US Coin Classifier and Counter: Update

By Brazos Fitch and Yves Ngenzi

Problem Statement

Given an assortment of mostly U.S. coins, can we create a Python script which can recognize the objects and sort the currency into different classes, finally returning a total amount.

=

$15.69

Classification Categories

These are the classes we expect to see:

- Quarter
- Dime
- Nickel
- Penny
- Unknown

The Unknown category is intended to collect all objects that are not a U.S. coin

What features will be used?

- Pixels (in case of an image)

Similar to the Optical Character Recognition

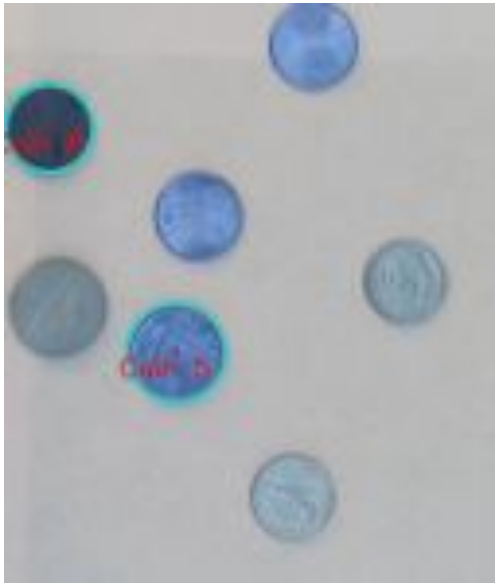
This of course would require new categories for the heads and tails sides of the coins

Design of software

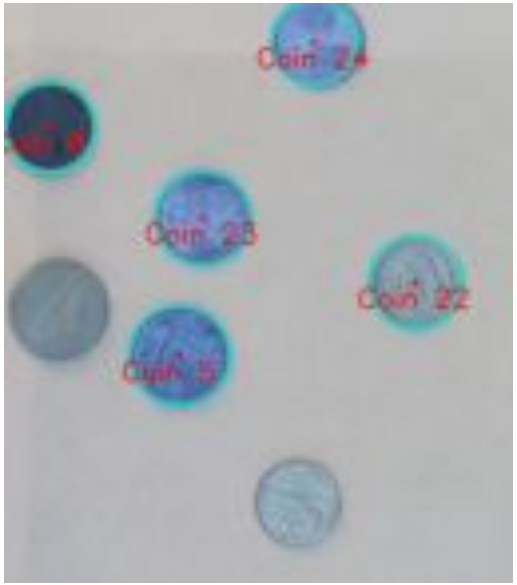
1. Image Processing - segmentation
    a. Here we use software such as OpenCV for object recognition, and to crop and copy an image of a single coin
    b. Resize the image and perform other processes to achieve best possible results. (i.e. rotation, color scale, contrast)
2. Classification
    a. Then use a classifier to classify the coin in the new image

Image Processing: HoughesCircles

1.



2.



3.

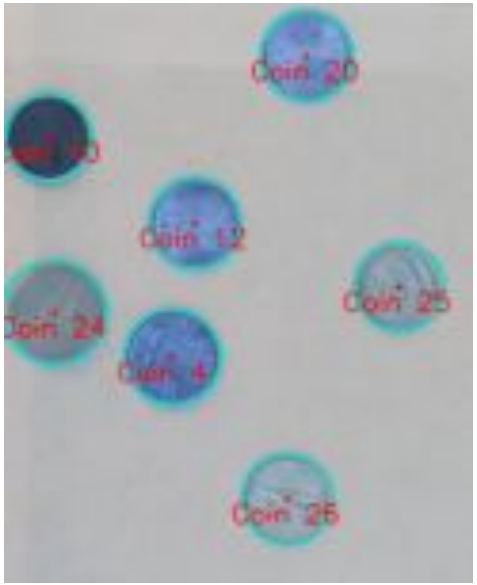
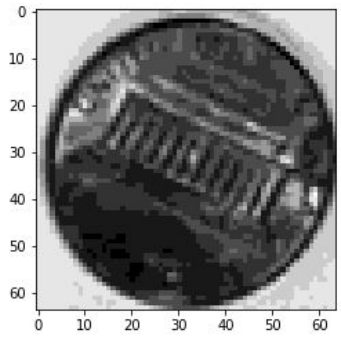


Image Processing: Coincutter

- Accepts image name
  - “Example.jpg”
- Finds all of the coins in the image and saves their locations and radii
- Returns images of all individual coins in a numpy array which has dimensions of (number of coins x 4096) which can be reshaped into a 64x64 image

```
[88] X_test = coincutter("attemptcoins1.jpg")
     test = X_test[0].reshape((64,64))
     plt.imshow(test,cmap = plt.cm.binary)
     plt.show()
```




```
[89] X_test = coincutter("attemptcoins2.jpg")
```

```
[90] test = X_test[0].reshape((64,64))
     plt.imshow(test,cmap = plt.cm.binary)
     plt.show()
```

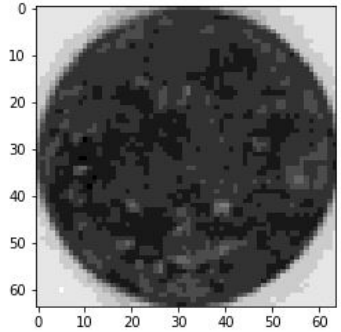
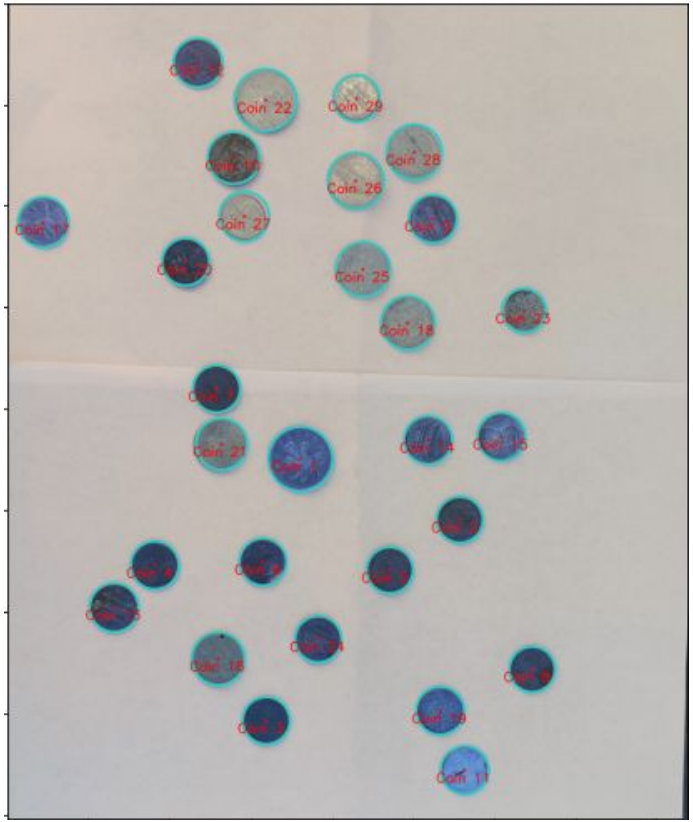


Image Processing: Examples



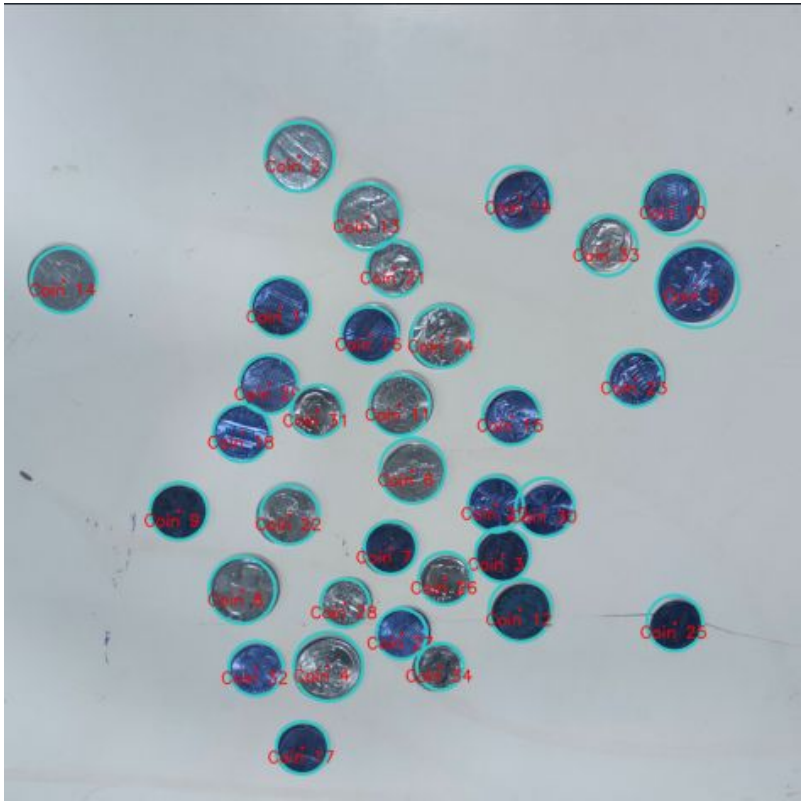


Image Processing: What's the next step?

1. Use the coin cutter function to create a large training and test dataset of real coins
2. Find a way to highlight important features (pixels)
3. Find ways to make the different classifiers more accurate through use of data preprocessing

Image classification

Downloaded a lot of images from google and gave them names.

Created the arrays of images

Factorised the images

Change image dimensions for the classfiler

Tried three classifiers:

Neural network

Naive bayes and Nearest neighbor

Some images from google and their names

```
[ ] #Lets print some of the data we have of coins

    Data_names = os.listdir(Data)
    print(Data_names[:10])
    Data_names.sort()
    print(Data_names[:10])
```

```
['Q37 copy.jpeg', 'Q38 copy.jpeg', 'Q45.jpeg', 'Q40 copy.jpeg', 'Q30.jpeg', 'Q5 copy.jpeg', 'Q46.jpeg', 'Q29.jpeg', 'Q28.jpeg', 'Q25.jpeg']
['D0 copy.jpeg', 'D1 copy.jpeg', 'D10.jpeg', 'D10.jpg', 'D11.jpeg', 'D11.jpg', 'D12 copy.jpeg', 'D13 copy.jpeg', 'D13.jpg', 'D14 copy.jpeg']
```

```
# read all images and show one
for img in files:
   img_array=cv2.imread(img,cv2.IMREAD_GRAYSCALE)
   plt.imshow(img_array,cmap="gray")
   plt.show()
   break
```

Classifiers performance

Neural Networks: did well with the score of 8.5%

And I tested 47 coins which were

13 quarters,

25 pennies,

4 Nickles, and

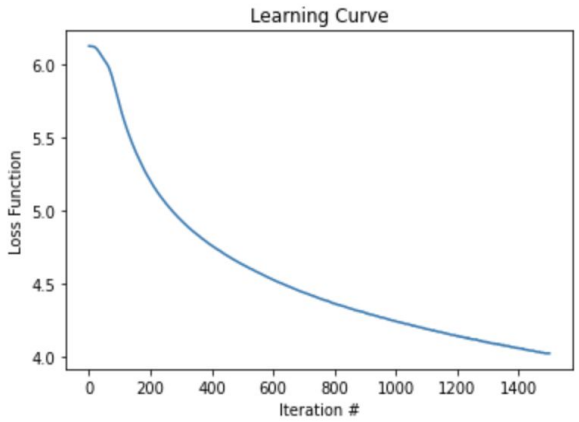
5Dimes

```
[ ]  clf.score(V, y0)
```

```
0.08552631578947369
```

```
[ ]  clf.loss_curve_[-30:];
```

```
[ ]  plt.figure()
     plt.plot(clf.loss_curve_)
     plt.title('Learning Curve')
     plt.xlabel('Iteration #')
     plt.ylabel('Loss Function')
     plt.show()
```



Neural network result

```
[ ]  quarters=[]
     Nickles=[]
     Dimes=[]
     Pennies=[]
     for i in H:
       if q in i:
         quarters.append([q])
       elif p in i:
         Pennies.append([p])
       elif n in i:
         Nickles.append([n])
       elif d in i:
          Dimes.append([d])

     print( "The number of quarters are:", len(quarters))

     print( "The number of Pennies are:", len(Pennies))

     print( "The number of Nickles are:", len(Nickles))

     print( "The number of Dimes are:", len(Dimes))
```

```
The number of quarters are: 12
The number of Pennies are: 5
The number of Nickles are: 19
The number of Dimes are: 11
```

Bayes classification

```
[ ] from sklearn import naive_bayes
```

```
[ ] clf1 = naive_bayes.GaussianNB()
    clf1.fit(V,y0)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/naive_bayes.
  y = column_or_1d(y, warn=True)
GaussianNB(priors=None, var_smoothing=1e-09)
```

```
[ ] clf1.score(V, y0)
```

```
0.9890350877192983
```

```
[ ] from sklearn.neighbors import KNeighborsClassifier
    knn = KNeighborsClassifier()
    knn.fit(V,y0)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.p
  This is separate from the ipykernel package so we can avc
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric
                     metric_params=None, n_jobs=None, n_nei
                     weights='uniform')
```

```
for i in R:
  if qu in i:
    a.append([qu])
  elif pe in i:
    b.append([pe])
  elif ni in i:
    c.append([ni])
  elif di in i:
     e.append([di])

print( "The number of quarters are:", len(a))

print( "The number of Pennies  are:", len(b))

print( "The number of Nickles are:", len(c))

print( "The number of Dimes are:", len(e))
```

```
The number of quarters are: 9
The number of Pennies  are: 10
The number of Nickles are: 15
The number of Dimes are: 13
```

Challenges

Images have different size especially the ones we got from google which makes it hard for pixels

Good news

Try to cut by radius and use that

The cut of the radius will also be applied to testing data

Nearest Neighbor

```
[ ] from sklearn.neighbors import KNeighborsClassifier
    knn = KNeighborsClassifier()
    knn.fit(V,y0)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launc
  This is separate from the ipykernel package so we ca
KNeighborsClassifier(algorithm='auto', leaf_size=30, m
                     metric_params=None, n_jobs=None,
                     weights='uniform')
```

```
[ ] knn.score(V, y0)
```

```
0.2565789473684211
```

```
[ ] u=[]
    v=[]
    w=[]
    m=[]

    for i in Z:
      if q in i:
        u.append([q])
      elif p in i:
        v.append([p])
      elif n in i:
        w.append([n])
      elif d in i:
        m.append([d])

    print( "The number of quarters are:", len(u))

    print( "The number of Pennies are:", len(v))

    print( "The number of Nickles are:", len(w))

    print( "The number of Dimes are:", len(m))
```

```
The number of quarters are: 5
The number of Pennies are: 2
The number of Nickles are: 13
The number of Dimes are: 27
```

Training Data

Training data can be made from using photos of individual coins found around the house or even exchanged at a bank

Training data (images) can even be found on Google and could undergo image preprocessing steps before being used to train the classifier

1 going to google and download bunch of images like a 1000 images and use these image for deep learning.

Process:

gooling the coin image that we want, and download the images

challenge :

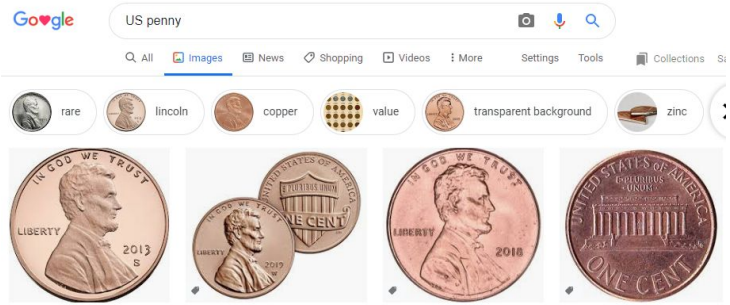
Unsure how we can use these images to create training pixels for other classifiers like Bayes, nearest neighbor, or decision tree.

Training data from coin images

Training data can be made from using photos of individual coins found around the house or even exchanged at a bank

Training data (images) can even be found on Google and could undergo image preprocessing steps before being used to train the classifier

A large dataset can be made with the y dimension being items and the x dimension being features and coin type



Pattern recognition techniques and why.

Deep learning is our number one go if we use google images because it can recognize images and their shapes easily

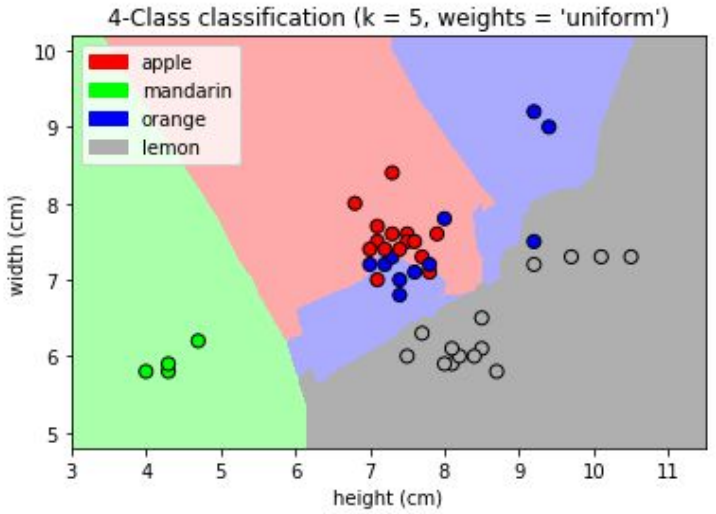
We can also use Bayes Naive classifier which is depend on conditional probability. For example, if a coin size is 17.91mm then classify that as a dime.

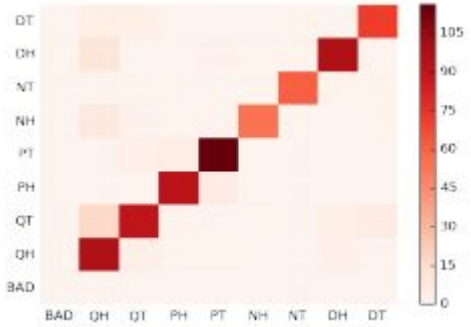
Design choice we have to make for both classifier is adjusting the pixels while we create training data.

Another plan is to find a way to recognize name of coins written on them and then classify those words. However, the words are at different places.

Results: what to know and see

- What percentage of what coin was classified correctly?
- Which classification was most difficult?
- How much money was there?
- What was the decision boundary?
- What was the decision boundary between the two most similar coins?





Schedule/Plan

Week 1

- Create training dataset
- Work with object recognition and OpenCV

Week 2

- Begin classifying
- Optimize and find best technique

Week 3

- Polish the code
- Provide a write-up which displays useful results

Tasks

Brazos - I would like to work on object recognition and image preprocessing.

Yves - I would like to work on finding the online data and train the coins over the already made data and compare with Brazos own data, and if I do not find that I would like to work on creating my own data and share it with Brazos so we can test our coins on it.

References

https://towardsdatascience.com/solving-a-simple-classification-problem-with-python-fruits-lovers-edition-d20ab6b071d2

https://andrewbfang.com/resources/coin.pdf

https://github.com/chen-yumin/euro-coin-classifier

https://github.com/j05t/coin_detector

https://tremaineconsultinggroup.com/opencv-coin-detection-project/