("PR_training_set2.mat");
    tTest = load("testing_set.mat");

    % Init parameters
    N     = 1000;

    % Init data vectors
    sqData   = zeros(4,4,N);
    circData = zeros(4,4,N);
    ovalData = zeros(4,4,N);
    rectData = zeros(4,4,N);
    blobData = zeros(4,4,N);

    % Init image crops
    sqCrop   = [10,   10,   99,  99]; % 100
    circCrop = [175,  35,   99,  99]; % 100
    ovalCrop = [20,   165,  99,  99]; % 110
    rectCrop = [170,  170,  99,  99]; % 60
    blobCrop = [95,   85,   99,  99]; % 80

    % Init confustion matrix
    confusion = zeros(5,5);
```

Appendix 6: BruteForce_MinDistance.m Part 1

```
%------------------------------------------------------------- Calculations
    disp("Gathering moments from training data...");
    % Get test figure and its moments
    % Loop through training data
    for i = 1:1000
        % Get next training image
        img = mat2gray(tData.img_stack(:,:,i));

        % Acquire crops of each shape
        square = imcrop(img, sqCrop);
        circle = imcrop(img, circCrop);
        oval   = imcrop(img, ovalCrop);
        rect   = imcrop(img, rectCrop);
        blob   = imcrop(img, blobCrop);

        % Find Hu moment features of each shape, make sure it is not NaN
        x = SI_Moment(square);
        if (~isnan(x(1,3)))
            sqData(:,:,i) = x;
        end
        x = SI_Moment(circle);
        if (~isnan(x(1,3)))
            circData(:,:,i) = x;
        end
        x = SI_Moment(oval);
        if (~isnan(x(1,3)))
            ovalData(:,:,i) = x;
        end
        x = SI_Moment(rect);
        if (~isnan(x(1,3)))
            rectData(:,:,i) = x;
        end
        x = SI_Moment(blob);
        if (~isnan(x(1,3)))
            blobData(:,:,i) = x;
        end
    end

    % Loop through testing data set images
    disp("Parsing shapes from test images...")
    for k = 1:100
        fprintf("Image %d\n", k);
        % Get next test image
        currImg = tTest.img_stack(:,:,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;
```

Appendix 7: BruteForce_MinDistance.m Part 2

```
    % Get moments of each shape
    data = zeros(4,4,5);
    data(:,:,1) = SI_Moment(testImgCrop(:,:,1));
    data(:,:,2) = SI_Moment(testImgCrop(:,:,2));
    data(:,:,3) = SI_Moment(testImgCrop(:,:,3));
    data(:,:,4) = SI_Moment(testImgCrop(:,:,4));
    data(:,:,5) = SI_Moment(testImgCrop(:,:,5));

    % Init estimate variables
    max = 999999;
    % Init estimate variables
    sqEst   = [max,max,max,max,max];
    circEst = [max,max,max,max,max];
    ovalEst = [max,max,max,max,max];
    rectEst = [max,max,max,max,max];
    blobEst = [max,max,max,max,max];

    % Loop through data sets
    for i = 1:N
        % Get current moment data
        sqMom   = sqData(:,:,i);
        circMom = circData(:,:,i);
        ovalMom = ovalData(:,:,i);
        rectMom = rectData(:,:,i);
        blobMom = blobData(:,:,i);

        % Check to make sure the moment data is valid
        check = sum(sum(abs(sqMom)));
        if (check == 0 || isnan(check))
            sqMom = [max,max,max,max;max,max,max,max;...
                     max,max,max,max;max,max,max,max];
        end
        check = sum(sum(abs(circMom)));
        if (check == 0 || isnan(check))
            circMom = [max,max,max,max;max,max,max,max;...
                       max,max,max,max;max,max,max,max];
        end
        check = sum(sum(abs(ovalMom)));
        if (check == 0 || isnan(check))
            ovalMom = [max,max,max,max;max,max,max,max;...
                       max,max,max,max;max,max,max,max];
        end
        check = sum(sum(abs(rectMom)));
        if (check == 0 || isnan(check))
            rectMom = [max,max,max,max;max,max,max,max;...
                       max,max,max,max;max,max,max,max];
        end
        check = sum(sum(abs(blobMom)));
        if (check == 0 || isnan(check))
            blobMom = [max,max,max,max;max,max,max,max;...
                       max,max,max,max;max,max,max,max];
        end

        % Get current estimates using moment data
        for j = 1:5
            sqE   = sum(sum(abs(sqMom-data(:,:,j))));
```

Appendix 8: BruteForce_MinDistance.m Part 3

```
            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

    % Make sure we have the minimal estimates
    for j = 1:5
        % Get current distances
        distances = [sqEst(1,j), circEst(1,j), ovalEst(1,j),...
            rectEst(1,j), blobEst(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

        % Create confusion matrix to store estimate
        % Square
        if (est == 1 && j == 1)
            confusion(1,1) = confusion(1,1)+1;
        end
        if (est == 2 && j == 1)
            confusion(1,2) = confusion(1,2)+1;
        end
        if (est == 3 && j == 1)
            confusion(1,3) = confusion(1,3)+1;
        end
        if (est == 4 && j == 1)
            confusion(1,4) = confusion(1,4)+1;
        end
```

Appendix 9: BruteForce_MinDistance.m Part 4

```
            end
            if (est == 5 && j == 1)
                confusion(1,5) = confusion(1,5)+1;
            end

            % Circle
            if (est == 1 && j == 2)
                confusion(2,1) = confusion(2,1)+1;
            end
            if (est == 2 && j == 2)
                confusion(2,2) = confusion(2,2)+1;
            end
            if (est == 3 && j == 2)
                confusion(2,3) = confusion(2,3)+1;
            end
            if (est == 4 && j == 2)
                confusion(2,4) = confusion(2,4)+1;
            end
            if (est == 5 && j == 2)
                confusion(2,5) = confusion(2,5)+1;
            end

            % Oval
            if (est == 1 && j == 3)
                confusion(3,1) = confusion(3,1)+1;
            end
            if (est == 2 && j == 3)
                confusion(3,2) = confusion(3,2)+1;
            end
            if (est == 3 && j == 3)
                confusion(3,3) = confusion(3,3)+1;
            end
            if (est == 4 && j == 3)
                confusion(3,4) = confusion(3,4)+1;
            end
            if (est == 5 && j == 3)
                confusion(3,5) = confusion(3,5)+1;
            end

            % Rectangle
            if (est == 1 && j == 4)
                confusion(4,1) = confusion(4,1)+1;
            end
            if (est == 2 && j == 4)
                confusion(4,2) = confusion(4,2)+1;
            end
            if (est == 3 && j == 4)
                confusion(4,3) = confusion(4,3)+1;
            end
            if (est == 4 && j == 4)
                confusion(4,4) = confusion(4,4)+1;
            end
            if (est == 5 && j == 4)
                confusion(4,5) = confusion(4,5)+1;
            end
```

Appendix 10: BruteForce_MinDistance.m Part 5

```
                % Blob
                if (est == 1 && j == 5)
                    confusion(5,1) = confusion(5,1)+1;
                end
                if (est == 2 && j == 5)
                    confusion(5,2) = confusion(5,2)+1;
                end
                if (est == 3 && j == 5)
                    confusion(5,3) = confusion(5,3)+1;
                end
                if (est == 4 && j == 5)
                    confusion(5,4) = confusion(5,4)+1;
                end
                if (est == 5 && j == 5)
                    confusion(5,5) = confusion(5,5)+1;
                end
            end
        end

    % Print true numbers compared to estimated numbers
    disp(confusion);
    disp("Finished!");
    tEnd = toc(tStart)
end
% EOF ============================================================================
```

Appendix 11: BruteForce_MinDistance.m Part 6