2 - 3.4. Color Models

October 9, 2024

Jarrian Vince G. Gojar
Instructor I
College of Information and Communications Technology, Sorsogon State University, Philippines

1 Introduction

In the field of image processing, color models play a crucial role in the representation and manipulation of colors. A color model is a mathematical framework that allows us to describe and encode colors as tuples of numbers, providing a standardized method for color representation across various applications.

There are several widely recognized color models, each serving different purposes and contexts within image processing. Some of the most commonly used models include `RGB`, `HSV`, `YUV`, and `CIE L*a*b`.

- **RGB (Red, Green, Blue):** The `RGB` color model represents colors by combining different intensities of red, green, and blue light. It is the most prevalent model used in digital displays, including computer screens, where colors are created through the additive mixing of these three primary colors.
- **HSV (Hue, Saturation, Value):** The `HSV` model offers an alternative approach by describing colors in terms of hue (the type of color), saturation (the intensity or purity of the color), and value (the brightness). This model is particularly useful in scenarios where color segmentation is important, as it aligns more closely with human perception of color.
- **YUV:** The `YUV` model separates a color into its luminance (Y) component and two chrominance (U and V) components. This separation allows for more efficient video compression, making `YUV` a preferred choice in video processing and broadcasting.
- **CIE L*a*b:** The `CIE L*a*b` model describes colors in terms of lightness (L*), and two color-opponent dimensions: a* (green to red) and b* (blue to yellow). This model is designed to be perceptually uniform, meaning that changes in the color model's values correspond to uniform changes in perceived color, making it ideal for tasks like color correction.

Each color model is selected based on its suitability for specific tasks within image processing. For instance, while `RGB` is the standard for digital displays, `YUV` is preferred for video compression, `HSV` is useful in color-based segmentation, and `CIE L*a*b` is valuable in achieving accurate color correction.

Read More:

- Color Model

2 Setup

[ ]:
```
%pip install opencv-python opencv-contrib-python numpy matplotlib
```

3 Initial Setup

[13]:
```
# Importing the OpenCV and Matplotlib libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt

image_path = '../../assets/images/parrot.jpg'

# Reading an image from the file using OpenCV
original_image = cv2.imread(image_path)

plt.figure("Original Image (Not Converted)")

# Displaying the original image
plt.imshow(original_image)
plt.axis('off')
plt.show()
```

The following code snippet demonstrates how to load an image using OpenCV:

```
original_image = cv2.imread(image_path)
```

In this example, the `cv2.imread()` function is used to read the image file specified by the variable `image_path`. The image is then stored in the variable `original_image`. It's important to note that OpenCV loads images in the BGR (Blue, Green, Red) format by default, which is different from the more common RGB (Red, Green, Blue) format. This distinction is crucial when performing color-sensitive operations later in your code.

4 Converting an Image to Grayscale

We convert an image to grayscale by taking the average of the three color channels. The average of the three color channels is calculated using the formula:

```
Gray = (R + G + B) / 3
```

where R, G, and B are the red, green, and blue color channels of the image, respectively.

Read More:

- Grayscale

[14]:
```
# Read Image in Grayscale Mode
grayscale_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

plt.figure("Grayscale Image")

# Displaying the grayscale image
plt.imshow(grayscale_image, cmap='gray')
plt.axis('off')
plt.show()
```

The following code snippet illustrates how to load an image in grayscale mode using OpenCV:

```
gray_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
```

By default, the `cv2.imread()` function loads images in the `BGR` (Blue, Green, Red) format. However, by specifying the `cv2.IMREAD_GRAYSCALE` flag, we can instruct OpenCV to read the image in grayscale mode instead. This flag converts the image into a single channel, where pixel values represent varying intensities of gray, rather than color information. Grayscale images are often used in scenarios where color is not critical, such as in edge detection or other image processing tasks that focus on intensity rather than hue.

5 Convert from BGR to RGB

When working with images in Python, it's important to be aware of how different libraries handle color spaces. For instance, `OpenCV` reads images in the `BGR` (Blue-Green-Red) color space by default, while `Matplotlib`, a popular plotting library, expects images in the `RGB` (Red-Green-Blue) color space. To ensure that the image is displayed correctly using `Matplotlib`, we need to convert it from the `BGR` color space to the `RGB` color space. This conversion is essential because the order of color channels directly affects how the image is rendered on screen.

Here's an example of how to perform this conversion:

```
rgb_image = cv2.cvtColor(bgr_image, cv2.COLOR_BGR2RGB)
```

In this code, `cv2.cvtColor()` is used to change the color space of the image from `BGR` to `RGB`. This step ensures that `OpenCV` and `Matplotlib` are compatible with each other, allowing for accurate

image display.

[15]:
```
# Convert color from BGR to RGB
original_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB)
grayscale_image = cv2.cvtColor(grayscale_image, cv2.COLOR_BGR2RGB)

# Displaying the image using Matplotlib
print("Original and Grayscale Image")
plt.figure('Original and Grayscale Image')

plt.subplot(1, 2, 1)
plt.imshow(original_image)
plt.axis('off')
plt.title('(a) Original Image', y=-0.15)

plt.subplot(1, 2, 2)
plt.imshow(grayscale_image, cmap='gray')
plt.axis('off')
plt.title('(b) Grayscale Image', y=-0.15)

plt.show()
```

```
Original and Grayscale Image
```

The code above shows the conversion of an image from `BGR` to `RGB` using OpenCV and displaying the image using Matplotlib. It converts the image from `BGR` to `RGB` using the `cv2.cvtColor()` function and displays the image using the `plt.imshow()` function.

6 RGB Color Model

The RGB color model is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue.

In the RGB color model, colors are represented as tuples of three numbers, where each number represents the intensity of one of the three primary colors. The intensity of each color is defined by the amount of red, green, and blue light present in the color.

The RGB color model is used to represent colors in digital images, computer graphics, and television screens. It is the most commonly used color model in image processing and computer vision.

Example of RGB Color Model:

- Red: (255, 0, 0)
- Green: (0, 255, 0)
- Blue: (0, 0, 255)
- White: (255, 255, 255)
- Black: (0, 0, 0)
- Yellow: (255, 255, 0)
- Cyan: (0, 255, 255)
- Magenta: (255, 0, 255)
- Gray: (128, 128, 128)

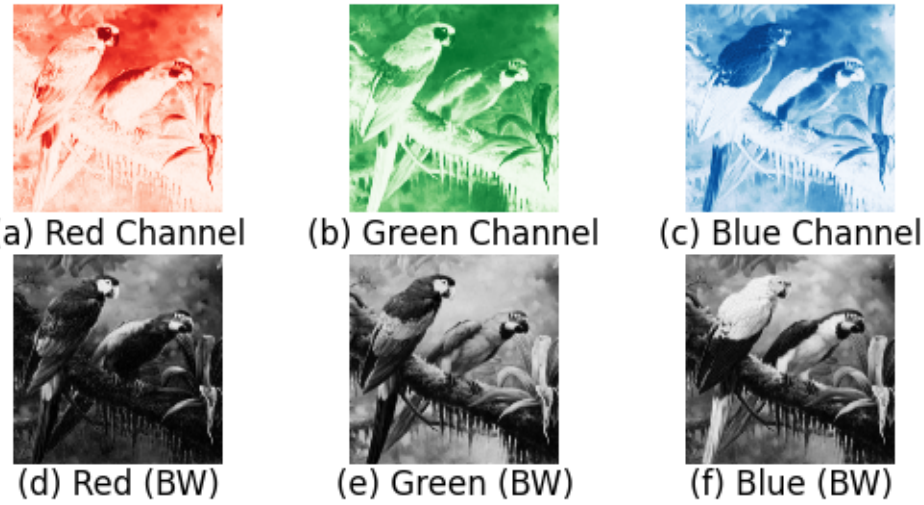
Read More:

- RGB Color Model

[16]:
```
# Splitting the RGB image into its three color channels
channel_b, channel_g, channel_r = cv2.split(original_image)

# Displaying the three color channels
print("Dividing the RGB Image into its Three Color Channels")
plt.figure('Dividing the RGB Image into its Three Color Channels')

# Display the Red Channel
plt.subplot(3, 3, 1)
plt.imshow(channel_r, cmap='Reds')
plt.title('(a) Red Channel', y=-0.225)
plt.axis('off')

# Display the Green Channel
plt.subplot(3, 3, 2)
plt.imshow(channel_g, cmap='Greens')
plt.title('(b) Green Channel', y=-0.225)
plt.axis('off')

# Display the Blue Channel
plt.subplot(3, 3, 3)
plt.imshow(channel_b, cmap='Blues')
```

```
plt.title('(c) Blue Channel', y=-0.225)
plt.axis('off')

# Display the Red Channel in Grayscale
plt.subplot(3, 3, 4)
plt.imshow(channel_r, cmap='gray')
plt.title('(d) Red (BW)', y=-0.225)
plt.axis('off')

# Display the Green Channel in Grayscale
plt.subplot(3, 3, 5)
plt.imshow(channel_g, cmap='gray')
plt.title('(e) Green (BW)', y=-0.225)
plt.axis('off')

# Display the Blue Channel in Grayscale
plt.subplot(3, 3, 6)
plt.imshow(channel_b, cmap='gray')
plt.title('(f) Blue (BW)', y=-0.225)
plt.axis('off')

plt.show()
```

Dividing the RGB Image into its Three Color Channels



The above code displays the three color channels of the RGB image using Matplotlib. The original image is split into three color channels using the `cv2.split()` function. The three color channels are displayed using Matplotlib. The first row displays the three color channels of the `RGB` image. The second row displays the three color channels in grayscale.

Important codes to remember:

- `cv2.split()`
  - Splits the image into its three color channels.
  - Syntax: `cv2.split(image)`

7 HSV Color Model

The `HSV` color model is a cylindrical color model that describes colors in terms of hue, saturation, and value. The hue component represents the color type, the saturation component represents the color intensity, and the value component represents the brightness of the color.

The `HSV` color model is used to represent colors in a way that is more intuitive to humans than the `RGB` color model. It is often used in image processing algorithms that require color segmentation, color correction, and color enhancement.

In the `HSV` color model, colors are represented as tuples of three numbers, where each number represents the hue, saturation, and value of the color. The hue component is represented as an angle between 0 and 360 degrees, the saturation component is represented as a percentage between 0 and 100%, and the value component is represented as a percentage between 0 and 100%.

Example of HSV Color Model:

- Red: (0, 100%, 100%)
- Green: (120, 100%, 100%)
- Blue: (240, 100%, 100%)
- White: (0, 0%, 100%)
- Black: (0, 0%, 0%)

In OpenCV, the maximum value for the hue component is 180, the maximum value for the saturation and value components is 255. This is because OpenCV uses the 8-bit unsigned integer data type to represent the color components which have a range of 0 to 255. So the hue component is scaled down to the range of 0 to 180 degrees in OpenCV or by dividing the original 0-360 degrees hue value by 2.

Read More:

- HSV

8 Converting an RGB Image to HSV Image

We convert an `RGB` image to an `HSV` image using the `cv2.cvtColor()` function in OpenCV. The `cv2.cvtColor()` function takes two arguments: the input image and the color space conversion code. The color space conversion code for converting an `RGB` image to an `HSV` image is `cv2.COLOR_RGB2HSV`.

The `cv2.cvtColor()` function returns the `HSV` image as a NumPy array. We can display the `HSV` image using Matplotlib by converting it to the `RGB` color space using the `cv2.cvtColor()` function with the `cv2.COLOR_HSV2RGB` color space conversion code.

Read More:

- Color Conversions in OpenCV

[17]:
```
# Converting the RGB image to the HSV image
hsv_image = cv2.cvtColor(original_image, cv2.COLOR_RGB2HSV)

# Splitting the HSV image into its three color channels
channel_h, channel_s, channel_v = cv2.split(hsv_image)

# Displaying the three color channels
print("Dividing the HSV Image into its Three Color Channels")
plt.figure('Dividing the HSV Image into its Three Color Channels')

plt.subplot(1, 3, 1)
plt.imshow(channel_h, cmap='gray')
plt.title('(a) Hue Channel', y=-0.2)
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(channel_s, cmap='gray')
plt.title('(b) Saturation Channel', y=-0.2)
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(channel_v, cmap='gray')
plt.title('(c) Value Channel', y=-0.2)
plt.axis('off')

plt.show()
```

```
Dividing the HSV Image into its Three Color Channels
```

The code above converts an image from the `RGB` color model to the `HSV` color model using OpenCV. The image is read using the `cv2.imread()` function, and the color model is converted using the `cv2.cvtColor()` function. The image is then split into its three color channels using the `cv2.split()` function. These channels are hue (H), saturation (S), and value (V). The channels

are then displayed using Matplotlib.

9 Masking Red Pixels in an Image

We can mask red pixels in an image by converting the image to the HSV color space and applying a mask to the hue channel. The mask is created by defining the lower and upper bounds of the red color range in the hue channel. The mask is then applied to the hue and saturation channels to remove the red pixels from the image.

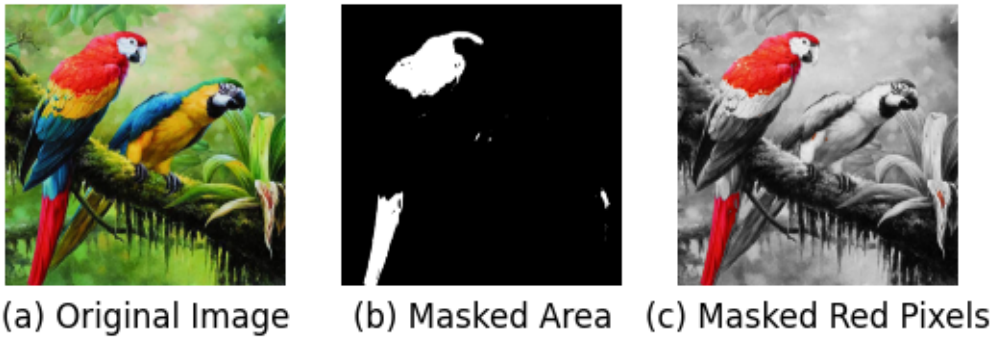
Read More:

- Color Mask

[18]:
```
# Convert image to HSV
hsv_image = cv2.cvtColor(original_image, cv2.COLOR_RGB2HSV)

# Divide the HSV image into its three color channels
channel_h, channel_s, channel_v = cv2.split(hsv_image)

# Define the lower and upper bounds of the red color range

# Mask 1 are the red pixels with hue values between 0 and 10
lower_red = np.array([0, 100, 100])
upper_red = np.array([10, 255, 255])
mask_red_1 = cv2.inRange(hsv_image, lower_red, upper_red)

# Mask 2 are the red pixels with hue values between 160 and 180
lower_red = np.array([160, 100, 100])
upper_red = np.array([180, 255, 255])
mask_red_2 = cv2.inRange(hsv_image, lower_red, upper_red)

# Combine the two masks using the cv2.bitwise_or() function
mask_red = cv2.bitwise_or(mask_red_1, mask_red_2)

# Apply the mask to the hue and saturation channels
channel_h[mask_red == 0] = 0
channel_s[mask_red == 0] = 0

# Combine the modified hue and saturation channels with the original value␣
 ↪channel
masked_HSV_red_image = cv2.merge([channel_h, channel_s, channel_v])

# Convert the modified HSV image to the RGB color space
masked_RGB_red_image = cv2.cvtColor(masked_HSV_red_image, cv2.COLOR_HSV2RGB)

# Display the modified image
print("Masking Red Pixels in the Image")
plt.figure('Masking Red Pixels in the Image')
```

```
plt.subplot(1, 3, 1)
plt.imshow(original_image)
plt.title('(a) Original Image', y=-0.2)
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(mask_red, cmap='gray')
plt.title('(b) Masked Area', y=-0.2)
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(masked_RGB_red_image)
plt.title('(c) Masked Red Pixels', y=-0.2)
plt.axis('off')

plt.show()
```

Masking Red Pixels in the Image



The above code is used to mask the red pixels in an image using the `HSV` color model. It reads an image from the file using OpenCV and converts it to the `HSV` color space. The `HSV` image is then divided into its three color channels: hue, saturation, and value. The lower and upper bounds of the red color range are defined using the `np.array()` function. Two masks are created to identify the red pixels with hue values between `0-10` and `160-180` using the `cv2.inRange()` function. These two masks are combined using the `cv2.bitwise_or()` function to create a single mask for the red pixels. The mask is applied to the hue and saturation channels to remove the red pixels from the image. The modified hue and saturation channels are combined with the original value channel to create the modified `HSV` image. The modified `HSV` image is converted back to the `RGB` color space using OpenCV. The original image, red mask, and modified image with masked red pixels are displayed using Matplotlib.

Important codes to remember:

- `cv2.inRange()`
  - This function is used to create a binary mask for a specific color range in an image.

  - Syntax: `cv2.inRange(src, lowerb, upperb[, dst]) -> dst`
- `cv2.bitwise_or()`
  - This function is used to perform a bitwise OR operation between two arrays.
  - Syntax: `cv2.bitwise_or(src1, src2[, dst[, mask]]) -> dst`
- `cv2.merge()`
  - This function is used to merge multiple single-channel arrays into a multi-channel array.
  - Syntax: `cv2.merge(mv[, dst]) -> dst`

10 Masking Green Pixels in an Image

We can mask green pixels in an image by converting the image to the HSV color space and applying a mask to the hue channel. The mask is created by defining the lower and upper bounds of the green color range in the hue channel. The mask is then applied to the hue and saturation channels to remove the green pixels from the image.

Read More:

- Color Mask

[19]:
```
# Convert image to HSV
hsv_image = cv2.cvtColor(original_image, cv2.COLOR_RGB2HSV)

# Divide the HSV image into its three color channels
channel_h, channel_s, channel_v = cv2.split(hsv_image)

# Define the lower and upper bounds of the green color range
lower_green = np.array([36, 40, 40])
upper_green = np.array([72, 255, 255])
mask_green = cv2.inRange(hsv_image, lower_green, upper_green)

# Apply the mask to the hue and saturation channels
channel_h[mask_green == 0] = 0
channel_s[mask_green == 0] = 0

# Combine the modified hue and saturation channels with the original value␣
 ↪channel
masked_HSV_green_image = cv2.merge([channel_h, channel_s, channel_v])

# Convert the modified HSV image to the RGB color space using the cv2.
 ↪cvtColor() function
masked_RGB_green_image = cv2.cvtColor(masked_HSV_green_image, cv2.COLOR_HSV2RGB)

# Display the modified image
print("Masking Green Pixels in the Image")
plt.figure('Masking Green Pixels in the Image')

plt.subplot(1, 3, 1)
plt.imshow(original_image)
```

```
plt.title('(a) Original Image', y=-0.2)
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(mask_green, cmap='gray')
plt.title('(b) Masked Area', y=-0.2)
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(masked_RGB_green_image)
plt.title('(c) Masked Green Pixels', y=-0.2)
plt.axis('off')

plt.show()
```

```
Masking Green Pixels in the Image
```

The code above masks the green pixels in the image using the `HSV` color model. The color green is defined by the hue values between `36` and `72`.

11 Masking Blue Pixels in an Image

We can mask blue pixels in an image by converting the image to the HSV color space and pplying a mask to the hue channel. The mask is created by defining the lower and upper bounds of the blue color range in the hue channel. The mask is then applied to the hue and saturation channels to remove the blue pixels from the image.

Read More:

- Color Mask

[20]:
```
# Convert image to HSV
hsv_image = cv2.cvtColor(original_image, cv2.COLOR_RGB2HSV)

# Divide the HSV image into its three color channels
```

```
channel_h, channel_s, channel_v = cv2.split(hsv_image)

# Define the lower and upper bounds of the blue color range
lower_blue = np.array([84, 40, 40])
upper_blue = np.array([140, 255, 255])
mask_blue = cv2.inRange(hsv_image, lower_blue, upper_blue)

# Apply the mask to the hue and saturation channels
channel_h[mask_blue == 0] = 0
channel_s[mask_blue == 0] = 0

# Combine the modified hue and saturation channels with the original value␣
 ↪channel
masked_HSV_blue_image = cv2.merge([channel_h, channel_s, channel_v])

# Convert the modified HSV image to the RGB color space using the cv2.
 ↪cvtColor() function
masked_RGB_blue_image = cv2.cvtColor(masked_HSV_blue_image, cv2.COLOR_HSV2RGB)

# Display the modified image
print("Masking Blue Pixels in the Image")
plt.figure('Masking Blue Pixels in the Image')

plt.subplot(1, 3, 1)
plt.imshow(original_image)
plt.title('(a) Original Image', y=-0.2)
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(mask_blue, cmap='gray')
plt.title('(b) Masked Area', y=-0.2)
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(masked_RGB_blue_image)
plt.title('(c) Masked Blue Pixels', y=-0.2)
plt.axis('off')

plt.show()
```

```
Masking Blue Pixels in the Image
```

(a) Original Image (b) Masked Area (c) Masked Blue Pixels

The code above masks the blue pixels in the image using the `HSV` color model. It uses the hue color range from `84` to `140` to define the blue color range.

12 Modifying the Hue of an Image

We can modify the hue of an image by converting the image to the `HSV` color space and applying a transformation to the hue channel. The transformation is applied to the hue channel by adding or subtracting a constant value from each pixel in the channel.

In the example below, we modify the hue of an image by adding a constant value to the hue channel. We then combine the modified hue channel with the original saturation and value channels to create a new `HSV` image. Finally, we convert the new `HSV` image to the `RGB` color space to display the modified image.

Note: *The hue channel in the HSV color space is represented as an angle between 0 and 360 degrees. Therefore, when adding or subtracting a constant value from the hue channel, we need to ensure that the resulting hue values are within the range of 0 to 360 degrees.*

Read More:

- Hue

[21]:
```
# Convert image to HSV
hsv_image = cv2.cvtColor(original_image, cv2.COLOR_RGB2HSV)

# Divide the HSV image into its three color channels
channel_h, channel_s, channel_v = cv2.split(hsv_image)

# Modify the hue channel by adding a constant value
hue_shift = 60
channel_h = (channel_h + hue_shift) % 180

# Combine the modified hue and saturation channels with the original value␣
↪channel
hue_shift_HSV_image = cv2.merge([channel_h, channel_s, channel_v])
```

```
# Convert the modified HSV image to the RGB color space using the cv2.
  ↪cvtColor() function
hue_shift_RGB_image = cv2.cvtColor(hue_shift_HSV_image, cv2.COLOR_HSV2RGB)

# Display the modified image
print("Shifting the Hue of the Image")
plt.figure('Shifting the Hue of the Image')

plt.subplot(1, 2, 1)
plt.imshow(original_image)
plt.title('(a) Original Image', y=-0.15)
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(hue_shift_RGB_image)
plt.title(f'(b) Hue Shifted by {hue_shift} Degrees', y=-0.15)
plt.axis('off')

plt.show()
```

```
Shifting the Hue of the Image
```

The above code shifts the hue of the image by adding a constant value to the hue channel. This results in a change in the color of the image, as the hue value determines the color of the pixels. The hue values are in the range of `0` to `179`, and since the maximum hue value in OpenCV is `179`, we use the modulo operator to ensure that the hue values remain within this range.

13 Selective Color Modification

We can selectively modify the color of an image by converting the image to the HSV color space and applying a mask to the hue channel. The mask is created by defining the lower and upper bounds of the color range in the hue channel. The mask is then applied to the hue channel to modify the color of the pixels within the specified range.

Read More: - Color Mask

[22]:
```
# Convert image to HSV
hsv_image = cv2.cvtColor(original_image, cv2.COLOR_RGB2HSV)

# Divide the HSV image into its three color channels
channel_h, channel_s, channel_v = cv2.split(hsv_image)

# Define the lower and upper bounds of the color range

# Mask 1 are the red pixels with hue values between 0 and 10
lower_color = np.array([0, 100, 100])
upper_color = np.array([10, 255, 255])
mask_color_1 = cv2.inRange(hsv_image, lower_color, upper_color)

# Mask 2 are the red pixels with hue values between 160 and 180
lower_color = np.array([160, 100, 100])
upper_color = np.array([180, 255, 255])
mask_color_2 = cv2.inRange(hsv_image, lower_color, upper_color)

# Combine the two masks using the cv2.bitwise_or() function
mask_color = cv2.bitwise_or(mask_color_1, mask_color_2)

# Modify the hue channel by adding a constant value to the pixels within the␣
 ↪color range
hue_shift = 15
channel_h[mask_color > 0] = (channel_h[mask_color > 0] + hue_shift) % 180

# Combine the modified hue and saturation channels with the original value␣
 ↪channel
color_change_HSV_image = cv2.merge([channel_h, channel_s, channel_v])

# Convert the modified HSV image to the RGB color space using the cv2.
 ↪cvtColor() function
color_change_RGB_image = cv2.cvtColor(color_change_HSV_image, cv2.COLOR_HSV2RGB)

# Display the modified image
print("Changing the Color of a Specific Range of Pixels in the Image")
plt.figure('Changing the Color of a Specific Range of Pixels in the Image')

plt.subplot(1, 3, 1)
```

```
plt.imshow(original_image)
plt.title('(a) Original Image', y=-0.2)
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(mask_color, cmap='gray')
plt.title('(b) Masked Area', y=-0.2)
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(color_change_RGB_image)
plt.title('(c) Color Change', y=-0.2)
plt.axis('off')

plt.show()
```

Changing the Color of a Specific Range of Pixels in the Image

The code above modifies the hue value of a particular color range in an image. The code reads an image from a file and converts it to the `HSV` color space. It then divides the `HSV` image into its three color channels: hue, saturation, and value. The code defines the lower and upper bounds of the color range to be modified where, in this example, the color range is red with hue values between `0` and `10` and `160` and `180`. It creates two masks to identify the pixels within the color range. The code combines the two masks using the `cv2.bitwise_or()` function. It modifies the hue channel by adding a constant value to the pixels where, in this case, the hue value is increased by 15. The code then combines the modified hue and saturation channels with the original value channel. Finally, the code converts the modified `HSV` image to the `RGB` color space using the `cv2.cvtColor()` function and displays the modified image.

14 YUV Color Family

The `YUV` color family is a color model used in video compression and television broadcasting.

It consists of three components:

- Y (luminance)
  - Represents the brightness of the color
  - Similar to the grayscale image
- U (chrominance, blue projection)
- V (chrominance, red projection)

The Y component represents the brightness of the color, while the U and V components represent the color information.

The `YUV` color family is used to represent colors in a way that is more efficient for video compression than the RGB color model. It separates the brightness and color information of the image, allowing for better compression and transmission of video data.

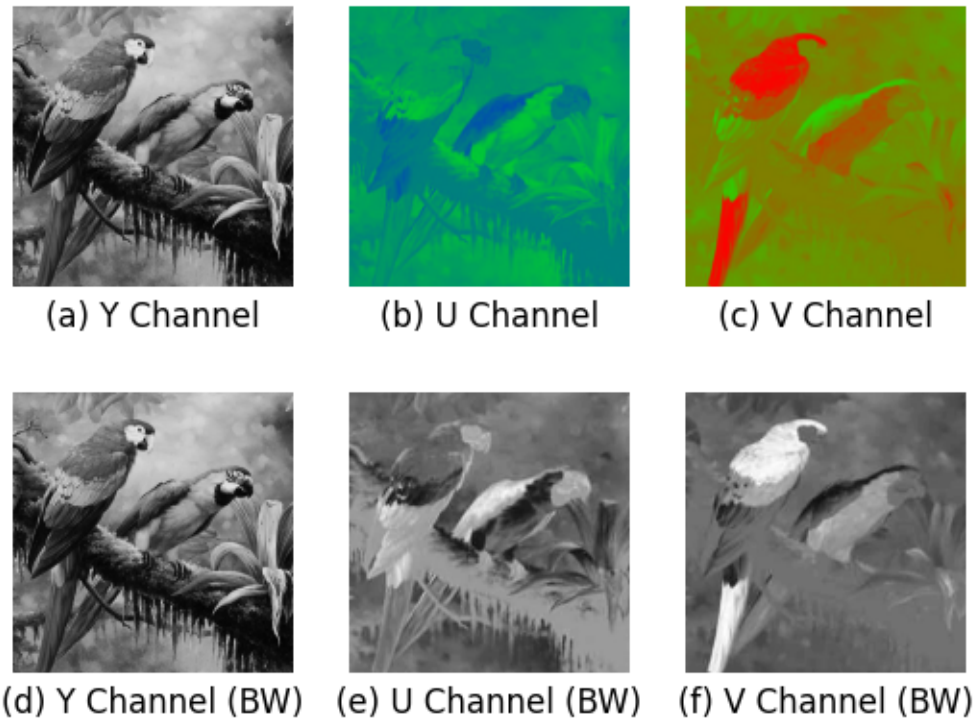
Read More:

- YUV Color Model

[11]:
```
# Convert image to YUV
yuv_image = cv2.cvtColor(original_image, cv2.COLOR_RGB2YUV)

# Divide the YUV image into its three color channels
channel_y, channel_u, channel_v = cv2.split(yuv_image)

# Create LUTs for the U and V channels
lut_u = np.array([[[0,255-i,i] for i in range(256)]],dtype=np.uint8)
lut_v = np.array([[[i,255-i,0] for i in range(256)]],dtype=np.uint8)

# Convert the U and V channels to RGB color space using the LUTs
y_rgb = cv2.cvtColor(channel_y, cv2.COLOR_GRAY2RGB)
u_rgb = cv2.cvtColor(channel_u, cv2.COLOR_GRAY2RGB)
v_rgb = cv2.cvtColor(channel_v, cv2.COLOR_GRAY2RGB)

# Apply the LUTs to the U and V channels
u_mapped = cv2.LUT(u_rgb, lut_u)
v_mapped = cv2.LUT(v_rgb, lut_v)

# Displaying the three color channels
print("Dividing the YUV Image into its Three Color Channels")
plt.figure('Dividing the YUV Image into its Three Color Channels')

# Display the Y Channel
plt.subplot(2, 3, 1)
plt.imshow(y_rgb)
plt.title('(a) Y Channel', y=-0.2)
plt.axis('off')

plt.subplot(2, 3, 2)
plt.imshow(u_mapped)
plt.title('(b) U Channel', y=-0.2)
```

```
plt.axis('off')

plt.subplot(2, 3, 3)
plt.imshow(v_mapped)
plt.title('(c) V Channel', y=-0.2)
plt.axis('off')

plt.subplot(2, 3, 4)
plt.imshow(channel_y, cmap='gray')
plt.title('(d) Y Channel (BW)', y=-0.2)
plt.axis('off')

plt.subplot(2, 3, 5)
plt.imshow(channel_u, cmap='gray')
plt.title('(e) U Channel (BW)', y=-0.2)
plt.axis('off')

plt.subplot(2, 3, 6)
plt.imshow(channel_v, cmap='gray')
plt.title('(f) V Channel (BW)', y=-0.2)
plt.axis('off')

plt.show()
```

Dividing the YUV Image into its Three Color Channels



The above code converts an image from the `RGB` color space to the `YUV` color space using the `cv2.cvtColor()` function. It then divides the `YUV` image into its three color channels: `Y` (luminance), `U` (chrominance), and `V` (chrominance). The three color channels are displayed using Matplotlib with the original channel values and the grayscale channel values.

15 CIE L*a*b* Color Model

The `CIE L*a*b*` color model is a color model used in color science and color management. `CIE` stands for the International Commission on Illumination or Commission Internationale de l'Eclairage in French, an organization that develops standards for color science.

The `CIE L*a*b*` color model consists of three components:

- L* (lightness)
  - Represents the brightness of the color
- a* (green-red)
  - a* = (green - red) component
- b* (blue-yellow)
  - b* = (blue - yellow) component

The `L*` component represents the lightness of the color, while the `a*` and `b*` components represent the color information. The `CIE L*a*b*` color model is used to represent colors in a way that is more perceptually uniform than the RGB color model. It is often used in color science, color management, and color correction applications.

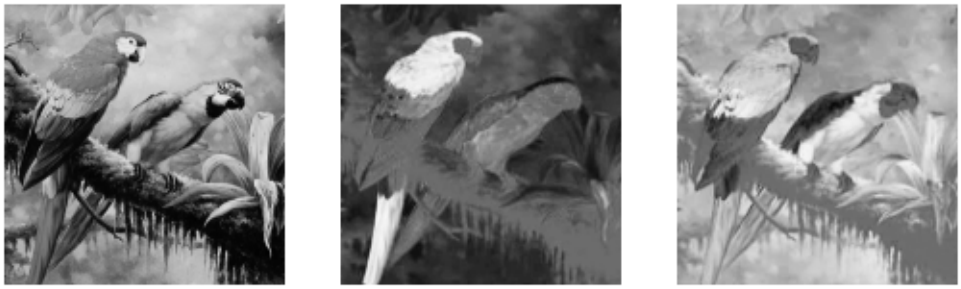
Read More:

- CIE L*a*b*

[12]:
```
# Convert image to CIE L*a*b*
lab_image = cv2.cvtColor(original_image, cv2.COLOR_RGB2LAB)

# Divide the CIE L*a*b* image into its three color channels
channel_l, channel_a, channel_b = cv2.split(lab_image)

# Displaying the three color channels
print("Dividing the CIE L*a*b* Image into its Three Color Channels")
plt.figure('Dividing the CIE L*a*b* Image into its Three Color Channels')

plt.subplot(1, 3, 1)
plt.imshow(channel_l, cmap='gray')
plt.title('(a) L* Channel', y=-0.2)
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(channel_a, cmap='gray')
plt.title('(b) a* Channel', y=-0.2)
```

```
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(channel_b, cmap='gray')
plt.title('(c) b* Channel', y=-0.2)
plt.axis('off')

plt.show()
```

Dividing the CIE L*a*b* Image into its Three Color Channels



The code above converts an image from the RGB color space to the CIE L*a*b* color space using the cv2.cvtColor() function. It then divides the CIE L*a*b* image into its three color channels, namely the L* (luminance), a* (green-red), and b* (blue-yellow) channels using the cv2.split() function. The three color channels along with their grayscale versions are displayed using the plt.imshow() function from the Matplotlib library.

16 Summary

In this section, we learned about the RGB, HSV, YUV, and CIE L*a*b* color models used in image processing. We learned how to convert images from one color model to another using OpenCV and Matplotlib. We also learned how to manipulate the color channels of an image and apply masks to selectively modify the color of an image.

The RGB color model is used to represent colors in terms of red, green, and blue components. The HSV color model is used to represent colors in terms of hue, saturation, and value components. The YUV color model is used to represent colors in terms of luminance and chrominance components. The CIE L*a*b* color model is used to represent colors in terms of lightness, green-red, and blue-yellow components.

Color conversion is an essential part of image processing and computer vision. It allows us to represent colors in different ways and perform color-based operations on images. Understanding the different color models and how to convert images between them is important for developing image processing algorithms and applications.

Masking and selective color modification are common techniques used in image processing to isolate

and modify specific colors in an image. These techniques are used in various applications such as image segmentation, color correction, and color enhancement.

17 References

- Thomas G. (2022). Graphic Designing: A Step-by-Step Guide (Advanced). Larsen & Keller. ISBN: 978-1-64172-536-1
- Singh M. (2022). Computer Graphics and Multimedia. Random Publications LLP. ISBN: 978-93-93884-95-4
- Singh M. (2022). Computer Graphics Science. Random Publications LLP. ISBN: 978-93-93884-03-9
- Singh M. (2022). Computer Graphics Software. Random Publications LLP. ISBN: 9789393884114
- Tyagi, V. (2021). Understanding Digital Image Processing. CRC Press.
- Ikeuchi, K. (Ed.). (2021). Computer Vision: A Reference Guide (2nd ed.). Springer.
- Bhuyan, M. K. (2020). Computer Vision and Image Processing. CRC Press.
- Howse, J., & Minichino, J. (2020). Learning OpenCV 4 Computer Vision with Python 3: Get to grips with tools, techniques, and algorithms for computer vision and machine learning. Packt Publishing Ltd.
- Kinser, J. M. (2019). Image Operators: Image Processing in Python. CRC Press.