EST 1892 LSBU

LONDON SOUTH BANK UNIVERSITY
COMPUTER VISION AND PATTERN RECOGNITION
A DROP-SESSION
Hannah Oh • March 2025

1. IMAGE ANALYSIS WITH HISTOGRAM

This week, we will explore how to perform image analysis using **histograms**, with a focus on the Python libraries *scikit-image* and *Matplotlib*. Also, we will revisit the materials you've already covered from previous weeks. Please take some time to understand these concepts.

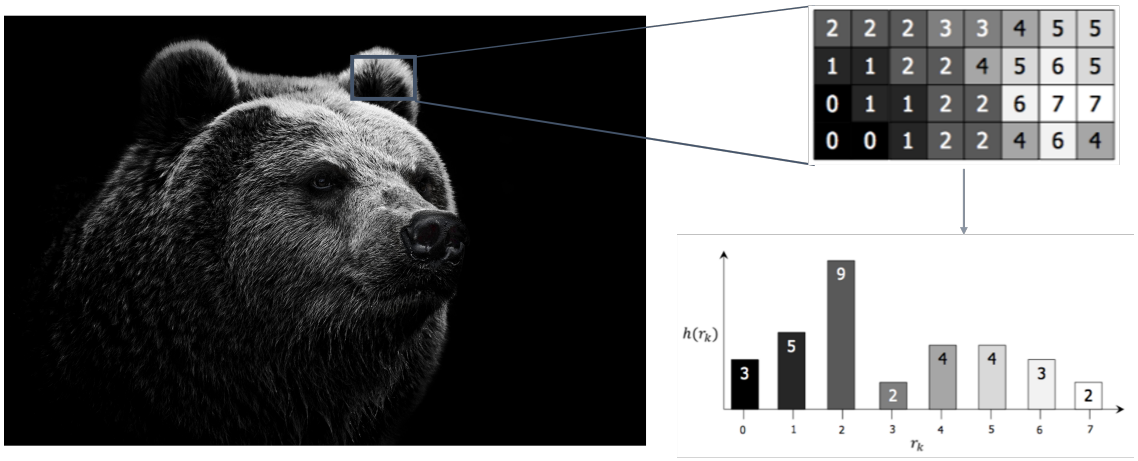
(a) Grayscale image

(b) Colour image

Figure 1: Sample Images for this week

- `Histograms`
  - Images are composed of pixels, each with an intensity value defining its colour.
  - Histograms show the frequency distribution of these pixel intensities - the Figure 2.
  - Histograms provide a global description of the contents of an image and they summarise how much of which colour is in the image.
  - Histograms throw away all of the spatial relationships, i.e. patterns and textures
  - An image histogram is the graphical representation of this distribution.
  - In image processing, histograms allow for the analysis of image brightness, contrast, dynamic range, and saturation.
    * `h(i)` = the number of pixels in `I`(image) with the intensity value `i`.
    * For example, if `i=0`, the `h(0)` is the number of pixels with a value of `0`.
- `Application of Histograms in Image Analysis`
  - **Image Enhancement**: Histograms can be used to improve the visual quality of an mage by adjusting its contrast and brightness, Techniques like histogram equalisation and contrast stretching use histograms.
  - **Image Thresholding**: Histograms can help in segmenting an image by identifying intensity levels that separate different objects or regions.

– **Image Analysis**: Histograms provide statistical information about the distribution of pixel intensities, which can be used to for various image analysis tasks such as comparing images or identifying features.



$$h(i_k) = n_k$$

- i_k *is the* k^{th} *intensity value*
- n_k *is the number of pixels in the image with intensity* i_k

Figure 2: Understand how the histogram represents the image

1.1 Explore the grayscale image - Figure 1-(a)

```
# import the io module from the skimage library
from skimage import io

# import the pyplot module from the matplotlib library
import matplotlib.pyplot as plt
```

Example 1: Import libraries

1. **Intro - Python packages**
   To proceed with this session, ensure the following Python packages are imported: Matplotlib and Scikit-image please check the Example 1.
   a) **`from skimage import io`**: A powerful Python library for image processing and the `skimage.io` module is used for reading, saving, and displaying images.
   b) **`Matplotlib(matplotlib.pyplot)`**: A library designed for creating static, interactive, and animated visualizations in Python and the `matplotlib.pyplot` module provides functions for plotting histogram.

```
# Load the grayscale image and display it
image = io.imread('apple gray.jpeg')
plt.imshow(image)
plt.show()
# Generate the grayscale image's histogram
ax = plt.hist(image.ravel(), bins=256)
plt.title("Apple Gray")
plt.xlabel("Intensity value")
plt.ylabel("Count")
plt.show()
```

Example 2: Grayscale Image Display and Initial Histogram

2. **Grayscale imageGet your image data ready and Remove spatial dimensions**
   We need to remove spatial dimensions in image processing, essentially converting a multi-dimensional image array into a 1D array. **Why Remove Spatial Dimensions?**
   - **Feature Extraction**: Many machine learning and image processing algorithms require input data to be in a 1D vector format. For example, when feeding image data into a fully connected neural network, you need to flatten the image[1].
   - **Data Analysis**: Flattening can be useful for certain statistical analyses or when you need to process pixel data sequentially.
   - **Simplified Processing**: Some image processing operations might be easier to implement or faster if you work with a 1D array.

   i) **Line 2-4**: For a grayscale image, this array will be 2-dimensional (height x width), where each element represents the brightness of a pixel.

   (a) ***Remind*** Image Representation as Arrays. You should understand that digital images are fundamentally numerical data arranged in arrays. For grayscale, it's a **2D array** representing pixel brightness.

Figure 3: Output- Line 2-4

ii) **`ax = plt.hist(image.ravel(), bins=256)`**

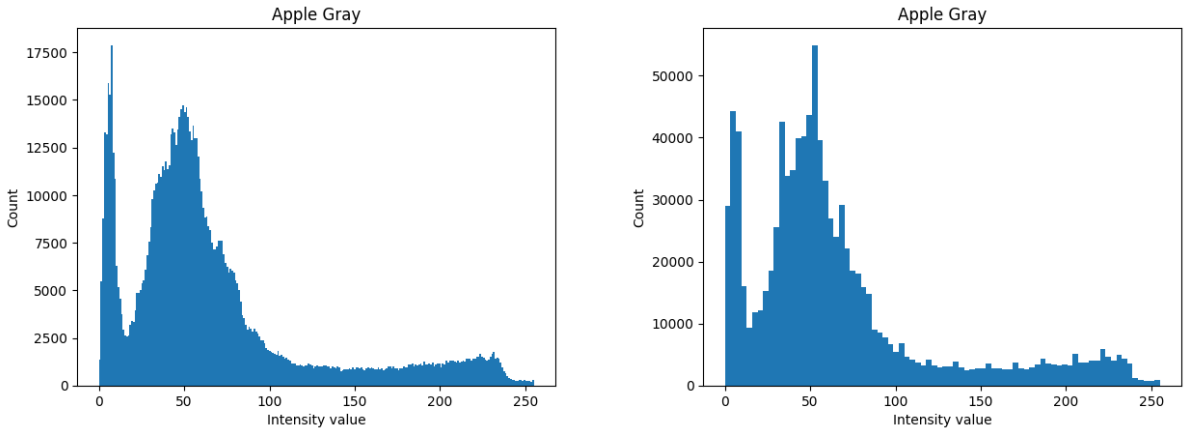
(a) `plt.hist()`: This function calculates the histogram.

- The first argument `image.ravel()` is the 1D array of pixel intensities.
- The `bins=256` argument specifies that the intensity range (0-255 for a typical 8-bit grayscale image) should be divided into 256 intervals (bins).
- The function counts how many pixels fall into each bin.

(b) `image.ravel()`: The **`.ravel()`** method flattens the 2D NumPy array image into a 1-dimensional array.

- This is necessary because the **`plt.hist()`** function expects 1D data.
- The flattened array contains all the pixel intensity values in a single sequence.

(c) The return value of '**`plt.hist()`** is a tuple containing the bin counts, bin edges, and other information.



(a) Output: 256 bins **(b)** Output: 80 bins

Figure 4: Different numbers of bin

(d) The Figure 4 demonstrates the impact of different bin counts on the resulting output.

iii) **!Important!** You should understand why the image array needs to be flattened (using `.ravel()`') before calculating the histogram. The histogram function expects a 1D sequence of data, not a 2D image representation.

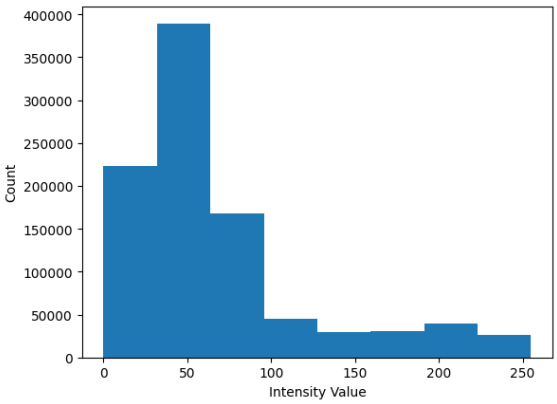


Figure 5: The number of bins = 8

Notes:

- Explore matplotlib to plot the histogram of a image plot the histogram.
- **What's difference between `.ravel()` and `.flatten()` functions?**
  The main difference between **NumPy**'s `.ravel()` and `.flatten()` functions lies in how they handle memory and whether they return a copy or a view of the original array.
  - **`.ravel()`**: Fast, but potentially risky(if you modify the output. That is, if performance is a priority and you are cautious about potential side effect, then use this function.
  - **`.flatten()`**: Safe, but potentially slower. That is, if you need to ensure that the original array remains unchanged, then use it. click
  - You can find more info here.
- **What's histogram binning '`bin =`'**?
  - Usually, the range of intensity values of images is from [0–255] in 8bits representation 2^8. But images can be also represented using 2^{16}, 2^{32} bits and so on.
  - In such cases the intensity range is high and it is hard to represent each intensity value in a histogram.
  - We can quantise the range into several buckets. For example, if we quantize 0-255 into 8 bins(**`plt.hist(image.ravel(), bins=8)`**), here our bins will be 0-31, 32-63, 64-95, 96-127, 128-159, 160-191, 192-223, 224-255. The output looks like the Figure 5.
  - We need to find a way to put each intensity value into the appropriate bins.
    * **`k = 256`**: Number of possible integer values in 8 bit representation
    * **`b = 8`**: Number of bins
    * **`j = floor((h(i)*b)/k)`**
      · `j` is the bin number of the intensity value at position i.
      · In image processing, `h(i)` represent the intensity of the i^{th} pixel, or the value of the i^{th} element of an array.

1.2 Explore the colour image - Figure 1-(b)

```
1  # Load the colour image and display it
2  image = io.imread('flower.jpeg')
3  plt.imshow(image)
4  plt.show()
5
6  # Generate the colour image's histogram
7  _ = plt.hist(image.ravel(), bins=256, color='yellow')
8  _ = plt.hist(image[:, :, 0].ravel(), bins=256, color='red', alpha=0.5)
9  _ = plt.hist(image[:, :, 1].ravel(), bins=256, color='Green', alpha=0.5)
10 _ = plt.hist(image[:, :, 2].ravel(), bins=256, color='Blue', alpha=0.5)
11 _ = plt.xlabel('Intensity Value')
12 _ = plt.ylabel('Count')
13 _ = plt.legend(['Total', 'Red_Channel', 'Green_Channel', 'Blue_Channel'])
14 plt.show()
```

Example 3: Colour Image Display and Initial Histogram

1. **Colour imageGet your image data ready and Remove spatial dimensions**
   Following the Example 3, the combined colour histogram visually displays the intensity count, which is the sum of the **RGB** channel values.

   i) **Line 2-4**: This is similar process like a grayscale image. However, **for a colour image, the array will be 3-dimensional (height x width x 3)**, where the third dimension represents the Red, Green, and Blue colour channels.

Figure 6: Output- Line 2-4

   ii) **Line 7**: `_ = plt.hist(image.ravel(), bins=256, color='yellow')`

   (a) `image.ravel()`: Flattens the 3D colour image array into a 1D array. This combines the pixel intensities from all three colour channels (R, G, B) into a single distribution.

(b) `bins=256`: Calculates the histogram with 256 bins.

(c) `color='yellow'`: Sets the colour of the histogram bars to yellow.

iii) **Line 8**: **`_ = plt.hist(image[:, :, 0].ravel(), bins=256, color='red', alpha=0.5)`**

(a) `plt.hist()`: Calculates the histogram of the red channel intensities.

(b) `image[:, :, 0]`: This is array slicing. It selects all rows and all columns, but *only the first color channel*, which is typically the Red channel. The result is a 2D array representing the red component of the image.

(c) `.ravel()`: Flattens the 2D red channel array into a 1D array.

(d) `bins=256`: Calculates the histogram with 256 bins.

(e) `color='red'`: Sets the colour of the histogram bars to red.

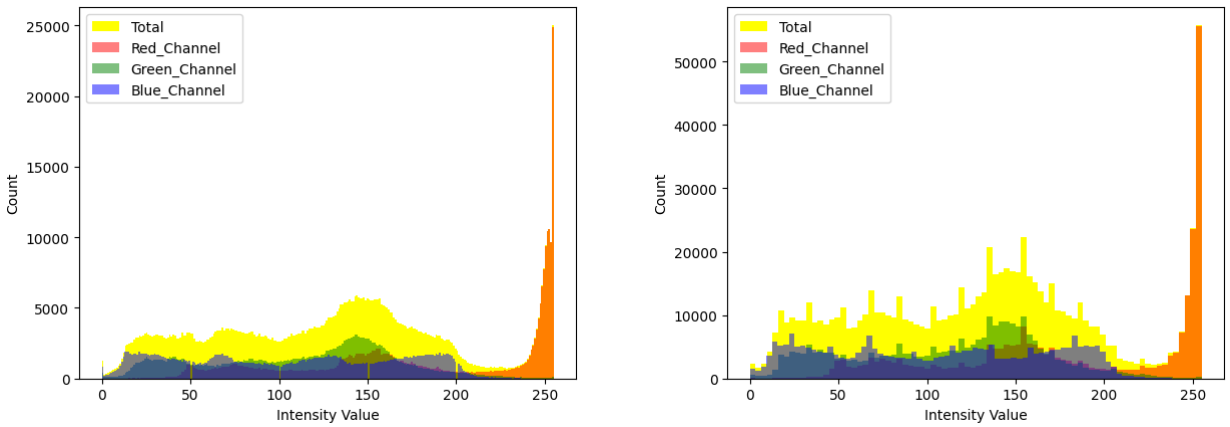
(f) `alpha=0.5`: Sets the transparency (alpha) of the red histogram to 0.5 (semi-transparent).

iv) **Line 9 &10**: These two lines do the same like the line 8, but for the **Green** '`image[:, :, 0`' and **Blue** '`image[:, :, 2]`' channels, setting the colours to 'Green' and 'Blue', respectively, and keeping the alpha at 0.5.

v) **Line 13**: **`plt.legend(['Total', 'Red_Channel', 'Green_Channel', 'Blue_Channel'])`**

(a) `plt.legend()`: This in Matplotlib is a function used to display a legend on a plot. A legend is a **box** that labels the different lines, markers, or other plotted elements, helping you understand which data series corresponds to which visual representation.

vi) The Figure **??** demonstrates how the outputs look different with the Figure 4.



(a) Output: 256 bins **(b)** Output: 80 bins

Figure 7: Colour image's histogram

1.3 Cumulative Histogram

```python
# Import libraries
from skimage import io
import matplotlib.pyplot as plt

# Load the grayscale image and generate the histogram
image = io.imread('apple gray.jpeg')
_ = plt.hist(image.ravel(), bins = 256, cumulative = True)
_ = plt.xlabel('Intensity Value')
_ = plt.ylabel('Count')

plt.show()
```

Example 4: Cumulative Histogram

1. **What is the Cumulative Histogram?**
   The **Cumulative Histogram** is a special histogram that can be derived from the normal histogram. At the fist step in creating both a normal and a cumulative histogram, we find the counts of each intensity value from 0–255. And then add each subsequent counts. This is the key difference between a normal and a cumulative histogram. That is, instead of just keeping the individual counts, you start adding them together.

   `if i = 0 then H(i) = h(0)`
   `else H(i) = H(i-1) + h(0)`

   [Notes]

   - `h(i)`: where ***i*** represents the intensity value (0 to 255).
     - `h(0)` is the count of pixels with intensity 0.
     - `h(1)` is the count of pixels with intensity 1, and so on...
   - `if i = 0 then H(i) = h(0)`
     - `H(i)` represents the cumulative histogram.
     - When ***i*** is 0 (the first intensity value), the cumulative count is simply the count of pixels with intensity 0. That is, the first value of the cumulative histogram is the same as the first value of the normal histogram.
   - `else H(i) = H(i-1) + h(0)`
     - This is the recursive part that builds the cumulative histogram.
     - For any intensity value ***i*** greater than 0;
       * `H(i-1)`: This is the cumulative count for the previous intensity value.
       * `h(i)`: This is the count of pixels with the current intensity value ***i***.
       * `H(i) = H(i-1) + h(i)`: You add the count of the current intensity value to the cumulative count of the previous intensity value. This gives you the cumulative count for the current intensity value.

[Example]:

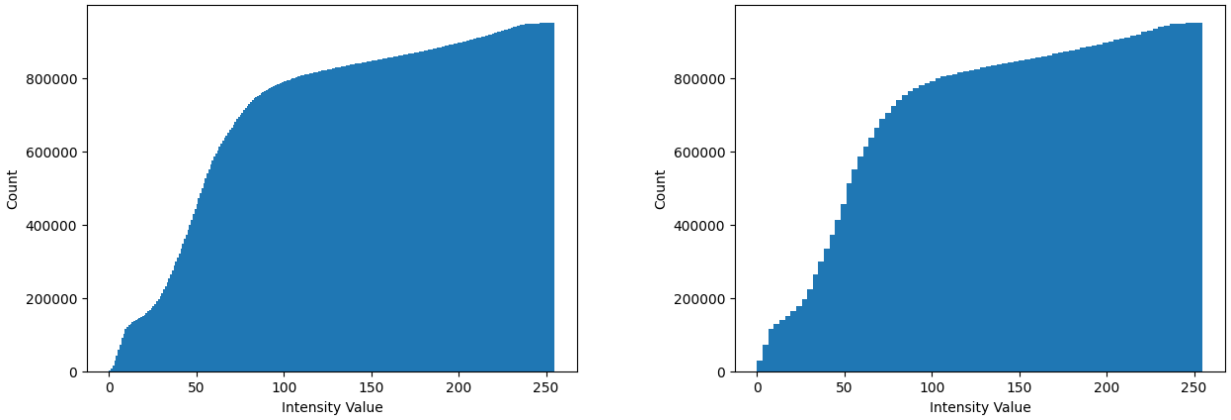
- Let's say you have these counts for the first few intensity values.
  - **`h(0) = 10, h(1) = 5, h(2) = 8, h(3) = 12`**
- Then the cumulative histogram would be:
  - **`H(0) = 10`**
  - **`H(1) = H(0) + h(1) = 10 + 5 = 15`**
  - **`H(2) = H(1) + h(2) = 15 + 8 = 23`**
  - **`H(3) = H(2) + h(3) = 23 + 12 = 35`**

Cumulative histograms are indeed very useful in image processing, primarily because they provide information about the distribution of pixel intensities in a way that's particularly helpful for image enhancement and analysis[2].

In the Example 4, it generates a cumulative histogram.

a) **Line 8**: **`_ = plt.hist(image.ravel(), bins=256, cumulative=True)`**

- `cumulative=True`: This argument to '`plt.hist()`' makes it generate a cumulative histogram. Instead of showing the count of pixels at each intensity, it shows the cumulative count of pixels with intensities less than or equal to that value.
- The Figure 5 demonstrates how the outputs look different with the Figure 3.



(a) Output: 256 bins **(b)** Output: 80 bins

Figure 8: Cumulative histograms with different number of bins

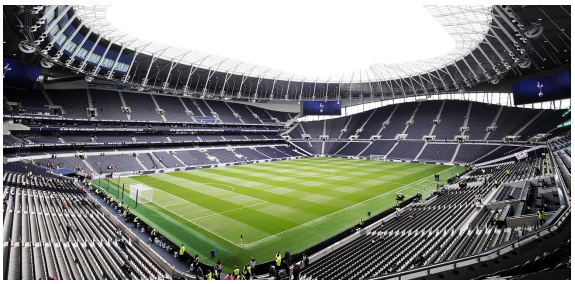
Remark: What's difference between cumulative histograms and histogram binning?

- **Histogram Binning**
  - Histogram binning is the process of dividing the range of data values (e.g., pixel intensities) into a series of intervals, known as "bins.".
  - Then, it counts how many data points fall within each of these bins.
  - The result is a representation of the frequency distribution of the data. In image processing, this shows how often each intensity value (or range of intensity values) occurs in the image.
- **Cumulative Histogram**
  - A cumulative histogram is derived from a regular histogram.
  - Instead of showing the count of values within each individual bin, it shows the cumulative count.
  - In other words, for each bin, it represents the total number of values that fall within that bin and all preceding bins.
- There is a difference in data representation.
  - A standard histogram shows the frequency of values within each bin.
  - A cumulative histogram shows the accumulated frequency up to and including each bin.

2. Try again, Grayscale conversion and Histogram

Let's recap last week's activities and explore histogram creation with a new NumPy function.

(a) London Bigeye



(b) Stadium

(c) Paddington

Figure 9: Sample Images for practice

2.1 Import standard packages, load images, and image histogram

```
%matplotlib inline
from matplotlib import pyplot as plt
import cv2
import numpy as np

img = cv2.imread('./LondonBigeye.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

img_ss = cv2.imread('./the-stadium.jpg')
img_ss = cv2.cvtColor(img_ss, cv2.COLOR_BGR2RGB)

img_pt = cv2.imread('./Paddington.jpg')
img_pt = cv2.cvtColor(img_bb, cv2.COLOR_BGR2RGB)
```

Example 5: Load Colour Image

1. **convert any images we load to RGB from BGR**
   Following the Example 5, we need to convert the images we load to **RGB** channel values.

```
# Convert the image to grayscale by taking the mean of the colour values
# (note that openCV can also do this)
# Make sure we cast the image back to a uint8 after the mean!
img_gray = np.uint8(np.mean(img, axis=-1))

# Remove the spatial dimension from the image
flat_gray_img = img_gray.flatten()

# Create and show a histogram
plt.hist(flat_gray_img)
plt.show()

# Now try making a new histogram, with different numbers of bins.
plt.hist(img_gray.flatten(), bins=50)
plt.title('50 bins')
plt.show()

plt.title('100 bins')
plt.hist(img_gray.flatten(), bins=100)
plt.show()
```

Example 6: Grayscale Conversion and Histogram

2. Grayscale Conversion and Histogram

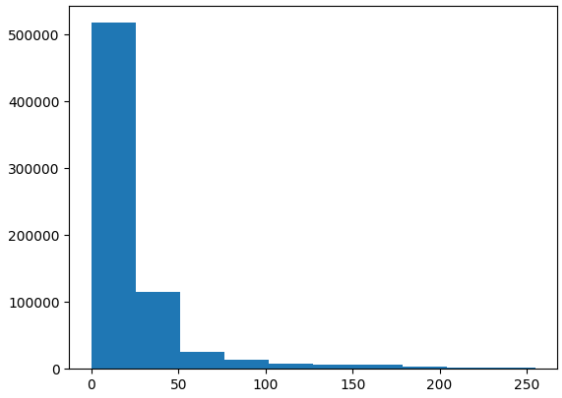
a) **Line 4**: **`img_gray = np.uint8(np.mean(img, axis=-1))`**

- `np.uint8()`: Casts the resulting grayscale values to the uint8 data type (unsigned 8-bit integer), which is the standard data type for pixel intensities (0-255).
- `np.uint8(np.mean(img, axis=-1))`: Calculates the mean along the last axis (axis=-1), which corresponds to the colour channels (R, G, B). This effectively averages the colour values to get a grayscale representation.

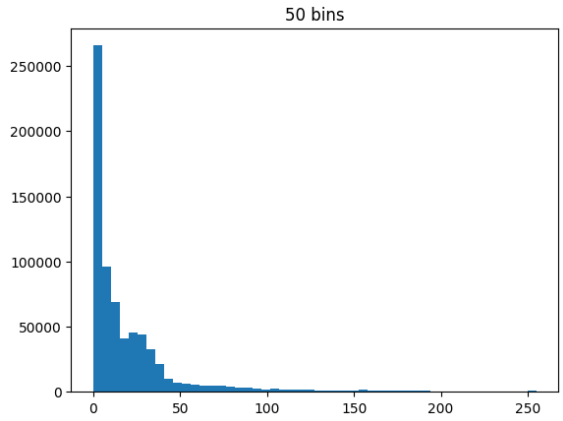
b) **Line 7**: **`img_gray.flatten()`**

- Flattens the grayscale image array into a 1D array using '`.flatten()`'

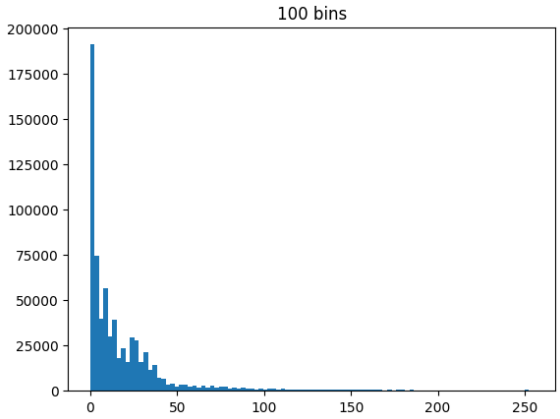
c) You can see outputs on the Figure 10.



(a) Create a histogram



(b) Create a histogram with 50 bins



(c) Create a histogram with 100 bins

Figure 10: Create histograms

```
1 # Histogram equalisation on the grayscale image,
2 # compare the results with the original grayscale image
3 img_gray_eq = cv2.equalizeHist(img_gray)
4 plt.subplot(1,2,1)
5 plt.imshow(img_gray, cmap='gray')
6 plt.subplot(1,2,2)
7 plt.imshow(img_gray_eq, cmap='gray')
8 plt.show()
```

Example 7: Perform histogram equalisation on the grayscale image

3. Histogram Equalization

a) **Line 3**: **`img_gray_eq = cv2.equalizeHist(img_gray)`**

- `cv2.equalizeHist()`: Performs histogram equalisation on the grayscale image using this function.

b) **Line 5 & 7**: **`cmap='gray'`**

- It is used to display the images in grayscale.

c) You can see outputs on the Figure 11.

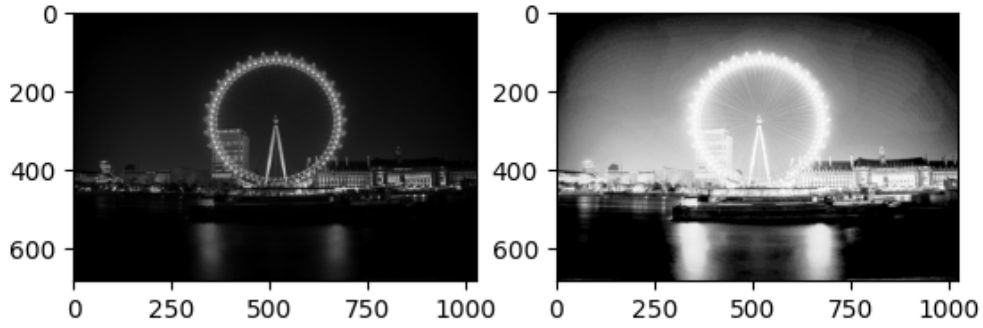


Figure 11: Histogram Equalisation

```
# Reshape the colour image to remove the spatial dimensions
# but keeping the 3 colour channels
reshaped_img = np.reshape(img, (-1, 3))
plt.hist(reshaped_img, bins=10)
plt.show()

# Load two colour images and plot colour histograms next to them.
plt.subplot(2,2,1)
plt.imshow(img_ss)
plt.subplot(2,2,2)
plt.hist(np.reshape(img_ss, (-1, 3)), bins=10)
plt.subplot(2,2,3)
plt.imshow(img_pt)
plt.subplot(2,2,4)
plt.hist(np.reshape(img_pt, (-1, 3)), bins=10)
plt.show()
```

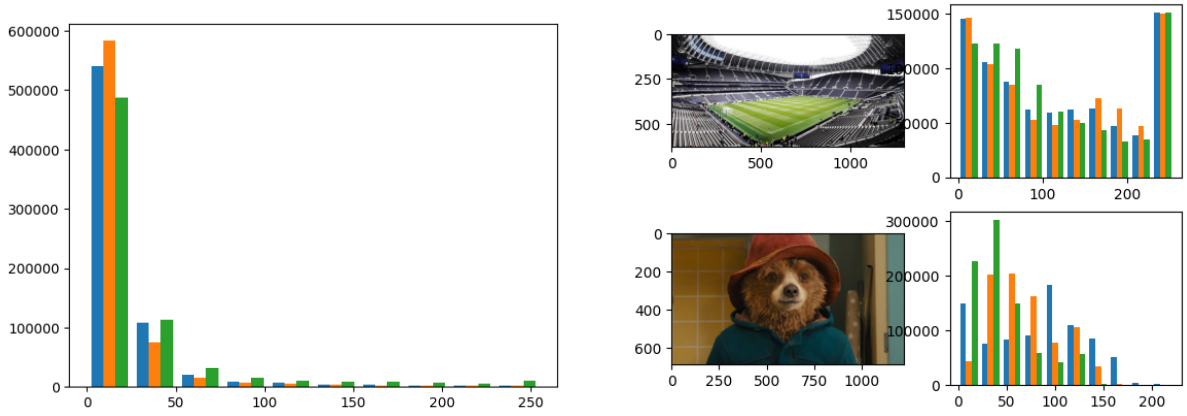
Example 8: Colour histograms

4. Colour histograms

a) **Line 3**: **`reshaped_img = np.reshape(img, (-1, 3))`**

- `np.reshape()`: Using this function to reshapes the colour image. You can know about this function in here.
- We can make histograms of RGB images by changing the shape of the data we pass to `plt.hist`. We still want to remove the spatial dimension but keeep the colours, so the array we pass to hist should have shape [num_of_pixels, 3]. Read the histogram documentation to understand why.
- If you don't know the exact size of a shape dimension you can write `-1` and it will work it out.
- '`(-1, 3)`': Reshapes the array to have an unknown number of rows (`-1`) and 3 columns (corresponding to the R, G, B channels). This effectively flattens the height and width dimensions while keeping the colour channels separate.

b) You can see outputs on the Figure 12. Histograms, in isolation, can be employed as features in certain image classification tasks, for example, to differentiate images of Paddington Bear from those depicting stadiums.



(a) Reshape and Create a histogram with 10 bins

(b) Two colour images and plot their histograms with 10 bins

Figure 12: Histograms as Features

3. 1D Convolutions

Let's try and implement our own simple convolution function! Read this for your understanding.

- **Why do you need to know about convolutions?**
  - Convolutions are a fundamental and powerful operation in signal and image processing.
  - They are employed in nearly every computer vision application.
  - Convolutions are hypothesized to model low-level biological visual processing effectively.
  - Convolutional Neural Networks (CNNs), which leverage convolutions, have driven significant advancements in deep learning across various tasks.
- **What are convolutions' practical uses?**
  - Hand-designed convolutional filters can perform several image processing tasks, including:
    * Edge detection
    * Image denoising
    * Edge enhancement/sharpening
    * Object localization
- **A more thorough comprehension of convolutions is required....**
  - Convolution involves applying a local function across neighborhoods of an image.
  - This differs from pixel-wise operations by considering groups of adjacent pixels.
  - The application of this function, often visualised as a "sliding window," processes small image regions at a time.

– Convolutions are inherently parallelisable, contributing to their efficiency and widespread use.

3.1 Import standard packages and 1D signal from Image Row

```
1 %matplotlib inline
2 from matplotlib import pyplot as plt
3 import cv2
4 import numpy as np
5
6 # Take a row of the image to use as array
7 y = img_gray[200, :]
8
9 # If we use range it counts from 0 to the length of y,
10 # and we can see our 1D signal
11 plt.plot(range(len(y)), y)
12 plt.show()
```

Example 9: Take a row of the image

1. **Image as a Signal**
   Following the Example 9, we take a row for the image to use as data.
   a) **Line 7**: **`y = img_gray[200, :]`**
   - '`img_gray[200, :]`': Extracts the 200th row of the grayscale image img_gray and assigns it to the variable y.
   - This row represents the pixel intensities along that horizontal line.

   b) **Line 11**: **`plt.plot(range(len(y)), y)`**
   - Plots the pixel intensities in '`y`' against their index using '`plt.plot()`'.
   - '`(range(len(y))`': It generates a sequence of numbers from 0 to the length of the row.

   c) On the Figure 13, a line plot showing the pixel intensity values along the 200th row of the grayscale image.

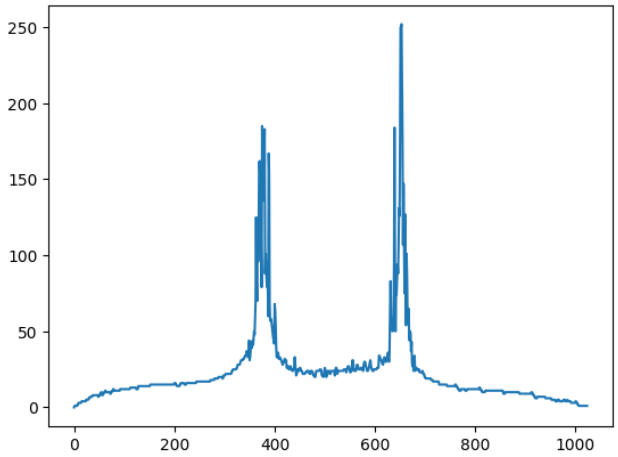


Figure 13: 1D Signal

```
1  # Write a convolution function
2  def convolve1D(data,kernel):
3    # output array of this function will look similar to
4    # scipy.signal.convolve with `mode='valid'`
5    output = []
6    template = kernel[::-1] # flipping the kernel filter to a template
7
8    for i in range(0,len(data)-len(template)+1):
9      res_i = 0
10     for j in range(len(template)):
11       res_i += data[i+j] * template[j]
12     output.append(res_i)
13
14   return np.array(output)
```

Example 10: 1D Convolution Function and Application

2. 1D Convolution

a) **Example 10**: **`def convolve1D(data,kernel):`**

- Defines a function `convolve1D()` that performs 1D convolution.
- Convolution is a fundamental operation in signal and image processing. It involves sliding a kernel (a small array of values)[1] over the input data and calculating a weighted sum.
- The function takes two arguments: '`data`'(the input 1D signal) and '`kernel`'(the convolution kernel).

b) **Line 6**: **`template = kernel[::-1]`**

- Reverses the kernel. This is because convolution, as defined here, is equivalent to correlation with a reversed kernel.

c) **Line 8-12**: **For loops**

- The nested loops calculate the convolution output.
- The outer loop iterates over the input data.
- The inner loop multiplies the data with the kernel values.
- Within this example, we use for loops however a better option would be to use matrix multiplication as they can be done in parallel as the Example 11.

```
1  def convolve1D(data, kernel):
2      output = []
3      template = kernel[::-1] # flipping the kernel filter to a template
4      for i in range(1,len(data)-1):
5          out.append(data[i-1]*template[0] + data[i]*template[1] + data[i+1]*
       template[2])
6
7      return np.array(output)
```

Example 11: Different version of the convolution function

[1] https://medium.datadriveninvestor.com/convolutional-neural-networks-3b241a5da51e

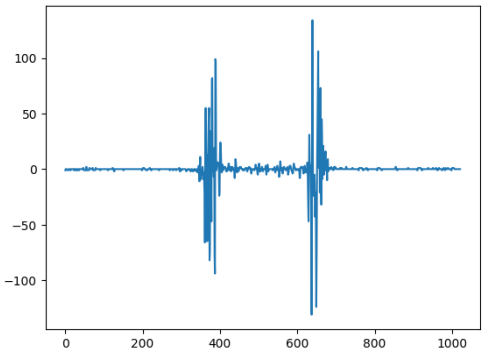
```
# Plot the output of the convolution function applied to y
k1 = np.array([-1, 0, 1.0])
plt.plot(convolve1D(y, k1))
plt.show()

# Plot y, and the output of your convolution function
plt.plot(range(len(y)), y, 'g')
plt.plot(convolve1D(y, k1))
plt.show()

# Try changing the filter you're applying.
k2 = np.array([0.1, 0.8, 0.1])
plt.plot(convolve1D(y, k2))
plt.show()
```

Example 12: Applies the Convolution() function

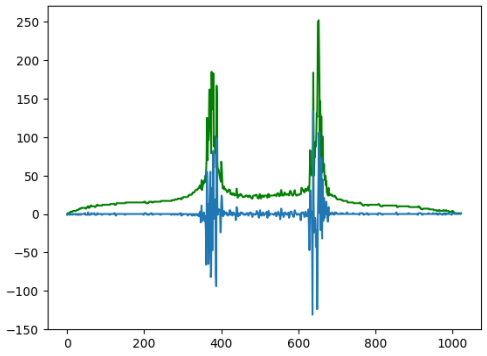
3. Using the function and plotting the output

a) **Line 2 & 12**: **`'k1' & 'k2'`**

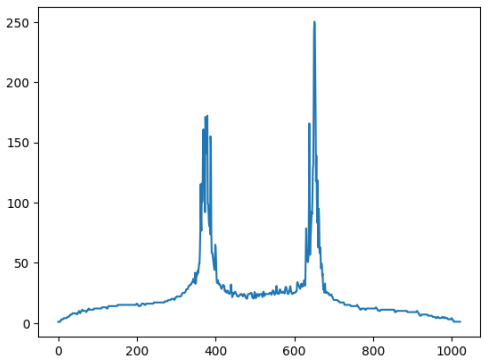
- `k1 = np.array([-1, 0, 1.0])`: A simple edge detection kernel.
- `k2 = np.array([0.1, 0.8, 0.1])`: A smoothing kernel.
- You can find outputs on the Figure 14.



(a) The result of convolving y with k1 (edge detection)



(b) Both the original signal y (in green) and the result of convolving it with k1



(c) The result of convolving y with k2 (smoothing).

Figure 14: 1D Convolution

4. References

[1] Agnieszka Ulrich. *How to understand a flattened image linear representation? - EITCA Academy*. June 2024. URL: https://eitca.org/artificial-intelligence/eitc-ai-dlpp-deep-learning-with-python-and-pytorch/data-eitc-ai-dlpp-deep-learning-with-python-and-pytorch/datasets/how-to-understand-a-flattened-image-linear-representation/.

[2] NguyenKhanhSon. *Unlocking Image Enhancement: A Guide to Histogram Processing and Equalization with OpenCV*. Oct. 2023. URL: https://medium.com/@khanhson0811/unlocking-image-enhancement-a-guide-to-histogram-processing-and-equalization-with-opencv-8db4477e6ba6.