Topic 7: Applying color manipulation techniques

1: Title with the topic "Applying color manipulation techniques using Python" and a relevant background image.

title of the presentation and does not contain any specific content related to color manipulation techniques using Python. It typically includes the title of the presentation, the presenter's name and affiliation, and any relevant visual elements.

However, to provide an example of Python code that could be used in this presentation, here is a simple script for displaying an image using Python's Matplotlib library:

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

img = mpimg.imread('example_image.jpg')
plt.imshow(img)
plt.show()
```

This code imports the necessary libraries, reads in an image file, and displays it using Matplotlib's imshow function. This basic code can be used as a starting point for more complex image processing and color manipulation techniques using Python.

2: Introduction that provides an overview of the importance of color manipulation in digital images and its relevance in various fields such as photography, graphic design, and data visualization.

introduction slide of the presentation and provides an overview of the importance of color manipulation in digital images and its relevance in various fields such as photography, graphic design, and data visualization.

Here's an example of Python code that demonstrates the importance of color manipulation in image processing:

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

# Create a grayscale image with a color gradient
img = np.zeros((100, 100))
for i in range(100):
    img[i, :] = i / 100

# Display the original image
plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')

# Apply a color map to the image
plt.subplot(1, 2, 2)
```

```
plt.imshow(img, cmap='jet')
plt.title('Color Map Applied')

plt.show()
```

This code creates a grayscale image with a color gradient and displays it using Matplotlib's imshow function. It then applies a color map to the same image and displays it again. This example demonstrates how color manipulation can be used to enhance the visual quality of an image and improve its interpretation.

3: Basic concepts of color manipulation, including color models such as RGB and HSV, and color space conversions between them.
Here's an example of Python code that demonstrates color space conversions using Python's colorsys library:

```
import colorsys

# Define an RGB color
r, g, b = 128, 64, 32

# Convert RGB to HSV
h, s, v = colorsys.rgb_to_hsv(r/255, g/255, b/255)

# Convert back to RGB
r2, g2, b2 = colorsys.hsv_to_rgb(h, s, v)

# Print the original and converted colors
print(f"Original RGB: ({r}, {g}, {b})")
print(f"Converted HSV: ({h*360}, {s*100}, {v*100})")
print(f"Converted RGB: ({int(r2*255)}, {int(g2*255)}, {int(b2*255)})")
```

This code defines an RGB color and then converts it to the HSV color space using rgb_to_hsv. It then converts the color back to RGB using hsv_to_rgb. Finally, it prints the original RGB color and the converted HSV and RGB colors to the console. This example demonstrates how Python's colorsys library can be used to perform color space conversions in image processing.

4: Explanation of the color histogram, its importance in image processing, and how it can be used to manipulate and adjust the color balance in an image. This slide may include a brief overview of each technique along with examples of Python code that demonstrate their implementation. For example:

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

# Read in an image file
img = plt.imread('example_image.jpg')

# Adjust brightness
bright_img = np.clip(img * 1.5, 0, 1)
```

```
# Adjust contrast
gray_img = np.dot(img[...,:3], [0.2989, 0.5870, 0.1140])
mean = np.mean(gray_img)
cont_img = np.clip((img - mean) * 1.5 + mean, 0, 1)

# Adjust saturation
hsv_img = colorsys.rgb_to_hsv(img[...,0], img[...,1], img[...,2])
saturated_img = colorsys.hsv_to_rgb(hsv_img[0], hsv_img[1] * 1.5, hsv_img[2])

# Display the original and manipulated images
plt.subplot(2, 2, 1)
plt.imshow(img)
plt.title('Original Image')

plt.subplot(2, 2, 2)
plt.imshow(bright_img)
plt.title('Brightness Adjusted')

plt.subplot(2, 2, 3)
plt.imshow(cont_img)
plt.title('Contrast Adjusted')

plt.subplot(2, 2, 4)
plt.imshow(saturated_img)
plt.title('Saturation Adjusted')

plt.show()
```

This code demonstrates how brightness, contrast, and saturation can be adjusted using simple mathematical operations on the image data. The clip function is used to ensure that the resulting image values are within the valid range of 0 to 1. The example also includes an implementation of adjusting contrast using grayscale conversion and saturation adjustment using the colorsys library. These examples demonstrate how Python can be used to implement various color manipulation techniques for digital images.

5: Code example for creating a color histogram using Python's Matplotlib library. For example:

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

# Read in an image file
img = cv2.imread('example_image.jpg')

# Perform color correction using white balancing
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray = cv2.medianBlur(gray, 5)
```

```
edges = cv2.Canny(gray, 100, 200)
color_corrected = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)

# Perform color grading using lookup tables
color_table = np.zeros((256, 1, 3), dtype=np.uint8)
color_table[:, :, 0] = np.arange(256)
color_table[:, :, 1] = np.ones((256, 1)) * 100
color_table[:, :, 2] = np.ones((256, 1)) * 200
color_graded = cv2.LUT(img, color_table)

# Perform image segmentation using k-means clustering
data = img.reshape((-1, 3)).astype(np.float32)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
K = 4
ret, label, center = cv2.kmeans(data, K, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
center = np.uint8(center)
segmented_img = center[label.flatten()].reshape((img.shape))

# Display the original and manipulated images
plt.subplot(2, 2, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Original Image')

plt.subplot(2, 2, 2)
plt.imshow(cv2.cvtColor(color_corrected, cv2.COLOR_BGR2RGB))
plt.title('Color Corrected')

plt.subplot(2, 2, 3)
plt.imshow(cv2.cvtColor(color_graded, cv2.COLOR_BGR2RGB))
plt.title('Color Graded')

plt.subplot(2, 2, 4)
plt.imshow(cv2.cvtColor(segmented_img, cv2.COLOR_BGR2RGB))
plt.title('Segmented Image')

plt.show()
```

This code demonstrates how advanced color manipulation techniques can be implemented using Python and OpenCV library. The example includes color correction using white balancing, color grading using lookup tables, and image segmentation using k-means clustering. These techniques demonstrate the power and versatility of Python and OpenCV in advanced image processing and analysis.

6: Explanation of the concept of color grading and how it can be used to adjust and enhance the color balance in an image. For example:

```
import cv2

# Read in an image file
img = cv2.imread('example_photo.jpg')

# Convert the image to LAB color space
lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)

# Apply histogram equalization to the L channel
l, a, b = cv2.split(lab)
l = cv2.equalizeHist(l)
lab = cv2.merge((l, a, b))

# Convert the image back to RGB color space
rgb = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)

# Adjust the color balance using a curve adjustment
out = cv2.LUT(rgb, lut)

# Display the original and manipulated images
cv2.imshow('Original', img)
cv2.imshow('Color Graded', out)
cv2.waitKey(0)
```

In this example, we first convert the image from the BGR color space to the LAB color space, which separates the image into three channels: L for luminance, A for green-magenta balance, and B for blue-yellow balance. We then apply histogram equalization to the L channel to adjust the contrast and brightness of the image. Next, we convert the image back to the BGR color space and use a curve adjustment technique to further adjust the color balance. Finally, we display the original and manipulated images.

Color grading is a powerful technique that can significantly enhance the visual impact and storytelling of an image or footage. By using Python and its libraries such as OpenCV, color grading can be applied to images and videos with ease, allowing photographers and videographers to achieve their desired look and feel.

7: Code example for color grading an image using Python's Scikit-Image library.

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import exposure
from skimage import io

# Load the image
image = io.imread('example_photo.jpg')

# Apply a gamma correction to increase the brightness of the image
image = exposure.adjust_gamma(image, gamma=1.5)
```

```
# Apply histogram equalization to increase the contrast of the image
image = exposure.equalize_hist(image)

# Apply color balance adjustment to achieve a desired look
r, g, b = np.mean(image[:, :, 0]), np.mean(image[:, :, 1]), np.mean(image[:, :, 2])
gray_world = [r, g, b]
image = image / gray_world * [1, 1.2, 1.5]

# Display the original and color graded images
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
ax = axes.ravel()
ax[0].imshow(io.imread('example_photo.jpg'))
ax[0].set_title('Original')
ax[1].imshow(image)
ax[1].set_title('Color Graded')
plt.show()
```

In this example, we first load the image using the io.imread() function from Scikit-Image. We then apply a gamma correction using the exposure.adjust_gamma() function to increase the brightness of the image. Next, we apply histogram equalization using the exposure.equalize_hist() function to increase the contrast of the image. Finally, we use a color balance adjustment technique to adjust the color balance of the image by dividing it by the mean of the R, G, and B channels and multiplying it by a desired factor for each channel.

The color balance adjustment can be further customized to achieve different looks and feels by adjusting the multiplication factors for each channel. The output of the code example is two images side by side, the original image and the color graded image.

8: Explanation of the concept of color quantization and how it can be used to reduce the number of colors in an image.

9: Code example for performing color quantization using Python's Scikit-Learn library.

```
import numpy as np
from sklearn.cluster import KMeans
from skimage import io

# Load the image
image = io.imread('example_photo.jpg')

# Flatten the image into a 2D array of pixels
pixels = np.reshape(image, (-1, 3))

# Perform color quantization using KMeans clustering
n_colors = 16
kmeans = KMeans(n_clusters=n_colors, random_state=0).fit(pixels)
quantized_pixels = kmeans.cluster_centers_[kmeans.labels_]

# Reshape the quantized pixels back into the original image shape
quantized_image = np.reshape(quantized_pixels, image.shape)
```

```
# Display the original and quantized images
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
ax = axes.ravel()
ax[0].imshow(io.imread('example_photo.jpg'))
ax[0].set_title('Original')
ax[1].imshow(quantized_image)
ax[1].set_title('Quantized')
plt.show()
```

In this example, we first load the image using the io.imread() function from Scikit-Image. We then flatten the image into a 2D array of pixels using the np.reshape() function. We perform color quantization using the KMeans clustering algorithm from the Scikit-Learn library, where we specify the number of colors to quantize to using the n_clusters parameter. We reshape the quantized pixels back into the original image shape using the np.reshape() function.

The output of the code example is two images side by side, the original image and the quantized image. The quantized image has a reduced color palette with the specified number of colors, which can be useful for reducing image size or for creating a specific visual effect.

10: Explanation of the concept of image filtering, how it can be used to smooth or blur an image, and various types of filters that can be used.
Image filtering is a technique used to modify an image by convolving it with a filter kernel, also known as a mask or window. The filter kernel is a small matrix of numbers that is used to perform a convolution operation on the image pixels. The result of the convolution operation is a new image where each pixel is a weighted sum of its neighboring pixels, with the weights defined by the filter kernel.

Image filtering can be used for various purposes, including smoothing or blurring an image, sharpening an image, or enhancing certain features in an image. Smoothing or blurring an image can be useful for removing noise or reducing detail, while sharpening an image can be useful for enhancing edges or details in an image.

There are various types of filters that can be used for image filtering, including:

Gaussian filter: This filter is used for smoothing an image and reducing noise. It applies a Gaussian distribution over the pixel values, with higher weights given to pixels closer to the center of the filter.

Median filter: This filter is used for removing noise from an image by replacing each pixel value with the median value of its neighboring pixels.

Bilateral filter: This filter is used for smoothing an image while preserving edges. It applies a Gaussian distribution over both the spatial distance and intensity difference between neighboring pixels, with higher weights given to pixels that are closer in both distance and intensity.

Laplacian filter: This filter is used for sharpening an image by enhancing the edges and details in an image. It calculates the second derivative of the image with respect to the x and y coordinates, and then combines these derivatives to highlight the edges in the image.

In Python, image filtering can be performed using various libraries such as Scikit-Image, OpenCV, and Pillow. The specific implementation of image filtering will depend on the library and the type of filter being used.

11: Code example for generating a random RGB color using Python's NumPy and Matplotlib libraries.

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

# Generate a random RGB color
color = np.random.rand(3)

# Plot a square of the generated color
fig, ax = plt.subplots()
ax.add_patch(plt.Rectangle((0, 0), 1, 1, color=color))
ax.axis('off')
plt.show()
```

In this example, we first import the NumPy and Matplotlib libraries. We then generate a random RGB color using the np.random.rand() function from NumPy, which returns an array of random numbers between 0 and 1 with the specified shape. In this case, we generate an array of shape (3,), which represents the RGB color values.

We then plot a square of the generated color using the plt.Rectangle() function from Matplotlib. The color parameter is set to the generated color, and the ax.axis('off') line is used to remove the axes from the plot.

The output of the code example is a square with a randomly generated RGB color. The color of the square will be different each time the code is run due to the random generation of the color values.

12: Code example for converting an RGB color to HSV color space using Python's Colorsys library.

```
import colorsys

# Define an RGB color
rgb_color = (0.2, 0.5, 0.8)

# Convert RGB to HSV
hsv_color = colorsys.rgb_to_hsv(*rgb_color)

# Print the original RGB color and the converted HSV color
print(f"Original RGB color: {rgb_color}")
print(f"Converted HSV color: {hsv_color}")
```

In this example, we first import the colorsys library, which provides functions for converting between different color spaces. We then define an RGB color as a tuple of three values between 0 and 1, representing the red, green, and blue color components.

We then use the colorsys.rgb_to_hsv() function to convert the RGB color to HSV color space. The function takes three arguments representing the red, green, and blue color components, which are passed as separate arguments using the * operator.

Finally, we print the original RGB color and the converted HSV color using Python's f-strings. The output of the code example will show the original RGB color and the converted HSV color as tuples of three values each.

13: Code example for applying a blur filter to an image using Python's PIL (Pillow) library.

```
from PIL import Image, ImageFilter

# Open the input image
input_image = Image.open("input_image.jpg")

# Apply a Gaussian blur filter
blur_image = input_image.filter(ImageFilter.GaussianBlur(radius=5))

# Save the blurred image
blur_image.save("blurred_image.jpg")
```

In this example, we first import the Image and ImageFilter modules from the Pillow library, which provides functions for working with images. We then open the input image using the Image.open() function, passing the filename of the image as a string.

We then apply a Gaussian blur filter to the image using the filter() function of the Image object, passing an instance of the GaussianBlur class from the ImageFilter module as an argument. The radius parameter of the GaussianBlur class is set to 5, which specifies the amount of blurring to apply.

Finally, we save the blurred image using the save() function of the Image object, passing the filename of the output image as a string. The output of the code example is a blurred version of the input image saved to a file.

16: Code example for adjusting the color balance of an image using Python's PIL (Pillow) library.

```
from PIL import ImageOps, ImageEnhance, Image

# Open the input image
input_image = Image.open("input_image.jpg")

# Adjust the color balance
color_balance = ImageOps.colorize(input_image.convert("L"), (0, 0, 0), (255, 128, 128))

# Enhance the color balance
enhancer = ImageEnhance.Color(color_balance)
color_balance_enhanced = enhancer.enhance(2)
```

 # Save the adjusted image
color_balance_enhanced.save("color_balance_image.jpg")

In this example, we first import the ImageOps, ImageEnhance, and Image modules from the Pillow library, which provide functions for working with images. We then open the input image using the Image.open() function, passing the filename of the image as a string.

We then adjust the color balance of the image using the ImageOps.colorize() function. First, we convert the image to grayscale using the convert() method, passing the string "L" as an argument to specify grayscale mode. We then pass the grayscale image, a tuple representing the color of the shadows (in RGB format), and a tuple representing the color of the highlights (in RGB format) as arguments to the colorize() function.

Next, we enhance the color balance using the ImageEnhance.Color() class. We create an instance of the Color class, passing the color balance image as an argument. We then use the enhance() method of the Color instance to enhance the color balance, passing a value of 2 as an argument to increase the saturation of the colors.

Finally, we save the adjusted image using the save() function of the Image object, passing the filename of the output image as a string. The output of the code example is an image with adjusted color balance saved to a file.