Andres Uribe
au2158

Biometrics Final Project Report

Coin Counter

Introduction

The main objective for the project was to build a program that could count the coins money value in a picture. The work was motivated by the counting and sorting of coins inside a piggy bank. Counting the money inside is a tedious and time consuming task and the purpose of this project is to automate it. The program consists of three parts: image preprocessing; feature extraction; and classification. During image preprocessing, transformations are applied to the image to segment the coins inside it from the background and the coins inside the image are located using the circular Hough transform. During feature extraction a feature vector is created for each coin located during the previous step. The features used were the normalized radial distance edge distribution for different number of bins. For the classification part different classifiers were used and results were compared for each one. This document details the problems found during the construction of the Coin Counter, the ins and outs of each part of the Coin Counter, how the system works, and the results obtained. The Coin Counter was implemented in MatLab using the Statistics and Image Processing toolboxes.

Data Acquisition

The first step of the project was to gather relevant data. The first approach was to take pictures of individual coins at a low resolution. The background used was a table with dots in it, which made it very hard to correctly locate the circles inside the image. Additionally, having individual coins as image samples would not resemble the data with which the Coin Counter has to deal with, namely pictures with various coins. Thus, a different approach was attempted. It's also noteworthy that the coins used in the US Currency have various different flavors. For example, the tail side of a US Quarter can have very different shapes that commemorate each state in the US. For the Coin Counter, this means trouble, so it was decided to narrow down the data to the standard coins. This affects mainly the Quarters and the Nickels. The type of quarters used were the ones with the eagle in the tail side, and the nickels used were the old ones, not the new ones issued in 2010.

The data gathered were pictures taken with a Canon PowerShot SD1000 with a 7.1 MegaPixel resolution. For each coin and side of each coin, a picture with 10 coins was taken. This means that information about 80 coin samples was available. The classes were: quarter back; quarter front; dime back; dime front; nickel back; nickel front; penny back; and penny front. The images can be found inside the `images` folder. The coins' metallic surface makes it hard to have good images, so the lighting environment was controlled and the camera settings had to be tweaked so that the ISO speed was set to 80 and the exposure time was 2 seconds.

Image Preprocessing

In order to acquire coin information from each of the images, some necessary transformations need to be applied. First the image was scaled to a have a height of 1024 pixels. Experiment changing this parameter were performed. This scaling also helped to reduce noise in the image. Next, an image

contrast enhancement operation was performed. This step was necessary to distinguish pennies inside the image, because pennies showed low intensities given that they are made from copper. After this, a threshold was used to build a mask so that only the background was segmented from the actual coins. A threshold of 70 proved to be very useful in practice.

After this basic steps in image manipulation, a routine for localization of the circles inside the image was called. The code for this was taken from Tao Peng' submission[1] in matlab central. Some modifications of the parameters had to be made to accommodate the input image files that were to be used. The routine is able to determine the location and radii of all the circles inside the image. See below for illustations of the Image Preprocessing step.

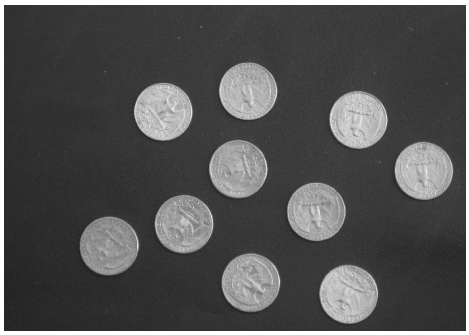

Illustration 1: Before preprocessing step

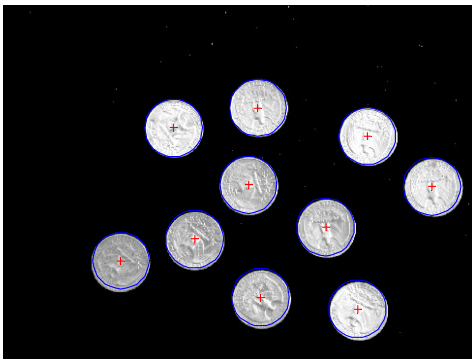

Illustration 2: After preprocessing step

Feature Selection and Extraction

After obtaining the positions and radii of the coins inside the image, the next step was to assemble a feature vector. Naturally, the most basic discriminating feature in coin classification is the radius of the coin. This was considered, but was disregarded because it would make a very simple classifier, and would not be useful in an image in which only one coin was shown. The feature would have to be a ratio of the radius of the coin with respect to the radius of a penny in order to be useful. The focus of the features detected was to use the shape of a coin rather than its size. So the features selected were following the approach of [1] and [2]. SIFT feature were also considered, but were not implemented. This is left as future work.

1 http://www.mathworks.com/matlabcentral/fileexchange/9168

Basically, the edge information of the image will be encoded into a histogram. For each edge pixel, the distance from the pixel to the center of the coin will be measured. This distances will be used to estimate the distribution of the radial distance. For the estimation, the coin's radius is divided into 2, 4, 8, 16, and 32 concentric circular bands (see Illustration 3) and each resulting histogram is normalized. Then each histogram is assembled together to obtain a 62 dimensional feature vector. This feature extraction results, by definition, in rotation and scale invariant features.

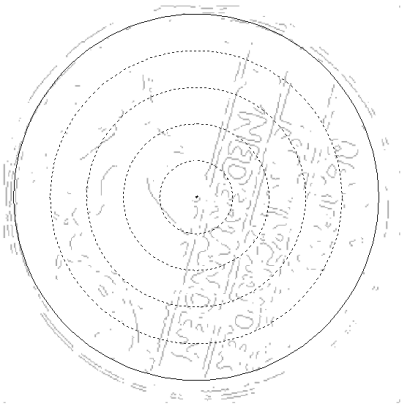

Illustration 3: Circular distance bands (From [1])

Classification

For the classifier, two different classifiers were used: a Bayes Classifier and a k-Nearest Neighbor. Five-fold cross-validation was used as the criteria for the accuracy rate of each classifier. The following table summarizes the results with different height resolutions.

	k	Res.: 1024	Res.: 800	Res.: 1200
Bayes		75.75%	56.25%	76.25%
kNNR	1	55.00%	38.75%	47.50%
kNNR	2	55.00%	38.75%	47.50%
kNNR	3	57.50%	45.00%	53.75%
kNNR	4	56.25%	52.50%	52.50%
kNNR	5	57.50%	51.25%	58.75%
kNNR	6	58.75%	52.50%	58.75%

From the results we can see that the Bayes classifier has a significantly higher accuracy rate. For this reason, the Coin Counter makes use of the Bayes classifier when estimating the amount of money in a given image.

The poor performance of the nearest neighbor classifiers is due to the metric used (Euclidean in this case).

System

The following files are provided are included:

Name: `CircularHough_Grd.m`

Description: function that locates the circles inside an image.

Name: `countMoney.m`

Description: script that attempts to count the money in an image with randomcoins scattered. The script will automatically do the image preprocessing, feature selection and classification and the output will be shown in an image with colored circles identifying the results of the classification.

Name: `createFeatures.m`

Description: function that given the positions and radii of the coins, builds the feature vector as described in the corresponding section above.

Name: `crossValidate.m`

Description: script that runs and and summarizes the different results of the different classifiers used. The output of the script is self-explanatory.

Name: `DrawCircle.m`

Description: function used to draw circles on top of an image.

Name: `getPreprocessedIm.m`

Description: function that returns an image given its file name. The returned image has the transformations and segmentations performed, but no circle detection.

Name: `locateCircles.m`

Description; function that locates the circles inside and image. The parameters for this function have been tweaked to optimize the performance in coin images.

Name: `saveFeaturesToFile.m`

Description: script that builds the training set data from scratch. Asks the user for input labels for each type of coin. The script will create a text file with the labels and feature vectors. The structure is the same as the structure of the training text files used in the previous labs.

Name: `saveToText.m`

Description: used in conjunction with `singleImageFeatures.m` to obtain the image features from full resolution images. This could not be automated because of the high memory usage in the circle detection routine.

Name: `singleImageFeatures.m`

Description: script to create a .mat file with the relevant information of the circle detection performed on a full sized image. Useful because of memory constraints.

Name: `training_data.txt`

Description: training data created from full sized images.

Name: `training_data_1024.txt`

Description: training data created from images downscaled to have 1024 rows.

Name: *.mat

Description: files saved to store important relevant data that took a long CPU time to assemble.

The code output can be seen just by typing

```
>> crossValidate
```

It is also pasted in the appendix A for completion.

The sample output of the Coin Counter in the wild (running the countMoney script) is shown in the illustration below.

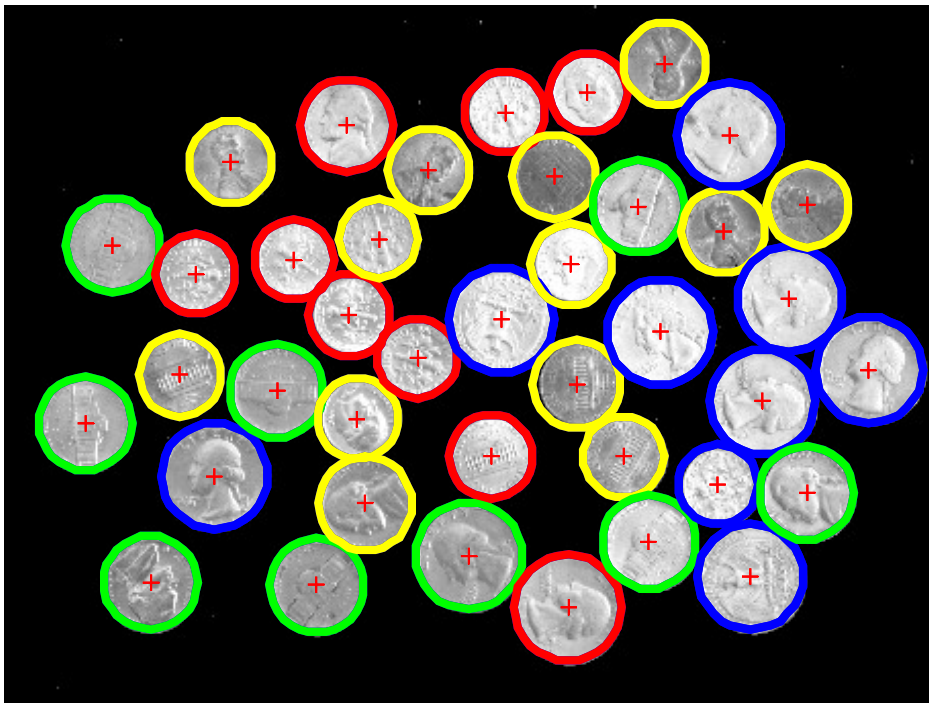

Illustration 4: Coin Counter in the wild. The color legend shows the Coin Counter output.

Analysis of Results

The results show that the classifier depends strongly on the quality and size of the training data. This is expected because in an image where a coin fits in a pixel window of approximately 65 by 65 pixels, there is not a lot of information available. If the pixel windows is reduced in size, then there is even less information, making the problem harder. This is exactly what happens with the Coin Counter.

The second remark is that the image preprocessing step is as important as the classification step, particularly the circle detection. If the coins inside the image can not be accurately located, then the feature extraction creates nonsensical feature vectors, that will basically output garbage.

The nearest neighbor classifiers did not perform nearly as well as the Bayes classifier. This stems out from the metric used to measure the distance between points. Since the feature vectors are a normalized histograms, a suitable metric should have been used, like the Earth Mover's Distance of the Diffusion metric.

The Coin Counter implemented has significant limitations. First of all the input data was all taken in a controlled environment and the distance from the camera to the coins varies very little. The images taken are sharp and have a dark background, which makes the image preprocessing step simpler. Another limitation is the size of the image and the quantity of coins detected. Only a specific number of

coins can fit inside the image. If the camera is moved further away from the coins in order to fit more coins inside the image, significant information will be lost and the classifier will perform poorly. Additionally, the circle detection routine is memory intensive so working with very big images where an estimation of the radii of the coins is not available makes the CPU run out of memory and starts paging. This is the reason why the images were downsclaed to have height resolutions of 800, 1024 and 1200 pixels. Lastly, the Coin Counter performs well only in the presence of non-occluded coins. If the coins were occluded, then the circle detection routine can fail, and even worse, there is not enough information to build the feature vectors.

Nevertheless, the Coin Counter serves as a practical starting point for the stated problem and its performance is acceptable. A better performance could have been achieved by using images where the coins are bigger with respect to the image. As explained before, having a 65 by 65 pixel image of a coin is very limiting.

Conclusions

A Coin Counter program was implemented in MatLab using the Statistics and Image Processing toolboxes. The input image was preprocessed, segmenting the background from the image and locating the circles inside it. Then, feature vectors were constructed using the radial distance edge distribution with different number of bins. Finally, a Bayes classifier was used to obtain a 76% of accuracy in detecting coin images.

A multiclass SVM approach can be used to identify the coin classes. This can also be used as a starting point so that the Coin Counter does not consider coins from a different currency. The Coin Counter developed serves as a starting point in low cost “home brewed” coin recognition. The money counted from an image gives a very rough first estimate of the amount of money inside it.

Apendix A – Output Code

The output of running the crossValidate script:

```
Bayes - Accuracy rate for the 1 attempt: 0.8125
Bayes - Accuracy rate for the 3 attempt: 0.75
Bayes - Accuracy rate for the 5 attempt: 0.5625
Bayes - Accuracy rate for the 7 attempt: 0.75
Bayes - Accuracy rate for the 9 attempt: 0.8125
The average accuracy rate for the Bayes classifier was: 0.7375
Now using nearest neighbor algorithm
K = 1
kNNR - Accuracy rate for the 1 attempt: 0.4375
kNNR - Accuracy rate for the 3 attempt: 0.5
kNNR - Accuracy rate for the 5 attempt: 0.625
kNNR - Accuracy rate for the 7 attempt: 0.5625
kNNR - Accuracy rate for the 9 attempt: 0.625
The average accuracy rate for the kNNR classifier was: 0.55
```

```
K = 2
kNNR - Accuracy rate for the 1 attempt: 0.4375
kNNR - Accuracy rate for the 3 attempt: 0.5
kNNR - Accuracy rate for the 5 attempt: 0.625
kNNR - Accuracy rate for the 7 attempt: 0.5625
kNNR - Accuracy rate for the 9 attempt: 0.625
The average accuracy rate for the kNNR classifier was: 0.55
K = 3
kNNR - Accuracy rate for the 1 attempt: 0.4375
kNNR - Accuracy rate for the 3 attempt: 0.625
kNNR - Accuracy rate for the 5 attempt: 0.625
kNNR - Accuracy rate for the 7 attempt: 0.5
kNNR - Accuracy rate for the 9 attempt: 0.6875
The average accuracy rate for the kNNR classifier was: 0.575
K = 4
kNNR - Accuracy rate for the 1 attempt: 0.5
kNNR - Accuracy rate for the 3 attempt: 0.5625
kNNR - Accuracy rate for the 5 attempt: 0.5
kNNR - Accuracy rate for the 7 attempt: 0.5
kNNR - Accuracy rate for the 9 attempt: 0.75
The average accuracy rate for the kNNR classifier was: 0.5625
K = 5
kNNR - Accuracy rate for the 1 attempt: 0.5
kNNR - Accuracy rate for the 3 attempt: 0.5625
kNNR - Accuracy rate for the 5 attempt: 0.5
kNNR - Accuracy rate for the 7 attempt: 0.625
kNNR - Accuracy rate for the 9 attempt: 0.6875
The average accuracy rate for the kNNR classifier was: 0.575
K = 6
kNNR - Accuracy rate for the 1 attempt: 0.5
kNNR - Accuracy rate for the 3 attempt: 0.625
kNNR - Accuracy rate for the 5 attempt: 0.5
kNNR - Accuracy rate for the 7 attempt: 0.6875
kNNR - Accuracy rate for the 9 attempt: 0.625
```

```
The average accuracy rate for the kNNR classifier was: 0.5875
K = 7
kNNR - Accuracy rate for the 1 attempt: 0.4375
kNNR - Accuracy rate for the 3 attempt: 0.5625
kNNR - Accuracy rate for the 5 attempt: 0.5
kNNR - Accuracy rate for the 7 attempt: 0.625
kNNR - Accuracy rate for the 9 attempt: 0.5625
The average accuracy rate for the kNNR classifier was: 0.5375
K = 8
kNNR - Accuracy rate for the 1 attempt: 0.4375
kNNR - Accuracy rate for the 3 attempt: 0.625
kNNR - Accuracy rate for the 5 attempt: 0.5
kNNR - Accuracy rate for the 7 attempt: 0.625
kNNR - Accuracy rate for the 9 attempt: 0.5625
The average accuracy rate for the kNNR classifier was: 0.55
```

References

[1]. Maaten LJPVD, Boon PJ. COIN - O - MATIC : A fast system for reliable coin classification. *Proceedings of the Muscle CIS Coin Competition Workshop*. 2006:07-18.

[2]. N M, Penz H, Rubik M, et al. Dagobert – A New Coin Recognition and Sorting System. *Circulation*. 2003:10-12.