Department of Computer Science and Engineering (Data Science)
Academic Year 2022-2023

Name : Sarvagya Singh

SAPID : 60009200030 BATCH : K1

Experiment - 1

AIM: To Perform Basic Image Processing Operations in Python

PROBLEM STATEMENT:

Read an Image

Display an Image

Convert in Grey Scale

Crop an Image

Arithmetic Operations

Logical Operations

THEORY:

Discuss about Different Image Processing Libraries available in Python.

Features & Limitations of each.

Libraries used:

OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as Numpy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e whatever operations one can do in Numpy can be combined with OpenCV.

Features:

• OpenCV is an open-source library, so it is free to use and can be easily customized to meet specific needs.

• OpenCV provides a comprehensive set of algorithms for image processing and computer vision, including object detection, recognition, tracking, and segmentation.

• Python is a popular programming language for scientific computing, and OpenCV's Python interface makes it easy to integrate computer vision algorithms with other Python libraries like NumPy, SciPy, and Matplotlib.

• OpenCV's Python bindings support both Python 2 and Python 3, making it accessible to a wide range of users.

• OpenCV is cross-platform, so code written in Python can be easily ported to other platforms like Linux, Windows, and macOS.

• OpenCV has a large and active community, so there are plenty of resources available for learning and troubleshooting, including documentation, tutorials, and forums.

Limitations:

• Speed: OpenCV in Python can be slower compared to other languages like C++ because of the GIL (Global Interpreter Lock) in Python. This can limit the performance of real-time applications.

• Memory management: Memory management in Python can be problematic, especially when working with large datasets. This can lead to slower processing times and memory leaks.

• Limited machine learning capabilities: Although OpenCV has some machine learning capabilities, it may not be as robust as other popular machine learning libraries like TensorFlow and PyTorch.

• Limited support for deep learning: OpenCV has limited support for deep learning, which may be a disadvantage for complex image processing tasks.

• Limited support for GPU processing: OpenCV in Python may not provide support for GPU processing, which may limit the speed of certain applications.

• Limited documentation: OpenCV documentation in Python may not be as extensive as in other languages, which may make it challenging for beginners to learn and use the library effectively.

RESULT:

Sarvagya Singh

600009200030 -- K1

IPCV -- lab1

In [ ]:
```
import cv2 as cv
import PIL
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
```

In [ ]:
```
%%capture
!wget https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTG8Ogg1R2WkRRVnsxMlr_WqdNwhVzo9vvW_42SZm8&s
```

In [ ]:
```
# %%capture
# !wget
```

In [ ]:
```
# Displaying the image
path1 = "/content/image.jpg"
image_np = cv.imread(path1)
print(image_np.shape)
image_np[1][:5]
```

```
(122, 123, 3)
```

Out[ ]:
```
array([[136, 126, 139],
       [142, 132, 148],
       [116, 107, 127],
       [ 54,  49,  70],
       [ 83,  80, 105]], dtype=uint8)
```

In [ ]:
```
path2 = "/content/banana.jpg"
im_np2 = cv.imread(path)
im_np2.shape,image_np.shape
cv2_imshow(im_np2)
```

In [ ]:
```
# Displaying the image
im_rgb = cv.cvtColor(image_np, cv.COLOR_BGR2RGB)
plt.axis('off')
plt.imshow(im_rgb);
```

In [ ]:
```
cv2_imshow(image_np)
```

In [ ]:
```
im_gray1 = cv.cvtColor(im_rgb, cv.COLOR_BGR2GRAY)
im_gray2 = cv.cvtColor(im_np2, cv.COLOR_BGR2GRAY)
plt.axis('off')
plt.imshow(im_gray1);
```

In [ ]:
```
cv2_imshow(im_gray1)
```

In [ ]:
```
plt.axis('off')
plt.imshow(im_gray2);
```

In [ ]:
```
cv2_imshow(im_gray2)
```

In [ ]:
```
from PIL import Image
im1 = Image.open(path1)
# im1.show()
# cv2_imshow(im1)
print(im1.filename)
print(im1.size)
print(im1.info)
print(im1.palette)
```

```
/content/image.jpg
(123, 122)
{'jfif': 257, 'jfif_version': (1, 1), 'jfif_unit': 0, 'jfif_density': (1, 1)}
```

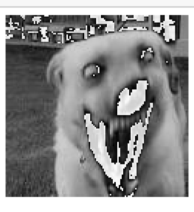
In [ ]:
```
#  Cropping an image
cropped_image = im_gray1[10:110, 20:110]
cv2_imshow(cropped_image)
```

In [ ]:
```
cropped_image = im_gray2[10:110, 20:110]
cv2_imshow(cropped_image)
```


In [ ]:
```
# Arithmetic operation
im_1 = im_gray1*2
cv2_imshow(im_1)
```

In [ ]:
```
im_2 = im_gray1+502
cv2_imshow(im_2)
```

In [ ]:
```
im_2 = im_gray1-50
cv2_imshow(im_2)
```



In [ ]:
```
im_2 = im_gray1+50
cv2_imshow(im_2)
```

In [ ]:
```
im_2 = im_gray1/2
cv2_imshow(im_2)
```

In [ ]:
```
resized_im1 = cv.resize(im_gray1,(122,122))
resized_im2 = cv.resize(im_gray2,(122,122))
resized_im2.shape,resized_im1.shape
```

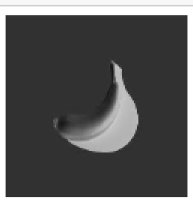
Out[ ]:
```
((122, 122), (122, 122))
```

In [ ]:
```
bitwise_and_np = cv.bitwise_and(resized_im1, resized_im2)
cv2_imshow(bitwise_and_np)
```

In [ ]:
```
bitwise_xor_np = cv.bitwise_xor(resized_im1, resized_im2)
cv2_imshow(bitwise_xor_np)
```

In [ ]:
```
bitwise_not1 = cv.bitwise_not(resized_im1)
cv2_imshow(bitwise_not1)
```

In [ ]:
```
bitwise_not2 = cv.bitwise_not(resized_im2)
cv2_imshow(bitwise_not2)
```



In [ ]: