1) Using built-in function

a) Reflection

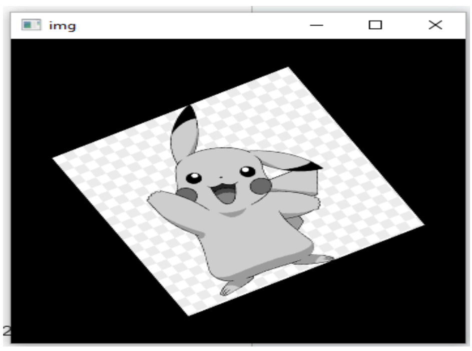
```
import numpy as np
import cv2 as cv
img = cv.imread('pikachu.png',0)
rows, cols = img.shape
M = np.float32([[1, 0, 0],[0, -1, rows],[0, 0, 1]])
reflected_img = cv.warpPerspective(img, M,(int(cols),int(rows)))
cv.imshow('img', reflected_img)
cv.imwrite('reflection_out.jpg', reflected_img)
cv.waitKey(0)
cv.destroyAllWindows()
```

Output:

b) Rotation

```
import numpy as np
import cv2 as cv
img = cv.imread('pikachu.png',0)
rows, cols = img.shape
M = np.float32([[1, 0, 0], [0, -1, rows], [0, 0, 1]])
img_rotation = cv.warpAffine(img,cv.getRotationMatrix2D((cols/2, rows/2),30, 0.6),(cols,
rows))
cv.imshow('img', img_rotation)
cv.imwrite('rotation_out.jpg', img_rotation)
cv.waitKey(0)
cv.destroyAllWindows()
```

Output:

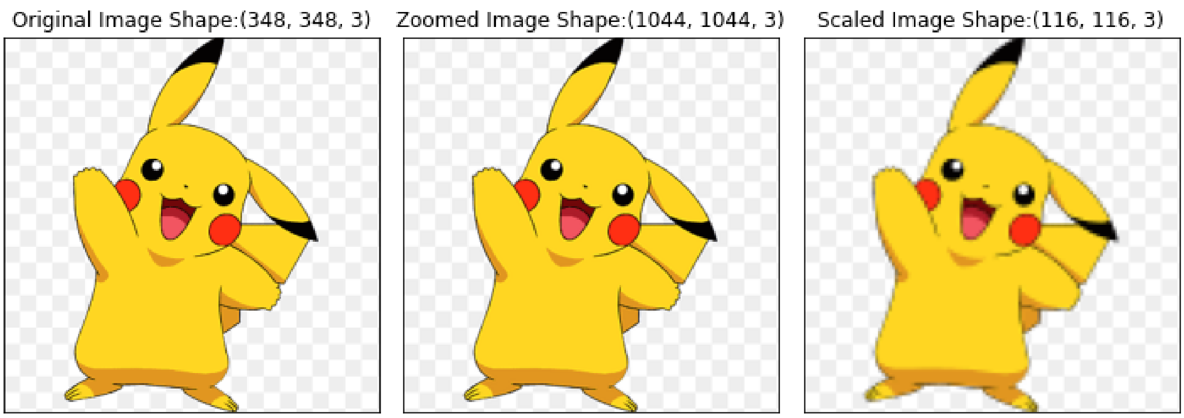


c) Resize

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

# Load the image
image =cv2.imread('pikachu.png')

# 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()
```

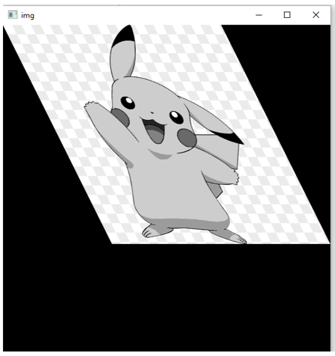
Output:



d) Shearing
i) X-axis

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

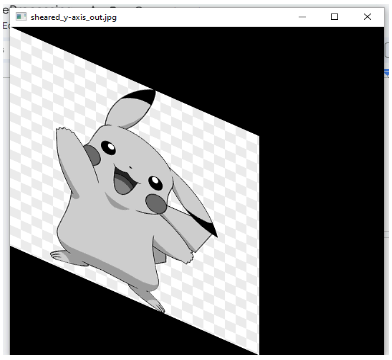
Output:



ii) Y-axis

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

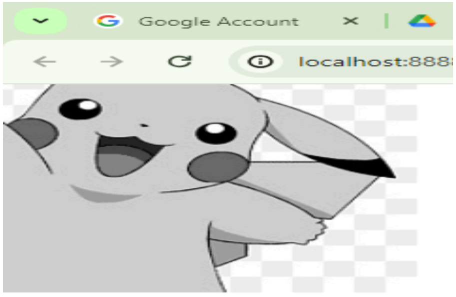
Output:



e) Cropping

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

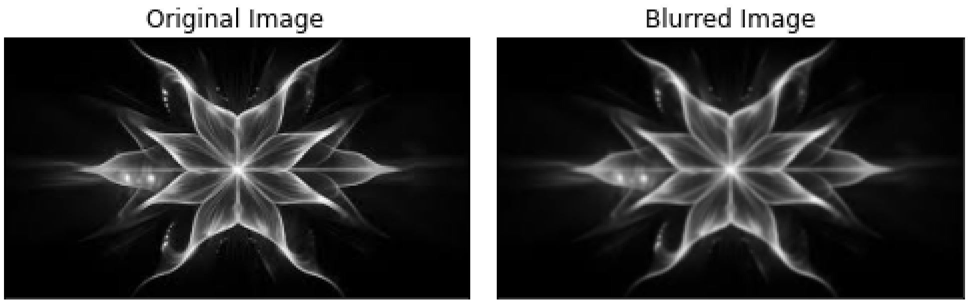
Output:



f) Blurring

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

# Load the image
image =  cv2.imread('xyz.jpg',0)

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

# Apply Gaussian blur
blurred = cv2.GaussianBlur(image, (3, 3), 0)

# Convert blurred image to RGB
blurred_rgb = cv2.cvtColor(blurred, cv2.COLOR_BGR2RGB)

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

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

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

Output:

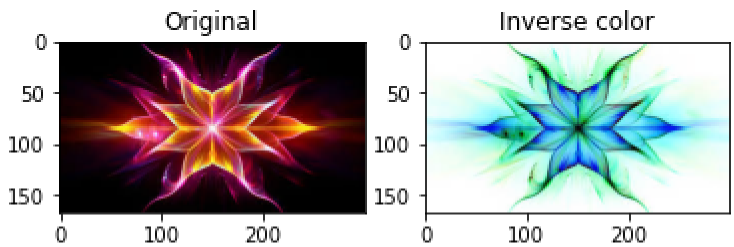


2) Without using built-in function

a) Transform (inverse)

```
import cv2
import matplotlib.pyplot as plt

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

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

# Inverse by subtracting from 255
inverse_image = 255 - image

#Save the image
cv2.imwrite('inverse_image.jpg', inverse_image)
#Plot the Inverse image
plt.subplot(1, 2, 2)
plt.title("Inverse color")
plt.imshow(inverse_image)
plt.show()
```

Output:

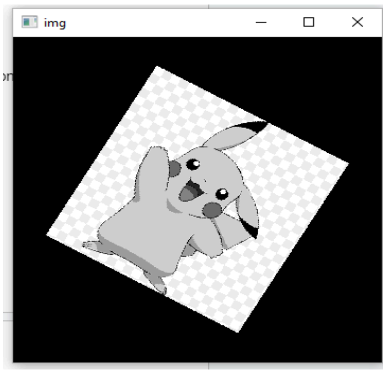


b) Rotation

```
import numpy as np
import cv2 as cv

# Load the image
img = cv.imread('pikachu.png', 0)

# Get image dimensions
rows, cols = img.shape

# Define rotation parameters
angle = 30
scale = 0.6

# Convert angle to radians
theta = np.radians(angle)

# Define rotation matrix
rotation_matrix = np.array([
    [np.cos(theta) * scale, -np.sin(theta) * scale, 0],
    [np.sin(theta) * scale, np.cos(theta) * scale, 0],
    [0, 0, 1]
])

# Define translation matrix to rotate about the center
translation_matrix = np.array([
    [1, 0, -cols / 2],
    [0, 1, -rows / 2],
    [0, 0, 1]
])

# Define inverse translation matrix
inv_translation_matrix = np.array([
    [1, 0, cols / 2],
    [0, 1, rows / 2],
    [0, 0, 1]
])

# Combine the transformations
transformation_matrix = np.dot(inv_translation_matrix, np.dot(rotation_matrix,
translation_matrix))

# Apply transformation to each pixel
img_rotation = np.zeros_like(img)
for y in range(rows):
    for x in range(cols):
```

```
            new_x, new_y, _ = np.dot(transformation_matrix, [x, y, 1])
            new_x, new_y = int(new_x), int(new_y)
            if 0 <= new_x < cols and 0 <= new_y < rows:
                img_rotation[new_y, new_x] = img[y, x]

    # Display and save the rotated image
    cv.imshow('img', img_rotation)
    cv.imwrite('rotation_out.jpg', img_rotation)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

Output:

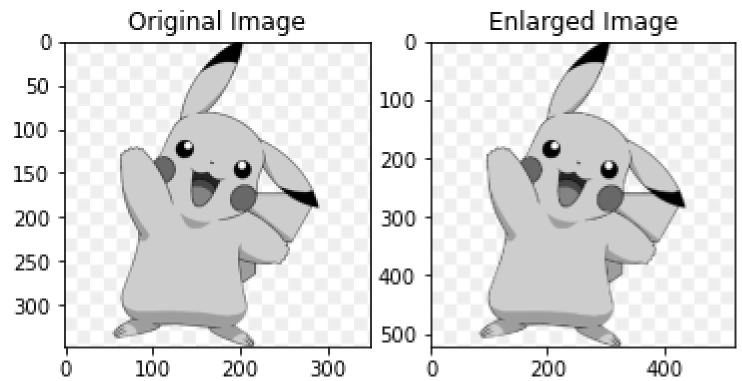


c) Scaling

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

# Load the image
img = cv2.imread('pikachu.png', 0)

# Define the scale factor
scale_factor = 1.5

# Calculate the new dimensions
new_height = int(img.shape[0] * scale_factor)
new_width = int(img.shape[1] * scale_factor)

# Initialize the new image
img_enlarged = np.zeros((new_height, new_width), dtype=np.uint8)

# Calculate the scale factors
scale_y = img.shape[0] / new_height
scale_x = img.shape[1] / new_width

# Iterate over each pixel in the new image
for y in range(new_height):
    for x in range(new_width):
```

```
        # Map the pixel coordinate in the new image to the original image
        original_x = int(x * scale_x)
        original_y = int(y * scale_y)

        # Clamp the original coordinates to stay within the original image boundaries
        original_x = max(0, min(original_x, img.shape[1] - 1))
        original_y = max(0, min(original_y, img.shape[0] - 1))

        # Assign the pixel value from the original image to the corresponding pixel in
the new image
        img_enlarged[y, x] = img[original_y, original_x]

# Plot the original and enlarged images
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(img, cmap='gray')

plt.subplot(1, 2, 2)
plt.title('Enlarged Image')
plt.imshow(img_enlarged, cmap='gray')

plt.show()
```

Output:



d) Reflection

```
import cv2 as cv

# Load the image
img = cv.imread('pikachu.png', 0)
rows, cols = img.shape

# Create a blank image to store the reflected image
reflected_img = np.zeros_like(img)

# Define the reflection matrix
```

```
M = np.float32([[1, 0, 0],
                [0, -1, rows],
                [0, 0, 1]])

# Apply the perspective transformation manually
for y in range(rows):
    for x in range(cols):
        # Apply the transformation matrix to get the new coordinates
        new_x, new_y, _ = np.dot(M, [x, y, 1])

        # Ensure new coordinates are within bounds
        new_x = int(np.clip(new_x, 0, cols - 1))
        new_y = int(np.clip(new_y, 0, rows - 1))

        # Assign the pixel value from the original image to the reflected image
        reflected_img[new_y, new_x] = img[y, x]

# Display the reflected image
cv.imshow('img', reflected_img)
cv.imwrite('reflection_out.jpg', reflected_img)
cv.waitKey(0)
cv.destroyAllWindows()
```

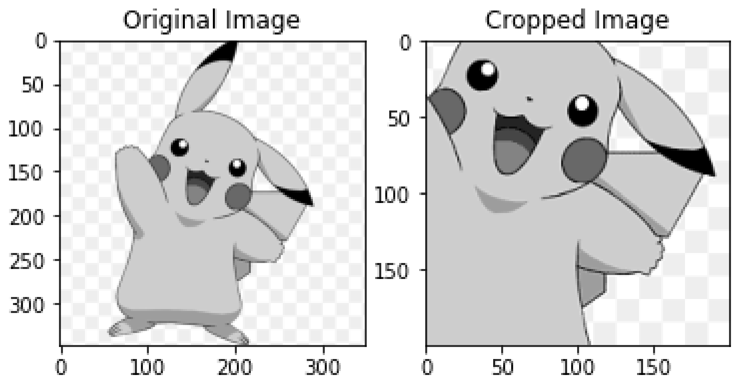
Output:

e) Cropping

```
import cv2 as cv
import matplotlib.pyplot as plt

# Load the image
img = cv.imread('pikachu.png', 0)

# Define the region to be cropped
top_left_x, top_left_y = 100, 100
bottom_right_x, bottom_right_y = 300, 300

# Crop the image
cropped_img = img[top_left_y:bottom_right_y, top_left_x:bottom_right_x]
```

```
# Plot the original and cropped images
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(img, cmap='gray')

plt.subplot(1, 2, 2)
plt.title('Cropped Image')
plt.imshow(cropped_img, cmap='gray')

plt.show()
```

Output:

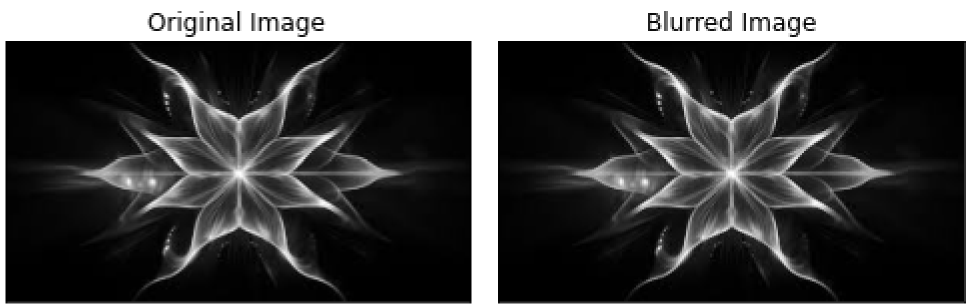


f) Blurring

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

def gaussian_kernel(size, sigma):
    epsilon = 1e-5  # Small value to avoid division by zero
    kernel = np.fromfunction(lambda x, y: (1/(2*np.pi*(sigma**2 + epsilon))) *
np.exp(-((x-size//2)**2 + (y-size//2)**2)/(2*(sigma**2 + epsilon))), (size, size))
    return kernel / np.sum(kernel)

def custom_gaussian_blur(image, kernel_size, sigma):
    kernel = gaussian_kernel(kernel_size, sigma)
    blurred_image = cv2.filter2D(image, -1, kernel)
    return blurred_image

# Load the image
image = cv2.imread('xyz.jpg', 0)

# Apply custom Gaussian blur
blurred = custom_gaussian_blur(image, 3, 0)  # You can adjust the kernel size and
sigma as needed
```

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

# Plot the original image
axs[0].imshow(image, cmap='gray')
axs[0].set_title('Original Image')

# Plot the blurred image
axs[1].imshow(blurred, cmap='gray')
axs[1].set_title('Blurred Image')

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

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

Output:

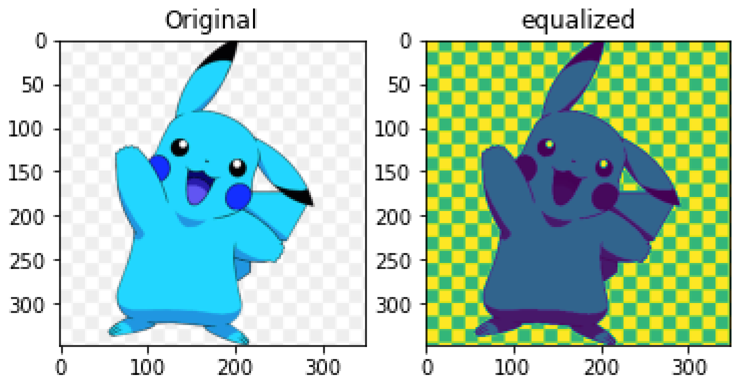


3) Histogram equalisation

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

# Load the image
image = cv2.imread('pikachu.png')

#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()
```

Output:



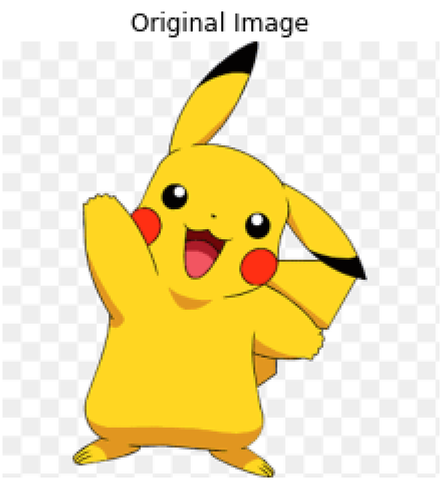
4) Mean, median, coefficient

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

# Accept user input for the image file path
image_path = input("Enter the path to the image file: ")

# Read the image
image = cv2.imread(image_path)

# Check if the image was read successfully
if image is None:
    print("Error: Unable to read the image.")
    exit()

# Convert the image to grayscale if it's a color image
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Calculate mean and standard deviation
mean_val = np.mean(gray_image)
std_dev = np.std(gray_image)

# Calculate the correlation coefficient
```

```
correlation = np.corrcoef(gray_image.flatten())

# Calculate the coefficient of variation
coefficient_of_variation = std_dev / mean_val

# Print the results
print("Mean:", mean_val)
print("Standard Deviation:", std_dev)
print("Correlation:", correlation)
print("Coefficient of Variation:", coefficient_of_variation)

# Display the image
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.title("Original Image")
plt.show()
```

Output:

```
Enter the path to the image file: pikachu.png
Mean: 224.27519322235435
Standard Deviation: 44.18586242525577
Correlation: 1.0
Coefficient of Variation: 0.19701627179715925
```

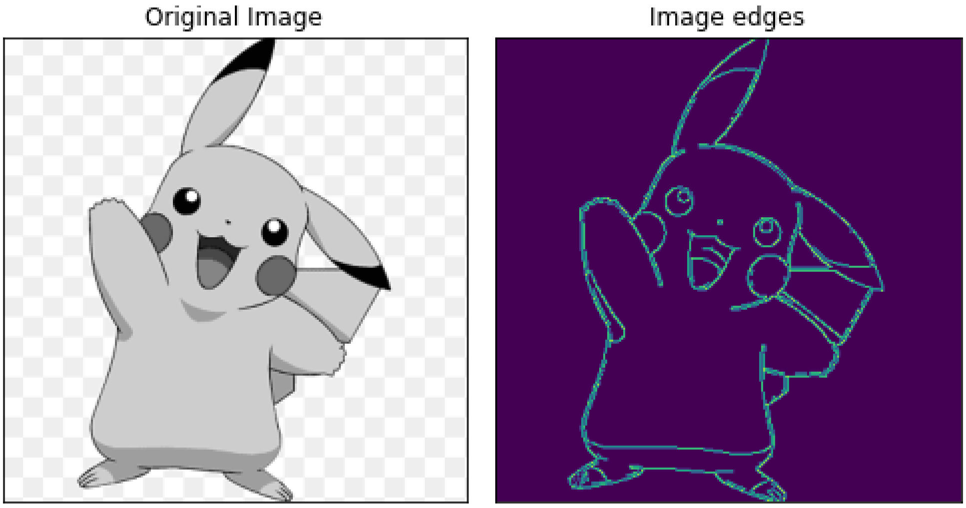


5) Edge detection

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

# Read image from disk.
img = cv2.imread('pikachu.png',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()
```

Output:

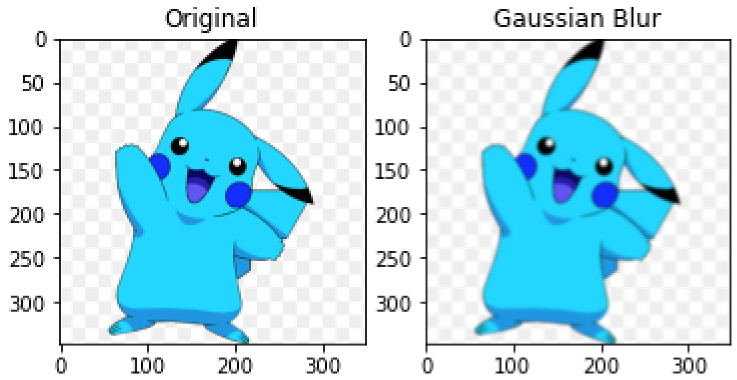


6) Gaussian blur

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

# Load the image
image = cv2.imread('pikachu.png')

#Plot the original image
plt.subplot(1, 2, 1)
```

```
plt.title("Original")
plt.imshow(image)

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

#Save the image
cv2.imwrite('Gaussian Blur.jpg', filtered_image2)

#Plot the blured image
plt.subplot(1, 2, 2)
plt.title("Gaussian Blur")
plt.imshow(filtered_image2)
plt.show()
```

Output:

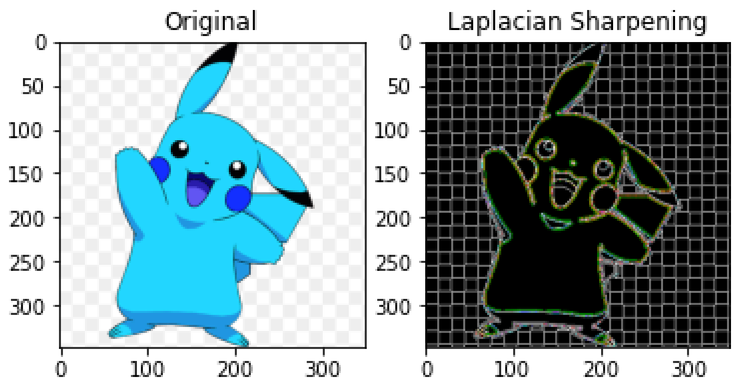


Laplcian sharpening

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

# Load the image
image = cv2.imread('pikachu.png')

#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()
```

Output:



7) Normalisation the image

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

# Load the image
image = cv2.imread('pikachu.png')

# 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,
norm_type)
g_normalized = cv2.normalize(g.astype('float'), None, min_value, max_value,
norm_type)
r_normalized = cv2.normalize(r.astype('float'), None, min_value, max_value,
norm_type)

# 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()
```

Output:

```
[[1.         1.         1.         ... 0.93227092 0.93227092 0.93227092]
 [1.         1.         1.         ... 0.93227092 0.93227092 0.93227092]
 [1.         1.         1.         ... 0.93227092 0.93227092 0.93227092]
 ...
 [0.93227092 0.93227092 0.93227092 ... 1.         1.         1.        ]
 [0.93227092 0.93227092 0.93227092 ... 1.         1.         1.        ]
 [0.93227092 0.93227092 0.93227092 ... 1.         1.         1.        ]]
```

Normalized Image