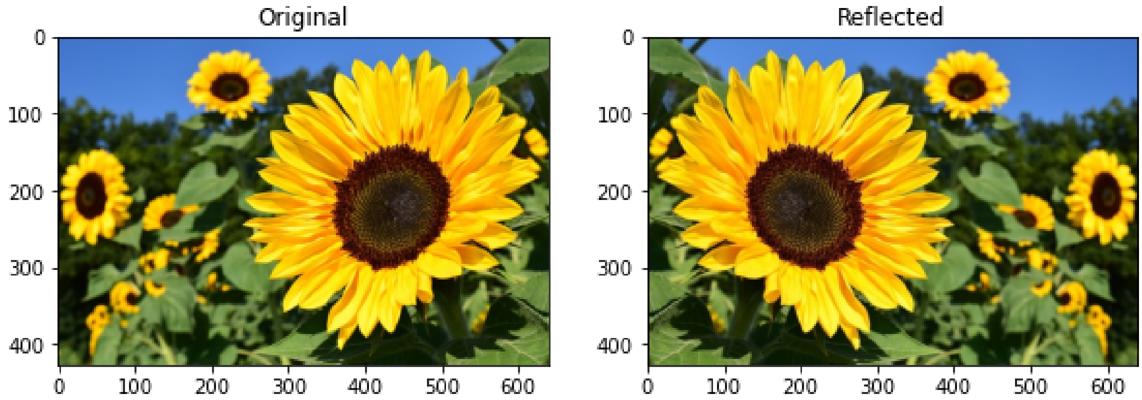
# Built in functions

## Reflection

In [ ]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt


img = cv2.imread('sunflower.jpg')

gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

rows, cols = gray_img.shape
#M = np.float32([[1, 0, 0],[0, -1, rows],[0, 0, 1]])    # To flip the image
M = np.float32([[-1, 0, cols], [0, 1, 0], [0, 0, 1]])  # To flip the image
reflected_img = cv.warpPerspective(img, M,(int(cols),int(rows)))


plt.figure(figsize = (10, 13))
plt.subplot(1,2,1)
plt.imshow(rgb_img)
plt.title("Original")

cv2.imwrite('reflection_out.jpg', reflected_img)

img = cv2.imread('reflection_out.jpg')

rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.subplot(1,2,2)
plt.imshow(rgb_img)
plt.title("Reflected")
```

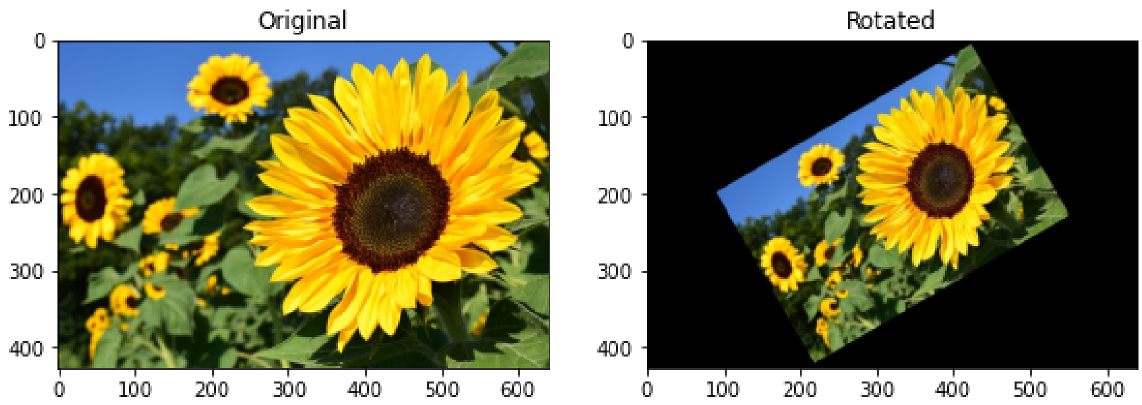
Out [28]: Text(0.5, 1.0, 'Reflected')



## Rotation

In [ ]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt


img = cv2.imread('sunflower.jpg')

gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

rows, cols = gray_img.shape
M = np.float32([[1, 0, 0], [0, -1, rows], [0, 0, 1]])
img_rotation = cv2.warpAffine(img,cv.getRotationMatrix2D((cols/2, rows/2),

plt.figure(figsize = (10, 13))
plt.subplot(1,2,1)
plt.imshow(rgb_img)
plt.title("Original")

cv2.imwrite('rotation_out.jpg', img_rotation)

img = cv2.imread('rotation_out.jpg')

rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.subplot(1,2,2)
plt.imshow(rgb_img)
plt.title("Rotated")
```

Out [27]: `Text(0.5, 1.0, 'Rotated')`




## Resize

In [ ]:
```
# Import the necessary libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```python
# Load the image
image =cv2.imread('sunflower.jpg')

# Convert BGR image to RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Define the scale factor
# Increase the size by 3 times
scale_factor_1 = 3.0
# Decrease the size by 3 times
scale_factor_2 = 1/3.0

# Get the original image dimensions
height, width = image_rgb.shape[:2]

# Calculate the new image dimensions
new_height = int(height * scale_factor_1)
new_width = int(width * scale_factor_1)

# Resize the image
zoomed_image = cv2.resize(src =image_rgb,
                          dsize=(new_width, new_height),
                          interpolation=cv2.INTER_CUBIC)

# Calculate the new image dimensions
new_height1 = int(height * scale_factor_2)
new_width1 = int(width * scale_factor_2)

# Scaled image
scaled_image = cv2.resize(src= image_rgb,
                          dsize =(new_width1, new_height1),
                          interpolation=cv2.INTER_AREA)

# Create subplots
fig, axs = plt.subplots(1, 3, figsize=(10, 4))

# Plot the original image
axs[0].imshow(image_rgb)
axs[0].set_title('Original Image Shape:'+str(image_rgb.shape))

# Plot the Zoomed Image
axs[1].imshow(zoomed_image)
axs[1].set_title('Zoomed Image Shape:'+str(zoomed_image.shape))

# Plot the Scaled Image
axs[2].imshow(scaled_image)
axs[2].set_title('Scaled Image Shape:'+str(scaled_image.shape))

# Remove ticks from the subplots
for ax in axs:
    ax.set_xticks([])
    ax.set_yticks([])
```

```
# Display the subplots
plt.tight_layout()
plt.show()
```


## Image shearing x - axis

In [ ]:

```
import numpy as np
import cv2
img = cv2.imread('dog.jpg',0)
rows, cols = img.shape
M = np.float32([[1, 0.5, 0], [0, 1, 0], [0, 0, 1]])
sheared_img = cv2.warpPerspective(img, M, (int(cols*1.5), int(rows*1.5)))
cv2.imshow('img', sheared_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## Image shearing y - axis

In [ ]:

```
import numpy as np
import cv2
img = cv2.imread('dog.jpg',0)
rows, cols = img.shape
M = np.float32([[1, 0, 0], [0.5, 1, 0], [0, 0, 1]])
sheared_img = cv2.warpPerspective(img, M, (int(cols*1.5), int(rows*1.5)))
cv2.imshow('sheared_y-axis_out.jpg', sheared_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## Cropping

In [ ]:

```
import numpy as np
import cv2 as cv
img = cv2.imread('dog.jpg',0)
cropped_img = img[100:300, 100:300]
cv.imwrite('cropped_out.jpg', cropped_img)
cv.imshow('cropped_out', cropped_img)
cv.waitKey(0)
cv.destroyAllWindows()
```

## Blurring

In [ ]:

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

image =  cv2.imread('img2.jpg')

image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

blurred = cv2.GaussianBlur(image, (3, 3), 0)

blurred_rgb = cv2.cvtColor(blurred, cv2.COLOR_BGR2RGB)

fig, axs = plt.subplots(1, 2, figsize=(13, 12))

axs[0].imshow(image_rgb)
axs[0].set_title('Original Image')

axs[1].imshow(blurred_rgb)
axs[1].set_title('Blurred Image')

for ax in axs:
    ax.set_xticks([])
    ax.set_yticks([])

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

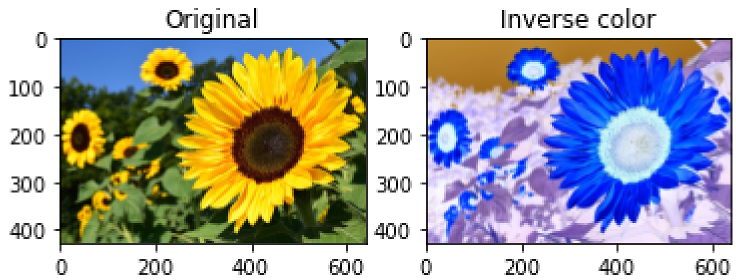

# Without built-in

## Inverse transform

In [ ]:

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

```python
# Load the image
image = cv2.imread('sunflower.jpg')
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

#Plot the original image
plt.subplot(1, 2, 1)
plt.title("Original")
plt.imshow(image_rgb)

# Inverse by subtracting from 255
inverse_image = 255 - image

#Save the image
cv2.imwrite('inverse_image.jpg', inverse_image)

img = cv2.imread("inverse_image.jpg")
image_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#Plot the Inverse image
plt.subplot(1, 2, 2)
plt.title("Inverse color")
plt.imshow(image_rgb)
plt.show()
```



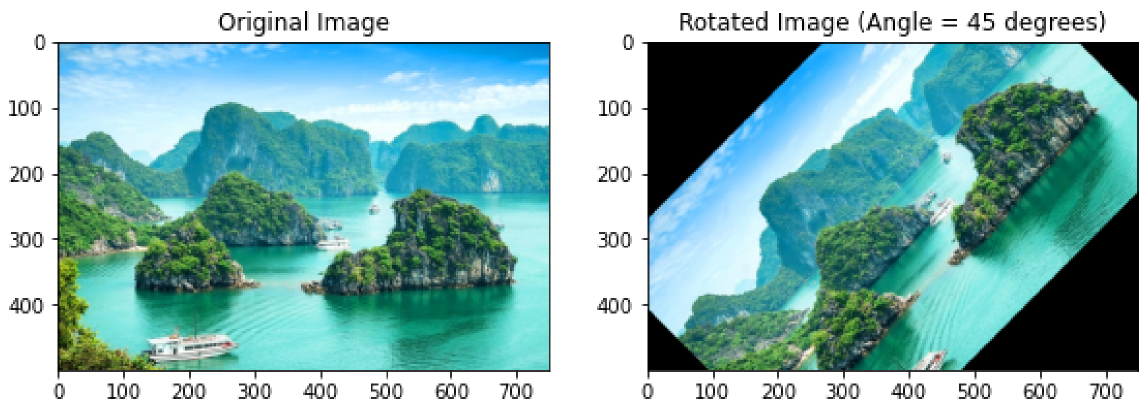

## Rotation

In [ ]:

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

def rotate_image(image, angle):
    # Get the dimensions of the original image
    height, width, channels = image.shape

    # Convert the angle from degrees to radians
    theta = math.radians(angle)

    # Calculate the rotation matrix
    rotation_matrix = np.array([
        [math.cos(theta), -math.sin(theta)],
        [math.sin(theta), math.cos(theta)]
```

```
    ])

    # Calculate the coordinates of the center of the image
    center_x = width / 2
    center_y = height / 2

    # Create an empty image to store the rotated image
    rotated_image = np.zeros_like(image)

    # Perform rotation by iterating over each pixel in the rotated image
    for y in range(height):
        for x in range(width):
            # Calculate the coordinates of the pixel relative to the cente
            x_rel = x - center_x
            y_rel = y - center_y

            # Apply the rotation transformation
            x_rotated, y_rotated = np.dot(rotation_matrix, [x_rel, y_rel])

            # Translate the rotated coordinates back to the original image
            x_rotated += center_x
            y_rotated += center_y

            # Round the rotated coordinates to the nearest integer
            x_rotated = int(round(x_rotated))
            y_rotated = int(round(y_rotated))

            # Check if the rotated coordinates are within the bounds of th
            if 0 <= x_rotated < width and 0 <= y_rotated < height:
                # Assign the pixel value from the original image to the ro
                rotated_image[y, x] = image[y_rotated, x_rotated]

    return rotated_image

# Load the input image
image = cv2.imread('img2.jpg')

# Convert BGR image to RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Define the angle of rotation in degrees (positive values for counter-clo
angle = 45

# Rotate the image using the custom function
rotated_image = rotate_image(image_rgb, angle)

# Display the original and rotated images
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image_rgb)
plt.title('Original Image')

plt.subplot(1, 2, 2)
```

```
plt.imshow(rotated_image)
plt.title('Rotated Image (Angle = {} degrees)'.format(angle))

plt.show()
```




## Scaling

In [ ]:

```
import cv2
import numpy as np

def scale_image(image, scale_factor):
    # Get the dimensions of the original image
    height, width, channels = image.shape

    # Calculate the new dimensions after scaling
    new_height = int(height * scale_factor)
    new_width = int(width * scale_factor)

    # Create an empty image to store the scaled image
    scaled_image = np.zeros((new_height, new_width, channels), dtype=np.ui

    # Iterate over each pixel in the scaled image
    for y in range(new_height):
        for x in range(new_width):
            # Calculate the corresponding pixel coordinates in the origina
            x_original = x / scale_factor
            y_original = y / scale_factor

            # Find the integer and fractional parts of the coordinates
            x_int = int(x_original)
            y_int = int(y_original)
            x_frac = x_original - x_int
            y_frac = y_original - y_int

            # Ensure that the coordinates are within the bounds of the ori
            if 0 <= x_int < width - 1 and 0 <= y_int < height - 1:
                # Perform bilinear interpolation
                top_left = image[y_int, x_int] * (1 - x_frac) * (1 - y_fra
                top_right = image[y_int, x_int + 1] * x_frac * (1 - y_frac
                bottom_left = image[y_int + 1, x_int] * (1 - x_frac) * y_f
```

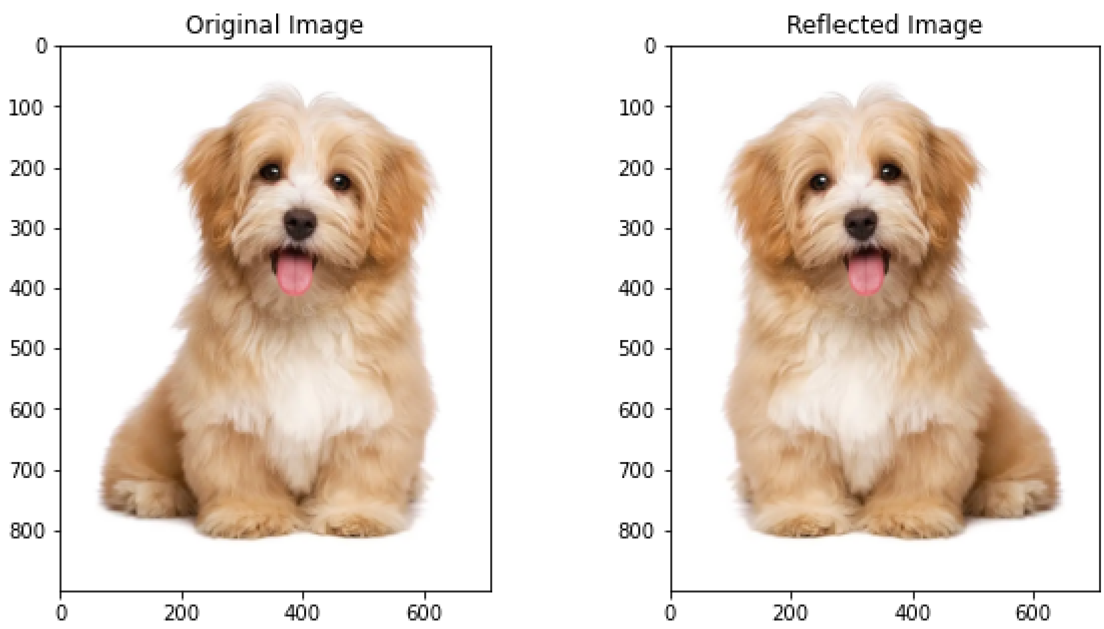
```
                bottom_right = image[y_int + 1, x_int + 1] * x_frac * y_fr

                # Assign the interpolated pixel value to the scaled image
                scaled_image[y, x] = top_left + top_right + bottom_left +

    return scaled_image

# Load the input image
image = cv2.imread('sunflower.jpg')

# Define the scale factor
scale_factor = 1.5  # Increase the size by 1.5 times

# Scale the image using the custom function
scaled_image = scale_image(image, scale_factor)

# Display the original and scaled images using cv2.imshow()
cv2.imshow('Original Image', image)
cv2.imshow('Scaled Image', scaled_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## Reflection

In [ ]:

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

def reflect_image(image, axis):
    # Get the dimensions of the original image
    height, width, channels = image.shape

    # Create an empty image to store the reflected image
    reflected_image = np.zeros_like(image)

    # Perform reflection along the specified axis
    if axis == 0:  # Vertical reflection
        for y in range(height):
            for x in range(width):
                reflected_image[height - y - 1, x] = image[y, x]
    elif axis == 1:  # Horizontal reflection
        for y in range(height):
            for x in range(width):
                reflected_image[y, width - x - 1] = image[y, x]

    return reflected_image

# Load the input image
image = cv2.imread('dog.jpg')

# Convert BGR image to RGB
```

```
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Define the axis of reflection (0 for vertical reflection, 1 for horizont
axis = 1  # Horizontal reflection

# Reflect the image using the custom function
reflected_image = reflect_image(image_rgb, axis)

# Display the original and reflected images
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image_rgb)
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(reflected_image)
plt.title('Reflected Image')

plt.show()
```

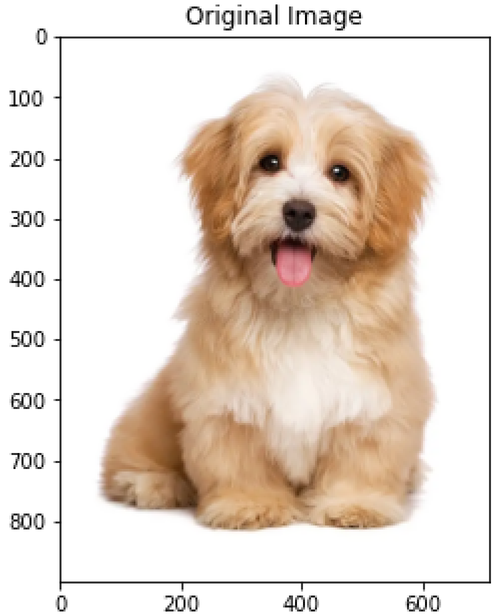


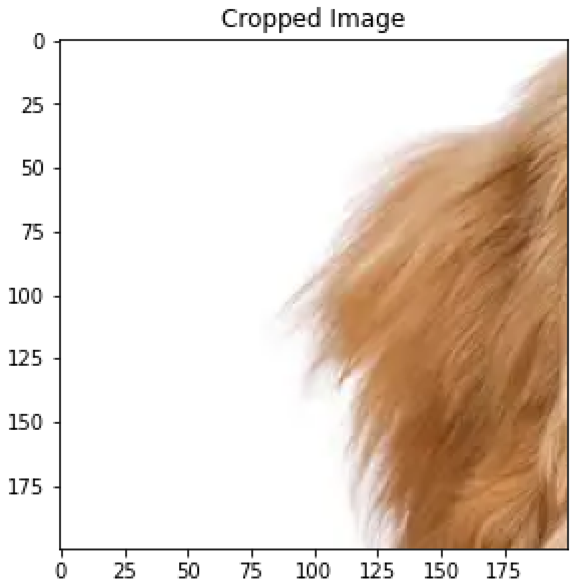

## Cropping

In [ ]:

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

def crop_image(image, x, y, w, h):
    """
    Crop the input image to the specified region of interest (ROI).

    Parameters:
        image: numpy.ndarray
            Input image.
        x: int
```

```
            X-coordinate of the top-left corner of the ROI.
        y: int
            Y-coordinate of the top-left corner of the ROI.
        w: int
            Width of the ROI.
        h: int
            Height of the ROI.

    Returns:
        numpy.ndarray
            Cropped image.
    """
    # Extract the region of interest (ROI) from the input image
    cropped_image = image[y:y+h, x:x+w]

    return cropped_image

# Load the input image
image = cv2.imread('dog.jpg')

# Convert BGR image to RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Define the region of interest (ROI) for cropping
x = 100   # X-coordinate of the top-left corner of the ROI
y = 100   # Y-coordinate of the top-left corner of the ROI
w = 200   # Width of the ROI
h = 200   # Height of the ROI

# Crop the image using the custom function
cropped_image = crop_image(image_rgb, x, y, w, h)

# Display the original and cropped images
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image_rgb)
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(cropped_image)
plt.title('Cropped Image')

plt.show()
```




## Blurring

In [ ]:

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

def gaussian_kernel(size, sigma=1):
    kernel = np.fromfunction(lambda x, y: (1/(2*np.pi*sigma**2)) * np.exp(
    return kernel / np.sum(kernel)

def convolution(image, kernel):
    m, n, _ = image.shape
    y, x = kernel.shape
    y = y // 2
    x = x // 2
    new_image = np.zeros_like(image)
    for i in range(y, m - y):
        for j in range(x, n - x):
            for k in range(image.shape[2]):  # Iterate over channels
                new_image[i, j, k] = np.sum(image[i-y:i+y+1, j-x:j+x+1, k]
    return new_image

def blur_image(image, kernel_size):
    kernel = gaussian_kernel(kernel_size)
    blurred_image = convolution(image, kernel)
    return blurred_image

# Load the image
image = plt.imread('sunflower.jpg')

# Apply Gaussian blur
blurred_image = blur_image(image, kernel_size=5)

# Plot the original and blurred images
plt.figure(figsize=(10, 5))
```

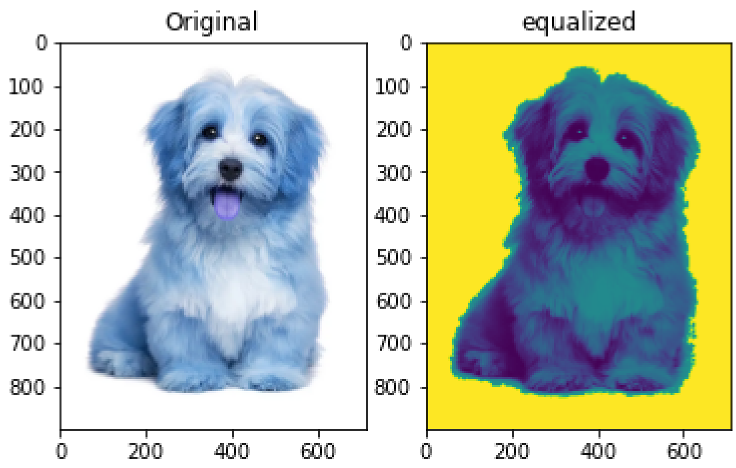
```
# Original image
plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title('Original Image')
plt.axis('off')

# Blurred image
plt.subplot(1, 2, 2)
plt.imshow(blurred_image.astype(np.uint8))
plt.title('Blurred Image')
plt.axis('off')

plt.show()
```


# Histogram Eualization

In [ ]:

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

# Load the image
image = cv2.imread('dog.jpg')

#Plot the original image
plt.subplot(1, 2, 1)
plt.title("Original")
plt.imshow(image)

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Equalize the histogram
equalized_image = cv2.equalizeHist(gray_image)

#Save the equalized image
cv2.imwrite('equalized.jpg', equalized_image)
```

```
#Plot the equalized image
plt.subplot(1, 2, 2)
plt.title("equalized")
plt.imshow(equalized_image)
plt.show()
```




# mean , median, and correlation coefficient

In [ ]:
```
import cv2
import numpy as np

# Read the image
img = cv2.imread("img2.jpg", cv2.IMREAD_GRAYSCALE)

# Calculate mean and median
mean_value = np.mean(img)

median_value = np.median(img)

# Calculate correlation coefficient
correlation_coefficient = np.corrcoef(img)[0, 1]

# Display results
print("Mean: {:.2f}".format(mean_value))
print("Median: {:.2f}".format(median_value))
print("Correlation Coefficient: {:.2f}".format(correlation_coefficient))
```

```
Mean: 150.33
Median: 151.00
Correlation Coefficient: 0.99
```

# Edge detection

In [ ]:
```
# Import the necessary Libraries
import cv2
import numpy as np
```

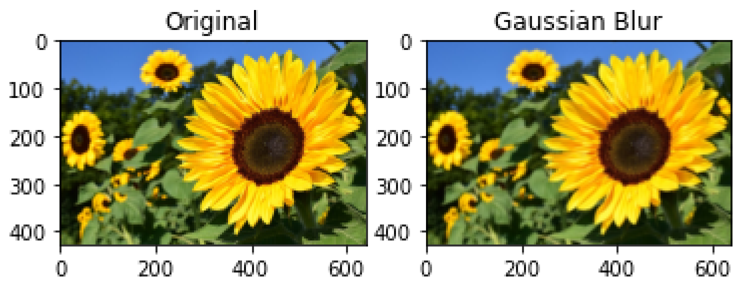
```python
import matplotlib.pyplot as plt

# Read image from disk.
img = cv2.imread('sunflower.jpg',0)
# Convert BGR image to RGB
image_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Apply Canny edge detection
edges = cv2.Canny(image= image_rgb, threshold1=100, threshold2=700)

# Create subplots
fig, axs = plt.subplots(1, 2, figsize=(7, 4))

# Plot the original image
axs[0].imshow(image_rgb)
axs[0].set_title('Original Image')

# Plot the blurred image
axs[1].imshow(edges)
axs[1].set_title('Image edges')

# Remove ticks from the subplots
for ax in axs:
    ax.set_xticks([])
    ax.set_yticks([])

# Display the subplots
plt.tight_layout()
plt.show()
```

# Gaussian blur

In [ ]:

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

# Load the image
image = cv2.imread('sunflower.jpg')
# Convert BGR image to RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

#Plot the original image
plt.subplot(1, 2, 1)
plt.title("Original")
plt.imshow(image_rgb)

# Remove noise using a Gaussian filter
filtered_image2 = cv2.GaussianBlur(image, (7, 7), 0)

#Save the image
```

```
cv2.imwrite('Gaussian Blur.jpg', filtered_image2)

img = cv2.imread("Gaussian Blur.jpg")
image_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#Plot the blured image
plt.subplot(1, 2, 2)
plt.title("Gaussian Blur")
plt.imshow(image_rgb)
plt.show()
```

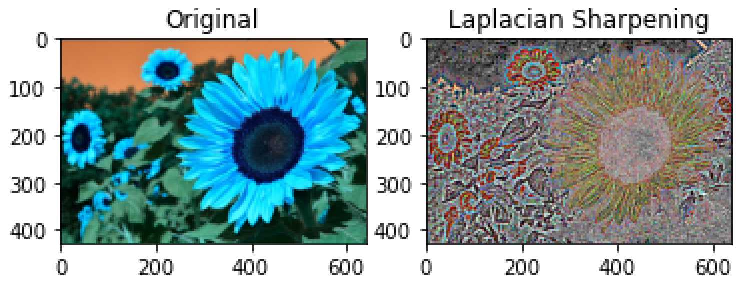



# Laplacian sharpening

In [ ]:

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

# Load the image
image = cv2.imread('sunflower.jpg')

#Plot the original image
plt.subplot(1, 2, 1)
plt.title("Original")
plt.imshow(image)

# Sharpen the image using the Laplacian operator
sharpened_image2 = cv2.Laplacian(image, cv2.CV_64F)

#Save the image
cv2.imwrite('Laplacian sharpened_image.jpg', sharpened_image2)

#Plot the sharpened image
plt.subplot(1, 2, 2)
plt.title("Laplacian Sharpening")
plt.imshow(sharpened_image2)
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```




# Normalisation

In [ ]:

```
# Import the necessary Libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread('sunflower.jpg')

# Convert BGR image to RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Split the image into channels
b, g, r = cv2.split(image_rgb)

# Normalization parameter
min_value = 0
max_value = 1
norm_type = cv2.NORM_MINMAX

# Normalize each channel
b_normalized = cv2.normalize(b.astype('float'), None, min_value, max_value
g_normalized = cv2.normalize(g.astype('float'), None, min_value, max_value
r_normalized = cv2.normalize(r.astype('float'), None, min_value, max_value

# Merge the normalized channels back into an image
normalized_image = cv2.merge((b_normalized, g_normalized, r_normalized))
# Normalized image
print(normalized_image[:,:,0])

plt.imshow(normalized_image)
plt.xticks([])
plt.yticks([])
plt.title('Normalized Image')
plt.show()
```

```
[[0.25490196 0.25490196 0.25490196 ... 0.3372549  0.33333333 0.32941176]
 [0.25490196 0.25490196 0.25490196 ... 0.33333333 0.32941176 0.32156863]
 [0.25490196 0.25490196 0.25490196 ... 0.33333333 0.32941176 0.3254902 ]
 ...
 [0.34117647 0.34901961 0.35686275 ... 0.21960784 0.17254902 0.1254902 ]
 [0.34117647 0.34901961 0.35294118 ... 0.28627451 0.26666667 0.23137255]
 [0.34117647 0.34901961 0.36078431 ... 0.24313725 0.24705882 0.21568627]]
```

Normalized Image

```
In [ ]:
```