1. **Histogram Equalization and Stretching**

Useful [URL](https://docs.opencv.org/4.x/d5/daf/tutorial_py_histogram_equalization.html#:~:text=OpenCV%20has%20a%20function%20to%20do%20this%2C%20cv.&text=Saves%20an%20image%20to%20a%20specified%20file.&text=Equalizes%20the%20histogram%20of%20a,confined%20to%20a%20particular%20region.)

Given an image the code below plots the histogram and cumulative density function of that image. [you will notice that the density of pixel is around the centre of the histogram]

import numpy as np

import cv2 as cv

from matplotlib import pyplot as plt

img = [cv.imread](https://docs.opencv.org/4.x/d4/da8/group__imgcodecs.html#gacbaa02cffc4ec2422dfa2e24412a99e2)('wiki.jpg', cv.IMREAD\_GRAYSCALE)

assert img is not None, "file could not be read, check with os.path.exists()"

hist,bins = np.histogram(img.flatten(),256,[0,256])

cdf = hist.cumsum()

cdf\_normalized = cdf \* float(hist.max()) / cdf.max()

plt.plot(cdf\_normalized, color = 'b')

plt.hist(img.flatten(),256,[0,256], color = 'r')

plt.xlim([0,256])

plt.legend(('cdf','histogram'), loc = 'upper left')

plt.show()

A good image should have pixel values from all regions of the image, which is the purpose of Histogram equalisation. (code in python using numpy)

cdf\_m = np.ma.masked\_equal(cdf,0)

cdf\_m = (cdf\_m - cdf\_m.min())\*255/(cdf\_m.max()-cdf\_m.min())

cdf = np.ma.filled(cdf\_m,0).astype('uint8')

img2 = cdf[img]

# we now calculate the histogram and cdf as before

**With Opencv**

img = [cv.imread](https://docs.opencv.org/4.x/d4/da8/group__imgcodecs.html#gacbaa02cffc4ec2422dfa2e24412a99e2)('wiki.jpg', cv.IMREAD\_GRAYSCALE)

assert img is not None, "file could not be read, check with os.path.exists()"

equ = [cv.equalizeHist](https://docs.opencv.org/4.x/d6/dc7/group__imgproc__hist.html#ga7e54091f0c937d49bf84152a16f76d6e)(img)

res = np.hstack((img,equ)) #stacking images side-by-side

[cv.imwrite](https://docs.opencv.org/4.x/d4/da8/group__imgcodecs.html#ga8ac397bd09e48851665edbe12aa28f25)('res.png',res)

1. **Image Thresholding**

Useful [URL](https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html)

The sample codes in the url basically contain different image thresholding techniques. example:

import cv2 as cv

import numpy as np

from matplotlib import pyplot as plt

img = [cv.imread](https://docs.opencv.org/4.x/d4/da8/group__imgcodecs.html#gacbaa02cffc4ec2422dfa2e24412a99e2)('gradient.png', cv.IMREAD\_GRAYSCALE)

assert img is not None, "file could not be read, check with os.path.exists()"

ret,thresh1 = [cv.threshold](https://docs.opencv.org/4.x/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57)(img,127,255,cv.THRESH\_BINARY)

ret,thresh2 = [cv.threshold](https://docs.opencv.org/4.x/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57)(img,127,255,cv.THRESH\_BINARY\_INV)

ret,thresh3 = [cv.threshold](https://docs.opencv.org/4.x/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57)(img,127,255,cv.THRESH\_TRUNC)

ret,thresh4 = [cv.threshold](https://docs.opencv.org/4.x/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57)(img,127,255,cv.THRESH\_TOZERO)

ret,thresh5 = [cv.threshold](https://docs.opencv.org/4.x/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57)(img,127,255,cv.THRESH\_TOZERO\_INV)

titles = ['Original Image','BINARY','BINARY\_INV','TRUNC','TOZERO','TOZERO\_INV']

images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]

for i in range(6):

plt.subplot(2,3,i+1),plt.imshow(images[i],'gray',vmin=0,vmax=255)

plt.title(titles[i])

plt.xticks([]),plt.yticks([])

plt.show()

1. **Image Enhancement with Sobel, LoG, Laplacian filters**

Useful [URL](https://docs.opencv.org/4.x/d5/d0f/tutorial_py_gradients.html)

We can easily obtain [from OpenCV library] a set of differential filters including Sobel and Laplace filters. Sample code:

import numpy as np

import cv2 as cv

from matplotlib import pyplot as plt

img = [cv.imread](https://docs.opencv.org/4.x/d4/da8/group__imgcodecs.html#gacbaa02cffc4ec2422dfa2e24412a99e2)('dave.jpg', cv.IMREAD\_GRAYSCALE)

assert img is not None, "file could not be read, check with os.path.exists()"

laplacian = [cv.Laplacian](https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#gad78703e4c8fe703d479c1860d76429e6)(img,cv.CV\_64F)

sobelx = [cv.Sobel](https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#gacea54f142e81b6758cb6f375ce782c8d)(img,cv.CV\_64F,1,0,ksize=5)

sobely = [cv.Sobel](https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#gacea54f142e81b6758cb6f375ce782c8d)(img,cv.CV\_64F,0,1,ksize=5)

plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')

plt.title('Original'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')

plt.title('Laplacian'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')

plt.title('Sobel X'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')

plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])

plt.show()

**LoG [Laplacian of Gaussian]**

import cv2

import numpy as np

import matplotlib.pyplot as plt

image = cv2.imread('image.jpg', cv2.IMREAD\_GRAYSCALE)

# Apply Gaussian Blur

sigma = 1.0

blurred\_image = cv2.GaussianBlur(image, (5, 5), sigma)

# Apply Laplacian

laplacian\_image = cv2.Laplacian(blurred\_image, cv2.CV\_64F)

laplacian\_image = cv2.convertScaleAbs(laplacian\_image)

plt.subplot(1, 2, 1)

plt.imshow(image, cmap='gray')

plt.title('Original Image')

plt.subplot(1, 2, 2)

plt.imshow(laplacian\_image, cmap='gray')

plt.title('Laplacian of Gaussian (LoG)')

plt.show()

1. **Finding 2D Fourier Transform and Plotting It**

Useful [URL](https://docs.opencv.org/3.4/de/dbc/tutorial_py_fourier_transform.html)

This code applies the **2D Fast Fourier Transform (FFT)** to the image using NumPy's fft2() function.

import cv2 as cv

import numpy as np

from matplotlib import pyplot as plt

img = [cv.imread](https://docs.opencv.org/3.4/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fce07b3bbd3a56)('messi5.jpg', cv.IMREAD\_GRAYSCALE)

assert img is not None, "file could not be read, check with os.path.exists()"

f = np.fft.fft2(img)

fshift = np.fft.fftshift(f)

The Fourier Transform result places the **low-frequency components** in the corners of the result. To visualize it better, we use fftshift() to shift these low frequencies to the **center** of the image.

magnitude\_spectrum = 20\*np.log(np.abs(fshift))

plt.subplot(121),plt.imshow(img, cmap = 'gray')

plt.title('Input Image'), plt.xticks([]), plt.yticks([])

plt.subplot(122),plt.imshow(magnitude\_spectrum, cmap = 'gray')

plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])

plt.show()

1. **Some sort of filtering in the frequency domain and then inverse transform and plotting the image**

Useful [URL](https://docs.opencv.org/3.4/de/dbc/tutorial_py_fourier_transform.html)

**# Reading the Image**

import cv2 as cv

import numpy as np

from matplotlib import pyplot as plt

img = [cv.imread](https://docs.opencv.org/3.4/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fce07b3bbd3a56)('messi5.jpg', cv.IMREAD\_GRAYSCALE)

assert img is not None, "file could not be read, check with os.path.exists()"

f = np.fft.fft2(img)

fshift = np.fft.fftshift(f)

**# Create a high-pass filter mask, you can also design low-pass filter or any filter of your choice**

rows, cols = img.shape

crow, ccol = rows//2, cols//2

fshift[crow-30:crow+31, ccol-30:ccol+31] = 0

f\_ishift = np.fft.ifftshift(fshift)

img\_back = np.fft.ifft2(f\_ishift)

img\_back = np.real(img\_back)

plt.subplot(131),plt.imshow(img, cmap = 'gray')

plt.title('Input Image'), plt.xticks([]), plt.yticks([])

plt.subplot(132),plt.imshow(img\_back, cmap = 'gray')

plt.title('Image after HPF'), plt.xticks([]), plt.yticks([])

plt.subplot(133),plt.imshow(img\_back)

plt.title('Result in JET'), plt.xticks([]), plt.yticks([])

plt.show()