Pucong Han
Visual Interfaces to Computer COMS W4735
Professor John Kender
Assignment 2

System and Tool Configurations

Operating System: Mac OS 10.7
Programming Language: Python 2.7 (JPEG library + Python Imaging Library + OpenCV)

JPEG library

Installation Process

```
$ curl -O http://www.ijg.org/files/jpegsrc.v8c.tar.gz
$ tar zxvf jpegsrc.v8c.tar.gz
$ cd jpeg-8c/
$ ./conFigure
$ make
$ sudo make install
```

OpenCV library and NumPy Array library for Python

Installation Process

```
##Install numpy with Macports
$ sudo port install py27-numpy

##Install OpenCV with Python:
$ sudo port install opencv +python27

##Edit your ~/.bash_profile with:
$ open -t ~/.bash_profile

##Add the line:
export
PYTHONPATH=/opt/local/var/macports/software/opencv/2.2.0_0+pytho
n27/opt/local/Library/Frameworks/Python.framework/Versions/2.7/lib/pyt
hon2.7/site-packages:$PYTHONPATH
```

Matplotlib.pyplot

Installation Process

Downloading latest library from: https://github.com/matplotlib/matplotlib

```
##Install matplotlib
$ sudo python setup.py install
```

Programming Tools: Sublime Text
Imported Libraries:

```
# This module provides a portable way of using operating
system dependent functionality, such as reading and writing
files
import os
```

```
#The glob module finds all the pathnames matching a
specified pattern
import glob
#Python OpenCV libraries
import cv2
import cv
#NumPy Array library
import numpy as np
#Terminal colored text library
from termcolor import colored
#Scipy cluster hierarchy library (dendrogram and linkage)
from scipy.cluster.hierarchy import dendrogram, linkage
#Python matplotlib pyplot library
import matplotlib.pyplot as plt
#The Image module provides provides a number of factory
functions, including functions to load images from files,
and to create new images.
import Image
```

Step 1: Gross Color Matching

Python app: color_matching.py

I compared the downloaded ppm images and jpg images using a three dimensional color histogram. According to the Python OpenCV Tutorial, image histogram is a graphical representation of the intensity distribution of an image. It quantifies the number of pixels for each intensity value considered and contains information about total color distributions. The Python OpenCV library provides a number of useful functions, such as calcHist() and normalize(), which simplified gross color matching process.

My program first read all ppm images from the downloaded folder:

```
for ppm_infile in glob.glob(os.path.join("images", '*.ppm') ):
      #Loads each ppm images from the image folder
      ppm_img = cv2.imread(ppm_infile)
```

My program generates a gray scale image for each loaded image and pass it to the threshold() function. According to the OpenCV documentation, the threshold() is typically used to get a bi-level (binary) image out of a grayscale image for removing a noise. Here is my implementation:

```
#Convert ppm image to gray scale
ppm_imgray = cv2.cvtColor(ppm_img,cv2.COLOR_BGR2GRAY)
#Create ppm binary mask, a 8-bit image, where white denotes that
region should be used for histogram calculations, and black means
it should not
ppm_ret, ppm_thresh_mask = cv2.threshold(ppm_imgray,127,255,0)
```

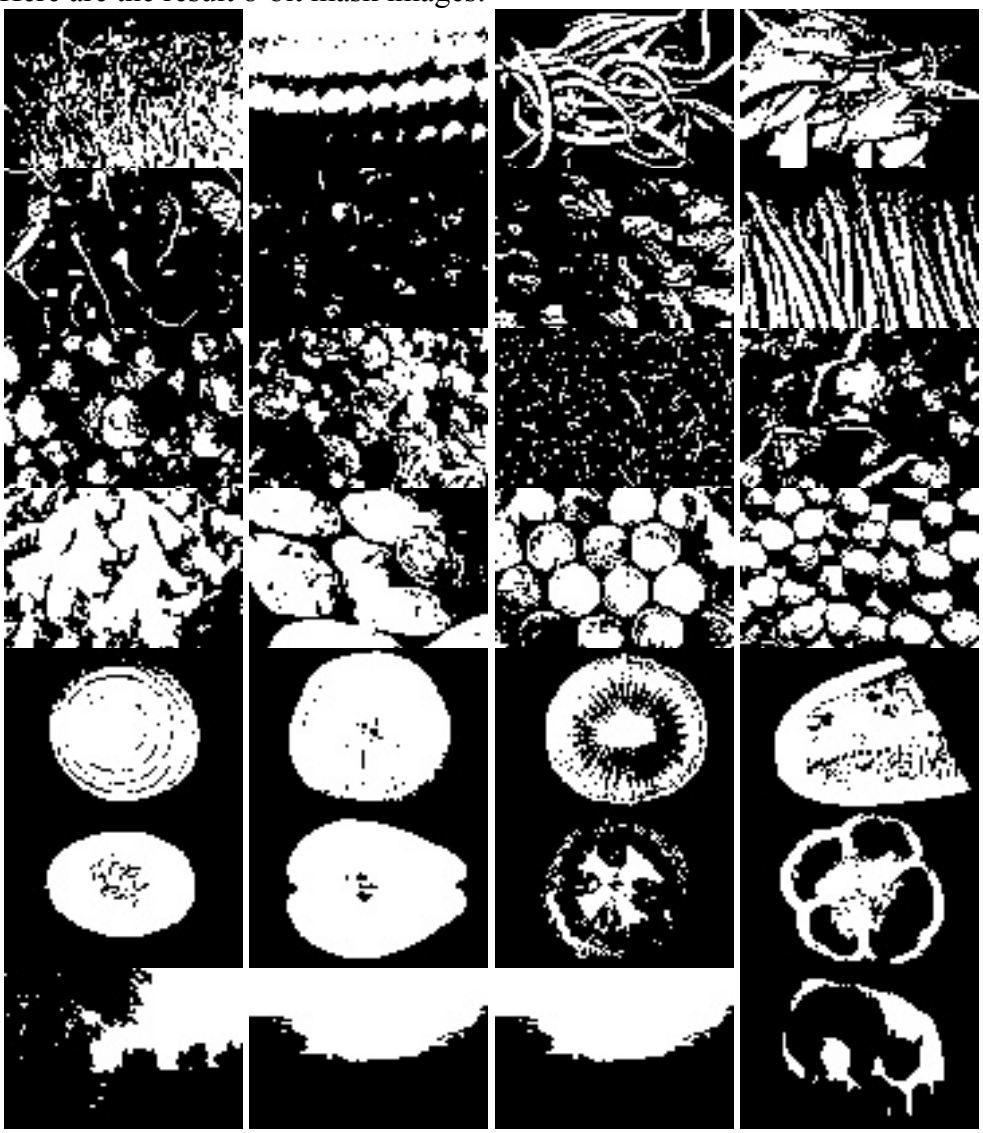
In my program, the threshold() function returns 8-bit mask images with white blobs of fruits, as shown in Figure 1. Using these returned images, my program can distinguish the

body of images from their black backgrounds. The calcHist() function, which I introduce later in this document, takes an 8-bit mask image input for computing histograms. The calcHist() function calculates three dimensional color histograms only in the white blob regions. The 8-bit mask image allows my program to ignore the black background and black pixels that are not related to the fruit images.

For demonstration purpose, I save these 8-bit mask images to the 8bitmask folder using the following code:

```
cv2.imwrite("8bitmask/" + ppm_infile.replace("images/",
"").replace(".ppm", ".jpg"), ppm_thresh_mask)
```

Here are the result 8-bit mask images:



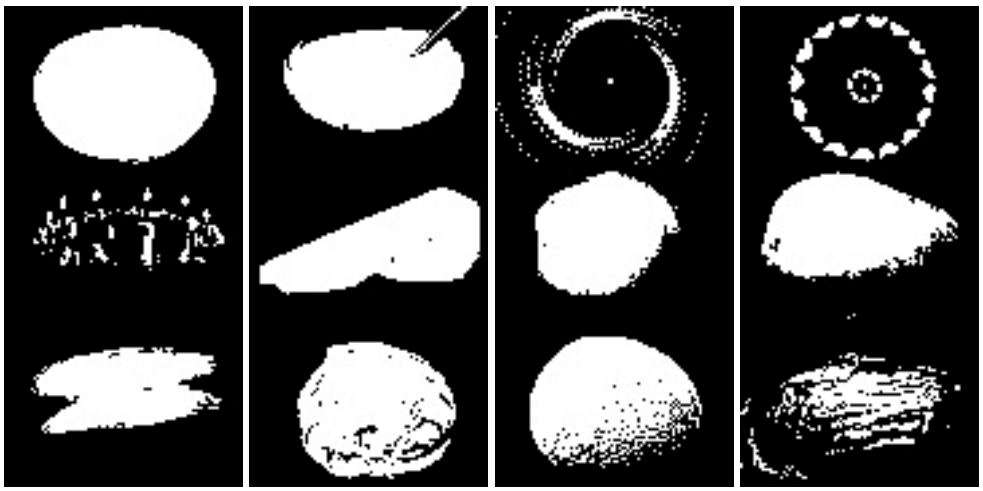


Figure 1: The 8-bit mask images with white blobs of fruits.

As we can see from Figure 1, main body (fruits) of the images are marked using white blobs. Using these mask images, my program can distinguish the main body (white blobs) of the images from the black backgrounds. By computing histograms for each image inside the white blobs, my program ignores the black background and any black pixels that are not related to the main body of an image.

Before calculating histograms, my program set up a bin for the histogram space. The following code represent the RGB axes as full 0 to 255 bins:

```
#Number of bins, since the histogram has 256 colors, it needs 256
bins. Bin is a multidimensional array
bins = np.arange(256).reshape(256,1)
```

My program calculates histograms in three-color spaces, red (255, 0, 0), green (0, 255, 0) and blue (0, 0, 255). The following code set up a color array:

```
color = [ (255,0,0),(0,255,0),(0,0,255) ]
```

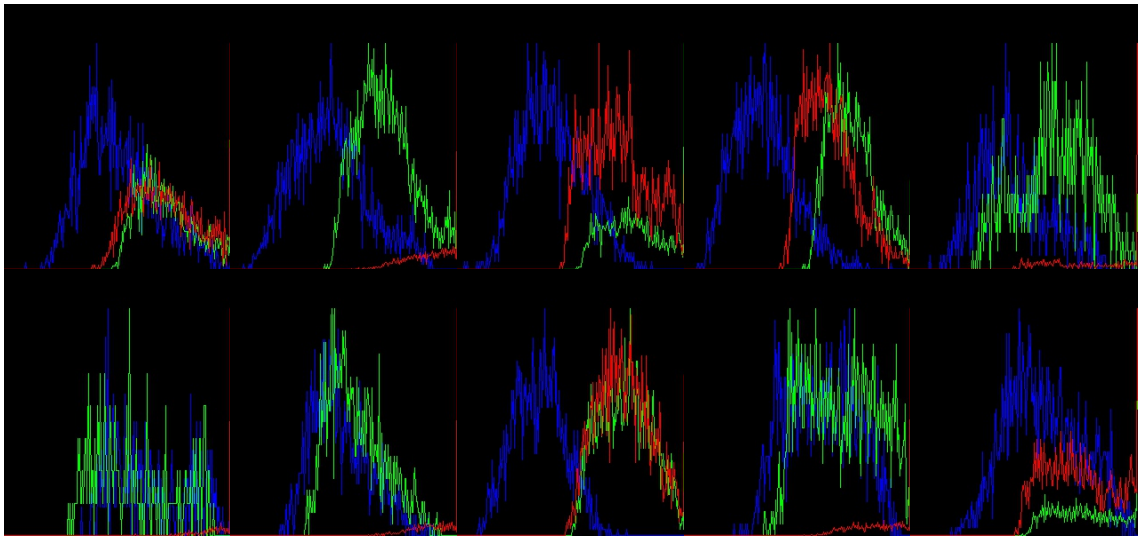
My program iterates an enumerate tuple containing a count and the values obtained from iterating over the list color. Within this iterating loop, my program generates the histogram for each iterated color using the calcHist() function provided by the Python OpenCV. According to the OpenCV documentation, the functions calcHist() calculates the histogram of one or more arrays. The elements of a tuple used to increment a histogram bin are taken from the corresponding input arrays at the same location. In Python, the calcHist() function takes five input parameters including target image, channel, mask, histogram size and range.

My program also normalizes the generated histogram using the normalize() function provided by the Python OpenCV. The reason for this normalization is to make histogram pixels fall below 255 and fit in image height. Here is my implementation:

```
#For drawing the histogram.
```

```
h = np.zeros((300,256,3))

#Iterate an enumerate tuple containing a count and the values
obtained from iterating over the list color.
for ch, col in enumerate(color):
      #Calculates a histogram for the 8-bit region marked in the
      mask using a set of arrays.
      ppm_hist =
      cv2.calcHist([ppm_img],[ch],ppm_thresh_mask,[256],[0,256])
      #Normalize the value to fall below 255 and fit in image
      height.
      cv2.normalize(ppm_hist,ppm_hist,0,255,cv2.NORM_MINMAX)
      #Evenly round to the given number of decimals. For values
      exactly halfway between rounded decimal values, Numpy
      rounds to the nearest even value.
      hist = np.int32(np.around(ppm_hist))
      #Stack bins and hist for drawing the polylines.
      pts = np.column_stack((bins,hist))
      #Draw polylines to represent the histogram
      cv2.polylines(h,[pts],False,col)

#Flip the image vertically.
h=np.flipud(h)
#Write result histogram to the target folder.
cv2.imwrite("histograms/ppm" + ppm_infile.replace("images/",
"").replace(".ppm", ".jpg"), h)
```

After generating each histogram, my program stacks the histogram into a bin and draw lines to represent the histogram. Since the screen x-axis and y-axis are different from the x-axis and y-axis that we are usually see, I flip the image vertically and write to the target folder. Here are my result histograms:



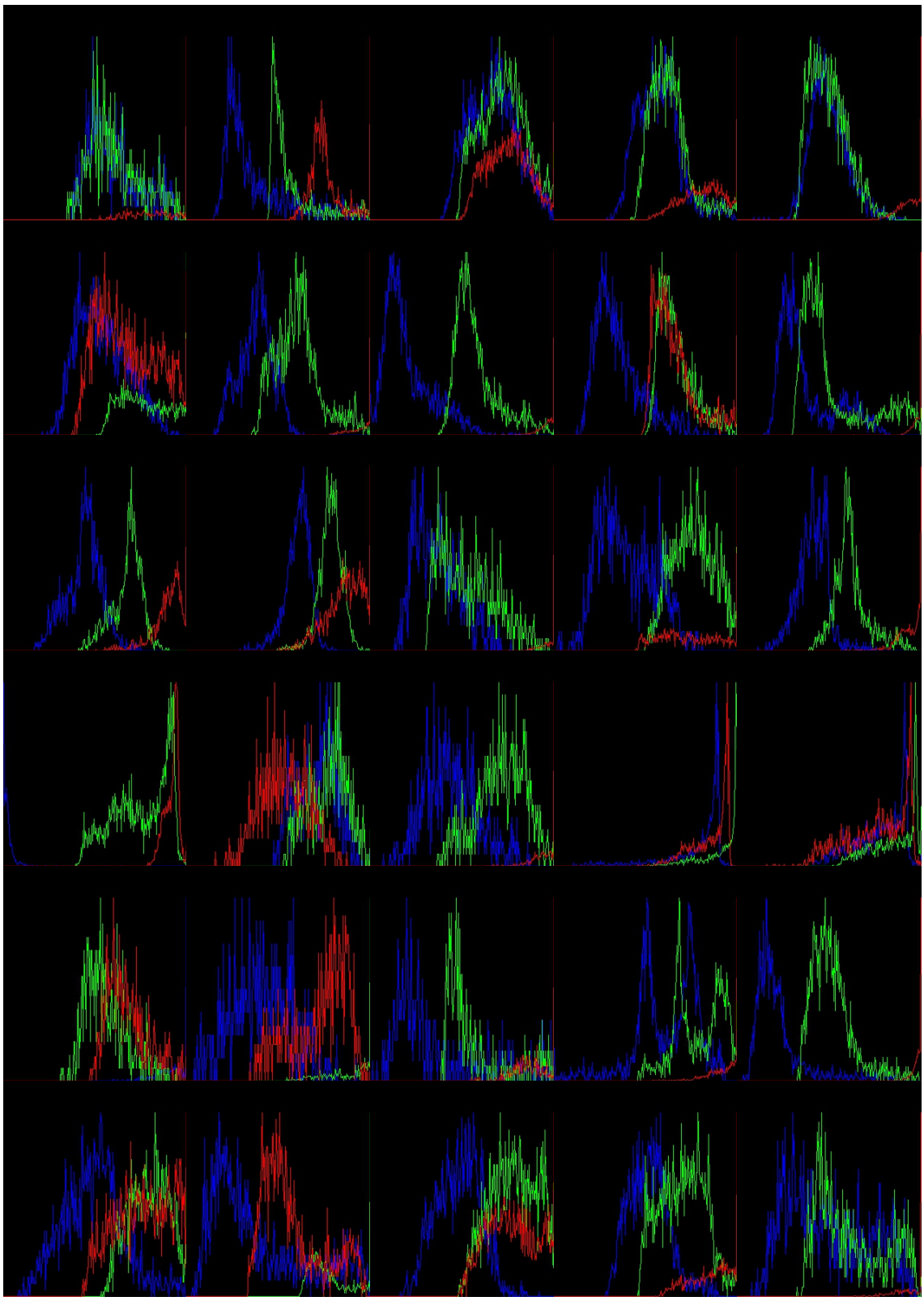


Figure 2: Histograms of fruit images (ppm images).

These 40 histograms contain color information only in the white blobs. In other words, they do not contain information in the black background. The black background has very big impact on the histogram. To show the difference, I print only the first five histograms without passing the mask image (without background ignoring process). Here are the results:

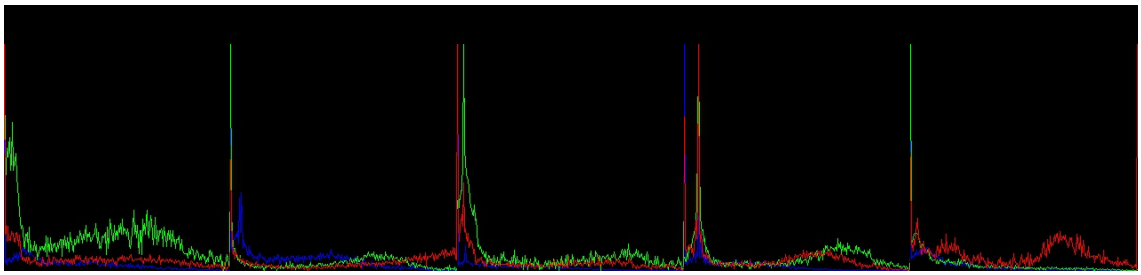


Figure 3: First five-ppm histograms without background ignoring (without mask image).

Comparing the five histograms in Figure 3 with the first five histograms in Figure 2, we can see the black backgrounds have an impact on the intensity color distributions. According to the Photographer Ron Bigelow, histogram displays color distributions. The darkest color is assigned a value of zero. The lightest color is assigned a value of 255. "The horizontal axis displays these tones/colors from the darkest on the left to the lightest on the right," said Bigelow. As we can see from figure 3, with the black background color, the histogram color distributions are pushed to the left of the scale. Images with dissimilar objects that have a lot of black background pixels will have similarities in histogram color distribution. Such impacts will make later histogram correlation comparison become less accurate.

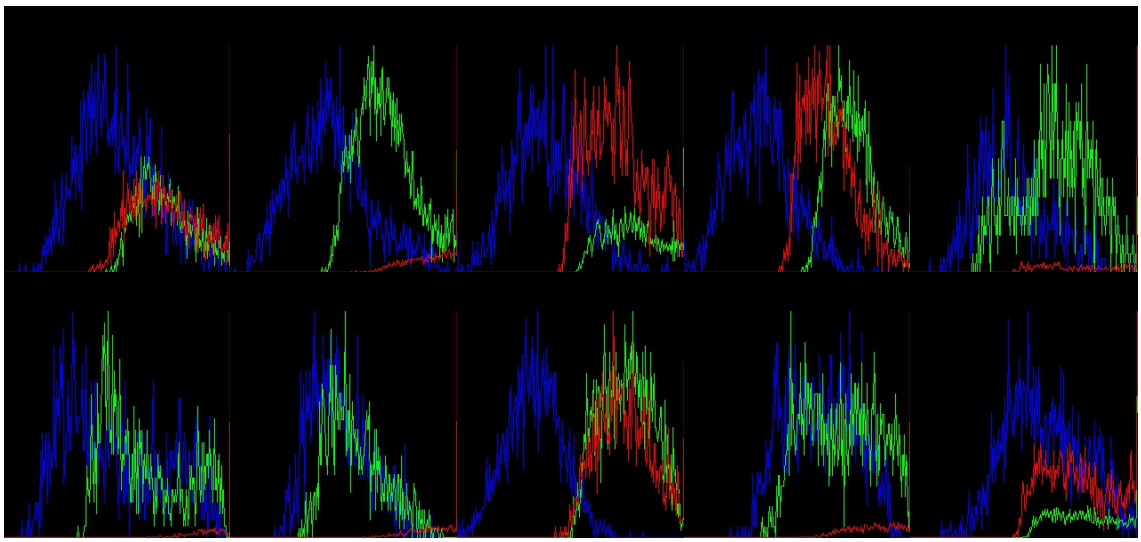
For each ppm image histogram, my program will compare it with all 40 jpg images histogram and find the most similar and dissimilar images using a for loop.

To generate histograms for the 40-jpg images, my program has similar codes (compared with the previous histogram generations):

```
for jpg_infile in glob.glob( os.path.join("images", '*.jpg') ):
      #Loads each jpg images from the image folder.
      jpg_img = cv2.imread(jpg_infile)
      #For drawing the histogram.
      jpg_h = np.zeros((300,256,3))
      #Convert jpg image to gray scale.
      jpg_imgray = cv2.cvtColor(jpg_img,cv2.COLOR_BGR2GRAY)
      #Create jpg binary mask, a 8-bit image, where white denotes
      that region should be used for histogram calculations, and
      black means it should not.
      jpg_ret, jpg_thresh_mask =
      cv2.threshold(jpg_imgray,127,255,0)
      # Save 8 bit mask images
      jpg_bins = np.arange(256).reshape(256,1)
      color = [ (255,0,0),(0,255,0),(0,0,255) ]
```

```
#Iterate an enumerate tuple containing a count and the
values obtained from iterating over the list color.
for ch, col in enumerate(color):
      #Calculates a histogram for the 8-bit region marked in
      the mask using a set of arrays.
      jpg_hist =
      cv2.calcHist([jpg_img],[ch],jpg_thresh_mask,[256],[0,2
      56])
      #Normalize the value to fall below 255 and fit in
      image height.
      cv2.normalize(jpg_hist,jpg_hist,0,255,cv2.NORM_MINMAX)
      #Evenly round to the given number of decimals. For
      values exactly halfway between rounded decimal values,
      Numpy rounds to the nearest even value.
      round_jpg_hist = np.int32(np.around(jpg_hist))
      #Stack bins and hist for drawing the polylines.
      jpg_pts = np.column_stack((jpg_bins,round_jpg_hist))
      #Draw polylines to represent the histogram.
      cv2.polylines(jpg_h,[jpg_pts],False,col)
#Flip the image vertically.
jpg_h=np.flipud(jpg_h)
#Write result histogram to the target folder.
cv2.imwrite("histograms/jpg/" +
jpg_infile.replace("images/", "").replace(".ppm", ".jpg"),
jpg_h)
```

The results of these 40 jpg histograms, as shown in Figure 4, are similar to the ppm histograms in Figure 3. Although many of these images are the same, there are some minor variations in pick values. For instance, if you look closely on the sixth histogram in ppm and the sixth histogram in jpg, you can see that the green and blue distribution and pick values are slightly different.



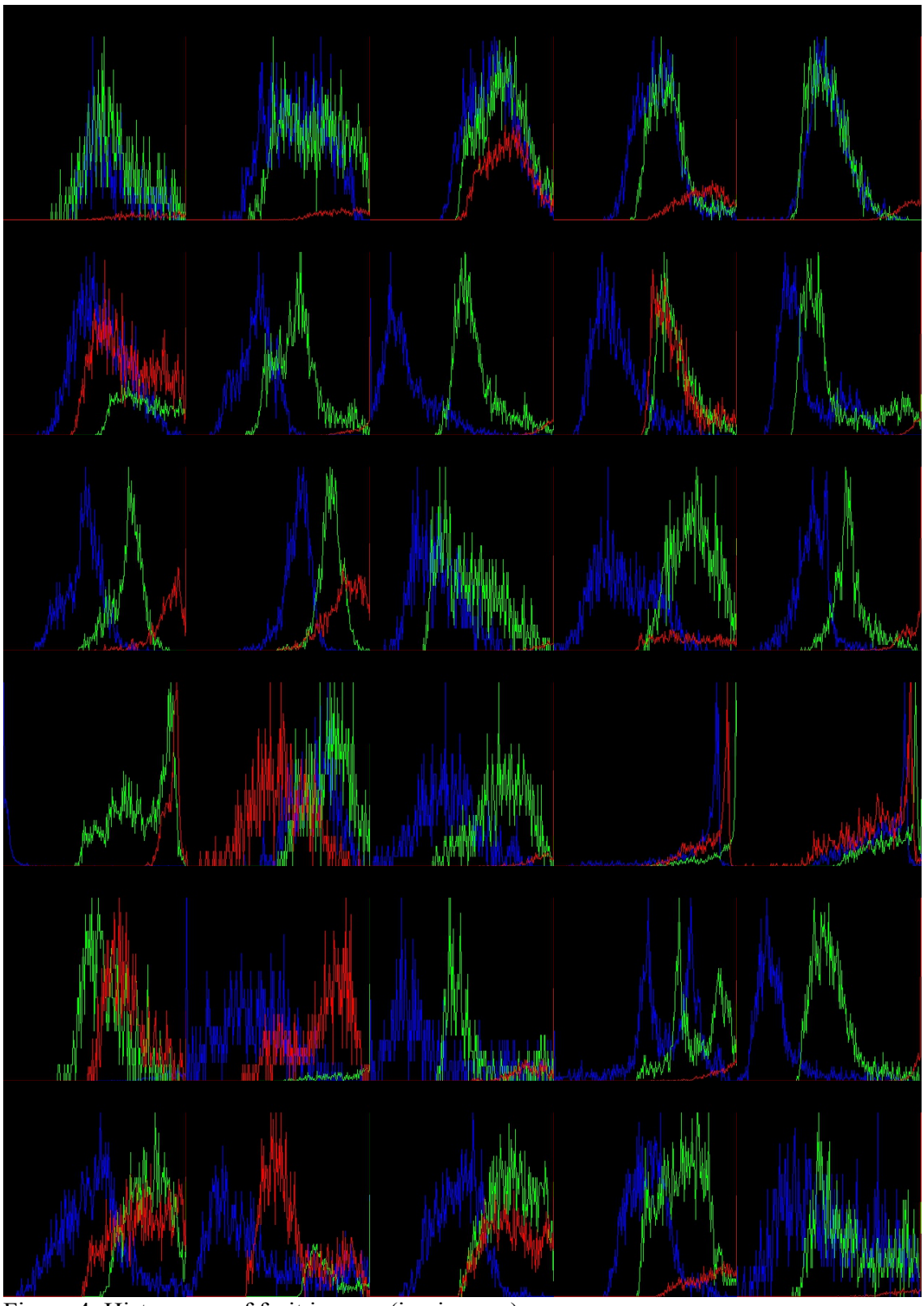


Figure 4: Histograms of fruit images (jpg images).

In my program, I put these codes of loading and calculating the jpg histograms inside the loop of loading and calculating ppm histograms. For each ppm image I use the compareHist() function provided by the Python OpenCV to compare with all 40-jpg histograms. According to the OpenCV documentation, the functions compareHist() compares two dense or two sparse histograms using specified methods, such as Correlation (method=CV_COMP_CORREL), Chi-Square (method=CV_COMP_CHISQR), Intersection (method=CV_COMP_INTERSECT) and Bhattacharyya distance (method=CV_COMP_BHATTACHARYYA). In this program, I use the simple the normalized L1 Correlation (CV_COMP_CORREL):

```
#Compare the correlation of two histograms and return a
numerical parameter that express how well two histograms
match with each other.
counter = cv2.compareHist(ppm_hist, jpg_hist,
cv.CV_COMP_CORREL)

#Initialization of the result interpretation.
if high_correlation == "" and low_correlation == "":
      high_counter = counter
      low_counter = counter
      high_correlation = jpg_infile
      low_correlation = jpg_infile
#Find the most similar jpg image.
if counter >= high_counter:
      high_counter = counter
      high_correlation = jpg_infile
#Find the most dissimilar jpg image.
if counter <= low_counter:
      low_counter = counter
      low_correlation = jpg_infile
```

The returned value of the compareHist() function is between -1 and 1. Number close to -1 means not related. Number close to 1 means related. The above code finds two-jpg image, one is most similar to the targeted ppm image (with the largest returned value) and the other is most dissimilar to the target image (with the lowest returned value) using three if-statements. These three if-statements compare new returned values with the current counter value. If it is greater than the counter value, the high_counter and high_correlation (image address) get updated. If it is lower than the counter value, the low_counter and low_correlation (image address) get updated.

Using the following codes, my program prints and saves the comparison results:

```
#Print the result using colored text supported by the termcolor
library of terminal.
print colored("Reading PPM Image: " + ppm_infile, 'red')
print colored("Most Similar JPG image: " + high_correlation + "
Correlation Value: " + str(high_counter), 'blue')
print colored("Most Dissimilar JPG image: " + low_correlation + "
Correlation Value: " + str(low_counter), 'blue')
print ""
```

```
img1 = cv2.imread(ppm_infile)
img2 = cv2.imread(high_correlation)
img3 = cv2.imread(low_correlation)

#Setup the result folders.
if not os.path.exists("color_matching_results/" +
ppm_infile.replace("images/", "").replace(".ppm", "/")):
      os.makedirs("color_matching_results/" +
      ppm_infile.replace("images/", "").replace(".ppm", "/"))

#Write result images to targeted folders.
cv2.imwrite("color_matching_results/" +
ppm_infile.replace("images/", "").replace(".ppm", "/") +
ppm_infile.replace("images/", ""), img1)

cv2.imwrite("color_matching_results/" +
ppm_infile.replace("images/", "").replace(".ppm", "/") +
high_correlation.replace("images/", ""), img2)

cv2.imwrite("color_matching_results/" +
ppm_infile.replace("images/", "").replace(".ppm", "/") +
low_correlation.replace("images/", ""), img3)
```

The printed results are:

```
dyn-160-39-29-75:assignment2 puconghan$ python color_matching_opencv_mask.py
Reading PPM Image: images/i01.ppm
Most Similar JPG image: images/i01.jpg     Correlation Value: 0.927718229972
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.0752414210177

Reading PPM Image: images/i02.ppm
Most Similar JPG image: images/i02.jpg     Correlation Value: 0.992992135948
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.200769161408

Reading PPM Image: images/i03.ppm
Most Similar JPG image: images/i03.jpg     Correlation Value: 0.961881816478
Most Dissimilar JPG image: images/i35.jpg  Correlation Value: 0.036659047028

Reading PPM Image: images/i04.ppm
Most Similar JPG image: images/i04.jpg     Correlation Value: 0.975827691357
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: -0.158224417865

Reading PPM Image: images/i05.ppm
Most Similar JPG image: images/i05.jpg     Correlation Value: 0.989066521849
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.0229586426177

Reading PPM Image: images/i06.ppm
Most Similar JPG image: images/i23.jpg     Correlation Value: 0.995593444527
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.100018966014

Reading PPM Image: images/i07.ppm
Most Similar JPG image: images/i07.jpg     Correlation Value: 0.993860999372
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.143095875381

Reading PPM Image: images/i08.ppm
Most Similar JPG image: images/i08.jpg     Correlation Value: 0.928374656879
Most Dissimilar JPG image: images/i37.jpg  Correlation Value: 0.0471230966661

Reading PPM Image: images/i09.ppm
Most Similar JPG image: images/i09.jpg     Correlation Value: 0.992888461666
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.134897343156

Reading PPM Image: images/i10.ppm
Most Similar JPG image: images/i10.jpg     Correlation Value: 0.971193492605
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.13584409714

Reading PPM Image: images/i11.ppm
Most Similar JPG image: images/i11.jpg     Correlation Value: 0.981924840683
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.0893359312828

Reading PPM Image: images/i12.ppm
Most Similar JPG image: images/i12.jpg     Correlation Value: 0.97519724949
```

```
Most Dissimilar JPG image: images/i37.jpg  Correlation Value: -0.0907931330532

Reading PPM Image: images/i13.ppm
Most Similar JPG image: images/i13.jpg     Correlation Value: 0.985231789442
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.00174290600438

Reading PPM Image: images/i14.ppm
Most Similar JPG image: images/i14.jpg     Correlation Value: 0.984970953954
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.205357544478

Reading PPM Image: images/i15.ppm
Most Similar JPG image: images/i15.jpg     Correlation Value: 0.996030398848
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.218545737133

Reading PPM Image: images/i16.ppm
Most Similar JPG image: images/i16.jpg     Correlation Value: 0.950659446779
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.132146075142

Reading PPM Image: images/i17.ppm
Most Similar JPG image: images/i17.jpg     Correlation Value: 0.998711544673
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.126863365324

Reading PPM Image: images/i18.ppm
Most Similar JPG image: images/i18.jpg     Correlation Value: 0.998367256798
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.136241875653

Reading PPM Image: images/i19.ppm
Most Similar JPG image: images/i19.jpg     Correlation Value: 0.982155576387
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: -0.082803579592

Reading PPM Image: images/i20.ppm
Most Similar JPG image: images/i20.jpg     Correlation Value: 0.999205571603
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.122092396766

Reading PPM Image: images/i21.ppm
Most Similar JPG image: images/i21.jpg     Correlation Value: 0.98006429026
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.338727611779

Reading PPM Image: images/i22.ppm
Most Similar JPG image: images/i22.jpg     Correlation Value: 0.989956928614
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.346059817043

Reading PPM Image: images/i23.ppm
Most Similar JPG image: images/i23.jpg     Correlation Value: 0.997613409889
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.103378791483

Reading PPM Image: images/i24.ppm
Most Similar JPG image: images/i24.jpg     Correlation Value: 0.970874602253
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: 0.0414107922321

Reading PPM Image: images/i25.ppm
Most Similar JPG image: images/i25.jpg     Correlation Value: 0.993006516899
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.240076221665

Reading PPM Image: images/i26.ppm
Most Similar JPG image: images/i26.jpg     Correlation Value: 0.995359916285
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.282653281866

Reading PPM Image: images/i27.ppm
Most Similar JPG image: images/i27.jpg     Correlation Value: 0.68580400586
Most Dissimilar JPG image: images/i22.jpg  Correlation Value: -0.333438623683

Reading PPM Image: images/i28.ppm
Most Similar JPG image: images/i28.jpg     Correlation Value: 0.992040557736
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.157629995778

Reading PPM Image: images/i29.ppm
Most Similar JPG image: images/i29.jpg     Correlation Value: 0.994074956087
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.226865032673

Reading PPM Image: images/i30.ppm
Most Similar JPG image: images/i30.jpg     Correlation Value: 0.959922242765
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.047646045301

Reading PPM Image: images/i31.ppm
Most Similar JPG image: images/i31.jpg     Correlation Value: 0.901602029107
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: -0.0996940859584

Reading PPM Image: images/i32.ppm
Most Similar JPG image: images/i32.jpg     Correlation Value: 0.776938968855
Most Dissimilar JPG image: images/i35.jpg  Correlation Value: -0.0598102928213

Reading PPM Image: images/i33.ppm
Most Similar JPG image: images/i33.jpg     Correlation Value: 0.944665387836
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.204057001875

Reading PPM Image: images/i34.ppm
Most Similar JPG image: images/i34.jpg     Correlation Value: 0.996523102256
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.18844056827

Reading PPM Image: images/i35.ppm
Most Similar JPG image: images/i35.jpg     Correlation Value: 0.997786895868
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.136752266684
```

```
Reading PPM Image: images/i36.ppm
Most Similar JPG image: images/i36.jpg     Correlation Value: 0.956678650144
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.0199393349813

Reading PPM Image: images/i37.ppm
Most Similar JPG image: images/i37.jpg     Correlation Value: 0.941149149512
Most Dissimilar JPG image: images/i39.jpg  Correlation Value: -0.114999984444

Reading PPM Image: images/i38.ppm
Most Similar JPG image: images/i38.jpg     Correlation Value: 0.97138177875
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.0360710625406

Reading PPM Image: images/i39.ppm
Most Similar JPG image: images/i39.jpg     Correlation Value: 0.989850769354
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.252548509281

Reading PPM Image: images/i40.ppm
Most Similar JPG image: images/i40.jpg     Correlation Value: 0.996929666618
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.115605230882
```

Figure 5: Printed results for the histogram comparison

Here is the table for detailed results

Format: (Original ppm image) + (Most Similar jpg Image) + (Most Dissimilar jpg Image)

i01.ppm 	i01.ppm 	i26.ppm 
i02.ppm 	i02.ppm 	i27.ppm 
i03.ppm 	i03.ppm 	i35.ppm 
i04.ppm 	i04.ppm 	i26.ppm 
i05.ppm 	i05.ppm 	i27.ppm 
i06.ppm	i23.ppm	i27.ppm

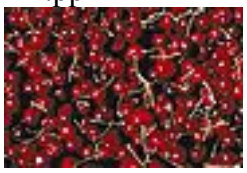
i07.ppm

i08.ppm

i09.ppm

i10.ppm

i11.ppm



i12.ppm

i13.ppm

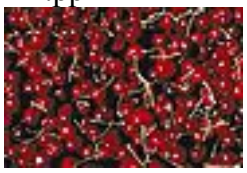
i07.ppm

i08.ppm

i09.ppm

i10.ppm

i11.ppm



i12.ppm

i13.ppm

i27.ppm

i37.ppm

i27.ppm

i26.ppm

i27.ppm

i37.ppm

i27.ppm

i14.ppm

i15.ppm

i16.ppm

i17.ppm

i18.ppm

i19.ppm

i20.ppm

i14.ppm

i15.ppm

i16.ppm

i17.ppm

i18.ppm

i19.ppm

i20.ppm

i27.ppm

i27.ppm

i26.ppm

i27.ppm

i27.ppm

i26.ppm

i27.ppm

i21.ppm

i22.ppm

i23.ppm

i24.ppm

i25.ppm

i26.ppm

i27.ppm

i21.ppm

i22.ppm

i23.ppm

i24.ppm

i25.ppm

i26.ppm

i27.ppm

i27.ppm

i27.ppm

i27.ppm

i27.ppm

i27.ppm

i27.ppm

i22.ppm

i28.ppm

i29.ppm

i30.ppm

i31.ppm

i32.ppm

i33.ppm

i34.ppm

i28.ppm

i29.ppm

i30.ppm

i31.ppm

i32.ppm

i33.ppm

i34.ppm

i27.ppm

i27.ppm

i27.ppm

i26.ppm

i35.ppm

i27.ppm

i27.ppm



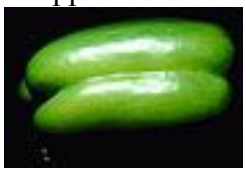
i35.ppm i35.ppm i27.ppm



i36.ppm i36.ppm i27.ppm



i37.ppm i37.ppm i39.ppm



i38.ppm i38.ppm i27.ppm



i39.ppm i39.ppm i27.ppm



i40.ppm i40.ppm i27.ppm



As we can see from the result, the sixth ppm image does not match the sixth jpg image. Comparing their histograms in Figure 6, we can see that they have various blue and green

distributions. Instead of matching color distributions of image i006.jpg, color distributions of image i006.ppm matches the color distribution of image i023.jpg.

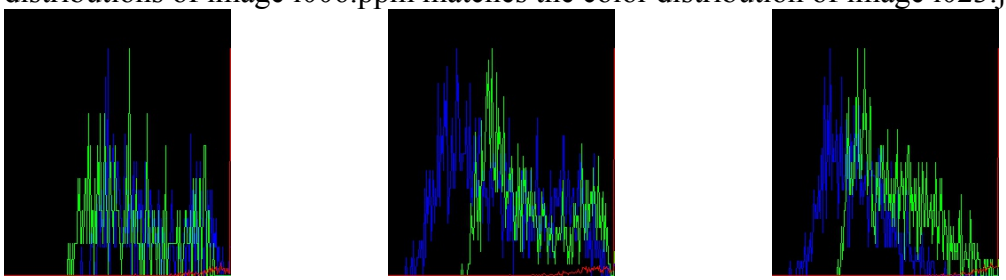


Figure 6: Histograms of i006.ppm, i006.jpg and i023.jpg

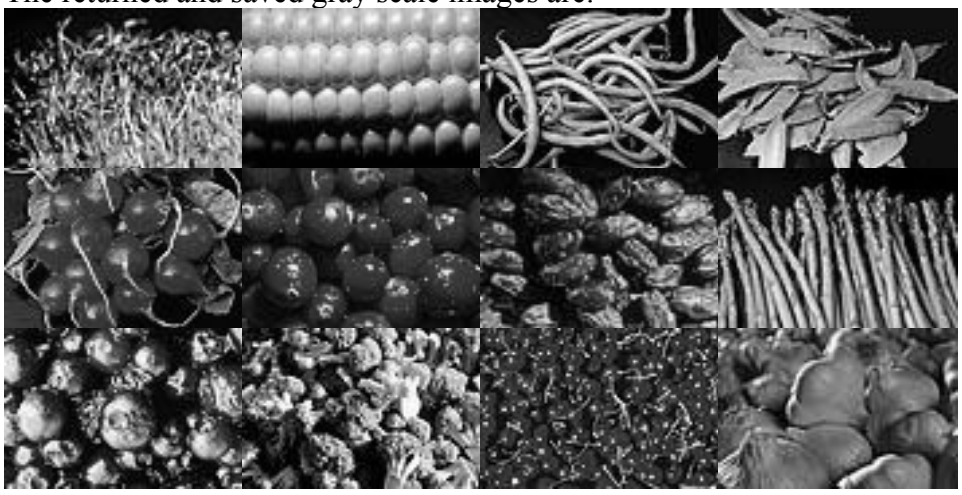
Step 2: Gross Texture Matching

Python app: texture_matching.py

My implementation for the second step is similar to my implementation for the first step. My program first reads each ppm image using a for-loop. Inside the for-loop, my program converts the color images into black and white ones (gray scale image) using the cvtColor(image,cv2.COLOR_BGR2GRAY) function:

```
for ppm_infile in glob.glob( os.path.join("images", '*.ppm') ):
      #Loads each ppm images from the image folder.
      ppm_img = cv2.imread(ppm_infile)
      #For drawing the histogram.
      ppm_h = np.zeros((300,256,3))
      #Convert ppm images to gray scale images.
      ppm_imgray = cv2.cvtColor(ppm_img,cv2.COLOR_BGR2GRAY)
      #Save gray scale ppm images.
      cv2.imwrite("grayscale/" + ppm_infile.replace("images/",
      "").replace(".ppm", ".jpg"), ppm_imgray)
```

The returned and saved gray scale images are:



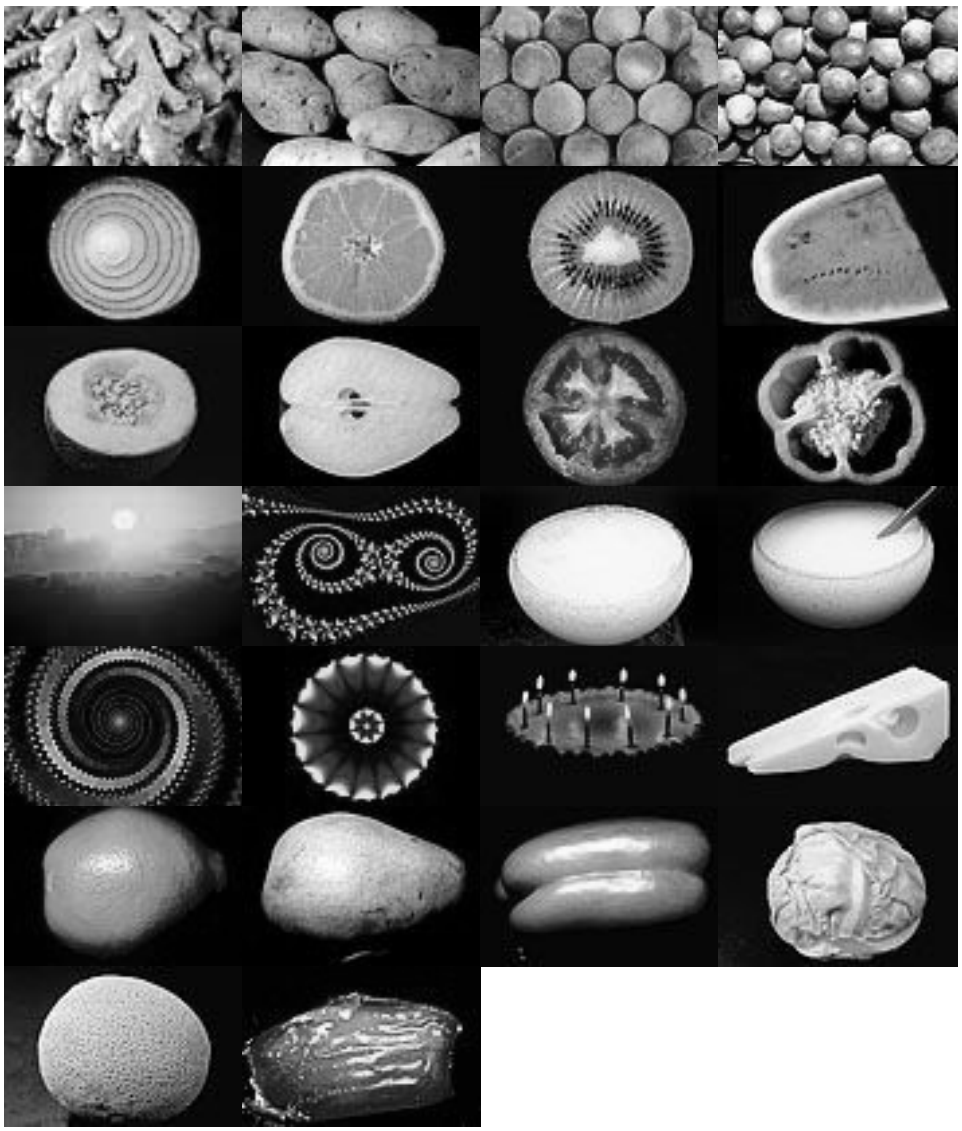


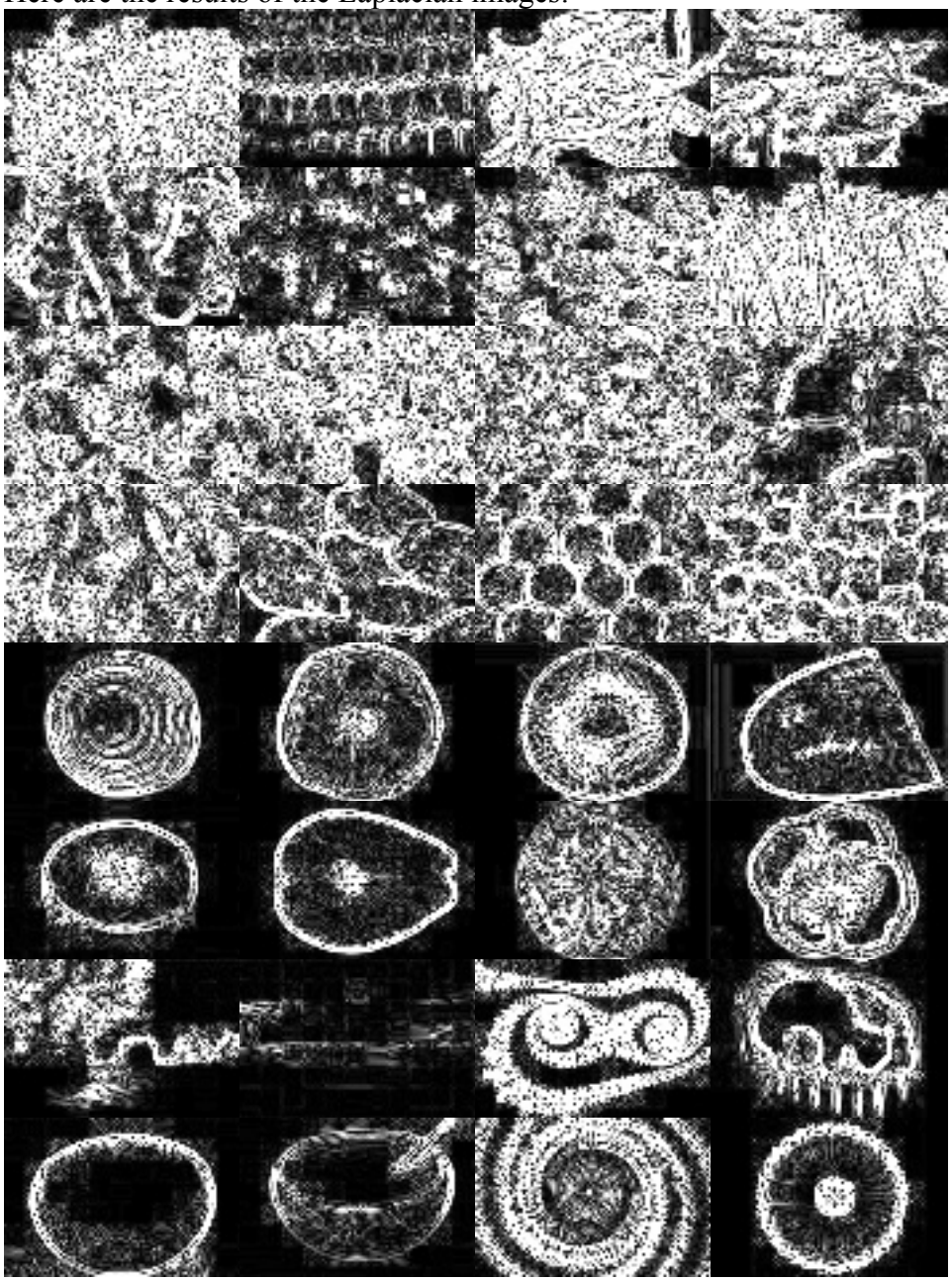
Figure 7: Gray scale images.

For each gray scale image, my program created a "Laplacian" image using the Laplacian() provided by Python OpenCV. According to the OpenCV documentation, the function calculates the Laplacian of the source image by adding up the second x and y derivatives calculated using the Sobel operator, a discrete differentiation operator computes an approximation of the gradient of an image intensity function (OpenCV Sobel Derivatives). In other word, the Laplacian() function is the sum of second derivatives in both the directions. It measures changes in both horizontal and vertical directions. My program also uses the convertScaleAbs() function to convert the output laplacian image to an 8-bit unsigned integer and save images to the laplacian folder.

Here is my implementation for converting "Laplacian" images:

```
#Calculates the Laplacian of an image. The function calculates
the Laplacian of the source image by adding up the second x and y
derivatives calculated using the Sobel operator.
ppm_gray_lap =
cv2.Laplacian(ppm_imgray,cv2.CV_16S,ksize=3,scale=1,delta=0)
#Converts input array elements to another 8-bit unsigned integer
with optional linear transformation.
ppm_dst = cv2.convertScaleAbs(ppm_gray_lap)
#Save paplacian ppm images.
cv2.imwrite("laplacian/" + ppm_infile.replace("images/",
"").replace(".ppm", ".jpg"), ppm_dst)
```

Here are the results of the Laplacian images:



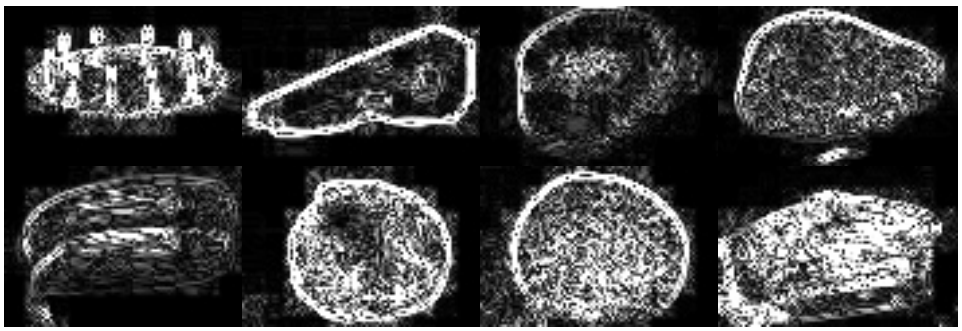


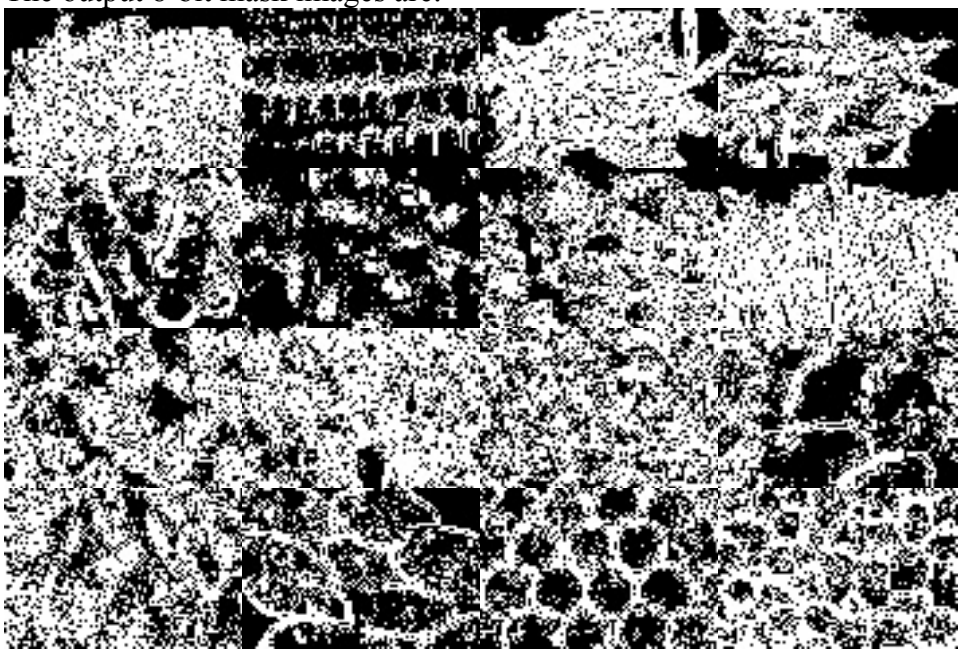
Figure 8: Laplacian images

Similar to step one, my program will compute the 8-bit mask of the images using the threshold() function. Using these returned images, my program can distinguish the body of images from their black backgrounds. Instead of using the gray scale images, my new implementation creates the mask images using the Laplacian images. Laplacian images allow the threshold() functions to pick up details of original images, which can improve the accuracy of ignoring black backgrounds.

Here is my implementation:

```
#Create ppm binary mask, a 8-bit image, where white denotes that
region should be used for histogram calculations, and black means
it should not.
ppm_ret, ppm_thresh_mask = cv2.threshold(ppm_dst,127,255,0)
#Save 8 bit mask images.
cv2.imwrite("8bitmask/" + ppm_infile.replace("images/",
"").replace(".ppm", ".jpg"), ppm_thresh_mask)
```

The output 8-bit mask images are:



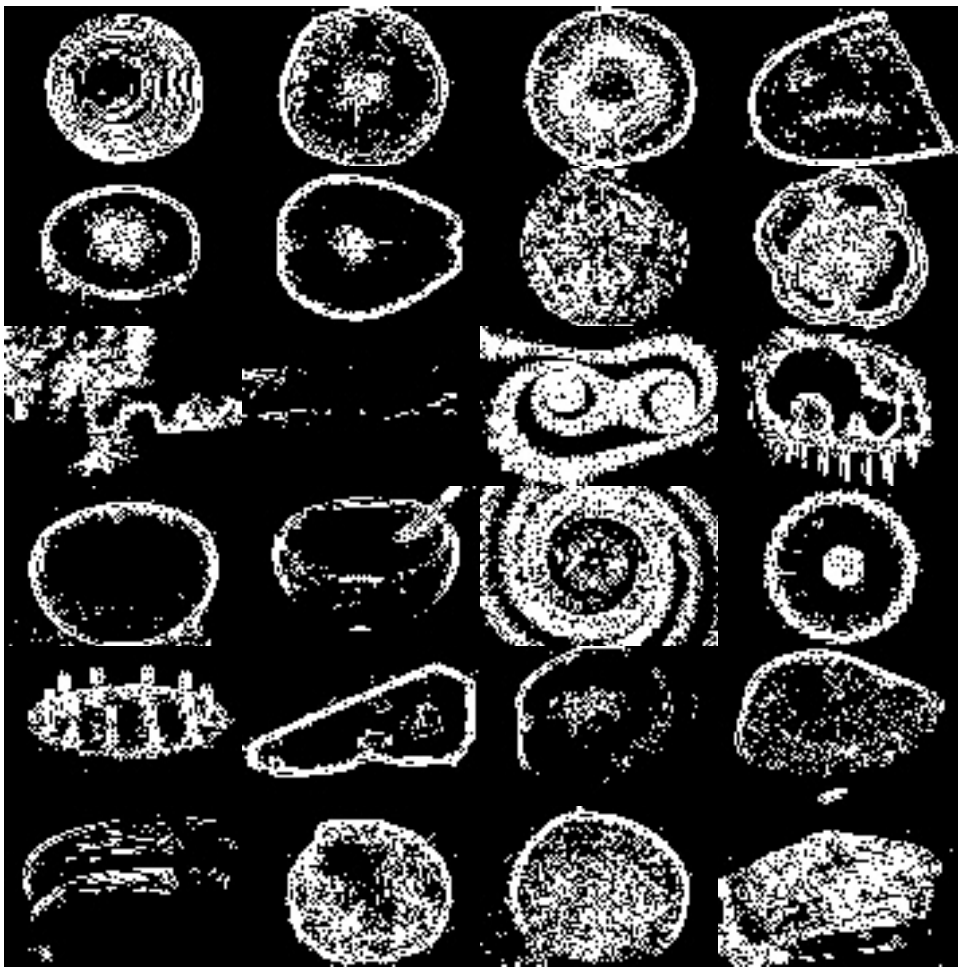


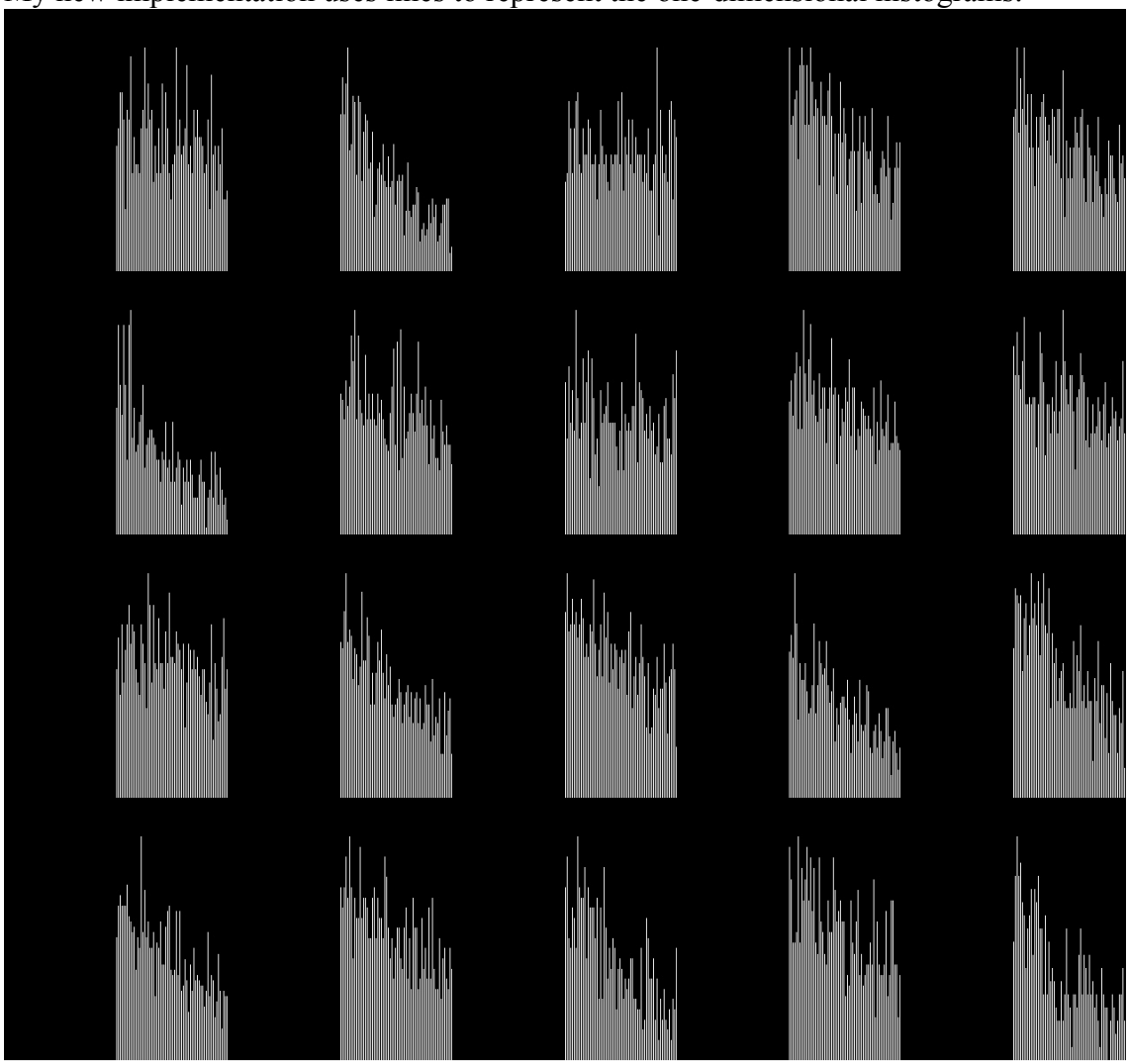
Figure 9: 8-bit mask images.

Similar to step one, my program compute histograms using the calcHist() function and normalizes the generated histogram using the normalize() function. Before saving the one-dimensional histogram, my new program again flips the y-axis:

```
#Calculates a histogram for the 8-bit region marked in the mask
using a set of arrays.
ppm_hist =
cv2.calcHist([ppm_dst],[0],ppm_thresh_mask,[256],[0,255])
#Normalize the value to fall below 255 and fit in image height.
cv2.normalize(ppm_hist,ppm_hist,0,255,cv2.NORM_MINMAX)
#Evenly round to the given number of decimals. For values exactly
halfway between rounded decimal values, Numpy rounds to the
nearest even value.
round_ppm_hist = np.int32(np.around(ppm_hist))
for x,fliped_ppm_y in enumerate(round_ppm_hist):
      cv2.line(ppm_h,(x,0),(x,fliped_ppm_y),(255,255,255))
      fliped_ppm_y = np.flipud(ppm_h)
#Save histograms.
```

```
cv2.imwrite("one_dimensional_histograms/ppm/" +
ppm_infile.replace("images/", "").replace(".ppm", ".jpg"),
fliped_ppm_y)
```

My new implementation uses lines to represent the one-dimensional histograms:



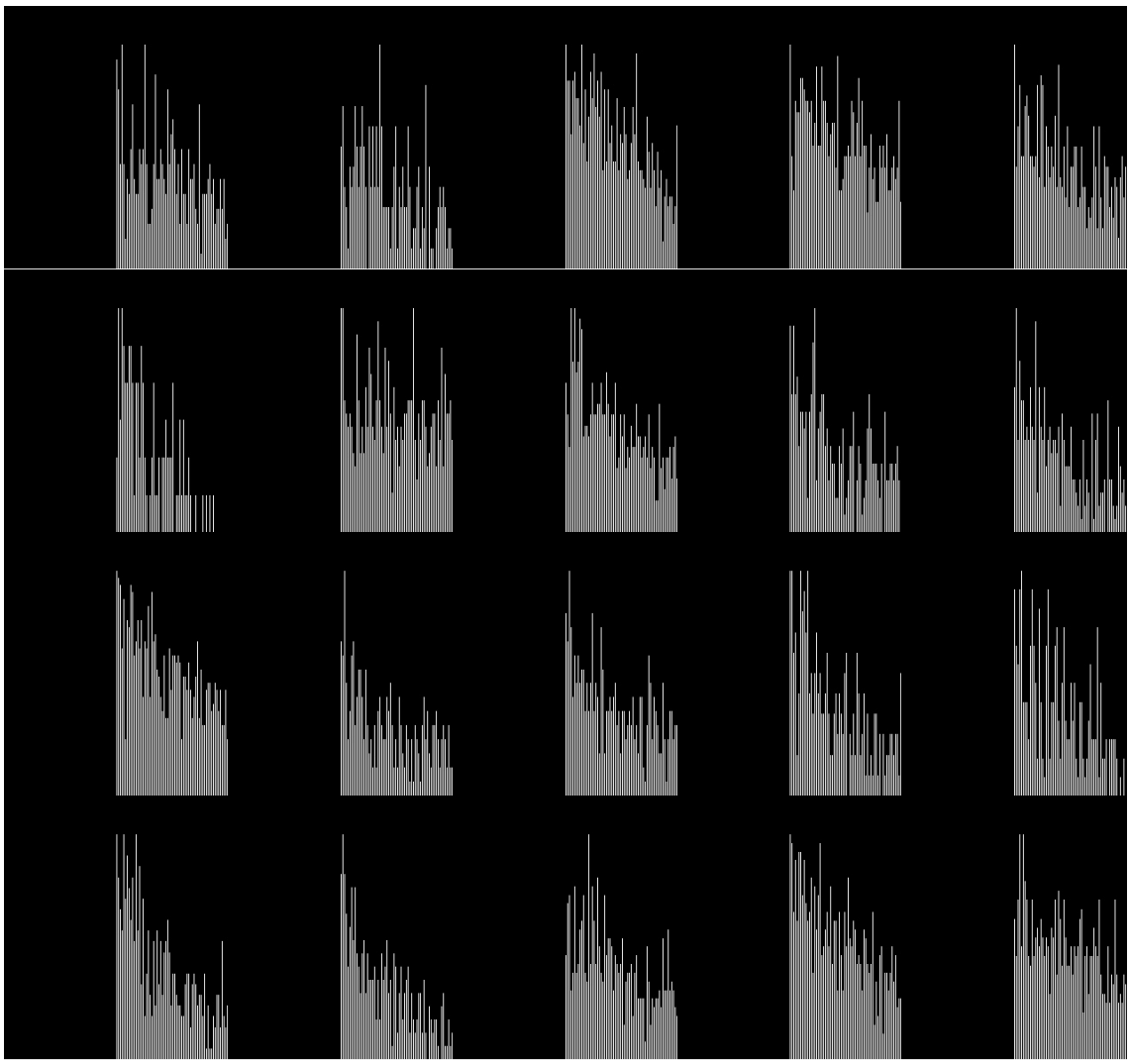


Figure 10: One-dimensional histograms for ppm images.

These 40 histograms contain black and white information only in the white blobs. In other words, they do not contain information in the black background. The black background has been filtered. To show the difference, I print only the first five histograms without passing the mask image (without background ignoring process). Here are the results:

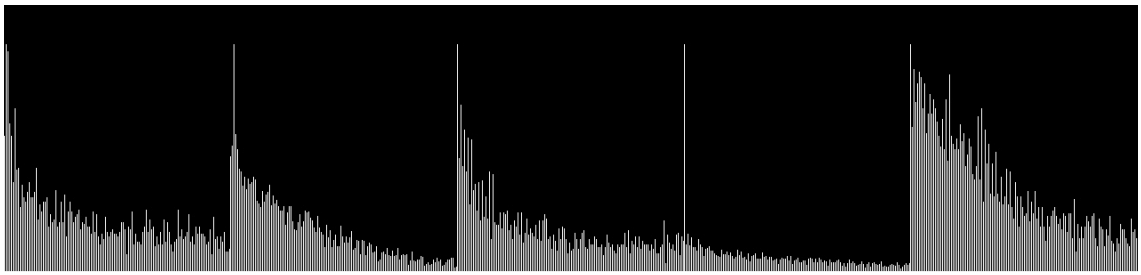


Figure 11: Selected one-dimensional histograms for ppm images with black background.

According to the Photographer Ron Bigelow, histogram displays tonal distributions. The darkest tone is assigned a value of zero. The lightest tone is assigned a value of 255. “The horizontal axis displays these tones from the darkest on the left to the lightest on the right,” said Bigelow. As we can see from figure 11, with the black background color, the histogram tone distributions are pushed to the left of the scale. Images with dissimilar objects that have a lot of black background pixels will have similarities in histogram color distribution. Such impacts will make later histogram correlation comparison to become less accurate.

To avoid matching the black background, my program takes advantages of mask images and filtered the black background.

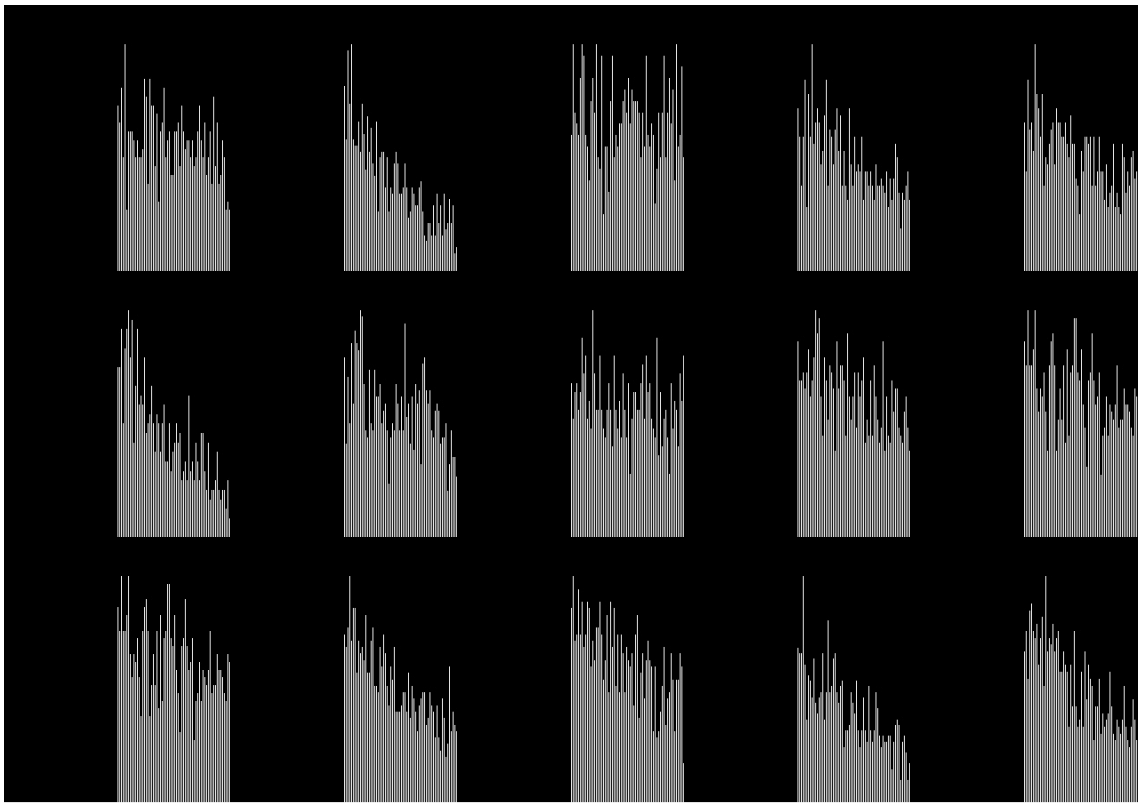
Similar to step one, for each ppm image histogram, my program will compare it with all 40 jpg images histogram and find the most similar and dissimilar images using a for loop.

Code for generating histograms for the 40-jpg images (inside the ppm for-loop):

```
for jpg_infile in glob.glob( os.path.join("images", '*.jpg') ):
      #Loads each jpg images from the image folder.
      jpg_img = cv2.imread(jpg_infile)
      #For drawing the histogram.
      jpg_h = np.zeros((300,256,3))
      #Convert jpg image to gray scale.
      jpg_imgray = cv2.cvtColor(jpg_img,cv2.COLOR_BGR2GRAY)
      #Save gray scale jpg images.
      cv2.imwrite("grayscale/jpg/" + jpg_infile.replace("images/",
      "").replace(".ppm", ".jpg"), jpg_imgray)

      #Calculates the Laplacian of an image. The function
      calculates the Laplacian of the source image by adding up
      the second x and y derivatives calculated using the Sobel
      operator.
      jpg_gray_lap =
      cv2.Laplacian(jpg_imgray,cv2.CV_16S,ksize=3,scale=1,delta=0)
      #Converts input array elements to another 8-bit unsigned
      integer with optional linear transformation.
      jpg_dst = cv2.convertScaleAbs(jpg_gray_lap)
      #Save paplacian jpg images.
      cv2.imwrite("laplacian/jpg/" + jpg_infile.replace("images/",
      "").replace(".ppm", ".jpg"), jpg_dst)

      #Create jpg binary mask, a 8-bit image, where white denotes
      that region should be used for histogram calculations, and
      black means it should not.
      jpg_ret, jpg_thresh_mask = cv2.threshold(jpg_dst,127,255,0)
      #Save 8 bit mask images.
      cv2.imwrite("8bitmask/jpg/" + jpg_infile.replace("images/",
      "").replace(".ppm", ".jpg"), jpg_thresh_mask)

      #Calculates a histogram for the 8-bit region marked in the
      mask using a set of arrays.
```

```
jpg_hist =
cv2.calcHist([jpg_dst],[0],jpg_thresh_mask,[256],[0,255])
#Normalize the value to fall below 255 and fit in image
height.
cv2.normalize(jpg_hist,jpg_hist,0,255,cv2.NORM_MINMAX)
#Evenly round to the given number of decimals. For values
exactly halfway between rounded decimal values, Numpy
rounds to the nearest even value.
round_jpg_hist = np.int32(np.around(jpg_hist))
for x,fliped_jpg in enumerate(round_jpg_hist):
     cv2.line(jpg_h,(x,0),(x,fliped_jpg),(255,255,255))
#Flip the image vertically.
fliped_jpg = np.flipud(jpg_h)
#Save one dimensional histogram
cv2.imwrite("one_dimensional_histograms/jpg/" +
jpg_infile.replace("images/", "").replace(".ppm", ".jpg"),
fliped_jpg)
```

The results of these 40 jpg one-dimensional histograms, as shown in Figure 12, are similar to the 40 ppm one-dimensional histograms in Figure 10. Although many of these images are the same, there are some minor variations in pick values.



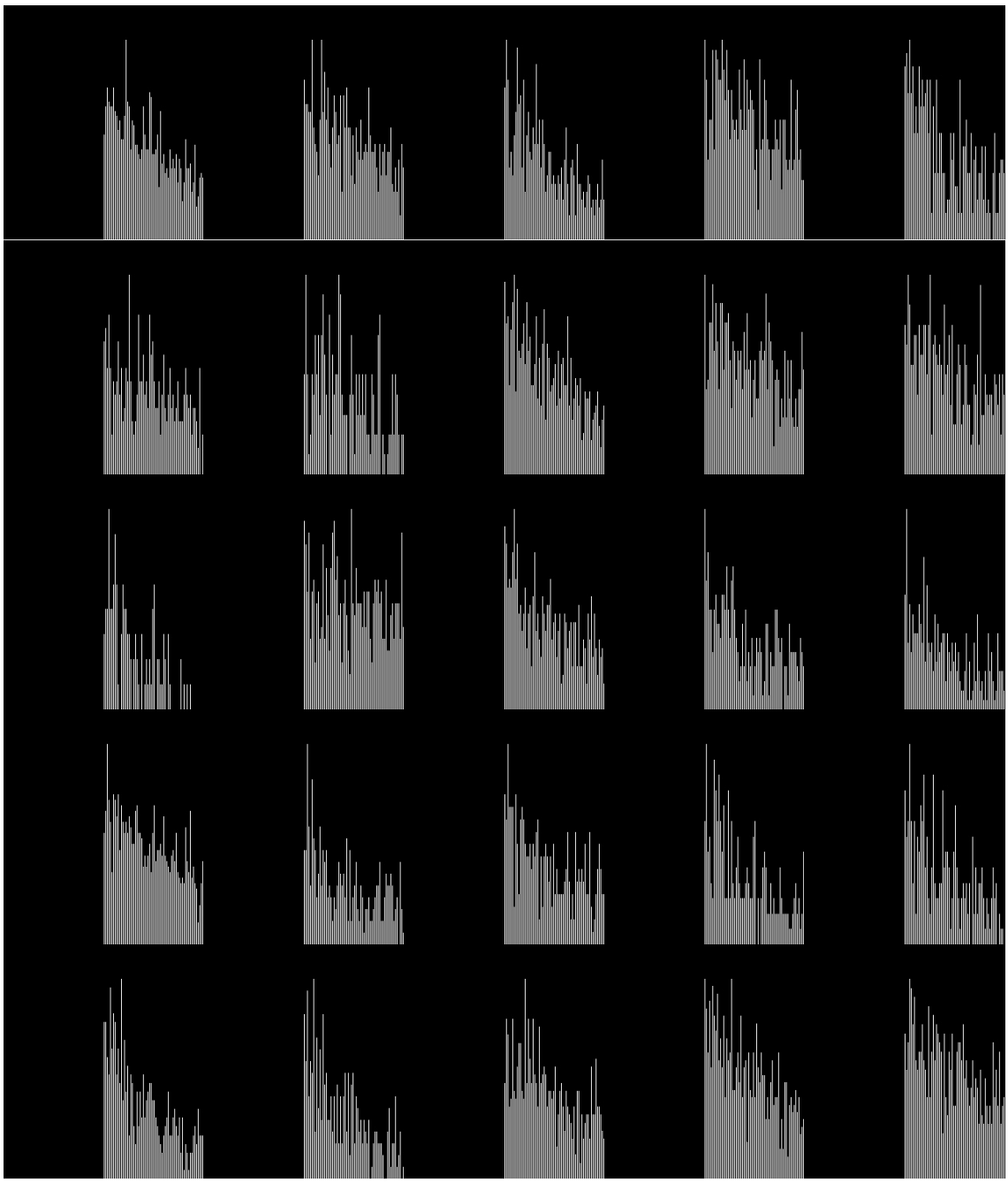


Figure 12: One-dimensional histograms for jpg images.

Similar to the previous step, my new program puts these codes of loading and calculating the jpg histograms inside the loop of loading and calculating ppm histograms. For each ppm image, I use the compareHist() to compare with all 40-jpg histograms:

```
#Compare the correlation of two histograms and return a numerical
parameter that express how well two histograms match with each
other.
counter = cv2.compareHist(ppm_hist, jpg_hist, cv.CV_COMP_CORREL)
```

```
#Initialization of the result interpretation.
if high_correlation == "" and low_correlation == "":
      high_counter = counter
      low_counter = counter
      high_correlation = jpg_infile
      low_correlation = jpg_infile
#Find the most similar jpg image.
if counter >= high_counter:
      high_counter = counter
      high_correlation = jpg_infile
#Find the most dissimilar jpg image.
if counter <= low_counter:
      low_counter = counter
      low_correlation = jpg_infile
```

Again, the returned value of the compareHist() function is between -1 and 1. Number close to -1 means not related. Number close to 1 means related. The above code finds two-jpg image, one is most similar to the targeted ppm image (with the largest returned value) and the other is most dissimilar to the target image (with the lowest returned value) using three if-statements. These three if-statements compare new returned values with the current counter value. If it is greater than the counter value, the high_counter and high_correlation (image address) get updated. If it is lower than the counter value, the low_counter and low_correlation (image address) get updated.

Using the following codes, my program prints and saves the comparison results:

```
      #Print the result using colored text supported by the
      termcolor library of terminal.
      print colored("Reading PPM Image: " + ppm_infile, 'red')
      print colored("Most Similar JPG image: " + high_correlation
      + "     Correlation Value: " + str(high_counter), 'blue')
      print colored("Most Dissimilar JPG image: " +
      low_correlation + "  Correlation Value: " +
      str(low_counter), 'blue')
      print ""

      img1 = cv2.imread(ppm_infile)
      img2 = cv2.imread(high_correlation)
      img3 = cv2.imread(low_correlation)

      #Setup the result folders.
      if not os.path.exists("texture_matching_results/" +
      ppm_infile.replace("images/", "").replace(".ppm", "/")):
            os.makedirs("texture_matching_results/" +
            ppm_infile.replace("images/", "").replace(".ppm", "/"))
      #Write result images to targeted folders.
      cv2.imwrite("texture_matching_results/" +
      ppm_infile.replace("images/", "").replace(".ppm", "/") +
      ppm_infile.replace("images/", ""), img1)
```

```
cv2.imwrite("texture_matching_results/" +
ppm_infile.replace("images/", "").replace(".ppm", "/") +
high_correlation.replace("images/", ""), img2)
cv2.imwrite("texture_matching_results/" +
ppm_infile.replace("images/", "").replace(".ppm", "/") +
low_correlation.replace("images/", ""), img3)
```

The printed results are:

```
dyn-160-39-29-75:assignment2 puconghan$ python texture_matching.py
Reading PPM Image: images/i01.ppm
Most Similar JPG image: images/i01.jpg     Correlation Value: 0.96895864939
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.645993159483

Reading PPM Image: images/i02.ppm
Most Similar JPG image: images/i02.jpg     Correlation Value: 0.973556303426
Most Dissimilar JPG image: images/i22.jpg  Correlation Value: 0.779256107336

Reading PPM Image: images/i03.ppm
Most Similar JPG image: images/i03.jpg     Correlation Value: 0.952303565933
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.659735628474

Reading PPM Image: images/i04.ppm
Most Similar JPG image: images/i04.jpg     Correlation Value: 0.962552808163
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.717015079341

Reading PPM Image: images/i05.ppm
Most Similar JPG image: images/i05.jpg     Correlation Value: 0.966845381494
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.682675129452

Reading PPM Image: images/i06.ppm
Most Similar JPG image: images/i02.jpg     Correlation Value: 0.93648523309
Most Dissimilar JPG image: images/i22.jpg  Correlation Value: 0.766724306192

Reading PPM Image: images/i07.ppm
Most Similar JPG image: images/i07.jpg     Correlation Value: 0.960729379584
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.698725953948

Reading PPM Image: images/i08.ppm
Most Similar JPG image: images/i08.jpg     Correlation Value: 0.964738217848
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.641376097916

Reading PPM Image: images/i09.ppm
Most Similar JPG image: images/i09.jpg     Correlation Value: 0.967919980611
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.695672799328

Reading PPM Image: images/i10.ppm
Most Similar JPG image: images/i10.jpg     Correlation Value: 0.989992301239
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.689937580411

Reading PPM Image: images/i11.ppm
Most Similar JPG image: images/i11.jpg     Correlation Value: 0.976967412017
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.656968263741

Reading PPM Image: images/i12.ppm
Most Similar JPG image: images/i12.jpg     Correlation Value: 0.987923624684
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.762071126158

Reading PPM Image: images/i13.ppm
Most Similar JPG image: images/i13.jpg     Correlation Value: 0.997807287499
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.702450255875

Reading PPM Image: images/i14.ppm
Most Similar JPG image: images/i14.jpg     Correlation Value: 0.992983891296
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.756651212029

Reading PPM Image: images/i15.ppm
Most Similar JPG image: images/i15.jpg     Correlation Value: 0.970260520432
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.752255550598

Reading PPM Image: images/i16.ppm
Most Similar JPG image: images/i16.jpg     Correlation Value: 0.988197821801
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.739358178838

Reading PPM Image: images/i17.ppm
Most Similar JPG image: images/i17.jpg     Correlation Value: 0.957166627944
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.706584638854

Reading PPM Image: images/i18.ppm
Most Similar JPG image: images/i12.jpg     Correlation Value: 0.921861902623
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.729824257212

Reading PPM Image: images/i19.ppm
Most Similar JPG image: images/i19.jpg     Correlation Value: 0.983082823699
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.669872061411

Reading PPM Image: images/i20.ppm
Most Similar JPG image: images/i02.jpg     Correlation Value: 0.933427147069
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.769560586686

Reading PPM Image: images/i21.ppm
Most Similar JPG image: images/i21.jpg     Correlation Value: 0.966799981046
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.698853541223

Reading PPM Image: images/i22.ppm
Most Similar JPG image: images/i22.jpg     Correlation Value: 0.93899380246
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.619701130583

Reading PPM Image: images/i23.ppm
Most Similar JPG image: images/i23.jpg     Correlation Value: 0.945861957487
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.720419927801

Reading PPM Image: images/i24.ppm
```

```
Most Similar JPG image: images/i24.jpg     Correlation Value: 0.968817916893
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.712223958897

Reading PPM Image: images/i25.ppm
Most Similar JPG image: images/i25.jpg     Correlation Value: 0.957527308361
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.70606561619

Reading PPM Image: images/i26.ppm
Most Similar JPG image: images/i26.jpg     Correlation Value: 0.912225033471
Most Dissimilar JPG image: images/i22.jpg  Correlation Value: 0.617480553714

Reading PPM Image: images/i27.ppm
Most Similar JPG image: images/i27.jpg     Correlation Value: 0.9116927309
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.59497627626

Reading PPM Image: images/i28.ppm
Most Similar JPG image: images/i12.jpg     Correlation Value: 0.952912510734
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.759541581223

Reading PPM Image: images/i29.ppm
Most Similar JPG image: images/i29.jpg     Correlation Value: 0.9449019088
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.701292083722

Reading PPM Image: images/i30.ppm
Most Similar JPG image: images/i30.jpg     Correlation Value: 0.962866131408
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.751924436415

Reading PPM Image: images/i31.ppm
Most Similar JPG image: images/i31.jpg     Correlation Value: 0.96043232871
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.732052690697

Reading PPM Image: images/i32.ppm
Most Similar JPG image: images/i32.jpg     Correlation Value: 0.905751844279
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.700767320888

Reading PPM Image: images/i33.ppm
Most Similar JPG image: images/i12.jpg     Correlation Value: 0.935556131004
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.72836674793

Reading PPM Image: images/i34.ppm
Most Similar JPG image: images/i34.jpg     Correlation Value: 0.955223703319
Most Dissimilar JPG image: images/i22.jpg  Correlation Value: 0.715276821477

Reading PPM Image: images/i35.ppm
Most Similar JPG image: images/i35.jpg     Correlation Value: 0.881974542226
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.670611767992

Reading PPM Image: images/i36.ppm
Most Similar JPG image: images/i36.jpg     Correlation Value: 0.985981597256
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.740145885774

Reading PPM Image: images/i37.ppm
Most Similar JPG image: images/i37.jpg     Correlation Value: 0.925380990277
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.767386676797

Reading PPM Image: images/i38.ppm
Most Similar JPG image: images/i38.jpg     Correlation Value: 0.994675202817
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.66254078696

Reading PPM Image: images/i39.ppm
Most Similar JPG image: images/i39.jpg     Correlation Value: 0.994704147332
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.748916441973

Reading PPM Image: images/i40.ppm
Most Similar JPG image: images/i12.jpg     Correlation Value: 0.942737826778
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.709933858208
```

Figure 13: Printed results for the histogram comparison

Here is the table for detailed results

Format: (Original ppm image) + (Most Similar jpg Image) + (Most Dissimilar jpg Image)

i01.ppm

i01.ppm

i26.ppm

i02.ppm

i02.ppm

i22.ppm

i03.ppm

i03.ppm

i26.ppm

i04.ppm

i05.ppm

i06.ppm

i07.ppm

i08.ppm

i09.ppm

i10.ppm

i04.ppm

i05.ppm

i02.ppm

i07.ppm

i08.ppm

i09.ppm

i10.ppm

i26.ppm

i26.ppm

i27.ppm

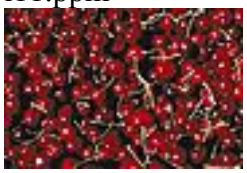
i26.ppm

i26.ppm

i26.ppm

i26.ppm

i11.ppm



i12.ppm

i13.ppm

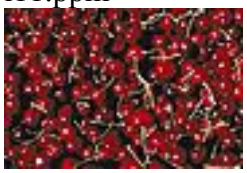
i14.ppm

i15.ppm

i16.ppm

i17.ppm

i11.ppm



i12.ppm

i13.ppm

i14.ppm

i15.ppm

i16.ppm

i17.ppm

i26.ppm

i26.ppm

i26.ppm

i26.ppm

i26.ppm

i26.ppm

i26.ppm

i18.ppm

i19.ppm

i20.ppm

i21.ppm

i22.ppm

i23.ppm

i24.ppm

i12.ppm

i19.ppm

i02.ppm

i21.ppm

i22.ppm

i23.ppm

i24.ppm

i26.ppm

i26.ppm

i26.ppm

i26.ppm

i26.ppm

i26.ppm

i26.ppm

i25.ppm

i26.ppm

i27.ppm

i28.ppm

i29.ppm

i30.ppm

i31.ppm

i25.ppm

i26.ppm

i27.ppm

i12.ppm

i29.ppm

i30.ppm

i31.ppm

i26.ppm

i22.ppm

i26.ppm

i26.ppm

i26.ppm

i26.ppm

i26.ppm

i32.ppm

i33.ppm

i34.ppm

i35.ppm

i36.ppm

i37.ppm

i38.ppm

i32.ppm

12.ppm

i34.ppm

i35.ppm

i36.ppm

i37.ppm

i38.ppm

i26.ppm

i26.ppm

i26.ppm

i22.ppm

i26.ppm

i26.ppm

i26.ppm



i39.ppm i39.ppm i26.ppm

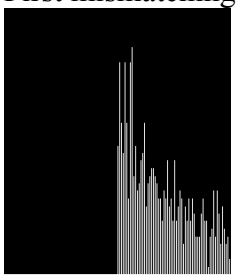


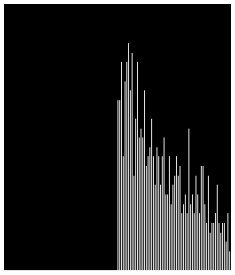
i40.ppm i12.ppm i26.ppm

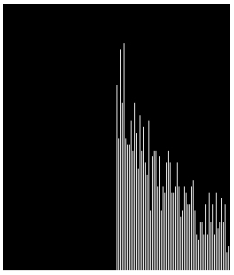


As we can see from the results, there are six ppm images do not match targeted jpg images (mismatching). As we can see from their one-dimensional histogram, their original ppm one-dimensional histograms have various texture distributions comparing with the second jpg one-dimensional histograms. The peak values between these two images are different. They are mismatched to the third one-dimensional histograms.

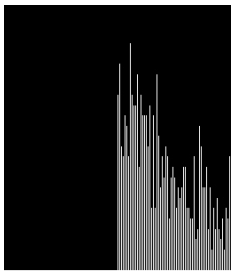
First mismatching

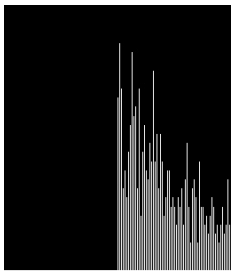

Histograms of i006.ppm

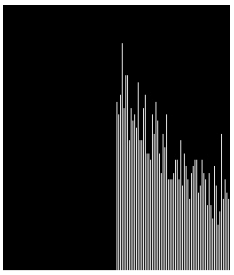

Histogram of i006.jpg


Histogram of i002.jpg

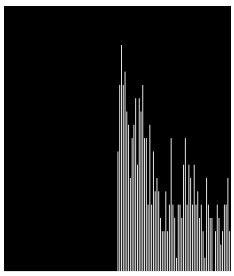
Second mismatching

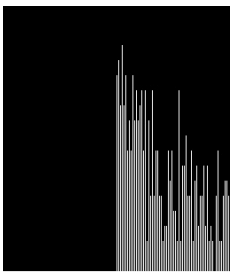

Histograms of i018.ppm

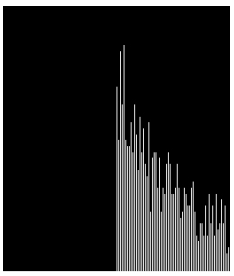

Histogram of i018.jpg


Histogram of i012.jpg

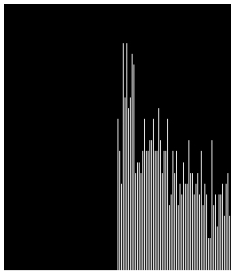
Third mismatching

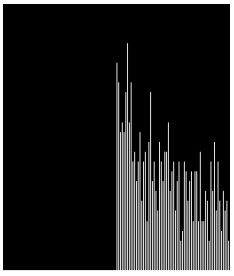

Histograms of i020.ppm

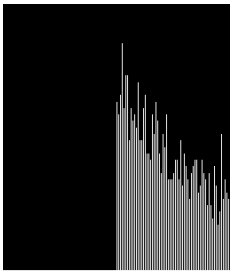

Histogram of i020.jpg


Histogram of i002.jpg

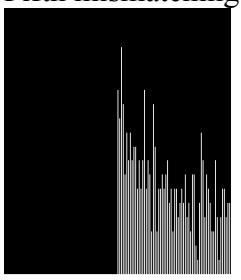
Fourth mismatching

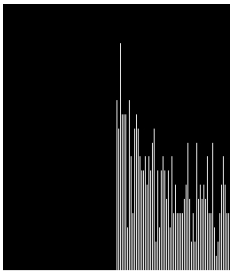

Histograms of i028.ppm

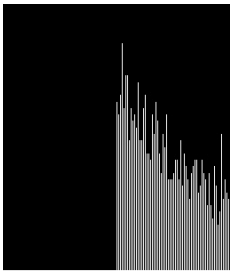

Histogram of i028.jpg


Histogram of i012.jpg

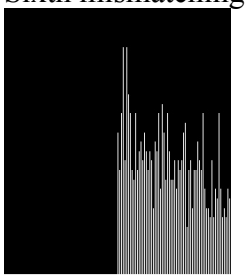
Fifth mismatching

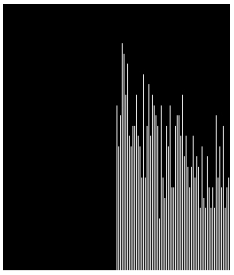

Histograms of i033.ppm

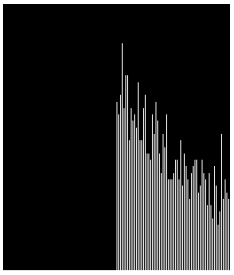

Histogram of i033.jpg


Histogram of i012.jpg

Sixth mismatching


Histograms of i040.ppm


Histogram of i040.jpg


Histogram of i012.jpg

Step 3: Combine Similarities, and Cluster

Python app: similarity_cluster.py

In this step, I first combine my measurements of color similarity (C) and measurements of texture similarity (T) by adding their codes. Using one two-dimensional for-loop, my new program computes both color histograms and texture histograms. Within the two for-loops, my new program conduct both color histogram matching and texture histogram matching. Here are my implementations:

```
for ppm_infile in glob.glob( os.path.join("images", '*.ppm') ):
    high_color_correlation = ""
    low_color_correlation = ""
    high_texture_correlation = ""
    low_texture_correlation = ""
    #Loads each ppm images from the image folder.
    ppm_img = cv2.imread(ppm_infile)
    #For drawing the histogram.
    ppm_h = np.zeros((300,256,3))
    #Convert ppm image to gray scale.
    ppm_imgray = cv2.cvtColor(ppm_img,cv2.COLOR_BGR2GRAY)

    ##Color Histogram Section
    #Create ppm binary mask, a 8-bit image, where white denotes that region should be used for histogram calculations, and black means it shou
    ppm_ret, ppm_thresh_mask = cv2.threshold(ppm_imgray,127,255,0)
    #Save 8 bit mask images
    cv2.imwrite("8bitmask/" + ppm_infile.replace("images/", "").replace(".ppm", ".jpg"), ppm_thresh_mask)
    #Number of bins, since the histogram has 256 colors, it needs 256 bins. Bin is a multidimensional array.
    ppm_bins = np.arange(256).reshape(256,1)
    color = [ (255,0,0),(0,255,0),(0,0,255) ]
    #Iterate an enumerate tuple containing a count and the values obtained from iterating over the list color.
    for ch, col in enumerate(color):
        #Calculates a histogram for the 8-bit region marked in the mask using a set of arrays.
        ppm_hist = cv2.calcHist([ppm_img],[ch],ppm_thresh_mask,[256],[0,256])
        #Normalize the value to fall below 255 and fit in image height.
        cv2.normalize(ppm_hist,ppm_hist,0,255,cv2.NORM_MINMAX)
        #Evenly round to the given number of decimals. For values exactly halfway between rounded decimal values, Numpy rounds to the nearest
        round_ppm_hist = np.int32(np.around(ppm_hist))
        #Stack bins and hist for drawing the polylines.
        ppm_pts = np.column_stack((ppm_bins,round_ppm_hist))
        #Draw polylines to represent the histogram.
        cv2.polylines(ppm_h,[ppm_pts],False,col)
    #Flip the image vertically.
    ppm_h=np.flipud(ppm_h)
    #Save color histograms
    cv2.imwrite("histograms/ppm/" + ppm_infile.replace("images/", "").replace(".ppm", ".jpg"), ppm_h)

    ##Black and White Texture Histogram Section
    #For drawing the histogram.
    ppm_h = np.zeros((300,256,3))
    #Calculates the Laplacian of an image. The function calculates the Laplacian of the source image by adding up the second x and y derivativ
    ppm_gray_lap = cv2.Laplacian(ppm_imgray,cv2.CV_16S,ksize=3,scale=1,delta=0)
    #Converts input array elements to another 8-bit unsigned integer with optional linear transformation.
    ppm_dst = cv2.convertScaleAbs(ppm_gray_lap)
    #Save paplacian ppm images.
    cv2.imwrite("laplacian/ppm/" + ppm_infile.replace("images/", "").replace(".ppm", ".jpg"), ppm_dst)
    #Create ppm binary mask, a 8-bit image, where white denotes that region should be used for histogram calculations, and black means it shou
    ppm_ret, ppm_lap_thresh_mask = cv2.threshold(ppm_dst,127,255,0)
    #Save 8 bit mask images.
    cv2.imwrite("8bitmask/ppm/" + ppm_infile.replace("images/", "").replace(".ppm", ".jpg"), ppm_lap_thresh_mask)
    #Calculates a histogram for the 8-bit region marked in the mask using a set of arrays.
    ppm_texture_hist = cv2.calcHist([ppm_dst],[0],ppm_lap_thresh_mask,[256],[0,255])
    #Normalize the value to fall below 255 and fit in image height.
    cv2.normalize(ppm_texture_hist,ppm_texture_hist,0,255,cv2.NORM_MINMAX)
    #Evenly round to the given number of decimals. For values exactly halfway between rounded decimal values, Numpy rounds to the nearest even
    round_ppm_texture_hist = np.int32(np.around(ppm_texture_hist))
```

```
    for x,fliped_ppm_texture_hist in enumerate(round_ppm_texture_hist):
        cv2.line(ppm_h,(x,0),(x,fliped_ppm_texture_hist),(255,255,255))
    fliped_ppm_texture_hist = np.flipud(ppm_h)
    #Save texture histograms
    cv2.imwrite("one_dimensional_histograms/ppm/" + ppm_infile.replace("images/", "").replace(".ppm", ".jpg"), fliped_ppm_texture_hist)

    for jpg_infile in glob.glob( os.path.join("images", '*.jpg') ):
        #Loads each jpg images from the image folder.
        jpg_img = cv2.imread(jpg_infile)
        #For drawing the histogram.
        jpg_h = np.zeros((300,256,3))
        #Convert jpg image to gray scale.
        jpg_imgray = cv2.cvtColor(jpg_img,cv2.COLOR_BGR2GRAY)

        ##Color Histogram Section
        #Create jpg binary mask, a 8-bit image, where white denotes that region should be used for histogram calculations, and black means it
        jpg_ret, jpg_thresh_mask = cv2.threshold(jpg_imgray,127,255,0)
        #Return number of bins (2-D array). Return 256 bins since we have 256 colors.
        jpg_bins = np.arange(256).reshape(256,1)
        color = [ (255,0,0),(0,255,0),(0,0,255) ]
        #Iterate an enumerate tuple containing a count and the values obtained from iterating over the list color.
        for ch, col in enumerate(color):
            #Calculates a histogram for the 8-bit region marked in the mask using a set of arrays.
            jpg_hist = cv2.calcHist([jpg_img],[ch],jpg_thresh_mask,[256],[0,256])
            #Normalize the value to fall below 255 and fit in image height.
            cv2.normalize(jpg_hist,jpg_hist,0,255,cv2.NORM_MINMAX)
            #Evenly round to the given number of decimals. For values exactly halfway between rounded decimal values, Numpy rounds to the nea
            round_jpg_hist = np.int32(np.around(jpg_hist))
            #Stack bins and hist for drawing the polylines.
            jpg_pts = np.column_stack((jpg_bins,round_jpg_hist))
            #Draw polylines to represent the histogram.
            cv2.polylines(jpg_h,[jpg_pts],False,col)
        #Flip the image vertically.
        jpg_h=np.flipud(jpg_h)
        #Save color histograms
        cv2.imwrite("histograms/jpg/" + jpg_infile.replace("images/", "").replace(".ppm", ".jpg"), jpg_h)

        ##Black and White Texture Histogram
        #For drawing the histogram.
        jpg_h = np.zeros((300,256,3))
        #Calculates the Laplacian of an image. The function calculates the Laplacian of the source image by adding up the second x and y deri
        jpg_gray_lap = cv2.Laplacian(jpg_imgray,cv2.CV_16S,ksize=3,scale=1,delta=0)
        #Converts input array elements to another 8-bit unsigned integer with optional linear transformation.
        jpg_dst = cv2.convertScaleAbs(jpg_gray_lap)
        #Save paplacian jpg images.
        cv2.imwrite("laplacian/jpg/" + jpg_infile.replace("images/", "").replace(".ppm", ".jpg"), jpg_dst)
        #Create jpg binary mask, a 8-bit image, where white denotes that region should be used for histogram calculations, and black means it
        jpg_ret, jpg_lap_thresh_mask = cv2.threshold(jpg_dst,127,255,0)
        #Save 8 bit mask images.
        cv2.imwrite("8bitmask/jpg/" + jpg_infile.replace("images/", "").replace(".ppm", ".jpg"), jpg_lap_thresh_mask)
        #Calculates a histogram for the 8-bit region marked in the mask using a set of arrays.
        jpg_texture_hist = cv2.calcHist([jpg_dst],[0],jpg_lap_thresh_mask,[256],[0,255])
        #Normalize the value to fall below 255 and fit in image height.
        cv2.normalize(jpg_texture_hist,jpg_texture_hist,0,255,cv2.NORM_MINMAX)
        #Evenly round to the given number of decimals. For values exactly halfway between rounded decimal values, Numpy rounds to the nearest
        round_jpg_texture_hist = np.int32(np.around(jpg_texture_hist))
        for x,fliped_jpg_texture_hist in enumerate(round_jpg_texture_hist):
            cv2.line(jpg_h,(x,0),(x,fliped_jpg_texture_hist),(255,255,255))
        #Flip the image vertically.
        fliped_jpg_texture_hist = np.flipud(jpg_h)
        #Save one dimensional histogram
        cv2.imwrite("one_dimensional_histograms/jpg/" + jpg_infile.replace("images/", "").replace(".ppm", ".jpg"), fliped_jpg_texture_hist)


        #Compare the correlation of two color histograms.
        color_counter = cv2.compareHist(ppm_hist, jpg_hist, cv.CV_COMP_CORREL)
        #Initialization of the result interpretation.
        if high_color_correlation == "" and low_color_correlation == "":
            high_color_counter = color_counter
            low_color_counter = color_counter
            high_color_correlation = jpg_infile
            low_color_correlation = jpg_infile
        #Find the most similar jpg image.
        if color_counter >= high_color_counter:
            high_color_counter = color_counter
            high_color_correlation = jpg_infile
        #Find the most dissimilar jpg image.
        if color_counter <= low_color_counter:
            low_color_counter = color_counter
            low_color_correlation = jpg_infile

        #Compare the correlation of two texture histograms.
        texture_counter = cv2.compareHist(ppm_texture_hist, jpg_texture_hist, cv.CV_COMP_CORREL)
        #Initialization of the result interpretation.
        if high_texture_correlation == "" and low_texture_correlation == "":
            high_texture_counter = texture_counter
            low_texture_counter = texture_counter
            high_texture_correlation = jpg_infile
            low_texture_correlation = jpg_infile
        #Find the most similar jpg image.
        if texture_counter >= high_texture_counter:
            high_texture_counter = texture_counter
            high_texture_correlation = jpg_infile
        #Find the most dissimilar jpg image.
        if texture_counter <= low_texture_counter:
            low_texture_counter = texture_counter
            low_texture_correlation = jpg_infile

    #Print the result using colored text supported by the termcolor library of terminal.
    print colored("Reading PPM Image: " + ppm_infile, 'red')
    print colored("Color Histogram Matching", 'green')
    print colored("Most Similar JPG image: " + high_color_correlation + "     Correlation Value: " + str(high_color_counter), 'blue')
    print colored("Most Dissimilar JPG image: " + low_color_correlation + "  Correlation Value: " + str(low_color_counter), 'blue')
    print colored("Texture Histogram Matching", 'green')
    print colored("Most Similar JPG image: " + high_texture_correlation + "     Correlation Value: " + str(high_texture_counter), 'blue')
    print colored("Most Dissimilar JPG image: " + low_texture_correlation + "  Correlation Value: " + str(low_texture_counter), 'blue')
    print ""
```

Figure 14: Implementation for combining color histogram matching and texture matching

The printed results are:

```
dyn-160-39-29-75:assignment2 puconghan$ python similarity_cluster.py
Reading PPM Image: images/i01.ppm
Color Histogram Matching
Most Similar JPG image: images/i01.jpg     Correlation Value: 0.927718229972
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.0752414210177
Texture Histogram Matching
Most Similar JPG image: images/i01.jpg     Correlation Value: 0.96895864939
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.645993159483

Reading PPM Image: images/i02.ppm
Color Histogram Matching
Most Similar JPG image: images/i02.jpg     Correlation Value: 0.992992135948
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.200769161408
Texture Histogram Matching
Most Similar JPG image: images/i02.jpg     Correlation Value: 0.973556303426
Most Dissimilar JPG image: images/i22.jpg  Correlation Value: 0.779256107336

Reading PPM Image: images/i03.ppm
Color Histogram Matching
Most Similar JPG image: images/i03.jpg     Correlation Value: 0.961881816478
Most Dissimilar JPG image: images/i35.jpg  Correlation Value: 0.036659047028
Texture Histogram Matching
Most Similar JPG image: images/i03.jpg     Correlation Value: 0.952303565933
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.659735628474

Reading PPM Image: images/i04.ppm
Color Histogram Matching
Most Similar JPG image: images/i04.jpg     Correlation Value: 0.975827691357
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: -0.158224417865
Texture Histogram Matching
Most Similar JPG image: images/i04.jpg     Correlation Value: 0.962552808163
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.717015079341

Reading PPM Image: images/i05.ppm
Color Histogram Matching
Most Similar JPG image: images/i05.jpg     Correlation Value: 0.989066521849
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.0229586426177
Texture Histogram Matching
Most Similar JPG image: images/i05.jpg     Correlation Value: 0.966845381494
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.682675129452

Reading PPM Image: images/i06.ppm
Color Histogram Matching
Most Similar JPG image: images/i23.jpg     Correlation Value: 0.995593444527
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.100018966014
Texture Histogram Matching
Most Similar JPG image: images/i02.jpg     Correlation Value: 0.93648523309
Most Dissimilar JPG image: images/i22.jpg  Correlation Value: 0.766724306192

Reading PPM Image: images/i07.ppm
Color Histogram Matching
Most Similar JPG image: images/i07.jpg     Correlation Value: 0.993860999372
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.143095875381
Texture Histogram Matching
Most Similar JPG image: images/i07.jpg     Correlation Value: 0.960729379584
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.698725953948

Reading PPM Image: images/i08.ppm
Color Histogram Matching
Most Similar JPG image: images/i08.jpg     Correlation Value: 0.928374656879
Most Dissimilar JPG image: images/i37.jpg  Correlation Value: 0.0471230966661
Texture Histogram Matching
Most Similar JPG image: images/i08.jpg     Correlation Value: 0.964738217848
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.641376097916

Reading PPM Image: images/i09.ppm
Color Histogram Matching
Most Similar JPG image: images/i09.jpg     Correlation Value: 0.992888461666
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.134897343156
Texture Histogram Matching
Most Similar JPG image: images/i09.jpg     Correlation Value: 0.967919980611
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.695672799328

Reading PPM Image: images/i10.ppm
Color Histogram Matching
Most Similar JPG image: images/i10.jpg     Correlation Value: 0.971193492605
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.13584409714
Texture Histogram Matching
Most Similar JPG image: images/i10.jpg     Correlation Value: 0.989992301239
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.689937580411

Reading PPM Image: images/i11.ppm
Color Histogram Matching
Most Similar JPG image: images/i11.jpg     Correlation Value: 0.981924840683
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.0893359312828
Texture Histogram Matching
Most Similar JPG image: images/i11.jpg     Correlation Value: 0.976967412017
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.656968263741

Reading PPM Image: images/i12.ppm
Color Histogram Matching
Most Similar JPG image: images/i12.jpg     Correlation Value: 0.97519724949
Most Dissimilar JPG image: images/i37.jpg  Correlation Value: -0.0907931330532
Texture Histogram Matching
```

```
Most Similar JPG image: images/i12.jpg     Correlation Value: 0.987923624684
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.762071126158

Reading PPM Image: images/i13.ppm
Color Histogram Matching
Most Similar JPG image: images/i13.jpg     Correlation Value: 0.985231789442
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.00174290600438
Texture Histogram Matching
Most Similar JPG image: images/i13.jpg     Correlation Value: 0.997807287499
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.702450255875

Reading PPM Image: images/i14.ppm
Color Histogram Matching
Most Similar JPG image: images/i14.jpg     Correlation Value: 0.984970953954
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.205357544478
Texture Histogram Matching
Most Similar JPG image: images/i14.jpg     Correlation Value: 0.992983891296
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.756651212029

Reading PPM Image: images/i15.ppm
Color Histogram Matching
Most Similar JPG image: images/i15.jpg     Correlation Value: 0.996030398848
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.218545737133
Texture Histogram Matching
Most Similar JPG image: images/i15.jpg     Correlation Value: 0.970260520432
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.752255550598

Reading PPM Image: images/i16.ppm
Color Histogram Matching
Most Similar JPG image: images/i16.jpg     Correlation Value: 0.950659446779
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.132146075142
Texture Histogram Matching
Most Similar JPG image: images/i16.jpg     Correlation Value: 0.988197821801
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.739358178838

Reading PPM Image: images/i17.ppm
Color Histogram Matching
Most Similar JPG image: images/i17.jpg     Correlation Value: 0.998711544673
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.126863365324
Texture Histogram Matching
Most Similar JPG image: images/i17.jpg     Correlation Value: 0.957166627944
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.706584638854

Reading PPM Image: images/i18.ppm
Color Histogram Matching
Most Similar JPG image: images/i18.jpg     Correlation Value: 0.998367256798
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.136241875653
Texture Histogram Matching
Most Similar JPG image: images/i12.jpg     Correlation Value: 0.921861902623
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.729824257212

Reading PPM Image: images/i19.ppm
Color Histogram Matching
Most Similar JPG image: images/i19.jpg     Correlation Value: 0.982155576387
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: -0.082803579592
Texture Histogram Matching
Most Similar JPG image: images/i19.jpg     Correlation Value: 0.983082823699
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.669872061411

Reading PPM Image: images/i20.ppm
Color Histogram Matching
Most Similar JPG image: images/i20.jpg     Correlation Value: 0.999205571603
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.122092396766
Texture Histogram Matching
Most Similar JPG image: images/i02.jpg     Correlation Value: 0.933427147069
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.769560586686

Reading PPM Image: images/i21.ppm
Color Histogram Matching
Most Similar JPG image: images/i21.jpg     Correlation Value: 0.98006429026
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.338727611779
Texture Histogram Matching
Most Similar JPG image: images/i21.jpg     Correlation Value: 0.966799981046
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.698853541223

Reading PPM Image: images/i22.ppm
Color Histogram Matching
Most Similar JPG image: images/i22.jpg     Correlation Value: 0.989956928614
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.346059817043
Texture Histogram Matching
Most Similar JPG image: images/i22.jpg     Correlation Value: 0.93899380246
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.619701130583

Reading PPM Image: images/i23.ppm
Color Histogram Matching
Most Similar JPG image: images/i23.jpg     Correlation Value: 0.997613409889
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.103378791483
Texture Histogram Matching
Most Similar JPG image: images/i23.jpg     Correlation Value: 0.945861957487
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.720419927801

Reading PPM Image: images/i24.ppm
Color Histogram Matching
Most Similar JPG image: images/i24.jpg     Correlation Value: 0.970874602253
```

```
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: 0.0414107922321
Texture Histogram Matching
Most Similar JPG image: images/i24.jpg     Correlation Value: 0.968817916893
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.712223958897

Reading PPM Image: images/i25.ppm
Color Histogram Matching
Most Similar JPG image: images/i25.jpg     Correlation Value: 0.993006516899
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.240076221665
Texture Histogram Matching
Most Similar JPG image: images/i25.jpg     Correlation Value: 0.957527308361
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.70606561619

Reading PPM Image: images/i26.ppm
Color Histogram Matching
Most Similar JPG image: images/i26.jpg     Correlation Value: 0.995359916285
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.282653281866
Texture Histogram Matching
Most Similar JPG image: images/i26.jpg     Correlation Value: 0.912225033471
Most Dissimilar JPG image: images/i22.jpg  Correlation Value: 0.617480553714

Reading PPM Image: images/i27.ppm
Color Histogram Matching
Most Similar JPG image: images/i27.jpg     Correlation Value: 0.68580400586
Most Dissimilar JPG image: images/i22.jpg  Correlation Value: -0.333438623683
Texture Histogram Matching
Most Similar JPG image: images/i27.jpg     Correlation Value: 0.9116927309
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.59497627626

Reading PPM Image: images/i28.ppm
Color Histogram Matching
Most Similar JPG image: images/i28.jpg     Correlation Value: 0.992040557736
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.157629995778
Texture Histogram Matching
Most Similar JPG image: images/i12.jpg     Correlation Value: 0.952912510734
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.759541581223

Reading PPM Image: images/i29.ppm
Color Histogram Matching
Most Similar JPG image: images/i29.jpg     Correlation Value: 0.994074956087
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.226865032673
Texture Histogram Matching
Most Similar JPG image: images/i29.jpg     Correlation Value: 0.9449019088
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.701292083722

Reading PPM Image: images/i30.ppm
Color Histogram Matching
Most Similar JPG image: images/i30.jpg     Correlation Value: 0.959922242765
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.047646045301
Texture Histogram Matching
Most Similar JPG image: images/i30.jpg     Correlation Value: 0.962866131408
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.751924436415

Reading PPM Image: images/i31.ppm
Color Histogram Matching
Most Similar JPG image: images/i31.jpg     Correlation Value: 0.901602029107
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: -0.0996940859584
Texture Histogram Matching
Most Similar JPG image: images/i31.jpg     Correlation Value: 0.96043232871
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.732052690697

Reading PPM Image: images/i32.ppm
Color Histogram Matching
Most Similar JPG image: images/i32.jpg     Correlation Value: 0.776938968855
Most Dissimilar JPG image: images/i35.jpg  Correlation Value: -0.0598102928213
Texture Histogram Matching
Most Similar JPG image: images/i32.jpg     Correlation Value: 0.905751844279
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.700767320888

Reading PPM Image: images/i33.ppm
Color Histogram Matching
Most Similar JPG image: images/i33.jpg     Correlation Value: 0.944665387836
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.204057001875
Texture Histogram Matching
Most Similar JPG image: images/i12.jpg     Correlation Value: 0.935556131004
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.72836674793

Reading PPM Image: images/i34.ppm
Color Histogram Matching
Most Similar JPG image: images/i34.jpg     Correlation Value: 0.996523102256
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.18844056827
Texture Histogram Matching
Most Similar JPG image: images/i34.jpg     Correlation Value: 0.955223703319
Most Dissimilar JPG image: images/i22.jpg  Correlation Value: 0.715276821477

Reading PPM Image: images/i35.ppm
Color Histogram Matching
Most Similar JPG image: images/i35.jpg     Correlation Value: 0.997786895868
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.136752266684
Texture Histogram Matching
Most Similar JPG image: images/i35.jpg     Correlation Value: 0.881974542226
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.670611767992

Reading PPM Image: images/i36.ppm
```

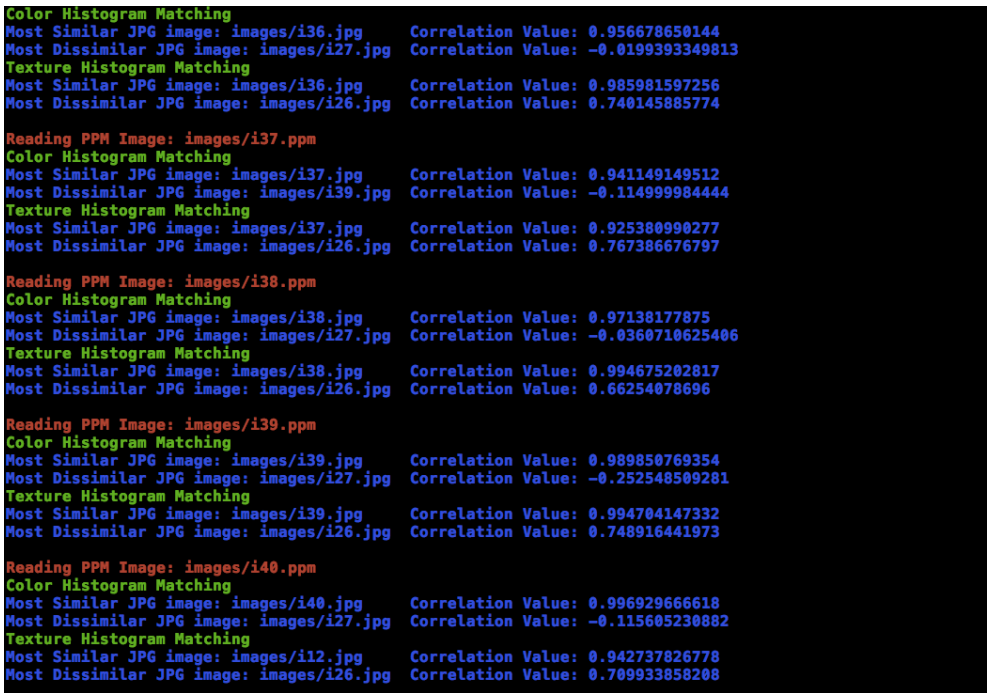


Figure 15: Results of combined measurements.

As we can see from the results, the color histogram matching produces one mismatch and the texture matching produces six mismatches. In my combined measurements, I make color histogram matching results weight more than the texture histogram matching results. Following the assignment suggestion, my implementation uses a linear sum: S = r*T + (1-r)*C, where r itself is in the interval [0,1].

In my program, I set r equals to 0.2. Before the for-loops, I declared:

```
r = 0.2
```

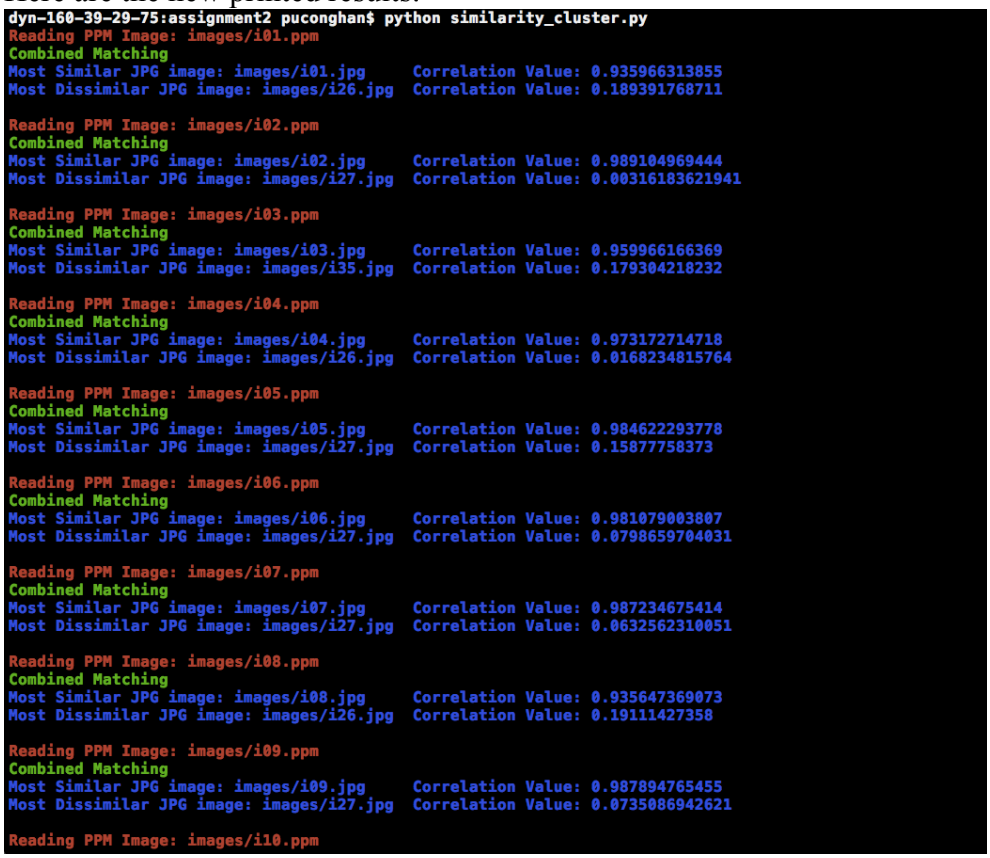
Applying the linear sum with r = 0.2 to my program, I have the following code to check for similarity:

```
if high_combined_correlation == "" and low_combined_correlation
== "":
      high_counter = texture_counter*r + color_counter*(1-r)
      low_counter = texture_counter*r + color_counter*(1-r)
      high_correlation = jpg_infile
      low_correlation = jpg_infile
if (texture_counter*r + color_counter*(1-r)) >= high_counter:
      high_counter = texture_counter*r + color_counter*(1-r)
      high_combined_correlation = jpg_infile
if (texture_counter*r + color_counter*(1-r)) <= low_counter:
      low_counter = texture_counter*r + color_counter*(1-r)
      low_combined_correlation = jpg_infile
```

The following code prints matching results:

```
print colored("Combined Matching", 'green')
print colored("Most Similar JPG image: " +
high_combined_correlation + "     Correlation Value: " +
str(high_counter), 'blue')
print colored("Most Dissimilar JPG image: " +
low_combined_correlation + "  Correlation Value: " +
str(low_counter), 'blue')
```

Here are the new printed results:



```
Combined Matching
Most Similar JPG image: images/i10.jpg     Correlation Value: 0.974953254332
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.246662793794

Reading PPM Image: images/i11.ppm
Combined Matching
Most Similar JPG image: images/i11.jpg     Correlation Value: 0.98093335495
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: 0.111067357629

Reading PPM Image: images/i12.ppm
Combined Matching
Most Similar JPG image: images/i12.jpg     Correlation Value: 0.977742524528
Most Dissimilar JPG image: images/i37.jpg  Correlation Value: 0.0996772735188

Reading PPM Image: images/i13.ppm
Combined Matching
Most Similar JPG image: images/i13.jpg     Correlation Value: 0.987746889053
Most Dissimilar JPG image: images/i37.jpg  Correlation Value: 0.170289562327

Reading PPM Image: images/i14.ppm
Combined Matching
Most Similar JPG image: images/i14.jpg     Correlation Value: 0.986573541422
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: 0.0147166244122

Reading PPM Image: images/i15.ppm
Combined Matching
Most Similar JPG image: images/i15.jpg     Correlation Value: 0.990876423164
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: 1.11390777822e-05

Reading PPM Image: images/i16.ppm
Combined Matching
Most Similar JPG image: images/i16.jpg     Correlation Value: 0.958167121783
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.253588495881

Reading PPM Image: images/i17.ppm
Combined Matching
Most Similar JPG image: images/i17.jpg     Correlation Value: 0.990402561327
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: 0.0801913822122

Reading PPM Image: images/i18.ppm
Combined Matching
Most Similar JPG image: images/i18.jpg     Correlation Value: 0.980378320343
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: 0.0549239663377

Reading PPM Image: images/i19.ppm
Combined Matching
Most Similar JPG image: images/i19.jpg     Correlation Value: 0.982341025849
Most Dissimilar JPG image: images/i26.jpg  Correlation Value: 0.0677315486086

Reading PPM Image: images/i20.ppm
Combined Matching
Most Similar JPG image: images/i20.jpg     Correlation Value: 0.984208123086
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: 0.0691807309373

Reading PPM Image: images/i21.ppm
Combined Matching
Most Similar JPG image: images/i21.jpg     Correlation Value: 0.977411428417
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.106761623653

Reading PPM Image: images/i22.ppm
Combined Matching
Most Similar JPG image: images/i22.jpg     Correlation Value: 0.979764303383
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.116216161585

Reading PPM Image: images/i23.ppm
Combined Matching
Most Similar JPG image: images/i23.jpg     Correlation Value: 0.987263119408
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: 0.0926672311812

Reading PPM Image: images/i24.ppm
Combined Matching
Most Similar JPG image: images/i24.jpg     Correlation Value: 0.970463265181
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: 0.21433489873

Reading PPM Image: images/i25.ppm
Combined Matching
Most Similar JPG image: images/i25.jpg     Correlation Value: 0.985910675191
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.0168000767787

Reading PPM Image: images/i26.ppm
Combined Matching
Most Similar JPG image: images/i26.jpg     Correlation Value: 0.978732939722
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: -0.0948680163292

Reading PPM Image: images/i27.ppm
Combined Matching
Most Similar JPG image: images/i27.jpg     Correlation Value: 0.730981750868
Most Dissimilar JPG image: images/i22.jpg  Correlation Value: -0.111807502536

Reading PPM Image: images/i28.ppm
Combined Matching
Most Similar JPG image: images/i28.jpg     Correlation Value: 0.977924602376
Most Dissimilar JPG image: images/i27.jpg  Correlation Value: 0.0488466999669
```

Figure 16: Results of combined measurements.

As we can see from Figure 16, the combined algorithm improves the accuracy of image matching. All similar images are matched correctly in the new results. The sixth image is no longer mismatched. This proves that r = 0.2 for the linear sum fit the histogram matching algorithms in my program.

Cluster Portion (Complete Clustering and Single Clustering)

My program builds clusters using the 40-ppm images. I have two implementations. One uses my own algorithm, which is not a good and accurate approach. The other uses two existing libraries, scipy.cluster.hierarchy (dendrogram, linkage) and matplotlib.pyplot (for plotting the dendrogram).

I will explain my own algorithms for clustering and drawing the dendrogram first, and explain its limitations. Before doing any clustering, my program stores the comparing result to a 2D list using the following code within the two for-loops:

```
            # Inside the inner for-loop
            sub_list.append([(texture_counter*r +
            color_counter*(1-r)), jpg_infile])
        # Inside the outside for-loop
        result_list.append([sub_list, ppm_infile])
```

The 2D for-loop contains similarity matrix. Each value in the 2D array is a list contains the value itself and the string of the image name with directory. This will simplify my implementation of comparing values and displaying directories. For analyzing the matrix and printing the dendrogram, I implement the following three functions:

```
def first_pair(n):
    closest_pair_one = ""
    closest_pair_two = ""
    counter = -1
    for sublist in n:
        for item in sublist[0]:
            if (item[0] != 1):
                if item[0] >= counter:
                    counter = item[0]
                    closest_pair_two = item[1]
                    closest_pair_one = sublist[1]
    return [counter, closest_pair_one, closest_pair_two]

def clustering():
    print colored("                    0.0 -- 0.1 -- 0.2 --
0.3 -- 0.4 -- 0.5 -- 0.6 -- 0.7 -- 0.8 -- 0.9 -- 1.0",
'red')
    firstpair = first_pair(result_list)
    counter_list.append(firstpair[0])
    current_counter = firstpair[0]
    cluster_pool.append(firstpair[1])
    cluster_pool.append(firstpair[2])
    #Print the first two item
    printitem(1 - firstpair[0], firstpair[1])
    print colored(1 - firstpair[0], 'blue')
    printitem(1 - firstpair[0], firstpair[2])
    print colored(1 - firstpair[0], 'blue')

    while len(cluster_pool) < 40:
        counter_figure = -1
        counter_name = ""
        for sublist in result_list:
            if sublist[1] in cluster_pool:
                for item in sublist[0]:
                    if (item[0] != 1):
```

```
                        if (item[0] >= counter_figure) and
item[0] < current_counter and (item[1] not in cluster_pool):
                            counter_figure = item[0]
                            counter_name = item[1]
        current_counter = counter_figure
        counter_list.append(counter_figure)
        cluster_pool.append(counter_name)
        printitem(1 - counter_figure, counter_name)
        print colored(1 - counter_figure, 'blue')
    print cluster_pool

def printitem(n, name):
    if 0.0 <= n < 0.05:
        print name + colored("      -|", 'green')
    if 0.05 <= n < 0.15:
        print name + colored("      --------|", 'green')
    if 0.15 <= n < 0.25:
        print name + colored("      -----------------|",
'green')
    if 0.25 <= n < 0.35:
        print name + colored("      ----------------------
---|", 'green')
    if 0.35 <= n < 0.45:
        print name + colored("      ----------------------
---|", 'green')
    if 0.45 <= n < 0.55:
        print name + colored("      ----------------------
------------|", 'green')
    if 0.55 <= n < 0.65:
        print name + colored("      ----------------------
--------------------|", 'green')
    if 0.65 <= n < 0.75:
        print name + colored("      ----------------------
----------------------------|", 'green')
    if 0.75 <= n < 0.85:
        print name + colored("      ----------------------
-------------------------------------|", 'green')
    if 0.85 <= n < 0.95:
        print name + colored("      ----------------------
-----------------------------------------------|",  'green')
    if n >= 0.95:
        print name + colored("      ----------------------
-----------------------------------------------------|",
'green')
```

My program calls the clustering() function after the similarity comparison process.

The first function first_pair() finds and returns the first two closest pairs in the matrix. The second function clustering() calls the function first_pair() to return the first two closest pair and stores in the cluster_pool list. The clustering() function has a while loop continually finds the next closest image and saves to the cluster_pool list. The while-loop will repeat this process 40 times, which decreases the total number of clusters by 1 at each step. The third printitem() function takes two parameters. The first parameter is the value of similarity. The second parameter is the string contains the image address. This function will print out a simi-dendrogram in command line, as shown in figure 17.

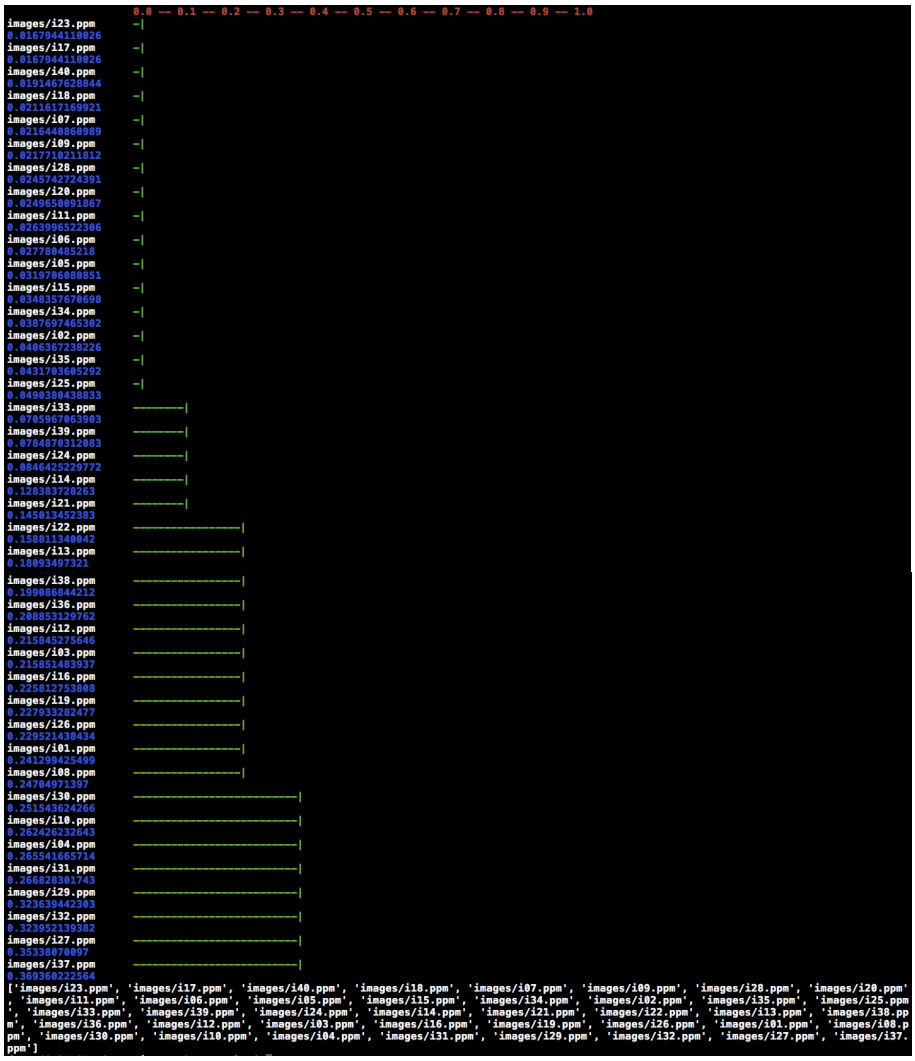


Figure 17: Printed result for the clustering (Not quite dendrogram).

The problem of this approach is that it requires programmers to change the while-loop every time. If we shift to a new list of images, developers need to count how many images and change this implementation to run through every image. This is the implementation for single-linkage clustering, calculating the shortest distance between two images. Developers also need to implement another different function (reverse the comparing process) for the complete-linkage cluster, calculating the longest distance between two images.

The printed dendrogram is not quite right. The graph is not clear and illustrative. I decided to use existing clustering libraries to analyze the matrix and existing MatLab libraries to build the dendrogram.

My program uses two existing libraries, scipy.cluster.hierarchy (dendrogram, linkage) and matplotlib.pyplot (for plotting the dendrogram) to handle matrix analyzing and dendrogram plotting. My program input these libraries using the following code:

```
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt
```

According to the documentation of scipy.cluster.hierarchy.dendrogram, the function plots the hierarchical clustering as a dendrogram. It illustrates how each cluster drawing a U-shaped link between a non-singleton cluster and its children composes. The function in my program takes three parameters, the matrix array, color threshold and label list. According to the documentation of scipy.cluster.hierarchy.linkage, the linkage() function performs hierarchical/agglomerative clustering on the condensed distance matrix. The function in my program takes two parameters, the distance_matrix (numpy array) and computational methods ("complete" or "single").

According to the documentation of matplotlib.pyplot, the library provides a MATLAB-like plotting framework.

My new implementation will stores only the similarity results to the 2D list using the following code:

```
      # Inside the inner for-loop
      sub_distance_list.append(texture_counter*r +
      color_counter*(1-r))
# Inside the outer for-loop
distance_list.append(sub_distance_list)
image_address_list.append(ppm_infile .replace("images/",
"").replace(".ppm", ""))
```

Because the dendrogram() function takes an numpy array as input, my program converts the distance_list to a numpy array using the following code:

```
distance_matrix = np.asarray(distance_list)
```

After this conversion, my program calls the linkage() function to perform hierarchical and agglomerative clustering on the distance matrix using the complete-linkage clustering method. My program generate and print the dendrogram using the dendrogram() function provided by Python matplotlab. Here is my implementation:

```
complete_link_matrix = linkage(distance_matrix, "complete")

plt.clf()
```

```
ddata = dendrogram(complete_link_matrix, color_threshold=1,
labels=image_address_list)
plt.show()
```

The printed results is:

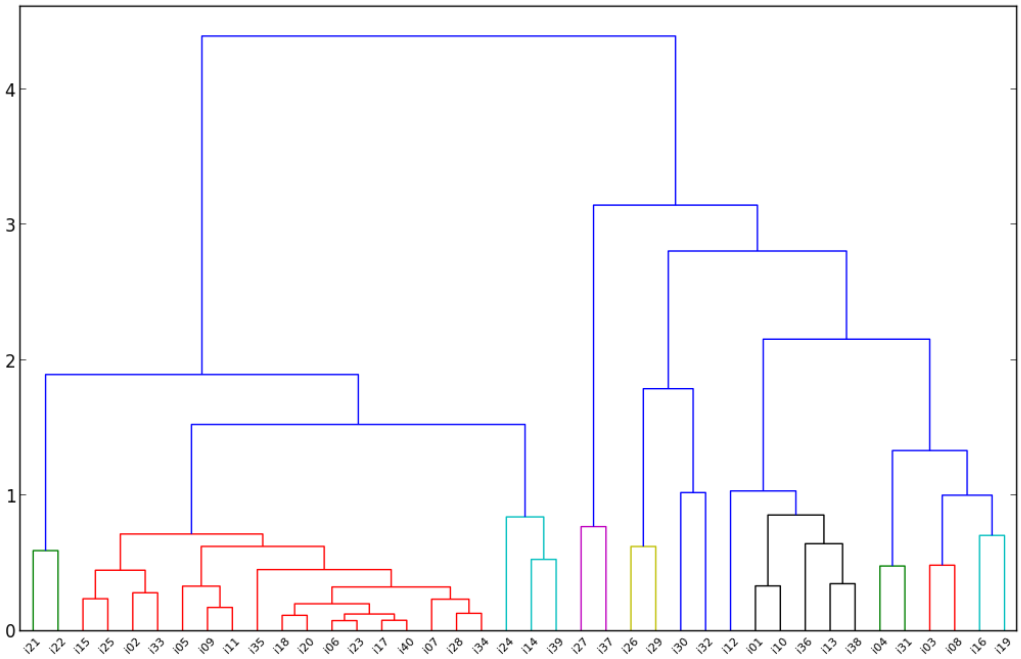


Figure 18: Complete-linkage clustering dendrogram.

To build and print a single-linkage clustering dendrogram, my program changes the method from "complete" to "single." Here is my implementation:

```
single_link_matrix = linkage(distance_matrix, "single")

plt.clf()
ddata = dendrogram(single_link_matrix, color_threshold=1,
labels=image_address_list)
plt.show()
```

The printed results is:

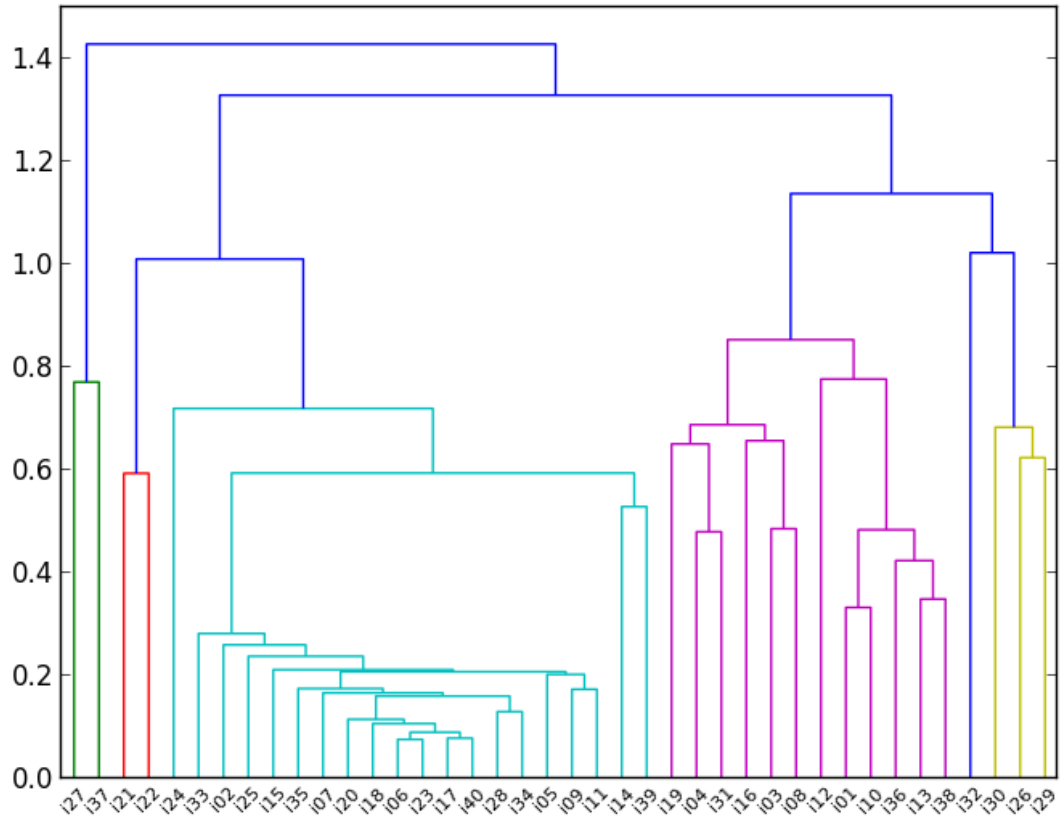


Figure 19: Single-linkage clustering dendrogram.

As we can see from Figure 19 and Figure 18, the single-linkage clustering dendrogram is different from the complete-linkage clustering dendrogram.

According to the documentation of dendrogram, the complete-linkage clustering (Farthest Point Algorithm) finds the most dissimilar pair of clusters with maximum distance. The single-linkage clustering (Nearest Point Algorithm) finds the most similar pair of clusters with minimum distance.

Both the complete-linkage clustering dendrogram and the single-linkage clustering dendrogram groups two pairs, including i06 with i23 and i17 with i40, at the first round:


i06


i23


i17


i40

After the first two iterations, these two algorithms group images differently. Compared with the single-linkage clustering dendrogram, the complete-linkage clustering

dendrogram constructs more pair clusters. As the assignment explains, at each step, complete-linkage algorithm gives clusters that are cliques. The assignment gives a good illustration: In every cluster, every image is related to every other image whereas single link at each step approximates the stages of growth of a minimum spanning tree.

The single-linkage cluster sometimes is not accurate. For instance, as shown in Figure 19, the algorithm groups image 13 and image 38 to one cluster.


i13


i38

These two images are not that similar.

Step 4: Creative step
Python apps: creativity.py

I take the second approach to approximate the overall spatial distance reflect both color similarity and textural similarity of images in two dimensions. I pick the reference image number 6, and place it at (0,0), and then place every other image horizontally away from it, proportionately based on its color dissimilarity, and vertically away from it, proportionately based on its texture dissimilarity.

The reason for me to pick image number 6 is it is the one with extreme color and texture spatial distance with other images. If we pick an image between these images with extreme color and texture characteristics, the resulted graph will have a lot of clustered images with ambiguous spatial relationship. This is because images from both extreme sides have average distance to the one in the middle of the scale.

Image number 6 (i06.ppm) and image number 23 (i23.ppm) are the first two pair of image been clustered in the histogram which indicated that they are two of the image with distinct characteristics (extreme red color or extreme textures). Another two options are the last two clusters images, such as image number 27 (i27.ppm) and image number 37 (i37.ppm). These two images have extreme green color and extreme textures.


i06


i23


i27


i37

In this step, I picked the first image i06, which contains extreme red colors and extreme textures. The rest 39 images will not be clustered with this image and will have a unambiguous spatial distance.

My new application creativity.py is built upon the previous step. Before plotting images in a graph, my program reorganizes comparison results in four lists, including color distance list, texture distance list, address list and combined distance list:

```
      sub_color_distance_list.append(color_counter)
      sub_texture_distance_list.append(texture_counter)
      sub_address_list.append(jpg_infile)
      sub_combined_list.append(color_counter*(1-r) +
      texture_counter*r)
color_distance_list.append(sub_color_distance_list)
texture_distance_list.append(sub_texture_distance_list)
address_list.append(sub_address_list)
combined_distance_list.append(sub_combined_list)
```

The first four lines of code are within the inner for-loop of my program. The rest of four lines are within the outer for-loop of my program. The color distance list stores the colored histogram similarity comparison results. The texture distance list stores the texture histogram similarity comparison results. The address list simply store image directories. The combined distance list stores the similarity comparison results combined both the color similarity distance and texture similarity distance with r = 0.2 (This r yield from step 3).

After building these four lists, we can plot the result using the matplotlib.pyplot library. We take advantages of the figimage() function in this step. According to the documentation of matplotlib, figimage() function adds a non-resampled image to the figure.

Here is my implementation to place image i06.ppm to the (0, 0) origin and other images horizontally away from it, proportionately based on its color dissimilarity, and vertically away from it, proportionately based on its texture dissimilarity:

```
im = Image.open("images/i06.ppm")
plt.figimage(im, 0, 0)

counter = 0
for item in address_list[05]:
      im = Image.open(item)
      plt.figimage(im, 400*color_distance_list[05][counter],
      300*texture_distance_list[05][counter])
      counter = counter + 1

plt.show()
```

The result is:

Figure 20: Results without reversing order and scaling.

As we can see from this intermediate result, we need to reverse the order of the images since value close to 1 means similar. I also need to scare these images to make the spatial distance more clear. Here is my improved implementation:

```
im = Image.open("images/i06.ppm")
plt.figimage(im, 0, 0)

counter = 0
for item in address_list[05]:
     im = Image.open(item)
     plt.figimage(im, 600*(1-
     color_distance_list[05][counter]), 1800*(1-
     texture_distance_list[05][counter]))
     counter = counter + 1

plt.show()
```

(1-color_distance_list[05][counter]) and (1-texture_distance_list[05][counter]) allows images on the graph to reverse the order. The images closer to the selected i06.ppm will be placed closer to it horizontally and vertically. In this new improved implementation, I scale the x-axis (1-color_distance_list[05][counter]) by 600 px and the y-axis (1-texture_distance_list[05][counter]) by 1800 pixel. This is because the color similarity results (between 0 and 1) have a more diverse range compared with the texture similarity results (between 0 and 0.5). Here are my new results:

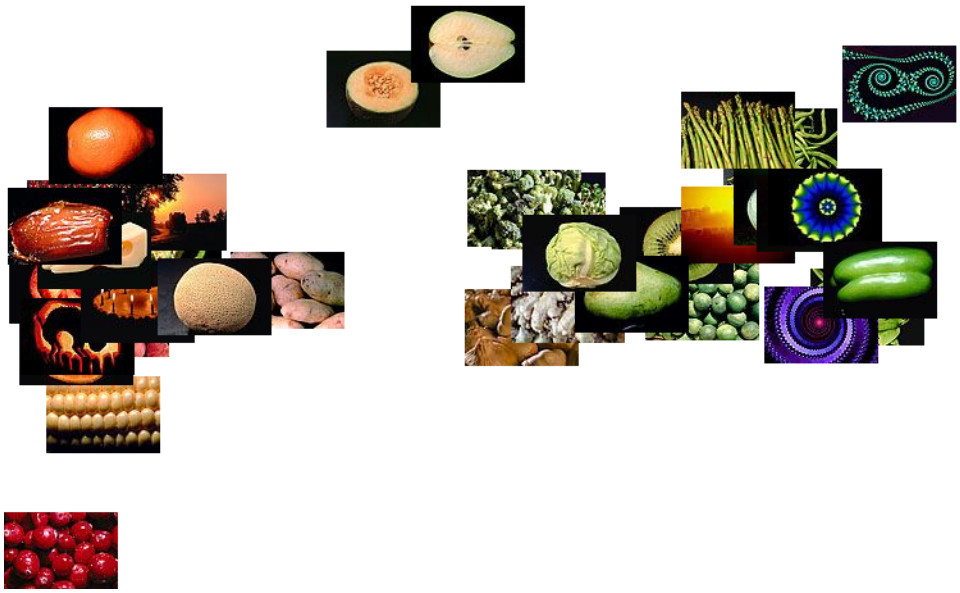


Figure 22: Results with reversing order and proper scaling.

As we can see from Figure 22, the results are relatively close to what I expected. The scaling offsets (600 for x-axis and 1800 for y-axis) make my images visually distinguishable from each other in the graph. Instead of clustering to a line as in Figure 20, images in Figure 22 have a more distributive image clusters.

Images with color close to the selected image (i06.ppm) are placed closer to the left of the graph. The selected image contains a lot of red color. Image closer to the left, it contains more red elements. Image closer to the right contains less red (contain more green in my results). This selected image make color spatial distribution clear to users.

Images with texture close to the selected image (i06.ppm) are placed closer to the bottom of the graph. The selected image contains a lot of curves. Images closer to the bottom contain more such elements. Images closer to the top contain less such elements.

Work Cited

Bigelow, Ron. "Reading Histograms-- Part I." *Article and Photography by Ron Bigelow*. N.p.. Web. 5 Mar 2013. <http://www.ronbigelow.com/articles/histograms-1/histograms-1.htm>.

"Histogram Equalization." *OpenCV Tutorial*. N.p.. Web. 22 Feb 2013. <https://code.ros.org/trac/opencv/browser/trunk/opencv/doc/tutorials/imgproc/histograms/histogram_equalization/histogram_equalization.rst?rev=6131>.

Hunter, John, Darren Dale, Eric Firing, and Michael Droettboom. "matplotlib.figure."

matplotlib. The matplot development team. Web. 3 Mar 2013. <http://matplotlib.org/api/figure_api.html>.
"Numpy and Scipy Documentation." *SciPy.org*. http://docs.scipy.org/doc/. Web. 27 Feb 2013. <http://docs.scipy.org/doc/>.
"Scipy.cluster.hierarchy.linkage." *Numpy and Scipy Documentation*. The Scipy community. Web. 27 Feb 2013. <http://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html>.
"Scipy.cluster.hierarchy.dendrogram." *Numpy and Scipy Documentation*. The Scipy community. Web. 27 Feb 2013. <http://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.dendrogram.html>.
"OpenCV Python Tutorials." OpenCV Python. Google Blogger. Web. 3 Feb 2013. <http://opencvpython.blogspot.com/2012/06/hi-this-article-is-tutorial-whichtry.html>.
"OpenCV 2.4.3 documentation." OpenCV Documentation. OpenCV Developer Team, 26 Dec 2012. Web. 3 Feb 2013. <http://docs.opencv.org/index.html>.