B_A3_Feature_Extraction

In [ ]:
```
pip install matplotlib
```

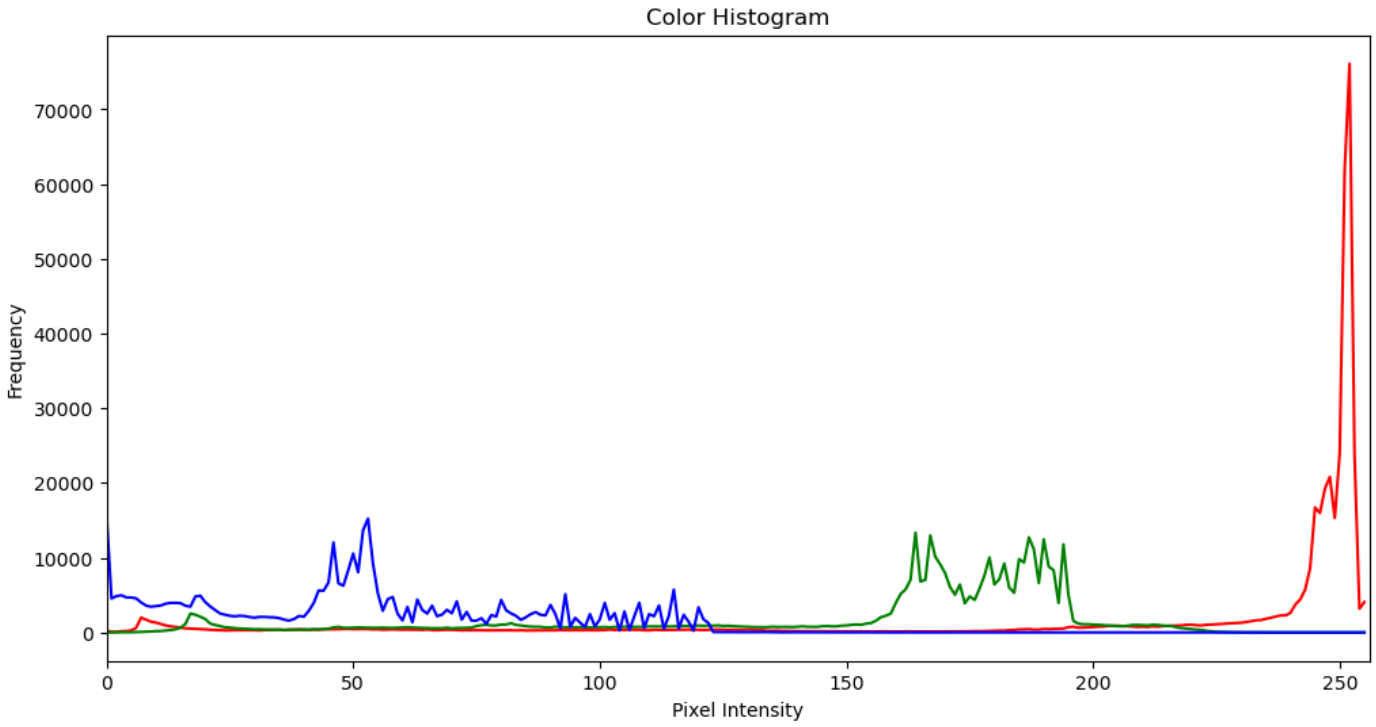
In [ ]:
```
pip install opencv-python
```

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import io
```

In [2]:
```
image_path = "123.jpg" # Replace with your image path
image = cv2.imread(image_path)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

In [3]:
```
def plot_color_histogram(image):
    color = ('r', 'g', 'b')
    plt.figure(figsize=(12, 6))
    for i, col in enumerate(color):
        hist = cv2.calcHist([image], [i], None, [256], [0, 256])
        plt.plot(hist, color=col)
        plt.xlim([0, 256])
    plt.title('Color Histogram')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')
    plt.show()
```

In [4]:
```
# Plot color histogram
plot_color_histogram(image_rgb)
```



In [5]:
```
def plot_glcm_texture_features(image):
    # Convert to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Calculate GLCM
```

```
    glcm = greycomatrix(gray, distances=[1], angles=[0], levels=256, symmetric=True, nor

    # Extract texture features
    contrast = greycoprops(glcm, 'contrast')[0, 0]
    dissimilarity = greycoprops(glcm, 'dissimilarity')[0, 0]
    homogeneity = greycoprops(glcm, 'homogeneity')[0, 0]
    energy = greycoprops(glcm, 'energy')[0, 0]
    correlation = greycoprops(glcm, 'correlation')[0, 0]

    # Plot Texture Feature Values
    texture_features = [contrast, dissimilarity, homogeneity, energy, correlation]
    feature_names = ['Contrast', 'Dissimilarity', 'Homogeneity', 'Energy', 'Correlation'

    plt.figure(figsize=(12, 6))
    plt.bar(feature_names, texture_features, color='skyblue')
    plt.title('Texture Features using GLCM')
    plt.xlabel('Texture Feature')
    plt.ylabel('Value')
    plt.show()
```

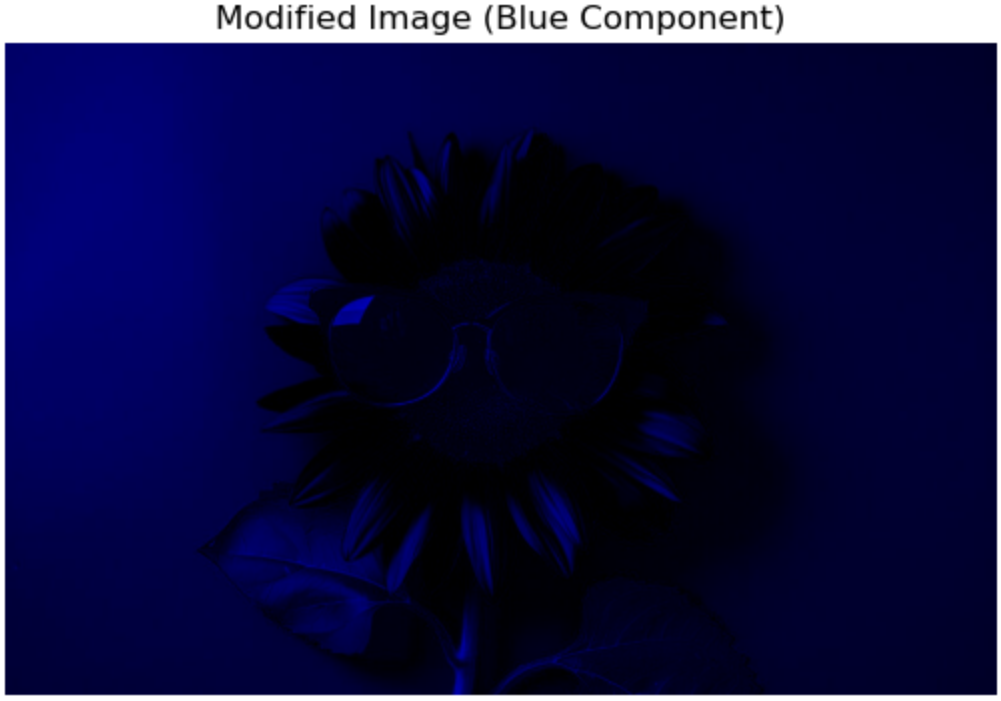
In [ ]:
```
# Plot texture features
plot_glcm_texture_features(image)
```

In [ ]:
```

```

In [ ]:
```
pip install Pillow
```

In [1]:
```
from PIL import Image
import os
import matplotlib.pyplot as plt

def color_to_rgb(alpha, red, green, blue):
    # Convert individual alpha, red, green, and blue components to a single integer repr
    new_pixel = 0
    new_pixel += alpha
    new_pixel = (new_pixel << 8) + red
    new_pixel = (new_pixel << 8) + green
    new_pixel = (new_pixel << 8) + blue
    return new_pixel

def image_histogram(input_image):
    # Create a new image to store the modified version
    red_graph = Image.new('RGB', input_image.size)
    pixels = input_image.load()

    for i in range(input_image.width):
        for j in range(input_image.height):
            # Get the pixel's red, green, and blue components
            r, g, b = pixels[i, j]

            # Set the pixel in the new image, modifying it to keep only the blue compone
            red_graph.putpixel((i, j), (0, 0, b))

    return red_graph

def write_image(output, image):
    # Save the modified image
    image.save(f"{output}.jpg")

def display_image(image, title):
    # Convert image to RGB for display and show using matplotlib
    plt.imshow(image)
    plt.title(title)
```

```
    plt.axis('off')  # Hide the axis
    plt.show()

if __name__ == "__main__":
    # Load the original image
    image_path = "123.jpg"

    if not os.path.exists(image_path):
        raise FileNotFoundError(f"Image file not found at {image_path}")

    original_image = Image.open(image_path)

    # Apply histogram extraction (modifying the blue component)
    answer_image = image_histogram(original_image)

    # Display the original and modified images
    display_image(original_image, "Original Image")
    display_image(answer_image, "Modified Image (Blue Component)")

    # Write the output image
    write_image("featureExtraction", answer_image)
```

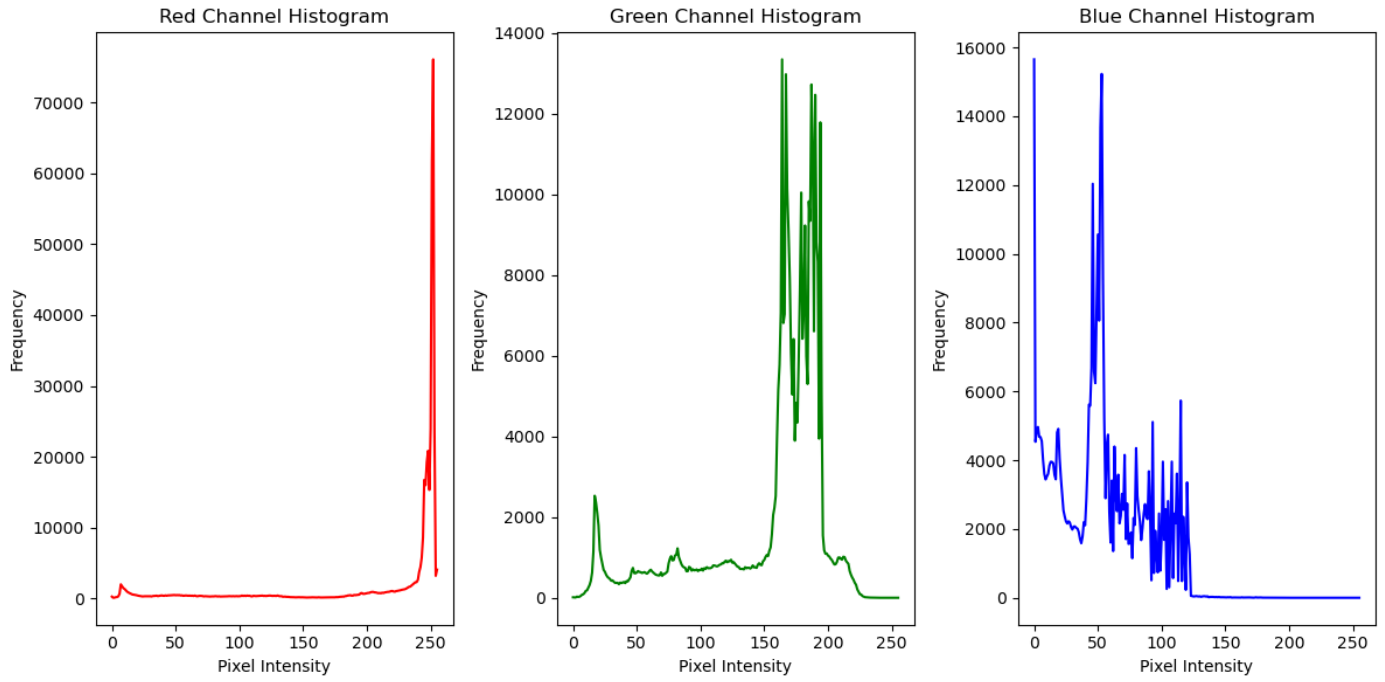


In [ ]:

In [ ]:
```
pip install pillow matplotlib numpy
```

In [2]:
```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import os

def extract_color_histogram(image):
    """
    Extracts and returns color histograms for R, G, B channels of the image.
    """
    # Convert image to numpy array for easier processing
    img_array = np.array(image)

    # Split the image into R, G, B channels
    red_channel = img_array[:, :, 0]
    green_channel = img_array[:, :, 1]
    blue_channel = img_array[:, :, 2]

    # Calculate histograms for each color channel
    red_hist = np.histogram(red_channel, bins=256, range=(0, 256))[0]
    green_hist = np.histogram(green_channel, bins=256, range=(0, 256))[0]
    blue_hist = np.histogram(blue_channel, bins=256, range=(0, 256))[0]

    return red_hist, green_hist, blue_hist

def plot_histogram(red_hist, green_hist, blue_hist):
    """
    Plots the histograms for R, G, B channels.
    """
    # Create a figure with three subplots, one for each channel
    plt.figure(figsize=(12, 6))

    # Plot Red Histogram
    plt.subplot(1, 3, 1)
    plt.plot(red_hist, color='red')
    plt.title('Red Channel Histogram')
    plt.xlabel('Pixel Intensity')
```

```
    plt.ylabel('Frequency')

    # Plot Green Histogram
    plt.subplot(1, 3, 2)
    plt.plot(green_hist, color='green')
    plt.title('Green Channel Histogram')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')

    # Plot Blue Histogram
    plt.subplot(1, 3, 3)
    plt.plot(blue_hist, color='blue')
    plt.title('Blue Channel Histogram')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')

    # Display the histograms
    plt.tight_layout()
    plt.show()

def main():
    # Load the original image
    image_path = "123.jpg"

    if not os.path.exists(image_path):
        raise FileNotFoundError(f"Image file not found at {image_path}")

    original_image = Image.open(image_path)

    # Extract color histograms from the image
    red_hist, green_hist, blue_hist = extract_color_histogram(original_image)

    # Plot histograms for each color channel
    plot_histogram(red_hist, green_hist, blue_hist)

if __name__ == "__main__":
    main()
```



In [ ]: