
Downsampling

```
import cv2
import matplotlib.pyplot as plt

img1 = cv2.imread("C:// banana.jpg", 0)
# 0, indicates that the image should be read as grayscale.
img2 = img1[::24, ::24]
#(The downsampling is performed by slicing the image array using img1[::24, ::24], which skips every 24th row and column of the image).

print('Original Image Shape:', img1.shape)
print('Down Sampled Image Shape:', img2.shape)

plt.subplot(121), plt.imshow(img1, cmap="gray")
plt.title('Original Image')

plt.subplot(122), plt.imshow(img2, cmap="gray")
plt.title('Down Sampled Image')

plt.show()
```

#upsampling

```
import cv2

import matplotlib.pyplot as plt

import numpy as np


img1 = cv2.imread("C://banana.jpg", 0)  # Read the image

factor = 4
```

 # (factor = 4: Specifies the upsampling factor. This factor determines how much the image will be enlarged during the upsampling process.)

```
# Upsample the image

Img2 = img1.repeat(factor, axis=0).repeat(factor, axis=1)
```

#(The repeat function is used to replicate each pixel factor times in both the vertical (axis=0) and horizontal (axis=1) directions.)

```
# Display the original and upsampled images

plt.subplot(121), plt.imshow(img1, cmap="gray")

plt.title('Original Image')


plt.subplot(122), plt.imshow(Img2 _img, cmap="gray")

plt.title('Upsampled Image')


plt.show()
```

```
# 1B. Fast Fourier Transform to compute DFT.
import cv2
```

```python
import numpy as np

import matplotlib.pyplot as plt


# Read the image

image = cv2.imread("C://Users//viggu//Downloads//banana.jpg", 0)


# Compute the FFT

fft_result = np.fft.fft2(image)

# This line computes the 2-dimensional fast Fourier transform (FFT) of the input image using np.fft.fft2.


# Shift the zero frequency component to the center of the spectrum

fft_shifted = np.fft.fftshift(fft_result)


# Compute the magnitude spectrum

magnitude_spectrum = np.abs(fft_shifted)

# function abs(), which computes the absolute value of a given array or complex number.


# Plot the original image and its magnitude spectrum

fig, ax = plt.subplots(1, 2, figsize=(10, 5))

ax[0].imshow(image, cmap="gray")

ax[0].set_title('Original Image')


ax[1].imshow(np.log(1 + magnitude_spectrum), cmap="gray")

ax[1].set_title('Magnitude Spectrum')
```

```python
plt.tight_layout()
plt.show()
```

Convolution and correlation

```python
# Import libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read the image and convert color format
image = cv2.imread('F:/nativeplace.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Display the original image
fig, ax = plt.subplots(1, figsize=(12,8))
plt.imshow(image)

# Image blurring using averaging kernel
abc = np.ones((3,3))
# The function np.ones generates a matrix of ones with the specified shape (3 rows and 3 columns). The resulting array abc will be used for image blurring, as part of the kernel.
kernel = np.ones((3, 3), np.float32) / 9
# Dividing by 9 ensures that the sum of the kernel values equals 1,
img = cv2.filter2D(image, -1, kernel)
```

```python
# Display original and blurred images
fig, ax = plt.subplots(1, 2, figsize=(10,6))
ax[0].imshow(image)
ax[1].imshow(img)

# Sharpening the image
kernel = np.array([[0, -1, 0],
                   [-1, 5, -1],
                   [0, -1, 0]])
img = cv2.filter2D(image, -1, kernel)
# The center element (5) has a higher weight compared to the surrounding elements. This enhances the central pixel value and sharpens the edges in the image.

# Display original and sharpened images
fig, ax = plt.subplots(1, 2, figsize=(10,6))
ax[0].imshow(image)
ax[1].imshow(img)
```

DFT of 4x4 Gray Scale Image.

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

# Read the image as grayscale
image = cv2.imread("C:// banana.jpg", 0)
```

```
# Resize the image to a 4x4 size

# This line resizes the image to a 4x4 size using cv2.resize. The resulting image will be a smaller version of the original image.

image = cv2.resize(image, (4, 4))


# Compute the DFT using np.fft.fft2()

dft_result = np.fft.fft2(image)


# Shift the zero frequency component to the center of the spectrum

dft_shifted = np.fft.fftshift(dft_result)


# Compute the magnitude spectrum

magnitude_spectrum = np.abs(dft_shifted)


# Display the original image

plt.subplot(121)

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

plt.title('Original Image')


# Display the magnitude spectrum

plt.subplot(122)

plt.imshow(np.log(1 + magnitude_spectrum), cmap="gray")

plt.title('DFT Magnitude Spectrum')

plt.colorbar()


plt.tight_layout()
```

```
plt.show()
```

Log and Power-law transformations

```
# log transform

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

# Open the image.
img = cv2.imread('F:/sample.jpg')

# Apply log transform.
c = 255 / (np.log(1 + np.max(img)))
log_transformed = c * np.log(1 + img)

# Specify the data type.
log_transformed = np.array(log_transformed, dtype=np.uint8)

# Display the original image.
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')
```

```
# Display the transformed image.
plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(log_transformed, cv2.COLOR_BGR2RGB))
plt.title('Log Transformed Image')
plt.axis('off')

# Show the plot.
plt.tight_layout()
plt.show()
```

4C. Histogram equalization

```
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('F:/ sample.jpg', 0)
eq = cv2.equalizeHist(img)
```

 # This line applies histogram equalization to the input image using cv2.equalizeHist. Histogram equalization enhances the contrast of an image by redistributing the pixel intensities to cover the full range of intensity values.

```
hist = cv2.calcHist([img], [0], None, [256], [0, 256])
histeq = cv2.calcHist([eq], [0], None, [256], [0, 256])
```

 # These lines calculate the histograms of the original image (img) and the equalized image (eq)

```
plt.subplot(221), plt.imshow(img, 'gray')
plt.subplot(222), plt.plot(hist)
plt.subplot(223), plt.imshow(eq, 'gray')
plt.subplot(224), plt.plot(histeq)
plt.xlim([0, 256])
```

 # These lines set the x-axis limits of the histogram plots to range from 0 to 256

```
plt.show()
```

4D. Thresholding, and halftoning operations

```
#4D. Thresholding, and halftoning operations
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('F:/ sunflower.jpg',0)
```

```python
# These lines perform different thresholding operations on the grayscale image using cv.threshold.
ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
ret,thresh2 = cv.threshold(img,127,255,cv.THRESH_BINARY_INV)
ret,thresh3 = cv.threshold(img,127,255,cv.THRESH_TRUNC)
ret,thresh4 = cv.threshold(img,127,255,cv.THRESH_TOZERO)
ret,thresh5 = cv.threshold(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()
```

```python
#erosion

import cv2
import matplotlib.pyplot as plt

# Read the image
image = cv2.imread("C://banana.jpg", 0)
```

```
# Define the kernel for erosion
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (24, 24))

# Apply erosion
eroded_image = cv2.erode(image, kernel, iterations=1)

# Display the original and eroded images
plt.subplot(121)
plt.imshow(image, cmap='gray')
plt.title('Original Image')

plt.subplot(122)
plt.imshow(eroded_image, cmap='gray')
plt.title('Eroded Image')

plt.tight_layout()
plt.show()
```

```
#dialation

import cv2
import matplotlib.pyplot as plt
```

```python
# Read the image
image = cv2.imread("C:\banana.jpg", 0)

# Define the kernel for dilation
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (24, 24))

# Apply dilation
dilated_image = cv2.dilate(image, kernel, iterations=1)

# Display the original and dilated images
plt.subplot(121)
plt.imshow(image, cmap='gray')
plt.title('Original Image')

plt.subplot(122)
plt.imshow(dilated_image, cmap='gray')
plt.title('Dilated Image')
plt.show()
```

```python
#opening

import cv2
import matplotlib.pyplot as plt
```

```python
# Read the image
image = cv2.imread("path_to_image.jpg", 0)

# Define the kernel for opening
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))

# Apply opening
opened_image = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)

# Display the original and opened images
plt.subplot(121)
plt.imshow(image, cmap='gray')
plt.title('Original Image')

plt.subplot(122)
plt.imshow(opened_image, cmap='gray')
plt.title('Opened Image')

plt.tight_layout()
plt.show()
```

```python
#closing

import cv2
import matplotlib.pyplot as plt

# Read the image
image = cv2.imread("path_to_image.jpg", 0)

# Define the kernel for closing
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))

# Apply closing
closed_image = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernel)

# Display the original and closed images
plt.subplot(121)
plt.imshow(image, cmap='gray')
plt.title('Original Image')

plt.subplot(122)
plt.imshow(closed_image, cmap='gray')
plt.title('Closed Image')

plt.tight_layout()
plt.show()
```