2-D Object Recognition with Open CV using C++

Shikha Tiwari[1], Chaitanya Kalagara[1]

[1]Khoury College of Computer Sciences, Northeastern University
tiwari.shi@northeastern.edu, kalagara.c@northeastern.edu

INTRODUCTION

In this project, we focused on 2-D Object Recognition in which our primary focus was on detecting different objects which were placed on white surface in a translation, scale, and rotation invariant manner. For, object recognition, we will be focusing on primary steps like starting with thresholding (determining foreground and background) and then performing cleanup on images using techniques like erosion and dilation. Once, the images are cleaned we will proceed to finding connected components, there are different algorithms like 2 pass segmentation and region growing for this. Once we have connected components, we can determine different features of these regions which will be used for object recognition based on these properties. The feature generated need to be translation, rotation, and scale invariant. We further created database of these features and assigned labels for each region and then utilized scaled Euclidean distance (as features were heterogenous) to identify the new images. We also implemented Deep network embedding for the thresholded object and utilized cosine distance metric.

PROJECT TASKS

1. **Thresholding:**
   Thresholding is the process of determining foreground and background pixels in the image. For every pixel, the same threshold value is applied. If the pixel value is smaller than the threshold, it is set to 0, otherwise it is set to a maximum value. We have implemented simple thresholding for BINARY and BINARY_INV. In Binary mode if pixel value is greater than threshold it will be set to max_value else it will be set to zero. In Binary_inv it will be opposite. We have implemented both Binary and Binary_Inv mode which can be passed to "createBinaryThresholdImage" function which will give threshold image.Also, before creating thresholdimage we have pre-processed the image using blur function.

   Image 1 -

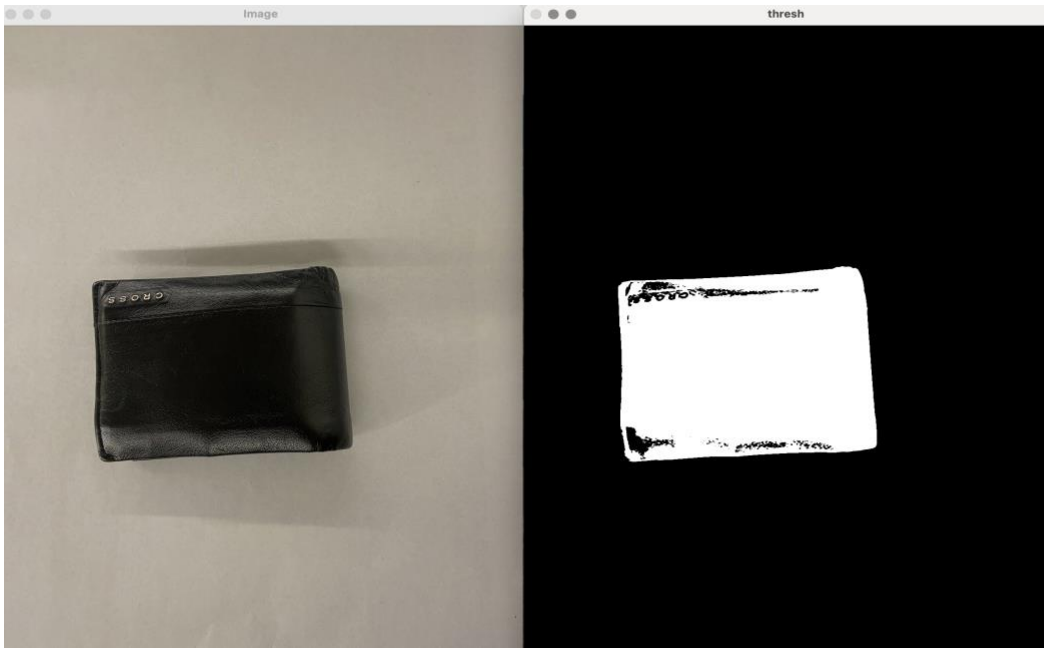


Image 2 -

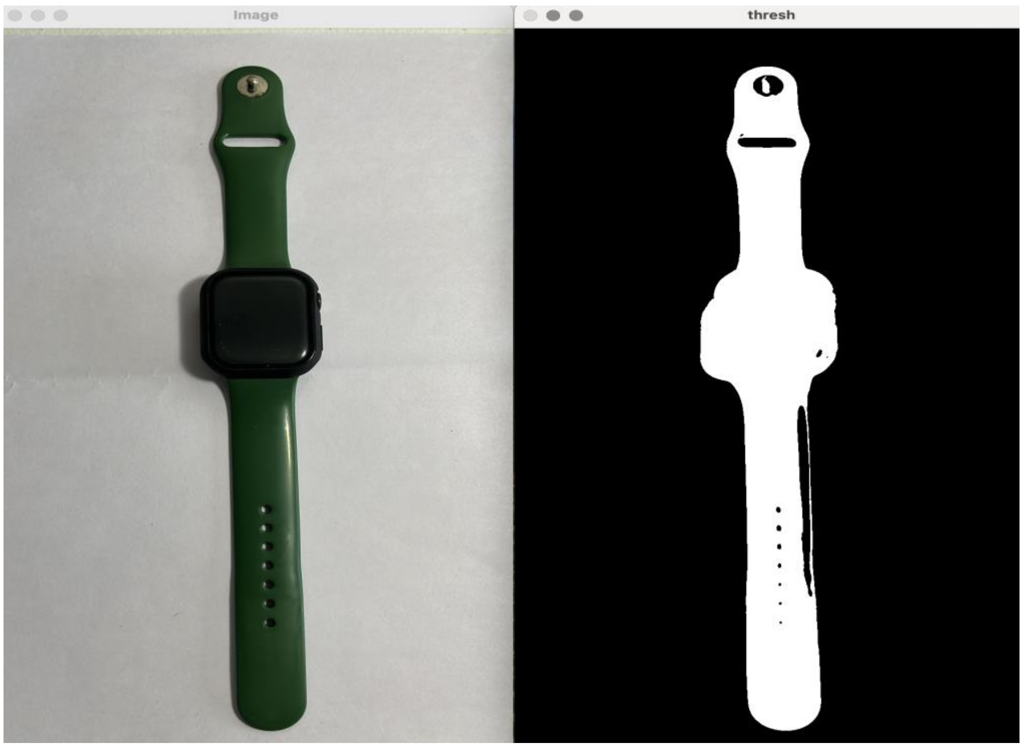
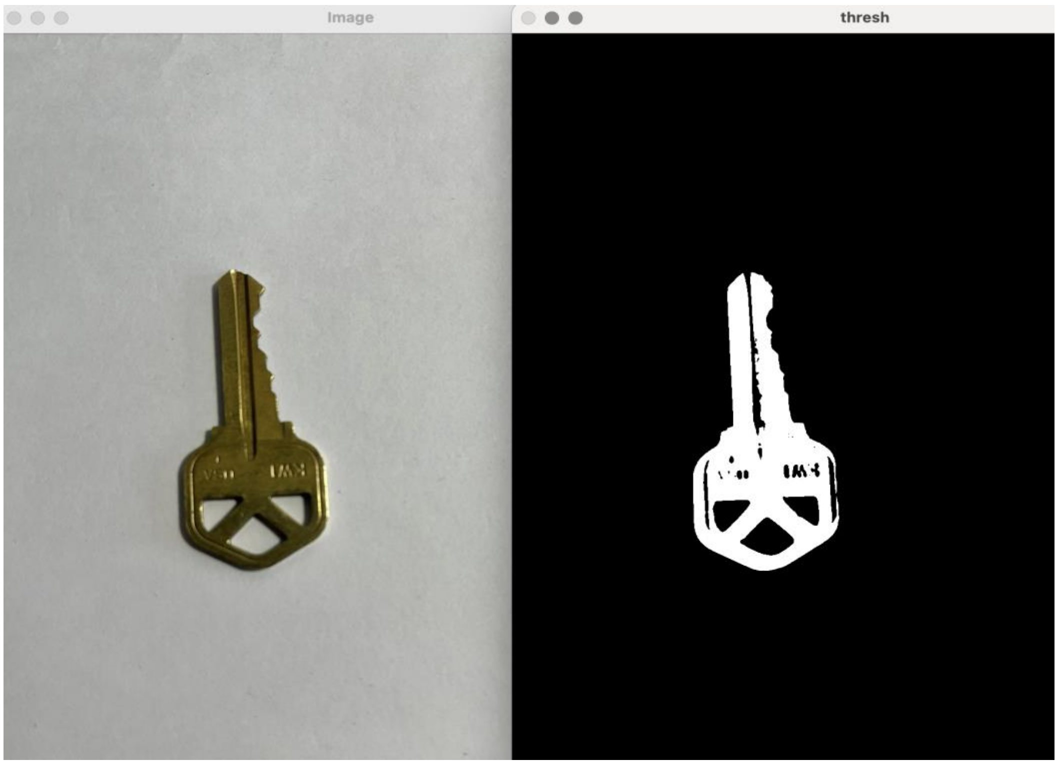


Image 3 -



2. Clean up the binary image:

After thresholding an image, the image may not clear and parts of the image may contain noise. In order to get rid of that noise, we have created morphological functions like erosion and/or dilation. Erosion is basically removal of foreground pixels that touch background pixels, while dilation is making a background pixel touching a foreground pixel into a foreground pixel. Once we have thresholded the image, it will send it to getdilatedImage which would dilate the image, which will then be sent to geterodedImage which will give an eroded image. As per our image we cand decide to call respective function to perform the clean-up operation. We have implemented this from scratch without using opencv function.

Image 1 - Dilation (Kernel size 10) and then performed Erosion (Kernel Size 8)

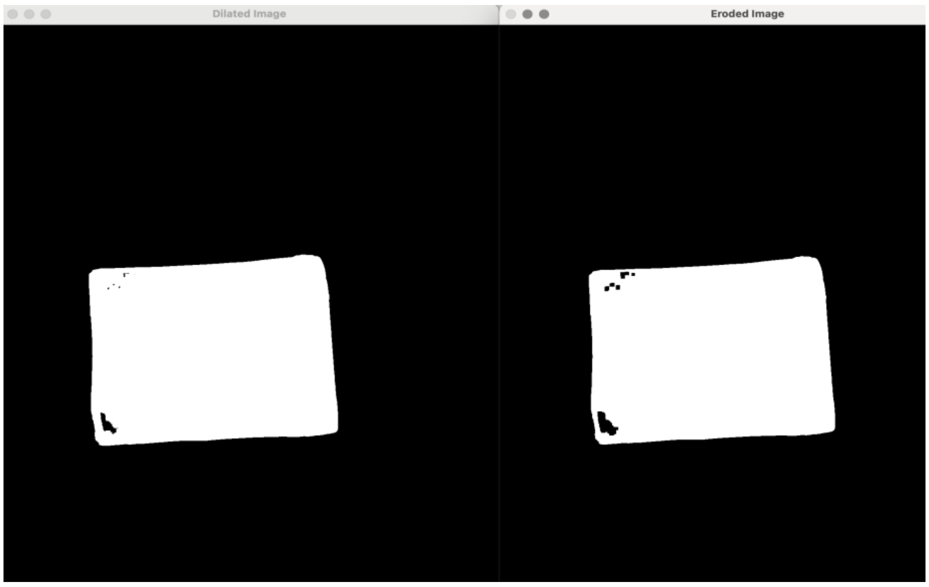


Image 2 - Dilation (Kernel size 15) and then performed Erosion (Kernel Size 8)

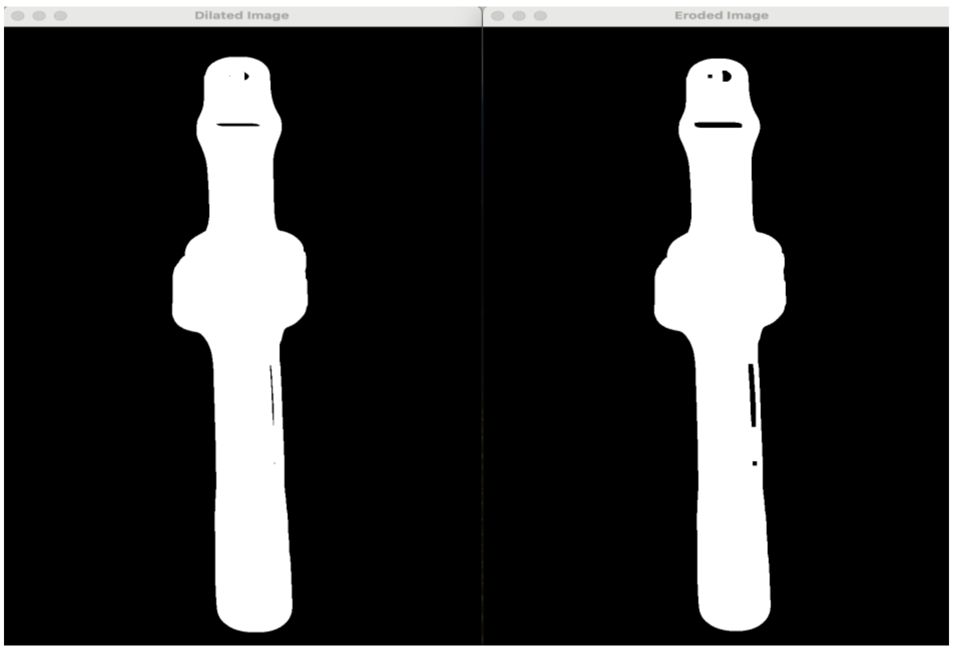
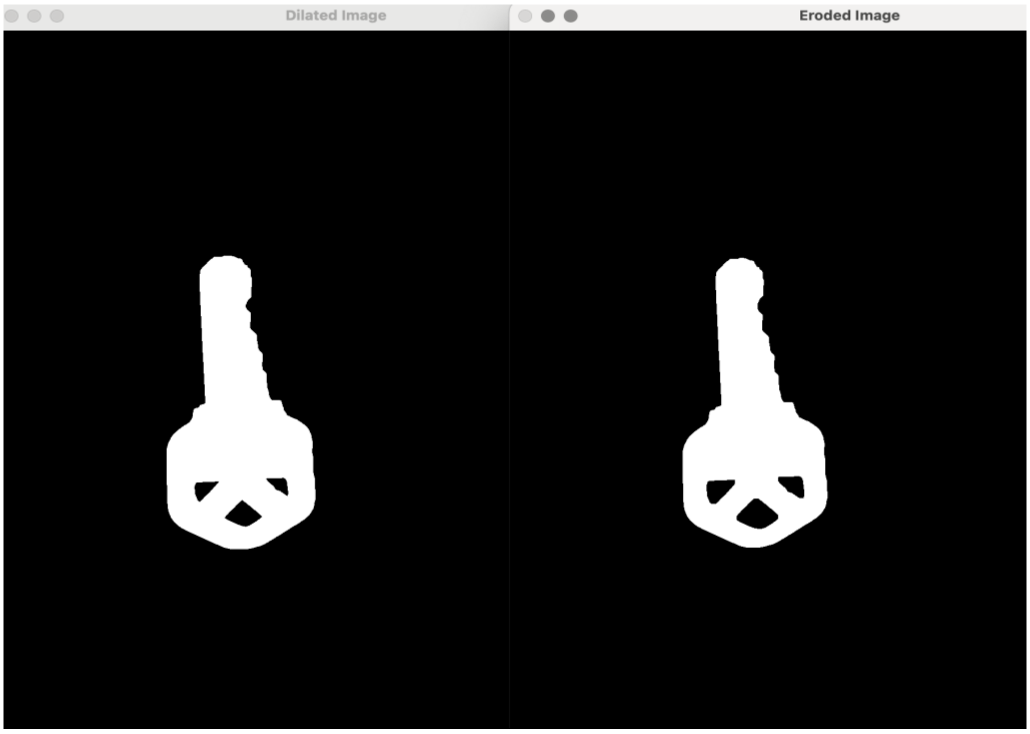


Image 3 - Dilation (Kernel size 10) and then performed Erosion (Kernel Size 8)



3. Segment the image into regions:

Segmentation is the process of dividing an image into different regions. It can be used to identify the similar and different objects/regions in an image. In the project, we send the cleaned-up image to the connected component segmentation algorithm, get the region map and its labels, ignore the small regions in the image and assign a color to each region and display it. It was implemented using “getConnectedComponentswithStats” function in the code.

Image 1-

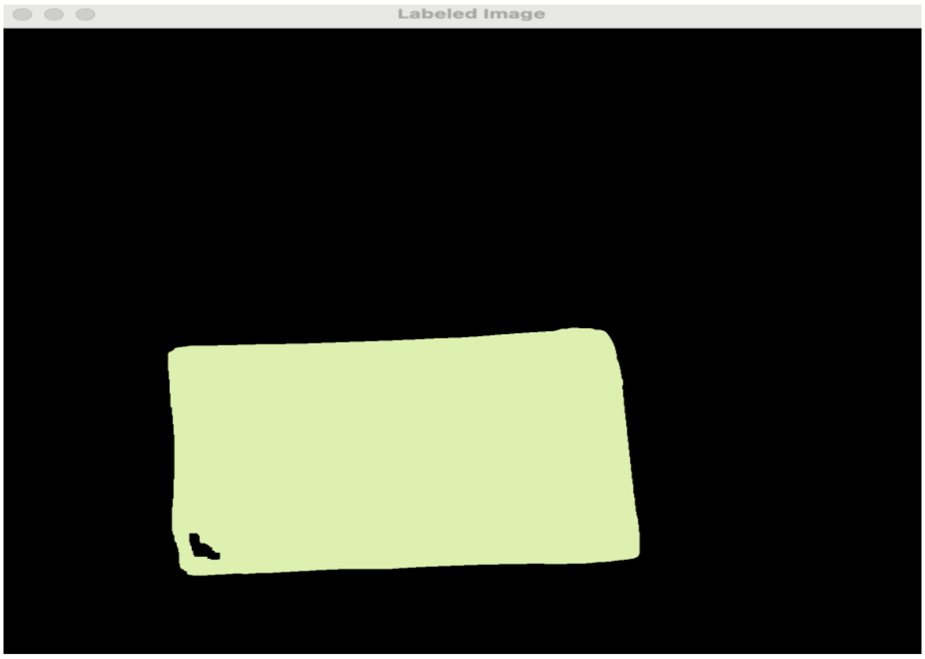


Image 2 -

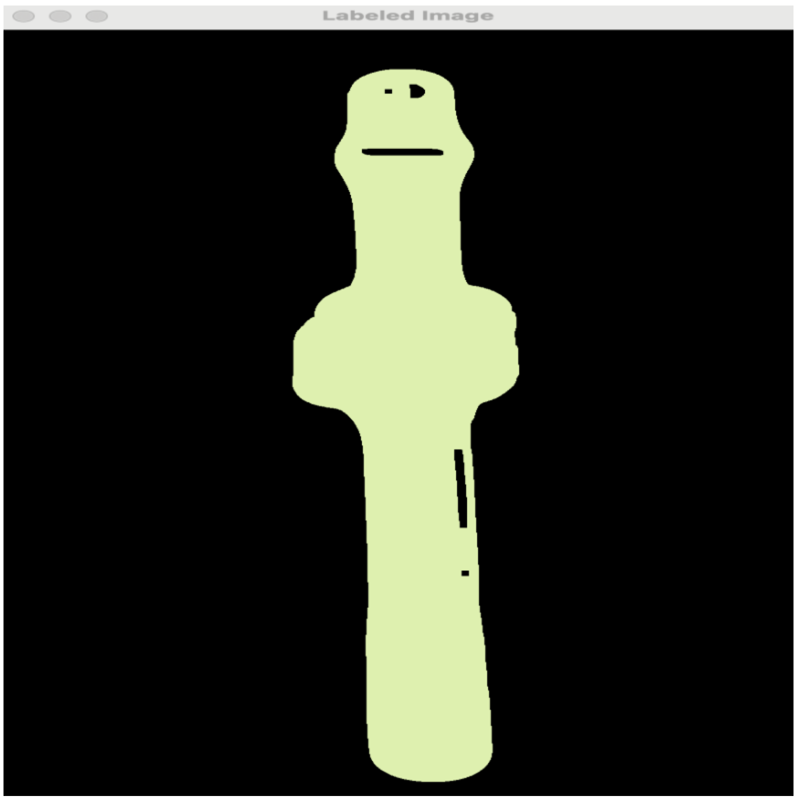
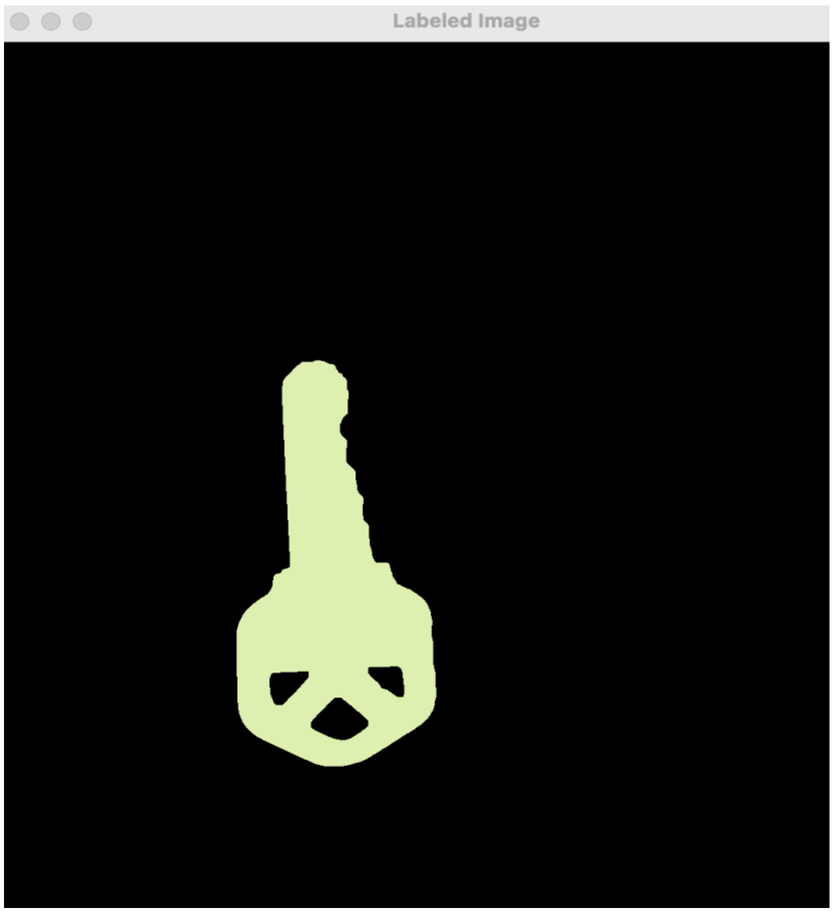


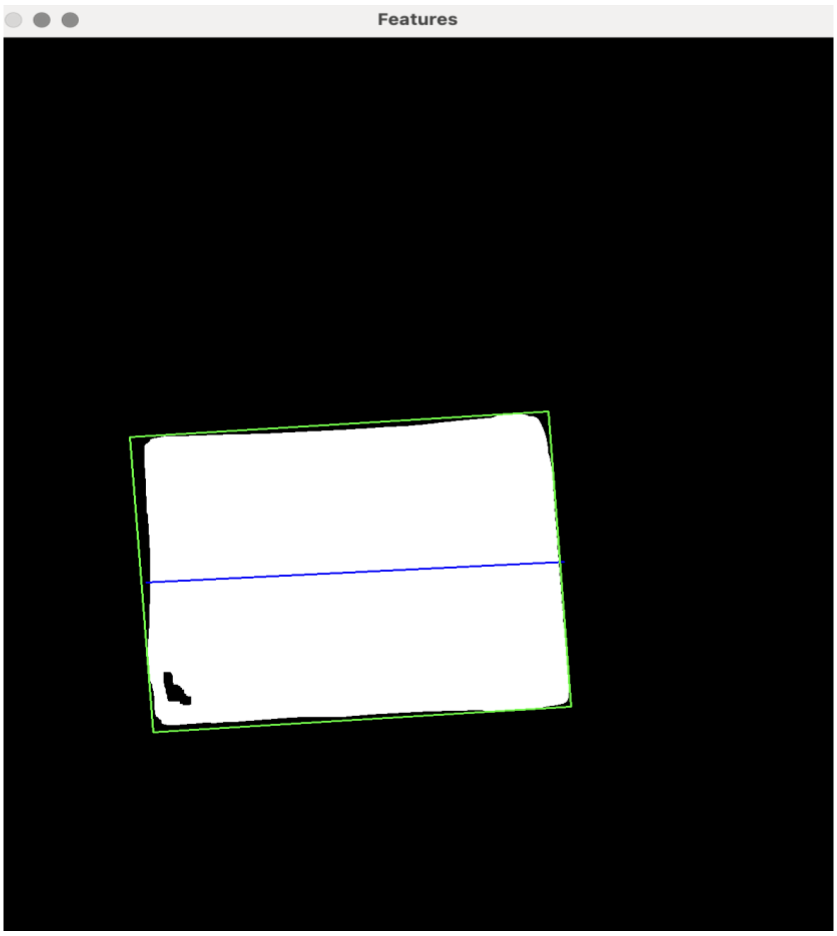
Image 3



4. **Compute features for each major region:**

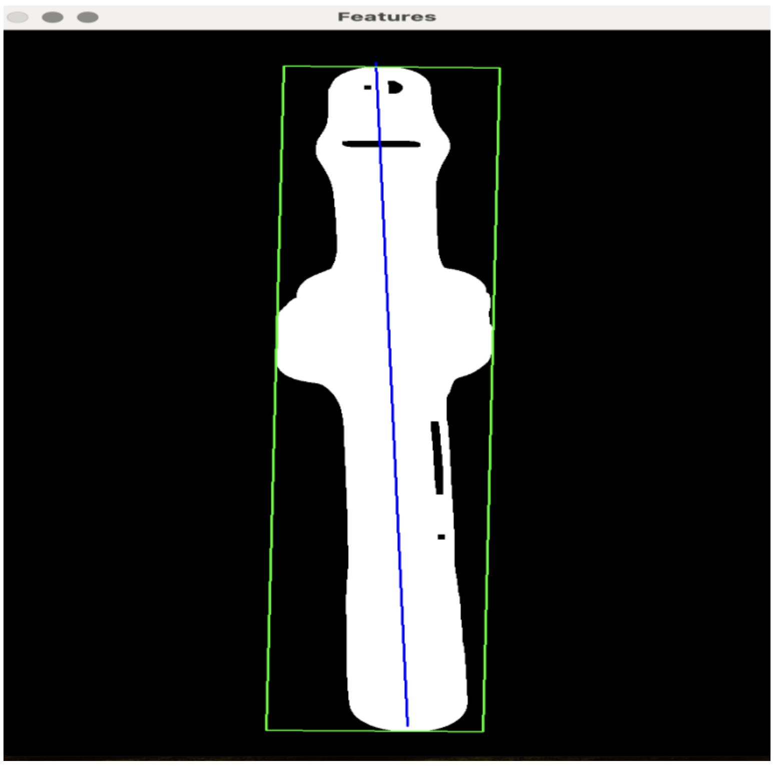
Region features are crucial to understand the properties of the area. In the project, we have used cv::moments function and we calculated the central moment to calculate the axis of least central moment. Furthermore, we have extracted feature bounding box ratio. It was implemented using computeAndDisplayFeatures and in addition to computing the features, we also display features like oriented bounding box and axis of least central moment

Image 1 -



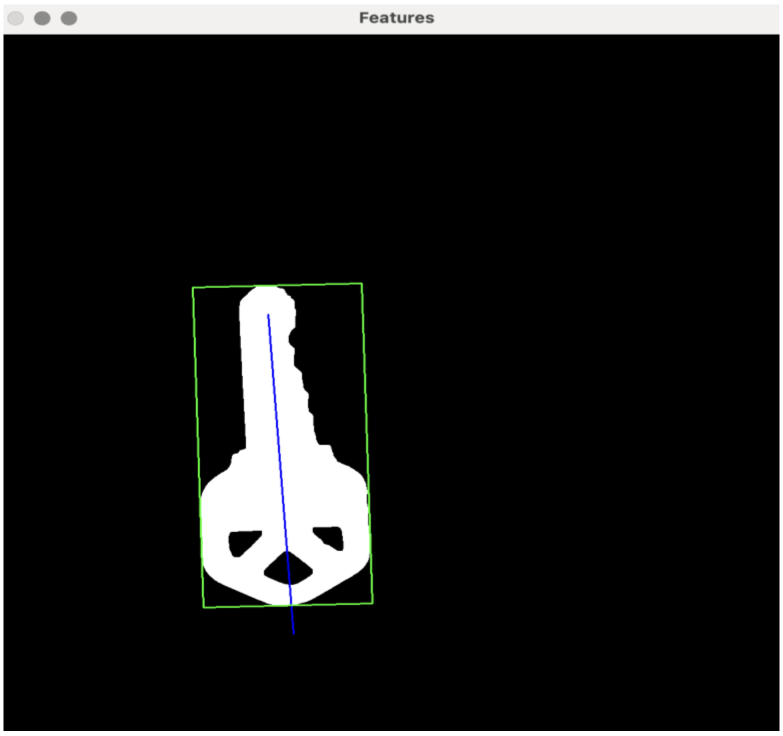
Feature Vector - 0.0564525,1.2473936

Image 2 -



Feature Vector - 1.5463721,5.9236670

Image 3 -



Feature Vector - 1.5026307,0.4560352

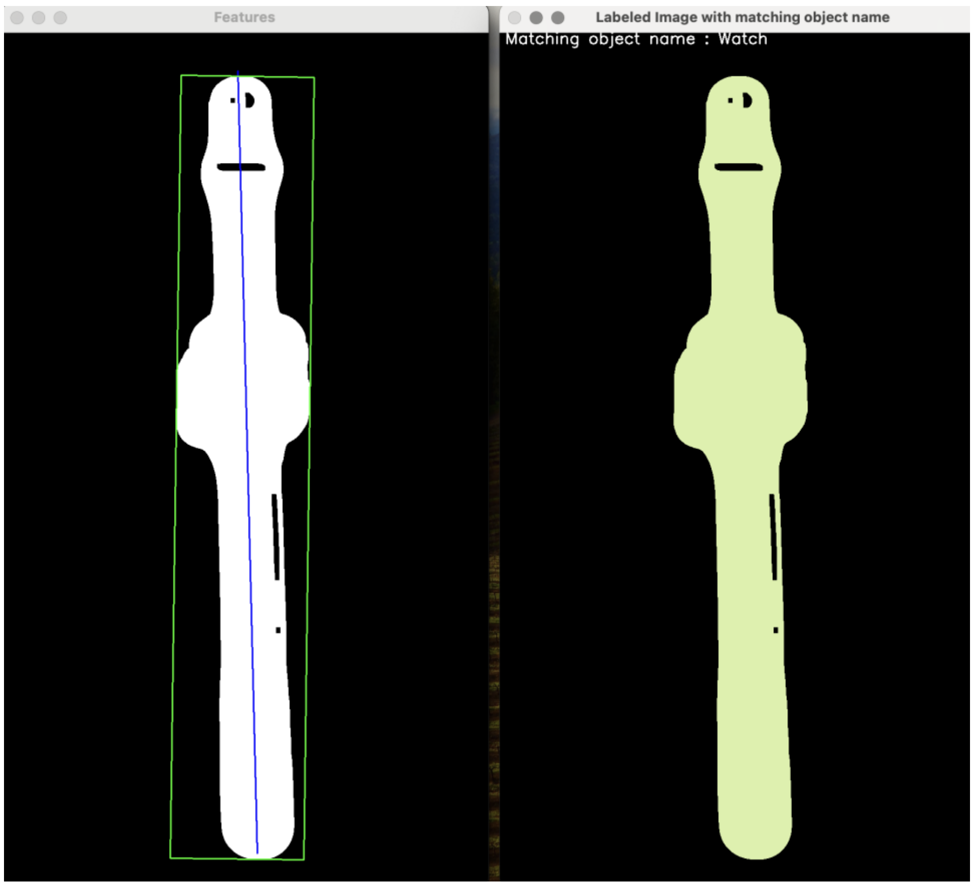
5. Collect training data:

As per the instructions, the features that we have computed from the above functions are being stored in a DB (csv file). When giving the input, the user in addition to specifying the image name, should also provide if the image is for "training" or inference. When the user states it is for training, we will display the segmented regions and ask which region do we want the user to store the features for. If the desired region appears, the user can press "n" or "N" and should provide the object name from the command line. Once we get the object name, we compute features and store them in the csv file. We have written "collectTrainingData" function in that we will pass source image, labels(identified from segmentation) and area_index_pairs vector(which have feature vector information). For each value we will be calling "append_image_data_csv" function from "csv_utils.cpp" file to append the data to "feature_vector.csv" file.

6. Classify new images:

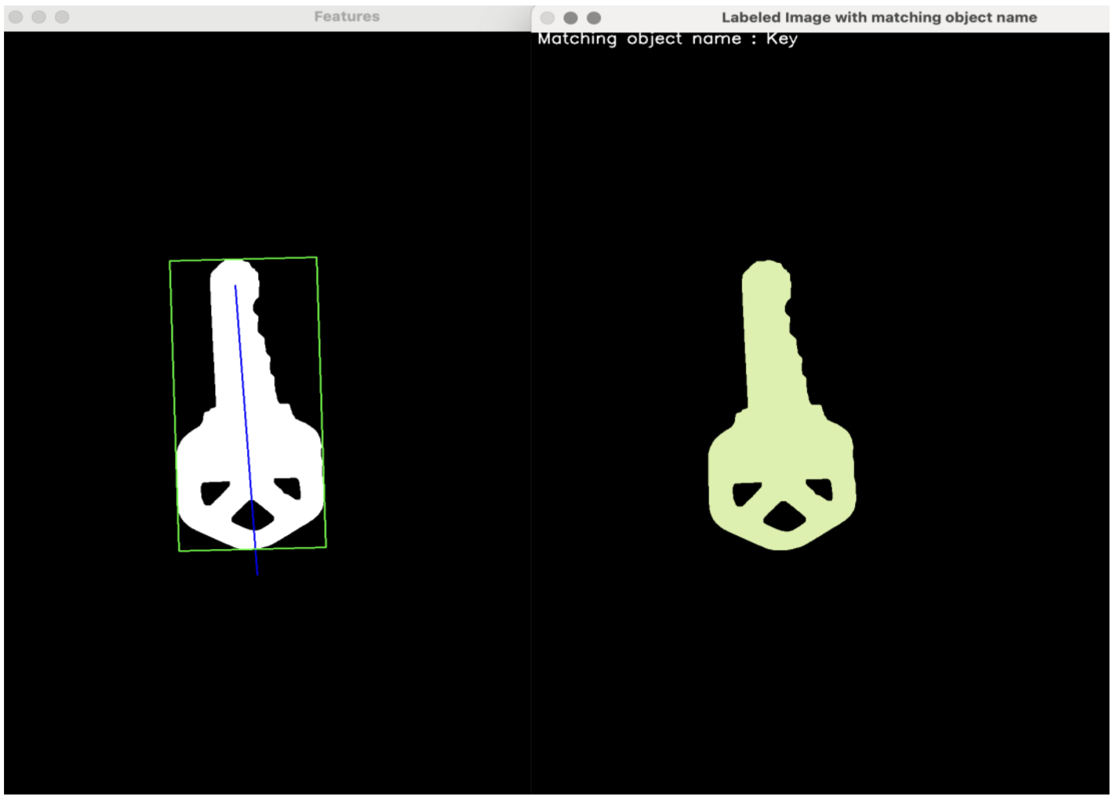
In order to classify images, we have coded scaled Euclidean distance as it works well with heterogeneous objects by doing a squared difference of the features and dividing it by standard deviation of that feature. To classify images, the user can pass any image they want to the program and must specify that is for inference or classification. We would compute the scaled Euclidean distance between the new image with the images in the database, sort it based on the scaled Euclidean distance and return the top matching one. Even if the new image is not in the database, we assign it to the nearest close distance object.

Image 1: Will display the labelled image with the object name.



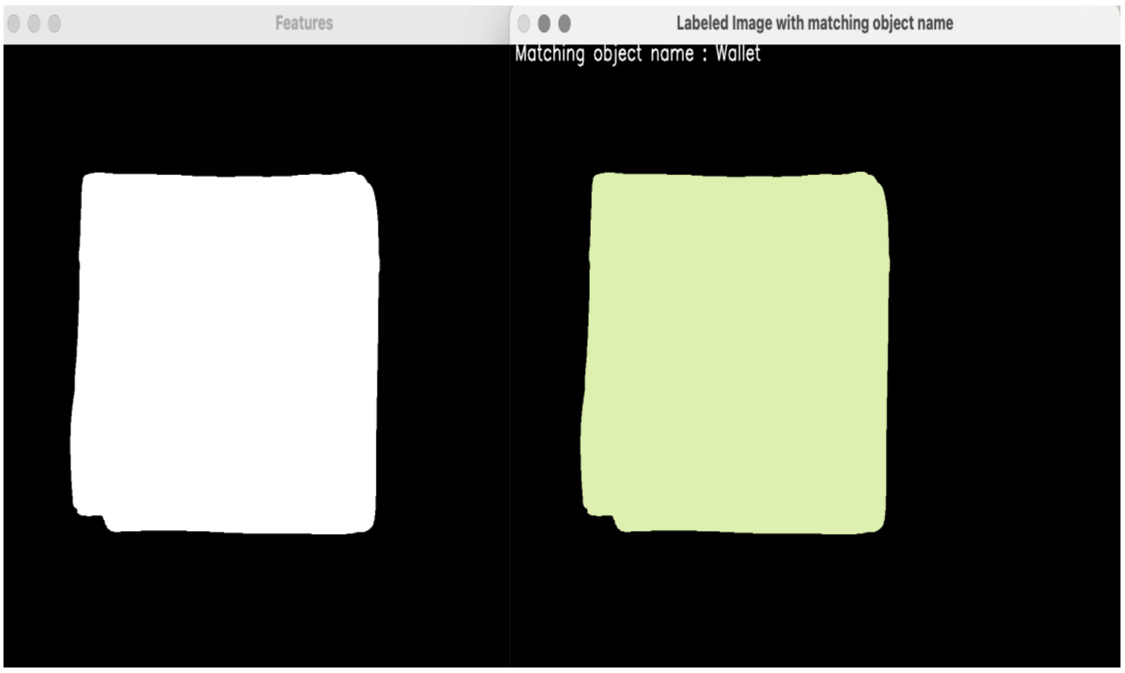
Matching object name : Watch

Image 2:



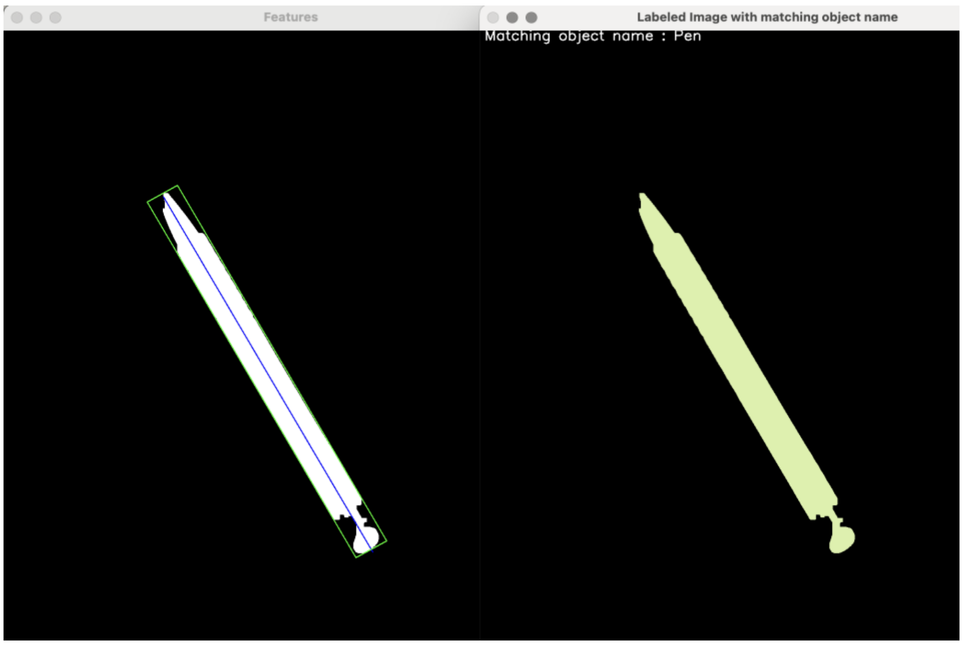
Matching object name : Key

Image 3:



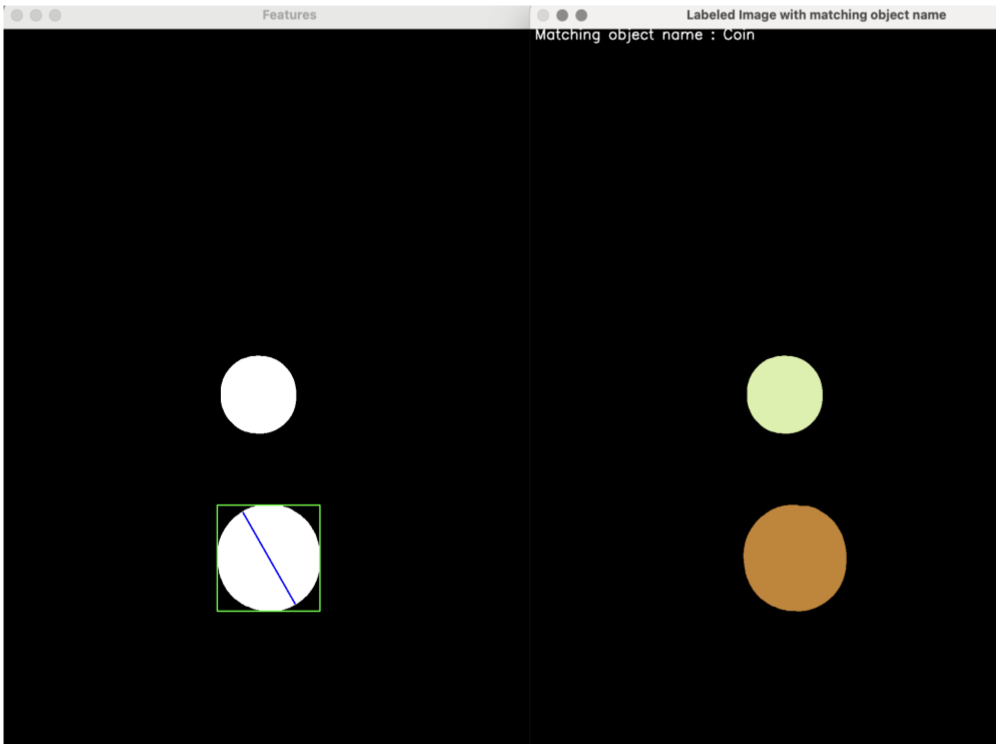
Matching object name : Wallet

Image 4:



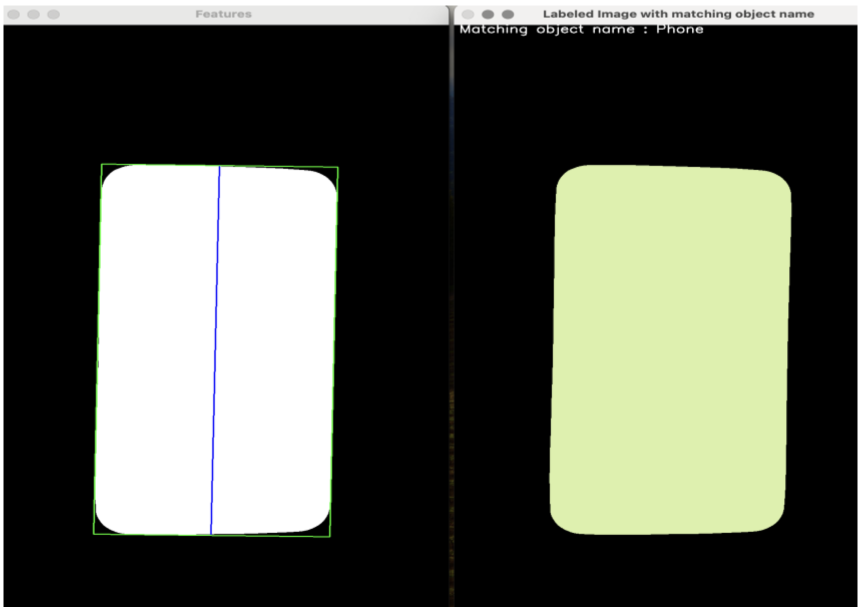
Matching object name : Pen

Image 5:



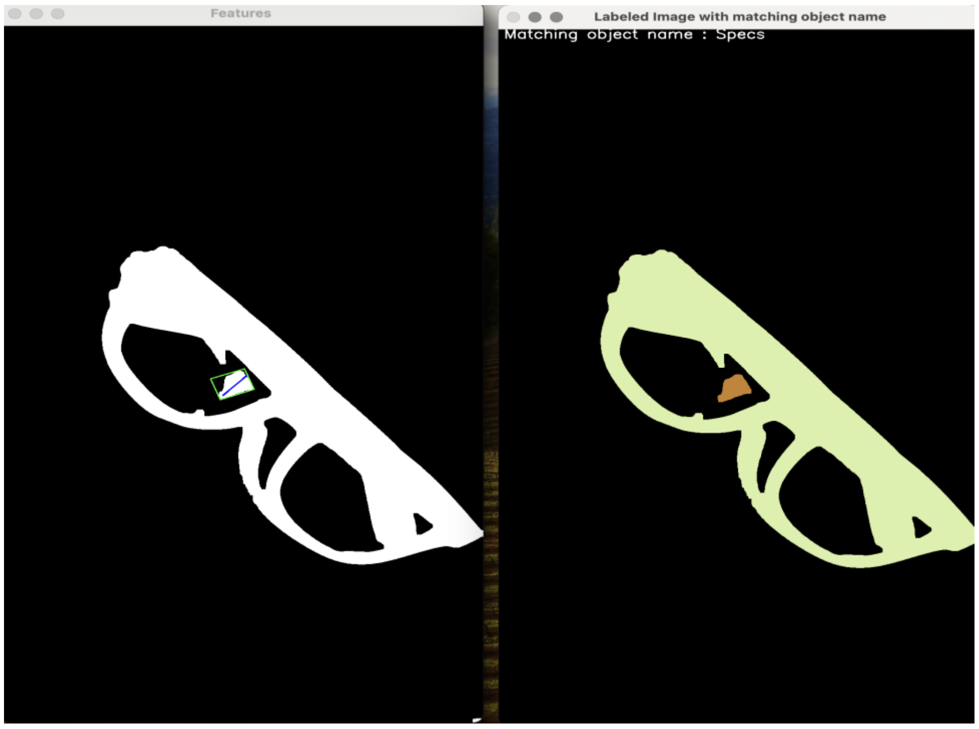
Matching object name : Coin

Image 6:



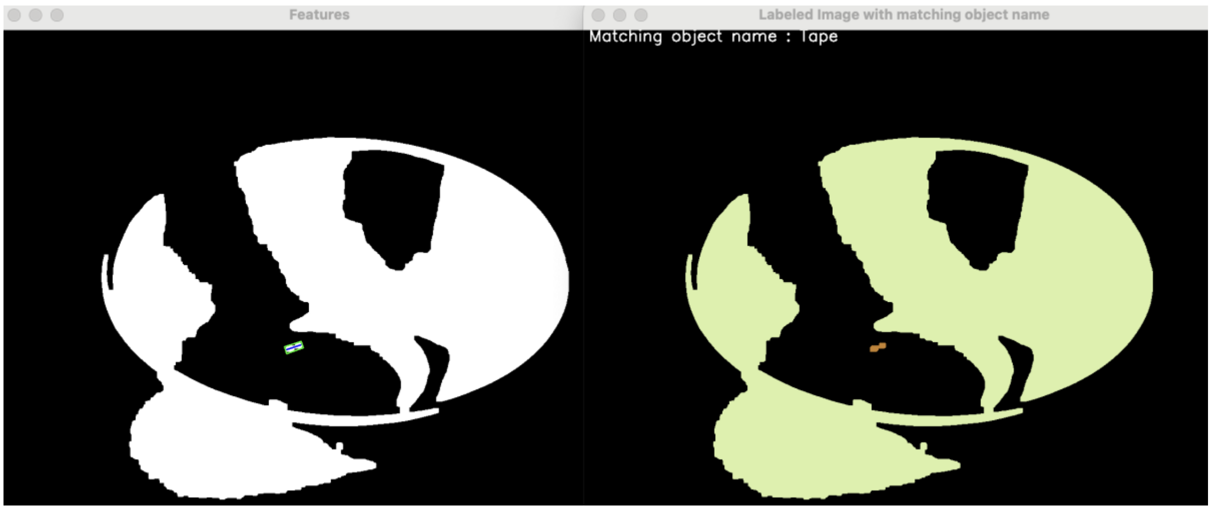
Matching object name : Phone

Image 7:



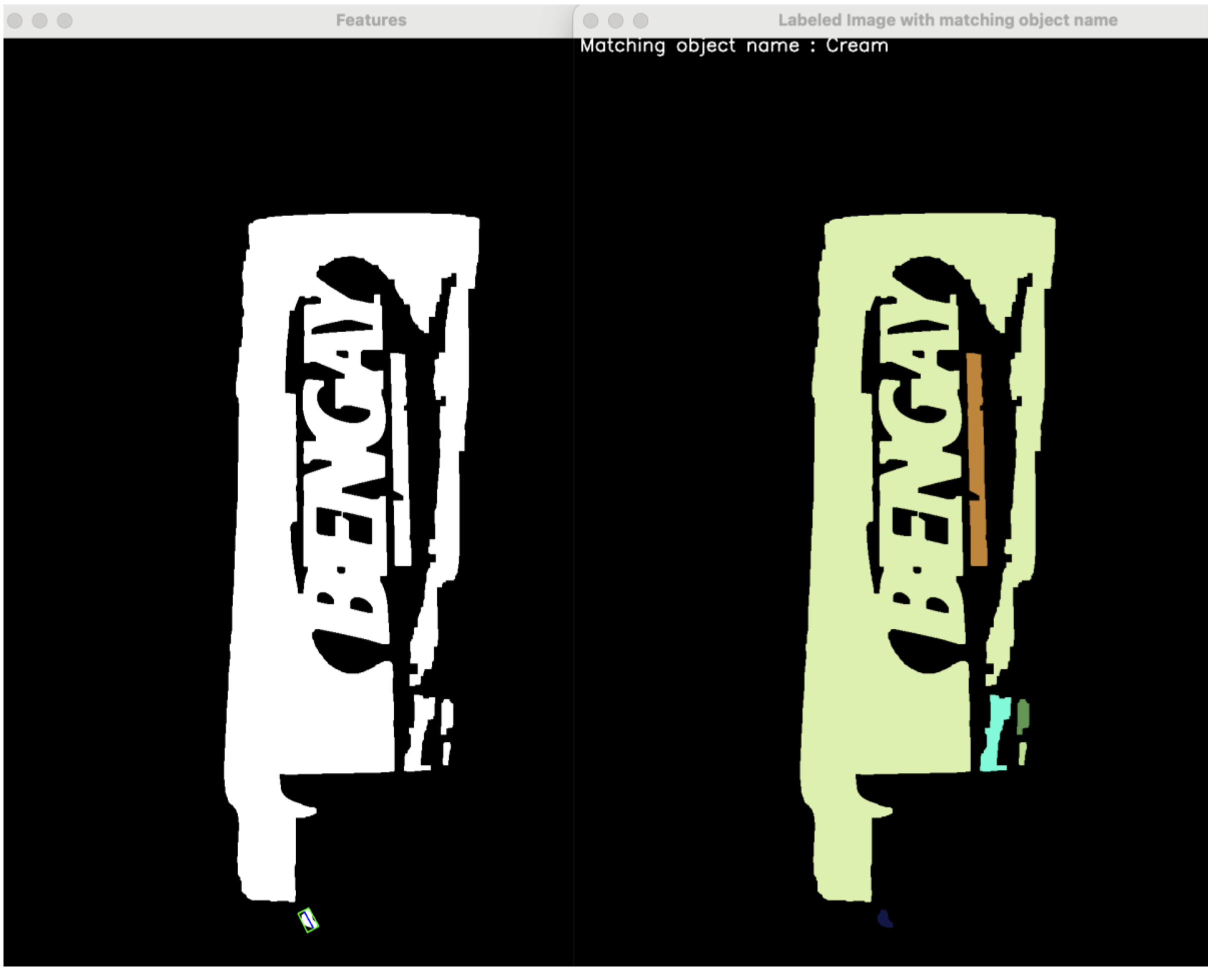
Matching object name : Specs

Image 8:



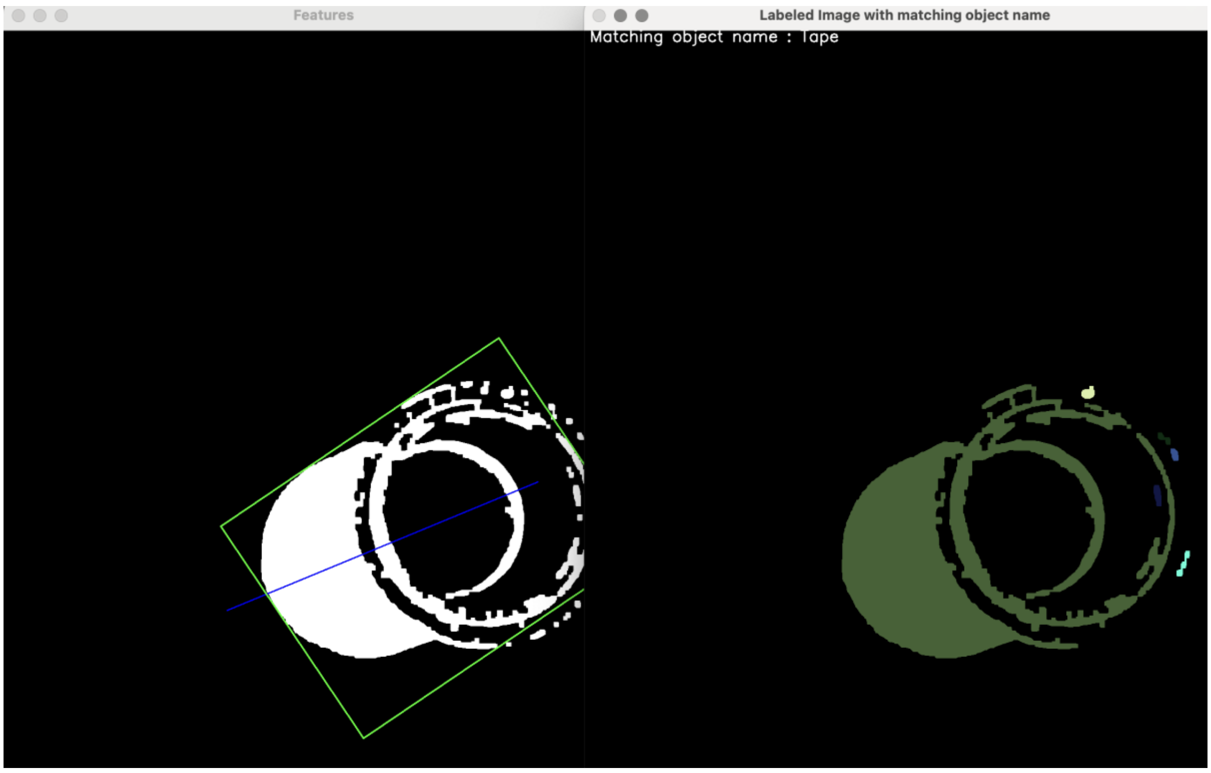
Matching object name : Tape

Image 9:



Matching object name : Cream

Image 10:

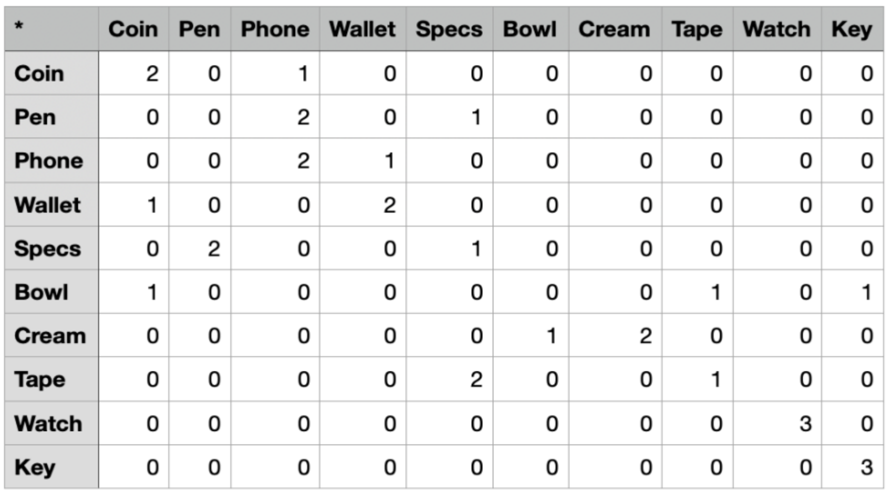

Features
Labeled Image with matching object name
Matching object name : Tape

7. **Evaluate the performance of your system:**

Confusion matrix provides a proper assessment of how a system is working on different images. In the project, we have created a dynamic confusion matrix where anytime you classify an image, it will be updated in the confusion matrix. We are creating a 10*10 confusion matrix, storing in csv file and if inference is done on any image, we would read the existing confusion matrix from the csv file, update it and load it to the csv file.

We can observe our system is working good for few objects like Watch and Key whereas for others it was performing decent and there were few mis-labelled objects like Pen was mis-labelled as Specs maybe because of area and height/width ratio was similar.

confusion_matrix



*	Coin	Pen	Phone	Wallet	Specs	Bowl	Cream	Tape	Watch	Key
Coin	2	0	1	0	0	0	0	0	0	0
Pen	0	0	2	0	1	0	0	0	0	0
Phone	0	0	2	1	0	0	0	0	0	0
Wallet	1	0	0	2	0	0	0	0	0	0
Specs	0	2	0	0	1	0	0	0	0	0
Bowl	1	0	0	0	0	0	0	1	0	1
Cream	0	0	0	0	0	1	2	0	0	0
Tape	0	0	0	0	2	0	0	1	0	0
Watch	0	0	0	0	0	0	0	0	3	0
Key	0	0	0	0	0	0	0	0	0	3

8. **Capture a demo of your system working:**

The system we have build works on still images. Given an image, it would get features like angle of least central moment, percentage filled and bounding box ratio and compares it with the objects in database (csv file) using scaled Euclidean distance and returns the closest match.

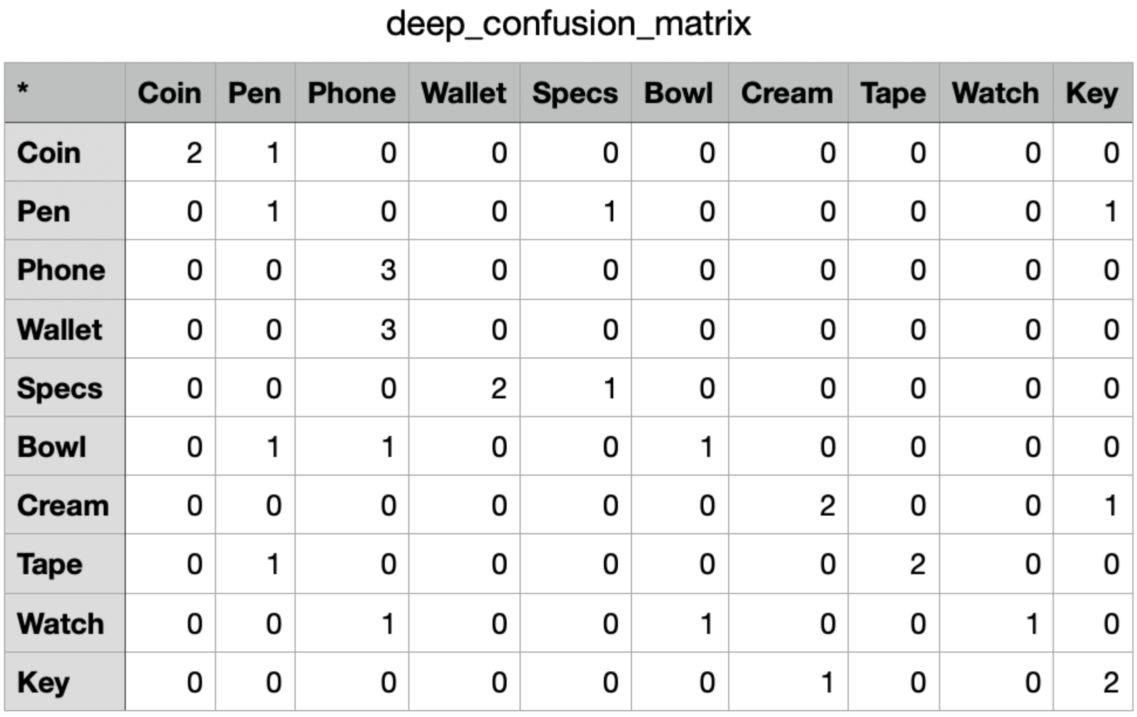
Link to recording:

https://drive.google.com/file/d/1WVrxc5KWM3lE6rrhHmk4CNhXmNKoS2_w/view?usp=sharing

9. **Implement a second classification method:**

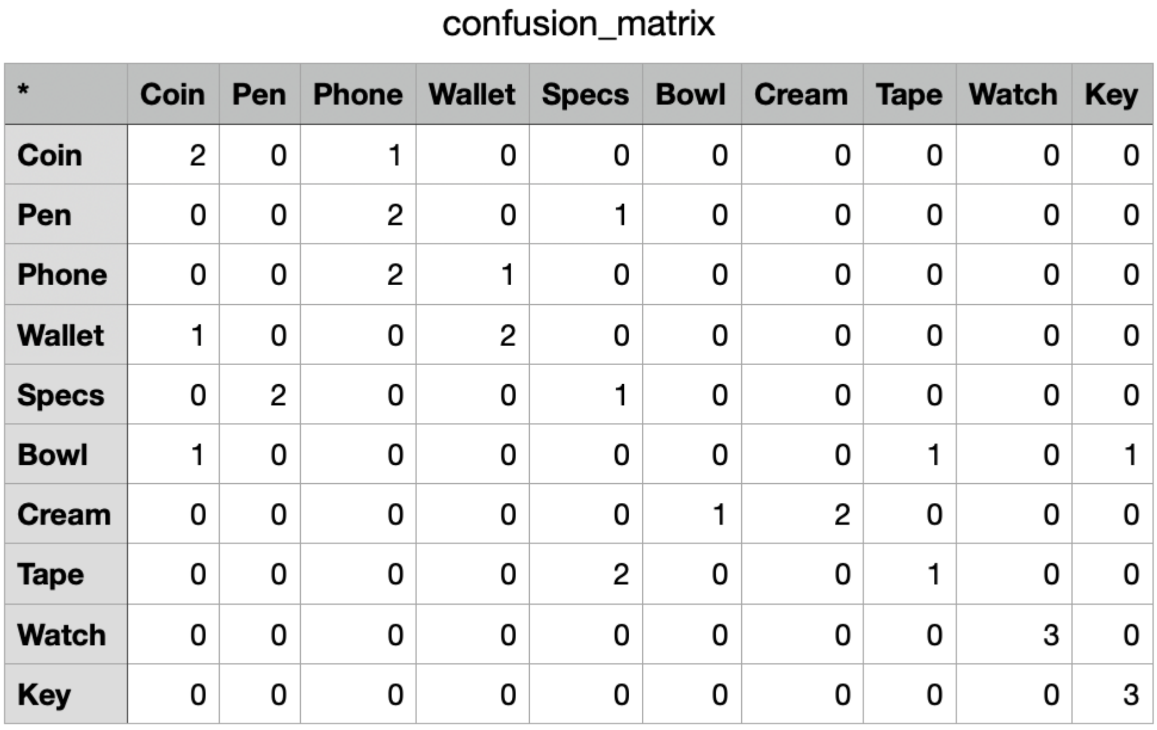
For the second classifier, we have used the pretrained deep learning network to create embeddings. To run this, the user should provide the program with image name, feature type as “deep embeddings”, since we are first creating a training set, “training” and pass the deep learning model path. This would store the embeddings of the training images in csv file. In order to performance inference, the user should specify “inference” in place of training. We would get the embeddings of the image, compare it with the features in training data using cosine distance and return the matching image. Also, to compare the performance and monitor the performance for DNN we have created a new csv file “deep_confusion_matrix” which will be helpful to monitor our system.

Confusion Matrix for deep_embedding:


deep_confusion_matrix

*	Coin	Pen	Phone	Wallet	Specs	Bowl	Cream	Tape	Watch	Key
Coin	2	1	0	0	0	0	0	0	0	0
Pen	0	1	0	0	1	0	0	0	0	1
Phone	0	0	3	0	0	0	0	0	0	0
Wallet	0	0	3	0	0	0	0	0	0	0
Specs	0	0	0	2	1	0	0	0	0	0
Bowl	0	1	1	0	0	1	0	0	0	0
Cream	0	0	0	0	0	0	2	0	0	1
Tape	0	1	0	0	0	0	0	2	0	0
Watch	0	0	1	0	0	1	0	0	1	0
Key	0	0	0	0	0	0	1	0	0	2

Confusion Matrix for Baseline system:


confusion_matrix

*	Coin	Pen	Phone	Wallet	Specs	Bowl	Cream	Tape	Watch	Key
Coin	2	0	1	0	0	0	0	0	0	0
Pen	0	0	2	0	1	0	0	0	0	0
Phone	0	0	2	1	0	0	0	0	0	0
Wallet	1	0	0	2	0	0	0	0	0	0
Specs	0	2	0	0	1	0	0	0	0	0
Bowl	1	0	0	0	0	0	0	1	0	1
Cream	0	0	0	0	0	1	2	0	0	0
Tape	0	0	0	0	2	0	0	1	0	0
Watch	0	0	0	0	0	0	0	0	3	0
Key	0	0	0	0	0	0	0	0	0	3

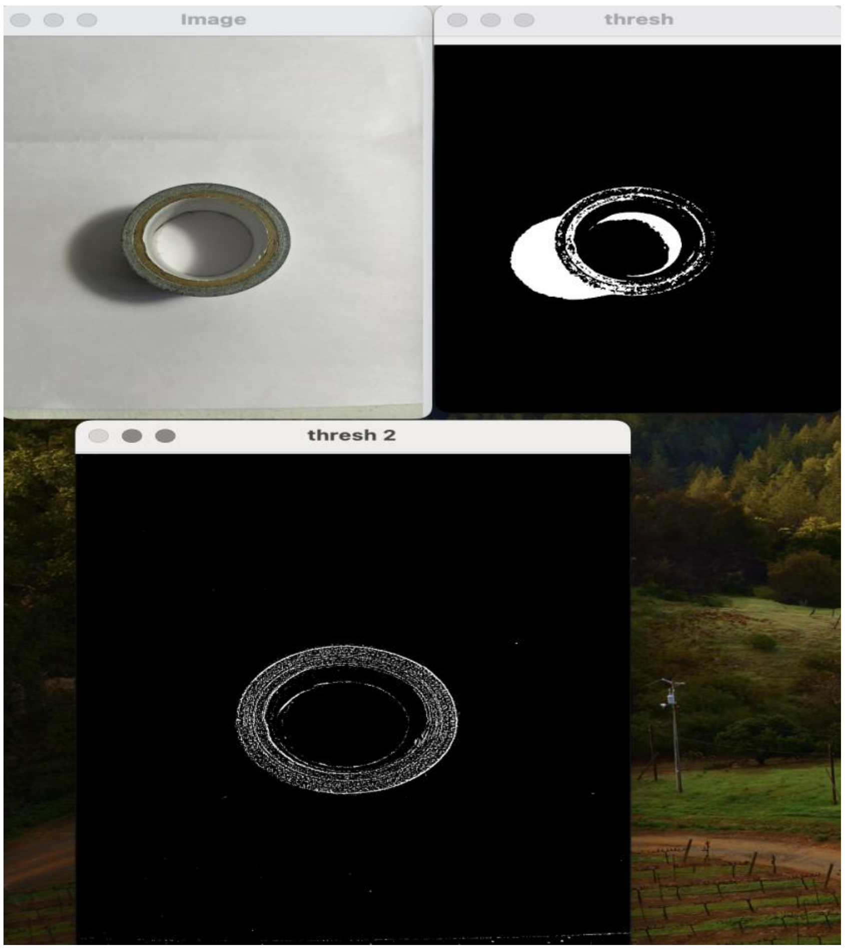
We can observe results were good for few objects in deep_embedding and out of 3 all were correctly recognized like for Phone and Wallet whereas in our baseline system we were able to identify Key and Watch with full success rate. Apart from that results were quite similar for rest of the objects.

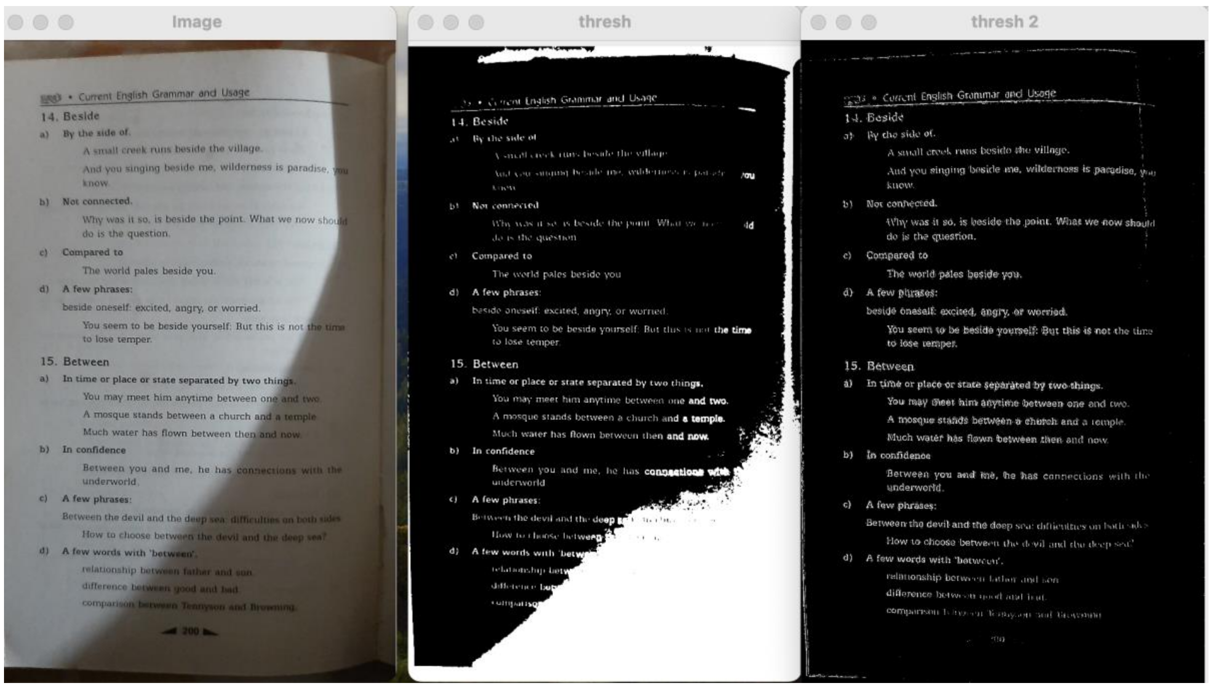
PROJECT EXTENSION

1. Adaptive thresholding

Adaptive thresholding is the same as thresholding but instead of having universal threshold, we take threshold based on the neighboring regions. If an image has different lighting conditions in different areas. In that case, adaptive thresholding can be helpful. The algorithm determines the threshold for a pixel based on a small region around it. So, we get different thresholds for different regions of the same image which gives better results for images with varying illumination. For this project we have implemented Mean Threshold adaptive method where the threshold value is the mean of the neighborhood area minus the constant C.

We can see in the following images when an object have shadow in binary thresholding it is considering shadow as part of region whereas in adaptive thresholding(thresh2) it is only looking for the object.





200 • Current English Grammar and Usage

14. Beside

a) By the side of.

A small creek runs beside the village.

And you singing beside me, wilderness is paradise, you know.

b) Not connected.

Why was it so, is beside the point. What we now should do is the question.

c) Compared to

The world pales beside you.

d) A few phrases:

beside oneself: excited, angry, or worried.

You seem to be beside yourself: But this is not the time to lose temper.

15. Between

a) In time or place or state separated by two things.

You may meet him anytime between one and two.

A mosque stands between a church and a temple

Much water has flown between then and now.

b) In confidence

Between you and me, he has connections with the underworld.

c) A few phrases:

Between the devil and the deep sea: difficulties on both sides

How to choose between the devil and the deep sea?

d) A few words with 'between'.

relationship between father and son

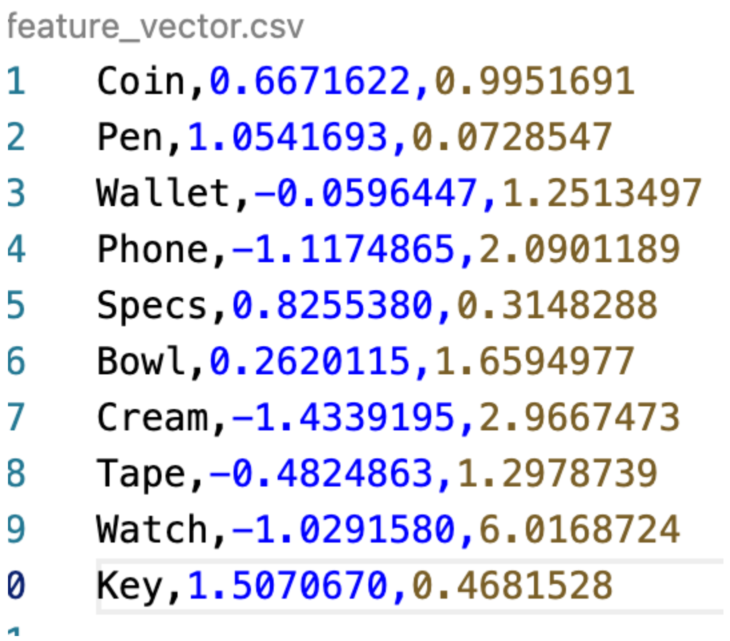
difference between good and bad

comparison between Tennyson and Browning.

200

2. Training 10 Objects

As an extension, we have trained the software to recognize 10 systems. The objects we trained are Coin, Pen, Phone, Wallet, Specs, Bowl, Cream, Tape, Watch, Key and evaluated the system on all 10 images. We have done this for both our baseline system and also for deep_embeding and have included confusion matrix for both of them.

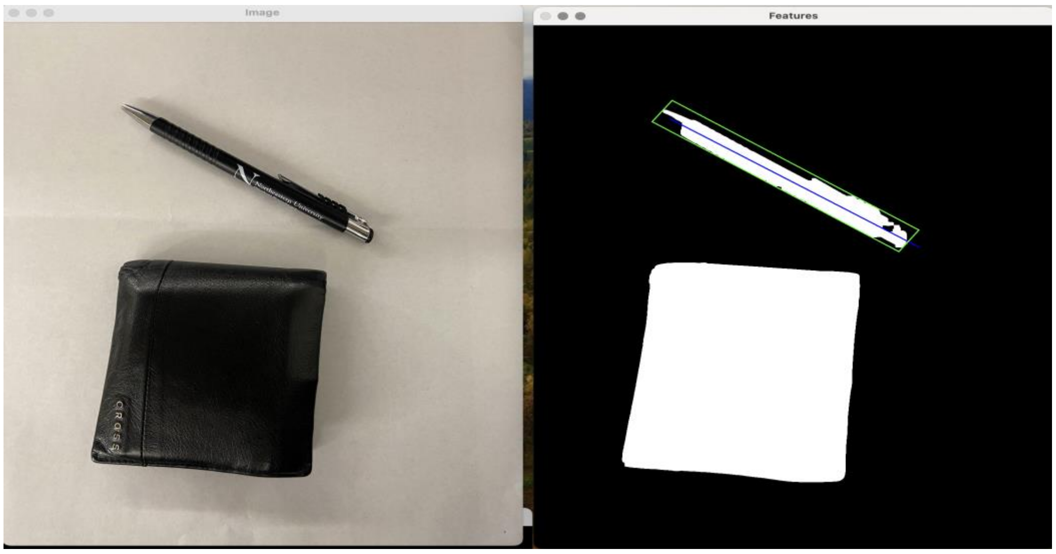

feature_vector.csv

```
Coin,0.6671622,0.9951691
Pen,1.0541693,0.0728547
Wallet,-0.0596447,1.2513497
Phone,-1.1174865,2.0901189
Specs,0.8255380,0.3148288
Bowl,0.2620115,1.6594977
Cream,-1.4339195,2.9667473
Tape,-0.4824863,1.2978739
Watch,-1.0291580,6.0168724
Key,1.5070670,0.4681528
```

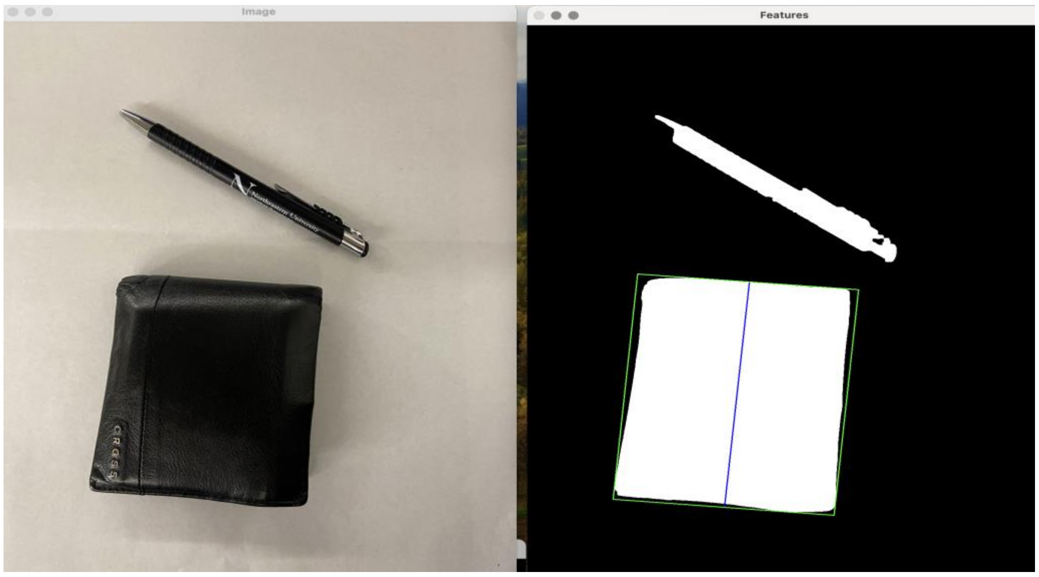
3. Allowing the user to train multiple objects from just a single image:

We have designed the system such that a user can input an image with multiple objects and can train or multiple objects from just a single image. A user would input an image the program would show each region to the user. If user wants to train the image, upon pressing q it will go to next region without saving and upon pressing n, the system would ask the user to enter the object name and store it in database and it would go to the next region and the user can select what they want.

Demo Recording:
https://drive.google.com/file/d/1FreDYOVFgGn5ZZO3jr9WiJBe-2MOmvN5/view?usp=sharing

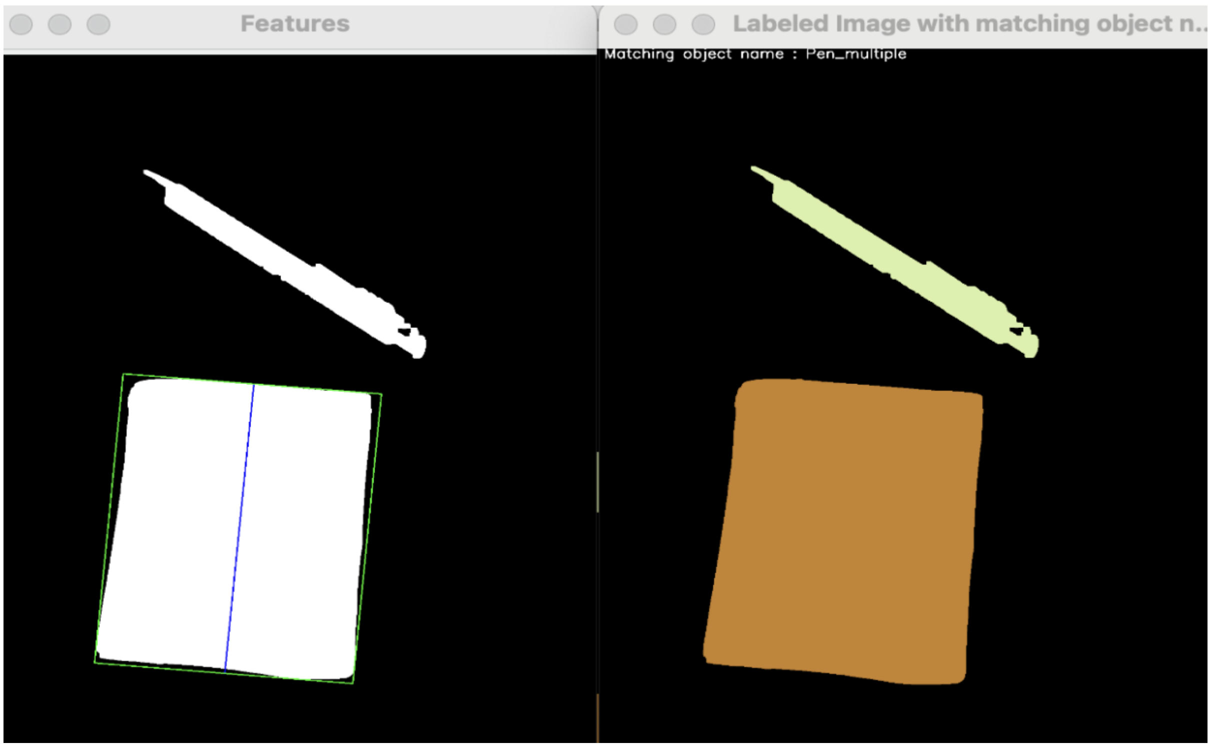


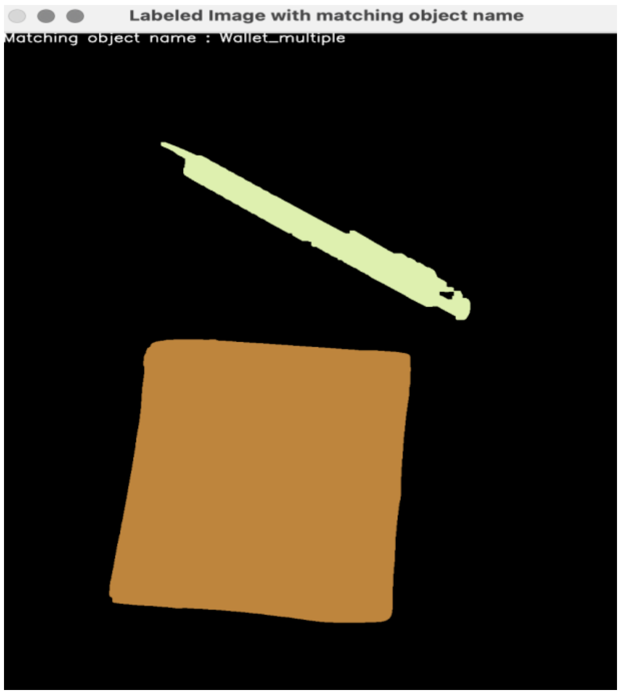
After pressing "q" it will go to other identified region.



4. Allow the user to perform inference on multiple objects from a single image

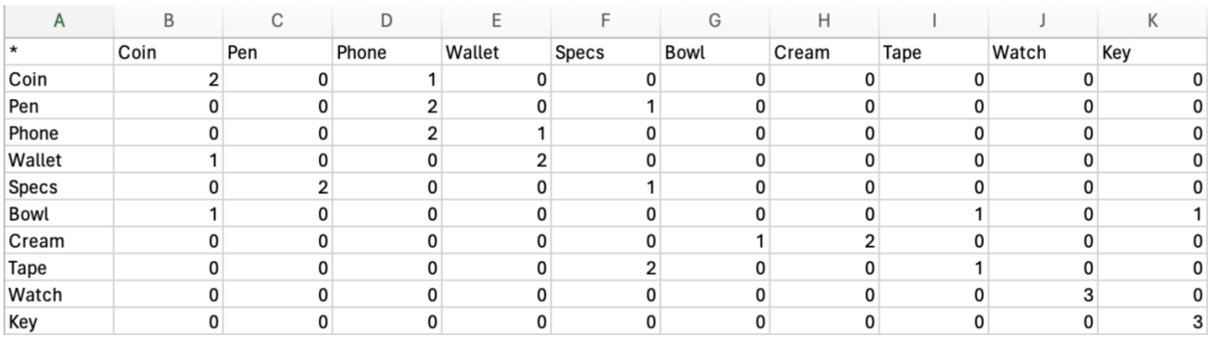
The system works in such a way that it would segment multiple regions in an image. If a user gives an image with multiple, the system would show each region one by one and if the user wants to perform inference, he/she can press s and it would recognize that object and would go to next region and it keeps on happening until there are no objects to recognize.
Demo Recording:
https://drive.google.com/file/d/1MpsWXtd-6o1mV5qTV6RFDXqVEtMUQTLA/view?usp=sharing


Features
Labeled Image with matching object n..
Matching object name : Pen_multiple


Labeled Image with matching object name
Matching object name : Wallet_multiple

5. Dynamic Confusion Matrix

We have implemented the confusion matrix in such a way that anytime we perform inference on an image, it will get update the existing confusion matrix. Therefore, we can keep on tracking the performance of the system for multiple instances of an object using function"createConfusionMatrix" function.



A	B	C	D	E	F	G	H	I	J	K
*	Coin	Pen	Phone	Wallet	Specs	Bowl	Cream	Tape	Watch	Key
Coin	2	0	1	0	0	0	0	0	0	0
Pen	0	0	2	0	1	0	0	0	0	0
Phone	0	0	2	1	0	0	0	0	0	0
Wallet	1	0	0	2	0	0	0	0	0	0
Specs	0	2	0	0	1	0	0	0	0	0
Bowl	1	0	0	0	0	0	0	1	0	1
Cream	0	0	0	0	0	1	2	0	0	0
Tape	0	0	0	0	2	0	0	1	0	0
Watch	0	0	0	0	0	0	0	0	3	0
Key	0	0	0	0	0	0	0	0	0	3

LEARNING

The project extended our learning from project 1 and project 2. We have become more adept at manipulating images from pixels level. In addition to that, we have learned how the system 'looks' at different regions of the image and how we can perform 2D object recognition on multiple objects from both the classical and deep learning network perspective.

Also since we implemented few algorithm for binary image processing from scratch it deepen the understanding more as to how object recognition process works and each steps involved in that.

ACKNOWLEDGEMENT/SOURCES

1. https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html
2. https://opencv.org/
3. https://pyimagesearch.com/
4. https://towardsdatascience.com/
5. https://medium.com/spinor/a-straightforward-introduction-to-image-thresholding-using-python-f1c085f02d5e