Part 1. Intro

In this project, we are implementing Content Based Image Recognition using various methods of histogram and distance functions.
In the first task, we just cropped the image to only 9X9 in the center and scan through our photo database to match the same pixel.
The rest of the task implemented RG Chromaticity, RGB 3D, magnitude, and orientation histogram and some combinations of them. We evaluate the performance using sum squared difference to measure the error rate as well as using histogram intersection which is measuring similarity. All of these matching techniques are programmed as a pipeline so we can pick an image, and pick the kind of feature function we want to implement to get top n matching images.

Task 1: Baseline matching

Use the 9x9 square in the middle of the image as a feature vector. Use sum-of-squared-difference as the distance metric. Make sure that comparing the image with itself results in a distance of 0.
Feature vector: 9x9
Distance metric: squared sum difference

Target image: pic.0986.jpg

```
(base) ➜  src ./src ../olympus

Compute feature 1..
Reading ../res/fi1.csv
Finished reading CSV file
finsih read image

1: pic.1016.jpg
error: 0

2: pic.0986.jpg
error: 31299

3: pic.0641.jpg
error: 44463

4: pic.0233.jpg
error: 88997
Reading ../res/fi2.csv
```



Top three matches are pic.0986.jpg, pic.0641.jpg, and pic.0233.jpg.

Task 2: Histogram Matching

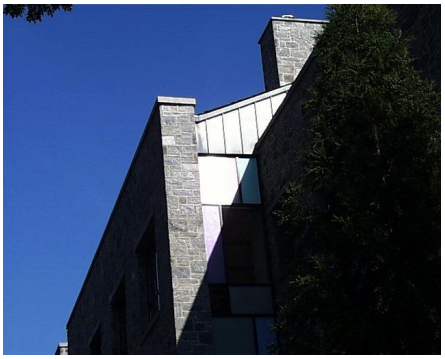
Use a single normalized color histogram of choice (at least two-dimensional) as the feature vector. Use histogram intersection as the distance metric. Calculate the histograms from the image. We will use the cv::Mat to hold the 2-D or 3-D histogram data.

Feature vector: 3D RGB histogram
Bin size : 8
Distance metric: histogram intersection (1 - symmetry)

```
Reading ../res/fi2.csv
Finished reading CSV file
finsih read image

1: pic.0164.jpg
error: 0.00113589

2: pic.0110.jpg
error: 0.575465

3: pic.1032.jpg
error: 0.57632

4: pic.0092.jpg
error: 0.61054

Compute feature 5..
```

Target image: pic.0164.jpg

1. *Pic.0110.jpg*



2. *Pic.1032.jpg*

3. *pic.0092.jpg.*

Task 3: Multi-histogram Matching

Use two or more color histograms of your choice as the feature vector. The histograms should represent different spatial parts of the image. The parts can be overlapping or disjoint. For example, you could use a whole-image histogram and a second histogram that looks at only the center of the image
Feature vector: Multiple 3D RGB histogram (top and bottom)
Bin size : 8
Distance metric: histogram intersection

Target image: pic.0923.jpg

Weighting 1
Top weight = 0.7
Bottom weight = 0.3

```
Finished reading CSV file
finsih read image

1: pic.0923.jpg
error: 0.00127924

2: pic.1021.jpg
error: 0.457708

3: pic.1055.jpg
error: 0.476128

4: pic.0409.jpg
error: 0.488049
```

Pic.1021.jpg

pic.1055.jpg

pic.0409.jpg

Weighting 2
Top weight = 0.8
Bottom weight = 0.2
Flipping the weight to the bottom removes the 3rd photo with the green grass at the bottom from the ranking. Instead it picks and image with more gray tone(picture 2) even though there is less blue sky on its top left hand side.

```
finsih read image

1: pic.0923.jpg
error: 0.00122908

2: pic.1021.jpg
error: 0.472443

3: pic.0426.jpg
error: 0.488854

4: pic.1055.jpg
error: 0.493393
```

Top 3 Matches

1. pic.1021.jpg

2. pic.0426.jpg

3. pic.1055.jpg.

Task 4: Texture and Color matching:

Texture and Color:
Use a whole image color histogram and a whole image texture histogram as the feature vector. Choose a texture metric of your choice for this task. Design a distance metric that weights the two types of histograms equally.

Feature vector: 3D RGB histogram and magnitude
Bin size : 8
Distance metric: histogram intersection

Target image: pic.1012.jpg

Weighting 1

```
1: pic.1012.jpg
error: 0.0049956

2: pic.0098.jpg
error: 0.184134

3: pic.0143.jpg
error: 0.188271

4: pic.0490.jpg
error: 0.196676
```

Rgb weight = 0.2
Texture weight = 0.8

Pic.0098.jpg

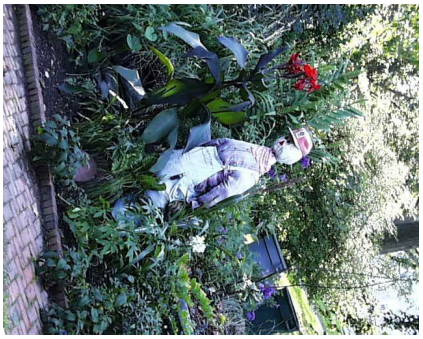
pic.0143.jpg

pic.0490.jpg

Rgb weight = 0.2
Texture weight = 0.8

Weighting 2

Switching the weight to distance give interesting result. We can see the first match is more similar in Color to the target image while also taking into account the texture.

Pic.0490.jpg *pic.0098.jpg* *pic.0430.jpg*

Rgb weight = 0.2

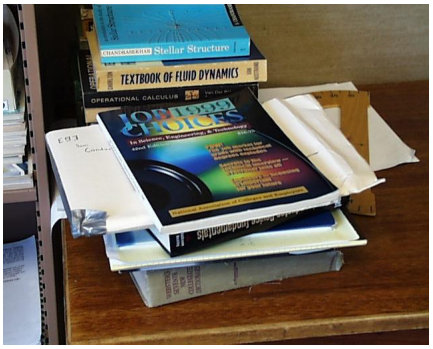
Texture weight = 0.8

Different image

Using the first weight (0.8 texture) we are able to capture pictures with wooden materials

```
1: pic.0438.jpg
error: 0.00324525

2: pic.0320.jpg
error: 0.269361

3: pic.0663.jpg
error: 0.280574

4: pic.0554.jpg
error: 0.296162
```

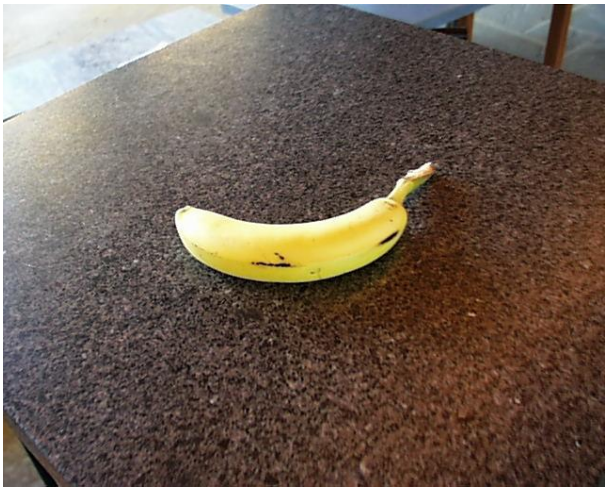
Target image: pic.0438.jpg

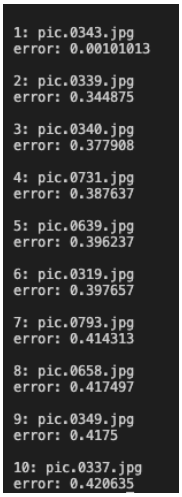



1 pic.1021.jpg *2. pic.0426.jpg* *3. pic.1055.jpg*

Task 5 - Custom Design Take 1

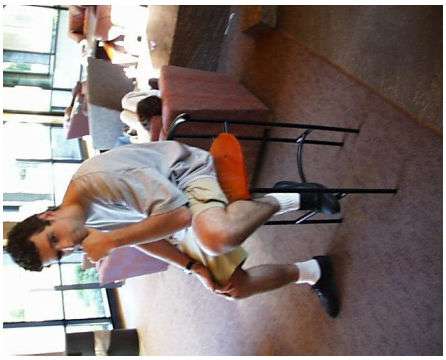
Feature vector: cropped 3D RGB histogram and 2D magnitude-orientation
Bin size : 8
Distance metric: histogram intersection
Cropped to : 150, 100, 400, 300 (x , y , width, height)
RGB_weight : 0.8
Textrure_weight : 0.2



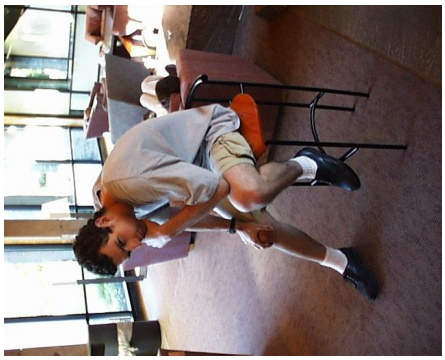


Target image: pic.0343.jpg *cropping pipeline for all images*

It's not picking any banana picture at all. Eventhough I have put cropping as part of the pipeline, its still taking too much of the surrounding effects. So it's trying to match the texture of whats around the banana.



1 pic.0339.jpg



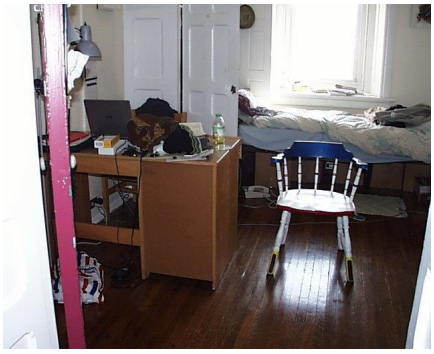
2. pic.0340.jpg

3. pic.0731.jpg

4 pic.0639.jpg

5. pic.0319.jpg

6. Pic.0793.jpg



7 pic.0658.jpg

8. pic.0349.jpg

9. Pic.0337.jpg

Task 5 - Custom design Take 2

Cropped to : 200, 200, 200, 100 (x , y , width, height)
Everything else stays. Now at least we get 1 banana picture.

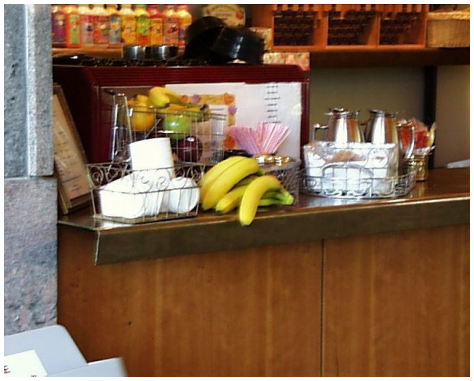
```
Reading ../res/fi5.csv
Finished reading CSV file
finsih read image

1: pic.0343.jpg
error: 0.00580997

2: pic.0514.jpg
error: 0.31699

3: pic.0731.jpg
error: 0.46351

4: pic.0347.jpg
error: 0.46546

5: pic.0220.jpg
error: 0.47841

6: pic.0406.jpg
error: 0.48096

7: pic.0135.jpg
error: 0.49349

8: pic.0053.jpg
error: 0.49622

9: pic.0972.jpg
error: 0.49688

10: pic.0948.jpg
error: 0.4985
```

Target image: pic.0343.jpg

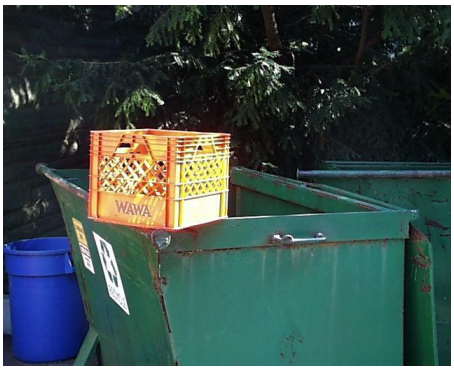
2

3

4



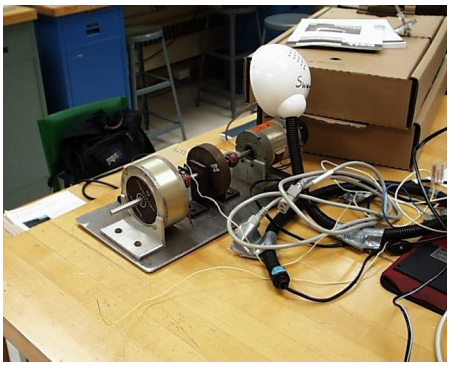
5



6

7

8



9

10

Task 5 - Custom design Take 3

Use RG chromaticity instead of RGB
Rg weight 0.8
Gradient_orientation weight 0.2
Everything else stays

It’s now capturing a more accurate shade of yellow. We got rid of the last 3 unrelated images in the previous take.

```
Finished reading CSV fi
finsih read image

1: pic.0343.jpg
error: 0.00358

2: pic.0462.jpg
error: 0.0731301

3: pic.0514.jpg
error: 0.13579

4: pic.0059.jpg
error: 0.13804

5: pic.0870.jpg
error: 0.14275

6: pic.0347.jpg
error: 0.17528

7: pic.0710.jpg
error: 0.17991

8: pic.0464.jpg
error: 0.18718

9: pic.1046.jpg
error: 0.1895

10: pic.0469.jpg
error: 0.19502
```

Target image: pic.0343.jpg

2

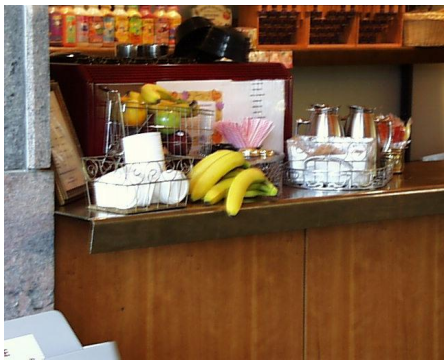
3

4

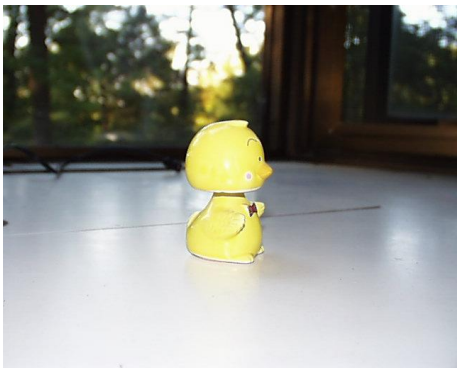
5

6

7



8 9 10



Task 5 - 2nd Image

```
1: pic.0344.jpg
error: 0.00406008

2: pic.0916.jpg
error: 0.18643

3: pic.0028.jpg
error: 0.27137

4: pic.0552.jpg
error: 0.31164

5: pic.0341.jpg
error: 0.37424

6: pic.0353.jpg
error: 0.38547

7: pic.0324.jpg
error: 0.40281

8: pic.0305.jpg
error: 0.40572

9: pic.0279.jpg
error: 0.40573

10: pic.0572.jpg
error: 0.41129
```

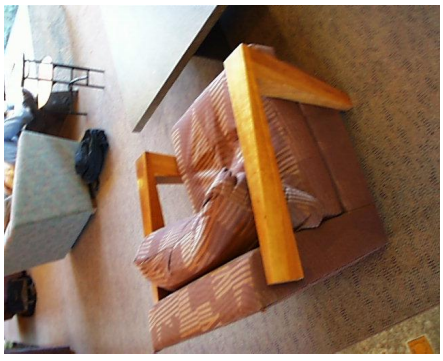
2

3

4

5

6



7

8

9

10