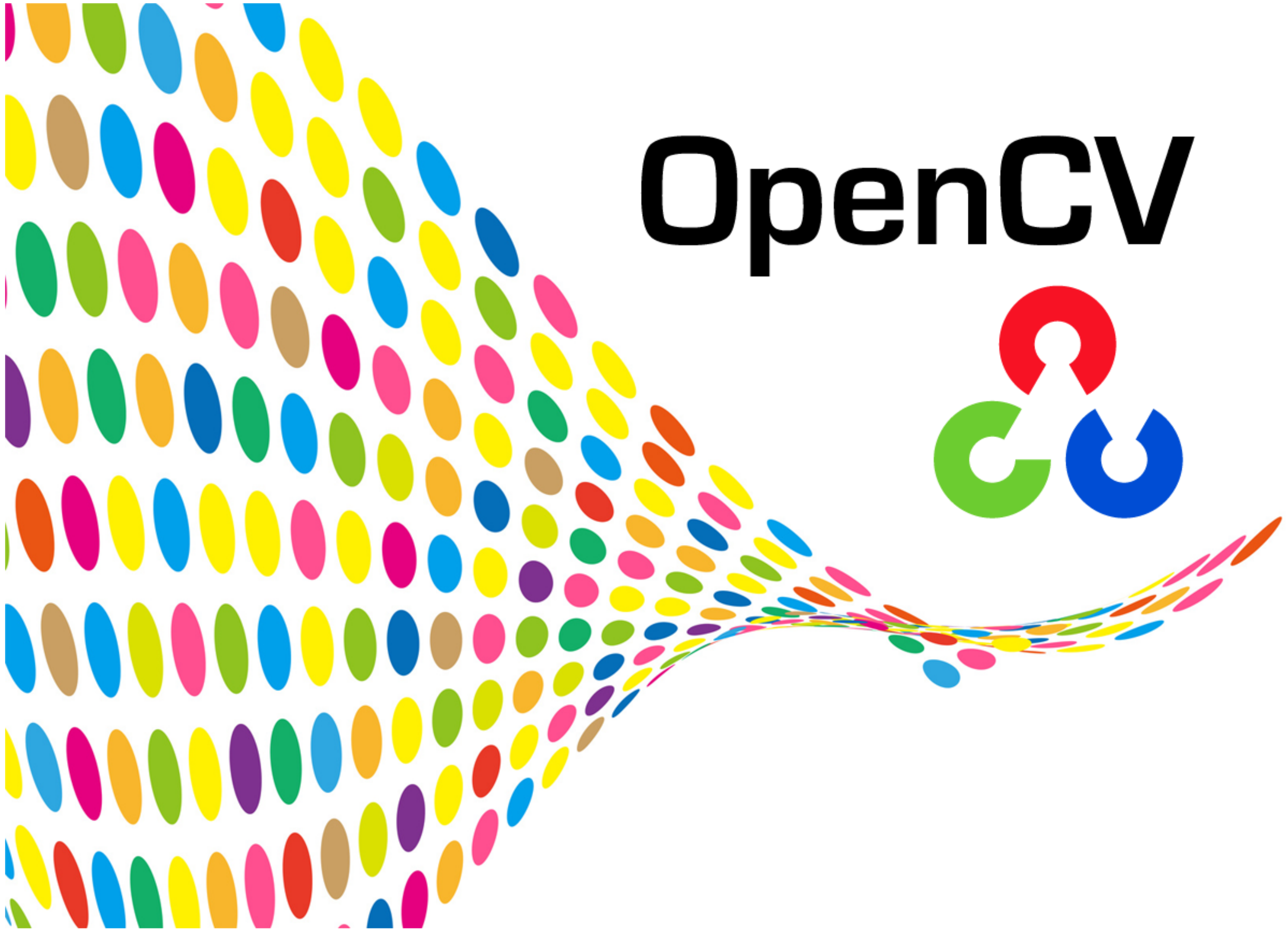
SESSION 1 - Image Processing & OpenCV

TOC :

---

1. Introduction

- Image processing is a field of computer vision that involves manipulating and analyzing digital images to extract useful information or enhance their visual quality. OpenCV (Open Source Computer Vision) is a popular library that provides a comprehensive set of functions and tools for image processing.



2. Installation and Setup of OpenCV

Before we start with image processing and manipulation with OpenCV, we need to install and set it up on our system.

You can install OpenCV using pip: `pip install opencv-python`

Once OpenCV is installed, you can import it in your Python code as follows: `import cv2`

3. Image Processing

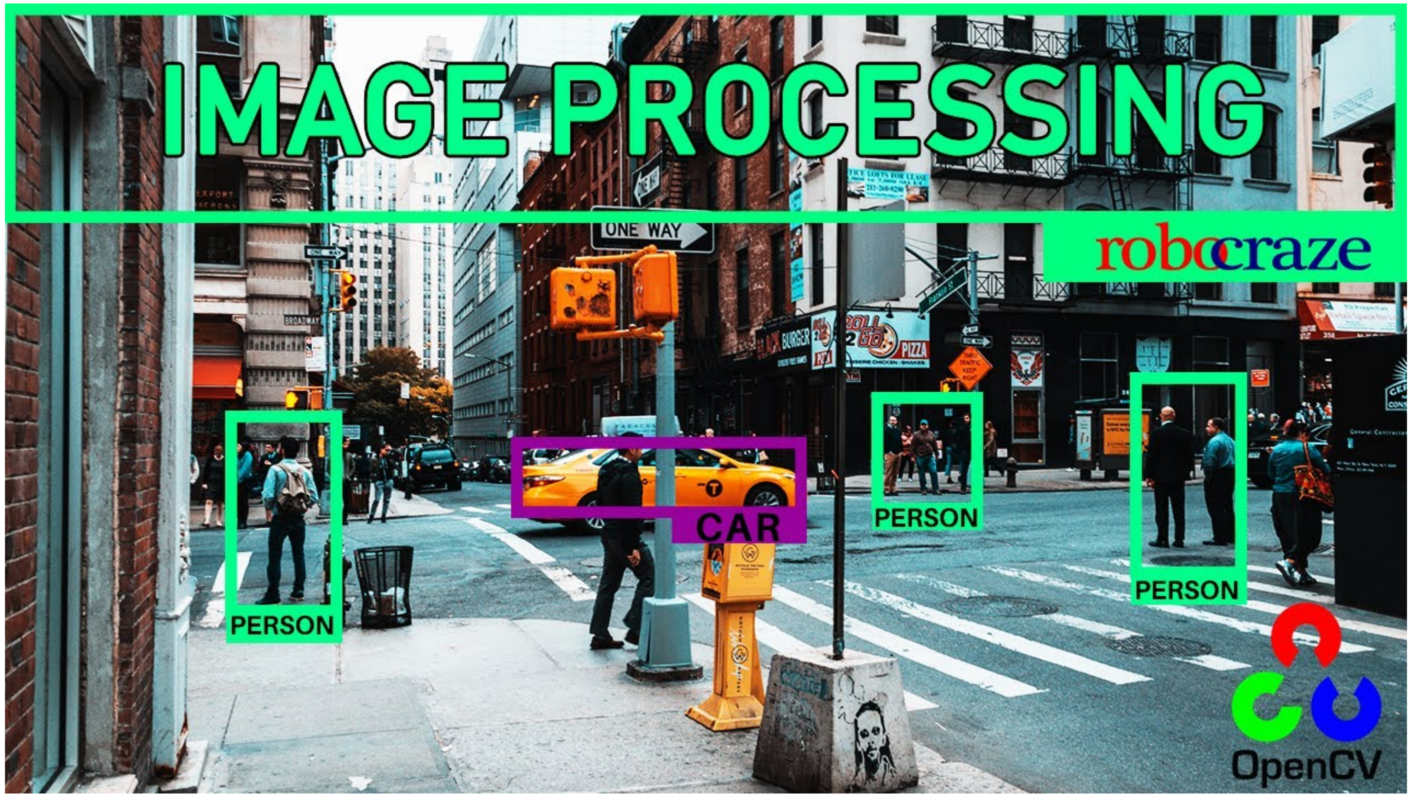


Image Acquisition & Other Basic Operations

- `Reading & Copying Images :` To acquire and represent an image in Python, we can use the OpenCV library. OpenCV provides functions for reading, displaying, and saving images in various formats.

  Here's an example code snippet to read and display an image :

In [ ]:
```
import cv2

# Read image
img = cv2.imread('Images/image.jpg')

# Convert image to grayscale
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Display grayscale image
```

```
cv2.imshow('Image', img)
cv2.imshow('Grayscale Image', gray_img)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Save grayscale image
cv2.imwrite('Images/image_gray.jpg', gray_img)
print("Grayscale image saved successfully.")
```

```
Grayscale image saved successfully.
```

- Resizing an image

In [ ]:

```
import cv2

# Read an image
img = cv2.imread('Images/image.jpg')

new_width = 400
new_height = 300

# Resize the image
resized = cv2.resize(img, (new_width, new_height))

# Display the resized image
cv2.imshow('resized', resized)

# Wait for a key event to close the window
cv2.waitKey(0)

# Destroy all windows
cv2.destroyAllWindows()
```

- Cropping an image

In [ ]:

```
import cv2

# Read an image
img = cv2.imread('Images/image.jpg')

# Define the region of interest (ROI) for cropping
```

```
x, y, width, height = 100, 100, 200, 200
roi = img[y:y+height, x:x+width]

# Display the cropped image
cv2.imshow('cropped', roi)

# Wait for a key event to close the window
cv2.waitKey(0)

# Destroy all windows
cv2.destroyAllWindows()
```

- Converting an image to grayscale

In [ ]:
```
import cv2

# Read an image
img = cv2.imread('Images/image.jpg')

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

# Display the grayscale image
cv2.imshow('gray', gray)

# Wait for a key event to close the window
cv2.waitKey(0)

# Destroy all windows
cv2.destroyAllWindows()
```

- Drawing shapes on an image

In [ ]:
```
import cv2

# Read an image
img = cv2.imread('Images/image.jpg')

start_x = 100
start_y = 100
```

```
end_x = 200
end_y = 200
center_x = 100
center_y = 100
radius = 50
thickness = 10

# Draw a line on the image
cv2.line(img, (start_x, start_y), (end_x, end_y), (255, 0, 0), thickness)

# Draw a rectangle on the image
cv2.rectangle(img, (x, y), (x+width, y+height), (0, 255, 0), thickness)

# Draw a circle on the image
cv2.circle(img, (center_x, center_y), radius, (0, 0, 255), thickness)

# Display the image with the shapes drawn on it
cv2.imshow('image', img)

# Wait for a key event to close the window
cv2.waitKey(0)

# Destroy all windows
cv2.destroyAllWindows()
```

- Image dimensions and color channels

In [ ]:

```
import cv2

# Read an image
img = cv2.imread('Images/image.jpg')

# Get the dimensions of the image
height, width, channels = img.shape
print('Image dimensions: {}x{} with {} channels'.format(width, height, channels))
```

```
Image dimensions: 872x586 with 3 channels
```

Image Transformation and Enhancement with OpenCV

OpenCV provides a number of functions for performing various image transformation and enhancement tasks. For example, we can rotate an image using the cv2.getRotationMatrix2D() and cv2.warpAffine() functions, apply perspective transforms using the cv2.getPerspectiveTransform() and cv2.warpPerspective() functions, and adjust the brightness and contrast of an image using the cv2.convertScaleAbs() function.

In [ ]:

```
import cv2
import numpy as np

# Load an image from file
img = cv2.imread('Images/image.jpg')

# Rotate the image
rows, cols, _ = img.shape
M = cv2.getRotationMatrix2D((cols/2, rows/2), 45, 1)
rotated_img = cv2.warpAffine(img, M, (cols, rows))

# Apply a perspective transform
pts1 = np.float32([[56,65],[368,52],[28,387],[389,390]])
pts2 = np.float32([[0,0],[300,0],[0,300],[300,300]])
M = cv2.getPerspectiveTransform(pts1,pts2)
transformed_img = cv2.warpPerspective(img,M,(300,300))

# Adjust the brightness and contrast
alpha = 1.5
beta = 50
adjusted_img = cv2.convertScaleAbs(img, alpha=alpha, beta=beta)

# Display the images on screen
cv2.imshow('Original Image', img)
cv2.imshow('Rotated Image', rotated_img)
cv2.imshow('Transformed Image', transformed_img)
cv2.imshow('Adjusted Image', adjusted_img)

# Wait for a key press
cv2.waitKey(0)

# Close the windows
cv2.destroyAllWindows()
```

Image Enhancement Techniques

- Histogram Equalization : Histogram equalization is a technique used to enhance the contrast of an image by redistributing the pixel intensities. In Python, we can use the cv2.equalizeHist() function to perform histogram equalization.

  Here's an example code snippet :

In [ ]:
```
import cv2

# Read image in grayscale
img = cv2.imread('Images/image.jpg', cv2.IMREAD_GRAYSCALE)

# Perform histogram equalization
equalized_img = cv2.equalizeHist(img)

# Display images
cv2.imshow('Original', img)
cv2.imshow('Equalized', equalized_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- Contrast Stretching : Contrast stretching is a technique used to enhance the contrast of an image by expanding the range of pixel intensities. In Python, we can use the cv2.normalize() function to perform contrast stretching.

  Here's an example code snippet :

In [ ]:
```
import cv2
import numpy as np

# Read image in grayscale
img = cv2.imread('Images/image.jpg', cv2.IMREAD_GRAYSCALE)

# Perform contrast stretching
normalized_img = cv2.normalize(img, None, 0, 255, cv2.NORM_MINMAX)

# Display images
cv2.imshow('Original', img)
cv2.imshow('Normalized', normalized_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Image Restoration Techniques

- In the context of image processing, noise refers to unwanted random variations or disturbances in the pixel values of an image. It is considered as an undesirable artifact that can degrade the quality and clarity of an image. Noise can arise from various sources such as sensor limitations, transmission errors, electromagnetic interference, or environmental factors.
- Noise Reduction : Noise reduction is a technique used to remove noise from an image. In Python, we can use various filters like Gaussian filter, median filter, etc. to perform noise reduction.

  Here's an example code snippet to perform noise reduction using a Gaussian filter:

In [ ]:
```
import cv2

# Read image
img = cv2.imread('Images/image.jpg')

# Perform Gaussian blur
blurred_img = cv2.GaussianBlur(img, (5, 5), 0)

# Display images
cv2.imshow('Original', img)
cv2.imshow('Blurred', blurred_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Image Segmentation Techniques

- Thresholding : Thresholding is a technique used to separate the foreground and background regions in an image. In Python, we can use the cv2.threshold() function to perform thresholding.

  Here's an example code snippet to perform thresholding :

In [ ]:
```
import cv2

# Read image in grayscale
```

```
img = cv2.imread('Images/image.jpg', cv2.IMREAD_GRAYSCALE)

# Perform thresholding
threshold_value = 127
max_value = 255
ret, thresholded_img = cv2.threshold(img, threshold_value, max_value, cv2.THRESH_BINARY)

# Display images
cv2.imshow('Original', img)
cv2.imshow('Thresholded', thresholded_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- Edge Detection : Edge detection is a technique used to detect the boundaries of objects in an image. In Python, we can use various edge detection algorithms like Canny edge detection, Sobel edge detection, etc. to perform edge detection.

  Here's an example code snippet to perform edge detection using Canny edge detection :

In [ ]:
```
import cv2

# Read image in grayscale
img = cv2.imread('Images/image.jpg', cv2.IMREAD_GRAYSCALE)

# Perform Canny edge detection
canny_img = cv2.Canny(img, 100, 200)

# Display images
cv2.imshow('Original', img)
cv2.imshow('Canny', canny_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Image Compression Techniques

JPEG Compression

JPEG compression is a lossy image compression technique that reduces the file size of an image by removing the high-frequency components of the image. In Python, we can use the cv2.imwrite() function with the appropriate JPEG compression quality parameter to save an image in JPEG format.

Here's an example code snippet to save an image in JPEG format:

In [ ]:
```
import cv2

# Read image
img = cv2.imread('Images/image.jpg')

# Save image in JPEG format
cv2.imwrite('Images/compressed_image.jpg', img, [cv2.IMWRITE_JPEG_QUALITY, 90])
```

Out[ ]:
```
True
```

PNG Compression

PNG compression is a lossless image compression technique that reduces the file size of an image without losing any information. In Python, we can use the cv2.imwrite() function with the appropriate PNG compression level parameter to save an image in PNG format.

Note: The compression level parameter ranges from 0 to 9, with 0 being no compression and 9 being maximum compression.

Here's an example code snippet to save an image in PNG format:

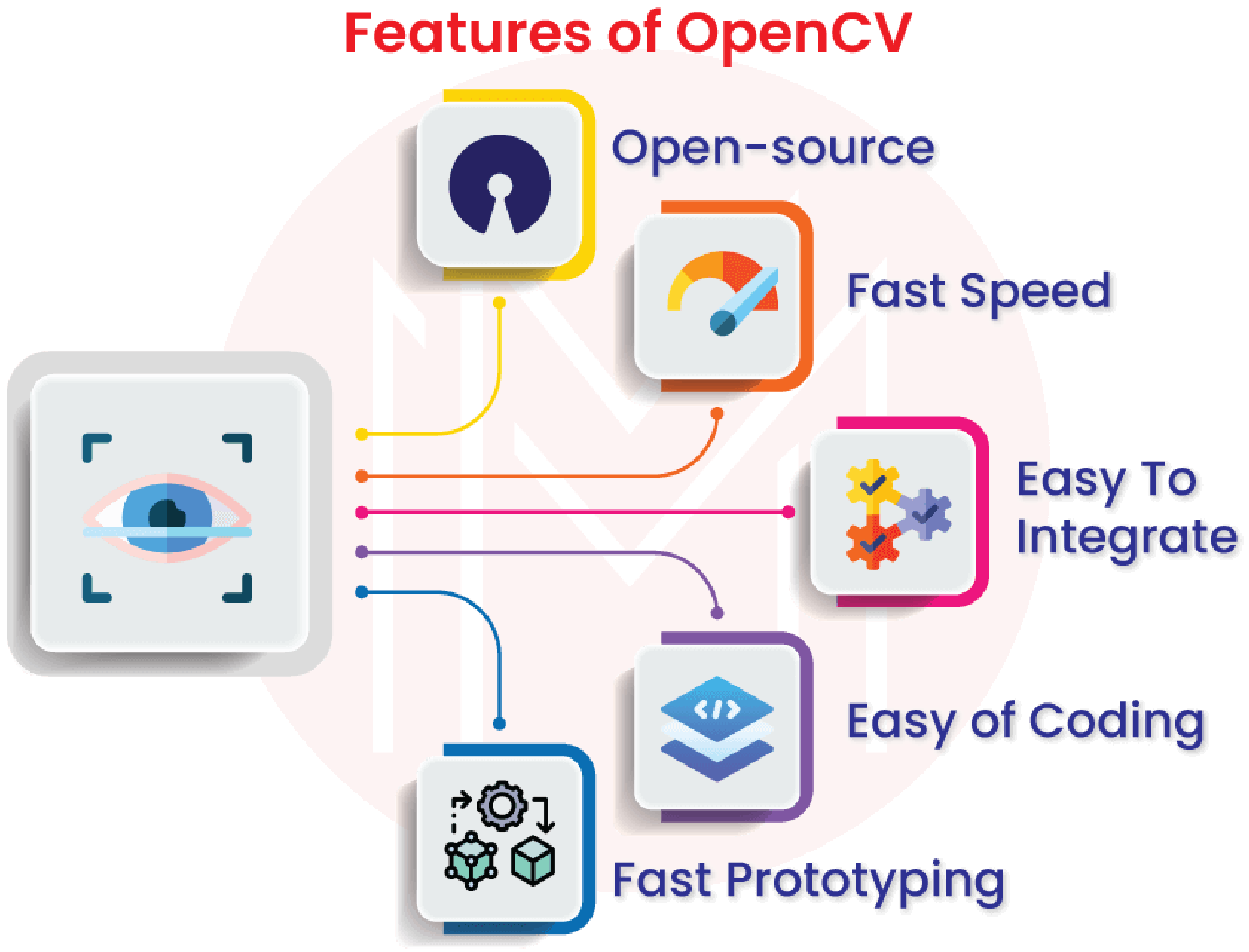
In [ ]:
```
import cv2

# Read image
img = cv2.imread('Images/image.jpg')

# Save image in PNG format
cv2.imwrite('Images/png_compressed_image.png', img, [cv2.IMWRITE_PNG_COMPRESSION, 9])
```

Out[ ]:
```
True
```

---

4. Open CV

OpenCV (Open Source Computer Vision) is a popular open-source library for computer vision and image processing. It provides a wide range of functions and algorithms that enable developers to perform various tasks related to image and video analysis.



Here are some real-world applications of OpenCV:

- Object Detection and Recognition: OpenCV can be used for detecting and recognizing objects in images and videos.
- Facial Recognition: OpenCV provides tools for detecting and recognizing faces in images and videos.
- Image and Video Processing: OpenCV offers a variety of image and video processing functions, such as filtering, enhancement, and transformation.
- Augmented Reality (AR): OpenCV can be employed in AR applications to track and overlay digital content on real-world scenes.

- Medical Image Analysis: OpenCV is utilized in medical imaging for tasks like image segmentation, tumor detection, and analysis of biological samples.
- Optical Character Recognition (OCR): OpenCV provides tools for extracting text from images and recognizing characters.
- Robotics: OpenCV is used in robotics for tasks such as object detection, navigation, and gesture recognition.
- Motion Tracking: OpenCV can track the movement of objects in videos and analyze their trajectories.
- Industrial Automation: OpenCV is used in industrial settings for tasks like quality control, defect detection, and object recognition.
- Driver Assistance Systems: OpenCV is employed in driver assistance systems to detect and track objects on the road, such as vehicles, pedestrians, and traffic signs.

Image Filtering with OpenCV

Image filtering is a technique that is used to enhance images, remove noise, and extract useful information from them. In Python, OpenCV is a popular library that is used for image filtering. OpenCV provides a wide range of image filtering functions that can be used to apply different types of filters to an image.

Here is a sample Python code that demonstrates image filtering using OpenCV:

In [ ]:
```python
import cv2
import numpy as np

# Load the image
img = cv2.imread('Images/image.jpg')

# Define the kernel for the filter
kernel = np.ones((5,5), np.float32)/25

# Apply the filter to the image
filtered_img = cv2.filter2D(img, -1, kernel)

# Display the original and filtered image
cv2.imshow('Original Image', img)
cv2.imshow('Filtered Image', filtered_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

In the above code, we first load the image using the cv2.imread() function. We then define the kernel for the filter. The kernel is a matrix that is used to define the filter. In this case, we have defined a 5x5 matrix with all values set to 1, and then divided the matrix by 25 to normalize it.

We then apply the filter to the image using the cv2.filter2D() function. This function applies the filter to the image using the kernel matrix. The -1 parameter specifies the depth of the image. If set to -1, the output image will have the same depth as the input image.

Finally, we display the original and filtered image using the cv2.imshow() function. The cv2.waitKey() function waits for a key event to occur, and the cv2.destroyAllWindows() function destroys all the windows that are created.

This is a basic example of image filtering using OpenCV. There are many other functions available in OpenCV that can be used to apply different types of filters to an image.

Image Blending and Compositing with OpenCV

Image blending and compositing is a common technique used in computer vision and image processing to combine multiple images into a single image. OpenCV is a popular Python library used for this purpose.

The following is a detailed Python code-based description for image blending and compositing with OpenCV:

In this code, we first load two images, image1.jpg and image2.jpg, using the cv2.imread() function. We then resize both images to the same size using the cv2.resize() function. Next, we create a mask for blending using the np.zeros_like() function to create an array of zeros with the same size as img1, and then set the region of interest to 200:400 for both the x and y axes, and assign it a value of 255.

We then apply the mask to both images using the cv2.bitwise_and() function to get the regions of interest from each image. We use the cv2.bitwise_not() function to invert the mask and apply it to img2.

We then blend the masked images using the cv2.addWeighted() function. The first two arguments are the two images we want to blend, followed by the blending weights for each image. In this case, we use equal weights for both images (0.5). The last argument is the gamma value, which we set to 0.

Finally, we display all the images using the cv2.imshow() function, and wait for a key press using cv2.waitKey(). We then destroy all the windows using cv2.destroyAllWindows().

In [ ]:
```
import cv2
import numpy as np
```

```python
# Load the images
img1 = cv2.imread('Images/image1.png')
img2 = cv2.imread('Images/image2.png')

# Resize the images to the same size
img1 = cv2.resize(img1, (640, 480))
img2 = cv2.resize(img2, (640, 480))

# Create a mask for blending
mask = np.zeros_like(img1)
mask[200:400, 200:400] = 255

# Apply the mask to the images
masked_img1 = cv2.bitwise_and(img1, mask)
masked_img2 = cv2.bitwise_and(img2, cv2.bitwise_not(mask))

# Blend the images using addWeighted function
blended_img = cv2.addWeighted(masked_img1, 0.5, masked_img2, 0.5, 0)

# Display the images
cv2.imshow('Image 1', img1)
cv2.imshow('Image 2', img2)
cv2.imshow('Mask', mask)
cv2.imshow('Blended Image', blended_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Color Spaces and Color Management with OpenCV

Color Spaces and Color Management are important concepts in computer vision and image processing. OpenCV provides various functions and utilities to work with different color spaces and manage colors in images. Here is a detailed Python code-based description of these topics.

Converting between Color Spaces

OpenCV provides functions to convert images between different color spaces. Some of the commonly used color spaces are BGR, RGB, HSV, and Gray.

To convert an image from one color space to another, we can use the cv2.cvtColor() function.

Here is an example of converting an image from BGR to Gray:

In this code, we first load an image in BGR format using the cv2.imread() function. We then use the cv2.cvtColor() function to convert the image from BGR to Gray using the cv2.COLOR_BGR2GRAY flag.

In [ ]:

```
import cv2

# Load the image in BGR format
image_bgr = cv2.imread('Images/image.jpg')

# Convert BGR to Gray
image_gray = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2GRAY)

# Display the images
cv2.imshow('BGR Image', image_bgr)
cv2.imshow('Gray Image', image_gray)
cv2.waitKey(0)
```

Out[ ]:

```
-1
```

Color Thresholding

Color thresholding is a technique used to separate regions of an image based on color. In OpenCV, we can use the cv2.inRange() function to perform color thresholding. This function takes two arguments: the input image and the color range to threshold.

Here is an example of using color thresholding to detect red regions in an image:

In this code, we first load an image and convert it to HSV color space using the cv2.cvtColor() function. We then define the lower and upper range of red color in HSV. We create two masks using the cv2.inRange() function for each range of red color. We then combine the masks using the bitwise OR operator. Finally, we apply the mask to the input image using the cv2.bitwise_and() function to obtain the result.

In [ ]:

```
import cv2
import numpy as np

# Load the image
image = cv2.imread('Images/image.jpg')

# Convert BGR to HSV
```

```
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

# Define the range of red color in HSV
lower_red = np.array([0,50,50])
upper_red = np.array([10,255,255])
mask1 = cv2.inRange(hsv, lower_red, upper_red)

lower_red = np.array([170,50,50])
upper_red = np.array([180,255,255])
mask2 = cv2.inRange(hsv, lower_red, upper_red)

# Combine the masks
mask = mask1 + mask2

# Apply the mask to the image
result = cv2.bitwise_and(image, image, mask=mask)

# Display the images
cv2.imshow('Original Image', image)
cv2.imshow('Mask', mask)
cv2.imshow('Result', result)
cv2.waitKey(0)
```

Out[ ]:

```
-1
```

Color Correction

Color correction is a technique used to adjust the colors in an image to improve its overall appearance. OpenCV provides various functions to perform color correction on images. In this section, we will discuss how to use OpenCV to perform color correction on an image.

First, let's import the necessary libraries and load an image:

In [ ]:

```
import cv2
import numpy as np

# Load the image
img = cv2.imread('Images/image.jpg')
```

Now, let's convert the color space of the image to LAB color space. The LAB color space is a color-opponent space with dimension L for lightness and a and b for the color-opponent dimensions. This color space is used for color correction because it separates the luminance (brightness) and color information in an image.

In [ ]:
```
# Convert the image to LAB color space
lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)

# Split the LAB color space into its 3 channels: L, a, and b
l, a, b = cv2.split(lab)
```

Next, let's normalize the L channel of the LAB color space to improve the contrast of the image. We will use the CLAHE (Contrast Limited Adaptive Histogram Equalization) algorithm to do this.

In [ ]:
```
# Apply CLAHE to the L channel of the LAB color space
clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8,8))
l_clahe = clahe.apply(l)
```

Now, let's merge the L, a, and b channels back into a single image and convert it back to the original color space.

In [ ]:
```
# Merge the L, a, and b channels back into a single image
lab_clahe = cv2.merge((l_clahe, a, b))

# Convert the image back to the original color space
color_corrected_img = cv2.cvtColor(lab_clahe, cv2.COLOR_LAB2BGR)
```

Finally, let's display the original image and the color-corrected image side by side for comparison.

In [ ]:
```
# Display the original image and the color-corrected image side by side
cv2.imshow('Original Image', img)
cv2.imshow('Color-Corrected Image', color_corrected_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

This code will display two windows showing the original image and the color-corrected image side by side. The color-corrected image will have improved contrast and overall appearance.

Text and Font Handling with OpenCV

Text and font handling with OpenCV involves adding text to an image or video stream using various fonts and styles. The following Python code provides an example of how to do this:

In [ ]:
```python
import numpy as np
import cv2

# Load image
img = cv2.imread('Images/image.jpg')

# Define font and text to add
font = cv2.FONT_HERSHEY_SIMPLEX
text = 'Hello, World!'

# Get text size
text_size, _ = cv2.getTextSize(text, font, 1, 2)

# Calculate text position
text_x = int((img.shape[1] - text_size[0]) / 2)
text_y = int((img.shape[0] + text_size[1]) / 2)

# Add text to image
cv2.putText(img, text, (text_x, text_y), font, 1, (255, 255, 255), 2, cv2.LINE_AA)

# Display image
cv2.imshow('Image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

In this example, we first load an image using cv2.imread(). We then define the font and text that we want to add. Next, we use cv2.getTextSize() to get the size of the text and calculate the position of the text in the image. Finally, we use cv2.putText() to add the text to the image and display the image using cv2.imshow().