SciComp with Py

Image Histograms

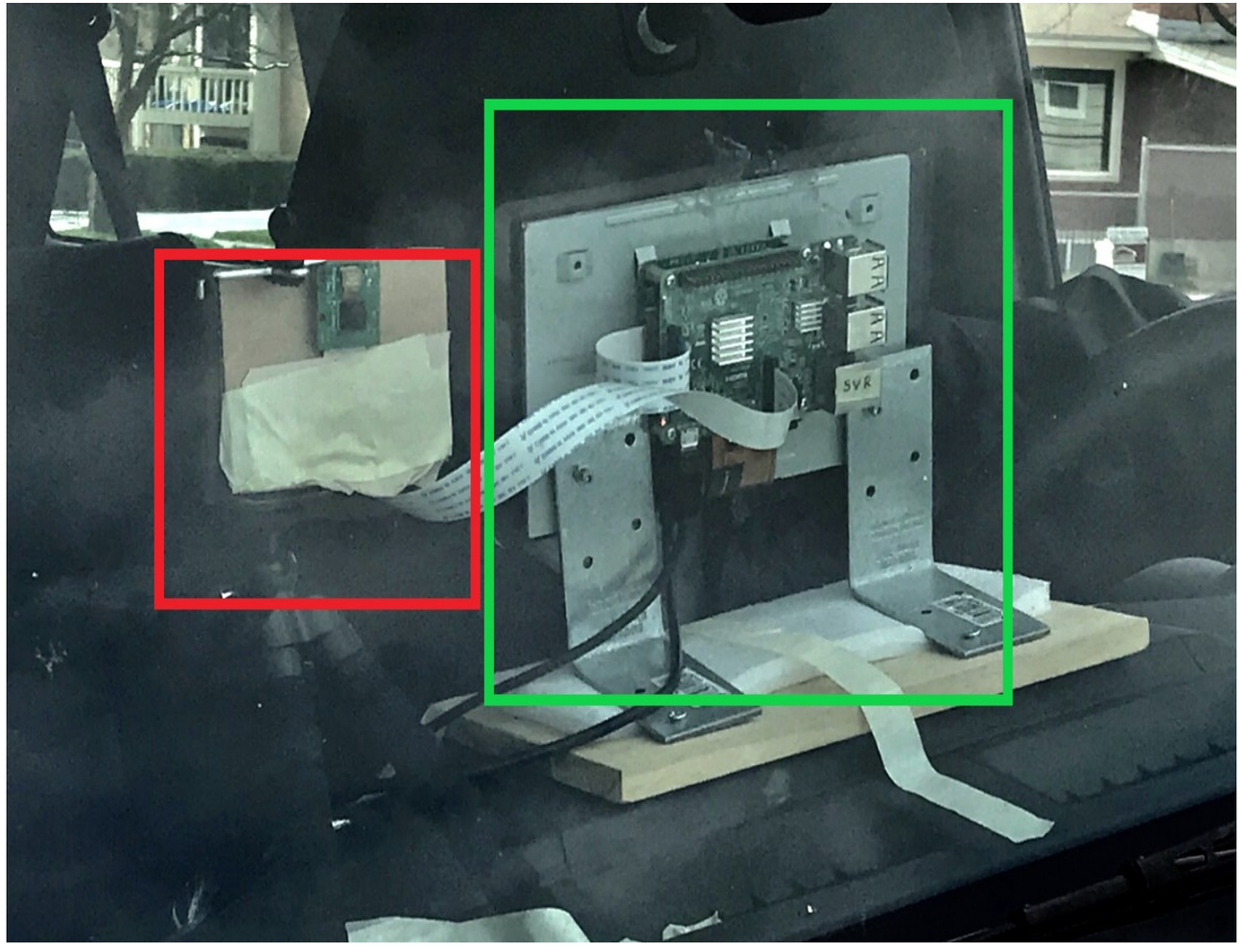
Vladimir Kulyukin
Department of Computer Science
Utah State University

Outline

- In Situ Detection of Highway Lane Boundaries: A Demo of Edge and Line Detection
- Computing & Plotting Image Histograms
- Turning Histograms into Feature Vectors
- Matching Image Histograms

In Situ Detection of Highway Lane Boundaries: A Demo of Real-Time Edge and Line Detection

Hardware Setup 1



This is a project that my graduate student Ashwani Chahal and I worked on in 2016 – 2018. The hardware consists of a camera on a tripod and a raspberry pi with a screen attached to a wooden board.

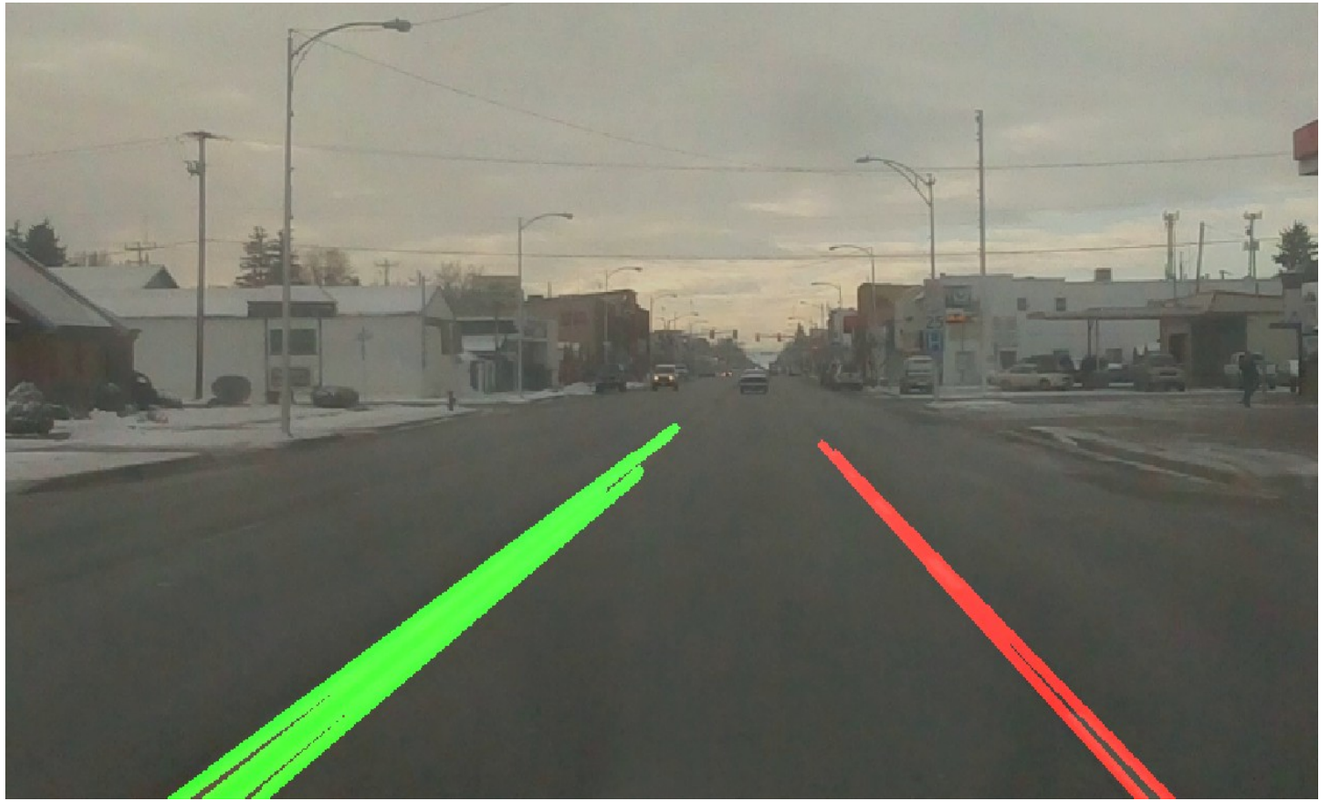
Hardware Setup 2

This is a project that my graduate student Ashwani Chahal and I worked on in 2016 – 2018. The hardware consists of a camera on a tripod and a raspberry pi with a screen attached to a wooden board.

Hardware Setup 3

This is a project that my graduate student Ashwani Chahal and I worked on in 2016 – 2018. The hardware consists of a camera on a tripod and a raspberry pi with a screen attached to a wooden board.

Reference & Live Demo



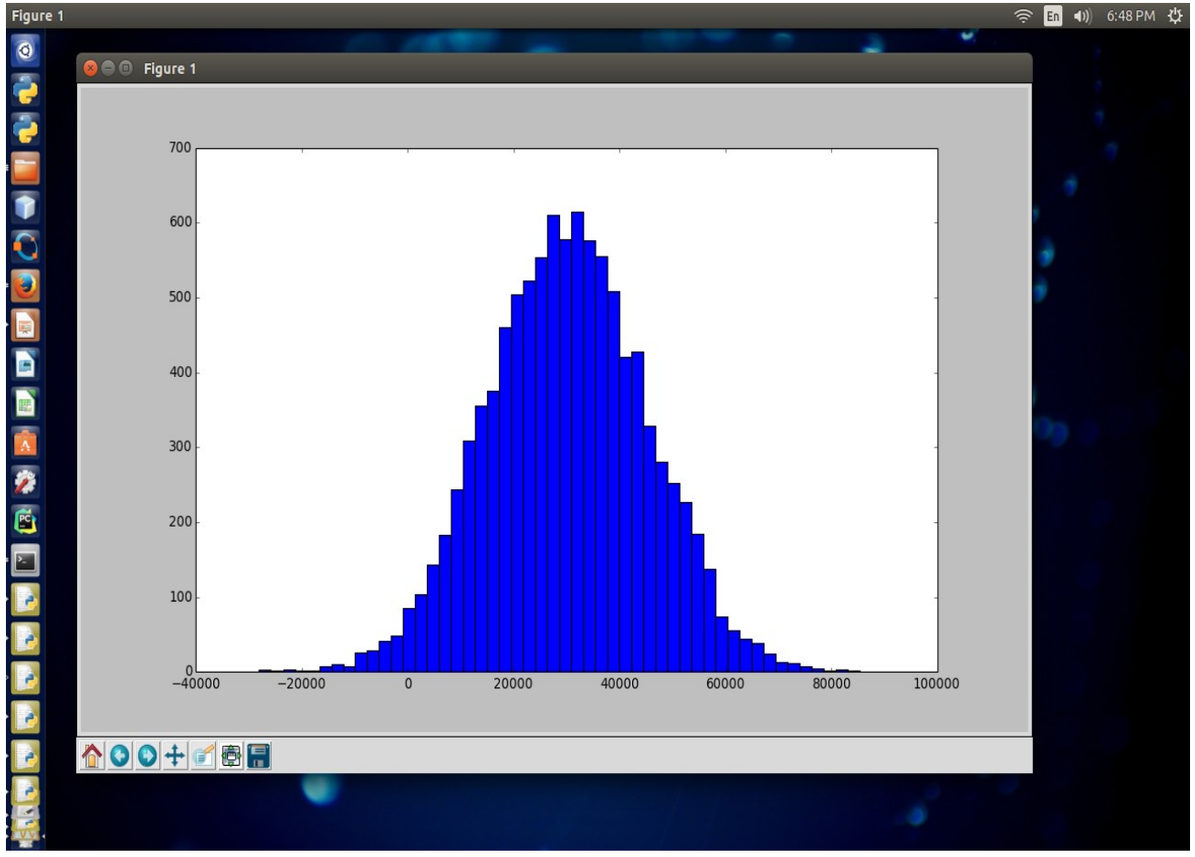
https://www.youtube.com/watch?v=1EdCja8NtUo

Kulyukin, V. and Chahal, A. “In Situ Detection of Highway Lane Boundaries: GreedyHaarSpiker vs. SlopeInterceptFilter: A Comparison of Two Lane Boundary Detection Algorithms.” Journal of Graphics, Vision, and Image Processing (GVIP), vol. 18, issue 2, pp. 1 – 11, Sept. 2018. (pdf)

Computing & Plotting Image Histograms

What's a Histogram?

A histogram is a 2D plot where the x-axis values are bins (i.e., numerical ranges) and the y-axis values are counts of some objects (e.g., salaries) falling into those ranges. The histogram below is a histogram of random salaries where the mean salary is 30,000 and the std is 15,000.




```
## use normal distribution to generate 10,000 points
## centered on 30,000 with an STD = 15,000
incomes = np.random.normal(30000, 15000, 10000)
plt.hist(incomes, 50)
plt.show()
```

What Histograms to Plot & Why

- It is possible to plot RGB, HSV, and grayscale histograms
- Plotting histograms is useful in image analysis: histograms show color properties of images in a specific domain
- Sometimes it is necessary to split images into individual channels (e.g., R, G, and B) and plot histograms of each channel
- Histograms are worth considering any time you need to retrieve images similar to a given image from a database of images or finding an object in an image

Computing Histograms in OpenCV

OpenCV function to compute histograms: **cv2.calcHist(images, channels, mask, histSize, ranges)**

- **images**: list of images to compute histograms of

- **channels**: list of indexes specifying channels; to compute a histogram of a grayscale image, the list is **[0]**;to compute a histogram for all three (blue, green, and red) channels, this list is **[0, 1, 2]**

- **mask**: mask is a uint8 image of the same shape as the original image, where pixels with a value of 0 are ignored and pixels with a value > 0 are included in the histogram computation; using masks allows one to compute a histogram for a particular region of an image; if mask is **None**, the histogram of the entire image is computed

- **histSize**: number of bins to use when computing a histogram; it is a list with one for each channel we are computing a histogram of; bin sizes do not all have to be the same; for example, if you want to use 8 bins for each channel, the list is **[8, 8, 8]**

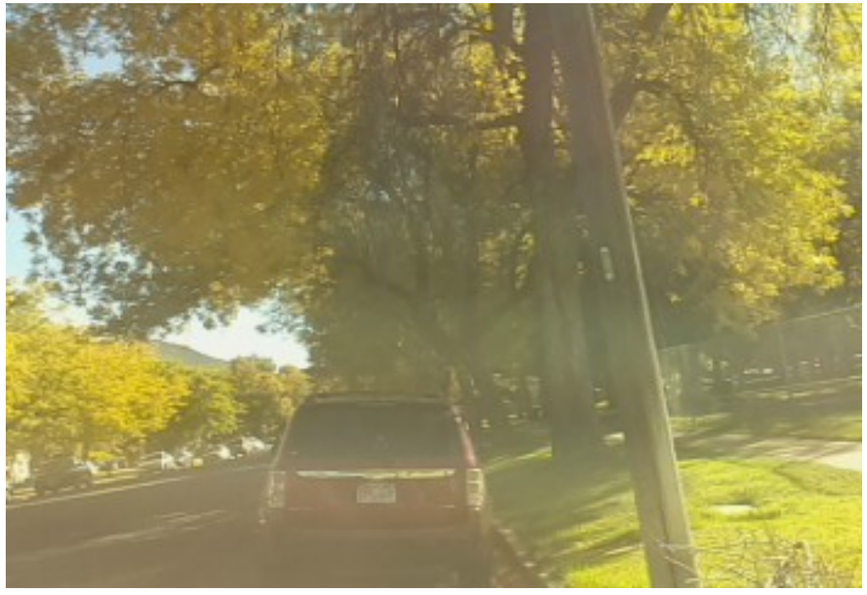
- **ranges**: range of possible pixel values; normally, this is **[0, 256]** for each channel, but if you are using a color space other than RGB (such as HSV), the ranges will be different

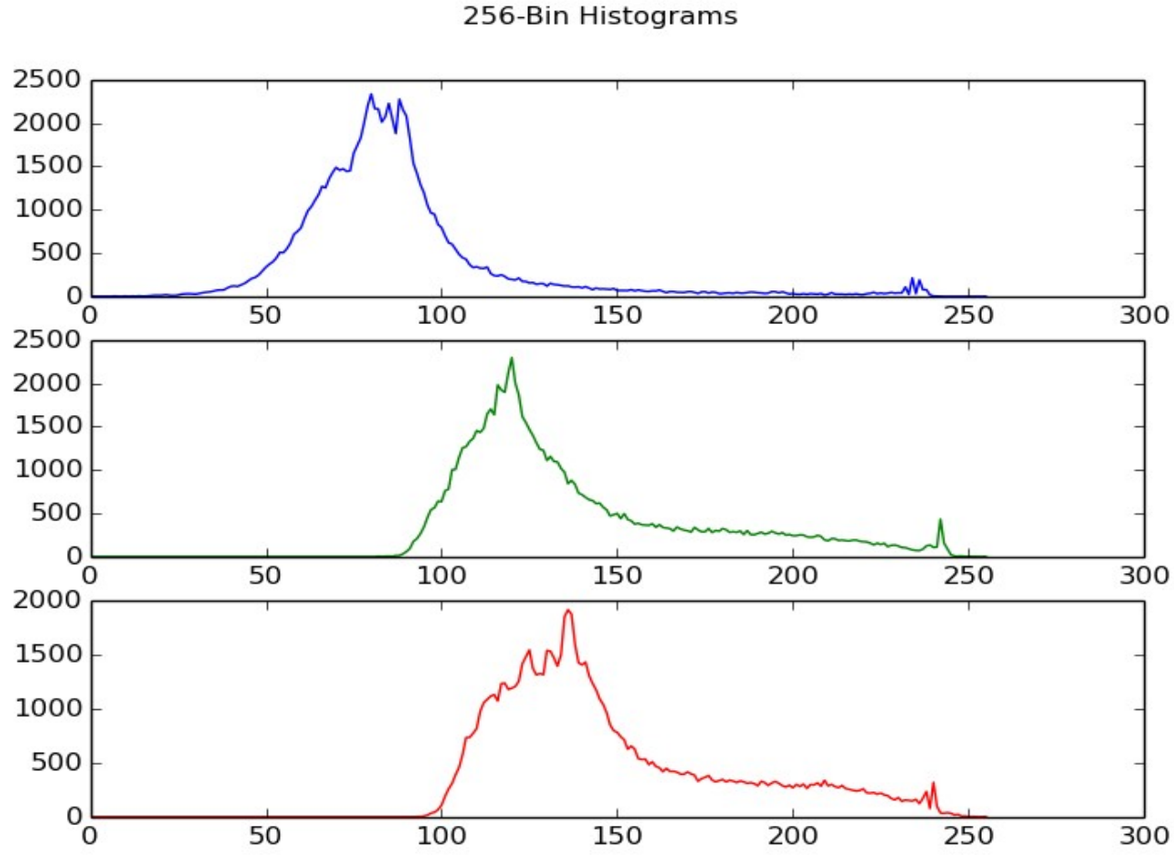
Problem

Write a program that takes an RGB image and computes and plots histograms for each color channel in the image. Histograms have either 256 bins or 32 bins. The program then converts the RGB image to grayscale and plot a 32-bin color histogram of the grayscale image.

py source in histo_01.py

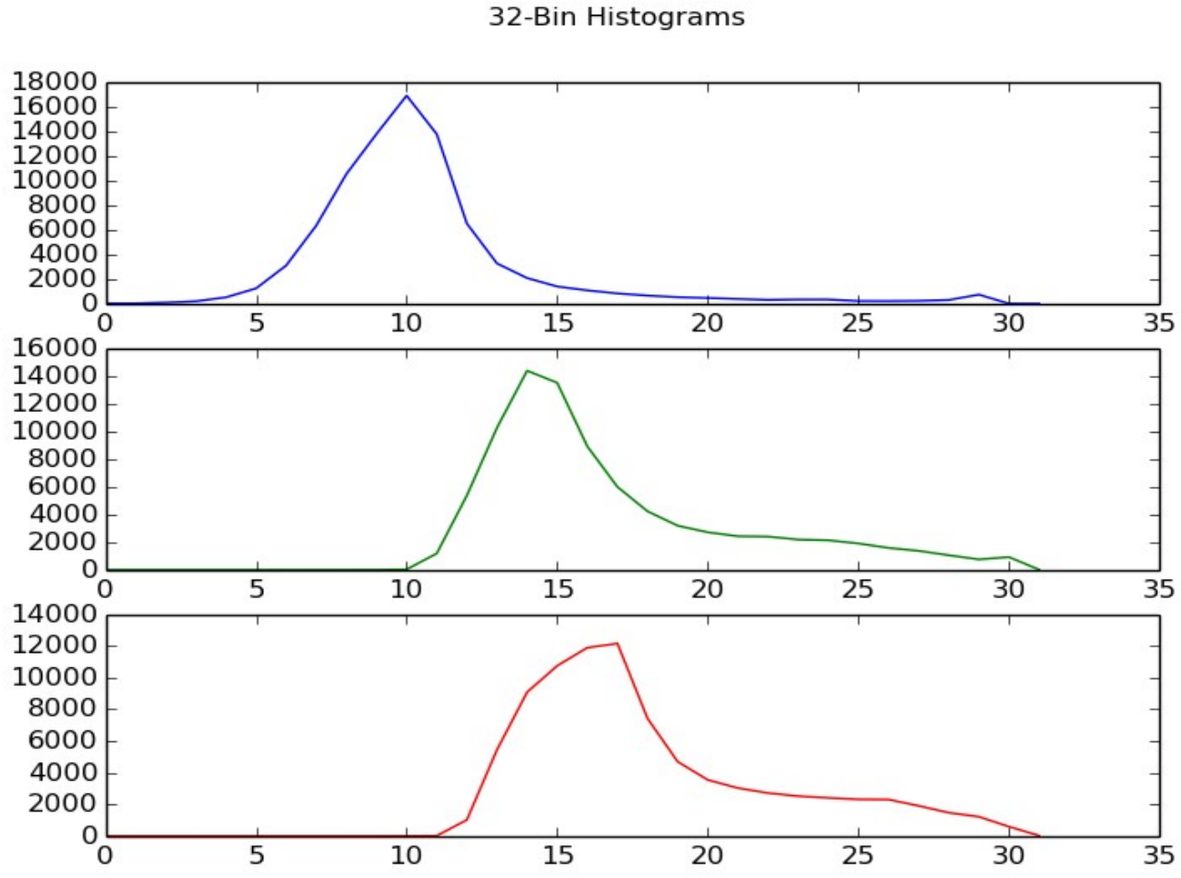
Sample Output: Three 256-Bin BGR Histograms





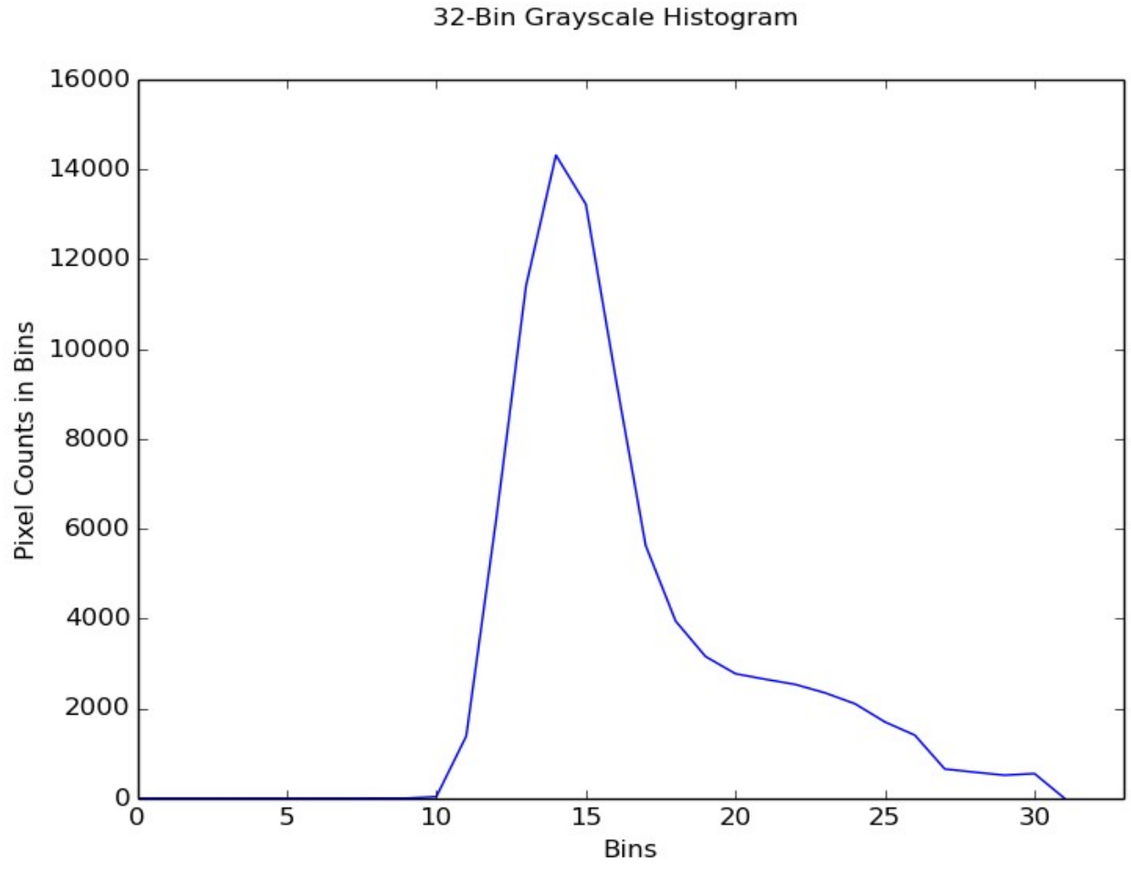
$ python histo_01.py -i car_test/img01.png

Sample Output: Three 32-Bin BGR Histograms



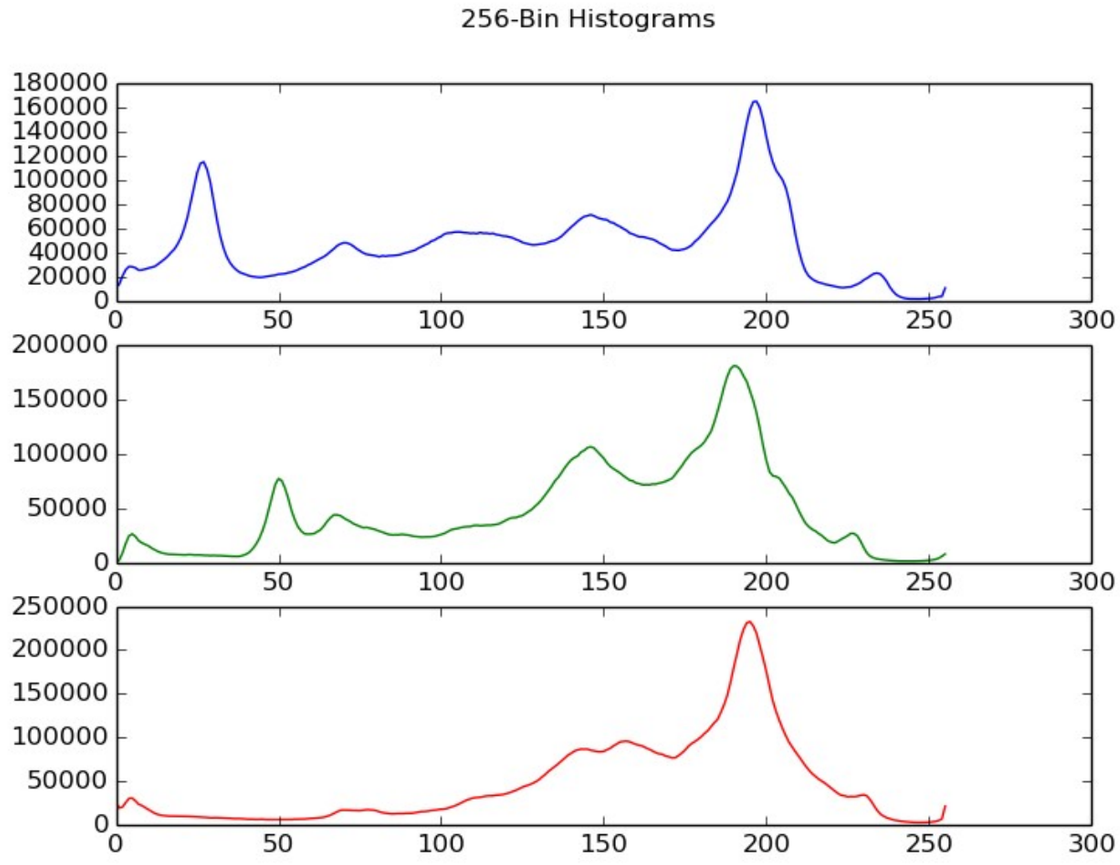
$ python histo_01.py -i car_test/img01.png

Sample Output: One 32-Bin Grayscale Histogram



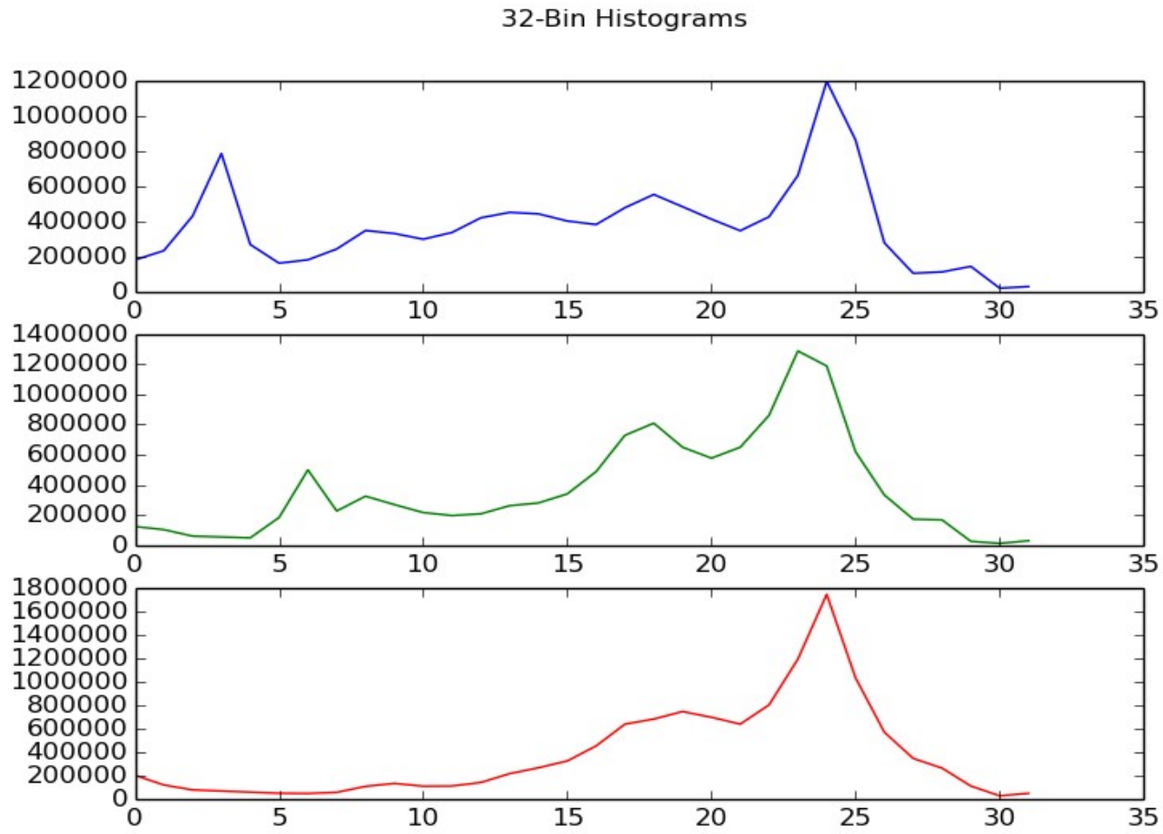
$ python histo_01.py -i car_test/img01.png

Sample Output: Three 256-Bin BGR Histograms



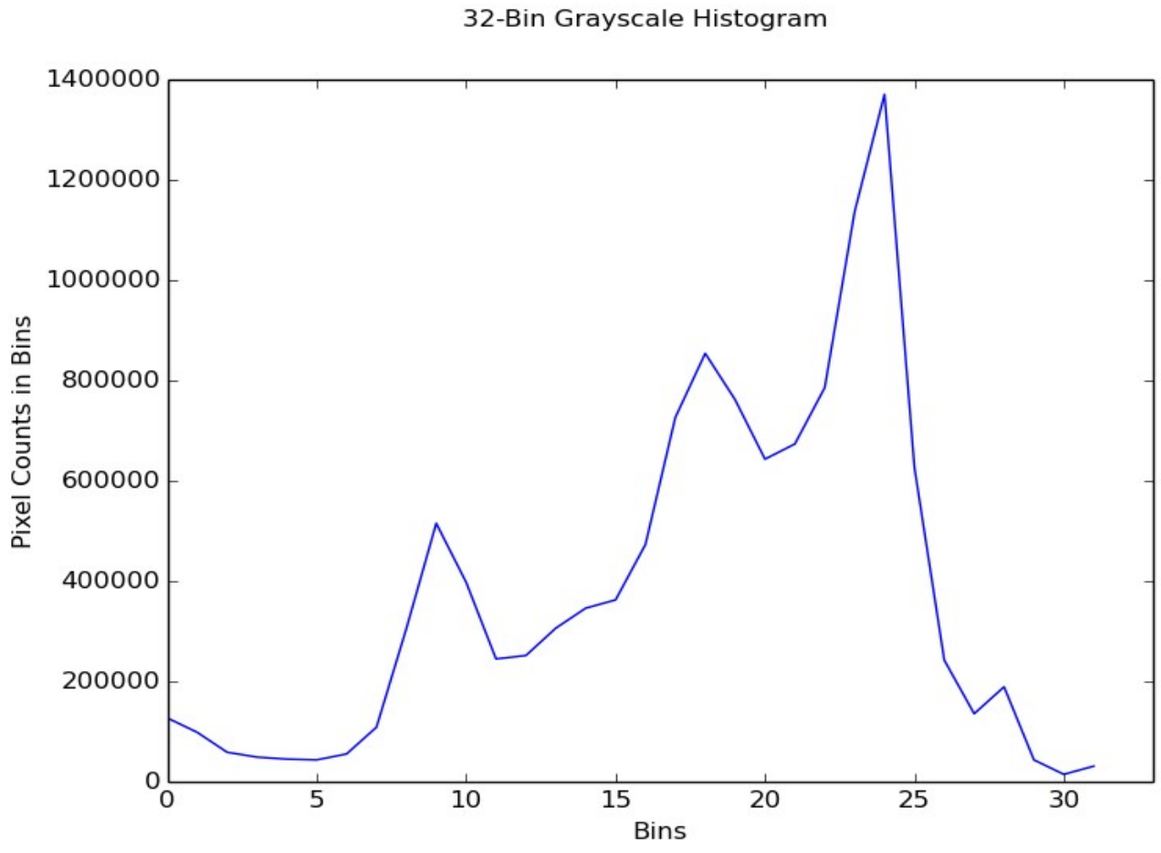
$ python histo_01.py -i img/123459076.JPG

Sample Output: Three 32-Bin BGR Histograms



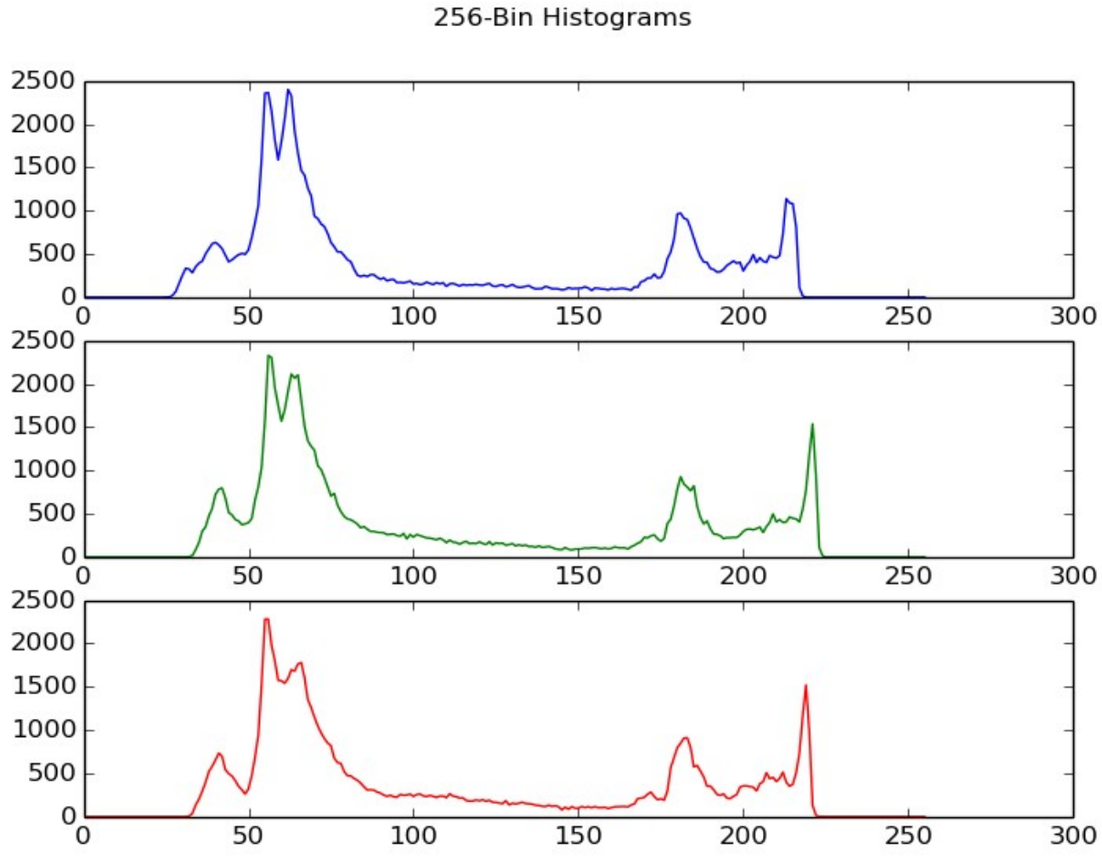
$ python histo_01.py -i img/123459076.JPG

Sample Output: Three 32-Bin Grayscale Histograms



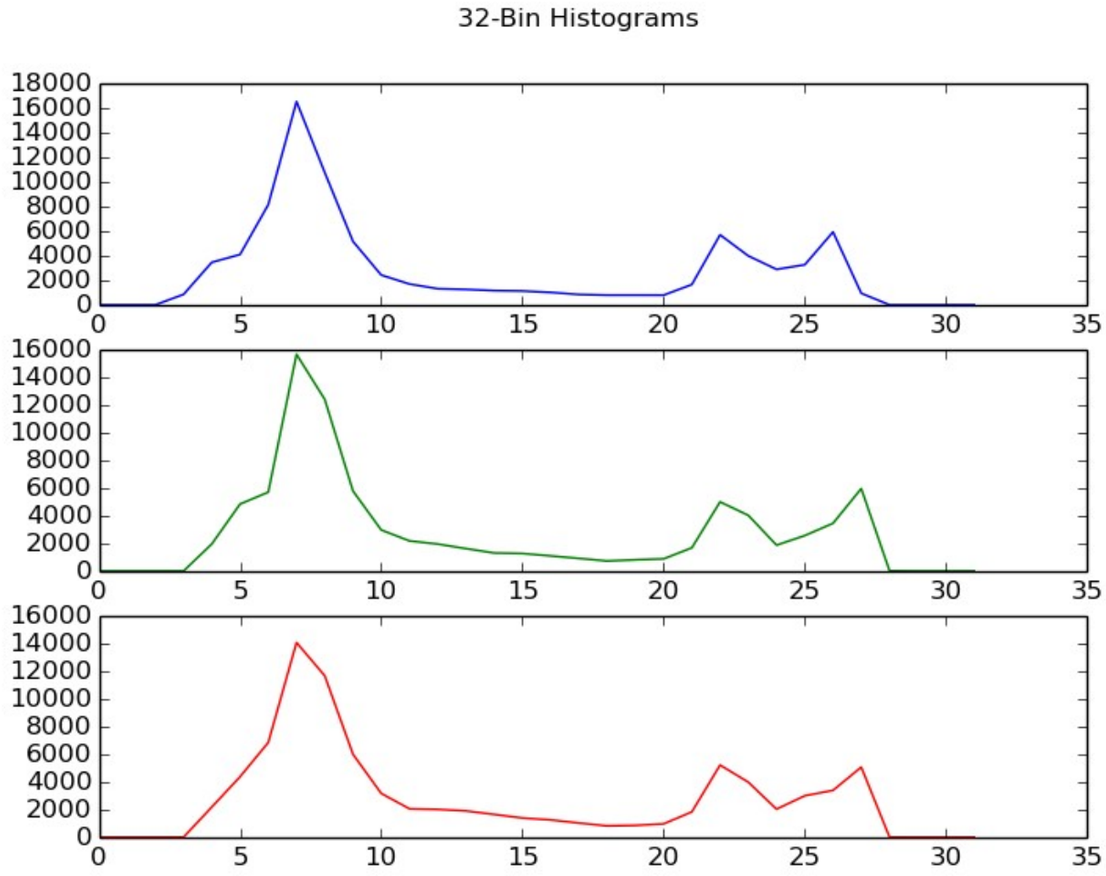
$ python histo_01.py -i img/123459076.JPG

Sample Output: Three 256-Bin BGR Histograms



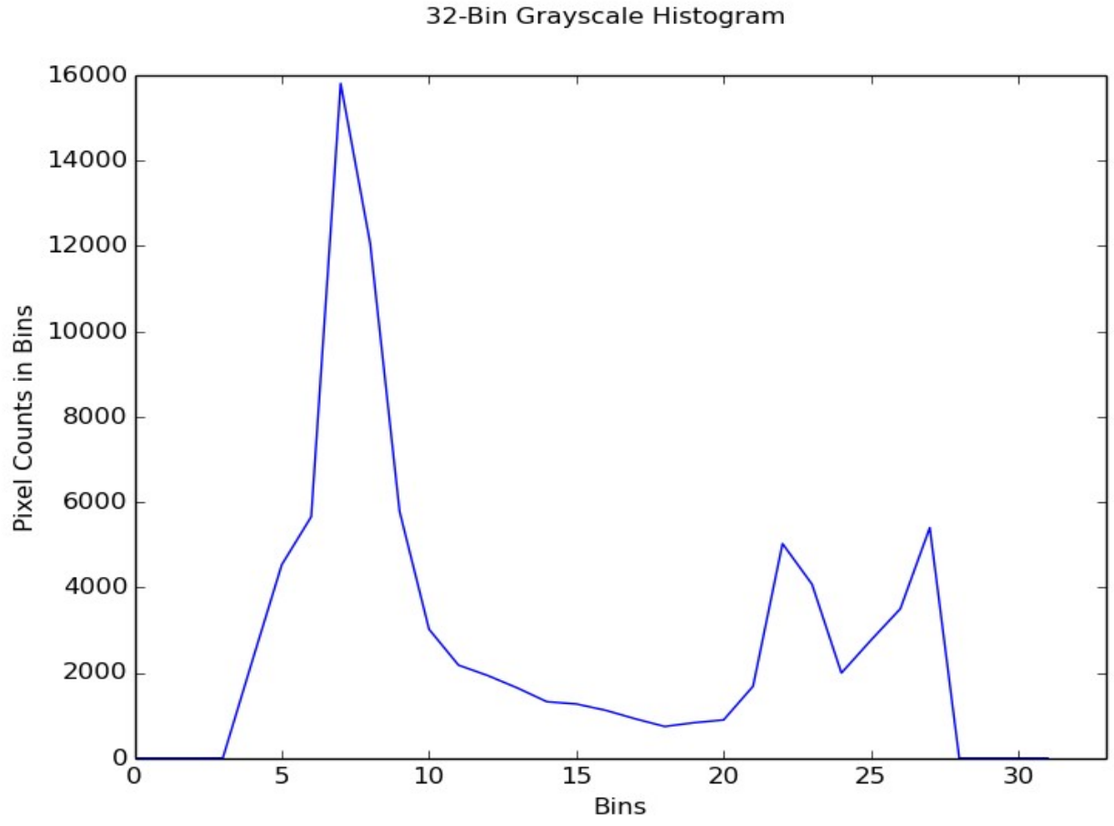
$ python histo_01.py -i car_test/img22.png

Sample Output: Three 32-Bin BGR Histograms



$ python histo_01.py -i car_test/img22.png

Sample Output: Three 32-Bin Grayscale Histograms



$ python histo_01.py -i car_test/img22.png

Solution: Computing Histograms

There is a gotcha in the code below: the RGB color space is [0, 255]. However, the specified range is [0, 256]. What gives? calcHist() uses numpy's histogram method that turns [0, 256] into [0, 256) which, in turn, becomes [0, 255]. In most cases, there is no difference - you are just one color range value short. But, definitely a gotcha!

```
blu_histo_256 = cv2.calcHist([image], [0], None, [256], [0, 256])
grn_histo_256 = cv2.calcHist([image], [1], None, [256], [0, 256])
red_histo_256 = cv2.calcHist([image], [2], None, [256], [0, 256])
```

Solution: Showing Image with MATPLOTLIB in Figure 1

This is another gotcha but this one is in matplotlib. Whenever, I display the image in matplotlib w/o converting it into the standard RGB format, the displayed image is grayish. Hence, explicit conversion from BGR to RGB to avoid the problem.

```
rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
fig1 = plt.figure(1)
fig1.suptitle('Image')
plt.imshow(rgb)
```

Solution: Plotting 256-Bin Color Histograms in Figure 2

Another gotcha I frequently fall far: it is suptitle, not subtitle!

Limiting the x-axis values to run from 0 to 256

Labeling x-axis

Labeling y-axis

Plotting the first of three subplots

Plotting the second of three subplots

Plotting the third of three subplots

```
fig2 = plt.figure(2)
fig2.suptitle('256-Bin Histograms')
plt.xlim([0, 256])
plt.xlabel('Bins')
plt.ylabel('Pixel Counts in Bins')
plt.subplot(311)
plt.plot(blu_histo_256, color='b')
plt.subplot(312)
plt.plot(grn_histo_256, color='g')
plt.subplot(313)
plt.plot(red_histo_256, color='r')
```

Solution: Plotting 32-Bin Color Histograms in Figure 3

Limiting the x-axis values to run from 0 to 33

Labeling x-axis

Labeling y-axis

Plotting the first of three subplots

Plotting the second of three subplots

Plotting the third of three subplots

```
fig3 = plt.figure(3)
fig3.suptitle('32-Bin Histograms')
plt.xlim([0, 33])
plt.xlabel('Bins')
plt.ylabel('Pixel Counts in Bins')
plt.subplot(311)
plt.plot(blu_histo_32, color='b')
plt.subplot(312)
plt.plot(grn_histo_32, color='g')
plt.subplot(313)
plt.plot(red_histo_32, color='r')
```

Solution: Plotting Grayscale 32-Bin Histogram in Figure 4

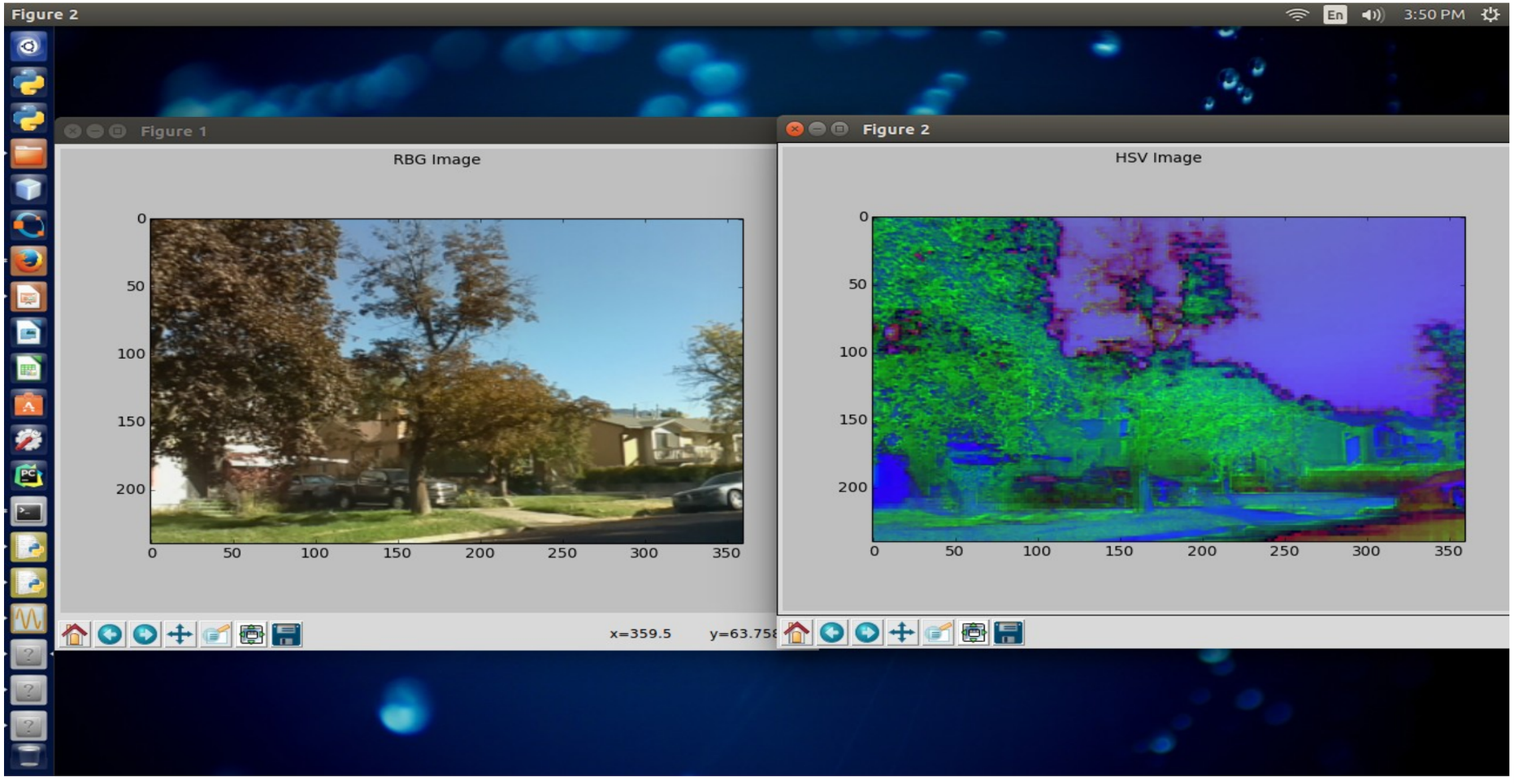
```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray_histo_32 = cv2.calcHist([gray_image], [0], None, [32], [0, 256])
fig4 = plt.figure(4)
fig4.suptitle('32-Bin Grayscale Histogram')
plt.xlim([0, 33])
plt.xlabel('Bins')
plt.ylabel('Pixel Counts in Bins')
plt.subplot(111)
plt.plot(gray_histo_32)
```

Problem

Write a program that takes an RGB image, converts it to HSV, and computes and plots histograms for H, S, and V channels in the image. Histograms should have either 180 bins (for H) 256 bins (for S and V) or 32 bins for all channels.

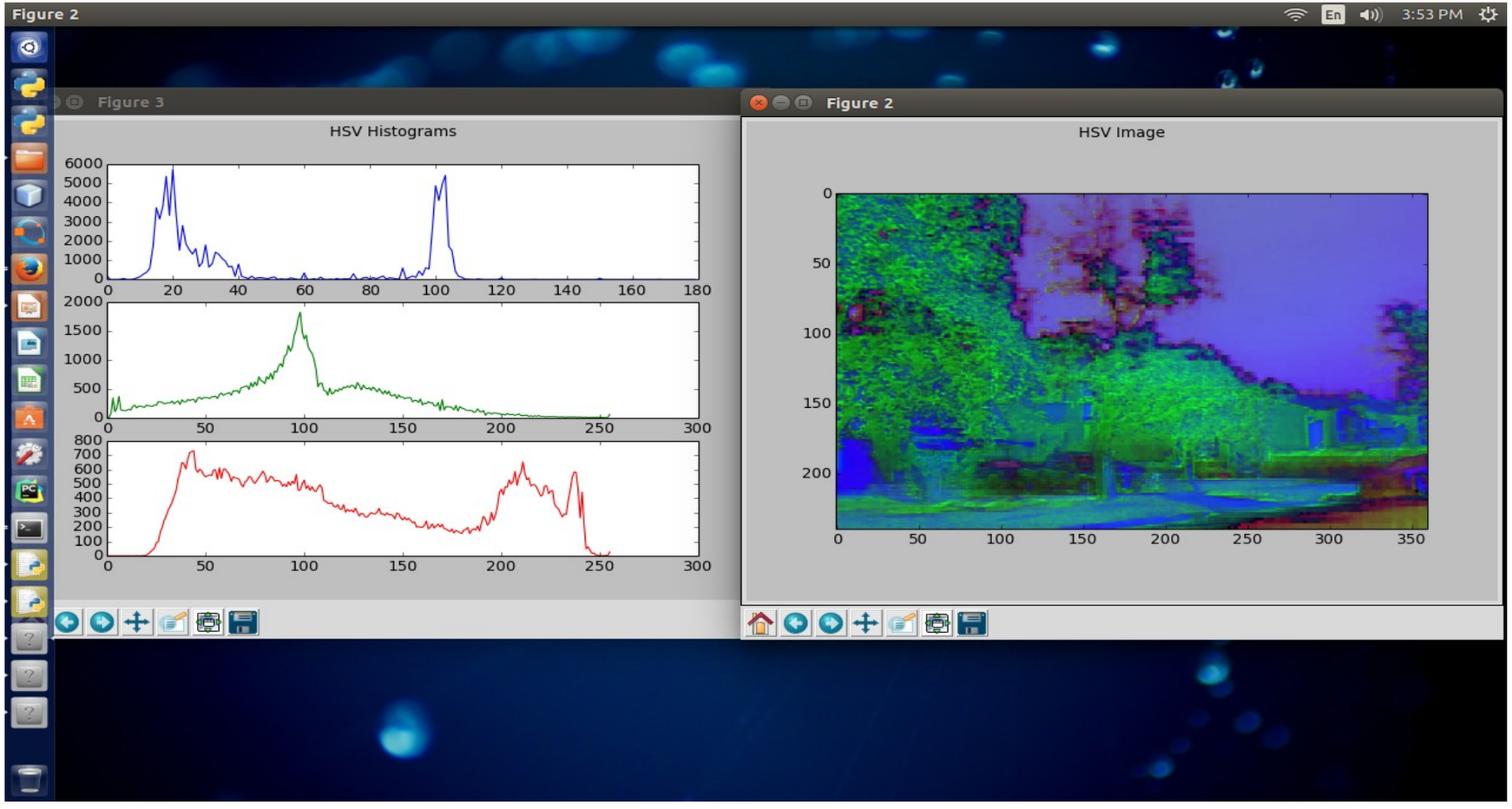
py source in histo_01b.py

Sample Output: RGB & HSV Images



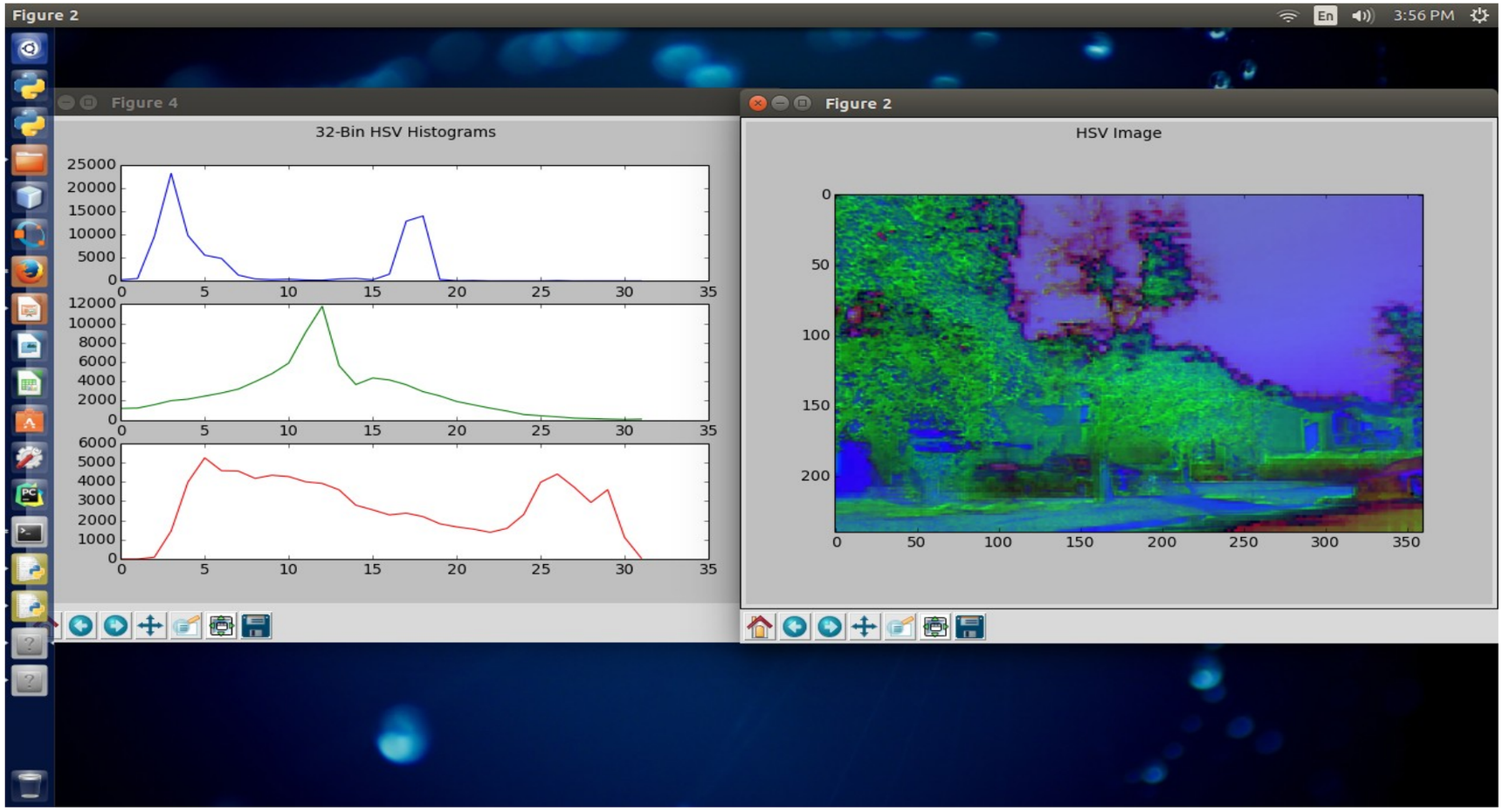
$ python histo_01b.py -i car_test/img03.png

Sample Output: 180- and 256-Bin HSV Histograms



$ python histo_01b.py -i img03.png

Sample Output: Three 32-Bin HSV Histograms



$ python histo_01b.py -i img03.png

Solution: Computing HSV Histograms

The hue range is [0, 179]

```
hue_histo_180 = cv2.calcHist([hsv_img], [0], None, [180], [0, 180])
sat_histo_256 = cv2.calcHist([hsv_img], [1], None, [256], [0, 256])
val_histo_256 = cv2.calcHist([hsv_img], [2], None, [256], [0, 256])
```

Solution: Computing 32-Bin HSV Histograms

The hue range is [0, 179]

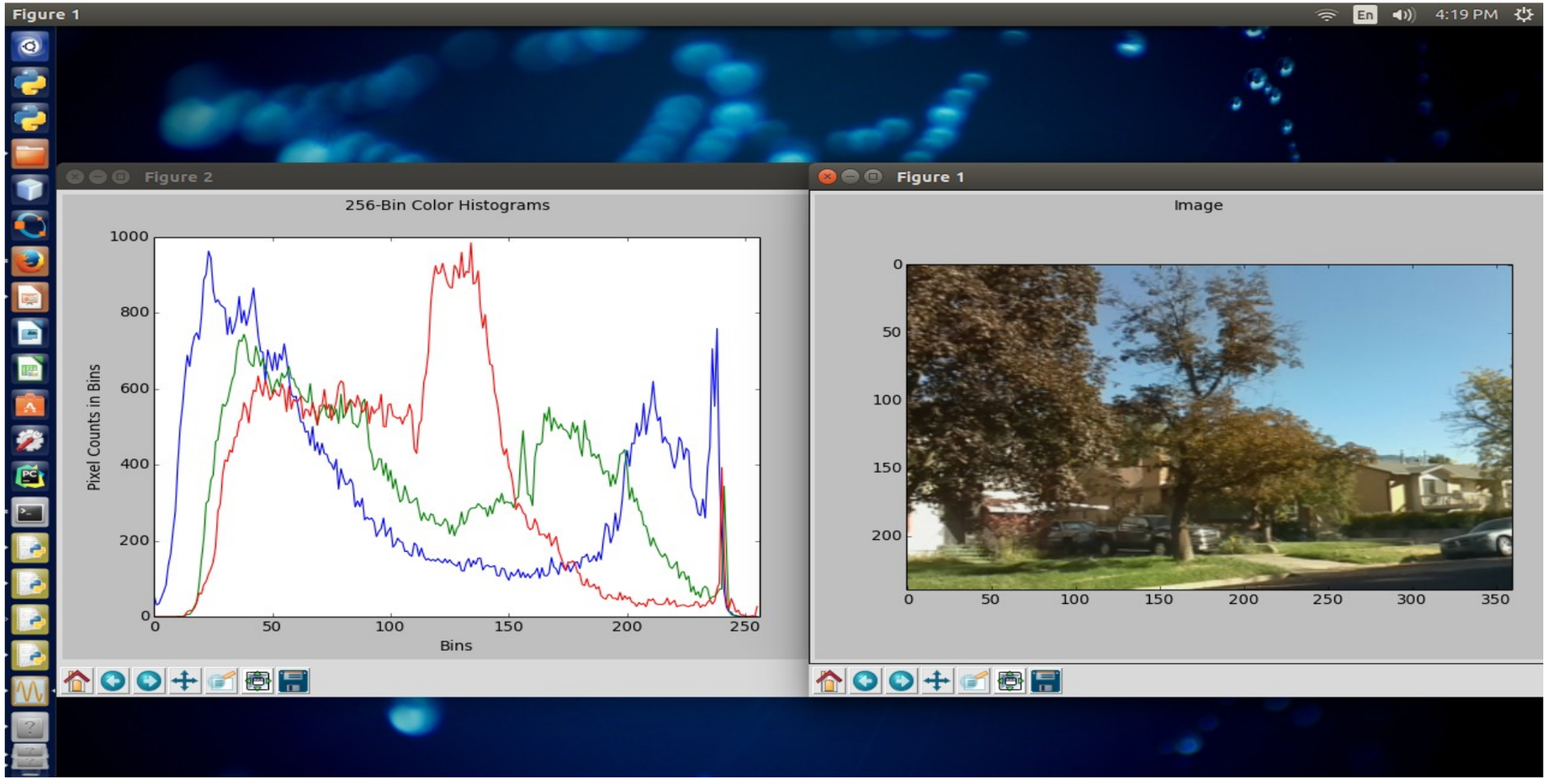
```
hue_histo_32 = cv2.calcHist([hsv_img], [0], None, [32], [0, 180])
sat_histo_32  = cv2.calcHist([hsv_img], [1], None, [32], [0, 256])
val_histo_32  = cv2.calcHist([hsv_img], [2], None, [32], [0, 256])
```

Problem

Write a program that takes an RGB image and computes and plots histograms for R, G, and B channels in the same plot. Histograms can have 256 bins, 32 bins, and 16 bins.

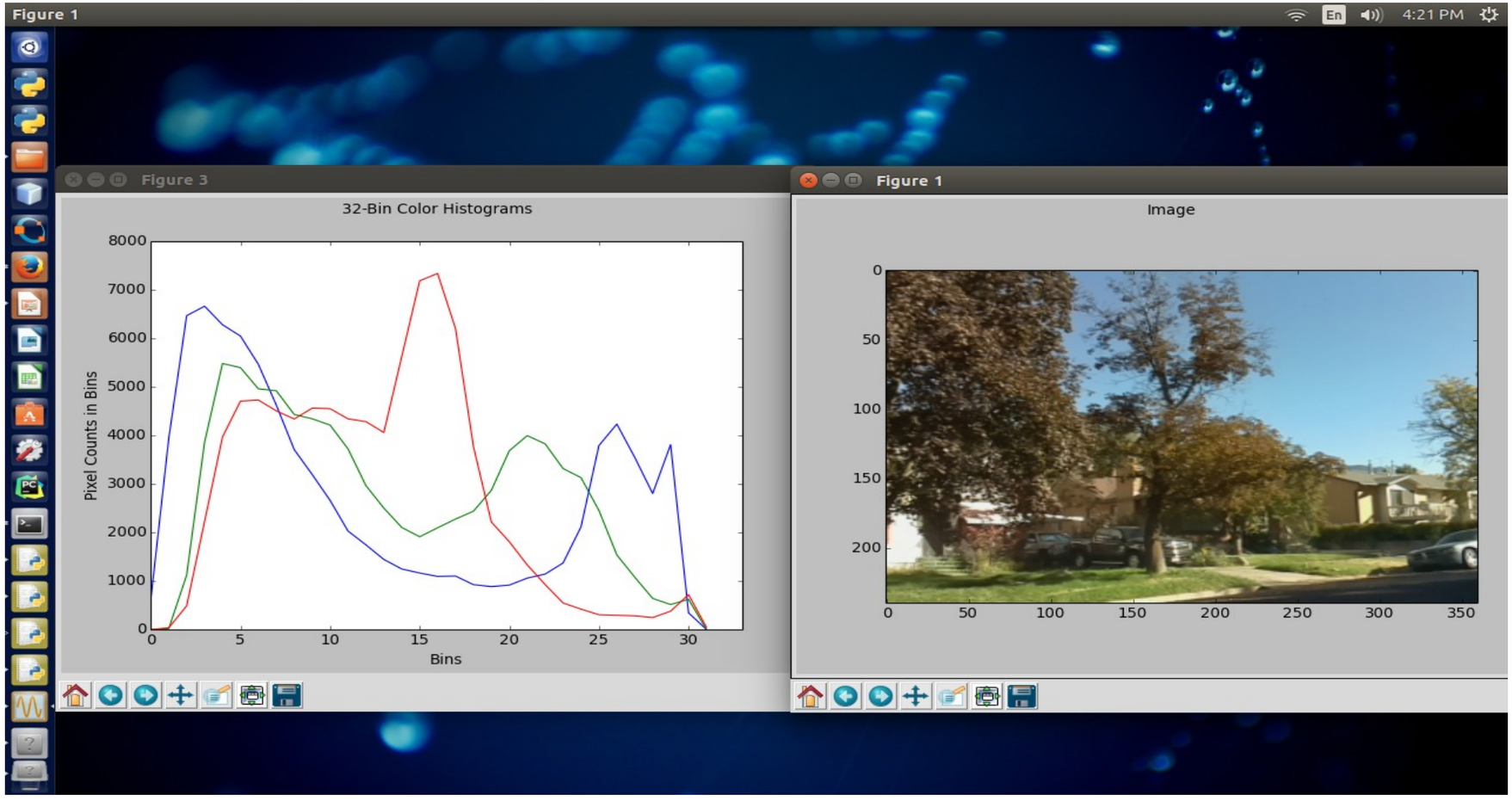
py source in histo_02.py

Sample Output: Three 256-Bin RGB Histograms



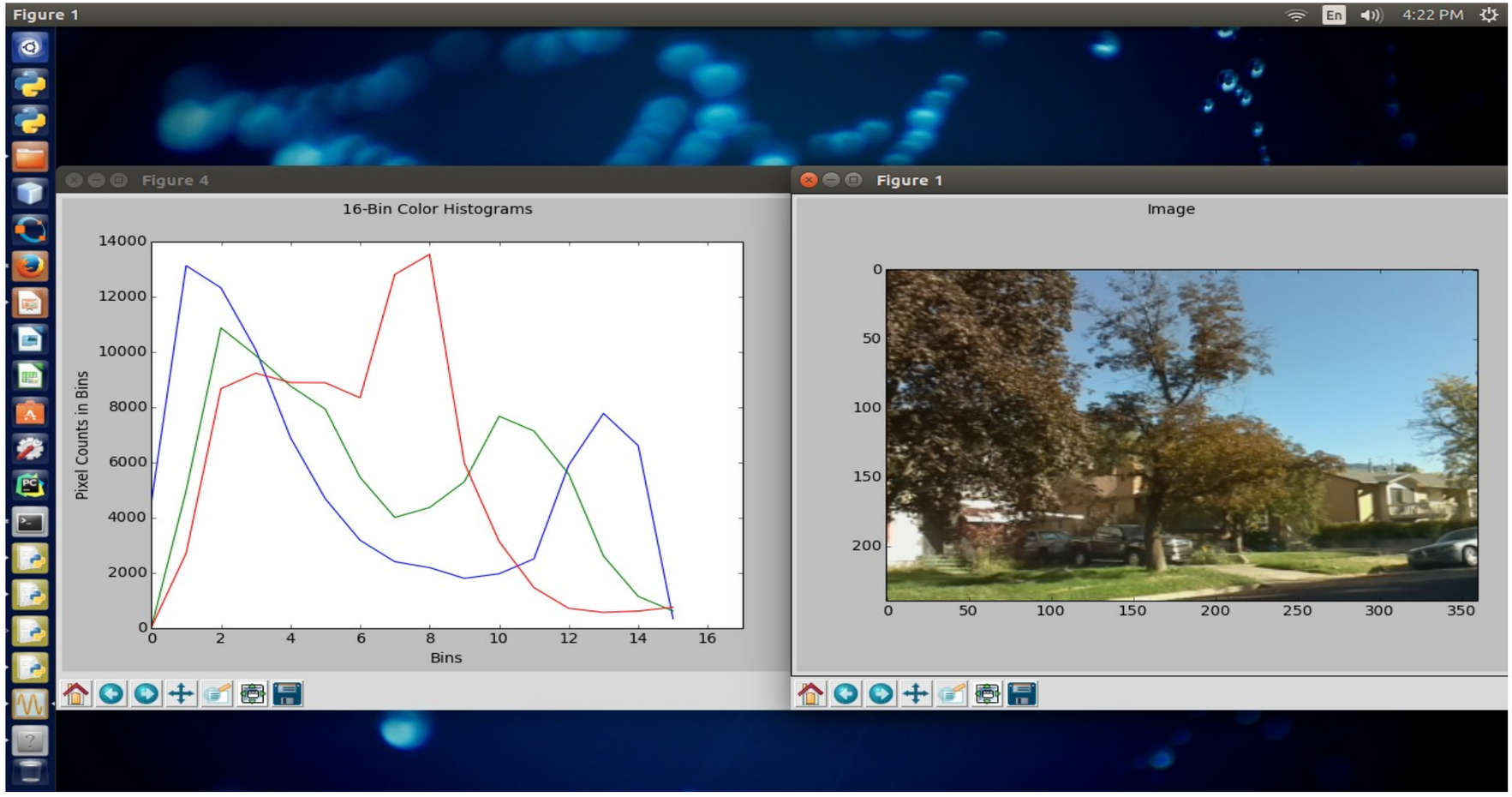

```
$ python histo_02.py -i img03.png
    flattened histo_list_256 size: 768
    flattened histo_list_32 size: 96
    flattened histo_list_16 size: 48
```

Sample Output: Three 32-Bin RGB Histograms




```
$ python histo_02.py -i img03.png
    flattened histo_list_256 size: 768
    flattened histo_list_32 size: 96
    flattened histo_list_16 size: 48
```

Sample Output: Three 16-Bin RGB Histograms




```
$ python histo_02.py -i img03.png
    flattened histo_list_256 size: 768
    flattened histo_list_32 size: 96
    flattened histo_list_16 size: 48
```

Solution: Displaying Dimensions of Flattened Histograms

```
# show the dimensionality of the flattened color histograms.
# hist_list_256, when flattened, has 256 bins for each channel x 3 channels = 768 values.
# histo_list_32, when flattened, has 32 bins for each channel x 3 channels = 96 values.
# hist_list_16, when flattened, hast 16 bins for each channel x 3 channels = 48 values.
print 'flattened histo_list_256 size: %d' % (np.array(histo_list_256).flatten().shape)
print 'flattened histo_list_32 size: %d' % (np.array(histo_list_32).flatten().shape)
print 'flattened histo_list_16 size: %d' % (np.array(histo_list_16).flatten().shape)
```

Turning Histograms into Feature Vectors

Turning Histograms into Feature Vectors

- OK, so we know how to compute a histogram of an image
- Can we use that histogram to find similar images?
- Yes, we can. But we need to turn histograms into a format easy to use in matching functions; this format is called **feature vector**
- To turn a histogram into a feature vector, two operations are needed: normalization and flattening

Normalization and Flattening

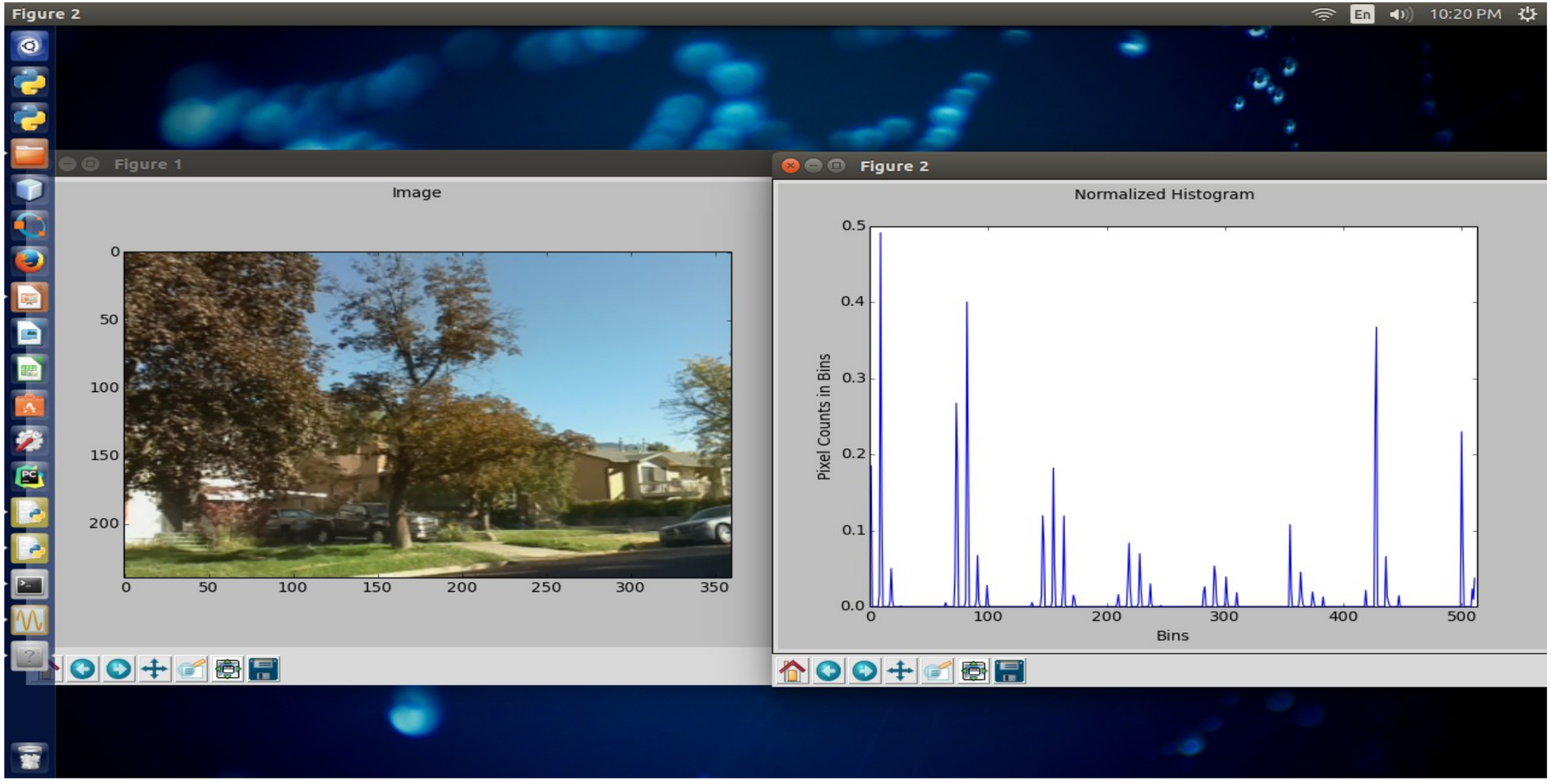
- Histogram normalization is a procedure that takes a histogram and scales the bin counts to be in **[0, 1]**
- **cv2.normalize(input_hist, dest_hist)** - histogram **input_hist** is normalized and placed into histogram **dest_hist**
- Histogram flattening simply turns a 2D histogram into a 1D array
- **cv2.normalize(input_hist, dest_hist).flatten()** is all it takes to flatten a normalized histogram

Problem

Write a program that takes an RGB image and plots its normalized flattened histogram.

py source in histo_03.py

Sample Output



$ python histo_03.py -i img03.png

Solution: Normalizing and Flattening Histogram

```
# Calculate a histogram of three colors with 8 bins per color.
# Since we have three colors with 8 bins per color, there will
# be 8^3 = 512 values in input_hist. The array [0, 256, 0, 256, 0, 256]
# specifies the range for each color. The first two numbers 0, 256 specify
# the range for blue; the second two numbers (i.e., 0, 256) specify the
# range for green; the third two numbers (i.e., 0, 256) specify the range for red.
input_hist = cv2.calcHist([image], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
norm_hist = cv2.normalize(input_hist, input_hist).flatten()
print(norm_hist.shape)
```

Matching Image Histograms

Histogram Matching

- Histogram matching is a fascinating topic in mathematics and statistics
- The histogram matching problem can be formulated as follows: given two histograms **H1** and **H2**, how does one compute how similar they are?
- The similarity coefficient (aka matching coefficient) is called distance and is typically referred to as **d(H1, H2)**

Distance Metrics in OpenCV

- OpenCV has four main histogram distance metrics:
  - Correlation
  - Chi Square
  - Intersection
  - Bhattacharrya
- The next four slides give the mathematical formulas for each distance metric; do not worry if they look hard (they are that hard!) and we will use them, not implement them

Correlation: cv2.HISTCMP_CORREL

Pixel count in bin I of H1

Mean pixel count in H1

Pixel count in bin I of H2

Mean pixel count in H2

Distance b/w histograms H1 and H2

$$d(H_1, H_2) = \frac{\sum_I (H_1(I) - \bar{H}_1)(H_2(I) - \bar{H}_2)}{\sqrt{\sum_I (H_1(I) - \bar{H}_1)^2 \sum_I (H_2(I) - \bar{H}_2)^2}}$$

when images are the same, the correlation coefficient is 1.0; the more dissimilar the images, the closer the coefficient is to 0.

Mean pixel count in Hk

$$\bar{H}_k = \frac{1}{N} \sum_J H_k(J)$$

Chi Square: cv2.HISTCMP_CHISQR

$$d(H_1, H_2) = \sum_I \frac{(H_1(I) - H_2(I))^2}{H_1(I)}$$

when the images are similar, the coefficient is 0 or close to 0; the more dissimilar the images, the greater the value of this coefficient.

Intersection: cv2.HISTCMP_INTERSECT

$$d(H_1, H_2) = \sum_I \min(H_1(I), H_2(I))$$

the more similar the images, the greater the value of the intersection.

Bhattacharyya: cv2.HISTCMP_BHATTACHARYYA

$$d(H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{\bar{H}_1 \bar{H}_2 N^2}} \sum_I \sqrt{H_1(I) \cdot H_2(I)}}$$

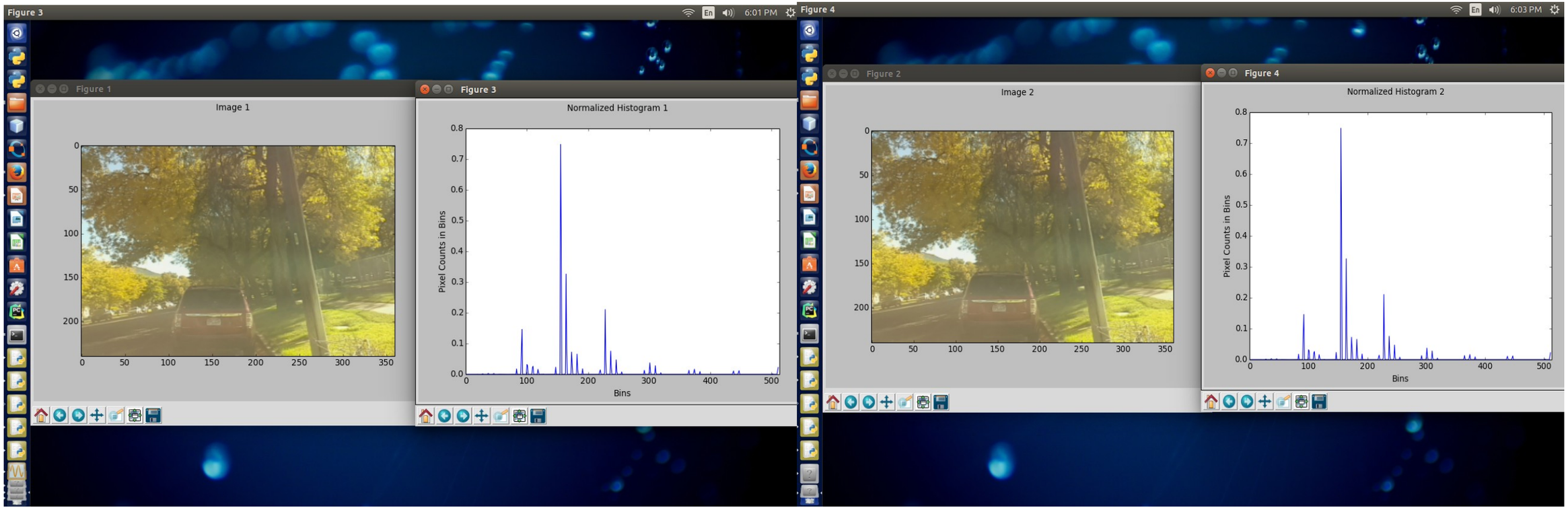
if images are similar, the distance is 0 or close to 0;
if images are dissimilar, the distance is 1 or close to 1.

Problem

Write a program that takes a command line for two images, computes their flattened histograms and prints out the values of the four histogram match metrics.

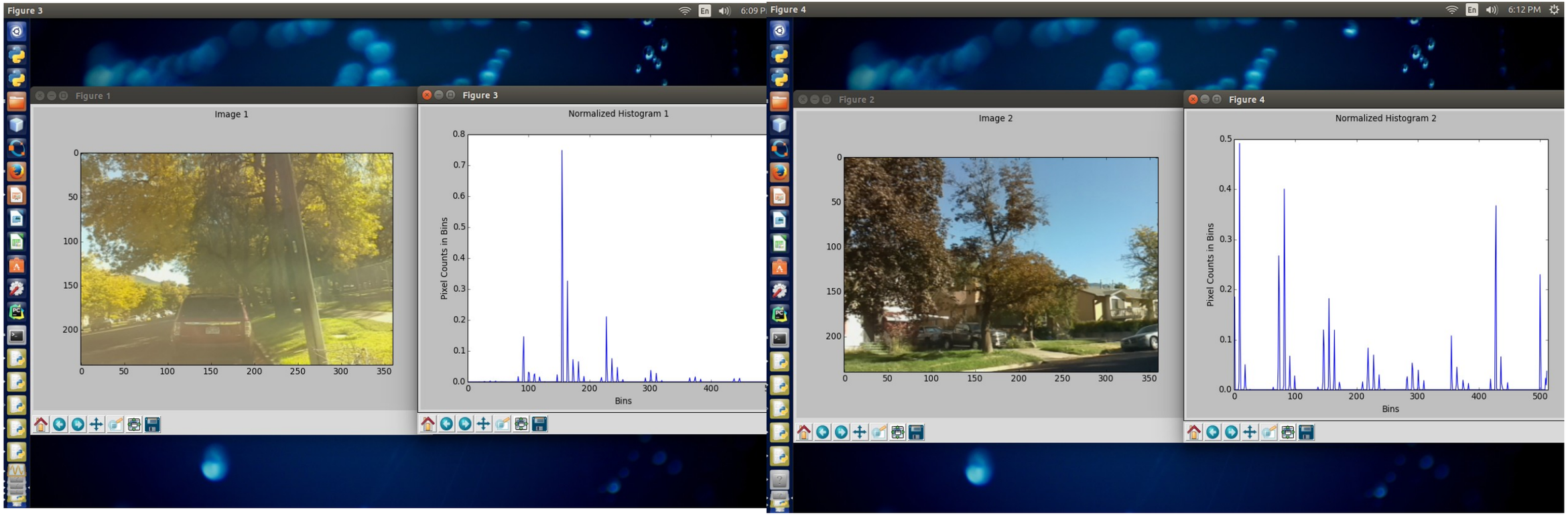
py source in histo_04.py

Sample Output



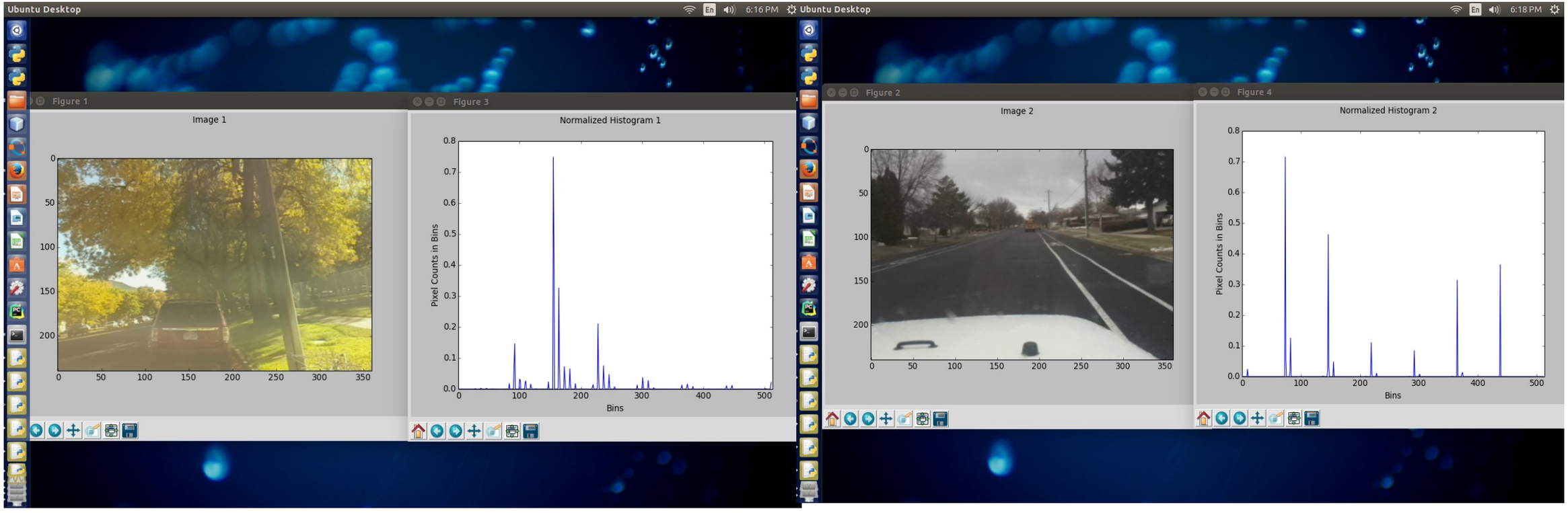

```
$ python histo_04.py -i1 img01.png -i2 img01.png
  cv2.HISTCMP_BHATTA = 0.0
  cv2.HISTCMP_CORREL = 1.0
  cv2.HISTCMP_INTERSECT = 2.93410793478
  cv2.HISTCMP_CHISQR = 0.0
```

Sample Output




```
$ python histo_04.py -i1 img01.png -i2 img03.png
  cv2.HISTCMP_BHATTA = 0.765386057275
  cv2.HISTCMP_CORREL = 0.218552132353
  cv2.HISTCMP_INTERSECT = 0.837307250531
  cv2.HISTCMP_CHISQR = 1109.86647865
```

Sample Output




```
$ python histo_04.py -i1 car_test/img01.png -i2 car_test/img22.png
  cv2.HISTCMP_BHATTA = 0.892502687061
  cv2.HISTCMP_CORREL = 0.0360421920083
  cv2.HISTCMP_INTERSECT = 0.175548713978
  cv2.HISTCMP_CHISQR = 1163.53214697
```

Solution: Computing & Flattening Histograms

```
image1 = cv2.imread(args['img1'])
image2 = cv2.imread(args['img2'])
# compute histograms
hist1 = cv2.calcHist([image1], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
hist2 = cv2.calcHist([image2], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
# normalize and flatten histograms to compute feature vectors
norm_hist1 = cv2.normalize(hist1, hist1).flatten()
norm_hist2 = cv2.normalize(hist2, hist2).flatten()
```

Solution: Matching Histograms & Displaying Match Scores

```
# let's create a dictionary mapping metrics to matching scores
dist_table = {}
dist_table['cv2.HISTCMP_CORREL'] = cv2.compareHist(norm_hist1, norm_hist2,  cv2.HISTCMP_CORREL)
dist_table['cv2.HISTCMP_CHISQR'] = cv2.compareHist(norm_hist1, norm_hist2, cv2.HISTCMP_CHISQR)
dist_table['cv2.HISTCMP_INTERSECT'] = cv2.compareHist(norm_hist1, norm_hist2, cv2.HISTCMP_INTERSECT)
dist_table['cv2.HISTCMP_BHATTA'] = cv2.compareHist(norm_hist1, norm_hist2, cv2.HISTCMP_BHATTACHARYYA)


# let's display the metrics and the scores
for metric, score in dist_table.items():
    print(metric + ' = ' + str(score))
```

References

http://docs.opencv.org/2.4/modules/imgproc/doc/histograms.html