Introduction to Vision and Robotics
Assessed Practical 1 - Coin Counter

Ramona Comanescu (s1427590)
Andreea Cucu (s1408218)

October 2016

Contents

Part I
Introduction

In this report we aim to detail how several vision techniques have been employed to detect, extract features and classify coins and other small objects from images.

As an initial step, we preprocessed the images by blurring them and increasing contrast. Background subtraction was used for obtaining the objects in each image, a technique that yielded good results because we were able to obtain a quiet good initial background.

Furthermore, we cropped individual objects from the initial image with the aid of the properties of image regions, specifically bounding boxes. The final stage consisted of extracting image signatures and complex moments that we then used for training a Naive Bayes Classifier.

We made use of material from the course slide, labs and some built in Matlab functions(mainly for removing noise and getting bounding boxes), while experimenting with a lot of different ideas and refining our approach. We analysed the results at each stage in depth by not only looking at accuracy, but at how much sense individual features made.

Part II
Method

1 Preprocessing

While working on the segmentation process, we found that our algorithms would be affected by factors like the difference in colour intensity or lightness. We solved these issues by modifying and applying some filters to the original image before any other processing.

1.1 Smoothing the Image

One of the most challenging problems we had to overcome during segmentation step was the noise around the objects that would decrease the accuracy of the edge detection. When doing background subtraction, we had to tackle the issue of light pollution: some objects are often placed in the shadow, which interferes with the subtraction algorithm, preventing it from correctly identifying the objects from the background. We decided to apply a Gaussian filter to remove the noise and decrease the difference in brightness intensity across the image. Since we still needed the edges of the image to be identifiable, we used a small standard deviation factor $\sigma = 0.1$.

1.2 Contrast Adjustment

When attempting to segment the image, we noticed that some of the coins were not easily detectable. The most obvious case is the pound coin, since its colour is very similar to the background. In order to correctly identify the objects in the image, we needed a way to differentiate between the background and the edges of the object. This can be accomplished by increasing the contrast of the images before processing them.

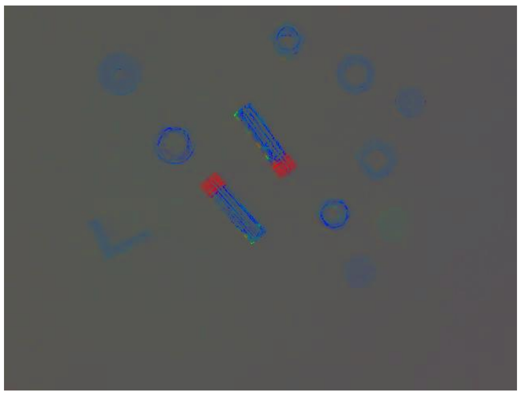
```
        %adjusting the contrast
        se = strel('disk',3);
        contrasted_image = (blurred_image + ...
            imtophat(blurred_image,se)) -
                          imbothat(blurred_image,se);
```

The top-hat filtering is defined as the difference between the original image and its opening, while the bot-hat filtering represents the difference between the original image and it's closing. By adding the top-hat filtered image and subtracting the bot-hat filtered image from the original, we can increase the contrast by removing the brighter parts of the image and enhancing the darker ones.

1.3 RGB Normalisation

In order to cope with changes in illumination, we tried to RGB normalise the picture. In order to do this, we divided each pixel's value by the sum of the pixel's value over all channels. However, most of the coins were similar in colour and we were not happy with the result, because after background subtraction we lost a lot of detail. We suspect that this technique would have worked better on images with a more varied range of colour values.

Figure 1: Thresholding Foreground objects



1.4 HSV Normalisation

Another attempt at solving the illumination problem was to convert the RGB image to HSV. In this colourspace, the last channel, Value stores the intensity of the colour. The advantage of using this format is that we can simply ignore the intensity, which would be influenced by illumination. This normalisation technique can used in the edge detection method, allowing us to correctly identify the objects regardless of how bright or dark they were.

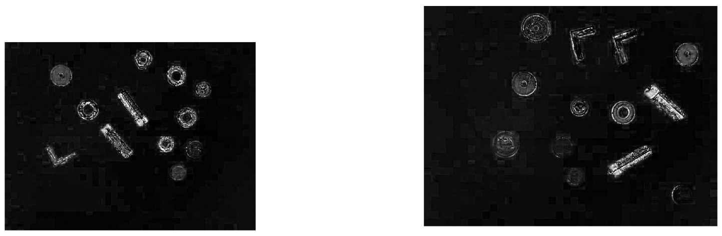

(a) Background shadow removed (b) Objects in the shadow are visible

Figure 2: HSV Normalisation in a simpler and more complex image

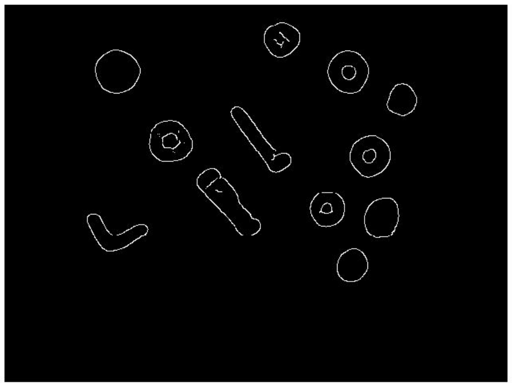
2 Segmentation

The first step of classifying the objects in the picture is to identify the different objects so we can extract individual features for each of them and feed them into the training algorithm. For this section, we each tried a different approach: background subtraction and edge detection. We will cover an in depth explanation of both methods, illustrating their limitation and the reasons we decided in favour or against them. After identifying the contents of the image as blobs of colour, we cropped the individual objects so we can perform the classification.

2.1 Edge detection

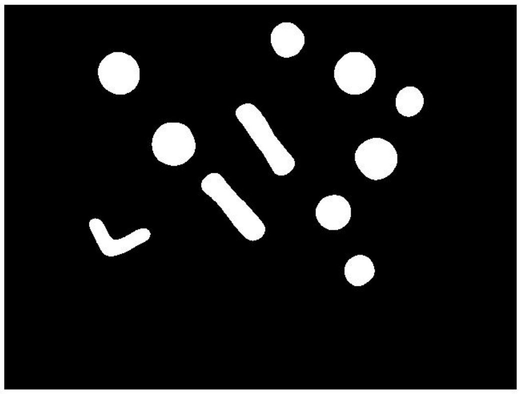
The idea behind edge detection is to use the difference in brightness in the image to identify the boundaries between the objects. Before applying the edge detection algorithm to the image we converted it to the HSV colourspace and ignored the values in the V channel, to make sure the picture is uniformly illuminated. Before processing the image, we apply a gaussian blur to minimise the noise. We performed thresholding on the image by computing the mean of all the colours and ignoring all the pixels below this value. On this new image, we applied the Canny edge detection algorithm to obtain a binary image with the outlines of the object. The resulting image identified all the objects correctly.

Figure 3: Object Edges obtained through Canny detection



However, because the edges were too thin, it was not possible to find an area to place a bounding box around to crop each object. Hence, we applied a series of operations on this image to make the object areas more noticeable. Through dilating and closing the edges, we managed to clearly see the objects as blobs of white pixels.

Figure 4: Binary mask indicating the position of the objects



While this is good for cropping out the objects in the original image, the binary mask wouldn't correctly identify whether an object had holes or not. Since most of our features are based on the binary version of the object, we had to abandon this method and use background subtraction, which produces a more accurate mask.

2.2 Background subtraction

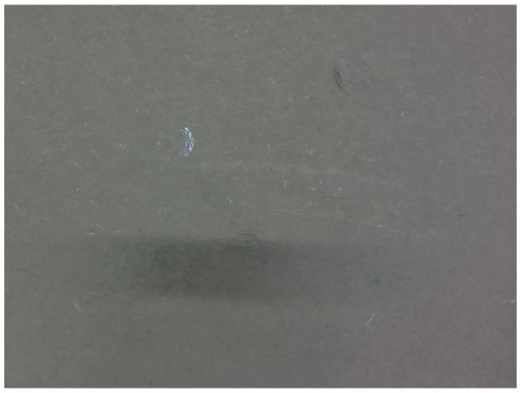
When trying to segment the actual objects, simple thresholding would not be enough because of the complexity of the scene. The technique we used for separating the foreground objects from the background was median background subtraction.
However, since there was no picture with just the background in it, we had to produce an approximate value for it by just taking the median of each pixel in each channel.
In our final background image, for a particular channel, a pixel (r,c) would have value $v = median(v_1, v_2, ..., v_n)$ where n is the number of images. Since we were provided with several images with the same background, we were able to use all of them in computing the median values.

```
% get median background
for i=1:n
  im=imread(allimages{i});
  list(:,:,:,i)=imgaussfilt(im ,0.1);
end

medianBackground=median(list,4);
```

The resulting background image was quite good. However, because some of the coins appeared many times in the initial images, they are still visible in the median background.

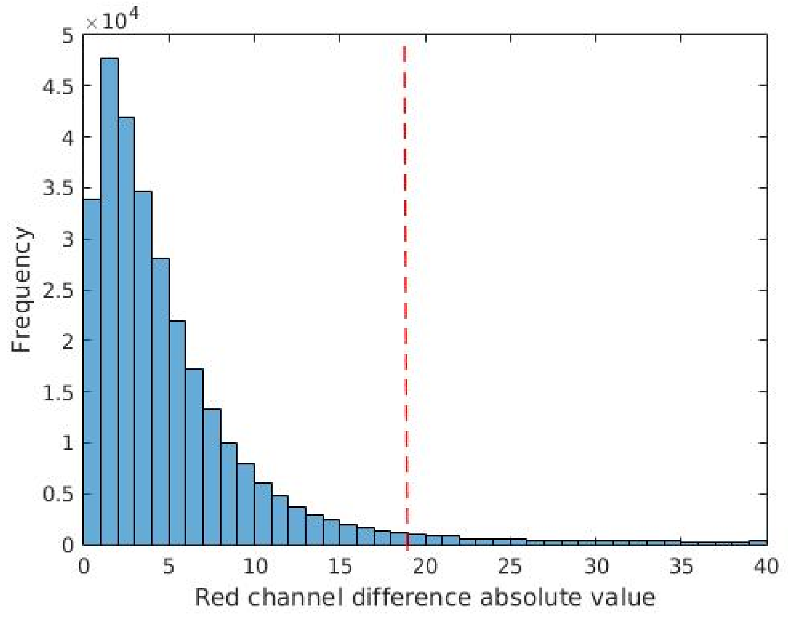
Figure 5: Median background



When given an input image, the (median) background can be subtracted from it, with the goal of considering only the objects that are different enough from it. The next step in to apply a threshold to the absolute difference, in order to obtain a *foreground mask*. This step is repeated for every colour channel and

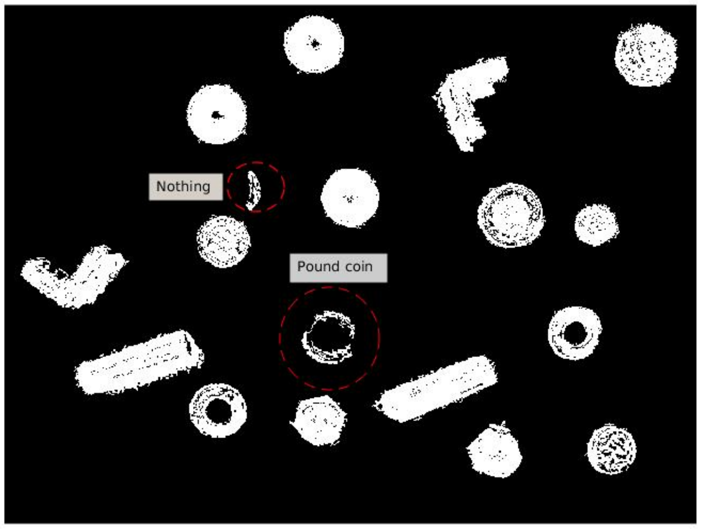
the result should be the disjunction of all the masks.
Individual thresholds have been set by analysing histograms for every channel and choosing a value that separated the foreground well.

Figure 6: Thresholding Foreground objects



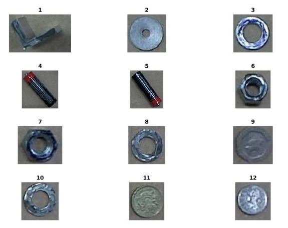
A threshold of 19 was chosen for every channel, mainly by trail and error, thus obtaining a binary array with 1(white) indicating object pixels. The lower the threshold value, the less of the objects are recovered. The main problem we encountered was the case of the *one pound* coin placed in a shadow, where not all the details could be recovered. We also removed every object with an area smaller than 60, since it proved to be noise.

Figure 7: Foreground detection



2.3 Cropping

Since our background subtraction resulted in connected (white) components, we extracted those as a bounding box and cropped out the object we where interested in. We made sure to apply the same box coordinates for cropping the original RGB image, so we could also use features that related to colour. This also helps for displaying the final result in a human distinguishable form. We managed to correctly segment every training picture, with the exception with some cases when the pound coin situated in a shadow was only partially present.


(a) Original Bounding boxes

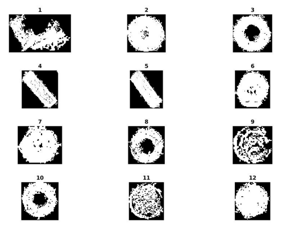

(b) Black and white

Figure 8: Foreground objects in image 02.jpg

3 Classification

After correctly separating the objects, we tried different classification methods to predict the classes of new images. This section presents a description of the features we extracted for each object and the classifiers we implemented for this. We trained four different classifiers and computed the accuracy for each one, in order to make the best decision. We also analyzed the mean and variance of each feature to see if they would be good at making a prediction.

3.1 Features

The first step of the classification process is extracting individual features from each of the test images. Having a numerical representation of the objects helps with computing probabilities and performing a statistical analysis on the objects.
The features that were extracted and used for classification were:

$$vec = [compactness, ci1, ci2, convexHullArea, hueThreshold] \tag{1}$$

We will proceed to explainin the reasoning behind each feature.

3.1.1 Compactness

Since most of the objects are either washers or coins, we thought it would be a great idea to help distinguish the round objects from batteries and angle brackets. We can do this by computing the circularity or compactness of the objects.

$$Compactness = \frac{Perim^2}{4 * \pi * Area} \tag{2}$$

Using this equation, the compactness of a circle will be 1 and the higher it goes the less circular the object is. We can see in the figure before how the round object, in this case, a small hole washer, has a low circularity value compared to the angle bracket, making it easy to differentiate between them.

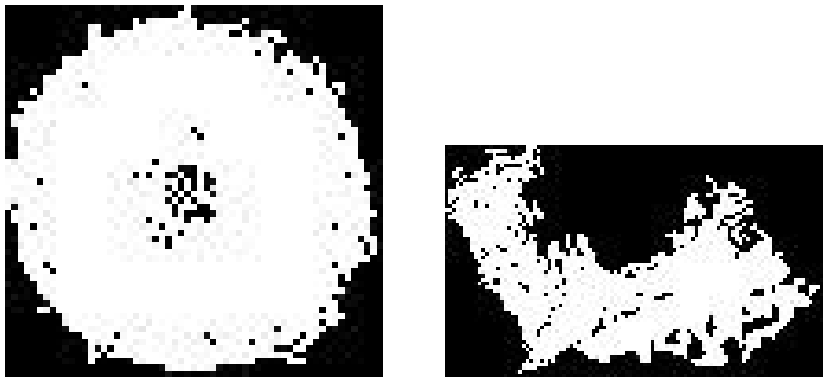

(a) circularity = 1.22 (b) circularity = 4.69

Figure 9: Difference between circularity values

3.1.2 Complex moments

Moments are particular weighted averages used to describe image segments, which are invariant to rigit transformation permitting us to get good shape transformation regardless of factors such as the orientation of the object or its distance from the camera. The first two complex moments $ci_1 ci_2$ have been included in the properties vector.

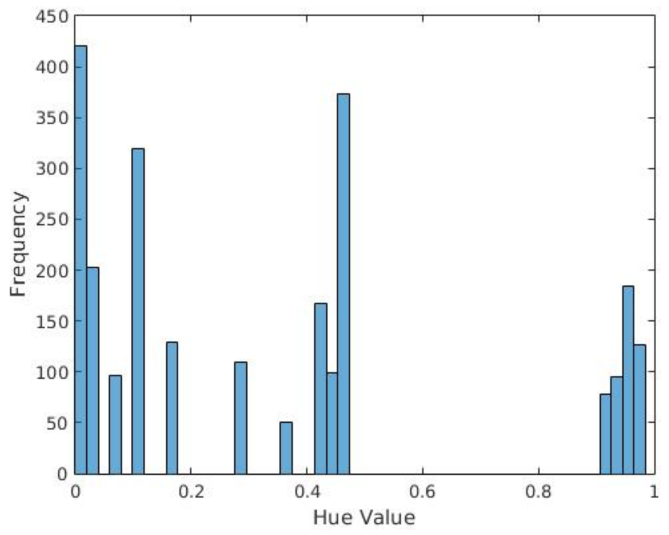
3.1.3 Convex Hull Area

Another feature worth taking into consideration when performing the statistical analysis on the objects is the area of the object. We can approximate the area by fitting the object into a convex polygon, since it would be easier to measure.// The regionprops operator *'ConvexHull'* returns the coordinated of the vetices of the smallest poligon that can contain the object. Using the *'Convex Area'* operator, we can calculate the area of the poligon, which is the most accurate approximation of the object area we can get.

3.1.4 Hue Thresholding

An effective heuristic that can be used is getting the ratio of the pixels that are between certain thresholds, compared to the total number of pixels. This is a good colour feature because different classes of objects have different peaks in their hue distribution.

Figure 10: Pound coin hue distribution



One of the most frequent problems we encountered with our classifier was that we couldn't easily differentiate between the coin values. The pound coin and the small hole washer have very similar dimentions, and the hole in the latter isn't big enough to be picked up by one of our other features. We are solving this problem by using the a colour heuristic to tell the pound coins apart from the silver objects. By analyzing different histograms, a hue range of $[0.36, 0.48]$ was chosen.

3.1.5 Solidity

Another feature that proved to be relevant was solidity, but this was not added to the feature vector because we tried to minimize its length.
After approximating the area of the object to that of the convex element, we can find out what fraction of that area is actually our object, which is referred to as computing the solidity. This can be done by dividing the *Area* by the *Convex Area*. The rounder the object is, the closer it will be to the actual area of the convex hull, giving a solidity of 1. This will again help differentiate between the round shapes and the batteries and angle brackets.

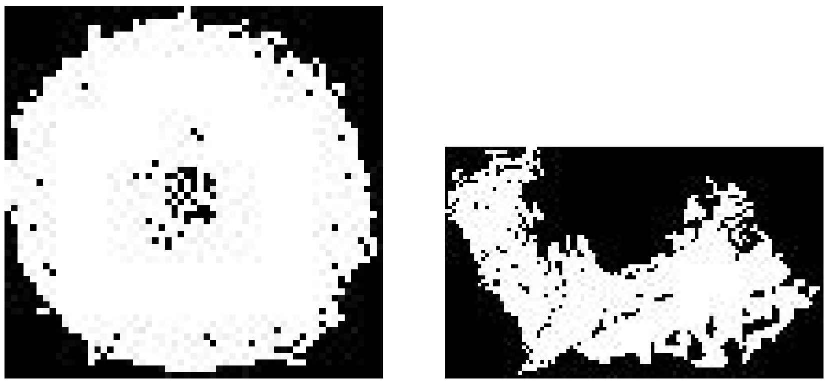

(a) solidity = 0.94 (b) solidity = 0.66

Figure 11: Difference between solidity values

3.2 Naive Bayes Classifier

A simple but effective Naive Bayes Classifier has been chosen, mainly because this classifier performs better on less amounts of training data than others, given that the assumption of independence between the features holds. The downside of using this algorithm is that, in the scenario where one of the classes does not have enough training data, the algorithm will assign a probability of 0, known as *Zero Frequency*, being unable to make a prediction. Since the training data we used did have multiple objects of the same class, this issue did not impact our ability to predict the classes for the test data.
For this classifier, we attempted two different distributions of the data, the normal and the kernel smoothed distribution.

The Kernel smoothing density estimate defines the shape of the probability distribution function used by the classifier. Unlike a histogram, it will sum the component smoothing functions for each feature, producing a continuous curve. We chose this distribution because a parametric one could not properly describe the data and we wanted to avoid making assumption about the distribution. This provides a more flexible data model, allowing for better classification when working with less amounts of test data.

The Gaussian distribution provided a more rigid model, but it allowed us to analyze our input vectors in a straightforward way, by working with the mean and covariance matrix.
For comparison, the mean feature vectors for *20p* and *battery* are as follows:

20p	30.348	0.191	0.0193	1725.6	0.09
battery	11.44	0.27	0.022	4948.011	0.076

However, this also limited the amount of features we could use, because in order to compute a covariance matrix, we needed the number of features to be less that the number of training data for each class.
Finally, qual priors have been assumed, since there were no assumptions to be made about the distribution of the classes.

3.3 Decision Tree Classifier

We came up with the idea of using a decision tree because, after observing the training data, we could see that the round objects would have very different features from the batteries and angle brackets. Similarly,the washers and nuts, because of the holes in the mask, would have different features from the coins. Since the Decision Tree would make a decision at every step, it could easily differentiate between round objects and the not round ones and the ones with or without holes. However, it did not perform well at finding the differences between the different coin values or the different sized washers.

4 Final Price

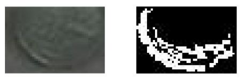
The final aim of this assignment is to find out the value of the items in the picture. We use a hash map to associate each class to it's monetary value. After determining the classes of all the objects, we then go through each of them and add up their value, which will be shown in the console output.

Part III
Results

5 Segmentation Results

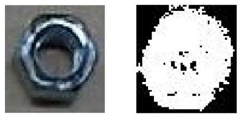
The segmentation stage managed to segment all objects from the given images. However, the pound coin could not be completely recovered from shadow:

Figure 12: Badly segmented Pound coin



Another problem that was encountered was not being able to small holes in nuts:

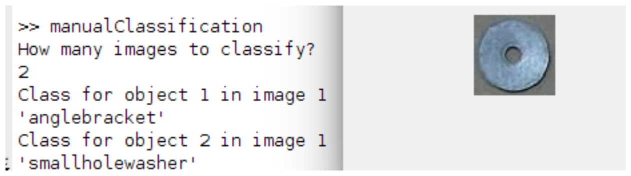
Figure 13: Losing the nut central hole



6 Training

We trained our classifier by first segmenting all the images and then asking the user for input, assigning one of the 10 possible classes. The training data was then obtained by computing feature vectors for each segmented object, giving us 125 datapoints to work with.

Figure 14: Obtaining training data



```
>> manualClassification
How many images to classify?
2
Class for object 1 in image 1
'anglebracket'
Class for object 2 in image 1
'smallholewasher'
```

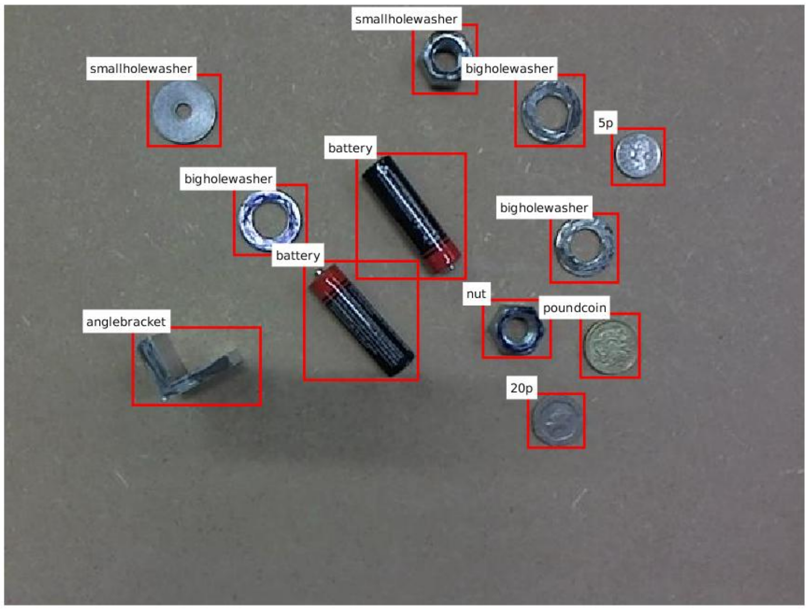
7 Testing and Accuracy

Because the amount of data was limited, it is easy to overfit or get the illusion of a high accuracy by just testing the classifier on easy to distinguish cases such as battery of angle bracket. Therefore, it is crucial to ensure that inputs from all classes are tested. ~
order to achieve this, cross validation was used, solving the problem of conflicting priorities (both a large training and testing set is needed).
By using a 5-fold cross validation, we train 5 non overlapping slices of the input at each time and testing on the non-train slice. Since the problem of imbalanced classes still persists, stratification sampling was used.

Accuracy: Using the 5-fold cross validation, a classification accuracy of *80%* was obtained.
Apart from cross validation, we have also randomly divided the input data into 80% training and 20% testing and get accuracies ranging between *70%* and *90%*, again depending on the split.

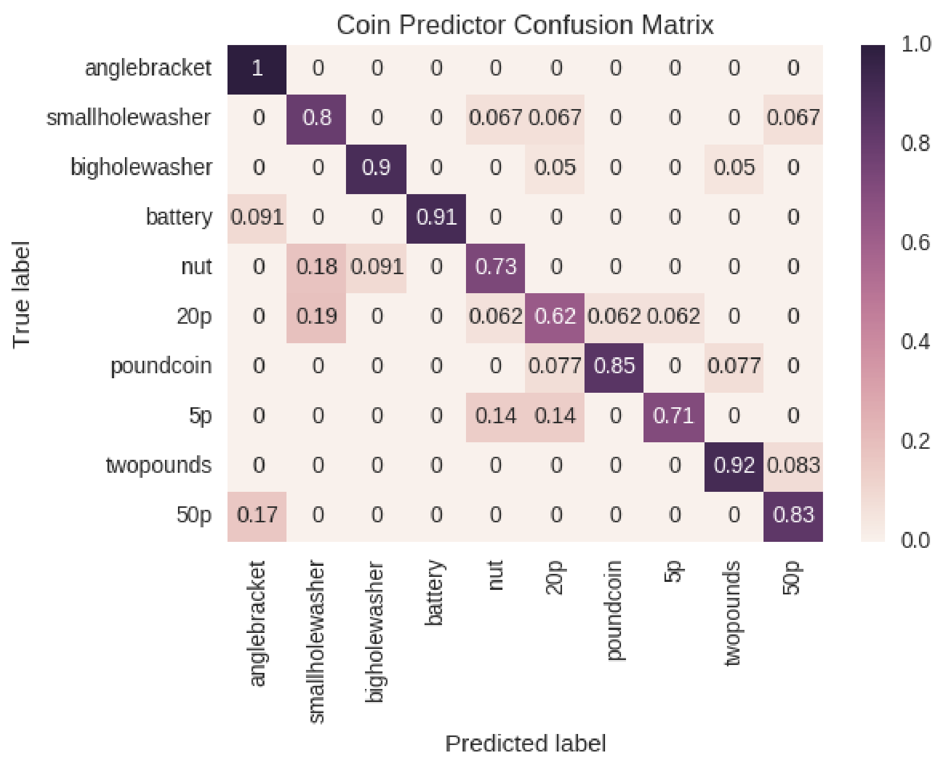
7.1 Classification Results

By setting aside one of the images and training only on the rest, we get good classification results, only assigned the wrong class to a nut object, which was predicted to be a small hole washer.



The confusion matrix for our (cross-validation) classification indicates good performance for most of the inputs, especially for the more distinctive non-coins ones (battery, angle bracket and big hole washer). The most missclassified class was *20p*, due to its similarity to other objects in terms of colour and shape.

Figure 15: Classification Results



	anglebracket	smallholewasher	bigholewasher	battery	nut	20p	poundcoin	5p	twopounds	50p
anglebracket	1	0	0	0	0	0	0	0	0	0
smallholewasher	0	0.8	0	0	0.067	0.067	0	0	0	0.067
bigholewasher	0	0	0.9	0	0	0.05	0	0	0.05	0
battery	0.091	0	0	0.91	0	0	0	0	0	0
nut	0	0.18	0.091	0	0.73	0	0	0	0	0
20p	0	0.19	0	0	0.062	0.62	0.062	0.062	0	0
poundcoin	0	0	0	0	0	0.077	0.85	0	0.077	0
5p	0	0	0	0	0.14	0.14	0	0.71	0	0
twopounds	0	0	0	0	0	0	0	0	0.92	0.083
50p	0.17	0	0	0	0	0	0	0	0	0.83

Part IV
Discussion

8 Limitations

We claim that the feature vectors we have chosen should be sufficient for correctly predicting most of the cases. However, feature extraction is based on correctly segmenting the objects and while we managed to recover all the objects, the black and white result included fuzziness and some distortion that could affect compactness.
A further aim would be to find methods for recovering brown objects situated in shadows and also making sure the central hole is not lost after background subtraction. Correctly detecting the proportion of a hole in objects would help with distinguishing hard cases such as the *5p* and *20* coins.
Moreover, it is desirable to automate all the thresholding values algorithms

(background difference, hue range) that have been obtained by trail and error at this initial stage.

8.1 SURF Point Features

Matlab's Machine Learning Toolbox provides a specialised image classifier which uses a Bag of Features as training data. The technique is adapted from natural language processing, but since images do not contrain words, a feature vector with numerical variables would be necessary to compute the vocabulary of the classifier. The Bag of Features is computed using SURF(Speed Up Robust Features) points. The advantage of using those would be that they are scale and rotation invariant

8.2 Hough Transform

While trying to come up with relevant features, we thought that there are some specific properties that each object class has.
Therefore, we could have used the Hough Transform in order to obtain all circles in the image, and then combining this with features such as the ratio between the diameter of the hole and the diameter of the circle(which would have helped with the small and big hole washer). This method could also tackle some of the 'harder' overlapping set of images by simply finding all the coins first.
By using this approach we could have improved accuracy, but we wanted to produce an algorithm that is more general and aligned with the course material.

Part V
Conclusion

This project enabled us to explore and understand how to use Matlab and it's Image Processing Toolbox to process images. It was a good introduction to computer vision, allowing us to understand concepts like segmentation, thresholding, background subtraction and foreground detection. Moreover, it was a good way to apply our knowledge of machine learning algorithms in the real world, encouraging us to develop algorithms for recognising features and classifying the data.

The most challenging part of the project was finding a good way to test our classifier with limited amounts of data. We believe that our apporach of randomly assigning 80% of the images for training and 20% for testing, followed by repeatedly applying the classification on the newly generated test data helps us overcome these issues and closely approximate the accuracy of our classification.

The objectives of this coursework have been met: we are correctly extracting the individual objects from the pictures, analysing them to form a feature vector and performing a classification with an accuracy rate of above 75%.

Part VI

Distribution of work

We consider that we both worked equally through this assignment, running experiments, arguing about the solution, developing algorithms and writing up the report. Thus the marks should be distributed 50:50.

Part VII

Appendix

```
function [ med ] = getMedianBackground()
% Get the background image by computing the median value at ...
    every pixel

all={'02.jpg';'03.jpg';'04.jpg';'05.jpg';'06.jpg';'07.jpg';
     '08.jpg';'09.jpg';'010.jpg';'017.jpg';'018.jpg';'019.jpg';
     '020.jpg';'021.jpg'};
all=strcat('input/',all);
n=size(all,1);

list=zeros(480,640,3,n);
for i=1:n
    im=imread(all{i});
    list(:,:,:,i)=imgaussfilt(im ,0.1);
end

med=median(list,4);

end
```

```
function [ im ] = getNormalizedImage(I )
% Perform RGB normalization
DI=double(I);
DI=DI+1e-10;
im = bsxfun(@rdivide, DI, sum(DI,3,'native'));

end
```

```
function [ q ] = thr( absv,thres)
% Threshold foreground objects
[x,y]=size(absv);
q=ones(x,y);
q(absv<thres)=0;

end
```

```
function [ fin ] = getForeground(I,med)

% Peform background subtraction and thresholding on a given image
original_image = double(I);
blurred_image = imgaussfilt(original_image,0.1);
se = strel('disk',3);
contrasted_image = (blurred_image + imtophat(blurred_image,se)) ...
    - imbothat(blurred_image,se);

absv=abs(contrasted_image-med);
r=absv(:,:,1);
g=absv(:,:,2);
b=absv(:,:,3);
diff_R=thr(r,19);
diff_G=thr(g,19);
diff_B=thr(b,25);
fin=diff_R|diff_B|diff_G;
fin=bwareaopen(fin,60);

end
```

```
function [ frac ] = hueThresh( i,bwi,bot,top )
% Return the ratio of pixels withing a certain hue range

i=rgb2hsv(i);
h=i(:,:,1);
obj_pixels=size(h(bwi~=0),1);
thr_pixels=size(h(h>bot & h<top),1);
frac=thr_pixels/obj_pixels;
%figure;
%xlabel('HueValue');
%ylabel('Frequency');
%histogram(h(:),50);
end
```

```
function vec = getfeatures(Image, ImageBW)
% Returns a feature vector for given object

area = bwarea(ImageBW);
perim = bwarea(bwperim(ImageBW,4));

% get compactness
compactness = perim*perim/(4*pi*area);


% get scale-normalized complex central moments
c11 = complexmoment(ImageBW,1,1) / (area^2);
c21 = complexmoment(ImageBW,2,1) / (area^2.5);
c12 = complexmoment(ImageBW,1,2) / (area^2.5);

% get invariants, scaled to [-1,1] range
ci1 = real(c11);
ci2 = real(1000*c21*c12);
```

```

% get area of convex hull
ch=bwconvhull(ImageBW);
convhull_area=bwarea(ch);

%hue range thresholding
hue_thresh=hueThresh(Image,ImageBW,0.36,0.48);

% return feature vector
vec = [compactness, ci1, ci2, convhull_area,hue_thresh];

end
```

```
function [ tags, inputs ] = manualClassification()

% Segemnt objects and return both feature vectors
% and their manually assigned classes

all={'02.jpg';'03.jpg';'04.jpg';'05.jpg';'06.jpg';'07.jpg';
    '08.jpg';'09.jpg';'010.jpg';'017.jpg';'018.jpg';'019.jpg';
    '020.jpg';'021.jpg'};


poss={'poundcoin';'twopounds';'bigholewasher';'smallholewasher';
     '50p';'5p';'20p';'nut';'battery';'anglebracket'};


medB=load('savedBackground.mat');
medB=medB.med;
n=input('How many images to classify?\n');
nrobjs=0;
tags={};
inputs=[];
for nrim=1:n
   I= imread(['input/' all{nrim}]);
   fin=getFin(I,medB);

   [labels,nan]=bwlabel(fin);
   rp=regionprops(labels);

   % remove small objects
   smalls=find([rp.Area]<300);

   for i=1:size(smalls,2)
     coord=round(rp(smalls(i)).BoundingBox);
     fin(coord(2):coord(2)+coord(4),coord(1):coord(1)+coord(3))=0;
   end

   % select objects by area
    bigs=find([rp.Area]≥300);


  for i=1:size(bigs,2)
     nrobjs=nrobjs+1;
     img = imcrop(I, rp(bigs(i)).BoundingBox);
     bwimg = imcrop(fin, rp(bigs(i)).BoundingBox);
```

```
        num=num2str(nrobjs);
        figure,imshow(img); title(num);
        % save objects
        imwrite(img,['output2/' num '.jpg']);
        imwrite(bwimg,['output2/bw' num '.jpg']);
        notok=true;
        while notok
        try
            theclass=input(['Class for object ', num, ' in image ',...
                num2str(nrim), '\n']);
            tags{nrobjs}=theclass;
            props=getfeatures(img,bwimg);
            inputs(nrobjs,:)=props;
            notok=false;
            if size(find(strcmp(poss,theclass)),1)==0
                notok=true;
                disp('This is not a permitted class, try again.');
            end
        catch
            disp('Invalid input, try again.')
        end
        end
        close all;
    end

end

% save all tags for the classified objects
save('savedtags.mat',tags);
save('savedinputs.mat',tags);

end
```

```
function [ priceMap ] = getPriceMap( )
% Return a map with coins and their values

KeySet = {'twopounds', 'poundcoin', '50p', '20p', '5p', ...
         'smallholewasher', 'bigholewasher', 'anglebracket',...
          'battery', 'nut'};
ValueSet = [200, 100, 50, 20, 5, 75, 25, 2, 0, 0];

priceMap = containers.Map(KeySet, ValueSet);


end
```

```
% compute accuracy using random split and 5-fold cv

theclasses=load('savedtags.mat');
theclasses=theclasses.tags;
input=load('savedinput.mat');
input=input.vec;

% divide random
```

```
[train,test]=dividerand(125,0.8,0.2);
trainIn=input(train,:);
trainOut=theclasses(train);

model=fitcnb(trainIn,trainOut);

testIn=input(test,:);
testOut=theclasses(test);
testOut=testOut(:);

predictions=predict(model,testIn);
gotRight=sum(cellfun(@strcmp,predictions,testOut));
accuracy=double(gotRight)/size(testIn,1);
disp(accuracy);

% cross-validation

modelcv=fitcnb(input,theclasses,'Prior','uniform');
cvmodel = crossval(model5,'kfold',5);
cvError = kfoldLoss(cvmodel);
yp=cvmodel.kfoldPredict();
order = {'twopounds', 'poundcoin', '50p', '20p', '5p', ...
         'smallholewasher', 'bigholewasher', 'anglebracket',...
          'battery', 'nut'};

cf=confusionmat(theclasses,yp,'order',order);
cf=bsxfun(@rdivide,cf,sum(cf,2));
HeatMap(cf, 'RowLabels', order, 'ColumnLabels', ...
    order,'ColorMap',colormap('summer'));

disp(order);
disp(cf);
disp(1-cvError);
save('fullcvmodel.mat','modelcv');
```

```
function [predv] = makePrediction (impath)
% Segment and classify objects in given image


% initialize model
theclasses=load('savedtags.mat');
theclasses=theclasses.tags;
themodel=load('fullcvmodel.mat');
themodel=themodel.modelcv;

% initialize variables
n=size(theclasses,2);
nrobjs=0;
thesum=0;
priceMap=getPriceMap;

I= imread(impath);

% bw final result after segmenting
medB=getMedianBackground();
fin=getForeground(I,medB);
```

```matlab

[labels,nan]=bwlabel(fin);
rp=regionprops(labels);
smalls=find([rp.Area]<300);

% remove small areas
for i=1:size(smalls,2)
   coord=round(rp(smalls(i)).BoundingBox);
   fin(coord(2):coord(2)+coord(4),coord(1):coord(1)+coord(3))=0;
end


bigs=find([rp.Area]≥300);

figure(1);
imshow(I);
f2=figure(2);
set(f2, 'Position', [100, 100, 1049, 895]);

for i=1:size(bigs,2)
   nrobjs=nrobjs+1;
   bBox=rp(bigs(i)).BoundingBox;
   img = imcrop(I, bBox);
   bwimg = imcrop(fin, bBox);
   figure(1);
   rectangle('Position', [bBox(1),bBox(2),bBox(3),bBox(4)],...
  'EdgeColor','r','LineWidth',2 )
   num=num2str(nrobjs);
   % get featue vector and make prediction
   props=getfeatures(img,bwimg);
   prediction=predict(themodel,props);
   value=priceMap(prediction{1});
   thesum=thesum+value;
   textColor = 'black';
   textBackground = 'white';
   text(bBox(1)-5, bBox(2)-5, ...
      prediction, ...
     'Color', textColor, ...
     'BackgroundColor', textBackground, ...
     'HorizontalAlignment', 'Center');

   % plot classified object
   figure(2);
   subplot(5,5,nrobjs);
   imshow(img);
   title([num,prediction]);

end


thesum = thesum/100;
disp(thesum);



end
```