HISTOGRAMS

ALIREZA SADEGHI NASAB

IMAGE PROCESSING COURSE - JUNE 2021

THEORICAL INTRODUCTION

- An image histogram is a type of histogram that reflects the tonal distribution of the image, plotting the number of pixels for each tonal value
- The number of pixels for each tonal value is also called frequency
- A histogram for a grayscale image with intensity values in the range [0, k-1] would contain exactly *K* entries
- Note that histograms show only statistical information and not the location of pixels

EXMAPLE

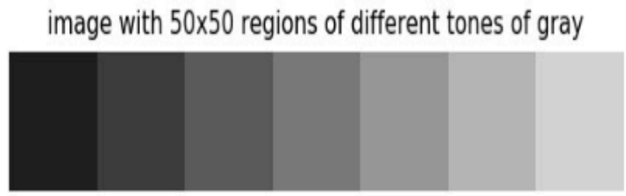
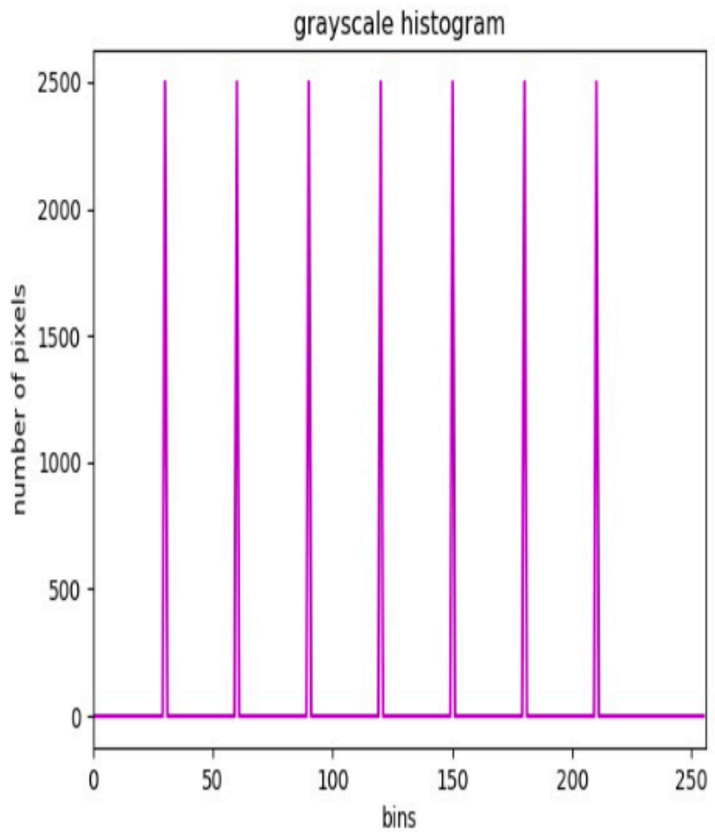


image with 50x50 regions of different tones of gray


grayscale histogram
number of pixels
2500
2000
1500
1000
500
0
0
50
100
150
200
250
bins

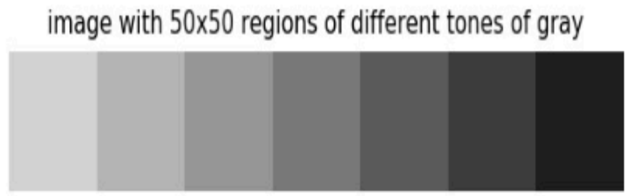
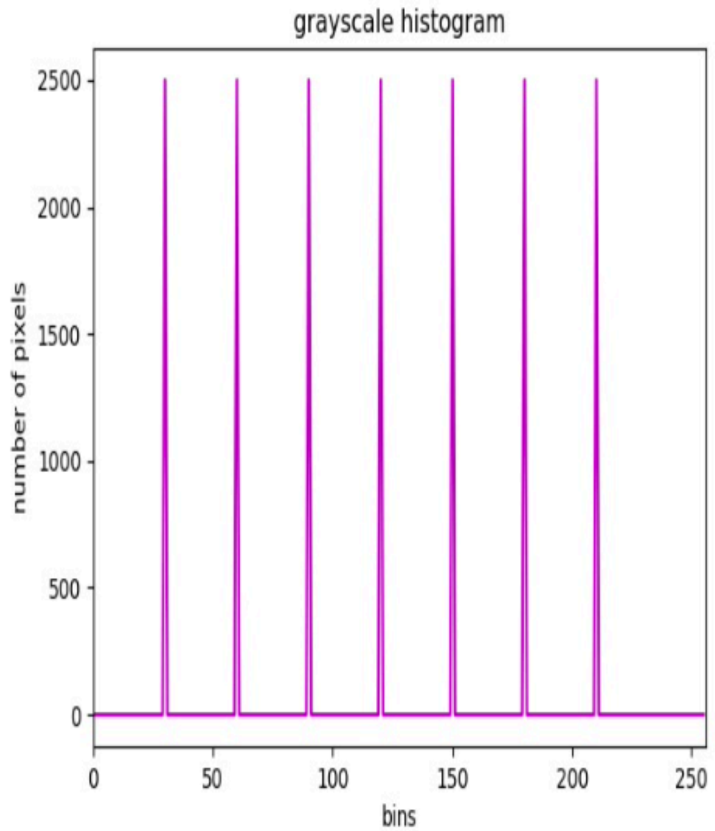


image with 50x50 regions of different tones of gray


grayscale histogram
number of pixels
2500
2000
1500
1000
500
0
0
50
100
150
200
250
bins

HISTOGRAM TERMINOLOGY

- Histogram shows the number of pixels (frequency) for every tonal value, ranging from 0 to 255. Each of these 256 values is called a ***bin*** in histogram terminology
- The number of bins can be selected as desired. Common values are 8, 16, 32, 64, 128, 256. OpenCV uses ***histSize*** to refer to *bins*
- Range of intensity values we want to measure is called a ***range*** in histogram terminology. Normally, it is [0, 255], corresponding to all the tonal values

CALCULATE HISTOGRAMS – FUNCTION SIGNATURE

- The signature for calculating histograms is as follows:

```
cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])
```

- To this, the following applies:
  - images: source images of type *uint8* or *float32*
  - channels: represents the index of the channel for which we calculate histogram provided as a list
  - mask: represents a mask image to calculate the histogram of a specific region of the image defined by the mask
  - histSize: represents the number of *bins* provided as a list
  - ranges: represents the range of intensity values we want to measure

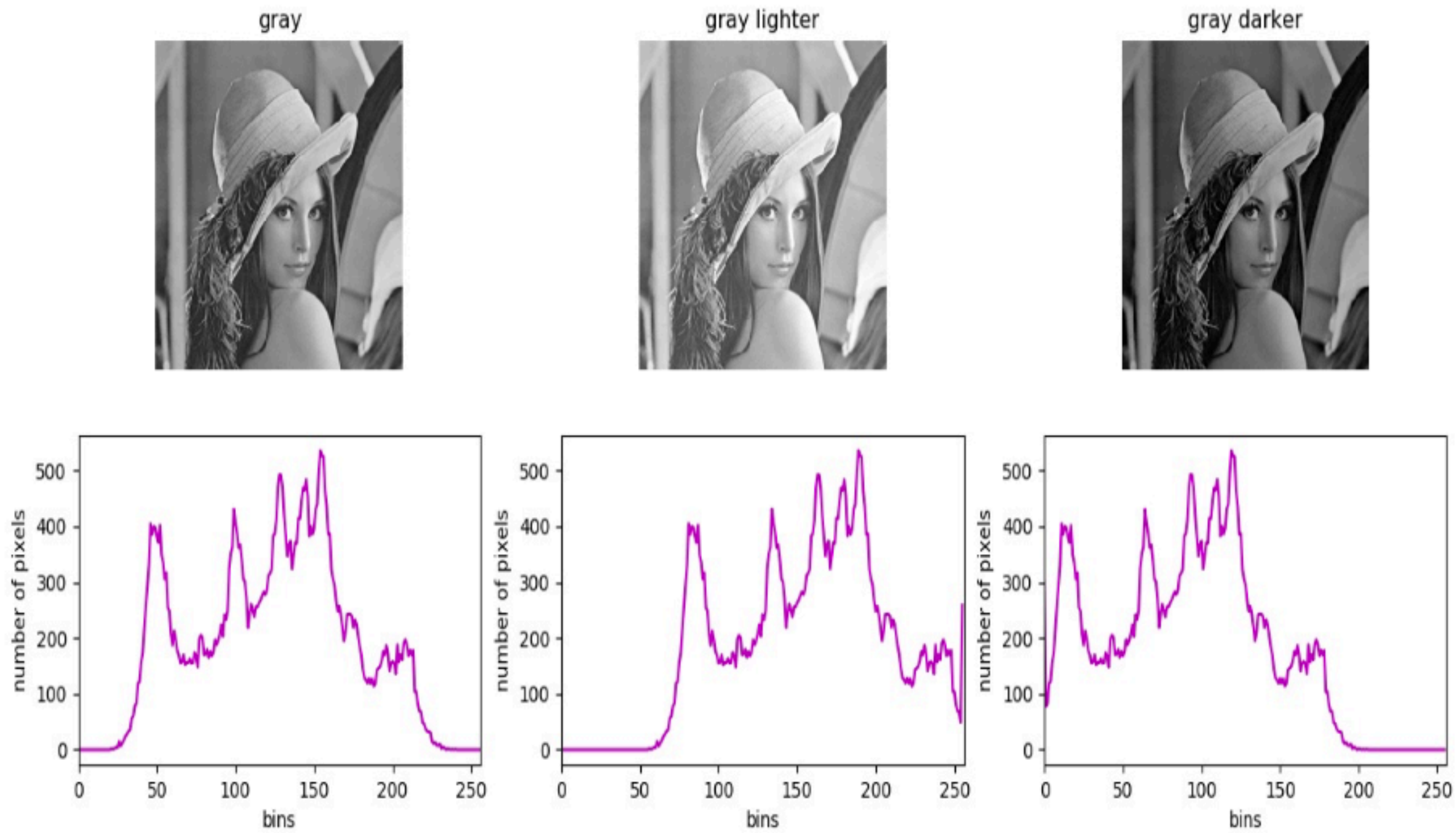
DETECT IMAGE BRIGHTNESS

- Histograms can be used to reveal or detect image acquisition issues
- The brightness of a grayscale image can be defined as the average intensity of all the pixels of the image

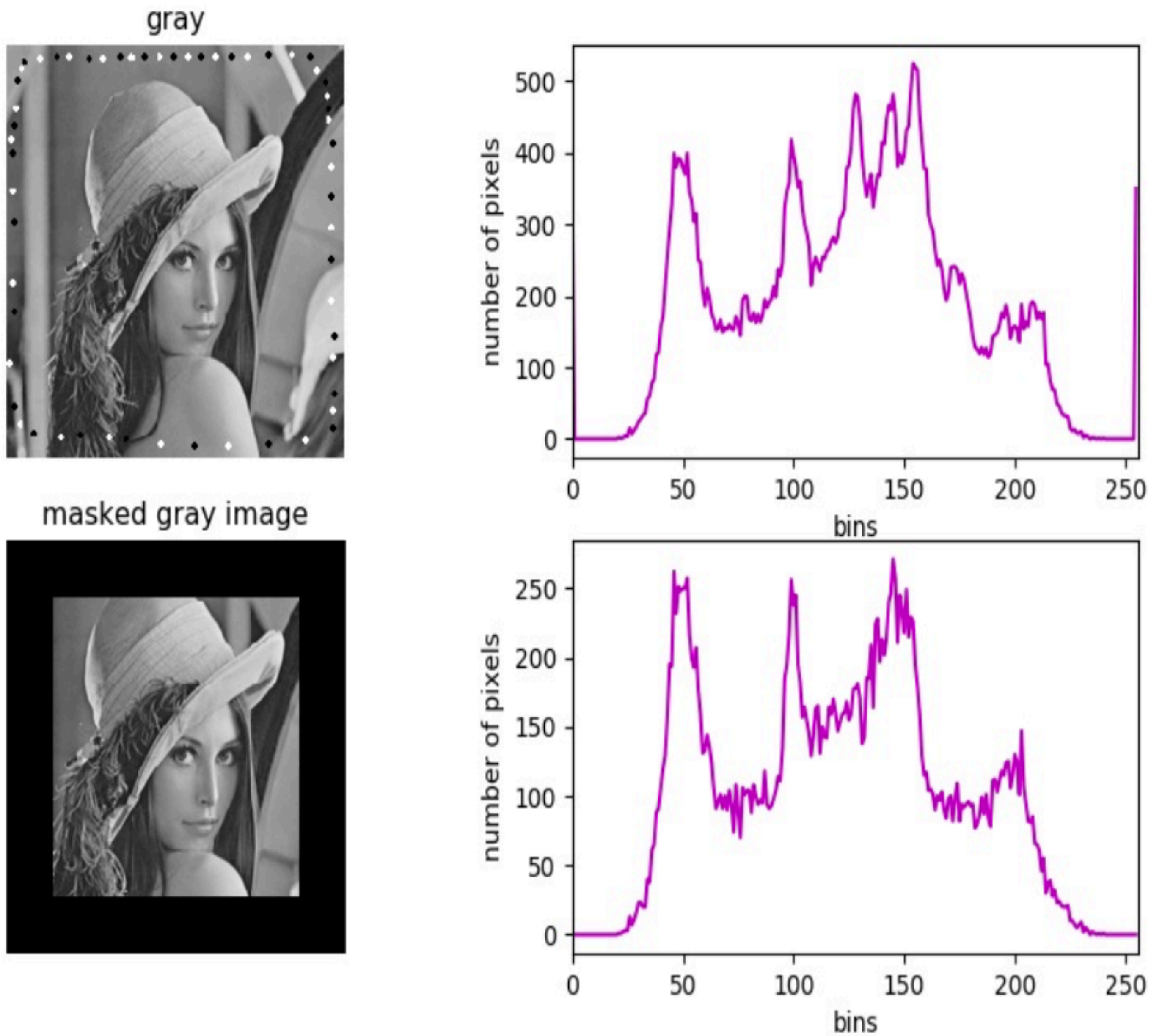
$$Brightness = \frac{1}{m \cdot n} \sum_{x=1}^{m} \sum_{y=1}^{n} I(x, y)$$

- Therefore, if the average tone of an image is high, this means that most pixels of the image will be very close to the white color and vice versa

EXMAPLE – GRAYSCALE HISTOGRAMS



EXMAPLE – MASKED HISTOGRAM

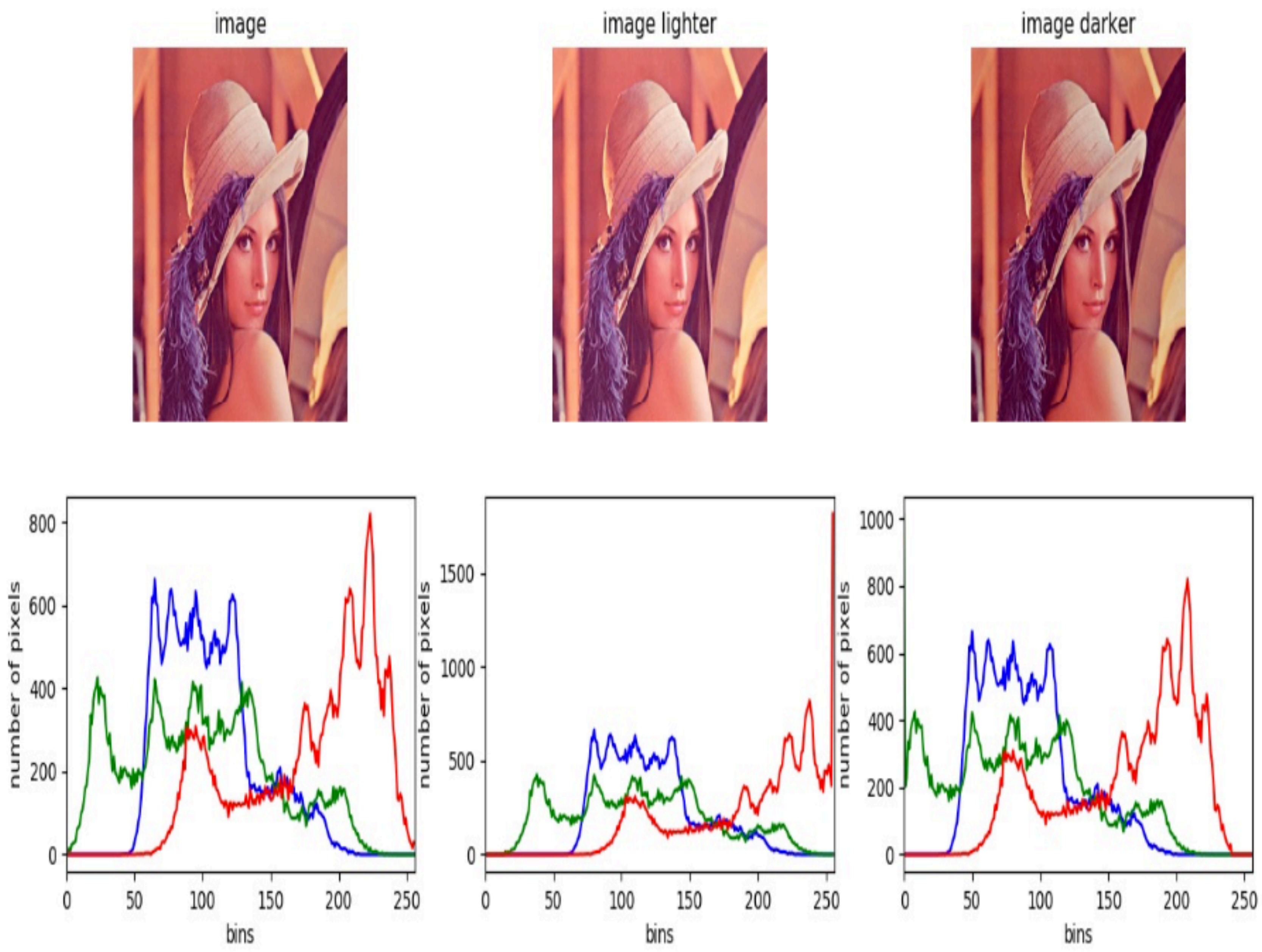


COLOR HISTOGRAMS

- In the case of a multi-channel image (ex: BGR image), the process of calculating the color histogram involves calculating the histogram in each of the channels

```
def hist_color_img(img):
    """Calculates the histogram from a three-channel image"""

    histr = []
    histr.append(cv2.calcHist([img], [0], None, [256], [0, 256]))
    histr.append(cv2.calcHist([img], [1], None, [256], [0, 256]))
    histr.append(cv2.calcHist([img], [2], None, [256], [0, 256]))
    return histr
```

EXMAPLE –COLOR HISTOGRAMS

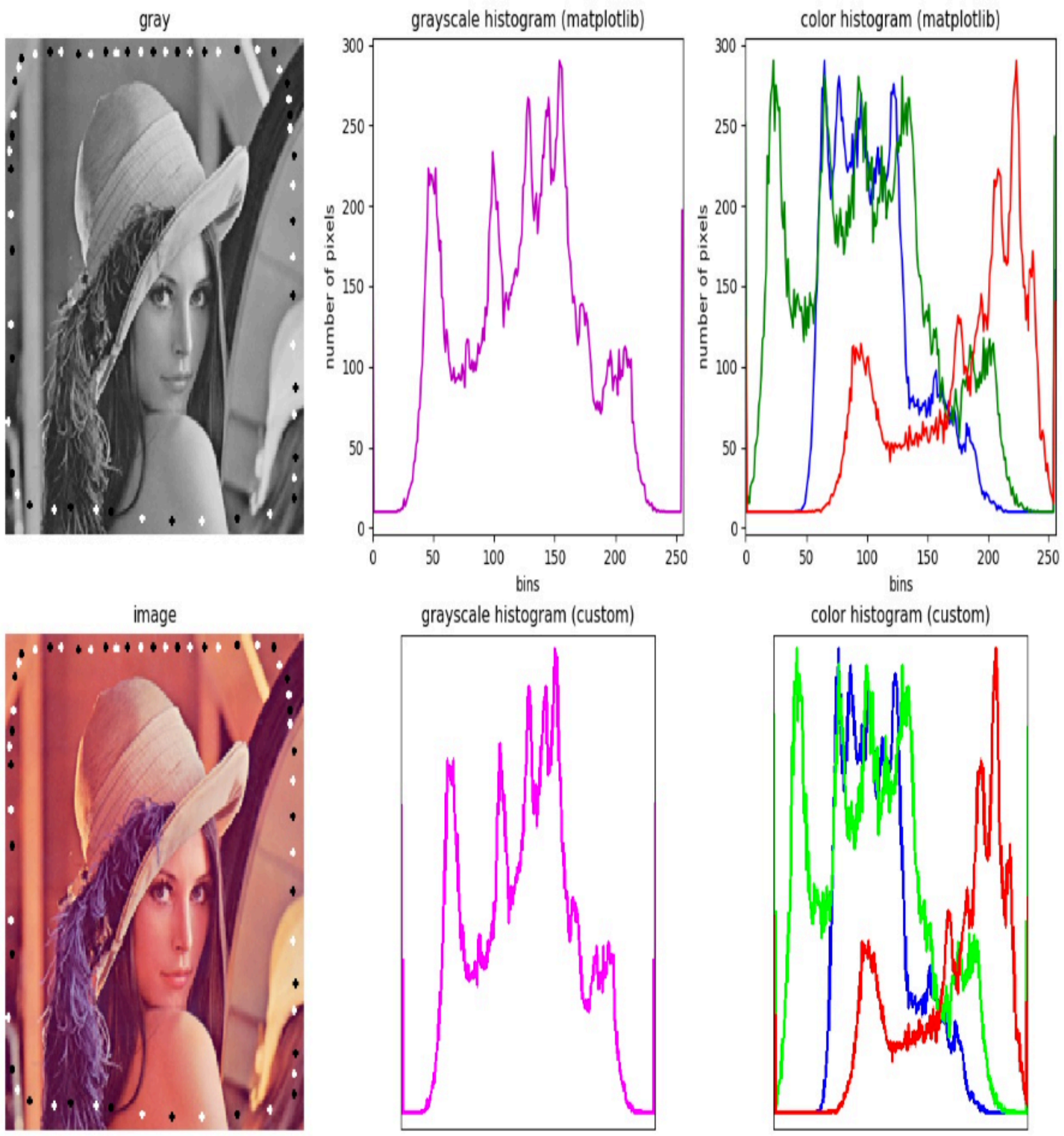


CUSTOM VISUALIZATIONS OF HISTOGRAMS

- If we want to visualize a histogram by using onle OpenCV capabilities, there is no OpenCV function to draw histograms. In this case, we have to make use of OpenCV primitives

```
def plot_hist(hist_items, color):
    """Plots the histogram of a image"""

    # For visualization purposes we add some offset:
    offset_down = 10
    offset_up = 10

    # This will be used for creating the points to visualize (x-coordinates):
    x_values = np.arange(256).reshape(256, 1)

    canvas = np.ones((300, 256, 3), dtype="uint8") * 255
    for hist_item, col in zip(hist_items, color):
        # Normalize in the range for proper visualization:
        cv2.normalize(hist_item, hist_item, 0 + offset_down, 300 - offset_up, cv2.NORM_M
        # Round the normalized values of the histogram:
        around = np.around(hist_item)
        # Cast the values to int:
        hist = np.int32(around)
        # Create the points using the histogram and the x-coordinates:
        pts = np.column_stack((x_values, hist))
        # Draw the points:
        cv2.polylines(canvas, [pts], False, col, 2)
        # Draw a rectangle:
        cv2.rectangle(canvas, (0, 0), (255, 298), (0, 0, 0), 1)

    # Flip the image in the up/down direction:
    res = np.flipud(canvas)

    return res
```

EXMAPLE –CUSTOM VISUALIZATIONS OF HISTOGRAMS

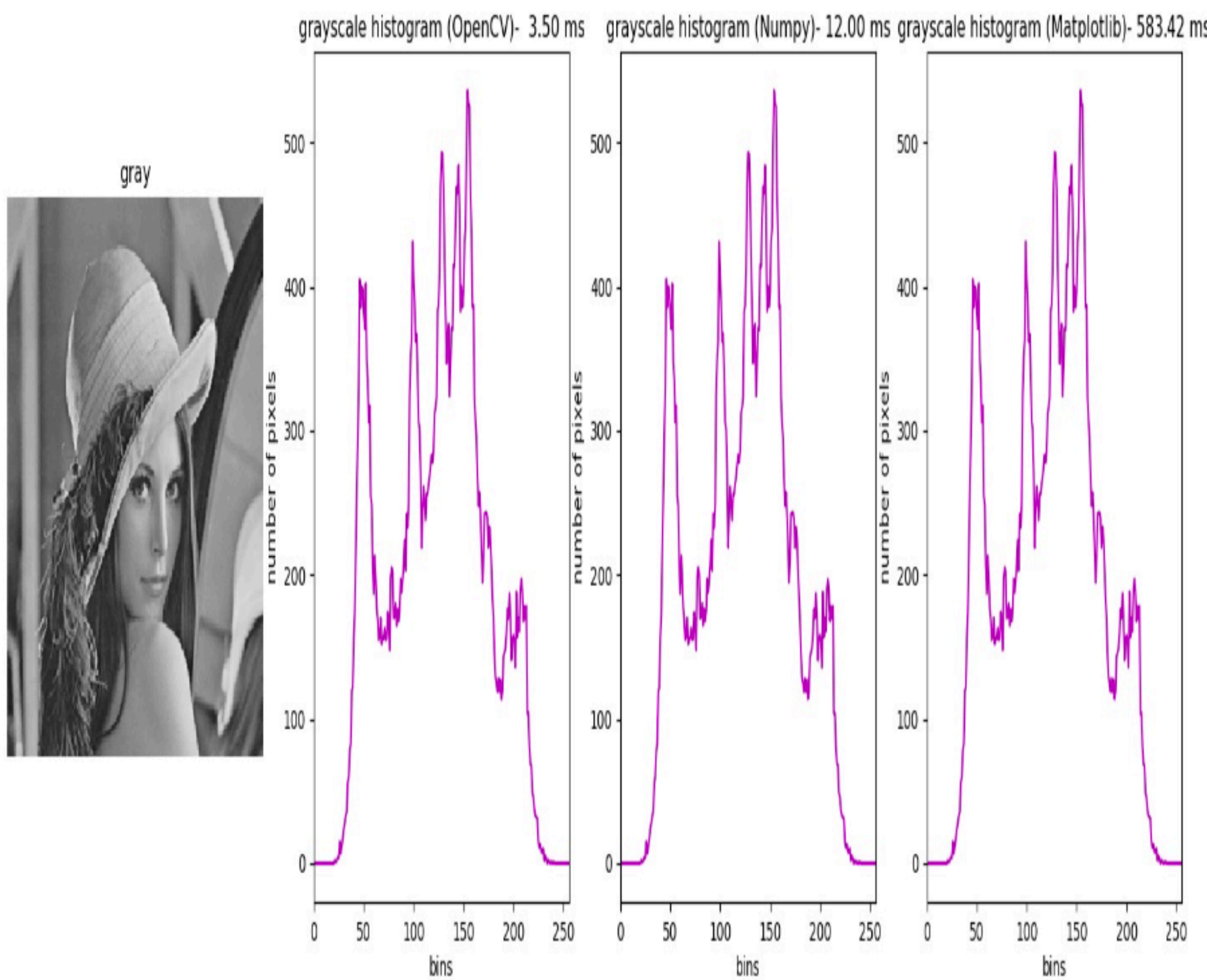


COMPARING OPENCV, NUMPY AND MATPLOTLIB HISTOGRAMS

- Comaparison is done for performance purposes

```
start = timer()
# Calculate the histogram calling cv2.calcHist()

hist = cv2.calcHist([gray_image], [0], None, [256], [0, 256])
end = timer()
exec_time_calc_hist = (end - start) * 1000

start = timer()
# Calculate the histogram calling np.histogram():
hist_np, bins_np = np.histogram(gray_image.ravel(), 256, [0, 256])
end = timer()
exec_time_np_hist = (end - start) * 1000

start = timer()
# Calculate the histogram calling plt.hist():
(n, bins, patches) = plt.hist(gray_image.ravel(), 256, [0, 256])
end = timer()
exec_time_plt_hist = (end - start) * 1000
```

EXMAPLE –COMPARING HISTOGRAMS



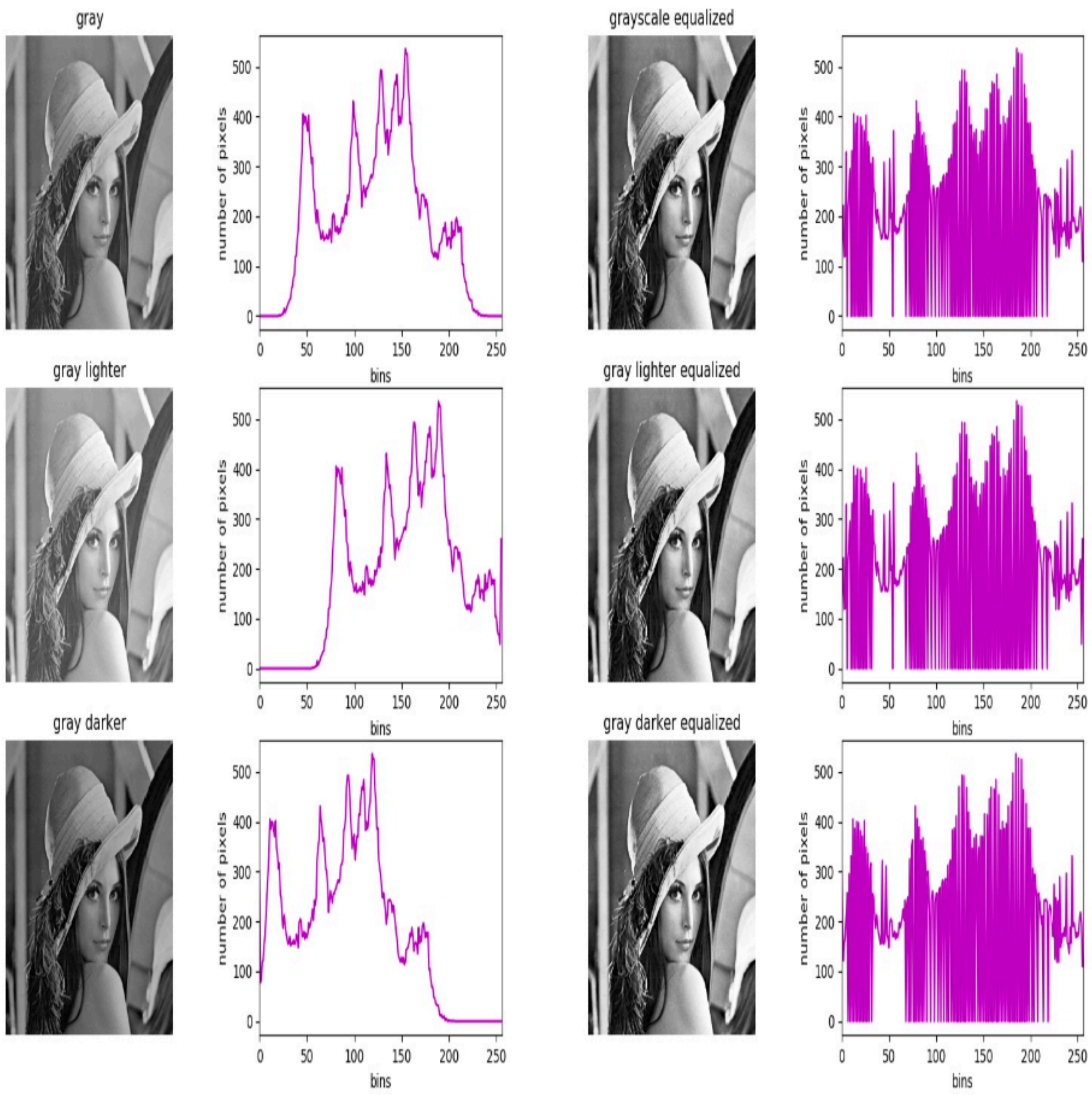
OpenVC is FASTER

GRAYSCALE HISTOGRAM EQUALIZATION

- The function normalizes the brightness and also increases the contrast of the image. Therefore, the histogram of the image is modified after applying this function

```
image = cv2.imread('lenna.png')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray_image_eq = cv2.equalizeHist(gray_image)
```

EXMAPLE –GRAYSCALE HISTOGRAM EQUALIZATION

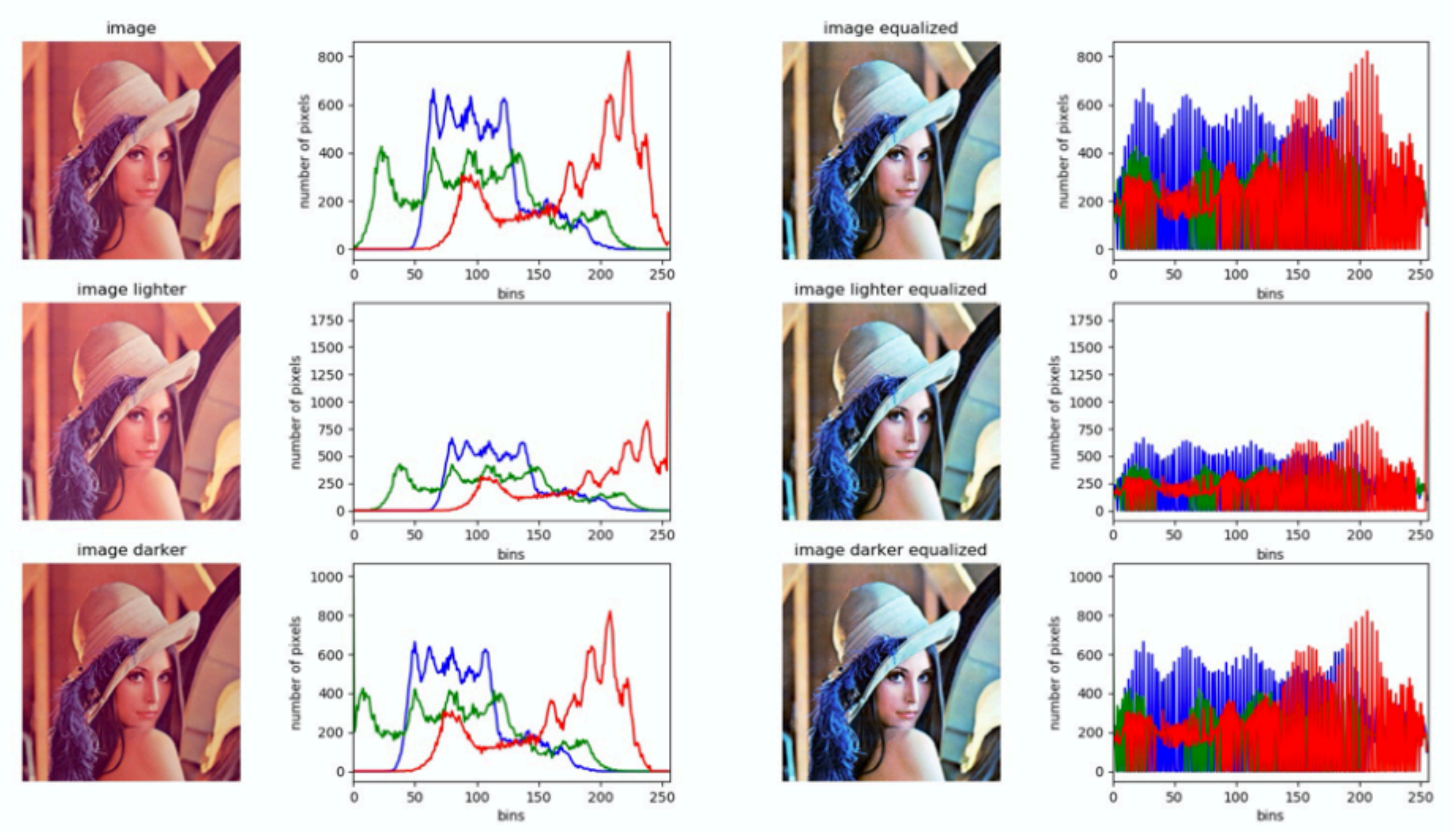


COLOR HISTOGRAM EQUALIZATION

- Following the same approach, we can perform histogram in color images. Note that this is not the best approach for histogram equalization in color images

```
def equalize_hist_color(img):
    """Equalize the image splitting the image applying cv2.equalizeHist() to each channe

    channels = cv2.split(img)
    eq_channels = []
    for ch in channels:
    eq_channels.append(cv2.equalizeHist(ch))

    eq_image = cv2.merge(eq_channels)
    return eq_image
```

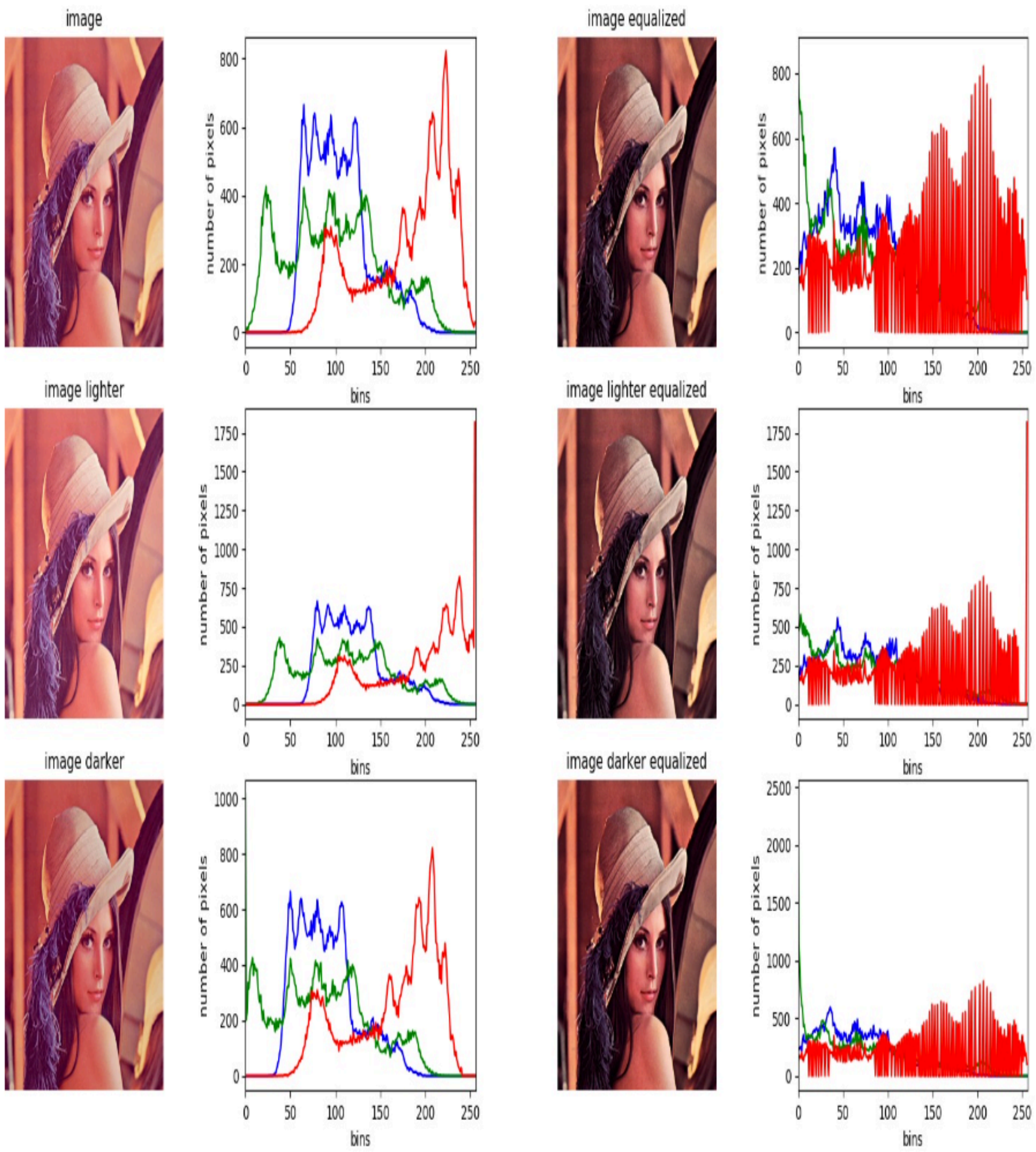
EXMAPLE –COLOR HISTOGRAM EQUALIZATION



COLOR HISTOGRAM EQUALIZATION - BETTER APPROACH

- Equalizaing the three channels is not a good approach because the color shade changes dramatically. This is due to the additive properties of the BGR color space
- As we are changing both the brightness and the contrast in the three channels independently, ths can lead to new color shades appearing in the image when merging the equalized channels
- A better approach is to convert the BGR image to a color space containing a luminance/intensity channel (Yuv, Lab, HSV and HSL). Then, we apply histogram equalization only on the luminance channel and finally, perform inverse transformation

EXMAPLE –COLOR HISTOGRAM EQUALIZATION IN V CHANNEL



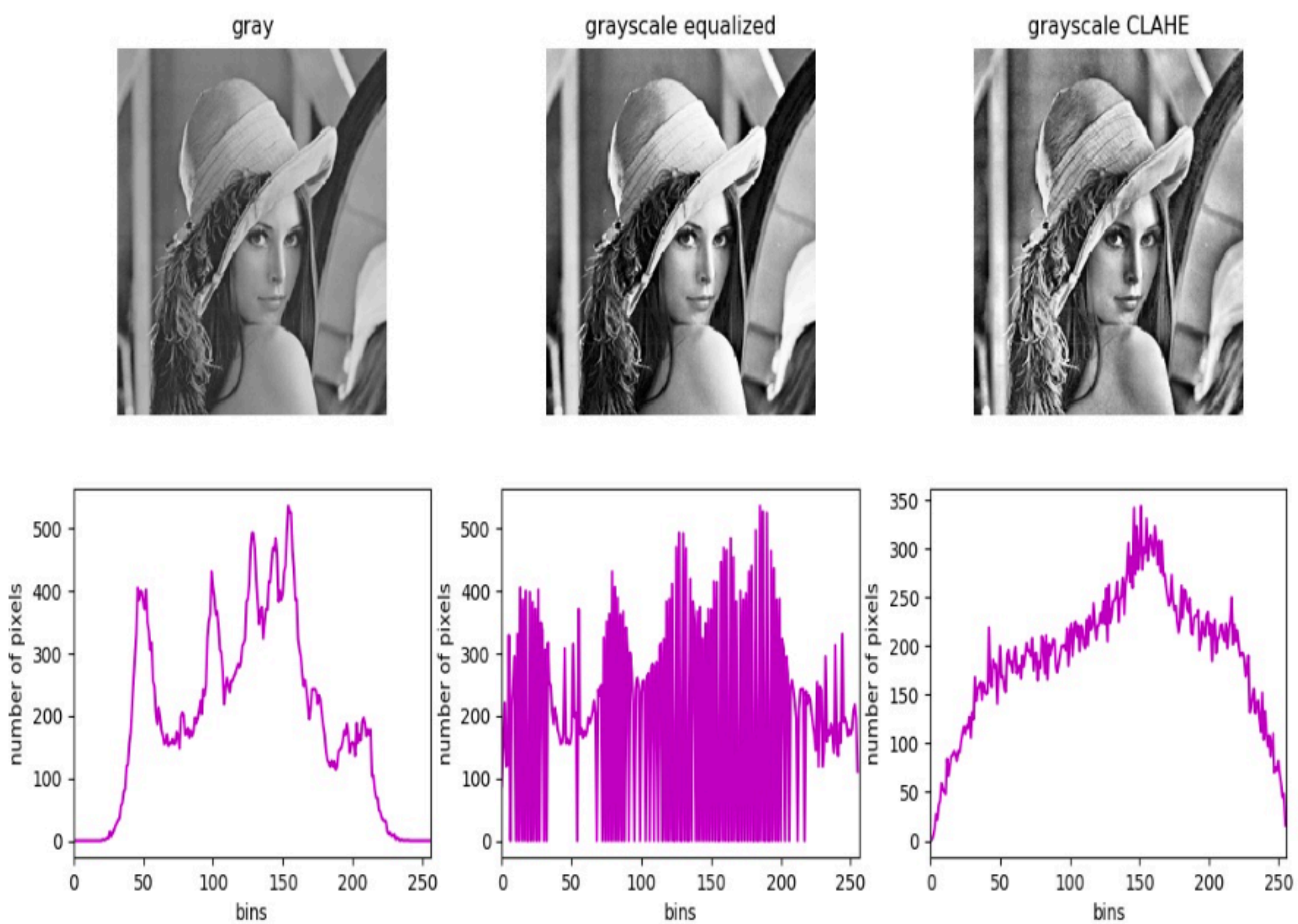
CONTRAST LIMITED ADAPTIVE HISTOGRAM EQUALIZATION

- The algorithm works by creating serveral histograms of the original image, and uses all of these histograms to redistribute the lightness of the image
- When applying CLAHE, there are two parameters to tune. The first one is ***clipLimit***, which sets the thresold for contrast limiting. The second one is ***tileGridSize***, which sets the number of tiles in the row and column
- When applying CLAHE, the image is divided into small blocks called **tiles** (8×8 by default) in order to perform its calculations

```
clahe = cv2.createCLAHE(clipLimit=2.0)
gray_image_clahe = clahe.apply(gray_image)
```

HISTOGRAM EQUALIZATION USING CLAHE

COMPARING CLAHE AND HISTOGRAM EQUALIZATION



CLAHE gives better results and perfrormance

HISTOGRAM COMPARISION

- One interesting functionality offered by OpenCV in connection with hsitograms is the ***cv2.compareHist()*** function, which can be used to get a numerical parameter expressing how well two histograms match each other.
- As histograms show only statistical information and not the location of pixels. Therefore, a common approach for image comparison is to divide the image into a certain number of regions, calculate the histogram for rach region and finally, concatenate all the histograms to create the feature representation of the image

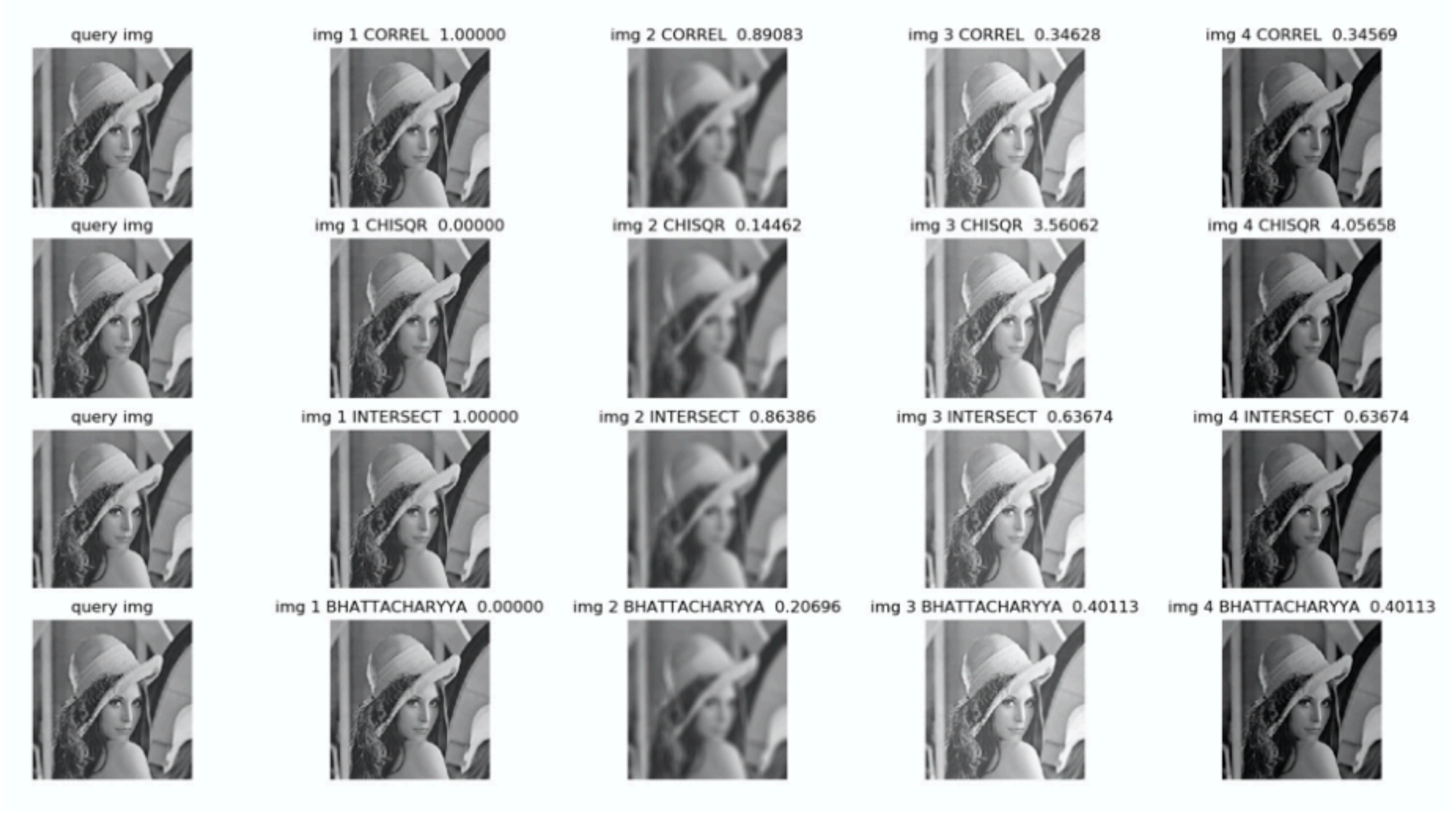
HISTOGRAM COMPARISION – CONTINUED

- The signature for the comparing function is as follows:

```
cv2.compareHist(H1, H2, method)
```

- OpenCV offers four different metrics to compute the matching:
  - **cv2.HISTCMP_CORREL:** computes the correlation between the two histograms
  - **cv2.HISTCMP_CHISQR:** computes the chi-squared distance between the two histograms
  - **cv2.HISTCMP_INTERSECT:** computes the intersection between the two histograms
  - **cv2.HISTCMP_BHATTACHARYYA:** computes the Bhattacharyya distance between the two histograms

HISTOGRAM COMPARISION




Thank You for Your Attention!