Case Study: Classification of Pool Balls
Case Study: Pool Ball Identification

- Assume we have detected a ball in an image, and have extracted a subimage of the ball.

What type of ball is it?
Ball Types

• There are 16 different pool balls

0: Cue ball

Solid Balls:
1: Yellow, 2: Blue, 3: Red,
4: Purple, 5: Orange, 6: Green,
7: Maroon, 8: Black

Striped Balls:
9: Yellow, 10: Blue, 11: Red,
12: Purple, 13: Orange,
14: Green, 15: Maroon

• A histogram of colors should be a good feature
Movies on course website

- For training:
  - TableA_movie1.MOV
  - TableA_movie2.MOV
  - TableA_movie3.MOV
  - TableA_movie4.MOV

- For testing:
  - TableB_movie1.MOV
  - TableB_movie2.MOV
  - TableB_movie3.MOV
Extracting training images

• Have the user interactively crop out a subimage of a ball using the mouse, and label the type

• See program “extractImages” on next page
• Edit it to specify the path and filename of the input movie
• Edit it to specify the output directory: “trainingImages” or “testImages”

• It will write out files called “ballXX_YYY.png”, where
  – XX is the ball id
  – YYY is image number
Program “extractBalls” (1 of 2)

% Extract images of pool balls.
clear all
close all

disp('Use this program to crop images of pool balls.');
disp('Here are the types of balls:');
disp('   0:Cue ball');
disp(' Solid Balls:');
disp('   1:Yellow, 2:Blue, 3:Red, 4:Purple, 5:Orange, 6:Green, 7: Maroon, 8:Black');
disp(' Striped Balls:');
disp('   9:Yellow, 10:Blue, 11:Red, 12:Purple, 13:Orange, 14:Green, 15:Maroon');
disp(' ');

inputDirectoryPathName = 'C:\Users\William\Documents\Research\DeepEnd\data\DenverRec';
inputFileName = 'TableA_movie1.MOV';
inputFullPathName = sprintf('%s\%s', inputDirectoryPathName, inputFileName);

% Read in movie file.
videoReader = VideoReader(inputFullPathName);

% Set output directory name "trainingImages" or "testImages".
outputDirectoryName = 'trainingImages';
if exist(outputDirectoryName, 'dir')==0
    fprintf('Hey!  You need to create the output directory %s!
', outputDirectoryName);
    pause
end
fprintf('Extract images to directory %s\n', outputDirectoryName);

% Set the starting time (if you don't set it, it starts at 0).
% A good way to figure out the approximate starting time is to look at your
% movie in Windows MovieMaker (assuming you have Windows).
minutes = 0.0;
seconds = 0.0;
t = 60*minutes + seconds;
videoReader.CurrentTime = t;    % time in seconds
while hasFrame(videoReader)  
    frame = readFrame(videoReader);  
    t = videoReader.CurrentTime;  
    figure(1), imshow(frame); 

    % Show the time on the displayed image.  
    text(20,20, sprintf('%.02f sec', t), 'BackgroundColor', 'w');  
    drawnow;

while true  
    s = input('Hit enter to continue to next image, or number of ball to crop: ', 's');  
    n = sscanf(s, '%d');  
    if isempty(s)  
        break;  
    else  
        C = imcrop;  
        % Let user crop the image  
        outputImageNumber = 0;  
        while true  
            % Write cropped image to a file.  If filename already exists,  
            % advance the image number to create a new filename.  
            outputImageNumber = outputImageNumber + 1;  
            fname = sprintf('%s\ball%02d_%04d.png', ...  
                            outputDirectoryName, n, outputImageNumber);  
            if ~exist(fname, 'file') break; end  
        end  
        fprintf('Saving image to %s...
', fname);  
        imwrite(C, fname);  
    end  
end  

    t = t + 1;  
    % Advance time by 1 second  
    if t > videoReader.Duration  
        break;  
    end  
end  

videoReader.CurrentTime = t;  

Program "extractBalls"  
(2 of 2)
Color Features

• Transform RGB color images to hue, saturation, value (HSV) space

  RGB  
  ![RGB Image]

  H  
  ![H Image]

  S  
  ![S Image]

  V  
  ![V Image]

• Extract only the pixels within a center circle

  RGB  
  ![RGB Image with Circle]

  H  
  ![H Image with Circle]

  S  
  ![S Image with Circle]

  V  
  ![V Image with Circle]
Histograms

- Create two 2D histograms:
  - Hue vs saturation
  - Saturation vs value

Then stack all histogram values into a single long column vector.
Extract Features

• See program “extractFeatures” on the next page
• Edit it to define the input directory (“trainingImages” or “testImages”)
• It will process all images starting with “ball” in that directory, and extract histogram features
• It will write out, to a “mat” file called “features.mat”
  – The feature vectors (‘featureVectors’)
  – The corresponding classes (‘featureClasses’)
  – The corresponding image filenames (‘imageFileNames’)
% Extract features from images of pool balls.
clear all
close all

% Number of bins in histograms.
% Histogram 1 is NHUExNSAT, histogram 2 is NSATxNVAL.
NHUE = 32;
NSAT = 8;
NVAL = 8;

% This is the number of dimensions of a feature vector.
Ndim = NHUE*NSAT + NSAT*NVAL;

% Define the input directory ('trainingImages' or 'testImages').
inputDirectory = 'trainingImages';
if ~exist(inputDirectory, 'dir')
    fprintf('Hey! can''t find directory named %s\n', inputDirectory);
    pause;
end

cd(inputDirectory);       % Go into the directory containing the images

% Process all png images in the folder.
folderInfo = dir('ball*.png');
Nimages = length(folderInfo);

% This will hold all the feature vectors. Each row is a feature vector.
featureVectors = zeros(Nimages, Ndim);

% This will hold all the feature class ids.
featureClasses = zeros(Nimages, 1);

% This will hold the corresponding image file names.
imageFileNames = cell(Nimages, 1);

for i=1:Nimages
    fileName = folderInfo(i).name;
    [vals,count] = sscanf(fileName, 'ball%d_%d.png');
    if count ~= 2
        fprintf('Hey! bad filename, should have form: ballXX_YYYY.png.\n');
        break;
    end

Program
“extractFeatures”
(1 of 3)
ballId = vals(1);
assert(ballId >= 0 && ballId <= 15);
imageNum = vals(2);

fprintf(' Processing image %s: ball %d, image %d
', ... 
    fileName, ballId, imageNum);

I = imread(fileName);
assert(size(I,3) == 3); % Should be an RGB image

% Convert RGB to HSV image.
HSV = rgb2hsv(I);
figure(3), imshow(HSV,[],), impixelinfo
H = HSV(:,:,1);
S = HSV(:,:,2);
V = HSV(:,:,3);

% Extract pixels in the center circle.
[h,w,~] = size(HSV);

x0 = round(w/2);
y0 = round(h/2);
r = min(x0,y0); % Radius of circle

% Get the x,y coordinates of all pixels inside this circle.
xMin = max(x0-r, 1);
xMax = min(x0+r, size(HSV,2));
yMin = max(y0-r, 1);
yMax = min(y0+r, size(HSV,1));
[Xi,Yi] = meshgrid(xMin:xMax, yMin:yMax);
R = ((Xi-x0).^2 + (Yi-y0).^2) .^ 0.5;

% Flag those that within radius r from the center.
Rinside = (R < r);
% Get (x,y) coordinates of the inside pixels.
Xinside = Xi(Rinside);
Yinside = Yi(Rinside);

% Get equivalent linear indices.
indices = sub2ind([h,w], Yinside, Xinside);

% Get HSV values of all pixels inside circle.
Vi = [H(indices), S(indices), V(indices)];

% Create a 2D histogram of hue vs saturation.
[h1,~,~] = histcounts2(Vi(:,1), Vi(:,2), 0:(1/NHUE):1, 0:(1/NSAT):1);
h1 = h1/sum(h1(:));  % Normalize counts
figure(1), imshow(h1,[], 'InitialMagnification', 800), title(sprintf('Ball %d', ballId));
%figure(1), histogram2(Vi(:,1), Vi(:,2), 0:(1/NHUE):1, 0:(1/NSAT):1), xlabel('H'), ylabel('S');

% Create a 2D histogram of saturation vs value.
[h2,~,~] = histcounts2(Vi(:,2), Vi(:,3), 0:(1/NSAT):1, 0:(1/NVAL):1);
h2 = h2/sum(h2(:));  % Normalize counts
figure(2), imshow(h2,[], 'InitialMagnification', 800), title(sprintf('Ball %d', ballId));
%figure(2), histogram2(Vi(:,2), Vi(:,3), 0:(1/NSAT):1, 0:(1/NVAL):1), xlabel('S'), ylabel('V');

% Assemble counts into a single long vector.
X = [h1(:); h2(:)];

% Store into the arrays.
featureVectors(i,:) = X';
featureClasses(i) = ballId;
imageFileNames{i} = fileName;

pause(0.25);
end

% Write out data to a "mat" file.
fprintf(' Writing out features to features.mat\n');
save('features', 'featureVectors', 'featureClasses', 'imageFileNames');

cd('..');  % Go back up to original directory
Classification

• Finally, train an SVM on the training data
• Use it to classify the test data
• See program “classifyBallsSVD” on next page

• This program reads in the feature vectors in the directory “trainingImages”
• It trains an SVM and applies it to the feature vectors in the directory “testImages”
clear all
close all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Get training data.
inputDirectory = 'trainingImages'; % Directory for training data
if ~exist(inputDirectory, 'dir')
    fprintf('Hey! can''t find directory named %s\n', inputDirectory);
    pause;
end
cd(inputDirectory); % Go into the directory containing the images

% Load feature vectors and classes from a "mat" file.
% This should load in 'featureVectors', 'featureClasses', 'imageFileNames'.
fprintf('Reading training features from features.mat\n');
load('features');

classes = unique(featureClasses);
disp('Feature classes present in training data: '), disp(classes);

cd('..'); % Go back up to original directory
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Train the SVM Classifier.
c1 = fitcecoc(featureVectors, featureClasses, ...
              'Verbose', 2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Program “classifyBallsSVD” (2 of 2)
Confusion Matrix

• A table to show the performance of a classifier; ie, the number of times a sample of one class was identified as a type of some other class
  – The name comes from the fact that it makes it easy to see if the system is confusing two classes (i.e. commonly mislabeling one as another)

• Example

<table>
<thead>
<tr>
<th>Actual class</th>
<th>Predicted</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cat</td>
<td>Dog</td>
</tr>
<tr>
<td>Cat</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Dog</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Rabbit</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

• Columns correspond to the predicted classes (the output of the classifier)
• Rows correspond to the actual (true) classes

From https://en.wikipedia.org/wiki/Confusion_matrix
Understanding SVM Features

• It is possible to understand the importance that an SVM assigns to particular feature dimensions

• Recall that a (two class) linear SVM uses a weight vector $\mathbf{w}$ to classify an unknown vector $\mathbf{x}$, using

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x} + b)$$

• The vector $\mathbf{w}$ is the same size as the input vectors $\mathbf{x}$

• A high magnitude for the $i^{th}$ element of $\mathbf{w}$ means that the $i^{th}$ dimension is important for determining the class
  • If the magnitude of $w_i$ is small, then this dimension is irrelevant to determining the class