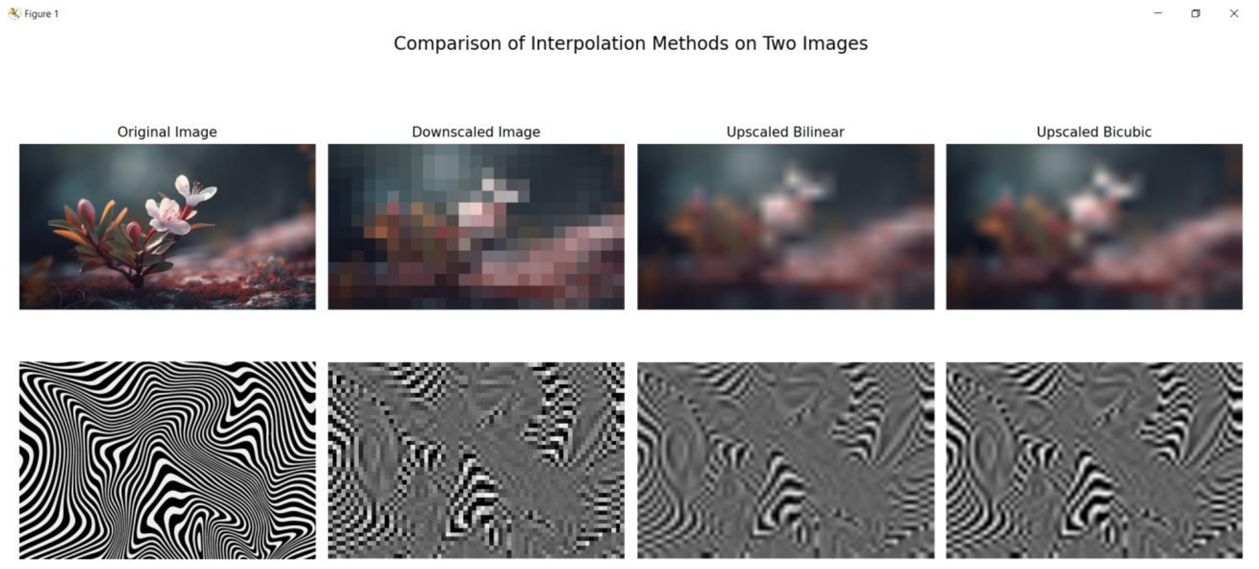
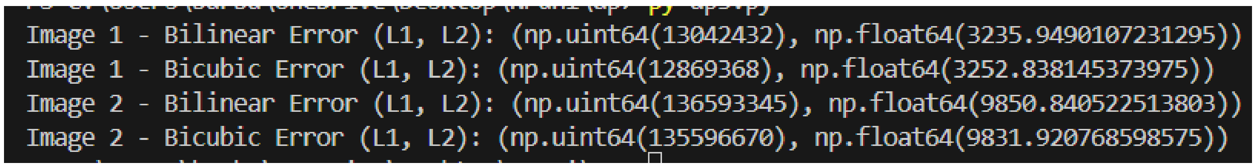
# AP 3

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

image1 = cv2.imread('./content/image1.jpg')
image2 = cv2.imread('./content/image2.png')

def downscale_image(image, scale):
    downscale_size = (int(image.shape[1] * scale), int(image.shape[0] * scale))
    downscaled_image = cv2.resize(image, downscale_size,
interpolation=cv2.INTER_AREA)
    return downscaled_image

def crop_to_match(original, resized):
    min_height = min(original.shape[0], resized.shape[0])
    min_width = min(original.shape[1], resized.shape[1])
    return original[:min_height, :min_width], resized[:min_height, :min_width]

def calculate_error(original, interpolated):
    original_cropped, interpolated_cropped = crop_to_match(original,
interpolated)
    difference = original_cropped - interpolated_cropped
    norm1 = np.sum(np.abs(difference))  # L1 norm
    norm2 = np.sqrt(np.sum(difference**2))  # L2 norm
    return norm1, norm2

# Bilinear interpolation function
def bilinear_interpolation(image, scale_factor):
    original_height, original_width, channels = image.shape
    new_width = int(original_width * scale_factor)
    new_height = int(original_height * scale_factor)
    resized_image = np.zeros((new_height, new_width, channels), dtype=np.uint8)

    for y in range(new_height):
        for x in range(new_width):
            original_x = x / scale_factor
            original_y = y / scale_factor

            x1 = int(original_x)
            x2 = min(x1 + 1, original_width - 1)
            y1 = int(original_y)
            y2 = min(y1 + 1, original_height - 1)

            dx = original_x - x1
            dy = original_y - y1

            for c in range(channels):
                top_left = image[y1, x1, c]
                top_right = image[y1, x2, c]
                bottom_left = image[y2, x1, c]
                bottom_right = image[y2, x2, c]

                top = (1 - dx) * top_left + dx * top_right
                bottom = (1 - dx) * bottom_left + dx * bottom_right
                resized_image[y, x, c] = (1 - dy) * top + dy * bottom

    return resized_image.astype(np.uint8)

# Cubic interpolation function
def cubic_interpolate(p, x):
    return (p[1] + 0.5 * x *
            (p[2] - p[0] + x *
            (2.0 * p[0] - 5.0 * p[1] + 4.0 * p[2] - p[3] +
            x * (3.0 * (p[1] - p[2]) + p[3] - p[0]))))

def bicubic_interpolation(image, scale_factor):
    original_height, original_width, channels = image.shape
    new_width = int(original_width * scale_factor)
    new_height = int(original_height * scale_factor)
    resized_image = np.zeros((new_height, new_width, channels), dtype=np.uint8)

    for y in range(new_height):
        for x in range(new_width):
            original_x = x / scale_factor
            original_y = y / scale_factor

            x1 = int(original_x)
            y1 = int(original_y)

            p = np.zeros((4, 4, channels), dtype=np.float32)

            for dy in range(-1, 3):
                for dx in range(-1, 3):
```

```python
                    px = min(max(x1 + dx, 0), original_width - 1)
                    py = min(max(y1 + dy, 0), original_height - 1)
                    p[dy + 1, dx + 1] = image[py, px]

            x_frac = original_x - x1
            y_frac = original_y - y1

            for c in range(channels):
                col = np.zeros(4)
                for i in range(4):
                    col[i] = cubic_interpolate(p[i, :, c], x_frac)

                value = int(cubic_interpolate(col, y_frac))
                value = max(0, min(255, value))
                resized_image[y, x, c] = value

    return resized_image.astype(np.uint8)

downscale_factor = 0.04
upscale_factor = 10

results = []
errors = []
for i, image in enumerate([image1, image2]):
    if image is None:
        print(f"Error loading image {i+1}")
        continue

    # Downscale the image
    low_res_image = downscale_image(image, downscale_factor)

    # Upscale using custom interpolation methods
    upscaled_nearest = bilinear_interpolation(low_res_image, upscale_factor)
    upscaled_bicubic = bicubic_interpolation(low_res_image, upscale_factor)

    # Calculate error norms
    error_nearest = calculate_error(image, upscaled_nearest)
    error_bicubic = calculate_error(image, upscaled_bicubic)

    # Collect the images and errors for display
    results.append([image, low_res_image, upscaled_nearest, upscaled_bicubic])
    errors.append([error_nearest, error_bicubic])
```

```
# Display all results side by side
titles = ['Original Image', 'Downscaled Image', 'Upscaled Bilinear', 'Upscaled
Bicubic']
plt.figure(figsize=(15, 10))

for row in range(len(results)):  # One row per image
    for col in range(4):  # Four columns for each type of image
        img = results[row][col]
        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        plt.subplot(2, 4, row * 4 + col + 1)
        plt.imshow(img_rgb)
        plt.title(titles[col] if row == 0 else "")
        plt.axis('off')

plt.suptitle('Comparison of Interpolation Methods on Two Images', fontsize=16)
plt.tight_layout()
plt.show()

# Print error norms
for i, (error_nearest, error_bicubic) in enumerate(errors):
    print(f"Image {i+1} - Bilinear Error (L1, L2): {error_nearest}")
    print(f"Image {i+1} - Bicubic Error (L1, L2): {error_bicubic}")
```





```
Image 1 - Bilinear Error (L1, L2): (np.uint64(13042432), np.float64(3235.9490107231295))
Image 1 - Bicubic Error (L1, L2): (np.uint64(12869368), np.float64(3252.838145373975))
Image 2 - Bilinear Error (L1, L2): (np.uint64(136593345), np.float64(9850.840522513803))
Image 2 - Bicubic Error (L1, L2): (np.uint64(135596670), np.float64(9831.920768598575))
```

I selected bilinear and bicubic interpolation methods, i chose these two because i liked the names 😊 , i downsized the image to see the results and after it, for the first image both methods work fine, while for the second image both methods work pretty badly, the reason is that the second image is a high frequency image as the intensity changes very rapidly from white to black. Also, I calculated errors using first and second norm.