Numerical Methods and Machine Learning for Image Processing

Week 3, Class 1: Basic Color Segmentation 2

October 11, 2021

Damon M. Chandler and Yi Zhang

Last Time: Basic Color Segmentation, Part 1

1. *k*-means clustering idea
2. Using *k*-means clustering of colors for image segmentation in the…
   - RGB color space
   - Hue, Value, Saturation (HSV) color space
   - CIE L*a*b* color space
3. Using color (and lightness) to find salient regions
   - How to compute lightness and color distance feature maps

Today: Basic Color Segmentation, Part 2

1. Grayscale thresholding
2. How to automatically find the threshold(s)
   - Otsu's method
   - Gaussian mixture model (GMM)
3. Using a GMM for grayscale segmentation

Today: Basic Color Segmentation, Part 2

1. Grayscale thresholding
2. How to automatically find the threshold(s)
   - Otsu's method
   - Gaussian mixture model (GMM)
3. Using a GMM for grayscale segmentation

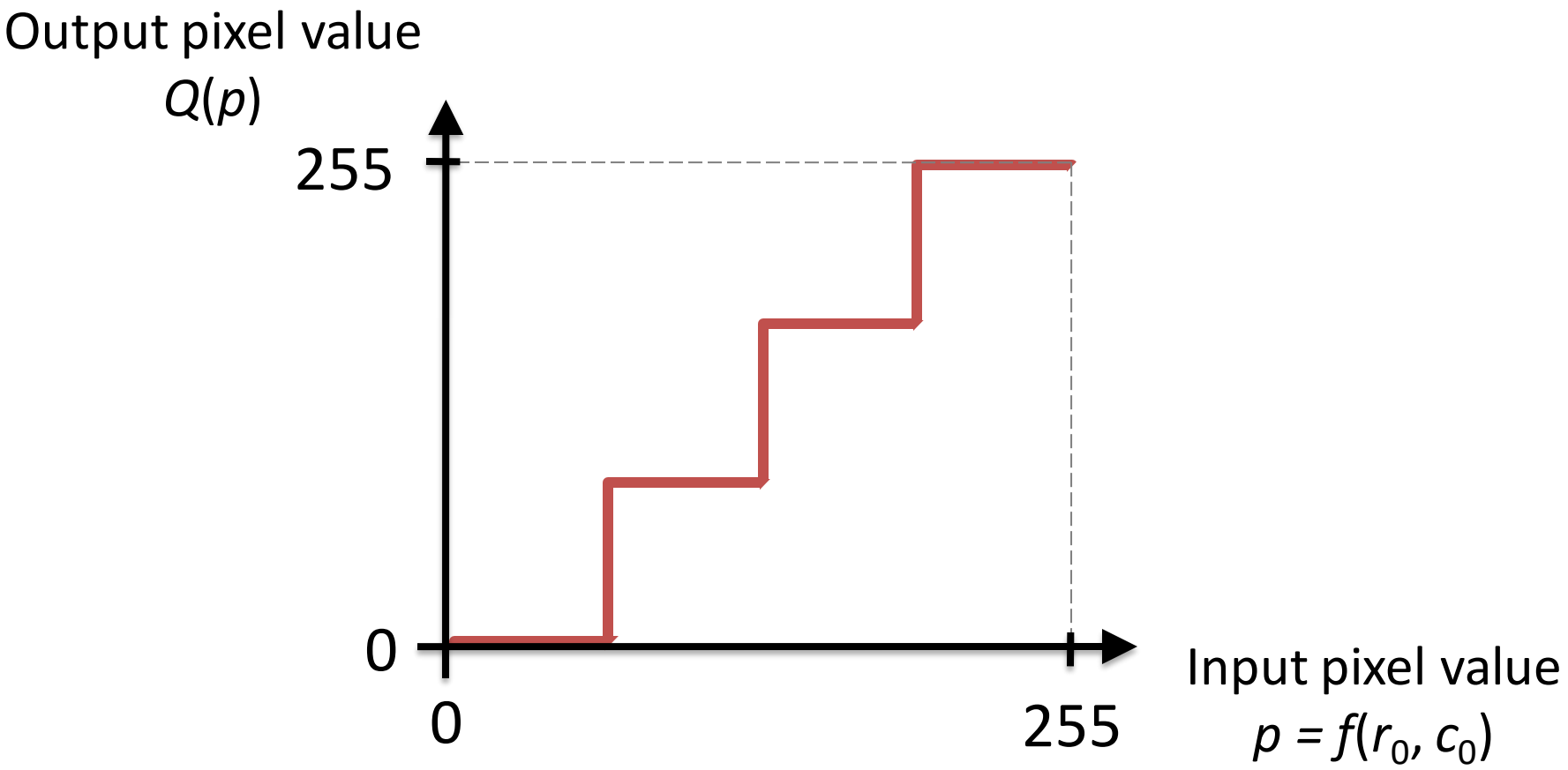
Gray-Levels and Bit-Depth

- The "intensity resolution" of images tells how you many shades of gray the image representation uses:
  - N-bpp → 2^N shades of gray, each pixel $\in$ [0, 2^N-1]
  - 8-bpp → 256 shades of gray, each pixel $\in$ [0, 255]
  - 7-bpp → 128 shades of gray, each pixel $\in$ [0, 127]
  - 6-bpp → 64 shades of gray, each pixel $\in$ [0, 63]
  - 2-bpp → 4 shades of gray, each pixel $\in$ [0, 3]
  - 1-bpp → 256 shades of gray, each pixel $\in$ [0, 1]

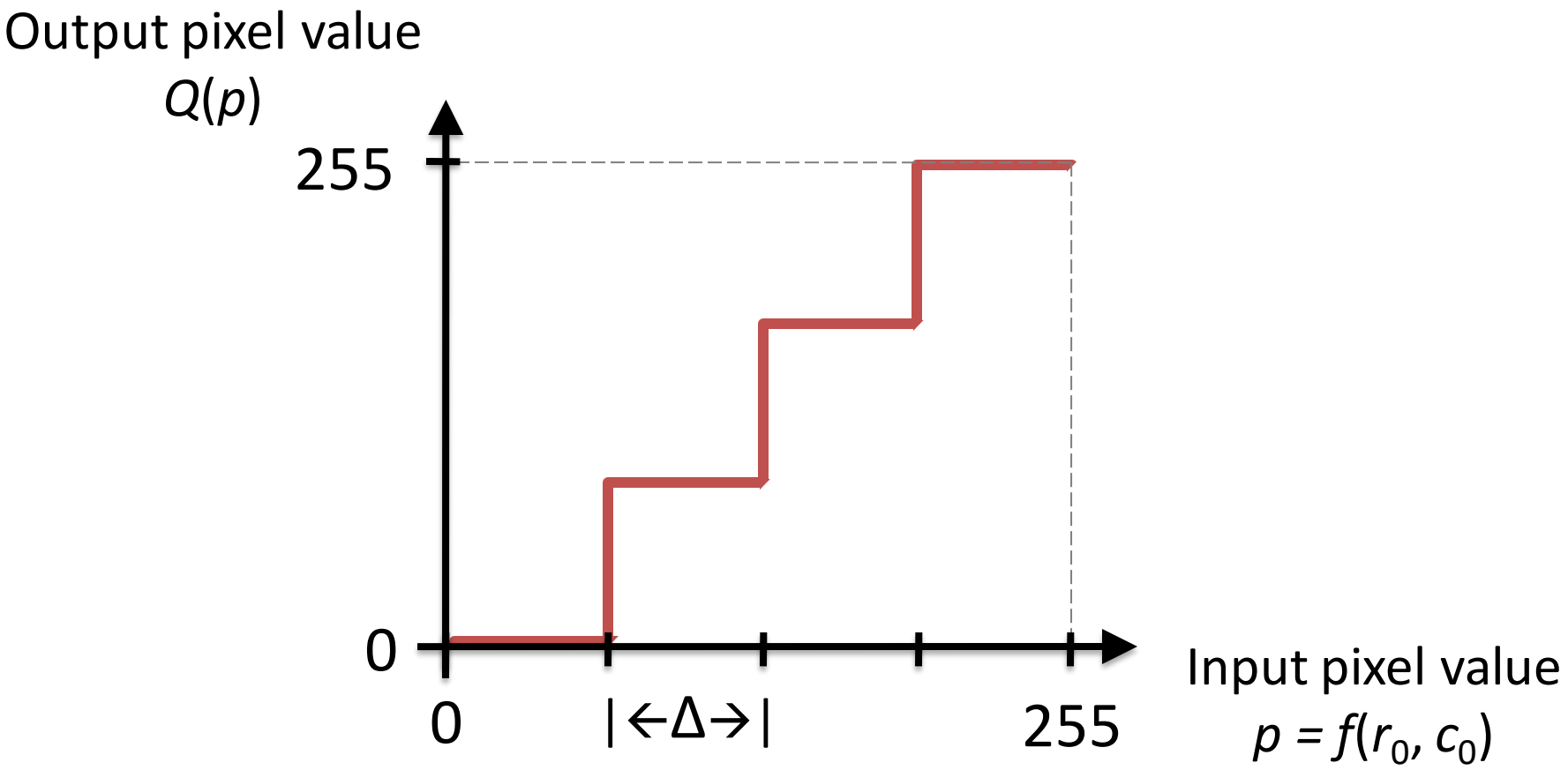
Gray-Levels and Bit-Depth

- Gray-level **quantization** is the process of mapping the pixel values to a lower bpp representation
- ***N*-bit quantization** means:
  - Convert a given image to use *N* bits per pixels
  - Easy (but crude) approach to image compression
  - Can result in severe degradation in image quality
  - No-op if image is already *N*-bpp

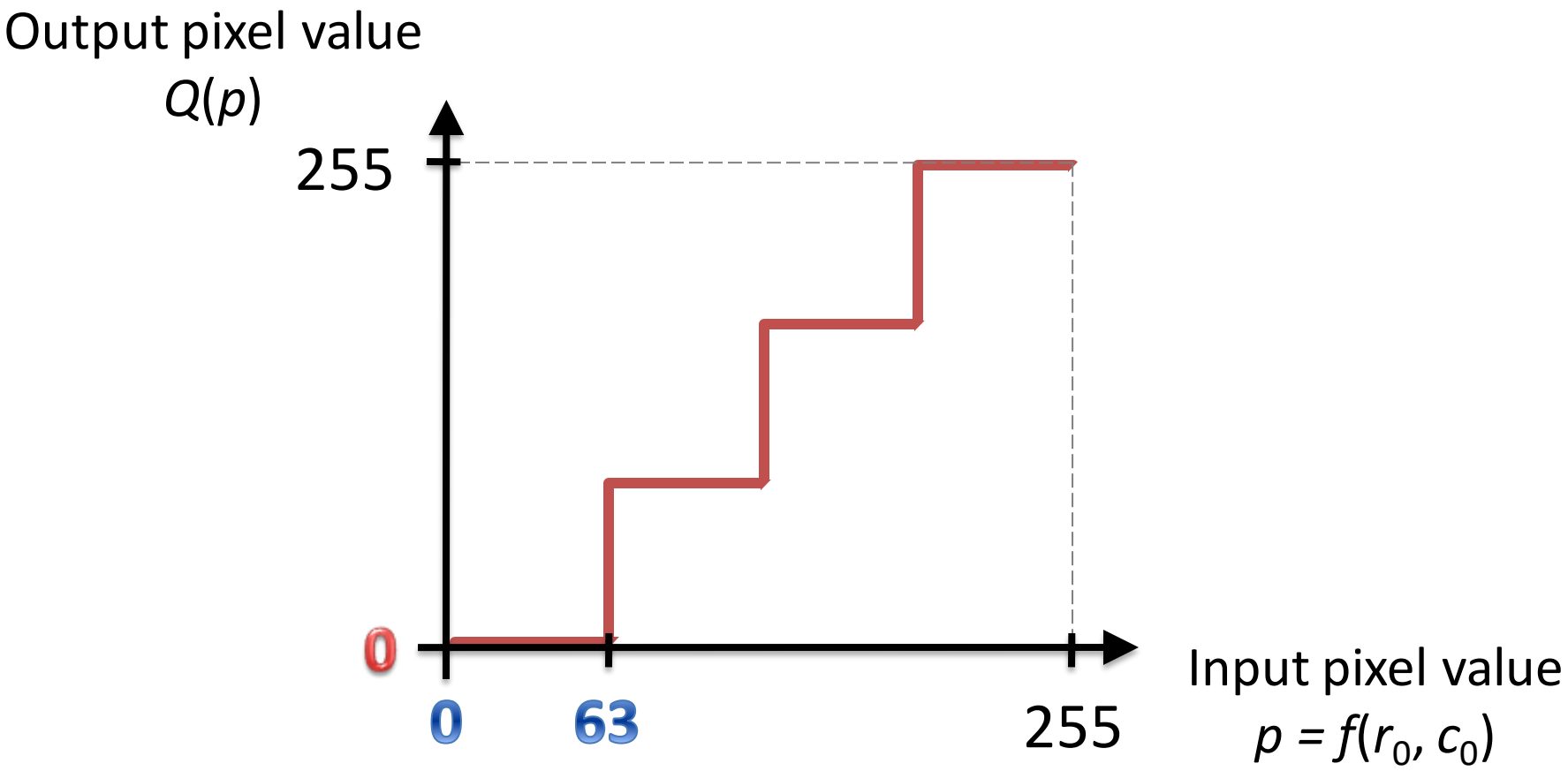
Gray-Level Quantization



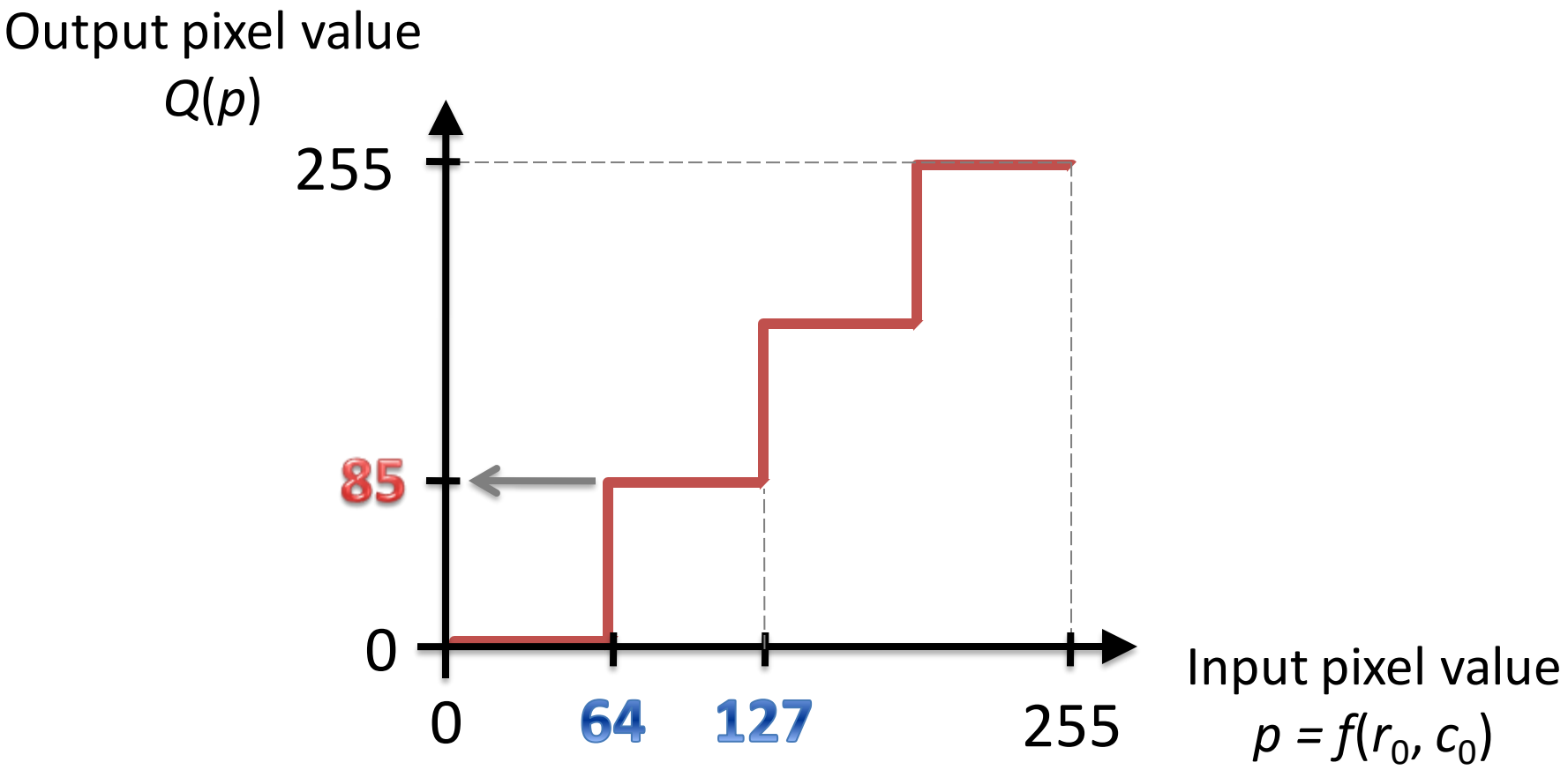
Gray-Level Quantization



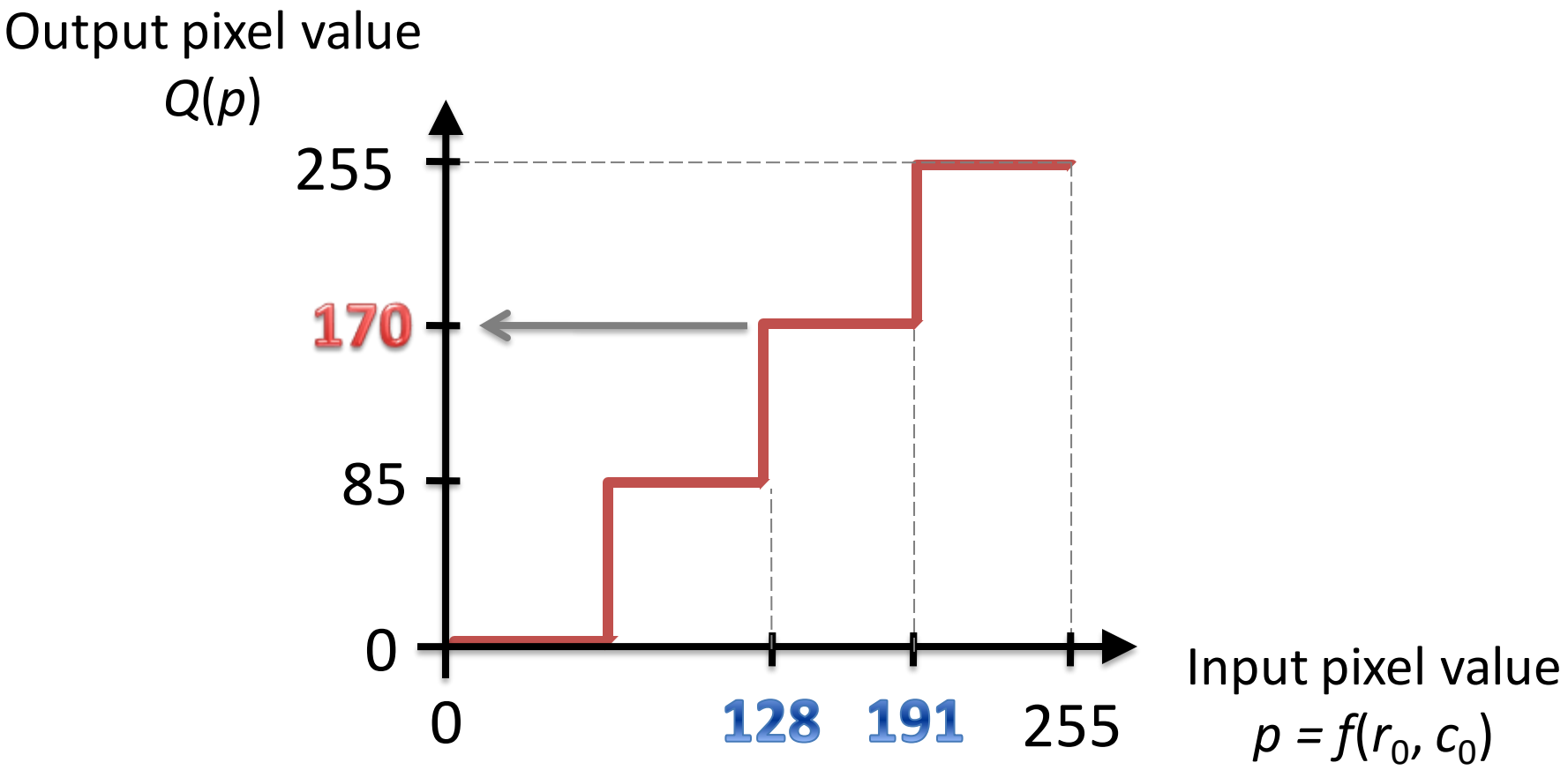
Gray-Level Quantization



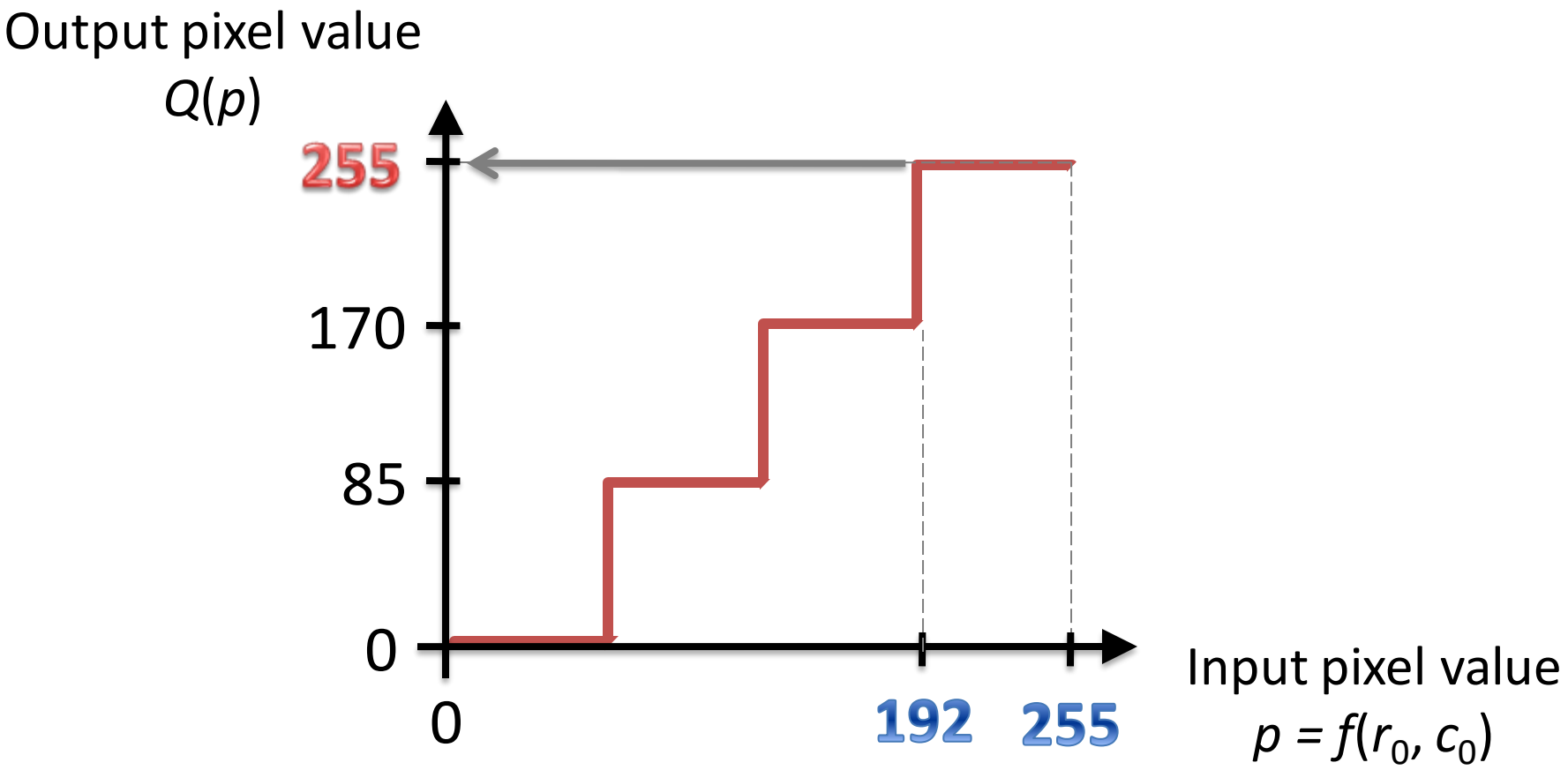
Gray-Level Quantization



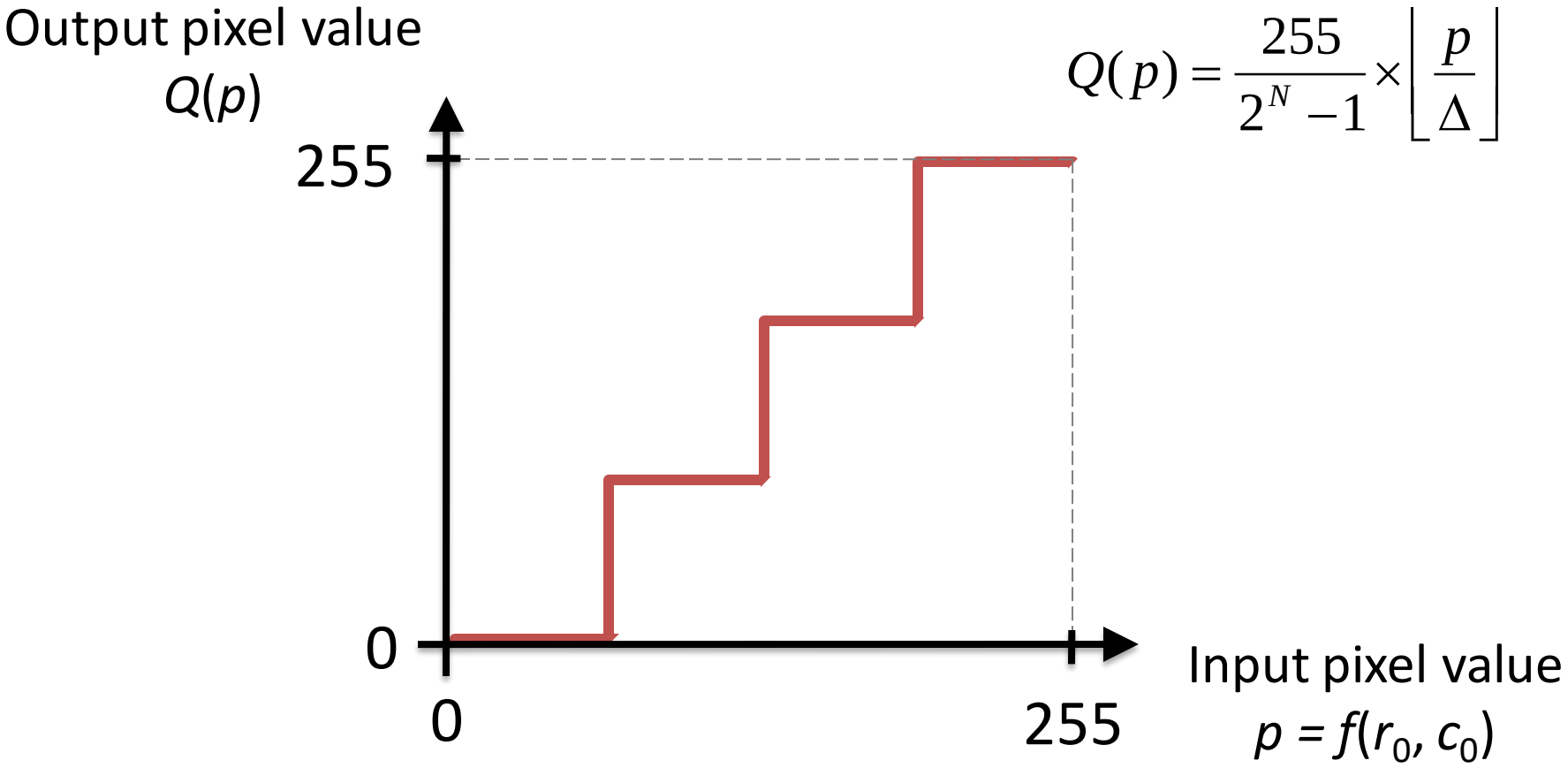
Gray-Level Quantization



Gray-Level Quantization

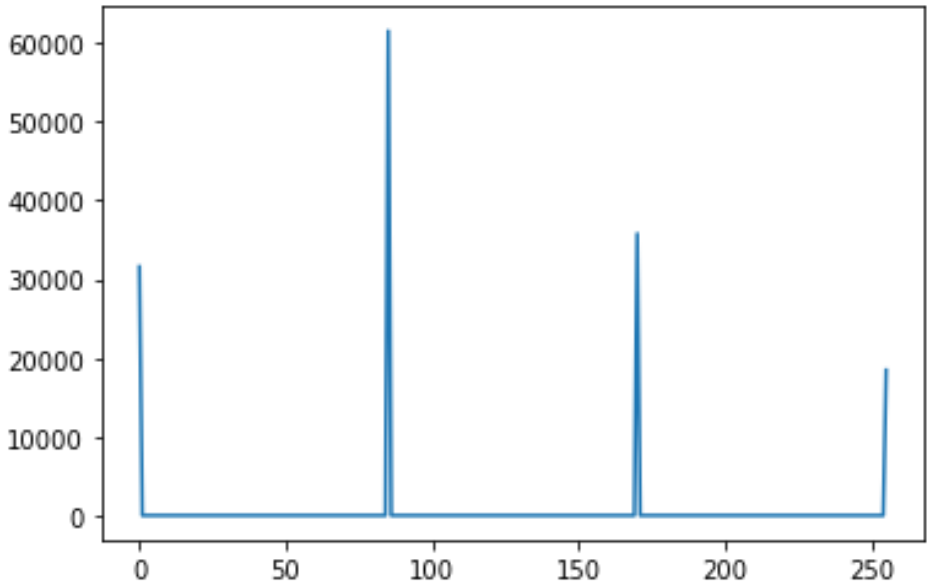


Gray-Level Quantization

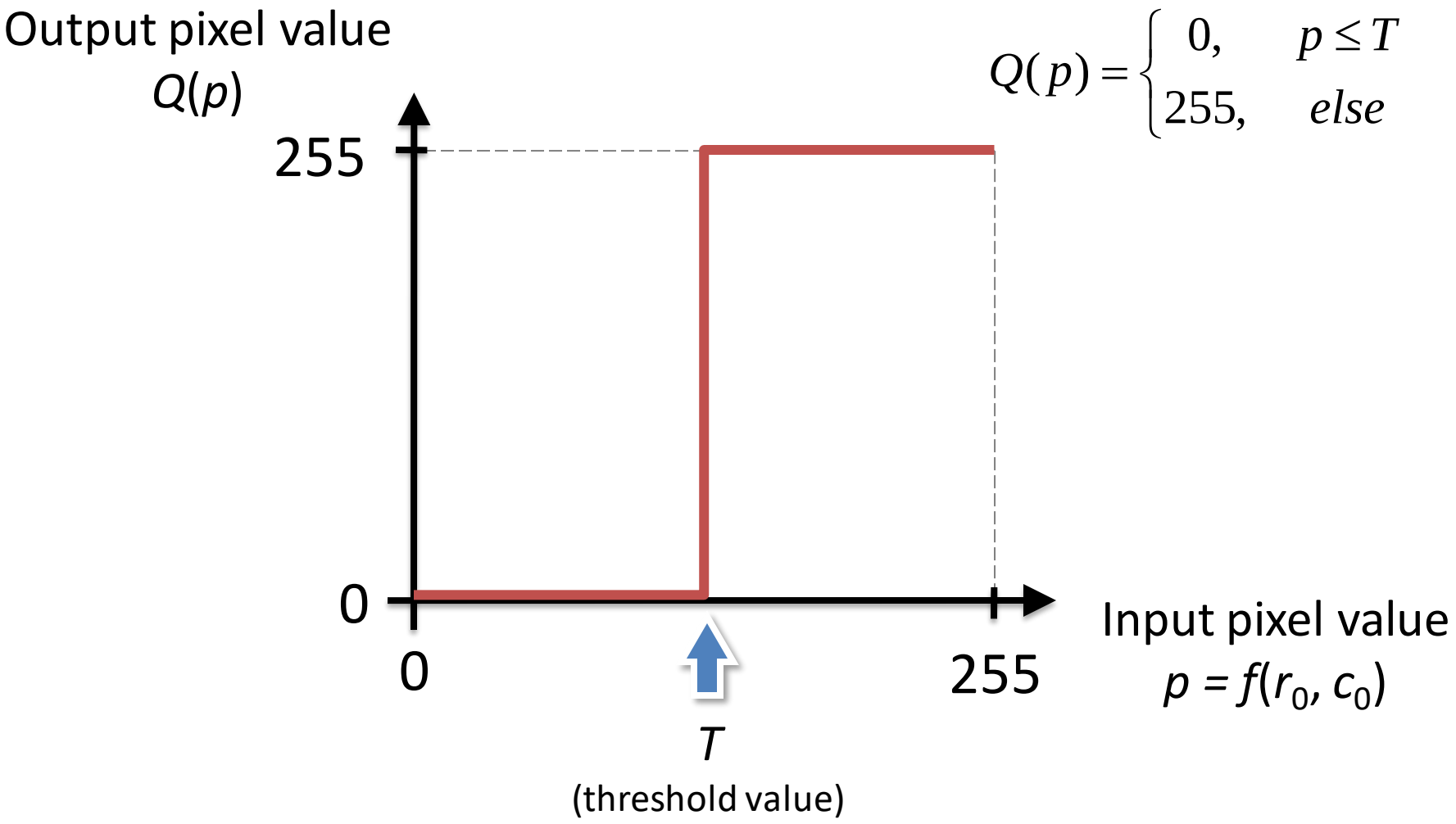


Grayscale quantization in Python

```
import numpy as np
import cv2 as cv
import ipcv_utils.utils as ipcv_plt
import matplotlib.pyplot as plt


img = cv.imread("imgs/jbeans.gry.png", cv.IMREAD_GRAYSCALE)
img = img.astype(np.float32)

N = 2 # 2-bit quantization (2^N = 16 shades of gray)
delta = 256/(2**N)
qnt_img = 255/(2**N - 1) * np.floor(img/delta)

H, k = np.histogram(qnt_img, bins=256, range=(0,256))
plt.plot(k[:-1], H)
plt.show()

ipcv_plt.imshow(img, cmap="gray", vmin=0, vmax=255)
ipcv_plt.imshow(qnt_img, cmap="gray", vmin=0, vmax=255)
```


60000
50000
40000
30000
20000
10000
0
0
50
100
150
200
250

Thresholding: 1-bit Quantization



Thresholding: 1-bit Quantization

Original 8-bpp image

Result of thresholding with $T = 210$

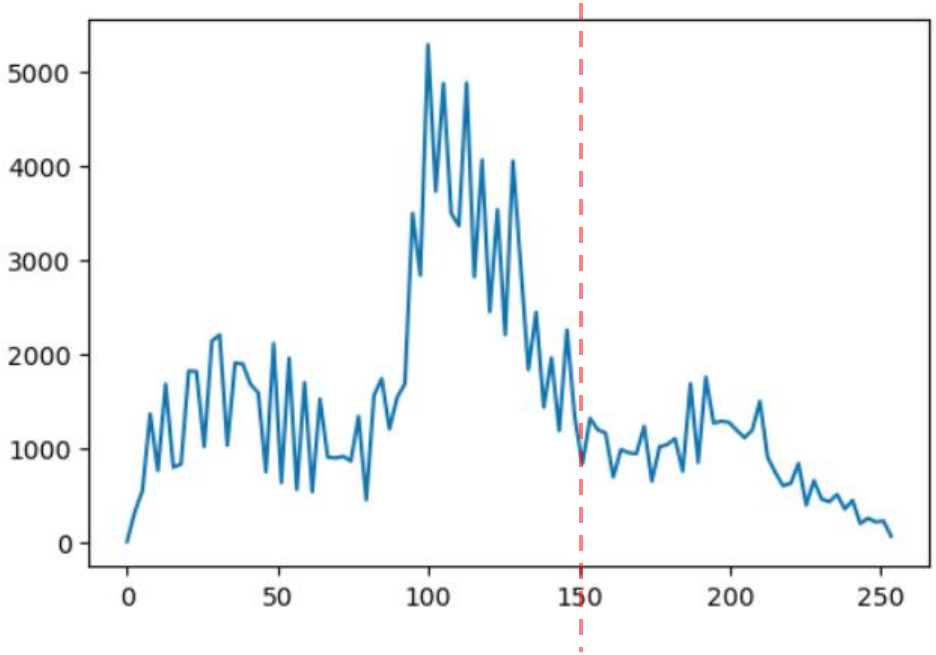
Thresholding: 1-bit Quantization

Original 8-bpp image

Result of thresholding with $T = 128$

Image histograms and thresholding in Python

```
import numpy as np
import cv2 as cv
import ipcv_utils.utils as ipcv_plt
import matplotlib.pyplot as plt


img = cv.imread("imgs/jbeans.gry.png", cv.IMREAD_GRAYSCALE)

H, k = np.histogram(img, bins=100, range=(0,256))
plt.plot(k[:-1], H)
plt.show()

T = 150
(_, thr_img) = cv.threshold(img, T, 255, cv.THRESH_BINARY)

ipcv_plt.imshow(img, cmap="gray", vmin=0, vmax=255)
ipcv_plt.imshow(thr_img, cmap="gray", vmin=0, vmax=255)
```



T = 150

Today: Basic Color Segmentation, Part 2

1. Grayscale thresholding
2. How to automatically find the threshold(s)
   - Otsu's method
   - Gaussian mixture model (GMM)

Otsu's method

- Method to automatically compute the "best" threshold for an image.
- Divides the normalized histogram $p(i)$ into two groups (foreground and background) based on a chosen T
- Searches for the T that maximizes the between-group variance

$$\sigma^2_{\text{Within}}(T) = n_B(T)\sigma^2_B(T) + n_O(T)\sigma^2_O(T)$$

$$\begin{aligned}\sigma^2_{\text{Between}}(T) &= \sigma^2 - \sigma^2_{\text{Within}}(T) \\ &= n_B(T)\left[\mu_B(T) - \mu\right]^2 + n_O(T)\left[\mu_O(T) - \mu\right]^2 \\ &= n_B(T)n_O(T)[\mu_B(T) - \mu_O(T)]^2\end{aligned}$$

$$\begin{aligned}n_B(T) &= \sum_{i=0}^{T-1} p(i) \\ n_O(T) &= \sum_{i=T}^{N-1} p(i) \\ \sigma^2_B(T) &= \text{the variance of the pixels in the background (below threshold)} \\ \sigma^2_O(T) &= \text{the variance of the pixels in the foreground (above threshold)}\end{aligned}$$

Otsu's method thresholding in Python

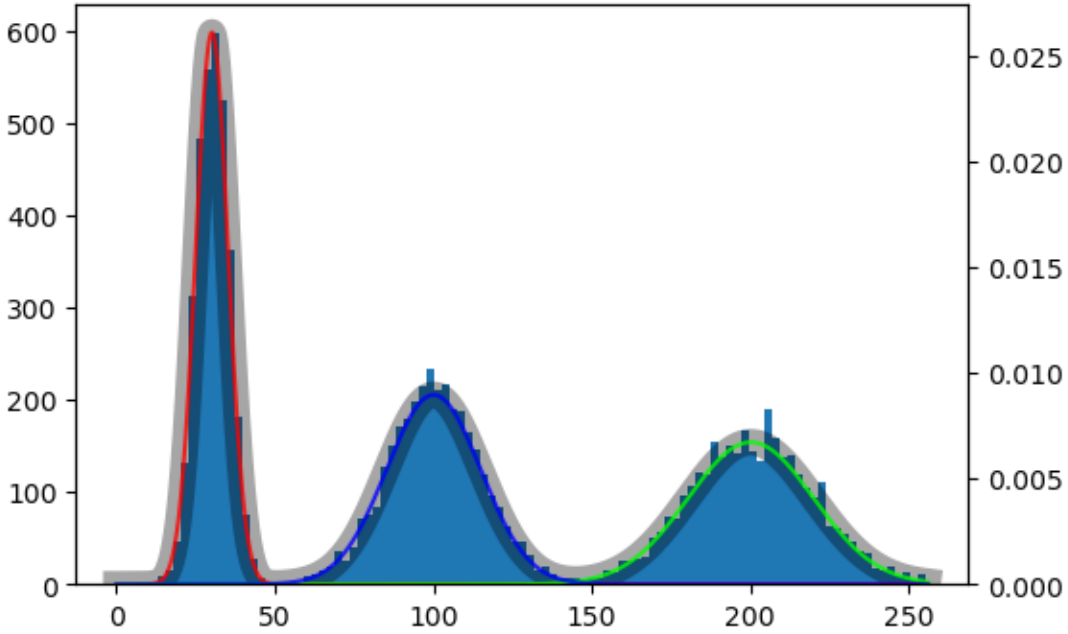
```
import numpy as np
import cv2 as cv
import ipcv_utils.utils as ipcv_plt
import matplotlib.pyplot as plt


img = cv.imread("imgs/jbeans.gry.png",
  cv.IMREAD_GRAYSCALE)

(T, thr_img) = cv.threshold(img, T, 255,
  cv.THRESH_BINARY)
print("T =", T)

ipcv_plt.imshow(img, cmap="gray",
  vmin=0, vmax=255)
ipcv_plt.imshow(thr_img, cmap="gray",
  vmin=0, vmax=255)
```

```
T = 124.0
```

GMM clustering of grayscale (1D) data

```
import numpy as np
import matplotlib.pyplot as plt
import ipcv_utils.utils as ipcv_plt
import cv2 as cv

from sklearn import datasets
from sklearn.mixture import GaussianMixture
from scipy.stats import norm


#%% CREATE THE RANDOM DATA

X, y = datasets.make_blobs(n_samples=10000,
  n_features=1, centers=[(30,), (100,), (200,)],
  cluster_std=[5, 15, 20])
X[X<0] = 0
X[X>255] = 255

img = X.flatten()
img.sort()
img = img.reshape((100, 100))

ipcv_plt.imshow(img, cmap="gray", vmax=255, zoom=5)


#%% FIT AND APPLY THE GMM

fig, ax1 = plt.subplots()
ax1.hist(X, bins=100)

gmm = GaussianMixture(n_components=3, n_init=10)
gmm.fit(X)

print("Weights:\n", gmm.weights_, "\n")
print("Means:\n", gmm.means_, "\n")
print("Variances:\n", gmm.covariances_, "\n")
```

```
x = np.arange(0, 256, 0.001)
g1 = norm.pdf(x, gmm.means_[0],
  np.sqrt(gmm.covariances_[0]))
g2 = norm.pdf(x, gmm.means_[1],
  np.sqrt(gmm.covariances_[1]))
g3 = norm.pdf(x, gmm.means_[2],
  np.sqrt(gmm.covariances_[2]))
g1 = g1.transpose()
g2 = g2.transpose()
g3 = g3.transpose()

p = gmm.weights_[0]*g1 + gmm.weights_[1]*g2 + \
  gmm.weights_[2]*g3

ax2 = ax1.twinx()
ax2.plot(x, p, color=(0,0,0,0.35), linewidth=10)

ax2.set_ylim([0, None])
ax2.plot(x, gmm.weights_[0]*g1, color=(1,0,0,0.8))
ax2.plot(x, gmm.weights_[1]*g2, color=(0,1,0,0.8))
ax2.plot(x, gmm.weights_[2]*g3, color=(0,0,1,0.8))
plt.show()

labels = gmm.predict(img.reshape(-1, 1))
centers = gmm.means_

seg_vals = centers[labels]
seg_img = seg_vals.reshape(img.shape)
ipcv_plt.imshow(seg_img, cmap="gray", vmin=0,
  vmax=255, zoom=5)
```

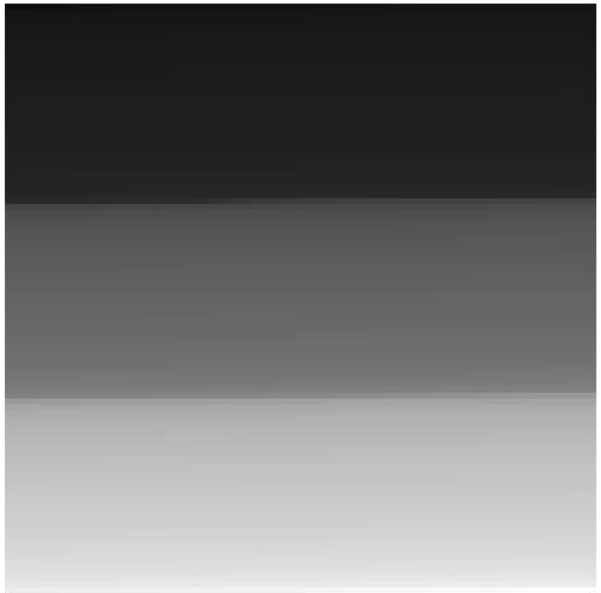


Histogram and GMM fit

(each cluster is a separate colored Gaussian line)

```
Weights:
 [0.3334398  0.33327624 0.33328396]

Means:
 [[ 30.09856968]
 [200.13639225]
 [100.04261499]]

Variances:
 [[[ 25.996352  ]]
 [[391.94360485]]
 [[220.47759781]]]
```

These could be used as thresholds



Original data (reshaped to image)

Segmented via GMM

GMM applied to jellybeans image

```python
import matplotlib.pyplot as plt
import ipcv_utils.utils as ipcv_plt
import cv2 as cv
from sklearn.mixture import GaussianMixture
from scipy.stats import norm

img = cv.imread("imgs/jbeans.gry.png",
  cv.IMREAD_GRAYSCALE)
img = cv.resize(img, (256, 256))
ipcv_plt.imshow(img, cmap="gray", vmax=255, zoom=1.5)

X = img.reshape(-1, 1)

fig, ax1 = plt.subplots()
ax1.hist(X, bins=100)

gmm = GaussianMixture(n_components=3, n_init=10)
gmm.fit(X)

x = np.arange(0, 256, 0.001)
g1 = norm.pdf(x, gmm.means_[0],
  np.sqrt(gmm.covariances_[0]))
g2 = norm.pdf(x, gmm.means_[1],
  np.sqrt(gmm.covariances_[1]))
g3 = norm.pdf(x, gmm.means_[2],
  np.sqrt(gmm.covariances_[2]))
g1 = g1.transpose()
g2 = g2.transpose()
g3 = g3.transpose()

p = gmm.weights_[0]*g1 + gmm.weights_[1]*g2 + gmm.weights_[2]*g3

ax2 = ax1.twinx()
ax2.plot(x, p, color=(0,0,0,0.35), linewidth=10)

ax2.set_ylim([0, None])
ax2.plot(x, gmm.weights_[0]*g1, color=(1,0,0,0.8))
ax2.plot(x, gmm.weights_[1]*g2, color=(0,1,0,0.8))
ax2.plot(x, gmm.weights_[2]*g3, color=(0,0,1,0.8))
plt.show()

labels = gmm.predict(img.reshape(-1, 1))

seg_vals = centers[labels]
seg_img = seg_vals.reshape(img.shape)
ipcv_plt.imshow(seg_img, cmap="gray", vmin=0, vmax=255, zoom=1.5)
```