Coin Detection

Anthony Sutardja, Andrew Fang

I. Introduction

Counting change is a tedious task. There are currently two widely adopted methods for counting change: (1) to dump your change into a machine that counts it for you at a fee, or (2) to manually count the coins. We believe that this process can be simplified using computer vision. Using a python implementation of opencv and a neural network trained on a hand-collected dataset of coins. Current methods of coin detection require Current methods of coin detection require fixed radii, so the difference in coin size between images cannot vary much. Accuracy is great when pictures are uniform, but parameters must be tweaked between images where the coins are large versus images where the coins are small. In addition, current algorithms are tested against coins placed on similar backgrounds, and internal values are optimized for such backgrounds. These approaches do not successfully detect coins against varied backgrounds. In this paper, we demonstrate methods and techniques to successfully extract coins from an image.

II. Method

A. Masking

The first step is to mask out the coins in the image so we get a binary coin/not-coin classification for each pixel in the image. We start with a rough estimation, and we iteratively refine and improve the mask. It really helps to resample the image to get rid of high-frequencies, so we begin by downsizing the image so it's biggest dimension is 500px long or wide. From this, we map the image to the HSV colorspace, and we extract the hue and value layers. Extensive testing has indicated that coins really stand out against red, purple, or orange backgrounds in the hue layer, whereas the value layer works best for all other backgrounds. (see Fig. 1, 2). Therefore, we take a sample of the pixels from the border of the image to determine the dominant background color, and we use this to determine whether to proceed with the hue or value layer.

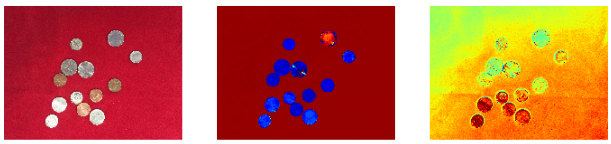


Fig. 1. (left to right): Coins on a red background, hue layer, value layer

After choosing the right layer to maximize coin/background differentiation, we perform a median blur on the layer, and then we apply an adaptive threshold[1] using the gaussian algorithm. This gives us a pretty good estimate, but a better mask is needed for even better segmentation. Thus, we use the grab cut algorithm[2] to refine the mask. After 5 iterations, the algorithm classifies each pixel of the image as either foreground or background. We take this as our new mask.

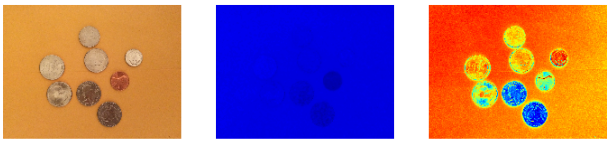


Fig. 2. (left to right): Coins on a yellow background, hue layer, value layer

Fig. 3. (left to right): Original image, adaptive threshold mask, grab cut mask

B. Segmentation

Now that we have the masks, we need to extract out each of the coins. We explored many methods for finding circles in the mask. First, we attempted to use Hough Circles[3], which is the approach suggested by Huber-Mrk (et al.) and Hassoubah (et al.). Turns out, this appraoch requires much adjustment of parameters which vary based on background colors and coin-sizes. This was not satisfactory, as we aim to build a system that will work on any background, and any size of coin. Instead, we employ an algorithm developed by Atherton and Kerbyson that allows for size invarient circle detection. This algorithm finds centers and radii of the circles in the image. We use this to segment out each of the coins (grab the bounding boxes of height and width $2 * r$ from $x = c_x - r$, $y = c_y - r$). (see Fig. 4). Once we've detected all the coins, we split them into individual images for processing and classification.

C. Data Preparation

In order to train our coin classifier, we needed a labeled dataset of individual coin images. Unfortunately, we were unable to find a publicly accessible United States coin dataset.

Hence, we created our dataset by manually labelling each image. For each of the coin images that we gathered, we labeled the coin as either a quarter, dime, nickel, penny, or none of the above. In addition to these categories, we also

[1] Bradski, et al.

[2] Rother et al.

[3] Yip et al.

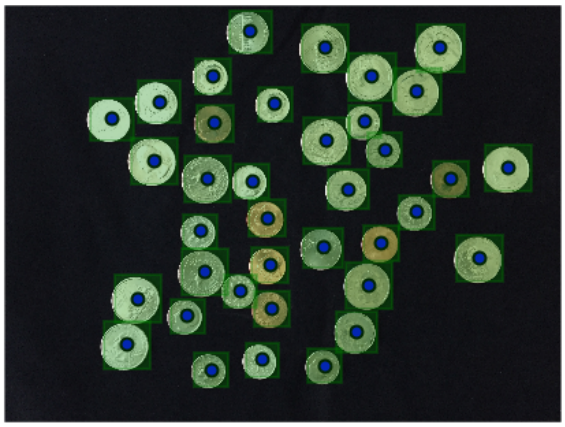


Fig. 4. The bounding boxes found for a given image.

Fig. 5. The segmented images that our algorithm produces.

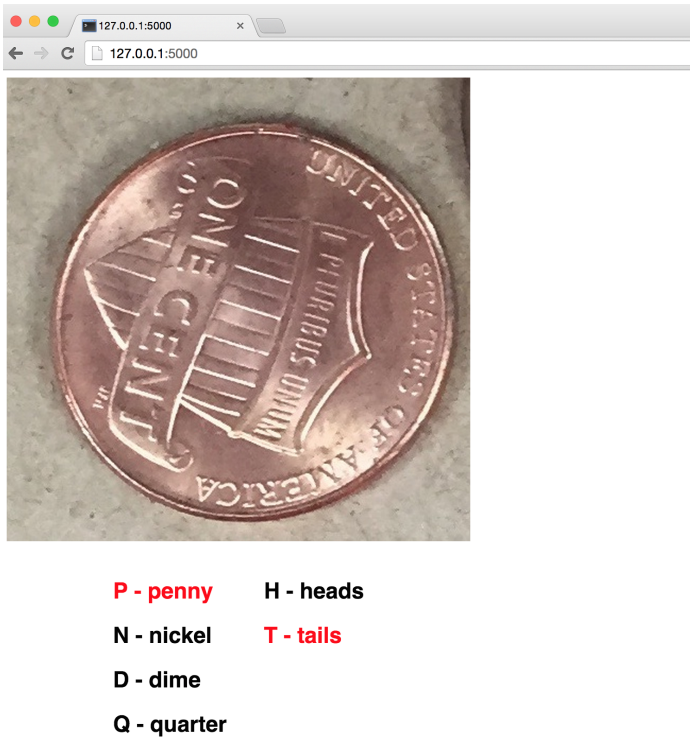


Fig. 6. The user interface for assisting in manual coin classification.

label the face of the coin: heads or tails. In order to streamline this process, we built a web application with a user interface based on keyboard shortcuts to help us manually classify the 3600 coin faces (see Fig. 6).

The distribution of the dataset was scewed towards quarters and pennies since we simply had more of these types of coins at hand. Hence, we chosen a random sample of an equal number of each coin for the training and test set.

D. Convolutional Neural Network

Previous methods performed classification only after preprocessing the image in some manner. Our approach was to let the convolutional neural network do all the feature extraction.

There are only 3 preprocessing steps that we take. First, we resize each image to a 256×256 image. We then subtract the mean image (generated during training) from the image undergoing classification (See Fig 7). Lastly, we crop the image to a 227×227 image.

We used Caffe for building a convolutional neural network (CNN). Our neural network architecture is a replica of the ImageNet architecture, except for the last layer having 9 nodes instead of 1000. Although the end goal was to classify a penny, nickel, dime, or quarter, the CNN's accuracy is greatly boosted when we classify the heads versus tails. We found that training on 5 classes (penny, nickel, dime, quarter, and no coin) yielded extremely poor accuracies, where our best training instance out of the 20 yielded around 18 percent accuracy after 250,000 iterations.

Rather, we architect our CNN to have 9 output classes, where each coin is separated based on heads and tails and the 9th class being the "no coin" class. This architecture allowed us to ultimately achieve a mean accuracy of 91 percent.

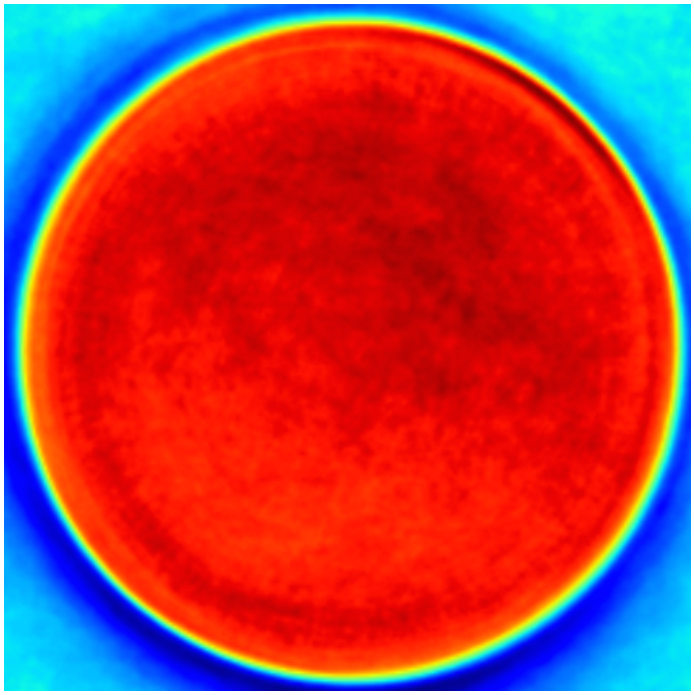
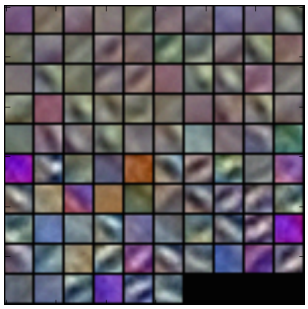
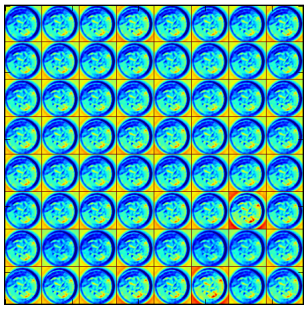


Fig. 7. The blue channel of the coin's image mean.

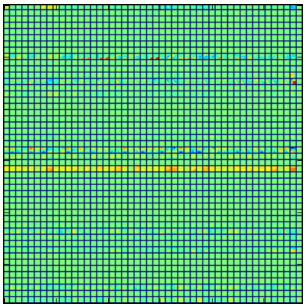
We also experimented with training modifications of ImageNet with fewer parameters and a smaller image size, but this ultimately failed to produce ussable results (yielding 25 percent accuracy after 250,000 iterations on all our training instances).



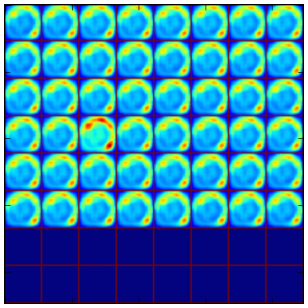
(a) The first layer's convolution filters



(b) The first layer's convolution output of a quarter



(c) The second layer's convolution filters



(d) A subset of the second layer's convolution output of a quarter

Fig. 8. The convolutional neural network was able to learn what looks to be useful edge filters at various orientations.

E. Results

Our results surprised us. Without any preprocessing, we were able to achieve rather good results.

We separated our results into two sets: one set that validated our classifier with images that contained just coins, and another set that validated our classifier with images that contained coins and bad segmentations that contained just the background of the image.

The table below shows our accuracy in detecting each type of coin face. The prefix represents the first letter of the type of coin (**Q**uarter, **P**enny, **N**ickel, **D**ime) and the suffix represents the first letter of the face of the coin (**H**eads or **T**ails).

Coin Face	Accuracy	Most common mistake
QT	0.758	QH
QH	0.924	QT
PT	0.935	PH
PH	0.930	QT
NH	0.806	QH
NT	0.910	NH
DH	0.874	QH
DT	0.878	QH

From these results, we notice that the most common mistake for each of the coin faces is mostly the other face of the same coin.

From this observation, we post-process our classifier results to consolidate the heads and tails of each coin prediction to be of just one particular value. After consolidation, our accuracy is as follows:

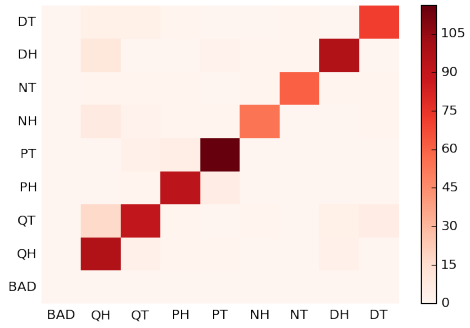


Fig. 9. The confusion matrix of the validation set without bad segmentations. The horizontal axis is the actual coin class, and the vertical axis is the predicted coin class.

Coin	Accuracy
Quarter	0.924
Penny	0.982
Dime	0.875
Nickel	0.865
Overall	**0.919**

We also measured accuracy across our validation set that contained images of poorly segmented images. As one might expect, this yielded poorer results with an overall classification accuracy of 84 percent.

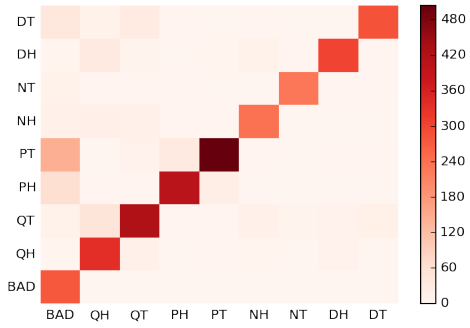
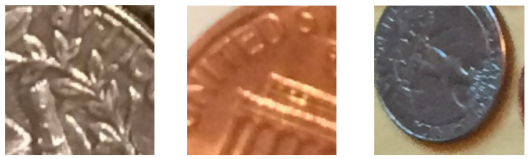


Fig. 10. The confusion matrix of the validation set with bad segmentations. The horizontal axis is the actual coin class, and the vertical axis is the predicted coin class.

On further inspection of the misclassifications, we noticed something interesting. Some of the coins that the neural network classified were images that we manually classified as "not a coin". However, upon further inspection, these images were actually an extremely small fragment of the coin that the classifier predited. In other words, the neural network actually outperformed us in labeling training data with the actual correct labels as shown in Figure 11.

Our inspection of these results also seem to indicate that camera angle skew does not seem to effect the classification results.



(a) Dime (b) Penny (c) Quarter

Fig. 11. These image fragments are successful detections by the neural network classifier, but not by the humans who labeled the data.

F. Web application

In order to add the finishing touch to our coin classifier, we packaged our segmentation algorithm and neural network consolidated classification into a web application. The web application is accessible via both a PC and a mobile phone's web browser. The user simply uploads a picture of a bunch of coins on a surface, and our application identifies the coins and totals the final amount of money on the table.

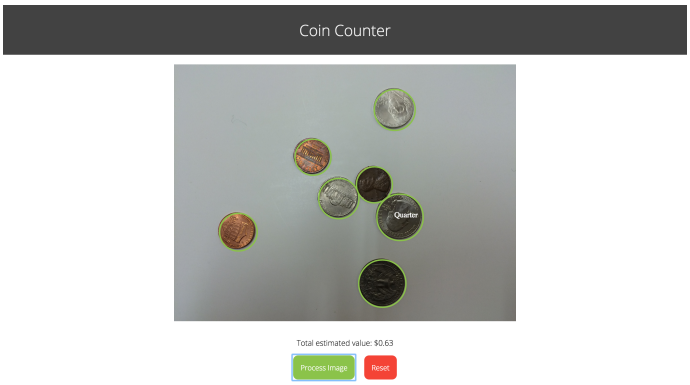


Fig. 12. A screenshot of our web service that classifies coins from a camera image. The user is able to hover over each coin to see it's predicted value.

III. Conclusion

Overall, we are very pleased with our results. We were able to achieve 92 percent accuracy with our perfectly segmented dataset, and 84 percent accuracy with our poorly segmented dataset. For production, we would have to assume the accuracy of the poorly segmented dataset since our coin segmentations are not perfect.

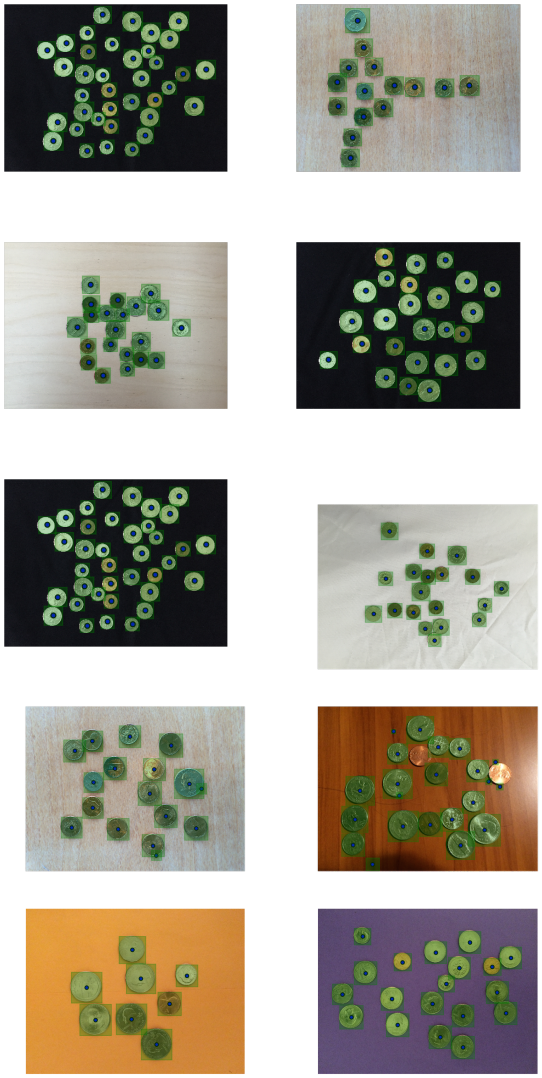
That being said, our accuracy is not yet good enough to make this a useful application. For the camera images that we did upload to our webservice, we noticed that every time at least one out of the many coins in the image were misclassified.

Future work needs to focus on tuning the segmentation algorithm such that it is more robust in detection circles.

Appendix

Detected Coins

Here are more examples of detected coins



Acknowledgment

Thanks so much to Jitendra Malik and the teaching assistants for a fantastic class.

References

[1] Bradski, Gary, and Adrian Kaehler. Learning OpenCV: Computer vision with the OpenCV library. " O'Reilly Media, Inc.", 2008.
[2] Rother, Carsten, Vladimir Kolmogorov, and Andrew Blake. "Grabcut: Interactive foreground extraction using iterated graph cuts." ACM Transactions on Graphics (TOG) 23.3 (2004): 309-314.
[3] Yip, Raymond KK, Peter KS Tam, and Dennis NK Leung. "Modification of Hough transform for circles and ellipses detection using a 2-dimensional array." Pattern Recognition 25.9 (1992): 1007-1022.
[4] Huber-Mrk, Reinhold, et al. "Automatic Coin Classification and Identification." Advances in object recognition systems (2012): 127.
[5] Hassoubah, Rawan S., Amel F. Aljebry, and Lamiaa A. Elrefaei. "Saudi riyal coin detection and recognition." Image Information Processing (ICIIP), 2013 IEEE Second International Conference on. IEEE, 2013.
[6] Atherton, Tim J., and Darren J. Kerbyson. "Size invariant circle detection." Image and Vision computing 17.11 (1999): 795-803.