Image Processing: Understanding and Transforming Histograms

Dr. Samuel Smith

October 2, 2023

Contents

1 Image Histograms

1.1 Definition

A histogram is a graphical representation that organizes a group of data points into user-specified ranges. In image processing, a histogram often represents the distribution of pixel intensities.

1.2 Purpose and Usefulness

Histograms provide insights into the contrast, brightness, and intensity distribution of an image. They are widely used in image analysis, enhancement, thresholding, and color correction.

1.3 Applications

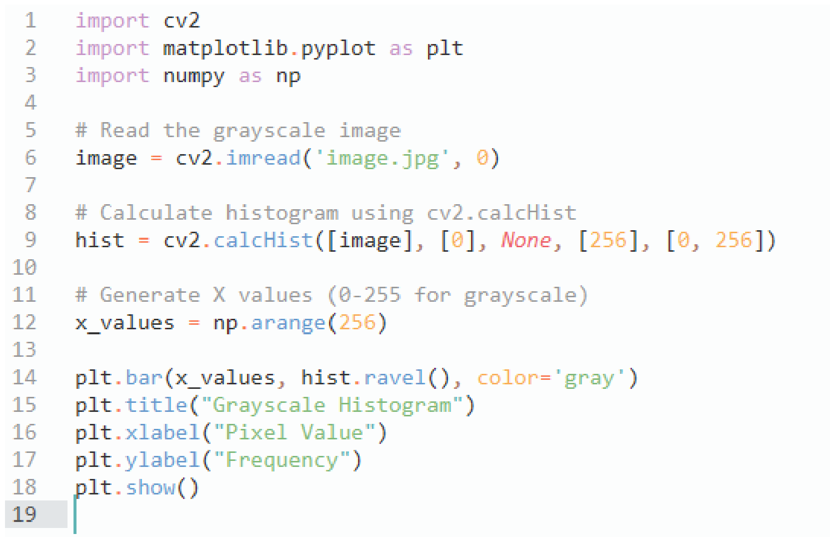
- Medical Imaging
- Satellite Image Processing
- Computer Vision
- Photography

1.4 Parameters and Characteristics

- **Bins**: The intervals you define to group the data points.
- **Range**: The range of pixel values you're interested in.

1.5 Example using Built-in Function (Grayscale)

The following code allows you to calculate the histogram for a grayscale image using the calcHist function within the open cv module. Run the following code with an image of your choice to show the grayscale histogram.

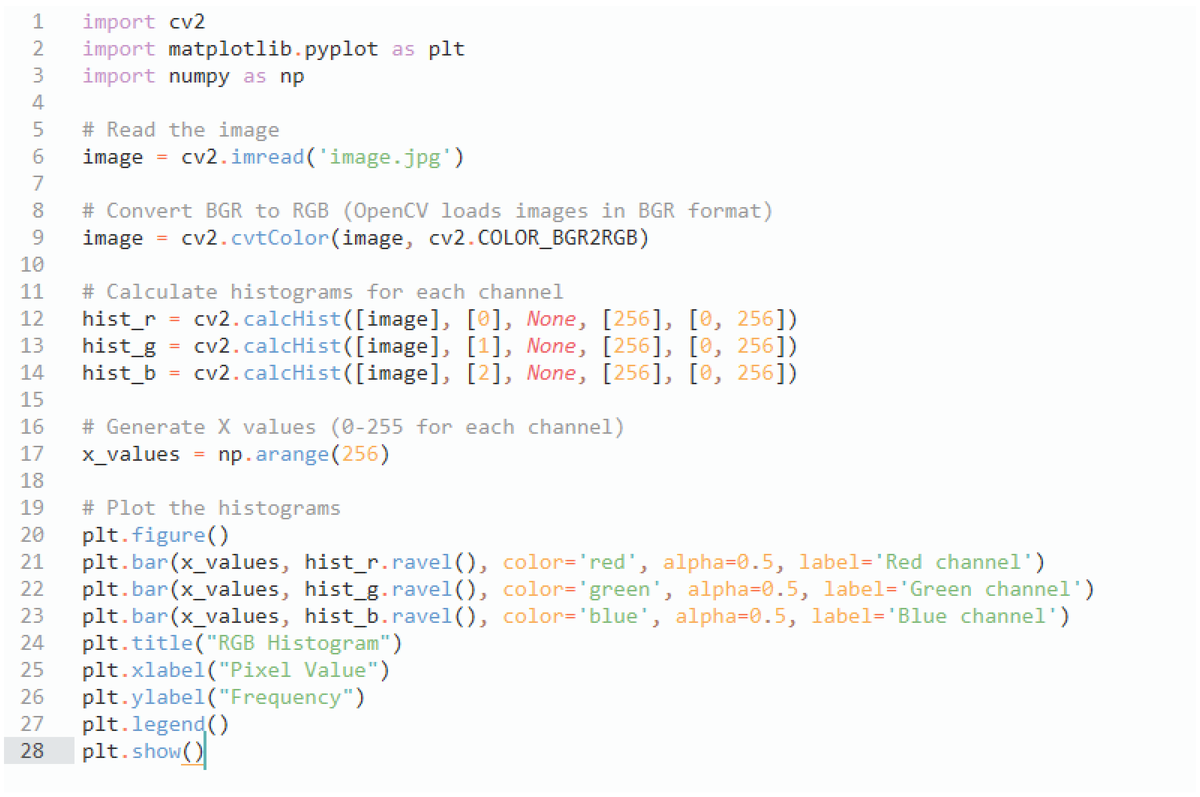


```python
import cv2
import matplotlib.pyplot as plt
import numpy as np

# Read the grayscale image
image = cv2.imread('image.jpg', 0)

# Calculate histogram using cv2.calcHist
hist = cv2.calcHist([image], [0], None, [256], [0, 256])

# Generate X values (0-255 for grayscale)
x_values = np.arange(256)

plt.bar(x_values, hist.ravel(), color='gray')
plt.title("Grayscale Histogram")
plt.xlabel("Pixel Value")
plt.ylabel("Frequency")
plt.show()

```

Figure 1: Histogram built using built-in functions

1.6 Example using Built-in Function (RGB)

Often in photography when you are editing an image, a histogram for each of the separate Red, Green and Blue channels is useful. The following code shows how to create a histogram of the colour channels. Run this code with an image of your choice to view the RGB histogram.



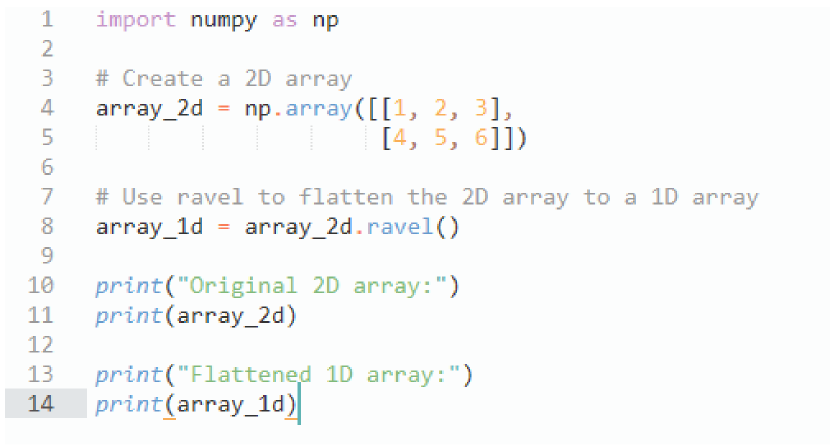
```python
import cv2
import matplotlib.pyplot as plt
import numpy as np

# Read the image
image = cv2.imread('image.jpg')

# Convert BGR to RGB (OpenCV loads images in BGR format)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Calculate histograms for each channel
hist_r = cv2.calcHist([image], [0], None, [256], [0, 256])
hist_g = cv2.calcHist([image], [1], None, [256], [0, 256])
hist_b = cv2.calcHist([image], [2], None, [256], [0, 256])

# Generate X values (0-255 for each channel)
x_values = np.arange(256)

# Plot the histograms
plt.figure()
plt.bar(x_values, hist_r.ravel(), color='red', alpha=0.5, label='Red channel')
plt.bar(x_values, hist_g.ravel(), color='green', alpha=0.5, label='Green channel')
plt.bar(x_values, hist_b.ravel(), color='blue', alpha=0.5, label='Blue channel')
plt.title("RGB Histogram")
plt.xlabel("Pixel Value")
plt.ylabel("Frequency")
plt.legend()
plt.show()
```

Figure 2: RGB Histogram built using built-in functions

When viewing the RGB histogram, it may be unclear if each of the channels are overlayed.
Create a 2x2 subplot that shows a histogram for each of the colour channels and the grayscale histogram.

1.7 DIY Histogram Function

The .ravel() method in Python is commonly used with NumPy arrays to return a contiguous, flattened 1D array. In other words, it takes a multi-dimensional array and ”flattens” it into a one-dimensional array. This makes the image straightforward to loop through all the pixel values, since the order of the pixels is unimportant for a histogram.

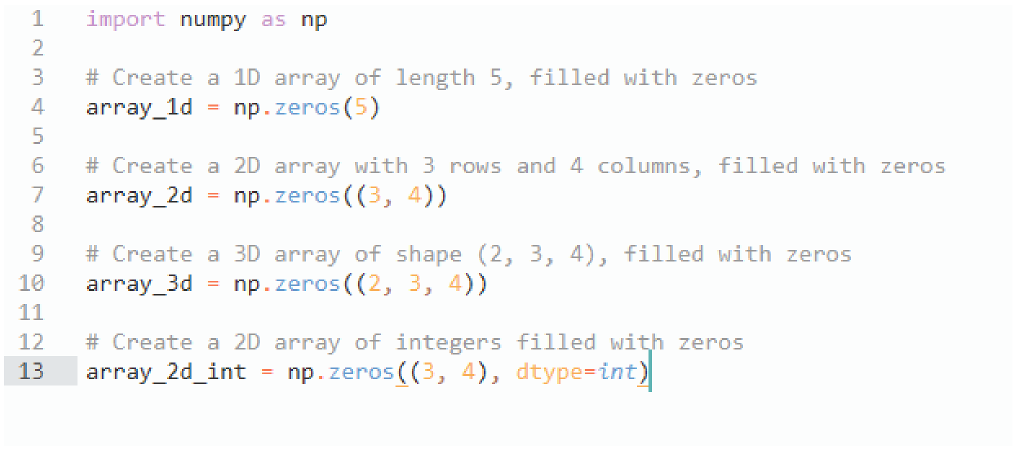


```python
import numpy as np

# Create a 2D array
array_2d = np.array([[1, 2, 3],
                     [4, 5, 6]])

# Use ravel to flatten the 2D array to a 1D array
array_1d = array_2d.ravel()

print("Original 2D array:")
print(array_2d)

print("Flattened 1D array:")
print(array_1d)
```

```
Original 2D array:
[[1 2 3]
 [4 5 6]]

Flattened 1D array:
[1 2 3 4 5 6]
```

Figure 3: .ravel()

The numpy.zeros function in Python’s NumPy library is used to create a new array filled with zeros. The shape and data type of the array are specified as arguments. Here’s the basic syntax:

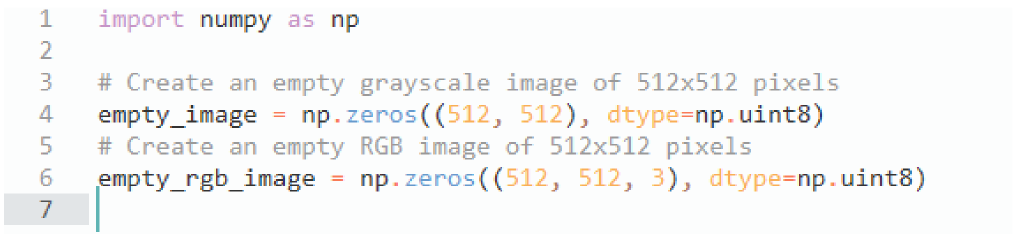


```python
import numpy as np

# Create a 1D array of length 5, filled with zeros
array_1d = np.zeros(5)

# Create a 2D array with 3 rows and 4 columns, filled with zeros
array_2d = np.zeros((3, 4))

# Create a 3D array of shape (2, 3, 4), filled with zeros
array_3d = np.zeros((2, 3, 4))

# Create a 2D array of integers filled with zeros
array_2d_int = np.zeros((3, 4), dtype=int)
```

```
array_1d: [0., 0., 0., 0., 0.]

array_2d:
[[0., 0., 0., 0.],
 [0., 0., 0., 0.],
 [0., 0., 0., 0.]]

array_3d:
[[[0., 0., 0., 0.],
  [0., 0., 0., 0.],
  [0., 0., 0., 0.]],
 [[0., 0., 0., 0.],
  [0., 0., 0., 0.],
  [0., 0., 0., 0.]]]

array_2d_int:
[[0, 0, 0, 0],
 [0, 0, 0, 0],
 [0, 0, 0, 0]]
```

Figure 4: .zeros()

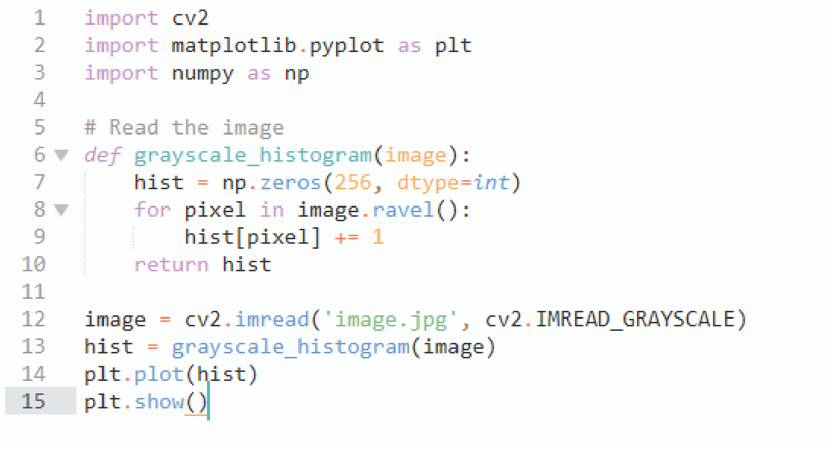
In the context of image processing, numpy.zeros is often used to create empty images or masks of a specific size. For example, you might use it to create an empty grayscale image of 512x512 pixels:



```python
import numpy as np

# Create an empty grayscale image of 512x512 pixels
empty_image = np.zeros((512, 512), dtype=np.uint8)
# Create an empty RGB image of 512x512 pixels
empty_rgb_image = np.zeros((512, 512, 3), dtype=np.uint8)

```

Figure 5: .zeros() for image processing

In order to see how a histogram is generated, take a look at the following function:



```python
import cv2
import matplotlib.pyplot as plt
import numpy as np

# Read the image
def grayscale_histogram(image):
    hist = np.zeros(256, dtype=int)
    for pixel in image.ravel():
        hist[pixel] += 1
    return hist

image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
hist = grayscale_histogram(image)
plt.plot(hist)
plt.show()
```

Figure 6: DIY histogram

Modify this code to dispay the hist data with a bar plot instead of a line plot.

Optional Task: Create your own RGB histogram function

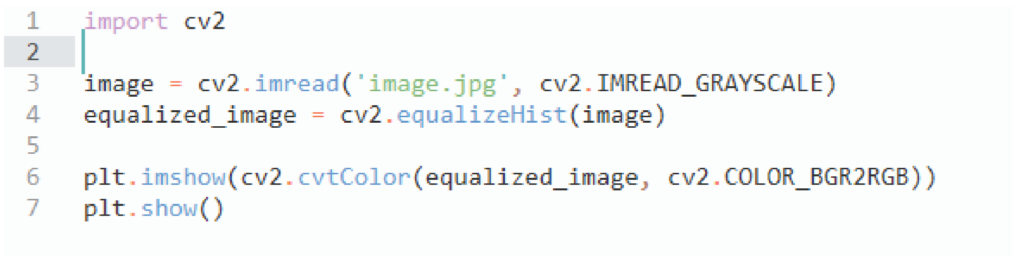
2 Histogram Equalisation

2.1 Theory and Formula

Histogram equalisation is a technique to stretch the contrast of an image, making it clearer. It aims to redistribute the pixel intensities so that they span the whole range.

$$s = T(r) = (L-1)\int_0^r p_r(w)dw$$

2.2 Example using Built-in Function



```
import cv2

image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
equalized_image = cv2.equalizeHist(image)

plt.imshow(cv2.cvtColor(equalized_image, cv2.COLOR_BGR2RGB))
plt.show()
```

Figure 7: Histagram Equalisation

Using a 1x2 subplot, display the low contrast “pout.tiff” image next to the corresponding grayscale histogram.

Apply histogram equalisation using the method provided, and display the new output and new histogram on another 1 x 2 subplot and observe the differences.

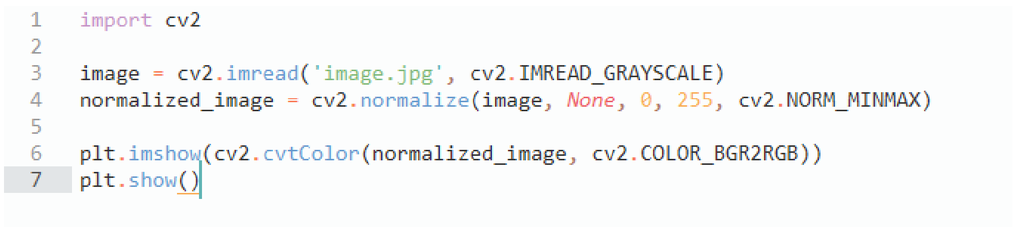
3 Histogram Normalisation

3.1 Theory and Formula

In histogram normalization, we aim to stretch or shrink the histogram so that it lies within a certain range [a, b].

$$s = T(r) = a + (I - I_{\min})\frac{(b - a)}{I_{\max} - I_{\min}}$$

3.2 Example using Built-in Function



```python
import cv2

image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
normalized_image = cv2.normalize(image, None, 0, 255, cv2.NORM_MINMAX)

plt.imshow(cv2.cvtColor(normalized_image, cv2.COLOR_BGR2RGB))
plt.show()
```

Figure 8: HistagramNormalisation

Using the formula provided create a historgram normalisation function:

```python
def histogram_normalisation(image, a, b):
    # Implement using the formula
    pass
```

4 Choosing Between Equalisation and Normalisation

4.1 Advantages and Disadvantages

- **Equalisation**
  - **Advantages**: Better for images with low contrast.
  - **Disadvantages**: May enhance noise.
- **Normalisation**
  - **Advantages**: Simple and easy to control.
  - **Disadvantages**: May not be effective for all images.

5 Extra Coding Questions

1. Implement a function to calculate the histogram of an image using 128 bins instead of 256.
2. **(Advanced Task)** Using the formula provided create a historgram equalisation function:

```python
def histogram_equalisation(image):
    # Implement using the formula
    pass
```

   In a DIY (Do-It-Yourself) histogram equalization function for grayscale images, you'd typically follow these steps:

   (a) Calculate Histogram: First, you count the occurrences of each pixel intensity value in the image.

   (b) Calculate Cumulative Distribution Function (CDF): Sum up the histogram values cumulatively. (.cumsum())

   (c) Normalize CDF: Make the CDF values to be in the range [0, 255] for an 8-bit image.

   (d) Map to New Pixel Values: Replace each pixel in the image with its new, equalized value based on the normalized CDF.