TÉCNICO LISBOA

Learning-based Multimedia Processing

2023/2024

Lab classes (3)

1 Image processing – Intensity Transformations

1.1 Mirror image, Negative image

To mirror an image, just read the horizontal component of the image matrix from end to start

Arithmetic operations can be applied to all pixels – see example of creating a negative image

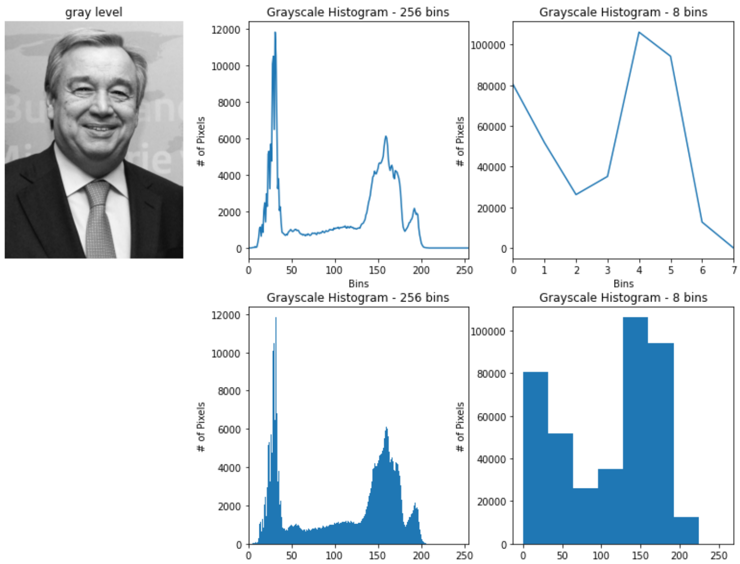
```
import cv2

img = cv2.imread("images/guterres-1.jpg")

# Mirror the image
height, width = img.shape[0:2]
mirror_img = img[:,width:0:-1]

# Grayscale image
gray_img = cv2.cvtColor(img,
cv2.COLOR_BGR2GRAY)

# Negative image
neg_img = 255 - gray_img
```

1.2 Image histogram

Grayscale histogram
(remember that in Python the ending range of an array is non-inclusive)

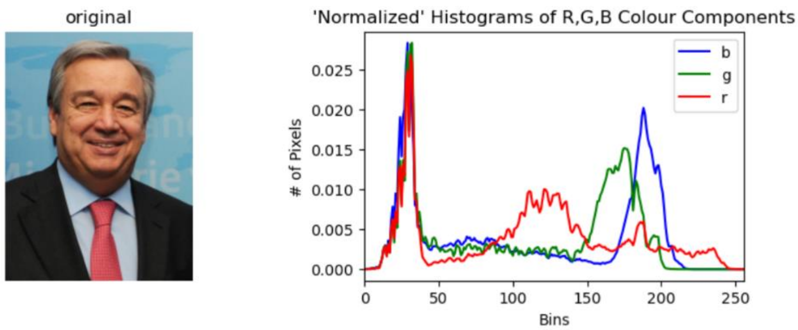
```
import cv2
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = [15, 3]    # Size of displayed images

img = cv2.imread("images/guterres-1.jpg")
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)    # grayscale

# cv2.calcHist(images, channels, mask, histSize, ranges)
hist_8 = cv2.calcHist([gray_img], [0], None, [8], [0, 256])   # 8 bins

# plot the histogram
plt.figure()
plt.plot(hist_8)
plt.xlim([0, 7])
plt.title("Grayscale Histogram - 8 bins")
plt.xlabel("Bins")
plt.ylabel("# of Pixels")
plt.show()
```

```
# ALTERNATIVE way to find histogram of an image
plt.figure()
plt.hist(gray_img.ravel(),8,[0,256])
plt.title("Grayscale Histogram - 8 bins")
plt.ylabel("# of Pixels")
plt.show()
```



Histograms of colour components (R,G,B)

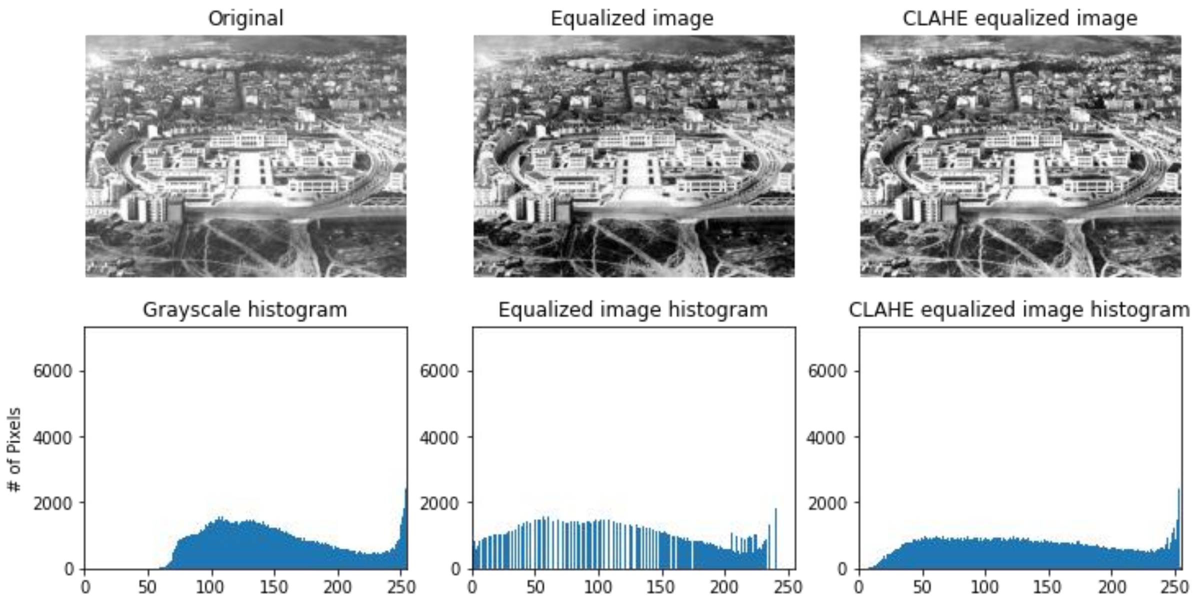
```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("images/guterres-1.jpg")

chans = cv2.split(img)      # split image into its B, G, R channels
colors = ("b", "g", "r")    # set colours to be used for plotting

plt.figure()
plt.title("'Flattened' Color Histogram")
# loop over the image channels
for (chan, color) in zip(chans, colors):
    # create a histogram for current channel and plot it
    hist = cv2.calcHist([chan], [0], None, [256], [0, 256])
    hist /= hist.sum()          # normalize the histogram
    plt.plot(hist, color=color, label=color)
    plt.xlim([0, 256])
plt.legend(loc="upper right")
```



1.3 Histogram equalization

To increase the range of pixel values in the image – more uniform histogram

```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("images/ist12.jpg")
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # grayscale

# apply histogram equalization
eq_img = cv2.equalizeHist(gray_img)

# apply contrast limited adapted histogram equalization (CLAHE)
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
cl_img = clahe.apply(gray_img)
```



2 Image processing – Spatial filtering

OpenCV has a function to convolve an image with a kernel:

```
dst = cv2.filter2D (src, ddepth, kernel)
```

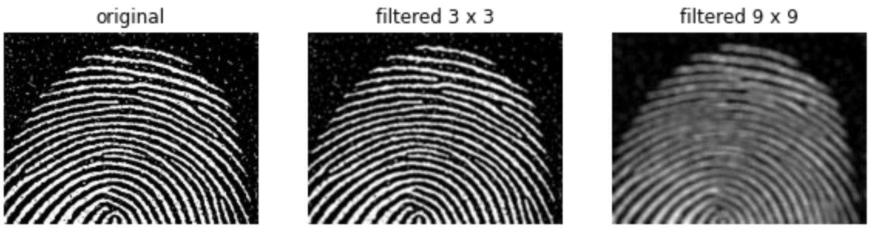
Parameters

- src input image.
- ddepth desired depth of the destination image;
  when `ddepth=-1`, the output image will have the same depth as the source
- kernel kernel defining the filter to apply; to apply different kernels to different channels, split the image into separate colour planes using `split` and process them individually.
- dst output image of the same size and the same number of channels as src.

2.1 Low pass filtering

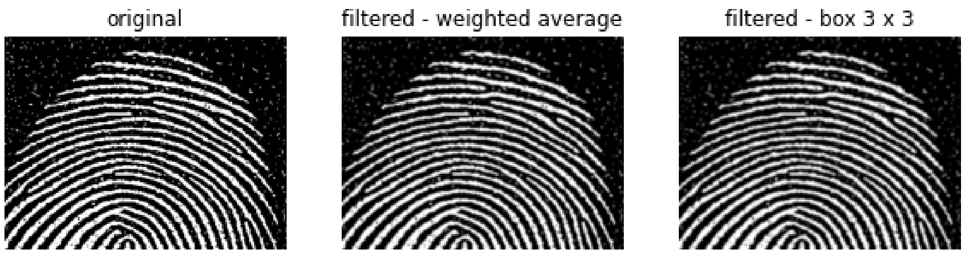
Box filter:

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

img = cv2.imread("images/i04.jpg")   # 600x400 image

# Define kernel - Box filter nxn
n1 = 3
kernel1 = np.ones((n1,n1),np.float32)/(n1*n1)
n2 = 9
kernel2 = np.ones((n2,n2),np.float32)/(n2*n2)

filtered1 = cv2.filter2D (img, -1, kernel1)
filtered2 = cv2.filter2D (img, -1, kernel2)

plt.subplot(1, 3, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('original')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(cv2.cvtColor(filtered1, cv2.COLOR_BGR2RGB))
plt.title('filtered {} x {}'.format(n1, n1))
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(cv2.cvtColor(filtered2, cv2.COLOR_BGR2RGB))
plt.title('filtered {} x {}'.format(n2, n2))
plt.axis('off')
plt.show()
```



Weighted low pass filter

```
# Define kernel - Weighted filter
kernel = np.array([
    [1, 2, 1],
    [2, 4, 2],
    [1, 2, 1]])
kernel = kernel / np.sum(kernel)
```



Gaussian kernel

A Gaussian kernel can be created using: **`cv2.getGaussianKernel`**`(ksize, sigma[, ktype])`

Just set the kernel size and the standard deviation, then apply the kernel. Example:

```
kernel = cv2.getGaussianKernel(9,2)
filtered = cv2.filter2D (img, -1, kernel)
```

As an alternative use the function **`GaussianBlur()`**. Example:

```
# cv2.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]])
filtered = cv2.GaussianBlur(img,(9,9),2)
```

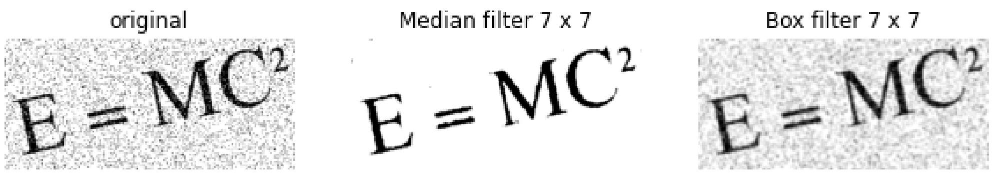
If only *sigmaX* is specified, *sigmaY* is taken as the same as *sigmaX*. If both are given as zeros, they are calculated from the kernel size.

2.2 Order filtering

Median filter

The median filter is a non-linear order statistical filter, implemented with the `medianBlur()` function. Example of application:

```
filtered = cv2.medianBlur(img, n1) # Add median filter to image
```



2.3 High pass filtering

Sobel

The Sobel kernels look for horizontal and vertical discontinuities. Example:

```
# Define kernels - Sobel
k_h = np.array([
    [-1, -2, -1],
    [0, 0, 0],
    [1, 2, 1]])
k_v = np.array([
    [-1, 0, 1],
    [-2, 0, 2],
    [-1, 0, 1]])
f_h = cv2.filter2D (img, -1, k_h)
f_v = cv2.filter2D (img, -1, k_v)
```

Laplacian and Unmask filtering

The Laplacian corresponds to the second derivative. When added to the original image it produces a sharpened image. Example:

```
# Define kernel - Laplacian
k_lap = np.array([
    [-1, -1, -1],
    [-1, 8, -1],
    [-1, -1, -1]])
# kernel = kernel / np.sum(kernel)

# Define kernel - Unsharp masking   (f_sharp = f + k . f_lap,  k=1)
k_sharp = np.array([
    [-1, -1, -1],
    [-1, 9, -1],
    [-1, -1, -1]])

f_lap = cv2.filter2D (img, -1, k_lap)
f_sharp = cv2.filter2D (img, -1, k_sharp)
```