

# IVR Coursework 1

## Report

Weiting Goh (S1450710) and Tomas Markevicius(S1452595)

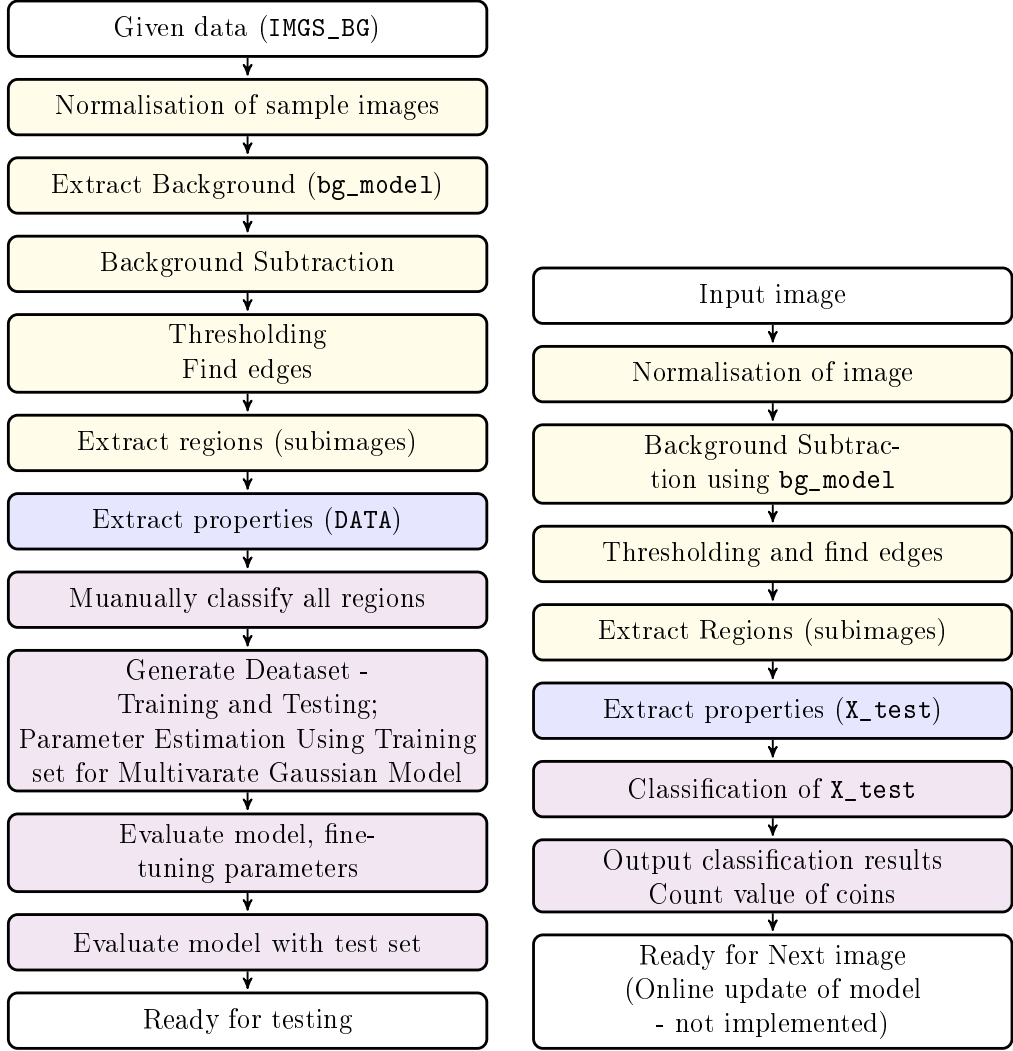
October 27, 2016

## 1 Introduction

To recognise and count the coins in an image, Coinsy, have three subtasks: 1) Image processing (includes image segmentation), 2) Features extraction, and 3) Classification (and then counting the coins).

For Coinsy to be a proficient detector, classifier and counter, we have to first train it. The training processes differs from evaluation as described in figure Figure 1 - the operation pipeline for training and evaluation. We will describe the methods that we use for each of these subtasks in the next section - methodology, and her evaluation results after. Lastly, we conclude with a discussion on Coinsy performance. The code for the project is listed in the appendix.

In the following subsections, we give an brief description of the operation pipeline for each subtasks from training to testing. The approach here is an abstract idea of what we did for our base model. Additional techniques were explored and will be discussed in later section. We start by understanding the data.



(a) Operation pipeline to train Coinsy (b) Pipeline for (trained) Coinsy to count coins

Figure 1: Differences in Pipeline; Yellow boxes indicates the image processing procedures; Blue indicates the feature extraction procedures; Violet indicates the classification procedures.

## Data

The data we were given are 14 images (consisting of 5 **harder** and 9 **simpler images**). Each consists of objects in the foreground for us to classify and calculate the values. The class labels and its respective object and value are

as follow:

Class	Object	Value
1	1 Pound coin	1 pound
2	2 Pound coin	2 pound
3	50 Pence coin	50 pence
4	20 Pence coin	20 pence
5	5 Pence coin	5 pence
6	Washer with small hole	75 pence
7	Washer with large hole	25 pence
8	Angle bracket	2 pence
9	AAA Battery	-
10	Nut	-
11	unknown	-

The objects differ in colors, size and shapes; some are very similar to the background - such as 1 pound coins (see Figure 2). Hence, the features extracted must be invariant to rotation, and importantly to detect the objects will the images to be processed, such that the objects are salient to the computer vision. On top of the images itself output from `imshow` or `imagesc`, we can understand an image from the distribution of the pixel intensities (such as a histogram) and its gradient magnitude in each channel.



Figure 2: Sample images from given data.



Figure 3: Sample images after their background is subtracted.

## Image Processing

The image processing step aims to 1) make all images (for training the model or for evaluation) comparable, 2) extract objects in the foreground (also known as image segmentation). The outcome of this stage is an array of subimages ready for feature extraction.

The central idea is to, first, model the background before subtracting it from all images; second - make the edges of the objects 'obvious' by finding a suitable threshold to binarise the image; third - crop out the objects to obtain the subimages, which will be our data points.

## Feature Extraction

Given the subimages, this stage aims to represent each subimage with a feature vector that contains properties to adequately describe the class it belongs to (or shape). Notably, we have many circular objects varying in size. This renders global descriptors such as convexity and elongation less useful for these classes.

## Classification

The task here is to label an unknown (sub)image given its set of feature vector. The model we consider is a multivariate gaussian classifier that classifies an image based on the parameters of each class (class mean and standard

deviation (or variance)). This multivariate gaussian classifier outputs the posterior probability of a given feature vector  $x$  ( $P(C_k|X)$ ) for each class  $k$ , with the class giving the highest probability being the label. That is  $x.class = \operatorname{argmax}_k P(C_k|x)$ .

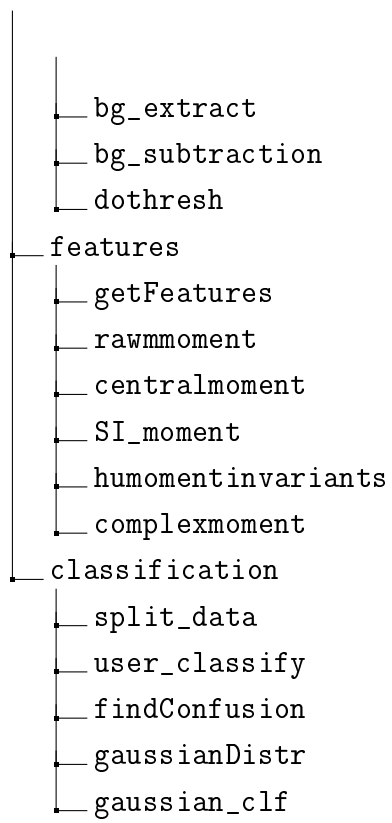
A slight tweak for Coinsy is her ability to reject the class label output by the classifier if the highest probability falls out of her confidence interval. In such cases, Coinsy will intervene and change the class to unknown (class 11).

The last step of Coinsy is to sum the value of all the objects she managed to identify from a given image.

## Code

The following directory trees will provide an overview of the code utilised for the project. Codes presented in the appendix are hyperlinked, although some may depend on the code repository given in <http://www.inf.ed.ac.uk/teaching/courses/ivr/matlab/flatpartrecog/>.

```
src
├── imgs ..... Store images from function
├── dataset ..... Datasets from experiments
├── setup
├── training
├── main
├── extract_features
├── manual_classification
├── trainclf_loglikelihood
├── filters
│   ├── median_filter
│   ├── median_filter_iter
│   ├── gaussian_filter_1d
│   └── gaussian_filter_2d
├── image_processing
│   └── normalise_RGB
```



## 2 Methodology

In this section we describe the techniques we considered and brainstormed during the project and those that are implemented.

Task	Subtask	Techniques	Implmentation
Image Pro- cessing	Background Model	Finding median for each pixel and each channel	✓
		Finding median of neighbourhood for each pixel and each channel	✓
		Probabilistic Modeling of background	
	Thresholding	Finding minimum of bimodal distribution	✓
		Otsu's method for thresholding	
		Using Gradient magnitude of image	
Feature Ex- traction	Global De- scriptors	<ul style="list-style-type: none"><li>• Area</li><li>• Perimeter</li><li>• Compactness</li><li>• Rectangularity</li><li>• Elongation</li></ul>	✓
	Moments	<ul style="list-style-type: none"><li>• Hu's Invariant moments (7 features)</li><li>• Complex moments (6 features)</li></ul>	✓
Classification	-	Multivariate Gaussian Model Linear Discriminant	✓

## Background Modelling

In this coursework, since a background image is not readily available, we have to model it. Noting that images varies in illumination, we have to make the images comparable by normalising it first, using the following formula.

$$P_{r,c}(R', G', B') = \left( \frac{R}{\sqrt{R^2 + G^2 + B^2}}, \frac{G}{\sqrt{R^2 + G^2 + B^2}}, \frac{B}{\sqrt{R^2 + G^2 + B^2}} \right)$$

With objects scattered around randomly in the images, we find the median of all image pixels for each channel separately in order to reconstruct the background.

The outcome of the background with and without normalisation is shown in Figure 4.

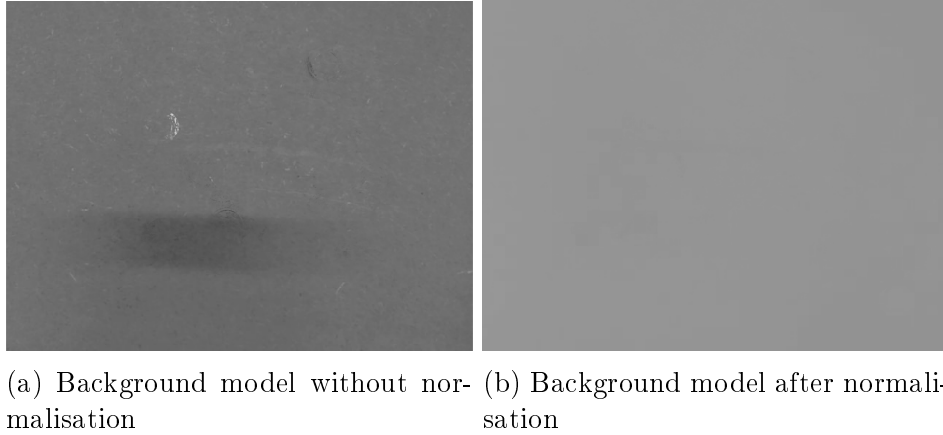


Figure 4: Background model generated from all 14 images

The sample images with their background removed is shown in Figure 3. It is evident that the background removal process removed the background - making the images appearing black. However, it also inevitably reduce the intensity for the bottom half of each images, such that the objects are no longer salient to our eyes. This is because the background we modelled have a lower intensity at the bottom, possibly due to presence of shadow in all 14 images.



Nevertheless, the histogram is still bimodal, which is essential for thresholding to be effective.

## **Segmentation**

### **2.1 Classification**

### 3 Result

documentclass[main.tex]article

## A Scripts

In this section, all the scripts used to call other scripts and/or functions are presented. Some functions are matlab's native functions.

### **training.m**

```
1 %% MASTER SCRIPT USE FOR TRAINING
2 % Steps:
3 %   1.  setup
4 %       a. load images and IMGS_BG (for bg modelling), IMGS (
5 %           all simpler images)
6 %   2.  image_processing
7 %       a.  normalisation          (normalise_rgb)
8 %       b.  background model      (bg_extract)
9 %       c.  background subtraction (bg_subtraction)
10 %      d.  thresholding           (dothresh)
11 %
12 %   3.  extract_features
13 %       a.  regionprops
14 %       b.  getFeatures
15 %           — rawmoment,
16 %           — centralmoment,
17 %           — complexmoment,
18 %           — SI_momment,
19 %           — humomentinvariant
20 %
21 %   4.  Classification
22 %       a.  manual_classifcation
23 %           — user_classify
24 %       b.  trainclf_loglikelihood
25 %           — split_data
```

```

26 %           — gaussianDistr, gaussian_clf, logdet
27 %           — findConfusion
28 %%
29 clear all; close all; clc;
30 setup % import images
31 % START TRAINING:
32 image_processing;
33 extract_features; % OUTPUT DATA!
34
35 %% Init Manual Classification
36 man_class = input('do you want to manually classify these
    images now? [0/1]');
37 if man_class
38     manual_classification;
39 end
40
41 %%

```

#### setup.m

```

1 %% START CODE:
2 clc,clf,clear all; close all;
3
4 % add all relevant folders && misc stuff
5 addpath('filters/', 'image_processing/', 'classification/', ...
6         'dataset/', 'imgs/', 'features');
7 addpath(' ../misc/export_fig.package/');
8
9 bar = '
    =====';
10 barbar = '
    _____';
11

```

```

12 disp(bar);
13 fprintf('\t\tIMPORTING IMAGES\n');
14 % add all given images for training
15 % ? SHOULD we add the harder ones too?
16 img2 = imread('../practice/simpler/02.jpg');
17 img3 = imread('../practice/simpler/03.jpg');
18 img4 = imread('../practice/simpler/04.jpg');
19 img5 = imread('../practice/simpler/05.jpg');
20 img6 = imread('../practice/simpler/06.jpg');
21 img7 = imread('../practice/simpler/07.jpg');
22 img8 = imread('../practice/simpler/08.jpg');
23 img9 = imread('../practice/simpler/09.jpg');
24 img10 = imread('../practice/simpler/10.jpg');
25 IMGS = {img2, img3, img4, img5, img6, img7, img8, img9,
        img10};
26
27 img11 = imread('../practice/harder/17.jpg');
28 img12 = imread('../practice/harder/18.jpg');
29 img13 = imread('../practice/harder/19.jpg');
30 img14 = imread('../practice/harder/20.jpg');
31 img15 = imread('../practice/harder/21.jpg');
32
33 IMGS_BG = {img2, img3, img4, img5, img6, img7, img8, img9,
            img10, ...
            img11, img12, img13, img14, img15 };
34
35 % IMGS = {img2, img3, img4, img5, img6, img7, img8, img9,
          img10, ...
          img11, img12, img13, img14, img15 };
36
37 [~, num_img_bg] = size(IMGS_BG);
38 fprintf('\t\t\t\t\t done\n');
39 %%
40 tmp = input('Continue? [1/0] ');
41 if ~tmp
42     return

```

```
43 end
44 disp(bar);
```

#### extract\_features.m

```
1 %% Script for feature extraction
2 %
3 % This script follows naturally from the segmentation script
  where the
4 % images are segmented and edges are found. The next step in
  the operation
5 % pipeline is then to find the obojects in the picture, then
  extract
6 % the features from the objects
7 %
8 % assume you have done called segmentation and the following
  are in
9 % the workspace:
10 % 1) bg_model      : the background model we generated
11 % 2) IMGS          : the original images (in cell array)
12 % 3) IMG_BGREMOVE  : the original iamges bg removed
13 % 4) IMG_THRESH    : the BW images thresholded. The objects
  are in white/1
14
15 %%
16 [~, num_imgs] = size(IMGS_THRESH);
17 PROP = {}; % define an array to hold the structs for each images
18 DATA = struct(); % struct to hold all the subimages
19 num_instance = 1; % counter for number of instances
20
21 % iterate through all the images to extract the subimages and
  its properties
22 for i=1:num_imgs
```

```

23
24 fprintf('image %d ',i);
25
26 % here, get the label from the threshold image, and extract
    information
27 % about each region
28 [L, ~] = bwlabel(IMGs_THRESH{i}, 4); %% THIS IS A
    PARAMETER TO PLAY WITH
29 imagery = regionprops(L, 'BoundingBox','Image'); % this
    is the BW image!
30 scalar = regionprops(L, 'MajorAxisLength', '
    MinorAxisLength', 'Area');
31
32 % remove regions with small pixel area, which may be blobs:
33 bad = [scalar.Area] <= 300;
34 scalar(bad) = []; % remove these instances
35 imagery(bad) = [];
36 disp('prune - Area<=300'); %% DEBUG
37
38 [num_subimages , ~] = size(imagery); % update the number of
    instances left!
39
40 % grab the colored subimages, and calculate the complex
    moments,..etc,
41 % for ease of classification:
42 for n=1:num_subimages
43
44     org_img = IMGs{i}; % get the original image
45     boundary = imagery(n).BoundingBox; % find the
        boundary
46     subImg = imcrop(org_img, boundary); % crop the
        original image according to boundary
47

```



```

48         % calculate the moments by calling classification/
           getProperties
49         DATA(num_instance).Features          = getFeatures(
           imagery(n), scalar(n));
50         DATA(num_instance).ColoredImage      = subImg;
51         DATA(num_instance).BoundingBox       = imagery(n).
           BoundingBox;
52         DATA(num_instance).Image             = imagery(n).Image;
53         DATA(num_instance).MajorAxisLength   = scalar(n).
           MajorAxisLength;
54         DATA(num_instance).MinorAxisLength   = scalar(n).
           MinorAxisLength;
55         DATA(num_instance).ParentID          = i;
56         DATA(num_instance).Class             = 0; % set to 0 =
           unclassified
57
58         fprintf('%d ', num_instance);
59         num_instance = num_instance + 1;
60     end
61
62     % store in struct
63     PROP{i} = struct('label', L, ...
64                     'num_of_obj', num_subimages, ...
65                     'ORIGINAL', IMGS{i}, ...
66                     'THRESH', IMGS_THRESH{i}, ...
67                     'SubImages', imagery, ...
68                     'Properties', scalar);
69
70     fprintf('\t\tDone\n');
71 end
72
73 % clear boundary;
74 % clear imagery;
75 % clear scalar;

```

76 %%

### setup.m

```
1 %% START CODE:
2 clc,clf,clear all; close all;
3
4 % add all relevant folders && misc stuff
5 addpath('filters/', 'image_processing/', 'classification/', ...
6         'dataset/', 'imgs/', 'features');
7 addpath('../misc/export_fig.package/');
8
9 bar = '
10      =====';
11 barbar = '
12      _____';
13
14 disp(bar);
15 fprintf('\t\tIMPORTING IMAGES\n');
16 % add all given images for training
17 % ? SHOULD we add the harder ones too?
18 img2 = imread('../practice/simpler/02.jpg');
19 img3 = imread('../practice/simpler/03.jpg');
20 img4 = imread('../practice/simpler/04.jpg');
21 img5 = imread('../practice/simpler/05.jpg');
22 img6 = imread('../practice/simpler/06.jpg');
23 img7 = imread('../practice/simpler/07.jpg');
24 img8 = imread('../practice/simpler/08.jpg');
25 img9 = imread('../practice/simpler/09.jpg');
26 img10 = imread('../practice/simpler/10.jpg');
27 IMGs = {img2, img3, img4, img5, img6, img7, img8, img9,
28         img10};
```



```

11         0.72078433, 0.84784315, 0.30039217;
12         0.96988851, 0.85064207, 0.19683199;
13         0.93882353, 0.80156864, 0.4219608 ;
14         0.83652442, 0.74771243, 0.61853136;
15         0.7019608 , 0.7019608 , 0.7019608
16         244/255, 66/255, 66/255]; % Class 11
17 total_instance = 0;
18 total_relevant = 0;
19 t = datetime('now'); % for image title
20
21
22 %%
23 [~, num_instance] = size(DATA);
24 for i=1:num_instance % for each datapoint:
25
26     img_num      = DATA(i).ParentID;
27     img_BIG      = PROP{img_num}.ORIGINAL; % original big image
28     subimg       = DATA(i).ColoredImage;
29     bw_subimg    = DATA(i).Image;
30
31     fprintf('\n\n\nObject %d/%d\n', i , num_instance);
32     close all; figure; % Close all opened windows
33
34     % Plot the images
35     subplot(1,2,1);
36     imshow(subimg);
37     subplot(1,2,2);
38     imshow(bw_subimg);
39
40     % call function to for classification
41     [relevance, class] = user_classify();
42     close all;
43     % if user need help, display the bigger image with a
       bounding box for object:

```

```

44     while class == 0
45         fig = figure;
46         imshow(img_BIG);
47         hold on;
48         rectangle('Position', DATA(i).BoundingBox,... % draw
                    rectangle around img
49                 'EdgeColor', 'r', 'LineWidth',3);
50         [relevance, class] = user_classify();
51         close all;
52     end
53
54     DATA(i).Class = class; % store the class; irrelevant ones
                    at 11
55
56     % SAVE THE IMAGE
57     imshow(subimg);
58     s = sprintf('./imgs/CLASS_%d/%s_%d.png', class, t,
                    num_instance);
59     export_fig(s);
60     close;
61
62 end
63
64 %% DISPLAY and drawings
65 close all; figure;
66 imshow(img_BIG);
67 titl = sprintf('Classification for picture %d (%s)',i,t);
68 title(titl);
69 hold on;
70 [~,num_imgs] = size(PROP); % num of images
71
72 ID = [DATA.ParentID];
73 for i=1:num_imgs % draw the boundary box with differernt color
                    for each image

```

```

74     close all;
75
76     list_ = ID == i; % logical
77     data_class = DATA(list_);
78     img_BIG = PROP{i}.ORIGINAL;
79     img_BW = PROP{i}.THRESH;
80     imshow(img_BW); hold on;
81     for n=1:sum(list_) % draw the boundary on BW image
82         boundary = data_class(n).BoundingBox;
83         class = data_class(n).Class;
84         % disp(cmap(class,:));
85         rectangle('Position', boundary, 'EdgeColor', cmap(class
            ,:), 'LineWidth', 2);
86         s = sprintf('./imgs/manual_classy/manual_clas_pic#%d_BW
            .(%s).png',i,t);
87         export_fig(s);
88     end
89
90     close all; % repeat for colored images
91     imshow(img_BIG);
92     for n=1:sum(list_) % draw the boundary on BW image
93         boundary = data_class(n).BoundingBox;
94         class = data_class(n).Class;
95         % disp(cmap(class,:));
96         rectangle('Position', boundary, 'EdgeColor', cmap(class
            ,:), 'LineWidth', 2);
97         s = sprintf('./imgs/manual_classy/manual_clas_pic#%d_BW
            .(%s).png',i,t);
98         export_fig(s);
99     end
100 end
101
102
103 %% Delete Class 11 instances

```

```

104 class_list = [DATA.Class];
105 logica_     = [class_list == 11];
106 DATA(logica_) = [];
107 [~,init_size] = size(class_list);
108 [~,after_size] = size(DATA);
109 fprintf('Number of datapoints removed (class 11) = %d\n',
        init_size - after_size);

```

#### trainclf\_loglikelihood.m

```

1  %% SCRIPT FOR TRAINING MULTIVARATE GAUSSIAN CLASSIFIER
2  %   Assume you have done extract_features and
   manual_classification
3  %   PROP must be in your workspace
4
5  %% COMPACT ALL YOUR DATA:
6  [~, num_instance] = size(DATA);
7  [~, num_feature] = size(DATA(1).Features);
8
9  num_data = 0;
10 X = []; % Feature
11 y = []; % classes
12
13
14 % first, put all images together matrix
15 for im=1:num_instance
16     X = [X; DATA(im).Features];
17     y = [y; DATA(im).Class];
18 end
19 % y = reshape(y, [],1); % convert into col vector
20
21

```

```

22 %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
23 %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
24 % CLASS 1 is missing (NO POUND COIN DETECTED!)
25 % CREATE BOGUS DATA:
26 for w=1:4
27     y(num_instance+w) = 1;
28     X(num_instance+w,:) = [rand(1,num_feature)]; % randomly
        give some data!
29 end
30 %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
31 %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
32
33 %% Do hold-out validation:
34 % 50% for training, 25% for validation 25% for test
35 [X_train, X_valid, X_test, y_train, y_valid, y_test] = ...
36     split_data(X, y, .5, 0, .5);
37
38 %% TRAINING THE CLASSIFIER:
39 % GROUP IN CLASS AND PARAMETER ESTIMATION:
40 classes      = unique(y);
41 num_class     = length(classes);
42 num_instance  = length(X);
43
44 % Sort data into struct:
45 DATA_CLASS = {};
46

```



```

47 for i = 1:num_class % create a DATA_CLASS for each class
48 %     disp(i); %% DEBUG
49     logica_      = [y_train == classes(i)];
50     prior_       = sum(logica_)/num_instance;
51     data         = X_train(logica_, :);
52     mean_        = mean(data,1); % take the mean along the cols
53     cov_         = cov(data,0); % number of observations -1;
                    % Maximmmum posterior
54
55     % Regularise COV:
56     reg = exp(-10);
57     reg_term = eye(length(cov_)) * reg;
58     cov_ = cov_ + reg_term; % add regularisation
59     disp(cov_);
60
61     % store the parameters
62     DATA_CLASS{i} = struct('Data', data, 'Prior', prior_, ...
63                             'Mean', mean_, 'Cov', cov_);
64 end
65
66 %% VALIDATION DATASET:
67 % [y_vali_pred, ~] = gaussian_clf(X_valid, DATA_CLASS);
68 %
69 % % Generate Statistics:
70 % [cm_valid, per] = findConfusion(y_vali_pred, y_valid);
71 % imshow(cm_valid, [], 'InitialMagnification', 1600); colormap(
    bone);
72 % title('Confusion Matrix for Validation Set');
73
74 %% Testing
75 p_limit = 0;
76 [y_test_pred,~] = gaussian_clf(X_test, DATA_CLASS, p_limit);
77
78 % Generate Statistics:

```

```

79 [cm_test, per] = findConfusion(y_test_pred, y_test, 11, p_limit
    );
80 %%

```

#### main.m

```

1  %% This is the main code for the assignment:
2  clc;clear;
3  start = 1;
4  bar = '
    =====';
5  barbar = '
    _____';
6
7  while start
8  % Part 1: Reading the image, query from the user
9
10     disp(bar); disp(barbar);
11     fprintf('This is the coinsy counter!\nYour current work
        directory is: \n\t');
12     disp(pwd); disp(barbar);
13     fprintf('To END: enter cltr + c\n');
14     prompt_start = 'To START: enter your image file (rel/abs
        dir) below:\n';
15
16     filename = input(prompt_start, 's');
17     if isempty(filename)
18         disp('Using trial image: practice/simpler/05.jpg');
19         filename = '../practice/simpler/05.jpg';
20     end
21
22 % load the image into original_image
23 original_image = imread(filename);

```

```

24
25     disp(barbar); disp(bar); fprintf('\n\n')
26
27 %% Part 2: Image segmentation.... ?
28
29     disp(bar); disp(barbar);
30     disp('NOW: Segmenting the images...');
31
32     disp(barbar); disp(bar); fprintf('\n\n');
33
34 %% Part 3: Classification ...?
35
36     disp(bar); disp(barbar);
37     disp('NOW: Classifying the objects...');
38
39     disp(barbar); disp(bar); fprintf('\n\n');
40
41 %% Part 4: Coinsy Counter:
42
43     disp(bar); disp(barbar);
44     disp('NOW: Initialising the counter...');
45     % counter starts at 0
46     counter = 0;
47
48
49     disp(barbar); disp(bar); fprintf('\n\n');
50
51 %% Part 5: Summary Statistics:
52     disp(bar); disp(barbar);
53     disp('SUMMARY STATISTICS');
54
55 % Expect something like:
56 % two_pound =
57 % one_pound =

```

```

58 % sevenfive_pence =
59 % fifty_pence =
60 % twofive_pence =
61 % twenty_pence =
62 % five_pence =
63 % two_pence =
64 % battery =
65 % nut =
66 % unclass =
67 % Total value =
68 % Confidence =
69
70     disp(barbar); disp(bar); fprintf('\n\n');
71
72 %% Next image?
73     % single loop for now:
74     prompt_end = ('Do you want to load another image? [y/n]');
75     x = input(prompt_end, 's');
76     switch x
77         case 'y'
78             start = 1;
79         case 'n'
80             start = 0;
81         case 'Y'
82             start = 1;
83         case 'N'
84             start = 0;
85         otherwise
86             start = 1;
87     end
88 end

```

## B Image Processing

normalise\_RGB.m

```
1 function [img_out, gray_out] = normalise_RGB(RGB, SHOW)
2 %% NORMALISE_INPUT_RGB(RGB, SHOW)
3 %   Normalise the RGB values for each pixel in the image RGB
4 %   Also, output the gray normalised output of RGB (i.e.
5 %   normalised RGB +
6 %   rgb2gray());
7 %   The algorithm for normalisation is the root sum of channels
8 %   squared.
9
10 %%
11 RGB = double(RGB); % cast into double
12 RED_Channel      = RGB(:,:,1);
13 GREEN_channel    = RGB(:,:,2);
14 BLUE_channel     = RGB(:,:,3);
15
16 [row,col,chn] = size(RGB);
17 img_out = zeros(row,col,chn);
18
19 for i = 1:row
20     for j = 1:col
21         r = RED_Channel(i,j);
22         g = GREEN_channel(i,j);
23         b = BLUE_channel(i,j);
24
25         sum_sq = sqrt(r^2 + g^2 + b^2);
26 %         sum_sq = r + g + b;
27
28         img_out(i,j,1) = r/sum_sq;
29         img_out(i,j,2) = g/sum_sq;
30         img_out(i,j,3) = b/sum_sq;
```

```

29
30     end
31 end
32
33 % CAST IT BACK TO INT!
34 RGB = uint8(RGB);
35 img_out = uint8(img_out*255); % IMPORTANT TO MULTIPLY BY 255!!!
36
37 %% GRAY OUT STRATEGY:
38 %   simple for now..!
39 gray_out = rgb2gray(img_out);
40
41
42 %% DISPLAY RESULT:
43 if SHOW
44     display_stats(RGB, img_out);
45     figure;
46     display_stats(rgb2gray(RGB), gray_out);
47 end
48
49 end

```

#### bg\_extract.m

```

1 function [ bg_model ] = bg_extract( IMGS, WINDOW_SIZE )
2 %% BACKGROUND_MODEL(IMG, WINDOW_SIZE
3 %   Given a series of image, we find the common background
   using median
4 %   filtering. For each pixel in the bg_model, we take the
   median of all
5 %   the pixels in the WINDOW_SIZE for all the images. If
   WINDOW_SIZE = 1,

```

```

6 % it is equivalent to taking the median of pixel intensity of
  all the
7 % images.
8 % If input image is RGB, then this is carried out for all
  channel.
9
10 % INPUT:
11 % - IMGS : A cell array of images. Images must be of the same
  size. IMGS
12 % have size of (1,num_imgs)
13 % - WINDOW_SIZE : the window of median_filter.
14 %     If undefined, WINDOW_SIZE = 1
15
16 % OUTPUT:
17 % - bg_model - an image of the same size as IMGS with the
  background
18 % extracted.
19
20 %% Setting parameters
21 if nargin == 1
22     WINDOW_SIZE = 1;
23 end
24
25 [~, num_imgs] = size(IMGS);
26 sample        = IMGS{1};
27 bg_model       = sample; % preallocation of memory
28
29 % Given a WINDOW_SIZE, find the number of cell to compensate:
30 % Window_Size    1 3 5 7 9...
31 % offset         ==  1 2 3 4 5
32 % ==> offset = (WS + 1) / 2
33
34 % !! prevent even number WINDOW_SIZE
35 if mod(WINDOW_SIZE, 2) ~= 1

```

```

36     error('Window_size must be an odd number!');
37 end
38
39 offset = uint64((WINDOW_SIZE + 1)/2);
40
41 % if offset == 1
42 %     offset = 0; % no need to offset if Window_size = 1
43 % end
44
45 disp('Extracting background from images....');
46 fprintf('\tWINDOW_SIZE = %d\n', WINDOW_SIZE);
47 fprintf('\tOffset = %d\n', uint8(offset));
48
49
50 %% iterate through all the images and set the
51 if ndims(sample) == 3
52
53     [rows, cols, ~] = size(sample);
54
55     % iterate each cell, neglecting offset cause of WINDOW_SIZE
56     for i = offset:rows-offset+1
57         for j = offset:cols-offset+1
58             %             disp([i,j]); %DEBUG
59
60             % store all the values from each img in IMGS
61             %             median_RED      = zeros(1, num_imgs);
62             %             median_GREEN   = zeros(1, num_imgs);
63             %             median_BLUE    = zeros(1, num_imgs);
64             median_RGB = zeros(num_imgs,1,3);
65
66             % find bounding box of pixels
67             x_low  = i - offset + 1;
68             x_high = i + offset - 1;
69             y_low  = j - offset + 1;

```



```

70         y_high = j + offset - 1;
71     %         disp([x_low,x_high,y_low,y_high]); % DEBUG
72
73     % iterate through all the pixels for the image
74     for k=1:num_imgs
75         temp = IMGS{k};
76         segment = temp(x_low:x_high, y_low:y_high, :);
77         med = median(median(segment)); % median along
            the color axis
78         median_RGB(k,1,:) = med;
79     %         % get the pixels belonging in the image:
80     %         pixels_RED      = IMGS{k}(x_low:x_high, y_low
: y_high, 1);
81     %         pixels_RED      = reshape(pixels_RED, [], 1);
82     %         median_RED(k)   = median(pixels_RED); % get
the median of the nieghbood!
83     %
84     %         pixels_GREEN    = IMGS{k}(x_low:x_high, y_low
: y_high, 2);
85     %         pixels_GREEN    = reshape(pixels_GREEN, [],
1);
86     %         median_GREEN(k) = median(pixels_GREEN); % get
the median of the nieghbood!
87     %
88     %         pixels_BLUE     = IMGS{k}(x_low:x_high, y_low
: y_high, 3);
89     %         pixels_BLUE     = reshape(pixels_BLUE, [], 1)
;
90     %         median_BLUE(k)  = median(pixels_BLUE); % get
the median of the nieghbood!
91     end
92
93     % Set the meidan for respecitve color channel to
the bg_model

```

```

94 %         bg_model(i,j,1) = median(median_RED);
95 %         bg_model(i,j,2) = median(median_GREEN);
96 %         bg_model(i,j,3) = median(median_BLUE);
97         bg_model(i,j,:) = median(median_RGB);
98     end
99     fprintf(' ');
100 end
101
102 else
103 %% 2D images:
104     [rows, cols] = size(sample);
105
106     % iterate each cell
107     for i = offset : rows-offset
108         for j = offset : cols-offset
109
110             median_val = zeros(1,num_imgs);
111
112             % finding bounding box:
113             x_low  = i - offset + 1;
114             x_high = i + offset - 1;
115             y_low  = j - offset + 1;
116             y_high = j + offset - 1;
117
118             % iterate through all the pixels for the image
119             for k=1:num_imgs
120                 % find bounding box of pixels
121                 pixels = IMGs{k}(x_low:x_high, y_low:y_high);
122                 pixels = reshape(pixels, [], 1);
123                 median_val(k) = median(pixels); % get the
                    median of the nieghbood!
124             end
125             % Set the meidan for respecitve color channel to
                    the bg_model

```

```

126         bg_model(i,j) = median(median_val);
127
128     end
129     fprintf('.');
130 end
131
132
133 end
134
135 bg_model = uint8(bg_model); % cast back to int
136 % imshow(bg_model); % DEBUG
137 disp('done');
138
139 end

```

#### bg\_subtraction.m

```

1 function [img_bgremove, bg_model] = bg_subtraction(img,
2         bg_model)
3 %% BG_SUBTRACTION(IMG, BG_MODEL)
4 %   returns a new image after subtracting it with the bg_model
5 %   Given a cell array of img, bg_model models after these img
6 %   and return a
7 %   cell array of img with their background removed using the
8 %   inferred
9 %   bg_model
10
11 %   INPUT:
12 %   — IMG:  a cell array or just an image. If it is just an
13 %   image, a
14 %   bg_model must be given
15 %   — bg_model : optional if you want the algorithm to infer
16 %   the bg model

```

```

12 %   from the img. in this case, img must be a cell array of
    image
13 %   N.B, if bg_model is given and img is cell, no bg_model is
    inferred, and
14 %   this will be just a simple straightforward bg subtraction
    algorithm.
15
16 %   OUTPUT:
17 %   — new_img : a cell array of image if input is cell array
18 %   — bg_model : if bg_model is inferred, otherwise just the
    bg_model
19
20 %%
21 % No bg_model given; img is cell array of images
22 if iscell(img)
23
24     [~, num_img] = size(img);
25     img_bgremove = img; % memory allocation
26
27     switch nargin
28     case 1
29         disp('extracting bg_model from cell array of images
30             ');
31         bg_model = bg_extract(img);
32
33         % carry out subtraction:
34         for i=1:num_img
35             img_bgremove{i} = abs(img{i} - bg_model); %
36                 take abs, avoid negative
37         end
38
39     case 2
40         disp('subtracting all images with given bg_model')
41         % carry out subtraction:

```

```

40         for i=1:num_img
41             img_bgremove{i} = abs(img{i} - bg_model);
42         end
43     end
44 else
45     % subtract bg from img;
46     img_bgremove = abs(img - bg_model);
47 end
48
49 disp('done');
50
51 end

```

#### dothresh.m

```

1 function [img_thresh, thresh_vals] = dothresh(IMGS, sizeparam)
2 %% DOTHRESH(IMGS, SIZEPARAM)
3 %   Function that find the threshold for an image then apply
4 %   thresholding
5 %   to get a binary image.
6 %
7 %   INPUT:
8 %   - IMGS : a cell array of images or just an image of
9 %   interest
10 %   - sizeparam : thte
11 %
12 %   OUTPUT:
13 %   - thresh_imgs : if IMGS is a cell array of images, so it
14 %   thresh_imgs.
15 %   the images are thresholded with its corresponding
16 %   threshold in
17 %   thresh_vals

```

```

14 %   – thresh_vals : if the image is RGB, then thresh_vals is a
    %       vvector of
15 %       threshold for each RGB channel
16 %
17 %   Dependencies:
18 %   – findthresh.m – from rbf's ivr repository; Standard
    %       filterlen = 50,
19 %       alpha = sizeparam.
20 %       N.B.   if filterlen is large, then curve is smoother!
21 %       if alpha is large, then width of the window is
    %       smaller!
22
23 %%
24
25 % % In case sizeparam is not passed
26 % try sizeparam
27 % catch
28 %     sizeparam = 16;
29 % end
30
31 if iscell(IMGs)
32     [~, num_imgs] = size(IMGs); % find number of images
33     thresh_vals = {}; % for storing all the threshold values
34
35     for k = 1:num_imgs % iterate through all the images
36
37         img          = IMGs{k}; % this image
38         imgX          = zeros(size(img)); % the output image
39
40         if ndims(img) == 3 % RGB Channel
41
42             thresh_vals{k} = zeros(1,3); % pre-allocation
43             for i=1:ndims(img) % iterate through each dim to
                get the BW pic

```

```

44         % call itself to get the threshold value (see `
           else` below)
45         [imgX(:,:,i), thresh_vals{k}(i)] = ...
46             dothresh(img(:,:,i), sizeparam);
47     end
48
49     % Now, `OR` the values together
50     img_thresh{k} = imgX(:,:,1) | imgX(:,:,2) | imgX
        (:,:,3);
51
52     else % for 2D case
53         [img_thresh{k}, thresh_vals{k}] = dothresh(img,
        sizeparam);
54     end
55
56 end
57 %% 2D image input:
58 %   For 2D array, use findthresh to get the threshold for the
        image and
59 %   then get the bw representation of it!
60
61 %   TODO: MAY need to toggle the bw = ~bw, depending if you
        want objects to be
62 %   white or black
63 else
64     hist = dohist(IMGs); % get the histogram of 2D image
65     thresh_vals = findthresh(hist, sizeparam, 0); % find the
        threshold of the iamge
66     [n,m] = size(IMGs);
67
68     % now, get the binary representation
69     for i=1:n
70         for j=1:m

```

```
71         if IMGS(i,j) >= thresh_vals % this is the objects!  
72             we want it!  
73             img_thresh(i,j) = 1;  
74         else  
75             img_thresh(i,j) = 0; % set background to 0  
76         end  
77     end  
78  
79 end  
80  
81 end
```



## C Feature Extraction

getFeatures.m

```
1 function vec = getFeatures(image, prop)
2 %% getproperties(Image)
3 %   gets property vector for a binary shape in an image
4 %   properties extracted:
5 %       1) Area
6 %       2) Perimeter
7 %       3)
8 Image = image.Image;
9
10 [H,W] = size(Image);
11 area = bwarea(Image);
12 perim = bwarea(bwperim(Image,8));
13
14 % compactness
15 compactness = perim*perim/(4*pi*area);
16
17 % rescale properties so all have size proportional
18 % to image size
19 area_ = 4*sqrt(area);
20 compactness_ = H*compactness;
21
22 % rectangularity
23 bb_width = image.BoundingBox(3);
24 bb_height = image.BoundingBox(4);
25 area_bb = bb_width * bb_height;
26 rectangularity = area / area_bb;
27
28 % Elongation — ratio of principal axis
29 elongation = prop.MajorAxisLength / prop.MinorAxisLength;
30
```

```

31 hu_invariant = humomentinvariants(Image);
32
33 % get scale-normalized complex central moments
34 c11 = complexmoment(Image,1,1) / (area^2);
35 c20 = complexmoment(Image,2,0) / (area^2);
36 c30 = complexmoment(Image,3,0) / (area^2.5);
37 c21 = complexmoment(Image,2,1) / (area^2.5);
38 c12 = complexmoment(Image,1,2) / (area^2.5);
39 %c=[c11,c20,c30,c21,c12]
40
41 % get invariants, scaled to [-1,1] range
42 ci1 = real(c11);
43 ci2 = real(1000*c21*c12);
44 tmp = c20*c12*c12;
45 ci3 = 10000*real(tmp);
46 ci4 = 10000*imag(tmp);
47 tmp = c30*c12*c12*c12;
48 ci5 = 1000000*real(tmp);
49 ci6 = 1000000*imag(tmp);
50
51 %ci=[ci1,ci2,ci3,ci4,ci5,ci6]
52
53 vec = [area_, perim, compactness_ , rectangularity, elongation,
        hu_invariant, ...
        ci1, ci2, ci3, ci4, ci5, ci6]; % 18 features
54
55
56
57 end

```

#### rawmoment.m

```

1 function M_ij = rawmoment(img,p,q)
2 %% rawmoment(img,p,q) calculates the (p+q)th raw moment of img

```

```

3 % image is a BW img
4
5 %%
6 [m,n] = size(img);
7 M_ij = 0;
8 for i=1:m
9     for j=1:n
10         x=i; y=j;
11         I_xy = img(i,j);
12         M_ij = M_ij + (x^p * y^q * I_xy);
13     end
14 end
15
16
17 end

```

#### centrilmoment.m

```

1 function miu_pq = centrilmoment(img,p,q)
2 %% centrilmoment(img,p,q) calculates the (p+q)th central moment
   of img
3 % image is a BW img
4 % covariance = miu_11; variance = miu_02 or miu_20;
5
6 %%
7 [m, n] = size(img);
8 M_00 = rawmoment(img,0,0);
9 M_10 = rawmoment(img,1,0); % mean x
10 M_01 = rawmoment(img,0,1); % mean y
11 centroid_x = M_10/M_00;
12 centroid_y = M_01/M_00;
13
14 miu_pq = 0;

```

```

15 for i=1:m
16     for j=1:n
17         diff_x = (i-centroid_x) ^ p;
18         diff_y = (j-centroid_y) ^ q;
19         I_xy = img(i,j);
20         miu_pq = miu_pq + (diff_x * diff_y * I_xy);
21     end
22 end
23
24
25 end

```

#### SI\_moment.m

```

1 function pi_pq = SI_moment(img,p,q)
2 %% Calculates the Scale invariant moment given the (p+q) moment
3
4     miu_00 = centralmoment(img,0,0); % the area
5     miu_pq = centralmoment(img,p,q);
6
7     pi_pq = miu_pq / (miu_00^(1+(p+q)/2));

```

#### complexmoment.m

```

1 % gets a given complex central moment value
2 function c_uv = complexmoment(Image,u,v)
3
4     [r,c] = find(Image==1); % get (r,c) of region's
5     pixels
6     rbar = mean(r);
7     cbar = mean(c);
8     n = length(r);

```

```
8      momlist = zeros(n,1);
9
10     for i = 1 : n
11         c1 = complex(r(i) - rbar, c(i) - cbar);
12         c2 = complex(r(i) - rbar, cbar - c(i));
13         momlist(i) = c1^u * c2^v;
14     end
15
16     c_uv = sum(momlist);
```

## D Classification

split\_data.m

```
1 function [X_train, X_vali, X_test, y_train, y_vali, y_test] =  
2     ...  
3     split_data(X, y, train, vali, test)  
4 %% SPLIT_DATA(X, y, train, vali, test);  
5 % Use hold out validation technique to randomly generate the  
6 % training, validation and testing set  
7 % Since the images are input, we will use create psuedo  
8 % samples from the  
9 % subimages  
10 % INPUT:  
11 % — train,vali,test : are double from [0,1] that indicate the  
12 % size of each  
13 % sets. Hence they must sum up to 1;  
14 %%  
15 num_instances = length(X);  
16 if length(X) ~= length(y)  
17     error('X and y does not have the same number of instances')  
18 ;  
19 end  
20 if (train + vali + test) ~= 1;  
21     error('train + vali + test ~= 1!!')  
22 end  
23 num_train = floor(num_instances * train);  
24 num_vali = floor(num_instances * vali);  
25 num_test = num_instances - num_train - num_vali;
```

```

26 X_train_idx = randperm(num_instances,num_train);
27 X_train     = X(X_train_idx, :);
28 y_train     = y(X_train_idx);
29 % remove these instances
30 X(X_train_idx,:) = [];
31 y(X_train_idx)   = [];
32
33 X_vali_idx = randperm(num_instances-num_train, num_vali);
34 X_vali     = X(X_vali_idx, :);
35 y_vali     = y(X_vali_idx, :);
36 % remove these instances
37 X(X_vali_idx,:) = [];
38 y(X_vali_idx)   = [];
39
40 % the rest for test:
41 X_test = X;
42 y_test = y;
43
44 end

```

#### user\_classify.m

```

1 function [relevance, class] = user_classify()
2 %% USER_CLASSIFY(IMG)
3 %   Given an img, ask the user which class it belongs to
4
5 %%
6 % fprintf('\n\nplease enter the two class for this img\n');
7 prompt = 'Is this relevant? [0/1]';
8 relevance = input(prompt); % to count the class or not!
9
10 if relevance
11     fprintf('\n====\nWhats the value?\n');

```

```

12     fprintf('[1] 1 POUND [2] 2 POUND [3] 50 P [4] 20 P [5]
        5 P\n')
13     fprintf('[6] 75 P (washer w small hole) [7] 25 P (washer w
        large hole)\n');
14     fprintf('[8] 2 P (angle bracket)\n[9] AAA battery (no val)
        [10] nut (no value)\n');
15 %     fprintf('[11] HELP!! (will display the bigger picture)\n\
n');
16     fprintf('[0] HELP!! (will display the bigger picture)\n\n')
        ;
17     prompt = '>> ';
18     class = input(prompt);
19
20     % Reject error in class input
21     while (class < 0 || class > 10)
22         fprintf('Classes ranges from 1 to 11 only\n');
23         class = input(prompt);
24     end
25
26     fprintf('\n===\n')
27 else
28     class = 11;
29 end
30
31
32 end

```

#### findConfusion.m

```

1 function[ CM, Per ] = findConfusion(result, test_class,
    num_class, p_limit)
2 %% findConfusion
3 % INPUT: [targets, output]

```



```

4 % S = number of features ( in this case, 10)
5 % Q = number of test data
6 % result      :   Q-by-1 data each (i,j) indicates the ith
    input's class,
7 % test_class  :   Q-by-1 data each (i,j) indicates the class
    given to ith
8 %              input.
9 % targets and output must be ordered the same way.
10
11 % OUTPUT: [c. cm, ind, per]
12 % cm :   S-by-S confusion matrix, where (i,j) is the number
    of samples
13 %           whose target is the ith class that was classified
    as j
14 % per :   S-by-4 matrix, where each row summarises four
    percentages
15 %           associated with the ith class:
16
17
18 %% setup:
19 [Q,S] = size(result); % Q = number of observation
20 [~,S1] = size(test_class);
21
22 % check for number of test-case
23 if S ~= S1
24     error('test_class and results doesnt match in size');
25 end
26
27 %% create the confusion matrix
28 % Row = actual
29 % column = predicted
30 cm = zeros(num_class, num_class); % dont need to show cm for
    class 11
31

```

```

32 % iterate through all the test data to add data into the
    confusion matrix
33 for q=(1:Q)
34     predictedClass = test_class(q,1);
35     actualClass = result(q,1);
36
37     if predictedClass == actualClass
38         % if the classifier successfully classsified the
            datapoint
39         cm(actualClass,actualClass) = ...
40             cm(actualClass,actualClass) + 1;
41     else
42         % classifier classifies the point wrongly.
43         cm(actualClass, predictedClass) = ...
44             cm(actualClass,predictedClass) + 1;
45     end
46 end
47
48 %% manipulate the cm to get per:
49 per = zeros(num_class,4); % ignore the unclassified class here
50 % each row corresponds to each class
51 %     per(i,1) false negative rate
52 %           = (false negatives)
53 %     per(i,2) false positive rate
54 %           = (false positives)
55 %     per(i,3) true positive rate
56 %           = (true positives)
57 %     per(i,4) true negative rate
58 %           = (true negatives)
59
60 % for each class find the FN, FP, TP, TN respectively.
61 for s=(1:num_class)
62     % generate the data;
63     TP = cm(s,s);

```

```

64     FP = sum(cm(:,s)) - TP;
65     FN = sum(cm(s,:)) - TP;
66     TN = sum(sum(cm)) - TP - FN - FP;
67
68     % store the values
69     per(s,1) = FN;
70     per(s,2) = FP;
71     per(s,3) = TP;
72     per(s,4) = TN;
73 end
74
75 CM = cm;
76 Per = per;
77
78 %%
79 figure; imshow(CM, [], 'InitialMagnification', 1600); colormap(
    bone);
80 title('Confusion Matrix for Testing Set');
81
82 fprintf('Done!\n\nThe confusion matrix is:\n(rows = actual
    class; columns = predicted class)\n');
83 disp(CM);
84 fprintf('\nThe classification results for each class are:\n (
    FN    FP    TP    TN)\n');
85 disp(per);
86
87 disp('=====');
88 fprintf('Summary:\nClassification using full gaussian model\n')
    ;
89 FN = sum(per(:,1));
90 FP = sum(per(:,2));
91 TP = sum(per(:,3));
92 TN = sum(per(:,4));

```

```

93 incorrect = FP + FN;
94 correct = TP;
95 acc_score = TP/ Q; % Q = number of obervation
96
97 fprintf('Number Incorrect = %d\n', incorrect);
98 fprintf('Number Correct = %d\n', correct);
99 fprintf('Number Unclassified (lesser than p = %.2f) = %d\n',
    p_limit, Per(11,1) );
100 fprintf('Accuracy = %3f percent\n\n', acc_score);
101 disp('=====');
102 );
103 end

```

#### gaussianDistr.m

```

1 function p = gaussianDistr(mean_, cov_, prior, data)
2 %% GAUSSIANDISTR(MEAN, COV, X)
3 % using log posterior probability:
4 %  $\ln P(C|x) = (-.5)(x-\mu)'(\text{inv}(\text{cov}))(x-\mu) - .5(\ln(\det(\text{cov}))$ 
   %  $+ \ln(P(C))$ 
5 %
6 % INPUT:
7 % mean = scalar; mean of gaussian distribution
8 % cov = D-by-D matrix; covariance of ditribution
9 % x = D dimension vector to calculate the pr.
10 %
11 % OUTPUT:
12 % p = probability of x being classified using this
   gaussian model
13
14 %% generate Probability;
15

```

```

16 diff = data - mean_;
17 dist = diff*cov_*diff';
18 n = length(data);
19 wgt = 1/sqrt(det(inv(cov_)));
20 p = prior * ( 1 / (2*pi)^(n/2) ) * wgt * exp(-0.5*dist);
21 disp(p); %% DEBUG
22 %
23 %
24 % [A,D] = size(cov_);
25 % mean_ = mean_'; % assume data and mean is presented as row
    vector
26 % data = data';
27 %
28 % logDet = (-.5) * logdet(cov_);
29 % firstPart = (-.5) * ((data - mean_)' / cov_) * (data - mean_)
    ;
30 % prior = log(prior);
31 %
32 % % calculate the probability using the formula:
33 % p = firstPart + logDet + prior;
34
35 end

```

#### gaussian\_clf.m

```

1 function [prediction, prob_all] = gaussian_clf(X_test,
    DATA_CLASS, p_limit)
2 %% GAUSSIAN_CLF(X_TEST, MEAN, COVARIANCE)
3 % Given a the features of some images (X_test), we use a
    gaussian model
4 % to find the most probable class (i.e. highest probability)
    ;
5 %

```

```

6 % INPUT:
7 % — DATA_CLASS is a cell array of struct where each cell
  gives us the
8 % information of the class. The number of class = length of
  DATA_CLASS
9 %
10 % OUTPUT:
11 % — predictions : a list of classes for each instances in
   X_test
12
13 %%
14
15 num_class = length(DATA_CLASS);
16 num_sample = length(X_test);
17
18 prediction = zeros(num_sample,1);
19 prob_all = zeros(num_sample, num_class);
20
21 for n=1:num_sample
22     for i=1:num_class
23 %         fprintf('%d,%d',n,i); %%DEBUG
24         prior_ = DATA_CLASS{i}.Prior;
25         mean_ = DATA_CLASS{i}.Mean;
26         cov_ = DATA_CLASS{i}.Cov;
27         data = X_test(n,:);
28         p = gaussianDistr(mean_, cov_, prior_, data);
29 %         disp(p);
30         prob_all(n,i) = p;
31
32     end
33
34     [probs, prediction(n)] = max(prob_all(n,:));
35
36 % toggle the use of p_limit

```

```
37     if nargin == 3
38         % If probability is larger than the confidence interval
           , thrash it!
39         for i = 1:num_sample
40             if probs < p_limit
41                 prediction(n) = 11; % set to unclassified
42             %             fprintf('x');
43             end
44         end
45     end
46
47
48 end
49
50 end
```

## E Basic Filters

median\_filter.m

```
1 function img_filtered = median_filter(img, show, SIZE )
2 %% MEDIAN_FILTER
3 %   Use median filter to reduce the impulse noise in the image
4 %   base on the
5 %   local intensity distribution. The distribution being
6 %   considered by the
7 %   filter is determined by SIZE.
8 %
9 %   If there are more than 2 dims in img (such as a HSV or RGB)
10 %   image,
11 %   median filter is passed through each dimension
12 %   independently. The
13 %   resulting image is then put together as img_filtered.
14
15 % INPUT:
16 %   SIZE — either a scalar or a vector representing the row and
17 %   col. If not
18 %   defined, the default value of 3x3 is used.
19 %   img — image to be filtered
20 %   show — 0/1 to imshow the images
21
22 %% Do Median Filtering for each channel (can be HSV/RGB)
23 if ndims(img) == 3
24     RGB = img;
25     [r, c, channel] = size(img);
26     red_org      = RGB(:, :, 1);
27     green_org    = RGB(:, :, 2);
28     blue_org     = RGB(:, :, 3);
29
30     if nargin == 3
```



```

26         red_medfilt      = medfilt2(red_org, SIZE, 'symmetric');
27         green_medfilt    = medfilt2(green_org, SIZE, 'symmetric'
28             );
29         blue_medfilt     = medfilt2(blue_org, SIZE, 'symmetric')
30             ;
29     else
30         red_medfilt      = medfilt2(red_org, 'symmetric');
31         green_medfilt    = medfilt2(green_org, 'symmetric');
32         blue_medfilt     = medfilt2(blue_org, 'symmetric');
33     end
34
35     img_filtered = cat(3, red_medfilt, green_medfilt,
36         blue_medfilt);
37
37     else
38         %% DO 2D Median Filtering
39         if nargin == 3
40             img_filtered = medfilt2(img, SIZE);
41         else
42             img_filtered = medfilt2(img);
43         end
44
45     end
46
47     if show
48         display_stats(img, img_filtered);
49         %     figure; imshow([img, img_filtered]);
50     end
51
52 end

```

median\_filter\_iter.m

```

1 function img_filtered = median_filter_iter(img, ITER, show,
    SIZE)
2 %% MEDIAN_FILTER_ITER
3 % Use median filtering for a definite number of times.
4 % INPUT:
5 % img      : initial image
6 % ITER     : Number of iteration
7 % show     : to display the images after
8 % SIZE     : SIZE of the filter window (OPTIONAL)
9
10 %%
11
12 img_temp = img;
13
14 try
15     for i = 1:ITER
16         img_temp = median_filter(img_temp, 0, SIZE);
17     end
18 catch
19     for i = 1:ITER
20         img_temp = median_filter(img_temp, 0);
21     end
22 end
23
24 if show
25     display_stats(img, img_temp);
26 end
27
28 img_filtered = img_temp;
29 end

```

gaussian\_filter\_1d.m

```

1 function smoothed_1d = gaussian_filter_1d(hist, show,
    window_size, alpha)
2 %% GAUSSIAN_FILTER_1D(HIST, SHOW, WINDOW_SIZE, ALPHA)
3 %   Uses the gausswin function to produce a gaussian window,
4 %   then apply a conv to the 1d-hist.
5 %   If hist is not 1d, coerce it into 1d
6
7 %   Note:
8 %   As alpha increase, width of window will decrease. Default =
    2.5
9 %   As window_size increase, the curve will be smoother.
10 %   Use dohist to get the histogram!
11
12 %%
13 % first check for the size of the image, if not 1d, coerce it
14 if ndims(hist) == 3 % a color image is input,
15     % convert to grayscale first:
16     hist = rgb2gray(hist);
17 end
18
19 % Use dohist to get the histogram of intensity
20 hist = dohist(hist);
21
22 % window_size and alpha not defined, use default
23 if nargin == 1 || nargin == 2
24     gauss_filter = gausswin(50, 6);
25 else
26     gauss_filter = gausswin(window_size, alpha);
27 end
28
29 filter = gauss_filter/ sum(gauss_filter);
30 smoothed_1d = conv(filter, hist);
31
32 try

```

```

33     if show
34         subplot(2,2,1); plot(hist); title('Original Image');
35         subplot(2,2,3); plot(filter); title('Filter');
36         subplot(2,2,2); plot(smoothed_1d); title('Smoothed
           Image');
37     end
38 catch
39     subplot(2,2,1); imshow(hist); title('Original Image');
40     subplot(2,2,3); plot(filter); title('Filter');
41     subplot(2,2,2); imshow(smoothed_1d); title('Smoothed Image'
           );
42 end

```

#### gaussian\_filter\_2d.m

```

1  function smoothed_2d = gaussian_filter_2d(img, show, HSIZE,
      SIGMA )
2  %% gaussian_filter_2d(img, show, HSIZE, SIGMA )
3  % Smooth an image using Gaussian lowpass filter and imfilter
4  % INPUT:
5  % — img : can be an RGB/HSV or GRAYSCALE image
6  % — HSIZE : corresponds to fspecial requirements, can be a
      vector
7  %   specifying the number of rows and columns or a scalar (
      infered to be a
8  %   squared matrix
9  % — SIGMA : the spread of the Gaussian
10 % N.B. Default HSIZE = [3,3], SIGMA = .5
11
12 if (nargin == 1 || nargin == 2)
13     H = fspecial('gaussian');
14 else
15     H = fspecial('gaussian', HSIZE, SIGMA);

```

```

16 end
17
18
19 % ensure the output is the same size as img
20 % use conv instead of filter function
21 smoothed_2d = imfilter(img, H, 'conv', 'same');
22
23 try
24     if show
25         subplot(2,2,1); imshow(img); title('Original Image');
26         subplot(2,2,3); surf(H); title('Filter');
27         subplot(2,2,2); imshow(smoothed_2d); title('Smoothed
                Image');
28     end
29 catch
30     % if fail to input show, will just output the images!
31     subplot(2,2,1); imshow(img); title('Original Image');
32     subplot(2,2,3); surf(H); title('Filter');
33     subplot(2,2,2); imshow(smoothed_2d); title('Smoothed Image'
        );
34 end

```

## References

- [1] E.R. Davies. *Computer and Machine Vision (Fourth Edition)*. Academic Press, Boston, fourth edition edition, 2012. ISBN 978-0-12-386908-1. doi: <http://dx.doi.org/10.1016/B978-0-12-386908-1.00001-X>. URL <http://www.sciencedirect.com/science/article/pii/B978012386908100001X>.
- [2] Noah Snavely. Lecture 2: Image filtering. URL [http://www.cs.cornell.edu/courses/cs6670/2011sp/lectures/lec02\\_filter.pdf](http://www.cs.cornell.edu/courses/cs6670/2011sp/lectures/lec02_filter.pdf).
- [3] A. Walker R. Fisher, S. Perkins and E. Wolfart. Hpr2 - image processing learning resources. URL <http://homepages.inf.ed.ac.uk/rbf/HIPR2/index.htm>.
- [4] Andrew Ng. Stanford machine learning class notes. URL <http://www.holehouse.org/mlclass/index.html>.