image-processing

October 11, 2024

1 Advanced Image Manipulation and Enhancement Using Python.

1.1 1. Image Transformation: Rotation, Cropping, and Resizing.

[5]:
```
pip install opencv-python matplotlib
```

```
Requirement already satisfied: opencv-python in
c:\users\roari\anaconda3\lib\site-packages (4.10.0.84)
Requirement already satisfied: matplotlib in c:\users\roari\anaconda3\lib\site-
packages (3.8.4)
Requirement already satisfied: numpy>=1.21.2 in
c:\users\roari\anaconda3\lib\site-packages (from opencv-python) (1.26.4)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\roari\anaconda3\lib\site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in
c:\users\roari\anaconda3\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\roari\anaconda3\lib\site-packages (from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
c:\users\roari\anaconda3\lib\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: packaging>=20.0 in
c:\users\roari\anaconda3\lib\site-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=8 in c:\users\roari\anaconda3\lib\site-
packages (from matplotlib) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\roari\anaconda3\lib\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in
c:\users\roari\anaconda3\lib\site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\users\roari\anaconda3\lib\site-
packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

[1]:
```
pip install opencv-python
```

```
Requirement already satisfied: opencv-python in
c:\users\roari\anaconda3\lib\site-packages (4.10.0.84)
Requirement already satisfied: numpy>=1.21.2 in
c:\users\roari\anaconda3\lib\site-packages (from opencv-python) (1.26.4)
```

```
Note: you may need to restart the kernel to use updated packages.
```

1.2 1) Rotate an image by 45°, 90°, and 180° using OpenCV and Pillow.

1.2.1 OpenCV

[7]:
```python
import cv2
import matplotlib.pyplot as plt

# Load the original image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
image = cv2.imread(image_path)

# Function to rotate the image by a specified angle
def rotate_image(image, angle):
    (h, w) = image.shape[:2]
    center = (w // 2, h // 2)
    M = cv2.getRotationMatrix2D(center, angle, 1.0)
    rotated = cv2.warpAffine(image, M, (w, h))
    return rotated

# Rotate the image by 45°, 90°, and 180°
rotated_45 = rotate_image(image, 45)
rotated_90 = rotate_image(image, 90)
rotated_180 = rotate_image(image, 180)

# Convert images from BGR to RGB for displaying with matplotlib
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
rotated_45_rgb = cv2.cvtColor(rotated_45, cv2.COLOR_BGR2RGB)
rotated_90_rgb = cv2.cvtColor(rotated_90, cv2.COLOR_BGR2RGB)
rotated_180_rgb = cv2.cvtColor(rotated_180, cv2.COLOR_BGR2RGB)

# Plot the images with labels
fig, axes = plt.subplots(1, 4, figsize=(20, 5))
axes[0].imshow(image_rgb)
axes[0].set_title("Original Image")
axes[1].imshow(rotated_45_rgb)
axes[1].set_title("Rotated 45°")
axes[2].imshow(rotated_90_rgb)
axes[2].set_title("Rotated 90°")
axes[3].imshow(rotated_180_rgb)
axes[3].set_title("Rotated 180°")

# Hide axes
for ax in axes:
    ax.axis('off')
```

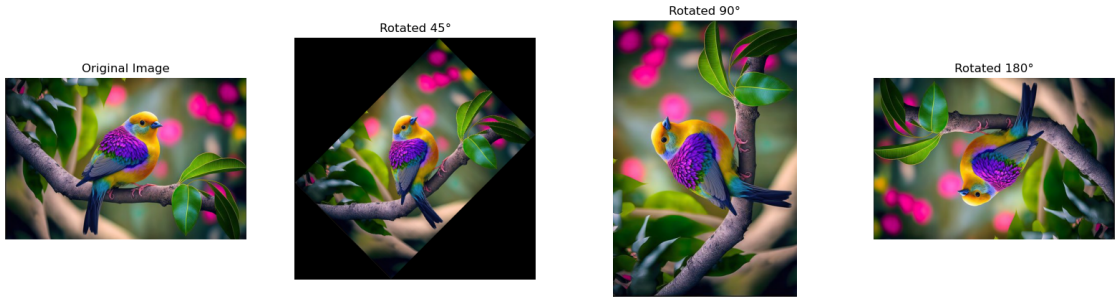
```
# Save the output as a single image file
output_path = r"C:\Users\roari\Downloads\Bird_rotated_output.jpg"
plt.savefig(output_path)
plt.show()

print(f"Rotated images with labels saved as: {output_path}")
```


```
Rotated images with labels saved as:
C:\Users\roari\Downloads\Bird_rotated_output.jpg
```

1.2.2 Pillow

[27]:
```
from PIL import Image
import matplotlib.pyplot as plt

# Load the original image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
image = Image.open(image_path)

# Function to rotate the image by a specified angle
def rotate_image(image, angle):
    return image.rotate(angle, expand=True)

# Rotate the image by 45°, 90°, and 180°
rotated_45 = rotate_image(image, 45)
rotated_90 = rotate_image(image, 90)
rotated_180 = rotate_image(image, 180)

# Convert images to RGB for displaying with matplotlib
image_rgb = image.convert("RGB")
rotated_45_rgb = rotated_45.convert("RGB")
rotated_90_rgb = rotated_90.convert("RGB")
rotated_180_rgb = rotated_180.convert("RGB")

# Plot the images with labels
fig, axes = plt.subplots(1, 4, figsize=(20, 5))
axes[0].imshow(image_rgb)
axes[0].set_title("Original Image")
```

```
axes[1].imshow(rotated_45_rgb)
axes[1].set_title("Rotated 45°")
axes[2].imshow(rotated_90_rgb)
axes[2].set_title("Rotated 90°")
axes[3].imshow(rotated_180_rgb)
axes[3].set_title("Rotated 180°")

# Hide axes
for ax in axes:
    ax.axis('off')

# Save the output as a single image file
output_path = r"C:\Users\roari\Downloads\Bird_rotated_output_pillow.jpg"
plt.savefig(output_path)
plt.show()

print(f"Rotated images with labels saved as: {output_path}")
```




```
Rotated images with labels saved as:
C:\Users\roari\Downloads\Bird_rotated_output_pillow.jpg
```

1.3 2) Crop a specific region from an image.

1.3.1 OpenCV

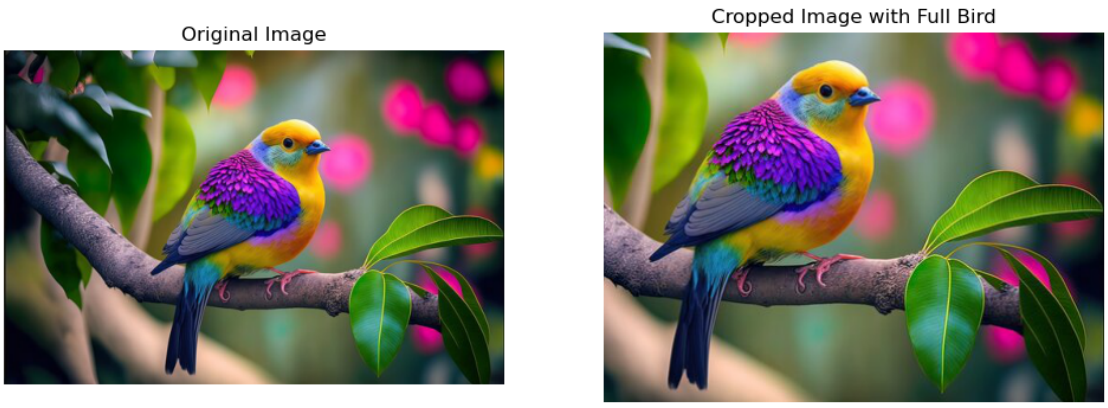
[21]:
```
import cv2
import matplotlib.pyplot as plt

# Load the image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
image = cv2.imread(image_path)

# Define the cropping coordinates and size
x, y, w, h = 120, 50, 450, 400  # Adjusted to include the full bird
```

```
# Crop the image to focus on the bird
cropped_image = image[y:y+h, x:x+w]

# Convert images from BGR to RGB for displaying with matplotlib
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
cropped_image_rgb = cv2.cvtColor(cropped_image, cv2.COLOR_BGR2RGB)

# Create a subplot with 1 row and 2 columns
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Display the original image
axes[0].imshow(image_rgb)
axes[0].set_title("Original Image")
axes[0].axis('off')  # Hide the axes

# Display the cropped image
axes[1].imshow(cropped_image_rgb)
axes[1].set_title("Cropped Image with Full Bird")
axes[1].axis('off')  # Hide the axes

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

1.3.2 Pillow

[31]:
```
from PIL import Image
import matplotlib.pyplot as plt

# Load the image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
image = Image.open(image_path)
```

```python
# Get image dimensions
width, height = image.size

# Define the cropping coordinates and size (left, upper, right, lower)
x, y, w, h = 120, 50, 450, 400
left = x
upper = y
right = min(x + w, width)  # Ensure the cropping box does not exceed the image dimensions
lower = min(y + h, height) # Ensure the cropping box does not exceed the image dimensions

# Crop the image to focus on the bird
cropped_image = image.crop((left, upper, right, lower))

# Convert images to RGB for displaying with matplotlib
image_rgb = image.convert("RGB")
cropped_image_rgb = cropped_image.convert("RGB")

# Create a subplot with 1 row and 2 columns
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Display the original image
axes[0].imshow(image_rgb)
axes[0].set_title("Original Image")
axes[0].axis('off')  # Hide the axes

# Display the cropped image
axes[1].imshow(cropped_image_rgb)
axes[1].set_title("Cropped Image with Full Bird")
axes[1].axis('off')  # Hide the axes

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



2 Advance analysis of resize image.

2.1 USING SSIM MAP

[5]:
```
import warnings
from cryptography.utils import CryptographyDeprecationWarning

# Suppress the deprecation warning
warnings.filterwarnings("ignore", category=CryptographyDeprecationWarning)
```

[7]:
```
from PIL import Image
import numpy as np
import cv2
import matplotlib.pyplot as plt
from skimage.metrics import structural_similarity as ssim

# Load the image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
image = Image.open(image_path)

# Get image dimensions
width, height = image.size

# Define the cropping coordinates and size (left, upper, right, lower)
x, y, w, h = 120, 50, 450, 400
left = x
upper = y
right = min(x + w, width)  # Ensure the cropping box does not exceed the image␣
 ↪dimensions
lower = min(y + h, height) # Ensure the cropping box does not exceed the image␣
 ↪dimensions

# Crop the image to focus on the bird
cropped_image = image.crop((left, upper, right, lower))

# Convert images to grayscale for SSIM calculation
image_gray = np.array(image.convert("L"))
cropped_image_gray = np.array(cropped_image.convert("L"))

# Resize the cropped image to match the original image's dimensions for␣
 ↪comparison
```

```
cropped_image_gray_resized = cv2.resize(cropped_image_gray, (image_gray.
 ↪shape[1], image_gray.shape[0]))

# Compute the SSIM map and the score
ssim_score, ssim_map = ssim(image_gray, cropped_image_gray_resized, full=True)

# Create a subplot with 1 row and 3 columns
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Display the original grayscale image
axes[0].imshow(image_gray, cmap='gray')
axes[0].set_title("Original Image (Grayscale)")
axes[0].axis('off')  # Hide the axes

# Display the resized cropped image (grayscale)
axes[1].imshow(cropped_image_gray_resized, cmap='gray')
axes[1].set_title("Resized Cropped Image (Grayscale)")
axes[1].axis('off')  # Hide the axes

# Display the SSIM map
axes[2].imshow(ssim_map, cmap='gray')
axes[2].set_title(f"SSIM Map (Score: {ssim_score:.2f})")
axes[2].axis('off')  # Hide the axes

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

SSIM Map (Score: 0.30): A visual representation of the Structural Similarity Index (SSIM) between the original and resized/cropped images. SSIM is a metric used to measure the perceptual similarity between two images, considering factors like luminance, contrast, and structure.

Insights from the SSIM Map:

The SSIM Map provides valuable information about the differences between the original and resized/cropped images:

Low SSIM Score (0.30): The relatively low SSIM score of 0.30 indicates that there are significant structural differences between the two images. This suggests that the resizing or cropping process has affected the image's overall appearance and quality. Dark Regions: The darker regions in the

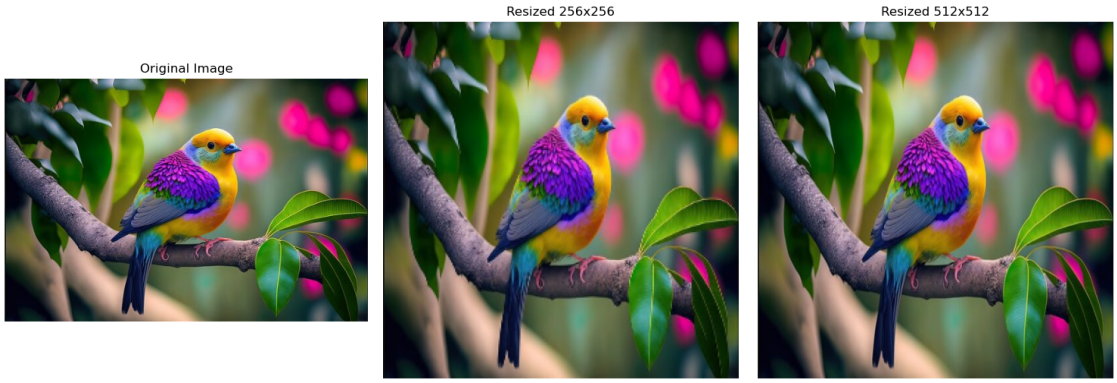
SSIM Map highlight areas where the differences between the images are more pronounced. These areas likely correspond to regions where the image has been distorted or lost information due to resizing or cropping. Structural Similarity: The SSIM metric considers structural similarity, which means it is sensitive to changes in the spatial arrangement of image features. Therefore, the dark regions in the SSIM Map likely indicate areas where the spatial relationships between elements have been altered.

2.2 3) Resizing to different image sizes.

2.2.1 OpenCV

[25]:
```
import cv2
import matplotlib.pyplot as plt

# Load the image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
image = cv2.imread(image_path)

# Define the resizing dimensions
sizes = [(256, 256), (512, 512)]

# Resize the image to the specified dimensions
resized_images = [cv2.resize(image, size) for size in sizes]

# Convert images from BGR to RGB for displaying with matplotlib
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
resized_images_rgb = [cv2.cvtColor(resized, cv2.COLOR_BGR2RGB) for resized in␣
 ↪resized_images]

# Create a subplot with 1 row and 3 columns
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# Display the original image
axes[0].imshow(image_rgb)
axes[0].set_title("Original Image")
axes[0].axis('off')

# Display the resized images
axes[1].imshow(resized_images_rgb[0])
axes[1].set_title("Resized 256x256")
axes[1].axis('off')

axes[2].imshow(resized_images_rgb[1])
axes[2].set_title("Resized 512x512")
axes[2].axis('off')
```

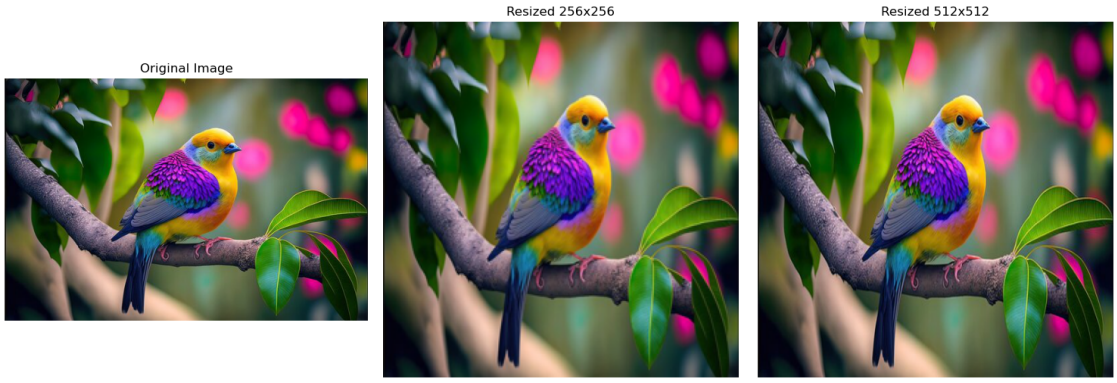
```
# Show the combined plot
plt.tight_layout()
plt.show()
```



2.2.2 Pillow

[33]:
```
from PIL import Image
import matplotlib.pyplot as plt

# Load the image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
image = Image.open(image_path)

# Define the resizing dimensions
sizes = [(256, 256), (512, 512)]

# Resize the image to the specified dimensions
resized_images = [image.resize(size) for size in sizes]

# Convert images to RGB for displaying with matplotlib
image_rgb = image.convert("RGB")
resized_images_rgb = [resized.convert("RGB") for resized in resized_images]

# Create a subplot with 1 row and 3 columns
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# Display the original image
axes[0].imshow(image_rgb)
axes[0].set_title("Original Image")
axes[0].axis('off')

# Display the resized images
axes[1].imshow(resized_images_rgb[0])
```

```
axes[1].set_title("Resized 256x256")
axes[1].axis('off')

axes[2].imshow(resized_images_rgb[1])
axes[2].set_title("Resized 512x512")
axes[2].axis('off')

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



---

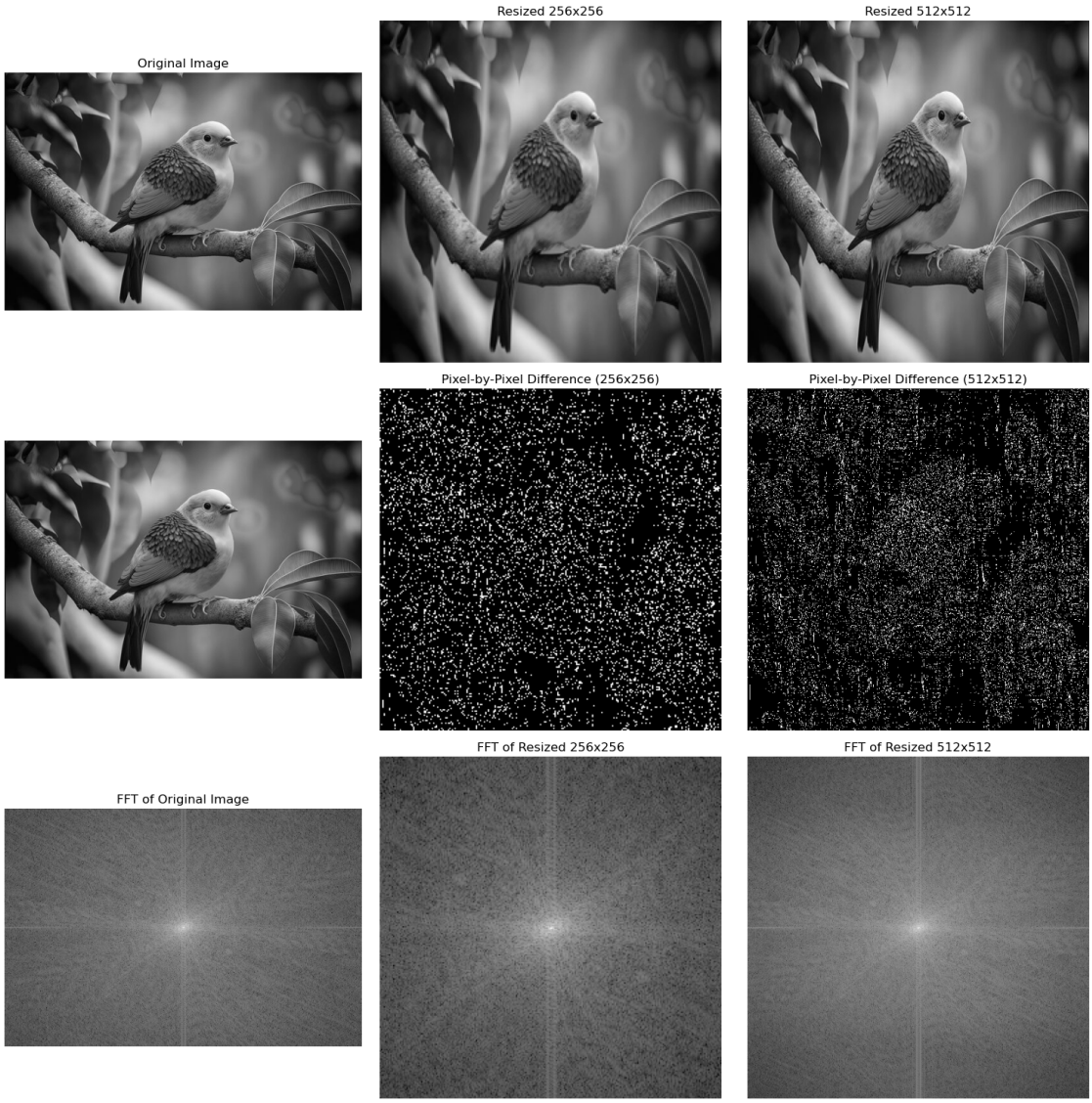
3 Advance analysis of effect of resizing on images of different resize images.

4 Pixel by pixel difference and Fast Fourier Transform (FFT)

[13]:
```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from numpy.fft import fft2, fftshift

# Load the image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
image = Image.open(image_path)

# Define the resizing dimensions
sizes = [(256, 256), (512, 512)]

# Resize the image to the specified dimensions
resized_images = [image.resize(size) for size in sizes]
```

```
# Convert images to grayscale for pixel-by-pixel difference and Fourier␣
 ↪transform
image_gray = image.convert("L")
resized_images_gray = [resized.convert("L") for resized in resized_images]

# Resize the original image to match resized images for pixel-by-pixel␣
 ↪comparison
image_resized_256 = image_gray.resize((256, 256))
image_resized_512 = image_gray.resize((512, 512))

# Convert images to numpy arrays
original_array_256 = np.array(image_resized_256)
original_array_512 = np.array(image_resized_512)
resized_array_256 = np.array(resized_images_gray[0])
resized_array_512 = np.array(resized_images_gray[1])

# Compute the pixel-by-pixel absolute difference
difference_256 = np.abs(original_array_256 - resized_array_256)
difference_512 = np.abs(original_array_512 - resized_array_512)

# Function to plot the 2D Fourier Transform of an image
def plot_fft(image_array, title):
    f_transform = fft2(image_array)
    f_shift = fftshift(f_transform)  # Shift the zero-frequency component to␣
 ↪the center
    magnitude_spectrum = np.log(np.abs(f_shift) + 1)  # Log scale for better␣
 ↪visualization
    plt.imshow(magnitude_spectrum, cmap='gray')
    plt.title(title)
    plt.axis('off')

# Plot results
fig, axes = plt.subplots(3, 3, figsize=(15, 15))

# Original images
axes[0, 0].imshow(image_gray, cmap='gray')
axes[0, 0].set_title("Original Image")
axes[0, 0].axis('off')

axes[0, 1].imshow(resized_images_gray[0], cmap='gray')
axes[0, 1].set_title("Resized 256x256")
axes[0, 1].axis('off')

axes[0, 2].imshow(resized_images_gray[1], cmap='gray')
axes[0, 2].set_title("Resized 512x512")
axes[0, 2].axis('off')
```

```
# Pixel-by-pixel difference
axes[1, 0].imshow(image_gray, cmap='gray')
axes[1, 0].axis('off')

axes[1, 1].imshow(difference_256, cmap='gray')
axes[1, 1].set_title("Pixel-by-Pixel Difference (256x256)")
axes[1, 1].axis('off')

axes[1, 2].imshow(difference_512, cmap='gray')
axes[1, 2].set_title("Pixel-by-Pixel Difference (512x512)")
axes[1, 2].axis('off')

# Fourier transforms
axes[2, 0].axis('off')
axes[2, 1].axis('off')
axes[2, 2].axis('off')

plt.subplot(3, 3, 7)
plot_fft(np.array(image_gray), "FFT of Original Image")

plt.subplot(3, 3, 8)
plot_fft(resized_array_256, "FFT of Resized 256x256")

plt.subplot(3, 3, 9)
plot_fft(resized_array_512, "FFT of Resized 512x512")

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



Analyzing the Image: Original and Pixel-by-Pixel Difference Maps Understanding the Components:

Original Image (Grayscale): The initial image in grayscale format, providing a baseline for comparison. Pixel-by-Pixel Difference (256x256): A difference map showing the pixel-wise differences between the original image and a resized (256x256) version of the image. Pixel-by-Pixel Difference (512x512): A difference map showing the pixel-wise differences between the original image and a resized (512x512) version of the image. Insights from the Difference Maps:

The difference maps provide valuable information about the impact of resizing on the image:

Noise and Artifacts: The difference maps are filled with noise and artifacts, indicating that the resizing process has introduced significant distortions into the image. This is especially evident in the 256x256 difference map, where the noise is more pronounced. Loss of Detail: The noise and artifacts in the difference maps suggest that the resizing process has led to a loss of detail in the image. This is because resizing often involves downsampling, which can reduce the number of pixels and blur fine features. Image Quality Degradation: The presence of noise and artifacts indicates a

significant degradation in image quality. This is particularly noticeable in the 256x256 difference map, where the noise is more severe. Conclusion:

Based on the analysis of the difference maps, it can be concluded that resizing the image has introduced significant noise and artifacts, leading to a degradation in image quality and a loss of detail. The impact of resizing is more pronounced in the 256x256 difference map, suggesting that smaller image sizes are more susceptible to distortion during resizing.Analyzing the FFTs of Resized Images Understanding the FFT:

The Fast Fourier Transform (FFT) is a mathematical technique used to decompose a signal into its frequency components. In the context of images, the FFT can be used to analyze the spatial frequency content of an image. High-frequency components represent fine details and edges, while low-frequency components represent overall brightness and smooth variations.

Insights from the FFTs:

Impact of Resizing on Frequency Content:

The FFTs of the resized images show a noticeable difference compared to the FFT of the original image. This indicates that resizing has altered the frequency content of the image. The FFT of the resized 256x256 image appears more spread out and contains more high-frequency components. This suggests that resizing to a smaller dimension has introduced additional high-frequency noise or artifacts. The FFT of the resized 512x512 image is closer to the original FFT but still shows some differences. This indicates that resizing to a larger dimension has less impact on the frequency content. Noise Introduction:

The increased high-frequency content in the FFTs of the resized images suggests that the resizing process has introduced noise. This noise can be attributed to interpolation techniques used during resizing, which can introduce artifacts. The amount of noise introduced may be more significant for smaller resized images, as the interpolation process has to fill in more pixels. Loss of Detail:

The changes in the frequency content due to resizing can lead to a loss of detail in the image. High-frequency components represent fine details, and if these components are affected by noise or artifacts, it can result in a loss of sharpness and clarity.

Conclusion: The FFT analysis reveals that resizing an image can significantly impact its frequency content. Smaller resized images are more likely to introduce noise and artifacts, leading to a loss of detail and a degradation in image quality. The choice of resizing algorithm and the extent of resizing can influence the severity of these effects.

---

5 2. Color Space Conversion.

[37]:
```
import cv2
import matplotlib.pyplot as plt

# Load the image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
image = cv2.imread(image_path)
```

```
# Convert the image from BGR to RGB for displaying with matplotlib
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Convert the image from RGB to HSV
image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

# Convert the image from RGB to Grayscale
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Convert the Grayscale image back to RGB
image_gray_rgb = cv2.cvtColor(image_gray, cv2.COLOR_GRAY2RGB)

# Convert the image back from HSV to RGB
image_hsv_rgb = cv2.cvtColor(image_hsv, cv2.COLOR_HSV2RGB)

# Create a subplot with 1 row and 4 columns
fig, axes = plt.subplots(1, 4, figsize=(20, 5))

# Display the original RGB image
axes[0].imshow(image_rgb)
axes[0].set_title("Original RGB")
axes[0].axis('off')

# Display the HSV image
axes[1].imshow(image_hsv)
axes[1].set_title("HSV Image")
axes[1].axis('off')

# Display the Grayscale image converted to RGB
axes[2].imshow(image_gray_rgb)
axes[2].set_title("Grayscale to RGB")
axes[2].axis('off')

# Display the HSV-to-RGB converted image
axes[3].imshow(image_hsv_rgb)
axes[3].set_title("HSV to RGB")
axes[3].axis('off')

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

1. RGB (Red, Green, Blue) Description: Color Model: RGB is an additive color model where colors are created by combining red, green, and blue light in various intensities. Representation: Each pixel is represented by three values corresponding to the intensities of red, green, and blue. Use: Colour Manipulation

2. HSV (Hue, Saturation, Value) Description: Color Model: HSV represents colors in terms of hue (color type), saturation (color intensity), and value (brightness). Representation: Hue describes the type of color, saturation represents the purity of the color, and value indicates the brightness. Use: Colour Adjustment

3. Grayscale Description: Color Model: Grayscale images represent intensity only, with no color information. Each pixel is a shade of gray from black to white. Representation: Each pixel is represented by a single value that denotes its intensity. Use: Edge detectionScenarios for Each Conversion: 1) RGB to HSV: Detect or track objects based on their color rather than their intensity 2) RGB to Grayscale: Edge detection 3) HSV to RGB: Colour-based processing

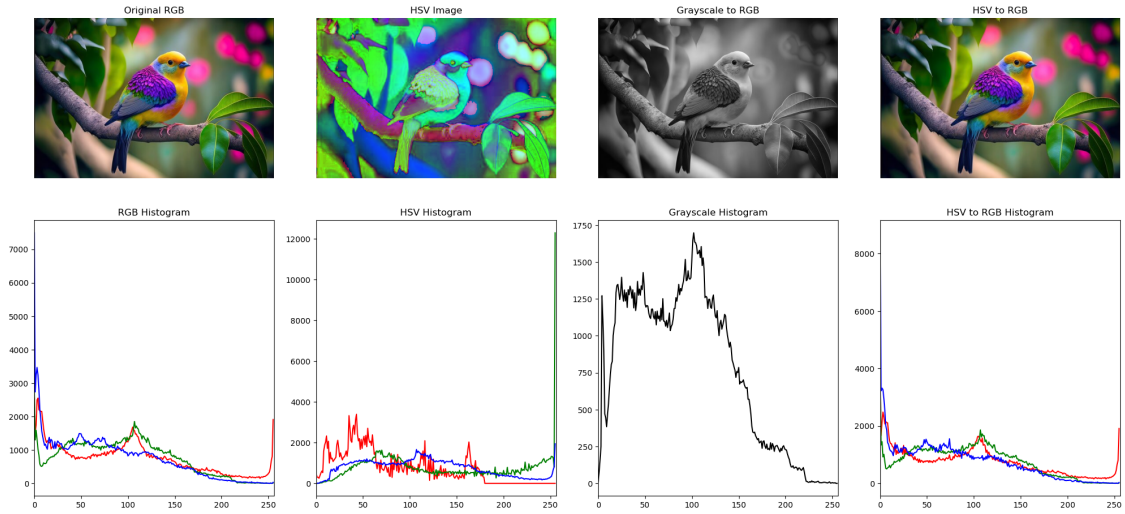
---

6 Advance colour space conversion analysis-1 (RGB histogram based analysis).

[1]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np

# Load the image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
image = cv2.imread(image_path)

# Convert the image from BGR to RGB for displaying with matplotlib
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Convert the image from RGB to HSV
image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

# Convert the image from RGB to Grayscale
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Convert the Grayscale image back to RGB
image_gray_rgb = cv2.cvtColor(image_gray, cv2.COLOR_GRAY2RGB)
```

```
# Convert the image back from HSV to RGB
image_hsv_rgb = cv2.cvtColor(image_hsv, cv2.COLOR_HSV2RGB)

# Function to compute the histogram
def compute_histogram(image, channels):
    if len(image.shape) == 2:  # Grayscale image
        hist = cv2.calcHist([image], [0], None, [256], [0, 256])
    else:  # Color image, compute histogram for each channel
        hist = []
        for ch in range(channels):
            hist_ch = cv2.calcHist([image], [ch], None, [256], [0, 256])
            hist.append(hist_ch)
    return hist

# Create a subplot with 2 rows and 4 columns (2 for images and 2 for histograms)
fig, axes = plt.subplots(2, 4, figsize=(20, 10))

# Image and histogram for Original RGB
axes[0, 0].imshow(image_rgb)
axes[0, 0].set_title("Original RGB")
axes[0, 0].axis('off')

hist_rgb = compute_histogram(image_rgb, 3)
for i, color in enumerate(('r', 'g', 'b')):
    axes[1, 0].plot(hist_rgb[i], color=color)
axes[1, 0].set_title("RGB Histogram")
axes[1, 0].set_xlim([0, 256])

# Image and histogram for HSV
axes[0, 1].imshow(image_hsv)
axes[0, 1].set_title("HSV Image")
axes[0, 1].axis('off')

hist_hsv = compute_histogram(image_hsv, 3)
for i, color in enumerate(('r', 'g', 'b')):
    axes[1, 1].plot(hist_hsv[i], color=color)
axes[1, 1].set_title("HSV Histogram")
axes[1, 1].set_xlim([0, 256])

# Image and histogram for Grayscale to RGB
axes[0, 2].imshow(image_gray_rgb, cmap='gray')
axes[0, 2].set_title("Grayscale to RGB")
axes[0, 2].axis('off')

hist_gray = compute_histogram(image_gray, 1)
axes[1, 2].plot(hist_gray, color='black')
```

```
axes[1, 2].set_title("Grayscale Histogram")
axes[1, 2].set_xlim([0, 256])

# Image and histogram for HSV to RGB
axes[0, 3].imshow(image_hsv_rgb)
axes[0, 3].set_title("HSV to RGB")
axes[0, 3].axis('off')

hist_hsv_rgb = compute_histogram(image_hsv_rgb, 3)
for i, color in enumerate(('r', 'g', 'b')):
    axes[1, 3].plot(hist_hsv_rgb[i], color=color)
axes[1, 3].set_title("HSV to RGB Histogram")
axes[1, 3].set_xlim([0, 256])

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```



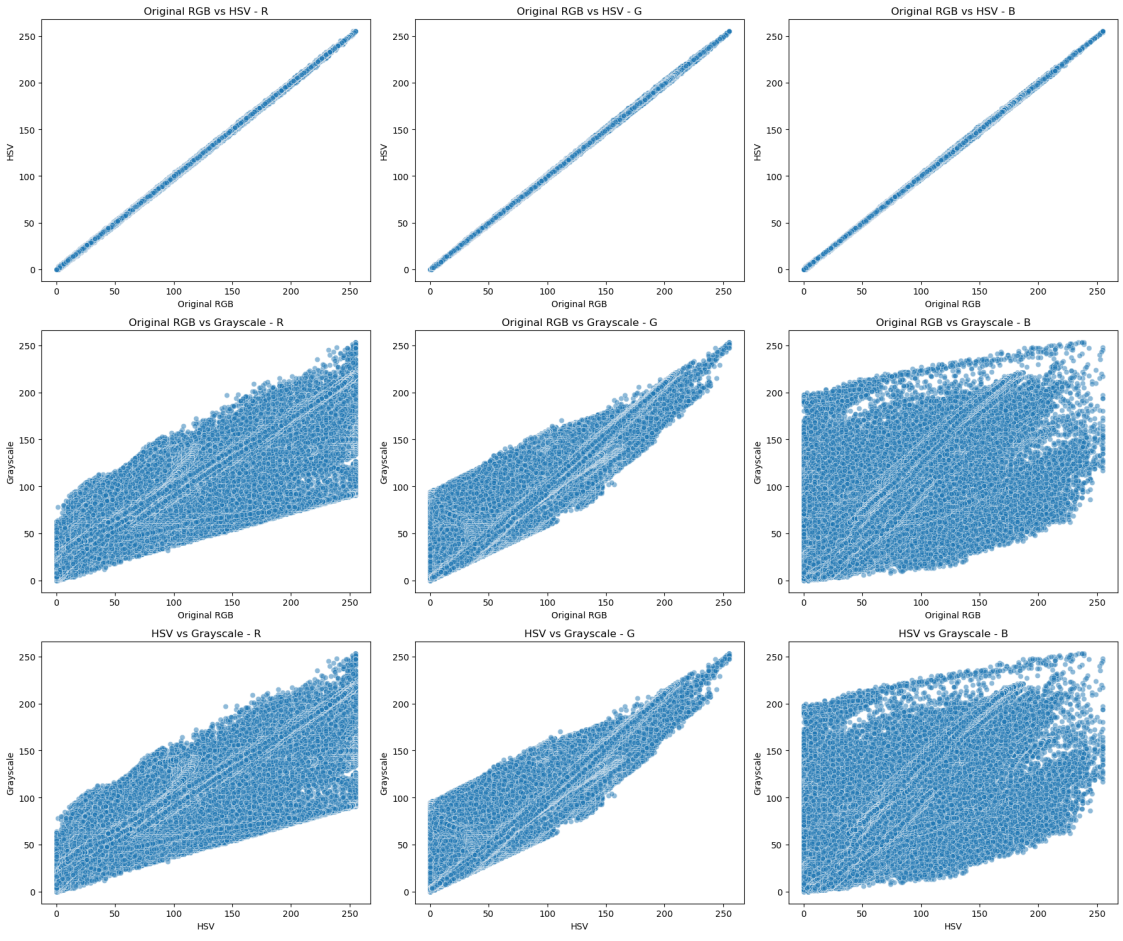
RGB TO HSV: The Red colour intensity increses and it is more concentrated in different peaks while there is sharp decrease in blue and green colour intensity in initial but at the end green intensity graph increases to peak. HSV TO RGB: If we observe closely there is some fluctuation in original vs this graph means some deteoration in image quality because original colour pixels are altered.

7 Advance colour space conversion Analysis-2 (For conservation of Colour information).

[94]:
```
import cv2
import numpy as np
import seaborn as sns
```

```
import matplotlib.pyplot as plt
import pandas as pd

# Load the image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
image = cv2.imread(image_path)

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

# Convert the image from RGB to HSV
image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

# Convert the image from RGB to Grayscale
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Convert the Grayscale image back to RGB
image_gray_rgb = cv2.cvtColor(image_gray, cv2.COLOR_GRAY2RGB)

# Convert the image back from HSV to RGB
image_hsv_rgb = cv2.cvtColor(image_hsv, cv2.COLOR_HSV2RGB)

# Flatten the images for scatter plot
def flatten_image(image):
    return image.reshape(-1, 3) if image.ndim == 3 else image.reshape(-1, 1)

# Flatten images
original_rgb_flat = flatten_image(image_rgb)
hsv_rgb_flat = flatten_image(image_hsv_rgb)
gray_rgb_flat = flatten_image(image_gray_rgb)

# Create DataFrames for plotting
df_original = pd.DataFrame(original_rgb_flat, columns=['R', 'G', 'B'])
df_hsv = pd.DataFrame(hsv_rgb_flat, columns=['R', 'G', 'B'])
df_gray = pd.DataFrame(gray_rgb_flat, columns=['R', 'G', 'B'])

# Create scatter plots using Seaborn
fig, axes = plt.subplots(3, 3, figsize=(18, 15))

# Scatter plots for RGB components
colors = ['R', 'G', 'B']

for i, color in enumerate(colors):
    sns.scatterplot(x=df_original[color], y=df_hsv[color], ax=axes[0, i],␣
 ↪alpha=0.5)
    axes[0, i].set_title(f'Original RGB vs HSV - {color}')
    axes[0, i].set_xlabel('Original RGB')
```

```
    axes[0, i].set_ylabel('HSV')

for i, color in enumerate(colors):
    sns.scatterplot(x=df_original[color], y=df_gray[color], ax=axes[1, i],␣
 ↪alpha=0.5)
    axes[1, i].set_title(f'Original RGB vs Grayscale - {color}')
    axes[1, i].set_xlabel('Original RGB')
    axes[1, i].set_ylabel('Grayscale')

for i, color in enumerate(colors):
    sns.scatterplot(x=df_hsv[color], y=df_gray[color], ax=axes[2, i], alpha=0.5)
    axes[2, i].set_title(f'HSV vs Grayscale - {color}')
    axes[2, i].set_xlabel('HSV')
    axes[2, i].set_ylabel('Grayscale')

# Adjust layout
plt.tight_layout()
plt.show()
```



Insights from the Scatter Plots:

RGB vs HSV:

The scatter plots show a strong positive correlation between the original RGB values and the corresponding HSV values. This indicates that the conversion from RGB to HSV preserves the overall color information and intensity. The diagonal lines in the scatter plots suggest a linear relationship between the pixel values. This is expected, as the HSV transformation is a mathematical conversion that preserves the underlying color information.

RGB vs Grayscale:

The scatter plots show a less clear relationship between the original RGB values and the grayscale values. This is because the grayscale conversion collapses the color information into a single intensity channel. The scatter plots are more scattered, indicating a loss of color information during the grayscale conversion.

Loss : Original/HSV to Greyscale :- Colour information loss more in Blue > than Red > than Green Insights from the Scatter Plots:

RGB vs Grayscale:

The scatter plots show a general linear relationship between the original RGB values and the grayscale values. This indicates that the grayscale conversion is primarily based on the overall intensity of the pixels. The scatter plots for the red and green channels show a slightly stronger correlation than the scatter plot for the blue channel, suggesting that the grayscale conversion may be more heavily influenced by the red and green components.

HSV vs Grayscale:

The scatter plots show a less clear relationship between the HSV values and the grayscale values. This is because the grayscale conversion collapses the color information into a single intensity channel, while the HSV representation includes information about hue and saturation. The scatter plots are more scattered, indicating a loss of color information during the grayscale conversion.

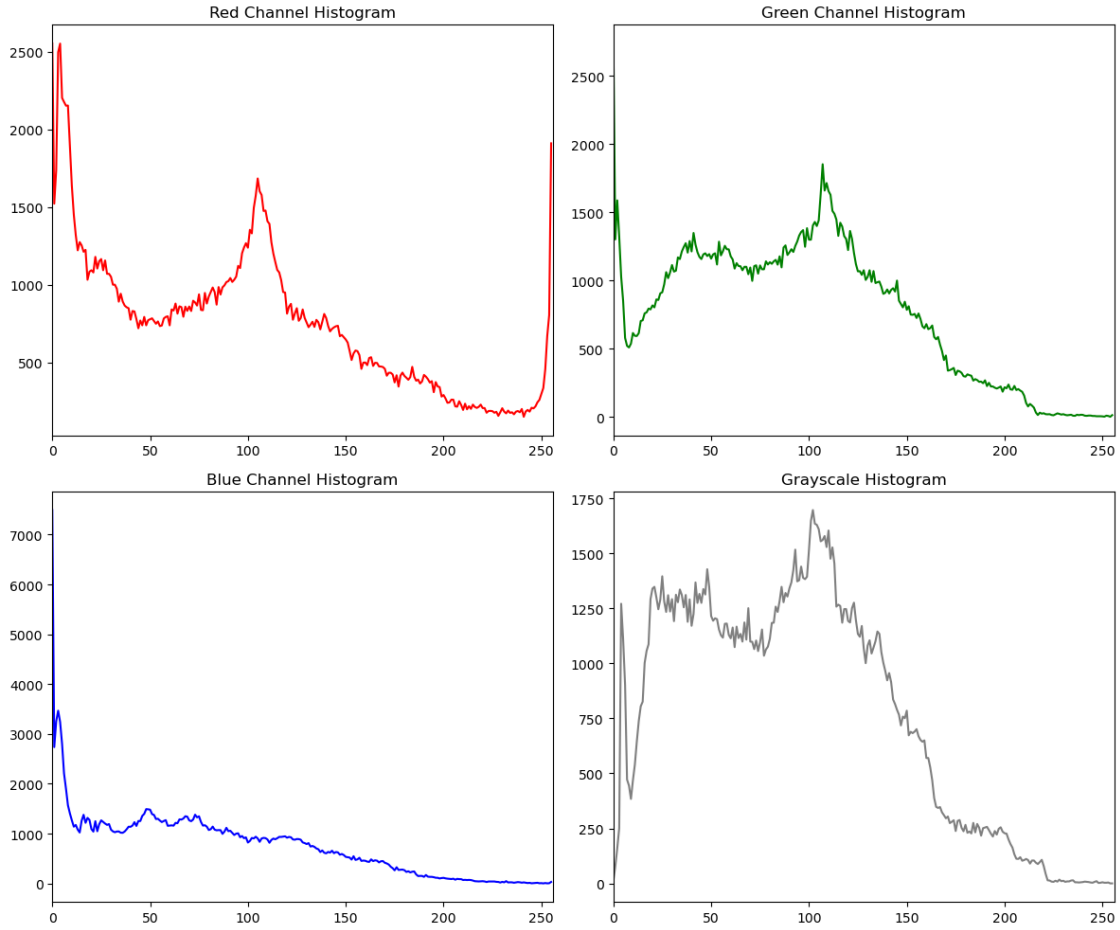
---

8 3. Histogram Analysis .

8.1 Generate and plot the histogram of the original image and the grayscale version.

[41]:
```
import cv2
import matplotlib.pyplot as plt

# Load the image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
image = cv2.imread(image_path)

# Convert the image from BGR to RGB for analysis
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

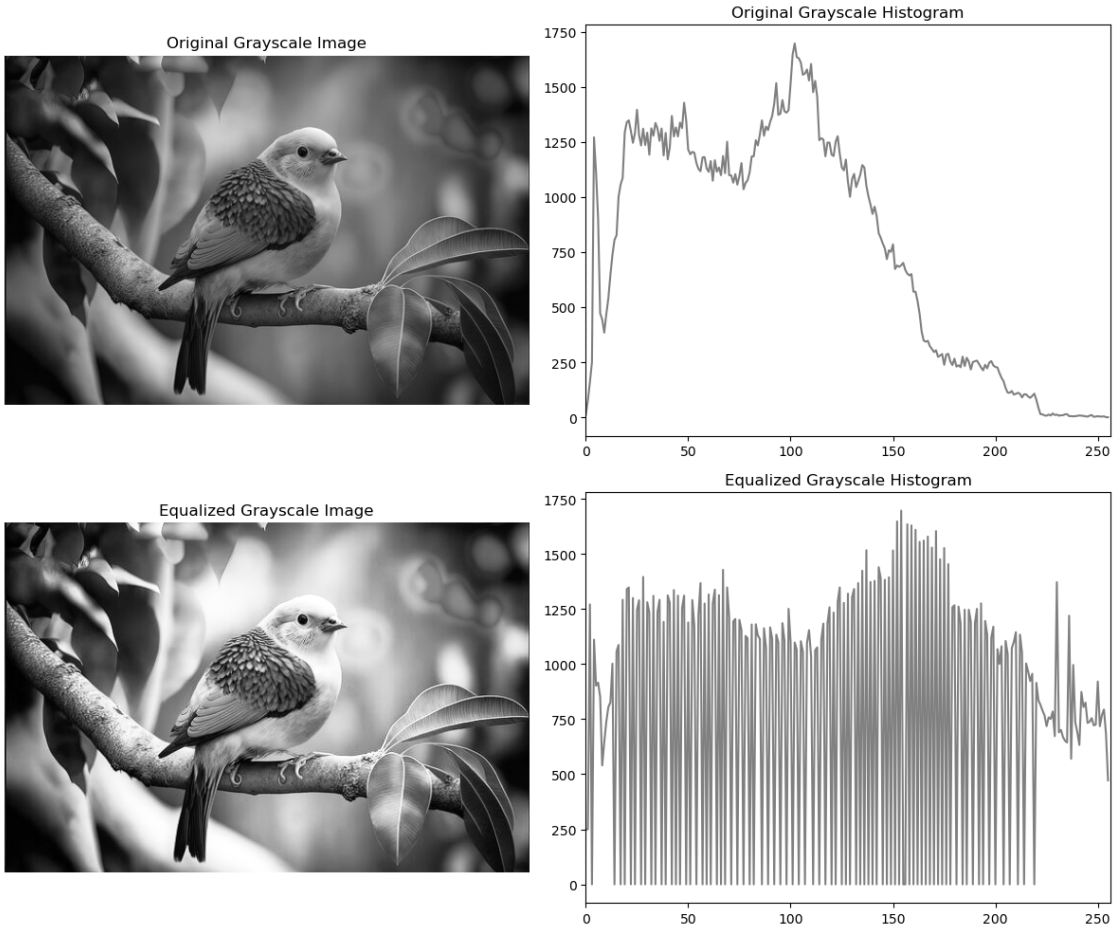
```
# Convert the image to grayscale
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Calculate histograms for the RGB channels
hist_r = cv2.calcHist([image_rgb[:, :, 0]], [0], None, [256], [0, 256])
hist_g = cv2.calcHist([image_rgb[:, :, 1]], [0], None, [256], [0, 256])
hist_b = cv2.calcHist([image_rgb[:, :, 2]], [0], None, [256], [0, 256])

# Calculate histogram for the grayscale image
hist_gray = cv2.calcHist([image_gray], [0], None, [256], [0, 256])

# Plot the histograms
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# RGB Histograms
axes[0, 0].plot(hist_r, color='red')
axes[0, 0].set_title("Red Channel Histogram")
axes[0, 0].set_xlim([0, 256])

axes[0, 1].plot(hist_g, color='green')
axes[0, 1].set_title("Green Channel Histogram")
axes[0, 1].set_xlim([0, 256])

axes[1, 0].plot(hist_b, color='blue')
axes[1, 0].set_title("Blue Channel Histogram")
axes[1, 0].set_xlim([0, 256])

# Grayscale Histogram
axes[1, 1].plot(hist_gray, color='gray')
axes[1, 1].set_title("Grayscale Histogram")
axes[1, 1].set_xlim([0, 256])

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



8.2 Performing histogram equalization on the grayscale image to enhance its contrast, and plot the histogram of the enhanced image.

[49]:
```
import cv2
import matplotlib.pyplot as plt

# Load the image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
image = cv2.imread(image_path)

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

# Perform histogram equalization
equalized_image = cv2.equalizeHist(image_gray)

# Calculate histograms for the original and equalized grayscale images
hist_gray = cv2.calcHist([image_gray], [0], None, [256], [0, 256])
```

```
hist_equalized = cv2.calcHist([equalized_image], [0], None, [256], [0, 256])

# Plot the histograms and images
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# Plot original grayscale image and its histogram
axes[0, 0].imshow(image_gray, cmap='gray')
axes[0, 0].set_title("Original Grayscale Image")
axes[0, 0].axis('off')

axes[0, 1].plot(hist_gray, color='gray')
axes[0, 1].set_title("Original Grayscale Histogram")
axes[0, 1].set_xlim([0, 256])

# Plot equalized grayscale image and its histogram
axes[1, 0].imshow(equalized_image, cmap='gray')
axes[1, 0].set_title("Equalized Grayscale Image")
axes[1, 0].axis('off')

axes[1, 1].plot(hist_equalized, color='gray')
axes[1, 1].set_title("Equalized Grayscale Histogram")
axes[1, 1].set_xlim([0, 256])

# Adjust layout
plt.tight_layout()
plt.show()

# Save the equalized image
equalized_image_path = r"C:\Users\roari\Downloads\Bird_Equalized.jpg"
cv2.imwrite(equalized_image_path, equalized_image)

print(f"Equalized image saved as: {equalized_image_path}")
```



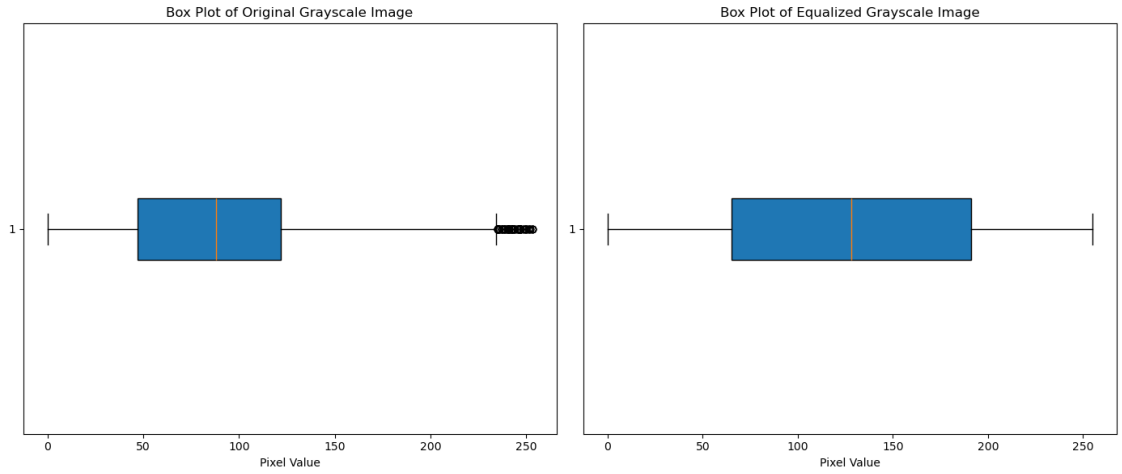

```
Equalized image saved as: C:\Users\roari\Downloads\Bird_Equalized.jpg
```

Insights from the Histograms:

Original Grayscale Histogram: The histogram shows a concentration of pixel values on the darker side, indicating a lack of contrast in the original image. Equalized Grayscale Histogram: The equalized grayscale histogram shows a more uniform distribution of pixel values, indicating that histogram equalization has successfully stretched the contrast.

[82]:
```
import cv2
import matplotlib.pyplot as plt

# Load the image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
image = cv2.imread(image_path)

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

```
# Perform histogram equalization
equalized_image = cv2.equalizeHist(image_gray)

# Create box plots for the pixel value distributions
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Plot box plot for the original grayscale image
axes[0].boxplot(image_gray.flatten(), vert=False, patch_artist=True)
axes[0].set_title("Box Plot of Original Grayscale Image")
axes[0].set_xlabel("Pixel Value")

# Plot box plot for the equalized grayscale image
axes[1].boxplot(equalized_image.flatten(), vert=False, patch_artist=True)
axes[1].set_title("Box Plot of Equalized Grayscale Image")
axes[1].set_xlabel("Pixel Value")

# Adjust layout
plt.tight_layout()
plt.show()
```



Insights from the Box Plots:

Median Pixel Value: Both box plots show a similar median pixel value, indicating that the overall brightness or intensity of the images is comparable. Interquartile Range (IQR): The IQR, represented by the box, is slightly larger in the box plot of the equalized grayscale image. This suggests that the equalized image has a wider range of pixel values, which is consistent with the contrast enhancement achieved through histogram equalization. Outliers: The box plots show a single outlier in the original grayscale image, indicating the presence of a pixel value that is significantly different from the rest of the data. Histogram equalization may have helped to reduce the impact of this outlier.

9 4. Image Filtering.

9.0.1 Apply the following filters to the grayscale version of an image and observe the changes:

Gaussian Blur Median Filter Sharpening Filter

[59]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
image = cv2.imread(image_path)

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

# Apply Gaussian Blur
gaussian_blur = cv2.GaussianBlur(image_gray, (5, 5), 0)

# Apply Median Filter
median_filter = cv2.medianBlur(image_gray, 5)

# Define a sharpening filter kernel
sharpening_kernel = np.array([[0, -1, 0],
                              [-1, 5, -1],
                              [0, -1, 0]])

# Apply Sharpening Filter
sharpened_image = cv2.filter2D(image_gray, -1, sharpening_kernel)

# Plot the images with filters applied
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# Display the original grayscale image
axes[0, 0].imshow(image_gray, cmap='gray')
axes[0, 0].set_title("Original Grayscale Image")
axes[0, 0].axis('off')

# Display the Gaussian Blur result
axes[0, 1].imshow(gaussian_blur, cmap='gray')
axes[0, 1].set_title("Gaussian Blur")
axes[0, 1].axis('off')

# Display the Median Filter result
axes[1, 0].imshow(median_filter, cmap='gray')
axes[1, 0].set_title("Median Filter")
```

```
axes[1, 0].axis('off')

# Display the Sharpening Filter result
axes[1, 1].imshow(sharpened_image, cmap='gray')
axes[1, 1].set_title("Sharpening Filter")
axes[1, 1].axis('off')

# Adjust layout
plt.tight_layout()
plt.show()
```

Why 5 taken at centre than 4 ?

Comparison with Different Kernels Kernel with Central Value of 5: This kernel is more aggressive in sharpening, emphasizing edges more strongly. Kernel with Central Value of 4: This kernel provides a subtler sharpening effect, which might be preferred in some cases to avoid excessive enhancement and artifacts.

Sum of the Kernel Values: The sum of all the kernel values equals 1 (i.e., 0 - 1 - 1 - 1 + 5 - 1 - 1 - 1 + 0), which ensures that the overall brightness of the image remains unchanged. This helps to avoid altering the brightness of the image while sharpening the details.

In this case, the central value (4) is still used to amplify the pixel values, but it might not be as strong as 5. The kernel [-1, 4, -1] is another commonly used sharpening filter known as the

Laplacian sharpening kernel. It is less aggressive compared to the kernel with a central value of 5, and might produce different results.1. Gaussian Blur Description:Gaussian Blur is a smoothing filter that reduces image noise and detail by averaging the pixel values within a kernel. It uses a Gaussian function to weight pixels around the center more heavily.

2. Median Filter Description:Median Filter is a non-linear filter that replaces each pixel's value with the median value of the pixels in its neighborhood. It is particularly effective at preserving edges while removing noise.

3. Sharpening Filter Description:Sharpening Filter increases the contrast between adjacent pixels to enhance the definition of edges and fine details in an image. It does so by amplifying the high-frequency components.Insights from the Images:

Gaussian Blur: The Gaussian blur image appears smoother and less detailed than the original image. This is because the Gaussian blur filter averages pixel values in a neighborhood, reducing noise and blurring edges. Median Filter: The median filter image also appears slightly smoother than the original image, but it preserves edges better than the Gaussian blur. This is because the median filter replaces each pixel with the median value of its neighbors, which can help to remove noise while preserving edges. Sharpening Filter: The sharpening filter image appears sharper and has more defined edges than the original image. This is because the sharpening filter enhances the contrast between edges, making them more prominent.

---

10 5. Combining Techniques.

10.1 Convert the image to Grayscale.

[76]:
```
import cv2
import matplotlib.pyplot as plt

# Load the image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
image = cv2.imread(image_path)

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

# Compute histogram of the grayscale image
hist_gray = cv2.calcHist([gray_image], [0], None, [256], [0, 256])

# Convert the grayscale image from BGR (OpenCV format) to RGB (matplotlib␣
 ↪format)
gray_image_rgb = cv2.cvtColor(gray_image, cv2.COLOR_GRAY2RGB)

# Create a subplot with 1 row and 2 columns
fig, axes = plt.subplots(1, 2, figsize=(12, 6))
```

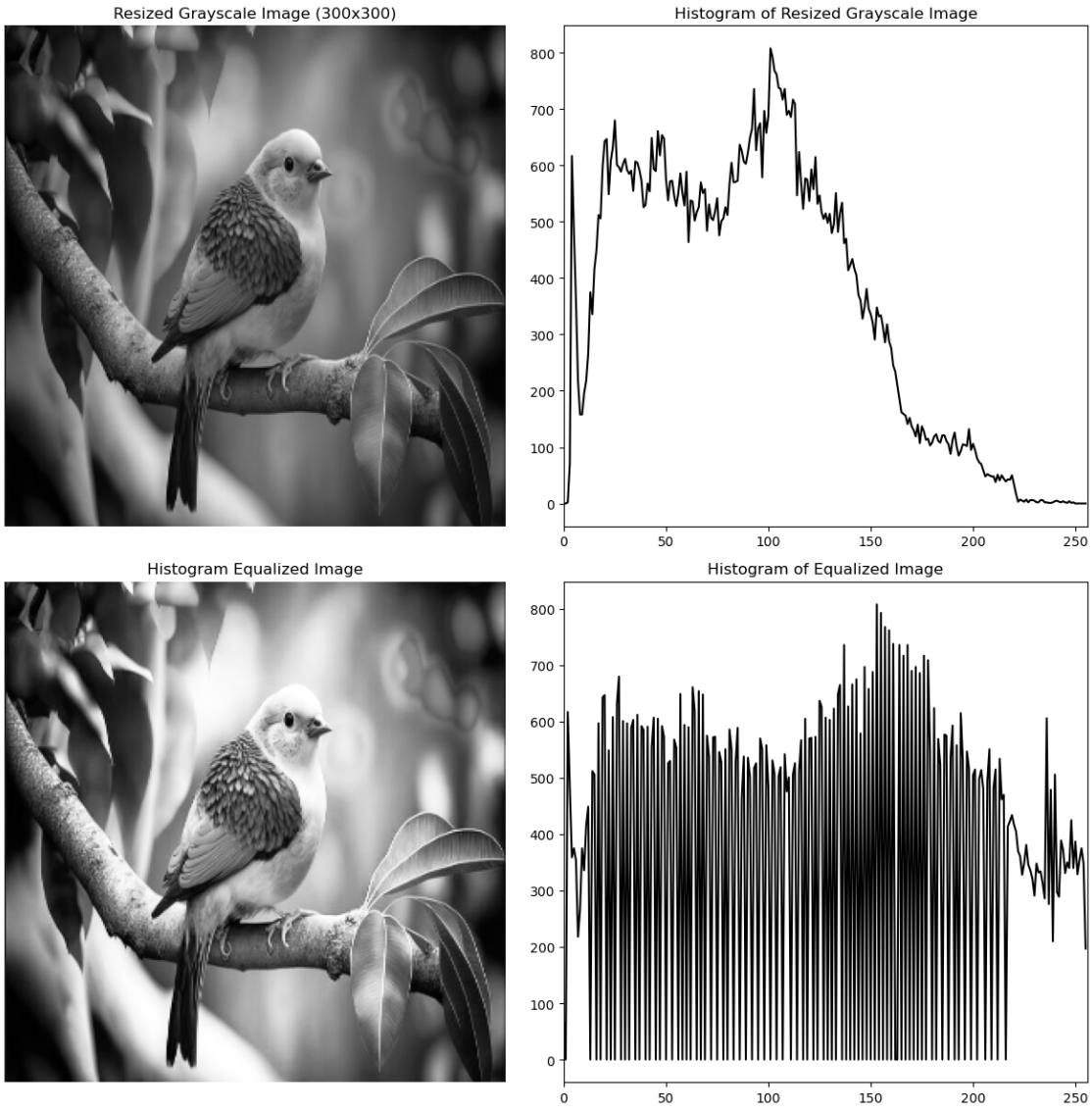
```
# Display the grayscale image
axes[0].imshow(gray_image_rgb, cmap='gray')
axes[0].set_title("Grayscale Image")
axes[0].axis('off')  # Hide the axes

# Display the histogram of the grayscale image
axes[1].plot(hist_gray, color='black')
axes[1].set_title("Histogram of Grayscale Image")
axes[1].set_xlim([0, 256])

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

10.2 Resize it to 300x300 pixels.

10.3 Apply histogram equalization.

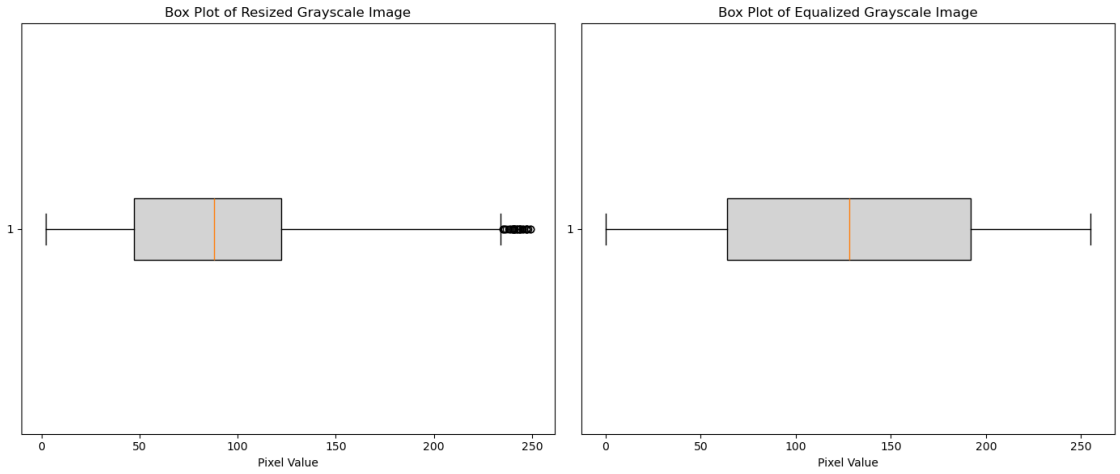
[72]:
```
import cv2
import matplotlib.pyplot as plt

# Load the image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
image = cv2.imread(image_path)

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

# Resize the grayscale image to 300x300 pixels
```

```
resized_gray_image = cv2.resize(gray_image, (300, 300))

# Apply histogram equalization to enhance the contrast of the grayscale image
equalized_gray_image = cv2.equalizeHist(resized_gray_image)

# Compute histograms
def compute_histogram(image):
    return cv2.calcHist([image], [0], None, [256], [0, 256])

hist_resized = compute_histogram(resized_gray_image)
hist_equalized = compute_histogram(equalized_gray_image)

# Plot the images and histograms
fig, axes = plt.subplots(2, 2, figsize=(12, 12))

# Display the resized grayscale image
axes[0, 0].imshow(resized_gray_image, cmap='gray')
axes[0, 0].set_title("Resized Grayscale Image (300x300)")
axes[0, 0].axis('off')  # Hide the axes

# Display the histogram of the resized grayscale image
axes[0, 1].plot(hist_resized, color='black')
axes[0, 1].set_title("Histogram of Resized Grayscale Image")
axes[0, 1].set_xlim([0, 256])

# Display the histogram equalized image
axes[1, 0].imshow(equalized_gray_image, cmap='gray')
axes[1, 0].set_title("Histogram Equalized Image")
axes[1, 0].axis('off')  # Hide the axes

# Display the histogram of the histogram equalized image
axes[1, 1].plot(hist_equalized, color='black')
axes[1, 1].set_title("Histogram of Equalized Image")
axes[1, 1].set_xlim([0, 256])

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



[84]:
```
import cv2
import matplotlib.pyplot as plt

# Load the image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
image = cv2.imread(image_path)

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

# Resize the grayscale image to 300x300 pixels
resized_gray_image = cv2.resize(gray_image, (300, 300))
```

```
# Apply histogram equalization to enhance the contrast of the grayscale image
equalized_gray_image = cv2.equalizeHist(resized_gray_image)

# Create box plots for the pixel value distributions
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Plot box plot for the resized grayscale image
axes[0].boxplot(resized_gray_image.flatten(), vert=False, patch_artist=True,
                boxprops=dict(facecolor='lightgray', color='black'))
axes[0].set_title("Box Plot of Resized Grayscale Image")
axes[0].set_xlabel("Pixel Value")

# Plot box plot for the equalized grayscale image
axes[1].boxplot(equalized_gray_image.flatten(), vert=False, patch_artist=True,
                boxprops=dict(facecolor='lightgray', color='black'))
axes[1].set_title("Box Plot of Equalized Grayscale Image")
axes[1].set_xlabel("Pixel Value")

# Adjust layout
plt.tight_layout()
plt.show()
```

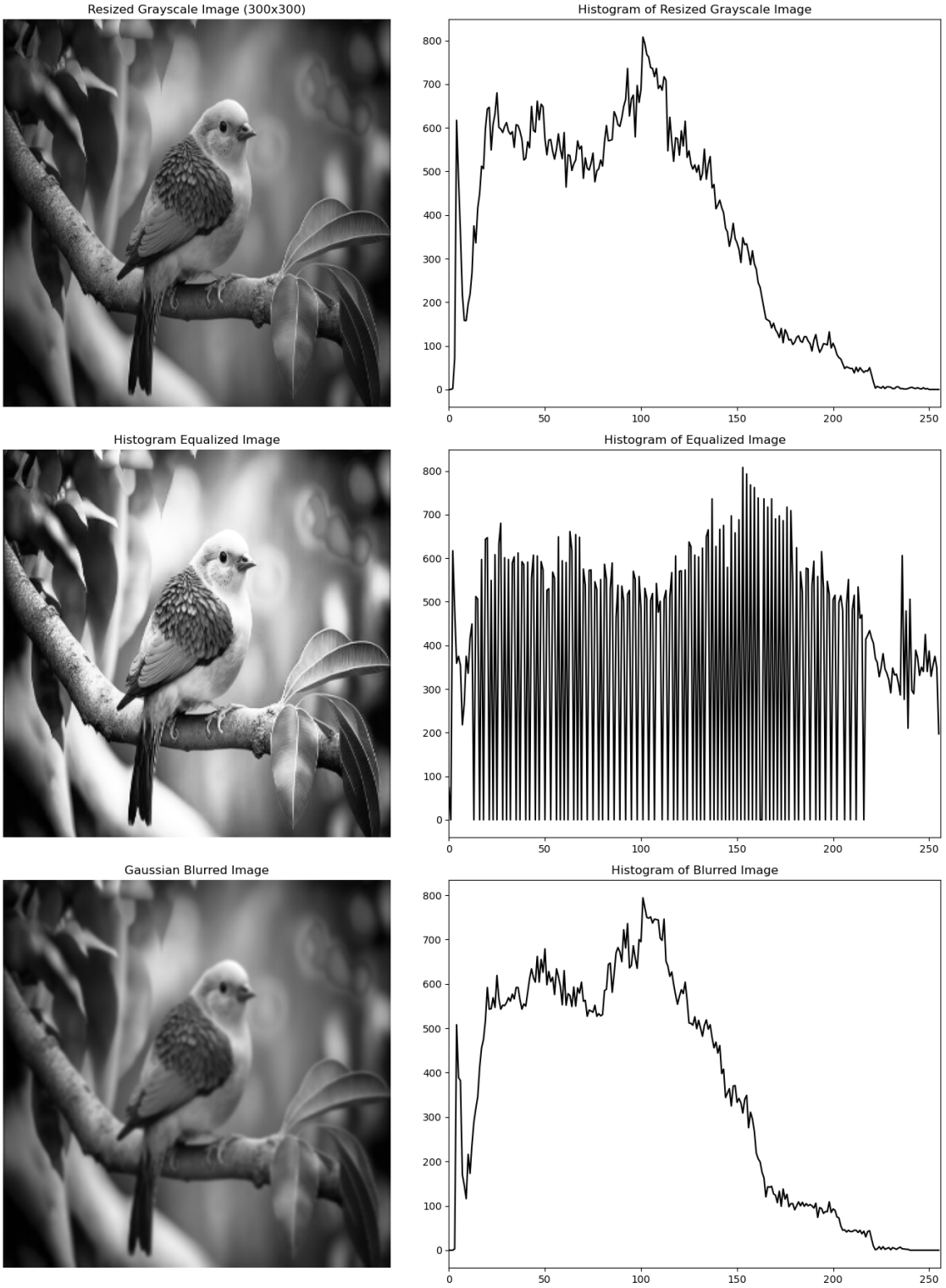


10.4 Comparing greyscale resize , equalized and gaussian blur

[80]:
```
import cv2
import matplotlib.pyplot as plt

# Load the image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
```

```
image = cv2.imread(image_path)

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

# Resize the grayscale image to 300x300 pixels
resized_gray_image = cv2.resize(gray_image, (300, 300))

# Apply histogram equalization to enhance the contrast of the grayscale image
equalized_gray_image = cv2.equalizeHist(resized_gray_image)

# Apply Gaussian blur to smoothen the image
blurred_image = cv2.GaussianBlur(resized_gray_image, (5, 5), 0)

# Compute histograms
def compute_histogram(image):
    return cv2.calcHist([image], [0], None, [256], [0, 256])

hist_resized = compute_histogram(resized_gray_image)
hist_equalized = compute_histogram(equalized_gray_image)
hist_blurred = compute_histogram(blurred_image)

# Plot the images and histograms
fig, axes = plt.subplots(3, 2, figsize=(14, 18))

# Display the resized grayscale image and its histogram
axes[0, 0].imshow(resized_gray_image, cmap='gray')
axes[0, 0].set_title("Resized Grayscale Image (300x300)")
axes[0, 0].axis('off')

axes[0, 1].plot(hist_resized, color='black')
axes[0, 1].set_title("Histogram of Resized Grayscale Image")
axes[0, 1].set_xlim([0, 256])

# Display the histogram equalized image and its histogram
axes[1, 0].imshow(equalized_gray_image, cmap='gray')
axes[1, 0].set_title("Histogram Equalized Image")
axes[1, 0].axis('off')

axes[1, 1].plot(hist_equalized, color='black')
axes[1, 1].set_title("Histogram of Equalized Image")
axes[1, 1].set_xlim([0, 256])

# Display the Gaussian blurred image and its histogram
axes[2, 0].imshow(blurred_image, cmap='gray')
axes[2, 0].set_title("Gaussian Blurred Image")
axes[2, 0].axis('off')
```

```
axes[2, 1].plot(hist_blurred, color='black')
axes[2, 1].set_title("Histogram of Blurred Image")
axes[2, 1].set_xlim([0, 256])

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```



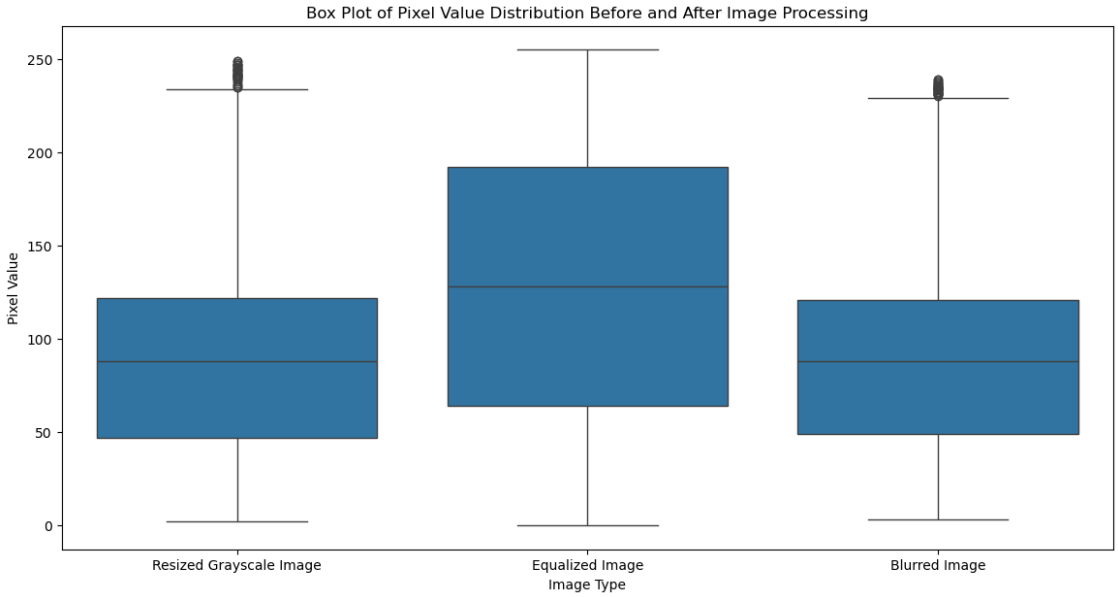
Insights from the Histograms:

Original Grayscale Histogram: The histogram shows a concentration of pixel values on the darker side, indicating a lack of contrast in the resized grayscale image. Histogram of Equalized Image: The

histogram shows a more uniform distribution of pixel values, indicating that histogram equalization has effectively stretched the contrast. Histogram of Blurred Image: The histogram shows a wider peak and a narrower range of pixel values, indicating that the Gaussian blur has reduced the contrast and smoothed the image.

[98]:
```
import cv2
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# Load the image
image_path = r"C:\Users\roari\Downloads\Bird 5.jpg"
image = cv2.imread(image_path)

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

# Resize the grayscale image to 300x300 pixels
resized_gray_image = cv2.resize(gray_image, (300, 300))

# Apply histogram equalization to enhance the contrast of the grayscale image
equalized_gray_image = cv2.equalizeHist(resized_gray_image)

# Apply Gaussian blur to the resized grayscale image
blurred_image = cv2.GaussianBlur(resized_gray_image, (5, 5), 0)

# Flatten the images for box plot
def flatten_image(image):
    return image.flatten()

# Flatten images
resized_flat = flatten_image(resized_gray_image)
equalized_flat = flatten_image(equalized_gray_image)
blurred_flat = flatten_image(blurred_image)

# Create a DataFrame for box plot
data = pd.DataFrame({
    'Pixel Value': pd.concat([pd.Series(resized_flat), pd.
 ↪Series(equalized_flat), pd.Series(blurred_flat)], axis=0),
    'Image Type': ['Resized Grayscale Image'] * len(resized_flat) +
                  ['Equalized Image'] * len(equalized_flat) +
                  ['Blurred Image'] * len(blurred_flat)
})

# Create box plot using seaborn
plt.figure(figsize=(14, 7))
sns.boxplot(x='Image Type', y='Pixel Value', data=data)
```

```
plt.title('Box Plot of Pixel Value Distribution Before and After Image␣
  ↪Processing')
plt.show()
```



Insights from the Box Plot:

Median Pixel Value: The median pixel values for the three image types are relatively similar, indicating that the overall brightness or intensity is comparable. Interquartile Range (IQR): The IQR, represented by the box, is slightly larger for the equalized image compared to the resized grayscale and blurred images. This suggests that the equalized image has a wider range of pixel values, which is consistent with the contrast enhancement achieved through histogram equalization. Outliers: The box plots show outliers in the resized grayscale and blurred images, but no outliers in the equalized image.Reasons for Outliers:

Resizing: The resizing process can introduce artifacts or distortions that may lead to outliers. These artifacts can appear as unusual pixel values that deviate significantly from the rest of the data. Blurring: While blurring can reduce noise and smooth the image, it may also introduce artifacts or distortions, especially if the blurring parameters are not chosen appropriately. Histogram Equalization: Histogram equalization can help to reduce the impact of outliers by stretching the histogram and distributing pixel values more evenly. This can make it less likely for individual pixel values to stand out as outliers.

11 END

[ ]: