Image Processing in Open-cv.

Practical – W7

Reading Images:

Digital images encompass various categories, including:

- Color Images: These images store color information for each pixel.
- Grayscale Images: These images are characterized by shades of gray as their sole colors.
- Binary Images: Binary images consist of exactly two colors, often black and white pixels.
- Multispectral Images: These images capture data across the electromagnetic spectrum within specific wavelength ranges.

```
pip install opencv-python    #To install the library
```

```
import cv2    #To import the library
```

Let's resume our coding journey by focusing on the task of reading an image. For instance, consider the image displayed below:

Reading an Image:

- To read an image using OpenCV (cv2), you can use the following method. **cv2.imread(path_to_image_with_file_extension, flag)**
- For instance, you can load an image using:
- **img = cv2.imread("mandrill.jpg", 1)**.
- The first parameter is the path to the image file, and the second parameter is a flag that determines how the image is read. Use 0 for grayscale, -1 to read as unchanged (including alpha channel if present), and 1 for color images.
- The image file should be in your current working directory.

```
img = cv2.imread('watch.jpg',1)
```

Examining Image Properties:

- Every image has properties like shape, type, data type, and pixel values.
- The shape of an image, which specifies its dimensions (height, width, and channels), canbe obtained using **img.shape**.

```
print(img.shape)
h, w, c = img.shape
print("Dimensions of the image is:\nHeight:", h, "pixel’s Width:", w,
"pixel’s Number of Channels:", c)
```

- The type of the image can be identified with **type(img)** as it represents a multidimensional container of items.

```
print(type(img))
```

- The data type of the image can be checked using **img.dtype**.

```
print(img.dtype)
```

Image Pixel Values:

- Images are made up of pixels, which are small samples.
- Pixels can be considered as elements in a matrix.
- The intensity or brightness of each pixel is represented by values ranging from 0 to 255.

```
print(img)
```

Viewing the Image:

- To display an image, use **cv2_imshow(img)**.

```
cv2_imshow(img)
```

Resize image

img = cv2.imread('watch.jpg',1):

- This line reads an image file named 'watch.jpg' located at the specified file path. The '1' argument indicates that the image should be loaded as a colour image (BGR format). If you use '0' instead of '1', it would load the image as grayscale.

resized_image = cv2.resize(img, (int(img.shape[1]/2), int(img.shape[0]/2)):

- This line resizes the loaded image.
- **img** is the original image.
- The **(int(img.shape[1]/2), int(img.shape[0]/2))** argument specifies the new dimensions for the image. In this case, it reduces both the width and height of the image to half their original values.

cv2_imshow(resized_image):

- This line displays the resized image using the cv2_imshow function provided by Google Colab/Jupyter. This function is used to show images directly in notebooks.

```
img = cv2.imread ('watch.jpg',1)

resized_image =cv2.resize(img,int(img.shape[1]/2),int(img.shape[0]/2)))

cv2_imshow(resized_image)
```

Greyscale Image:

You can display the greyscale image in two ways.

Method one: the code reads the 'watch.jpg' image as a colour image (BGR format) using cv2.imread. The '1' argument indicates that it should be loaded as a colour image. The next line converts the previously loaded colour image **img** into a grayscale image **img_gray** using cv2.cvtColor. It applies the transformation from the BGR colour space to grayscale, creating a single-channel image.

```
# Method one: read color image and convert to greyscale
img = cv2.imread ('watch.jpg',1)
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2_imshow(img_gray)
print("Greyscale image: ")
```

Method Two: Read the image as greyscale image by passing argument 0, instead of 1.

```
#Methode two: read greyscale image
img = cv2.imread ('watch.jpg',0)
cv2_imshow(img)
```

Display RGB Channels of Image

1. Load the image using cv2.imread.
2. Split the loaded image into its Red, Green, and Blue channels using cv2.split. This separates the colour channels into three separate matrices.
3. Create all-zero matrices (zeros) of the same size as the input image.
4. Merge each channel separately into a colour image using cv2.merge. You create three images: one for the Blue channel, one for the Green channel, and one for the Red channel. These images are effectively grayscale representations of their respective colour channels.
5. Display each channel separately using cv2_imshow.

```
# Load the image
image = cv2.imread('burano.jpg')

# Split the image into its channels (BGR order)
blue_channel, green_channel, red_channel = cv2.split(image)

# Create an all-zero channel (black) of the same size as the input
image
zeros = np.zeros_like(blue_channel)

# Merge each channel separately into a color image
blue_image = cv2.merge([blue_channel, zeros, zeros])
green_image = cv2.merge([zeros, green_channel, zeros])
red_image = cv2.merge([zeros, zeros, red_channel])

# Display each channel separately
cv2_imshow( blue_image)
cv2_imshow(green_image)
cv2_imshow(red_image)
```

Task 1: Read and display the image “Eva.jpg”.

Task 2: Convert it Eva’s image to greyscale and display it.

Save an image:

Save the grayscale image of Eva to your local working directory using.

```
cv2.imwrite('Eva_Grey.png', image_grey)
```

Drawing Functions in Open-CV

Draw a Line:

- Read the Eva.jpg image as colour image.
- img = cv2.line(img, (0, 0), (300, 511), (255, 0, 0), 5): This line draws a blue diagonal line on the image. Here's what each argument means:
- img: This is the image on which you want to draw the line.
- (0, 0): This is the starting point of the line, represented by a tuple of (x, y) coordinates. In this case, it starts at the top-left corner (0, 0).

- (300, 511): This is the ending point of the line, also represented by (x, y) coordinates. It ends at the coordinates (300, 511), which is closer to the bottom-right corner of theimage.
- (255, 0, 0): This tuple specifies the colour of the line. In BGR colour format, (255, 0, 0) corresponds to blue.
- 5: This is the thickness of the line in pixels. The line will be 5 pixels wide.
- cv2_imshow(img): Finally, the modified image (img) is displayed using cv2_imshow. This function is used to display the image within a Jupyter Notebook.

```
img = cv2.imread('Eva.jpg',1)
# Draw a diagonal blue line with thickness of 5 px
img = cv2.line(img,(0,0),(300,511),(255,0,0),5)
# display image
cv2_imshow(img)
```

Draw a Rectangle

The following code reads an image, draws a green rectangle that starts from the point (50, 30) and ends at (250, 230), and then displays the modified image. The rectangle has a green colour and a border with a thickness of 3 pixels. Here is the explanation of the code.

- Read image as colour image.
- img = cv2.rectangle(img, (50, 30), (250, 230), (0, 255, 0), 3): This line draws a green rectangle on the image. Here's what each argument means:
- img: This is the image on which you want to draw the rectangle.
- (50, 30): This is the top-left corner of the rectangle, represented by a tuple of (x, y) coordinates. In this case, it starts at the point (50, 30).
- (250, 230): This is the bottom-right corner of the rectangle, also represented by (x, y) coordinates. The rectangle ends at the coordinates (250, 230).
- (0, 255, 0): This tuple specifies the colour of the rectangle. In BGR colour format, (0, 255, 0) corresponds to green.
- 3: This is the thickness of the rectangle's border in pixels. The border of the rectangle will be 3 pixels wide. cv2_imshow(img): Finally, the modified image (img) is displayed using cv2_imshow. This function is used to display the image within a Jupyter Notebook.

```
img = cv2.imread('Eva.jpg',1)
# Draw a diagonal green line with thickness of 5 px
img = cv2.rectangle(img,(50,30),(250,230),(0,255,0),3)
# display image
cv2_imshow(img)
```

Draw a Circle

The following code loads an image, draws a red circle with a centre at (150, 200) and a radius of 150 pixels, and then displays the modified image. The circle is red and has a border with a thickness of 5 pixels.

- Load the image as colour image.
- img = cv2.circle(img, (150, 200), 150, (0, 0, 255), 5): This line draws a red circle on the image. Here's what each argument represents:
- img: This is the image on which you want to draw the circle.
- (150, 200): This tuple represents the centre of the circle, specified as (x, y) coordinates. In this case, the centre is at point (150, 200).
- 150: This is the radius of the circle in pixels. The circle's radius is set to 150 pixels.

- (0, 0, 255): This tuple specifies the colour of the circle. In BGR colour format, (0, 0, 255) corresponds to red, with zero blue and green components and a maximum value of 255 in the red component.
- 5: This is the thickness of the circle's border in pixels. The border of the circle will be 5 pixels wide.
- cv2_imshow(img): Finally, the modified image (img) is displayed using cv2_imshow. This function is used to display the image within a Jupyter Notebook.

```
img = cv2.imread('Eva.jpg',1)
# Draw a diagonal red line with thickness of 5 px
img = cv2.circle(img,(150,200), 150, (0,0,255), 5)
# display image
cv2_imshow(img)
```

Task 3: Using your mobile phone, take a selfie of yourself. Upload your selfie to the drive.Read it as colour image in your Jupyter notebook. Find out the shape of your image.

Task 4: Display your selfie as greyscale.

Task 5: Draw circles or same radius around your eyes.

Task 6: Draw a rectangle around your face.

Task 7: Save the edited image.

Region of Interest (ROI) or Image Cropping

Following is a dummy code to crop a region of interest (ROI) from an image. Here is the explanation of the code.

- Import the OpenCV library with import cv2.
- Load the input image using cv2.imread("input_image.jpg"). Make sure to replace "input_image.jpg" with the path to your image.
- Define the coordinates of the region you want to crop using the top_left and bottom_right points. These points represent the top-left and bottom-right corners of the rectangular region you want to crop.
- Crop the image using NumPy array slicing. image[top_left[1]:bottom_right[1], top_left[0]:bottom_right[0]] selects the region defined by the top_left and bottom_right points.
- Save the cropped image to a new file using cv2.imwrite().
- Optionally, display the cropped image using cv2_imshow(). This step is not necessary but can be useful for visual inspection. Make sure to close the displayed image window by pressing any key after viewing it.
- Make sure to replace "input_image.jpg" with the path to your input image and adjust the coordinates (top left and bottom right) to specify the desired region you want to crop from the image. The cropped image will be saved as "cropped_image.jpg" in the current working directory.

```python
import cv2

# Load the image
image = cv2.imread("input_image.jpg")

# Define the coordinates of the region you want to crop (top left andbottom
right)
top_left = (100, 100)
bottom_right = (400, 300)

# Crop the image
cropped_image =image[top_left[1]:bottom_right[1],top_left[0]:bottom_right[0]]

# Save the cropped image to a file
cv2.imwrite("cropped_image.jpg", cropped_image)

# Display the cropped image (optional)
cv2.imshow("Cropped Image", cropped_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Write your code to:

Task 8: Crop the eyes, nose, and Jewelry from the Eva.jpg image.

Task 9: Resize all the cropped images to the same size of 20*20.