डिजिटल इमेज प्रोसेसिंग (डीआईपी) प्रयोगशाला

Digital Image Processing (DIP) Lab

(EC 324)

(बी टेक **VI** सेमेस्टर / **BTech-VI Semester**)

(प्रयोगशाला मैनुअल)

Lab Manual

(2024-25)

इलेक्ट्रॉनिक्स एवं संचार अभियांत्रिकी विभाग

Department of Electronics and Communication Engineering

मौलाना आजाद राष्ट्रीय प्रौद्योगिकी संस्थान, भोपाल- 462003

Maulana Azad National Institute of Technology Bhopal-462003

डिजिटल इमेज प्रोसेसिंग (डीआईपी) प्रयोगशाला

Digital Image Processing (DIP) Lab

(प्रयोगशाला मैनुअल)

Lab Manual

कार्यक्रम	:	प्रौद्योगिकी में स्नातक
Program	:	**Bachelor of Technology**
विशेषज्ञता	:	इलेक्ट्रॉनिक्स और संचार अभियांत्रिकी
Specialization	:	**Electronics and Communication Engineering**
सेमेस्टर	:	**VI**
Semester	:	**VI**
पाठ्यक्रम कोड	:	**EC 324**
Course Code	:	**ईसी 324**

डॉ. विजयश्री चौरसिया (प्रयोगशाला समन्वयक) द्वारा तैयार

Prepared by Dr. Vijayshri Chaurasia (Laboratory Coordinator)

मौलाना आजाद राष्ट्रीय प्रौद्योगिकी संस्थान, भोपाल- 462003

Maulana Azad National Institute of Technology Bhopal-462003

इलेक्ट्रॉनिक्स एवं संचार अभियांत्रिकी विभाग

Department of Electronics and Communication Engineering

डिजिटल इमेज प्रोसेसिंगप्रयोगशाला (ईसी-324)

Digital image processing Lab (EC-324)

प्रयोगों की सूची

List of Experiments

क्रमांक S.N.	**प्रयोग Experiment**	**पृष्ठ संख्या Page No.**
1	छवियों को पढ़ने और प्रदर्शित करने के लिए: (A) स्थानिक रिज़ॉल्यूशन (बाइनरी, ग्रेस्केल और रंग) की पहचान करना और उसे बदलना। (B) तीव्रता रिज़ॉल्यूशन (बाइनरी, ग्रेस्केल और रंग) की पहचान करना और उसे बदलना। (C) छवियों के विभिन्न चैनलों की पहचान करना। (D) विभिन्न चैनलों (R, G, और B) में छवि को विघटित करना To read and display the images: (A)To identify and change spatial resolution (Binary, Grayscale and Color). (B)To identify and its change intensity resolution (Binary, Grayscale and Color). (C)To identify different channels of the images. (D)To decompose image in different channels (R, G, and B)	**06**
2	छवियों के अंकगणितीय संचालन का अध्ययन और निष्पादन करना: (A) जोड़ संचालन (B) घटाव संचालन (C) गुणा संचालन (D)विभाजन संचालन (E) रंगीन छवियों के विभिन्न चैनलों (R-G, B-R, B-G) में अंकगणितीय संचालन करना To study and perform arithmetic operations of images: (A) Addition operation (B) Subtraction operation (C) Multiplication operation	**12**

	(D) Division operation (E) To perform the arithmetic operations in different channels (R-G, B-R, B-G) of color images.	
3	छवियों के तार्किक संचालन का अध्ययन और निष्पादन करने के लिए: (A) NOT संचालन (B) AND/NAND संचालन (C) OR/NOR संचालन (D) XOR/XNOR संचालन To study and perform logical operations of images: (A) NOT operation (B) AND/NAND operation (C) OR/NOR operation (D) XOR/XNOR operation	**18**
4	छवियों पर ज्यामितीय संचालन का अध्ययन और निष्पादन करने के लिए: (A) छवियों का आकार बदलें (छोटा करें/ज़ूम करें) (B) रोटेशन ऑपरेशन (C)ट्रांसलेशन ऑपरेशन (D) शीयरिंग ऑपरेशन To study and perform geometric operations on images: (A) Resize (Shrink/Zoom) Images (B) Rotation operation (C) Translation operation (D) Shearing operation	**23**
5	छवियों पर रैखिक पड़ोस संचालन का अध्ययन और प्रदर्शन करने के लिए: (ए) विभिन्न संवलन कर्नेल के साथ। (बी) विभिन्न आकार के संवलन कर्नेल के साथ। (सी) विभिन्न पैडिंग संचालन (शून्य, 1, रेखा, दर्पण) के साथ To study and perform linear neighbourhood operations on images: (A) With different convolution kernels. (B) With convolution kernels of different size. (C) With various padding operations (zero, 1's, line, mirror)	**32**
6	छवियों पर गैर-रैखिक पड़ोस संचालन का अध्ययन और प्रदर्शन करने के लिए: (A) न्यूनतम (B) अधिकतम (C) माध्यिका (D) विभिन्न पैडिंग संचालन (शून्य, 1, रेखा, दर्पण) के साथ To study and perform non-linear neighbourhood operations on images: (A) Min (B) Max (C) Median	**44**

	(D) With various padding operations (zero, 1's, line, mirror)	
7	स्थानिक डोमेन छवि को शोरमुक्त करने के लिए अध्ययन करना और उसका निष्पादन करना: (A) विभिन्न घनत्वों के आवेग और गॉसियन शोर के लिए रैखिक फ़िल्टर। (B) विभिन्न घनत्वों के आवेग और गॉसियन शोर के लिए गैर-रैखिक फ़िल्टर। To study and perform spatial domain image denoising using: (A) Linear filters for Impulse and Gaussian noise of various densities. (B) Non-linear filters for Impulse and Gaussian noise of various densities.	**58**
8	विभिन्न थ्रेशोल्ड मानों का उपयोग करके इमेज बाइनरीकरण और मास्क निर्माण का अध्ययन और प्रदर्शन करना। मास्क का आगे का अनुप्रयोग: (A) गुणन ऑपरेशन (B) जोड़ ऑपरेशन (C) और ऑपरेशन (D) या ऑपरेशन To study and perform Image Binarization and mask formation using different threshold values. Further application of mask using: (A) Multiplication operation (B) Addition operation (C) AND operation (D) OR operation	**67**
9	छवियों पर रूपात्मक संचालन का अध्ययन और निष्पादन करना: (A) क्षरण (B) फैलाव (C) खोलना (D) बंद करना To study and perform morphological operations on images: (A) Erosion (B) Dilation (C) Opening (D) Closing	**72**
10	निम्नलिखित तकनीकों का उपयोग करके छवि पर एज डिटेक्शन करें: (A) सोबेल ऑपरेटर (B) लाप्लासियन ऑपरेटर (C) कैनी एज डिटेक्शन Perform Edge Detection on an image using techniques like: (A) Sobel Operator (B) Laplacian Operator (C) Canny Edge Detection	**81**

Digital Image Processing Lab
ECE-324
Experiment No-1
डिजिटल इमेज प्रोसेसिंग लैब
प्रयोग संख्या-1

Aim:

To read and display the images:

i. To identify and change spatial resolution (Binary, Grayscale and Color).
ii. To identify and its change intensity resolution (Binary, Grayscale and Color).
iii. To identify different channels of the images.
iv. To decompose image in different channels (R, G, and B).

उद्देश्य:

छवियों को पढ़ना और प्रदर्शित करना:

i. स्थानिक रिज़ॉल्यूशन (बाइनरी, ग्रेस्केल और रंग) की पहचान करना और उसे बदलना।

ii. तीव्रता रिज़ॉल्यूशन (बाइनरी, ग्रेस्केल और रंग) की पहचान करना और उसे बदलना।

iii. छवियों के विभिन्न चैनलों की पहचान करना।

iv. छवि को विभिन्न चैनलों (आर, जी, और बी) में विघटित करना।

Theory:

An image is defined as a two-dimensional function, **F(x,y)**, where x and y are spatial coordinates, and the amplitude of **F** at any pair of coordinates (x,y) is called the **intensity** of that image at that point. When x,y, and amplitude values of **F** are finite, we call it a **digital image**.
In other words, an image can be defined by a two-dimensional array specifically arranged in rows and columns.
Digital Image is composed of a finite number of elements, each of which elements have a particular value at a particular location. These elements are referred to as picture elements, image elements, and pixels. A Pixel is most widely used to denote the elements of a Digital Image.

सिद्धांत:

एक छवि को दो-आयामी फ़ंक्शन, F(x,y) के रूप में परिभाषित किया जाता है, जहाँ x और y स्थानिक निर्देशांक हैं, और निर्देशांक (x,y) के किसी भी जोड़े पर F के आयाम को उस बिंदु पर उस छवि की तीव्रता कहा जाता है। जब x, y और F के आयाम मान परिमित होते हैं, तो हम इसे डिजिटल छवि कहते हैं।

दूसरे शब्दों में, एक छवि को पंक्तियों और स्तंभों में विशेष रूप से व्यवस्थित दो-आयामी सरणी द्वारा परिभाषित किया जा सकता है।

डिजिटल छवि तत्वों की एक सीमित संख्या से बनी होती है, जिनमें से प्रत्येक तत्व का एक विशेष स्थान पर एक विशेष मान होता है। इन तत्वों को चित्र तत्व, छवि तत्व और पिक्सेल कहा जाता है। डिजिटल छवि के तत्वों को दर्शाने के लिए पिक्सेल का सबसे अधिक उपयोग किया जाता है।

Types of an image

1. **BINARY IMAGE**– The binary image as its name suggests, contain only two-pixel elements i.e 0 & 1, where 0 refers to black and 1 refers to white. This image is also known as Monochrome.
2. **BLACK AND WHITE IMAGE**– The image which consist of only black and white color is called BLACK AND WHITE IMAGE.
3. **8 bit COLOR FORMAT**– It is the most famous image format.It has 256 different shades of colors in it and commonly known as Grayscale Image. In this format, 0 stands for Black, and 255 stands for white, and 127 stands for gray.
4. **16 bit COLOR FORMAT**– It is a color image format. It has 65,536 different colors in it.It is also known as High Color Format. In this format the distribution of color is not as same as Grayscale image.

A 16 bit format is actually divided into three further formats which are Red, Green and Blue. That famous RGB format.

इमेज के प्रकार

1. बाइनरी इमेज- बाइनरी इमेज जैसा कि इसके नाम से पता चलता है, इसमें केवल दो पिक्सेल तत्व होते हैं यानी 0 और 1, जहाँ 0 काले रंग को दर्शाता है और 1 सफेद रंग को दर्शाता है। इस इमेज को मोनोक्रोम के नाम से भी जाना जाता है।

2. ब्लैक एंड व्हाइट इमेज- जिस इमेज में केवल काला और सफेद रंग होता है उसे ब्लैक एंड व्हाइट इमेज कहते हैं।

3. 8 बिट कलर फॉर्मेट- यह सबसे प्रसिद्ध इमेज फॉर्मेट है। इसमें 256 अलग-अलग शेड के रंग होते हैं और इसे आमतौर पर ग्रेस्केल इमेज के नाम से जाना जाता है। इस फॉर्मेट में, 0 का मतलब ब्लैक, 255 का मतलब व्हाइट और 127 का मतलब ग्रे होता है।

4. 16 बिट कलर फॉर्मेट- यह एक कलर इमेज फॉर्मेट है। इसमें 65,536 अलग-अलग रंग होते हैं। इसे हाई कलर फॉर्मेट के नाम से भी जाना जाता है। इस फॉर्मेट में रंगों का वितरण ग्रेस्केल इमेज के समान नहीं होता है।

16 बिट प्रारूप वास्तव में तीन और प्रारूपों में विभाजित है जो लाल, हरा और नीला हैं। वह प्रसिद्ध आरजीबी प्रारूप है।

Image as a Matrix

As we know, images are represented in rows and columns we have the following syntax in which images are represented:

जैसा कि हम जानते हैं, छवियों को पंक्तियों और स्तंभों में दर्शाया जाता है, हमारे पास निम्नलिखित सिंटैक्स है जिसमें छवियों को दर्शाया जाता है:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & f(0,2) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & f(1,2) & \dots & f(1,N-1) \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ f(M-1,0) & f(M-1,1) & f(M-1,2) & \dots & f(M-1,N-1) \end{bmatrix}$$

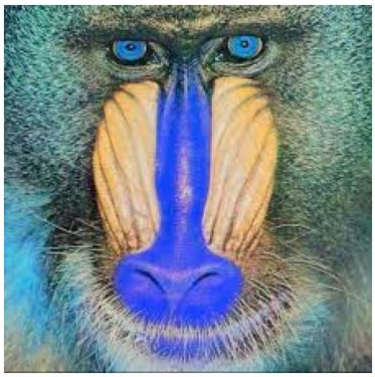
The right side of this equation is digital image by definition. Every element of this matrix is called image element, picture element, or pixel.

इस समीकरण का दाहिना भाग परिभाषा के अनुसार डिजिटल छवि है। इस मैट्रिक्स के प्रत्येक तत्व को छवि तत्व, चित्र तत्व या पिक्सेल कहा जाता है।

Code

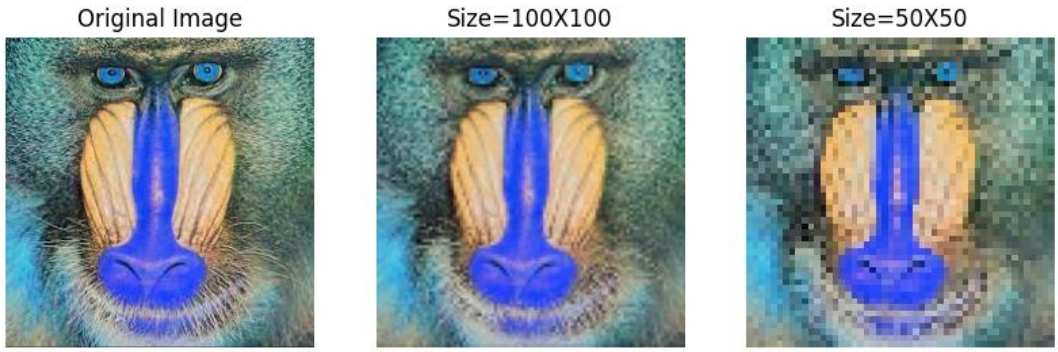
```
#!pip install opencv-python#
import cv2

install opencv first

img1 = cv2.imread("/content/Baboon.jpg")
from google.colab.patches import cv2_imshow
cv2_imshow(img1)
```

```
import matplotlib.pyplot as plt
plt.imshow(img1)
plt.axis("off")
```

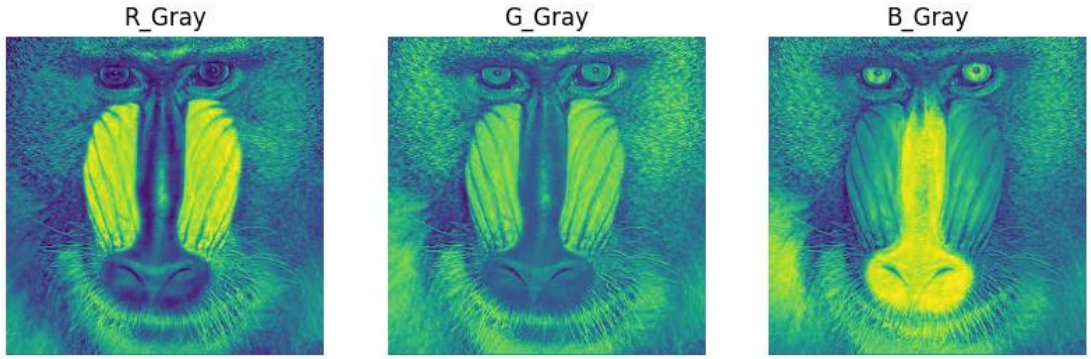


```
img1.shape
H = 225
W = 225

## resizing of image ###
img4 = cv2.resize (img1, (100, 100))
img5 = cv2.resize(img1, (50,50))

fig = plt.figure(figsize=(10, 7))
fig.add_subplot(1, 3, 1)
plt.imshow(img1)
plt.axis("off")
plt.title("Original Image")


fig.add_subplot(1, 3, 2)
plt.imshow(img4)
plt.axis("off")
plt.title("Size=100X100")

fig.add_subplot(1, 3, 3)
plt.imshow(img5)
```

```
plt.axis("off")
plt.title("Size=50X50")
```




```
img6 = img1[:, :, 0]
img7 = img1[:, :, 1]
img8 = img1[:, :, 2]

fig = plt.figure(figsize=(10, 7))
fig.add_subplot (1, 3, 1)
plt.axis("off")
plt.imshow(img6)
plt.title("R_Gray")

fig.add_subplot (1, 3, 2)
plt.axis("off")
plt.imshow(img7)
plt.title("G_Gray")

fig.add_subplot (1, 3, 3)
plt.axis("off")
plt.imshow(img8)
plt.title("B_Gray")
```




```
fig=plt.figure(figsize=(10, 7))
fig.add_subplot (1, 3, 1)
plt.axis("off")
```

```
plt.imshow(img1[:, :, 0], cmap='Reds', vmin=0, vmax=255)
plt.title("Red_Channel")

fig.add_subplot (1, 3, 2)
plt.axis("off")
plt.imshow(img1[:, :, 1], cmap='Greens', vmin=0, vmax=255)
plt.title("Green_Channel")

fig.add_subplot (1, 3, 3)
plt.axis("off")
plt.imshow(img1[:, :, 2], cmap='Blues', vmin=0, vmax=255)
plt.title("Blue_Channel")
```

Digital Image Processing Lab
ECE-324
Experiment No-2
डिजिटल इमेज प्रोसेसिंग लैब
प्रयोग संख्या-2

Aim:

To study and perform arithmetic operations of images:

i. Addition operation
ii. Subtraction operation
iii. Multiplication operation
iv. Division operation
v. To perform the arithmetic operations in different channels (R-G, B-R, B-G) of color images.

उद्देश्य:

छवियों के अंकगणितीय संचालन का अध्ययन करना और निष्पादित करना:

i. जोड़ संचालन
ii. घटाव संचालन
iii. गुणा संचालन
iv. भाग संचालन
v. रंगीन छवियों के विभिन्न चैनलों (आर-जी, बी-आर, बी-जी) में अंकगणितीय संचालन करना।

Theory:

The important requirement in image arithmetic is that all (input and output) the images are of the same size MxM.

Arithmetic operations are done pixelwise. Let p = A(x,y) and q = B(x,y) be the pixel values to be operated on and r =I(x,y) be the result of the operation.

लिखित:

छवि अंकगणित में महत्वपूर्ण आवश्यकता यह है कि सभी (इनपुट और आउटपुट) छवियाँ एक ही आकार MxM की हों। अंकगणितीय ऑपरेशन पिक्सेल के अनुसार किए जाते हैं। मान लें कि p = A(x,y) और q = B(x,y) पिक्सेल मान हैं जिन पर ऑपरेशन किया जाना है और r = I(x,y) ऑपरेशन का परिणाम है।

Addition :

$I(x,y) = A(x,y) + B(x,y) \longrightarrow r = p + q$

Subtraction :

$I(x,y) = A(x,y) - B(x,y) \longrightarrow r = p - q$

Difference :

$I(x,y) = |A(x,y) - B(x,y)| \rightarrow r = |p - q|$

Multiplication :

$I(x,y) = A(x,y) \times B(x,y) \rightarrow r = p \times q$

Division :

$I(x,y) = A(x,y) / B(x,y) \rightarrow r = p / q$

जोड़:

$I(x,y) = A(x,y) + B(x,y) \rightarrow r = p + q$

घटाना:

$I(x,y) = A(x,y) - B(x,y) \rightarrow r = p - q$

अंतर:

$I(x,y) = |A(x,y) - B(x,y)| \rightarrow r = |p - q|$

गुणन:

$I(x,y) = A(x,y) \times B(x,y) \rightarrow r = p \times q$

विभाजन:

$I(x,y) = A(x,y) / B(x,y) \rightarrow r = p / q$

Implementation issues:

Digital images are stored as b - bit images. Hence, the range of values a pixel can take is restricted to the range [0, 1,... (2b -1)]. With b= 8 this range is [0,1,...255]. The closed interval poses a problem when performing arithmetic operations in practice, as the results are not guaranteed to be within this interval.

कार्यान्वयन संबंधी मुद्दे:

डिजिटल छवियों को बी-बिट छवियों के रूप में संग्रहीत किया जाता है। इसलिए, पिक्सेल द्वारा लिए जा सकने वाले मानों की सीमा [0, 1,... (2b -1)] तक सीमित है। b= 8 के साथ यह सीमा [0,1,...255] है। व्यवहार में अंकगणितीय संचालन करते समय बंद अंतराल एक समस्या उत्पन्न करता है, क्योंकि परिणाम इस अंतराल के भीतर होने की गारंटी नहीं है।

Coding

```
import cv2
img1=cv2.imread("/content/Baboon.jpg")
img2=cv2.imread("/content/Cameraman1.jpg")
```

First off all know the shape of an img1 and img2 to perform the arithmatic operation

```
img1.shape
```

```
img2.shape
```

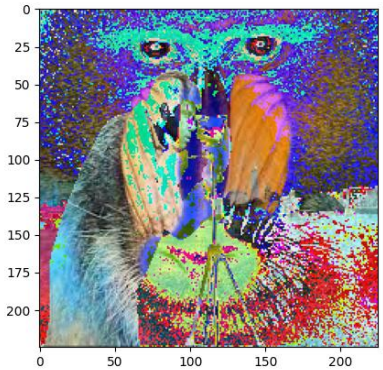
```
from google.colab.patches import cv2_imshow
# google colab is not supporting cv2.imshow #
# so at first we have to import cv2_imshow from google colab as above #
cv2_imshow(img1)
```

Because of large size img2 is resized according to the size of img1

```
img2 = cv2.resize(img2, (225, 225))
img2.shape
cv2_imshow(img2)
```

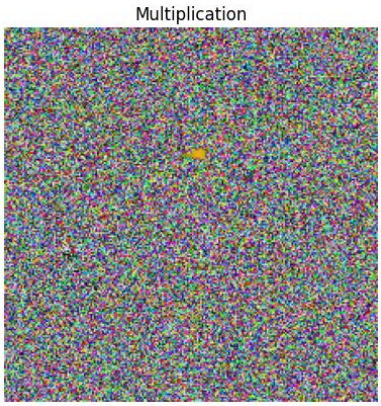
Perform Addition operation img3

```
import numpy as np
img3 = np.add (img1, img2)

## plot img3 using matplotlib package ##
import matplotlib.pyplot as plt
plt.imshow(img3)

## To remove axis in the image plot use plt.axis("off") ##
## For showing the title use plt.title("title")
plt.imshow(img3)
plt.axis("off")
plt.title("Addition")
```




```
## perform Subtraction Operation ##
img4 = np.subtract(img1, img2)

## Plot img4 ##
plt.imshow(img4)
plt.axis("off")
plt.title("Subtraction")
```

```
## perform Multiplication Operation ##
img5 = np.multiply(img1, img2)

## Plot img4 ##
plt.imshow(img5)
plt.axis("off")
plt.title("Multiplication")
```

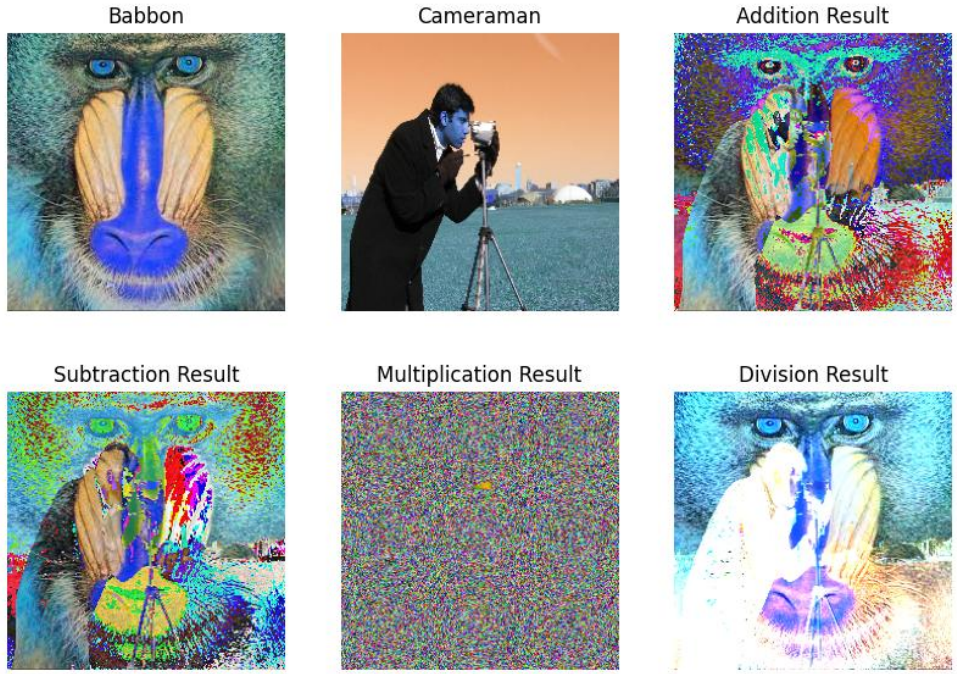


```
## perform Division Operation ##
img6 = np.divide(img1, img2)

## Plot img4 ##
plt.imshow(img6)
plt.axis("off")
plt.title("Division")
```

Plot multiple subfigures or images in one frame (figure)

```
fig = plt.figure(figsize=(10, 7))
fig.add_subplot(2, 3, 1)
plt.imshow(img1)
plt.axis('off')
plt.title("Babbon")

fig.add_subplot(2, 3, 2)
plt.imshow(img2)
plt.axis('off')
plt.title("Cameraman")

fig.add_subplot(2, 3, 3)
plt.imshow(img3)
plt.axis('off')
plt.title("Addition Result")

fig.add_subplot(2, 3, 4)
plt.imshow(img4)
plt.axis('off')
plt.title("Subtraction Result")

fig.add_subplot(2, 3, 5)
```

```
plt.imshow(img5)
plt.axis('off')
plt.title("Multiplication Result")

fig.add_subplot(2, 3, 6)
plt.imshow(img6)
plt.axis('off')
plt.title("Division Result")
```



Digital Image Processing Lab
ECE-324
Experiment No-3
डिजिटल इमेज प्रोसेसिंग लैब
प्रयोग संख्या-3

Aim:

To study and perform logical operations of images:

A. NOT operation

B. AND operation

C. OR operation

D. XOR operation

उद्देश्य:

छवियों के तार्किक संचालन का अध्ययन करना और निष्पादित करना:

A. NOT संचालन

B. AND संचालन

C. OR संचालन

D. XOR संचालन

Theory:

Bitwise Operations

Bitwise operations are used in image manipulation to extract important parts. The following Bitwise operations are used in this article:

1. **AND**
2. **OR**
3. **NOT**
4. **XOR**

सिद्धांत:

बिटवाइज़ ऑपरेशन

बिटवाइज़ ऑपरेशन का उपयोग इमेज मैनिपुलेशन में महत्वपूर्ण भागों को निकालने के लिए किया जाता है। इस लेख में निम्नलिखित बिटवाइज़ ऑपरेशन का उपयोग किया गया है:

1. AND
2. OR
3. NOT
4. XOR

Bitwise operations are also useful for image masking. These operations can be used to enable image creation. These operations can help to improve the properties of the input images.

NOTE: Bitwise operations should only be performed on input images of the same dimensions.

बिटवाइज़ ऑपरेशन इमेज मास्किंग के लिए भी उपयोगी होते हैं। इन ऑपरेशन का उपयोग इमेज निर्माण को सक्षम करने के लिए किया जा सकता है। ये ऑपरेशन इनपुट इमेज के गुणों को बेहतर बनाने में मदद कर सकते हैं।
नोट: बिटवाइज़ ऑपरेशन केवल समान आयामों वाली इनपुट इमेज पर ही किए जाने चाहिए।

AND Bitwise Operation of Image

The AND operator (and the NAND operator in a similar fashion) typically takes two binary or integer graylevel images as input and produces a third image whose pixel values are just those of the first image ANDed with the corresponding pixels from the second. This operator can be modified to produce the output by taking a single input image and ANDing each pixel with a predetermined constant value.

Syntax: cv2.bitwise_and(Image1, Image2, destination, mask)

Parameters:

1. Image1**:** First Input Image numpy array
2. Image1**:** Second Input Image numpy array
3. destination**:** Output array
4. mask**:** Operation mask image

छवि का AND बिटवाइज़ संचालन

AND ऑपरेटर (और इसी तरह NAND ऑपरेटर) आम तौर पर दो बाइनरी या पूर्णांक ग्रेलेवल छवियों को इनपुट के रूप में लेता है और एक तीसरी छवि बनाता है जिसके पिक्सेल मान ठीक पहले छवि के होते हैं, जो दूसरे से संगत पिक्सेल के साथ ANDed होते हैं। इस ऑपरेटर को एकल इनपुट छवि लेकर और प्रत्येक पिक्सेल को पूर्व निर्धारित स्थिर मान के साथ ANDing करके आउटपुट बनाने के लिए संशोधित किया जा सकता है।

वाक्यविन्यास: cv2.bitwise_and(Image1, Image2, destination, mask)

पैरामीटर:

1. Image1: पहली इनपुट छवि numpy array
2. Image1: दूसरी इनपुट छवि numpy array
3. destination: आउटपुट सरणी
4. mask: ऑपरेशन मास्क छवि

Code :

```
import cv2
import numpy as np
img1 = cv2.imread('input1.png')
img2 = cv2.imread('input2.png')
dest_and = cv2.bitwise_and(img2, img1, mask = None)
cv2.imshow('Bitwise And', dest_and)
cv2.waitKey(0)
```

OR Bitwise Operation of Image

The OR operator typically takes two binary or greyscale images as input and outputs a third image whose pixel values are the first image's pixel values ORed with the corresponding pixels from the second. A variant of this operator takes a single input image and ORs each pixel with a constant value to generate the output.

Syntax: cv2.bitwise_or(source1, source2, destination, mask)

Parameters:

1. source1**:** First Input numpy Image array
2. source2**:** Second Input numpy Image array
3. destination**:** Output array image
4. mask**:** Operation mask, input / output 8-bit single-channel mask.

छवि का OR बिटवाइज़ संचालन

OR ऑपरेटर आम तौर पर दो बाइनरी या ग्रेस्केल छवियों को इनपुट के रूप में लेता है और एक तीसरी छवि को आउटपुट करता है, जिसके पिक्सेल मान पहली छवि के पिक्सेल मान होते हैं, जो दूसरी छवि के संगत पिक्सेल के साथ ORed होते हैं। इस ऑपरेटर का एक प्रकार एक एकल इनपुट छवि लेता है और आउटपुट उत्पन्न करने के लिए प्रत्येक पिक्सेल को एक स्थिर मान के साथ ORs करता है।

सिंटेक्स: cv2.bitwise_or(source1, source2, destination, mask)

पैरामीटर:

1. source1: पहला इनपुट numpy इमेज सरणी
2. source2: दूसरा इनपुट numpy इमेज सरणी
3. गंतव्य: आउटपुट सरणी छवि
4. मास्क: ऑपरेशन मास्क, इनपुट / आउटपुट 8-बिट सिंगल-चैनल मास्क।

Code :

```
import cv2
import numpy as np
img1 = cv2.imread('input1.png')
img2 = cv2.imread('input2.png')
dest_or = cv2.bitwise_or(img1, img2, mask = None)
cv2.imshow('Bitwise OR', dest_or)
cv2.waitKey(0)
```

NOT Bitwise Operation of Image

Logical NOT, also known as invert, is an operator that takes a binary or grayscale image as input and generates its photographic negative.

Syntax: cv2.bitwise_not(Image1, Destination, mask)

Parameters:

1. Image1**:** Input Image array.
2. Destination**:** Output array image
3. mask**:** Operation mask

छवि का NOT बिटवाइज़ ऑपरेशन

लॉजिकल NOT, जिसे इनवर्ट के नाम से भी जाना जाता है, एक ऑपरेटर है जो बाइनरी या ग्रेस्केल छवि को इनपुट के रूप में लेता है और उसका फोटोग्राफिक नेगेटिव जेनरेट करता है।

सिंटेक्स: cv2.bitwise_not(Image1,Destination, mask)

पैरामीटर:

1. Image1: इनपुट इमेज ऐरे।
2. डेस्टिनेशन: आउटपुट ऐरे इमेज

3. मास्क: ऑपरेशन मास्क

Code:

```
import cv2
import numpy as np
img1 = cv2.imread('input1.png')
dest_not = cv2.bitwise_not(img1, mask = None)
cv2.imshow('Bitwise Not', dest_not)
cv2.waitKey(0)
```

XOR Bitwise Operation of Image

The operation is carried out simply and in a single pass. It is critical that all of the input pixel values being processed have the same number of bits, or else unexpected results may occur. When the pixel values in the input images are not simple 1-bit numbers, the XOR operation is typically (but not always) performed bitwise on each corresponding bit in the pixel values.

Syntax: cv2.bitwise_xor(source1, source2, destination, mask)

Parameters:

1. source1**:** First Input Image array (Single-channel, 8-bit or floating-point)
2. source2**:** Second Input Image array (Single-channel, 8-bit or floating-point)
3. destination**:** Output image array
4. mask**:** Operation mask, input/ output 8-bit single-channel mask.

छवि का XOR बिटवाइज़ ऑपरेशन

यह ऑपरेशन सरलता से और एक ही पास में किया जाता है। यह महत्वपूर्ण है कि संसाधित किए जा रहे सभी इनपुट पिक्सेल मानों में बिट्स की संख्या समान हो, अन्यथा अप्रत्याशित परिणाम हो सकते हैं। जब इनपुट छवियों में पिक्सेल मान सरल 1-बिट संख्याएँ नहीं होती हैं, तो XOR ऑपरेशन आमतौर पर (लेकिन हमेशा नहीं) पिक्सेल मानों में प्रत्येक संगत बिट पर बिटवाइज़ किया जाता है।

सिंटैक्स: cv2.bitwise_xor(source1, source2, destination, mask)

पैरामीटर:

1. source1: पहला इनपुट इमेज ऐरे (सिंगल-चैनल, 8-बिट या फ़्लोटिंग-पॉइंट)
2. source2: दूसरा इनपुट इमेज ऐरे (सिंगल-चैनल, 8-बिट या फ़्लोटिंग-पॉइंट)
3. डेस्टिनेशन: आउटपुट इमेज ऐरे
4. मास्क: ऑपरेशन मास्क, इनपुट/आउटपुट 8-बिट सिंगल-चैनल मास्क।

Code:

```
import cv2
import numpy as np
img1 = cv2.imread('input1.png')
img2 = cv2.imread('input2.png')
dest_or = cv2.bitwise_xor(img1, img2, mask = None)
cv2.imshow('Bitwise XOR', dest_xor)
```

cv2.waitKey(0)

Note

Many applications use processed images taken from the same scene at different points, such as noise reduction by adding successive images of the same scene or motion detection by subtracting two successive images. Logical operators are frequently used to combine two (mostly binary) images. In the case of integer images, the logical operator is typically used bitwise. Then, for example, we can use a binary mask to select a specific region of an image.

नोट

कई अनुप्रयोग एक ही दृश्य से अलग-अलग बिंदुओं पर ली गई संसाधित छवियों का उपयोग करते हैं, जैसे कि एक ही दृश्य की लगातार छवियों को जोड़कर शोर में कमी या दो लगातार छवियों को घटाकर गति का पता लगाना। तार्किक ऑपरेटरों का उपयोग अक्सर दो (अधिकांशतः बाइनरी) छवियों को संयोजित करने के लिए किया जाता है। पूर्णांक छवियों के मामले में, तार्किक ऑपरेटर का उपयोग आमतौर पर बिटवाइज़ किया जाता है। फिर, उदाहरण के लिए, हम किसी छवि के किसी विशिष्ट क्षेत्र का चयन करने के लिए बाइनरी मास्क का उपयोग कर सकते हैं।

Digital Image Processing Lab
Experiment No-4
डिजिटल इमेज प्रोसेसिंग लैब
प्रयोग संख्या-4

Aim:
To study and perform geometric operations on images:

A. **Translation operation**
B. **Rotation operation**
C. **Shearing operation**

उद्देश्य: छवियों पर ज्यामितीय संचालन का अध्ययन और निष्पादन करने के लिए:

A. अनुवाद संचालन

B. घूर्णन संचालन

C. कतरनी संचालन

Theory:
Translation
Image translation refers to the rectilinear shift of an object i.e. an image from one location to another. If we know the amount of shift in horizontal and the vertical direction, say (tx, ty) then we can make a transformation matrix e.g.

$$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \end{bmatrix}$$

where tx denotes the shift along the x-axis and ty denotes shift along the y-axis i.e. the number of pixels by which we need to shift about in that direction. Now, we can use the cv2.wrapAffine() function to implement these translations. This function requires a 2×3 array. The Numpy array should be of float type.

सिद्धांत: (i) अनुवाद

$$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \end{bmatrix}$$

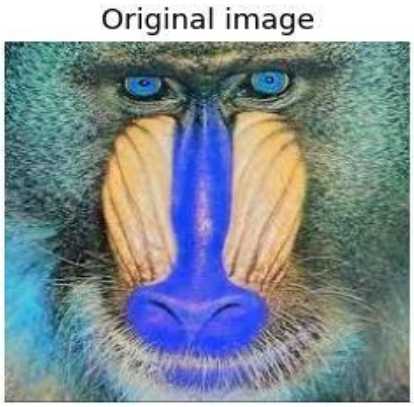
छवि अनुवाद किसी वस्तु यानी छवि के एक स्थान से दूसरे स्थान पर सीधे रेखीय बदलाव को संदर्भित करता है। यदि हम क्षैतिज और ऊर्ध्वाधर दिशा में बदलाव की मात्रा जानते हैं, मान लें (tx, ty) तो हम एक परिवर्तन मैट्रिक्स बना सकते हैं जैसे

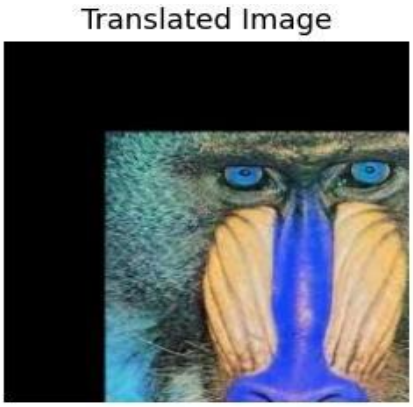
जहाँ tx x-अक्ष के साथ बदलाव को दर्शाता है और ty y-अक्ष के साथ बदलाव को दर्शाता है यानी पिक्सेल की संख्या जिसके द्वारा हमें उस दिशा में बदलाव करने की आवश्यकता है। अब, हम इन अनुवादों को लागू करने के लिए cv2.wrapAffine() फ़ंक्शन का उपयोग कर सकते हैं। इस फ़ंक्शन के लिए 2×3 सरणी की आवश्यकता होती है। Numpy सरणी फ़्लोट प्रकार की होनी चाहिए।

Python Code :-

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img1 = cv2.imread('/content/Baboon.jpg')
# Store height and width of the image
height, width = img1.shape[:2]
quarter_height, quarter_width = height / 4, width / 4
T = np.float32([[1, 0, quarter_width], [0, 1,
  quarter_height]]) # We use warpAffine to transform
```

```
# the image using the matrix, T
img_translation = cv2.warpAffine(img1, T, (width, height))

fig=plt.figure(figsize=(10, 3))
fig.add_subplot(1,2,1)
plt.imshow(img1) plt.axis("off")
plt.title("Original image")

fig.add_subplot(1,2,2)
plt.imshow(img_translation)
plt.axis("off")
plt.title("Translated Image")
```





Advantages/application of image translation are:

- **Hiding a part of the image**
- **Cropping an image**
- **Shifting an image**
- **Animating an image using image translations in loop.**

छवि अनुवाद के लाभ/अनुप्रयोग हैं:

• छवि का एक हिस्सा छिपाना

• छवि को क्रॉप करना

• छवि को शिफ्ट करना

• लूप में छवि अनुवाद का उपयोग करके छवि को एनिमेट करना।

Rotation

Images can be rotated to any degree clockwise or otherwise. We just need to define rotation matrix listing rotation point, degree of rotation and the scaling factor.

The cv2.getRotationMatrix2D () function is used to create a rotation matrix for an image. It takes the following arguments:

- **The center of rotation for the image.**
- **The angle of rotation in degrees.**
- **The scale factor.**

The cv2.warpAffine() function is used to apply a transformation matrix to an image.

It takes the following arguments:

The python image to be transformed.

The transformation matrix.

The output image size.

The rotation angle can be positive or negative. A positive angle rotates the image clockwise, while a negative angle rotates the image counterclockwise.

The scale factor can be used to scale the image up or down. A scale factor of 1 will keep the image the same size, while a scale factor of 2 will double the size of the python image.

(ii) रोटेशन

छवियों को घड़ी की दिशा में या अन्यथा किसी भी डिग्री पर घुमाया जा सकता है। हमें बस रोटेशन मैट्रिक्स को परिभाषित करने की आवश्यकता है जिसमें रोटेशन पॉइंट, रोटेशन की डिग्री और स्केलिंग फैक्टर शामिल हो।

cv2.getRotationMatrix2D() फ़ंक्शन का उपयोग किसी छवि के लिए रोटेशन मैट्रिक्स बनाने के लिए किया जाता है। यह निम्नलिखित तर्क लेता है:

• छवि के लिए रोटेशन का केंद्र।

• डिग्री में रोटेशन का कोण।

• स्केल फैक्टर।

• cv2.warpAffine() फ़ंक्शन का उपयोग किसी छवि पर परिवर्तन मैट्रिक्स लागू करने के लिए किया जाता है। यह निम्नलिखित तर्क लेता है:

• रूपांतरित की जाने वाली पायथन छवि।

• परिवर्तन मैट्रिक्स।

• आउटपुट छवि का आकार।

• रोटेशन एंगल सकारात्मक या नकारात्मक हो सकता है। सकारात्मक कोण छवि को दक्षिणावर्त घुमाता है, जबकि नकारात्मक कोण छवि को वामावर्त घुमाता है।

• स्केल फैक्टर का उपयोग छवि को ऊपर या नीचे स्केल करने के लिए किया जा सकता है। 1 का स्केल फैक्टर छवि को समान आकार में रखेगा, जबकि 2 का स्केल फैक्टर पायथन छवि के आकार को दोगुना कर देगा।

 # Import the necessary Libraries

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img2 =
cv2.imread('/content/Baboon.jpg
') rows, cols = img2.shape[:2]
M = np.float32([[1,  0, 0], [0, -1, rows], [0,  0, 1]])
img_rotation = cv2.warpAffine(img2,
cv2.getRotationMatrix2D((cols/2, rows/2),30, 0.6), (cols,rows))
```

 # Create subplots

```
fig=plt.figure(figsize=(10, 3))
fig.add_subplot(1,2,1)
plt.imshow(img2)
plt.axis("off")
plt.title("Original image")

fig.add_subplot(1,2,2)
plt.imshow(img_rotation)
plt.axis("off")
plt.title("Rotated Image")
```

(i) Image Shearing

The shear() function is an inbuilt function in the Python Wand Image Magick library which is used to slide one edge of an image along the X or Y axis to create a parallelogram. The X-direction shear slides an edge along the X-axis, while a Y direction shear slides an edge along the Y-axis. The shear angle is used to set shear of the image.

(iii) इमेज शियरिंग

शियर() फ़ंक्शन पायथन वैंड इमेज मैजिक लाइब्रेरी में एक इनबिल्ट फ़ंक्शन है जिसका उपयोग किसी छवि के एक किनारे को X या Y अक्ष के साथ स्लाइड करके समांतर चतुर्भुज बनाने के लिए किया जाता है। X-दिशा शियर किनारे को X-अक्ष के साथ स्लाइड करता है, जबकि Y दिशा शियर किनारे को Y-अक्ष के साथ स्लाइड करता है। शियर एंगल का उपयोग छवि के शियर को सेट करने के लिए किया जाता है।

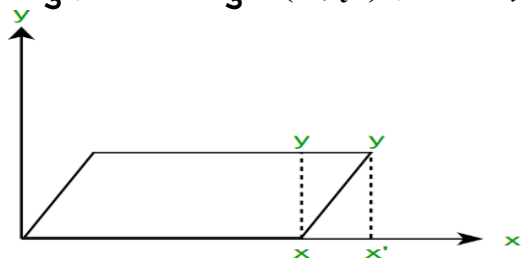
Shearing deals with changing the shape and size of the 2D object along x-axis and y-axis. It is similar to sliding the layers in one direction to change the shape of the 2D object. It is an ideal technique to change the shape of an existing object in a two-dimensional plane. In a two- dimensional plane, the object size can be changed along X direction as well as Y direction.

शियरिंग 2D ऑब्जेक्ट के आकार और साइज़ को x-अक्ष और y-अक्ष के साथ बदलने से संबंधित है। यह 2D ऑब्जेक्ट के आकार को बदलने के लिए परतों को एक दिशा में स्लाइड करने के समान है। यह दो आयामी तल में मौजूदा ऑब्जेक्ट के आकार को बदलने के लिए एक आदर्श तकनीक है। दो आयामी तल में, ऑब्जेक्ट का आकार X दिशा के साथ-साथ Y दिशा में भी बदला जा सकता है।

x-Shear: In x shear, the y co-ordinates remain the same but the x co-ordinates changes. If P (x, y) is the point then the new points will be P'(x', y') given as –

$$x' = x + Sh_x * y; y' = y$$

x- शियर: x शियर में, y निर्देशांक समान रहते हैं लेकिन x निर्देशांक बदल जाते हैं। यदि P(x, y) बिंदु है तो नए बिंदु P'(x', y') होंगे, जो इस प्रकार होंगे –



मैट्रिक्स फॉर्म:

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ Sh_x & 1 \end{bmatrix}$$

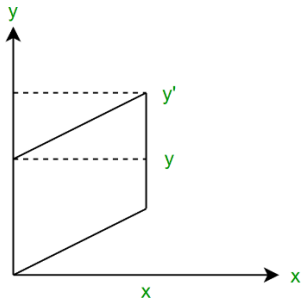
Matrix Form:

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ Sh_x & 1 \end{bmatrix}$$

$$x' = x; y' = y + Sh_y * x$$

y-Shear:

In y shear, the x co-ordinates remain the same but the y co-ordinates changes. If P(x, y) is the point then the new points will be P'(x', y') given as –



y- शियर:

y शियर में, x निर्देशांक समान रहते हैं लेकिन y निर्देशांक बदल जाते हैं। यदि P(x, y) बिंदु है तो नए बिंदु P'(x', y') होंगे, जो इस प्रकार दिए गए हैं

मैट्रिक्स फॉर्म:

$$[x' \quad y'] = [x \quad y] \cdot \begin{bmatrix} 1 & Sh_y \\ 0 & 1 \end{bmatrix}$$

$$[x' \quad y'] = [x \quad y] \cdot \begin{bmatrix} 1 & Sh_y \\ 0 & 1 \end{bmatrix}$$

Matrix Form:

x-y Shear:

In x-y shear, both the x and y co-ordinates changes. If P (x, y) is the point then the new points will be P'(x', y') given as –

$$x' = x + Sh_x * y; y' = y + Sh_y * x$$

x-y शियर:

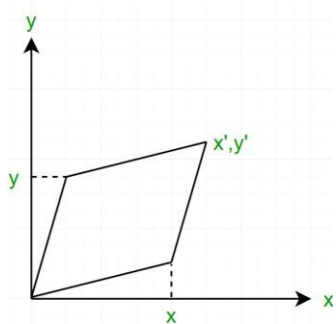
x-y शियर में, x और y दोनों निर्देशांक बदल जाते हैं। यदि P(x, y) बिंदु है तो नए

$$x' = x + Sh_x * y; y' = y + Sh_y * x$$

बिंदु P'(x', y') होंगे, जो इस प्रकार दिए गए हैं –

$$[x' \quad y'] = [x \quad y] \cdot \begin{bmatrix} 1 & Sh_y \\ Sh_x & 1 \end{bmatrix}$$

Matrix Form:



Example:
Given a triangle with points (1, 1), (0, 0) and (1, 0). Find out the new coordinates of the object along x-axis, y-axis, xy-axis. (Applying shear parameter 4 on X-axis and 1 on Y-axis.).

Given,
Old corner coordinates of the triangle = A (1, 1), B (0, 0), C(1, 0)
Shearing parameter along X-axis (Shx) = 4 Shearing parameter along Y-axis (Shy) = 1
Along x-axis:
A'=(1+4*1, 1)=(5, 1)
B'=(0+4*0, 0)=(0, 0)
C'=(1+4*0, 0)=(1, 0)

Along y-axis:
A''=(1, 1+1*1)=(1, 2)
B''=(0, 0+1*0)=(0, 0)
C''=(1, 0+1*1)=(1, 1)

Along xy-axis:
A'''=(1+4*1, 1+1*1)=(5, 2)
B'''=(0+4*0, 0+1*0)=(0, 0)
C'''=(1+4*0, 0+1*1)=(1, 1)

उदाहरण:

बिन्दुओं (1, 1), (0, 0) और (1, 0) वाला एक त्रिभुज दिया गया है। x-अक्ष, y-अक्ष, xy-अक्ष के साथ वस्तु के नए निर्देशांक ज्ञात करें। (X-अक्ष पर कतरनी पैरामीटर 4 और Y-अक्ष पर 1 लागू करके।) दिया गया है,

त्रिभुज के पुराने कोने निर्देशांक = A (1, 1), B(0, 0), C(1, 0)

X-अक्ष के साथ कतरनी पैरामीटर (Shx) = 4

Y-अक्ष के साथ कतरनी पैरामीटर (Shy) = 1

x-अक्ष के साथ:

A'= (1+4*1, 1) = (5, 1)

B'= (0+4*0, 0) = (0, 0)

C'= (1+4*0, 0) = (1, 0)

y-अक्ष के साथ:
A''=(1, 1+1*1)=(1, 2)
B''=(0, 0+1*0)=(0, 0)
C''=(1, 0+1*1)=(1, 1)

xy-अक्ष के साथ:

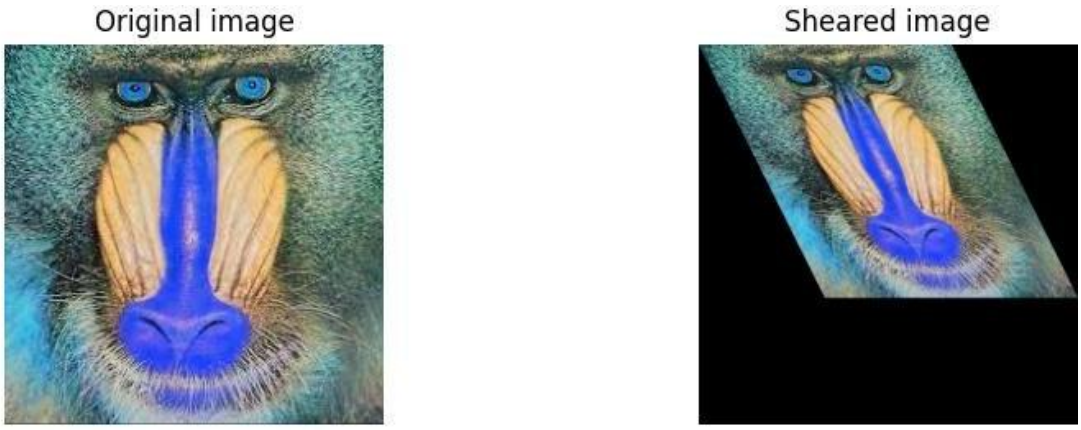
A'''= (1+4*1, 1+1*1) =(5, 2)

बी'''=(0+4*0, 0+1*0)=(0, 0)

सी'''=(1+4*0, 0+1*1)=(1, 1)

########### Python Code #################

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
img = cv2.imread('/content/Baboon.jpg')
rows, cols = img.shape[:2]
M = np.float32([[1, 0.5, 0], [0, 1, 0], [0, 0, 1]])
sheared_img = cv2.warpPerspective(img, M, (int(cols*1.5), int(rows*1.5)))
from google.colab.patches import cv2_imshow
fig=plt.figure(figsize=(10, 3))
fig.add_subplot(1,2,1)
plt.imshow(img)
plt.axis("off")
plt.title("Original image")


fig.add_subplot(1,2,2)
plt.imshow(sheared_img)
plt.axis("off")
```



Digital Image Processing Lab
Experiment No-5
डिजिटल इमेज प्रोसेसिंग लैब
प्रयोग संख्या-5

Aim:
To study and perform linear neighborhood operations on images:

- **A.** With different convolution kernels.
- **B.** With convolution kernels of different size.
- **C.** With various padding operations (zero, 1's, line, mirror)

Neighborhood operations are a generalization of the point operations. A pixel in the processed image now depends not only on the corresponding pixel in the input image but also its neighboring pixels. This generalization also allows for defining linear as well nonlinear filtering operations

उद्देश्य:

छवियों पर रैखिक पड़ोस संचालन का अध्ययन और प्रदर्शन करना:

A. विभिन्न संवलन कर्नेल के साथ।

B. विभिन्न आकार के संवलन कर्नेल के साथ।

C. विभिन्न पैडिंग संचालन (शून्य, 1, रेखा, दर्पण) के साथ

पड़ोस संचालन बिंदु संचालन का एक सामान्यीकरण है। संसाधित छवि में एक पिक्सेल अब न केवल इनपुट छवि में संबंधित पिक्सेल पर निर्भर करता है, बल्कि इसके पड़ोसी पिक्सेल पर भी निर्भर करता है। यह सामान्यीकरण रैखिक और साथ ही गैर-रैखिक फ़िल्टरिंग संचालन को परिभाषित करने की भी अनुमति देता है।

Convolution Operation
Given an input image (x, y) an output image $g\ (x, y)$ is computed by applying some operation on a local neighbourhood N of each pixel in the image f. This can be visualized as follows: a window or mask is placed at every pixel location in (x, y) and some operation is performed on the pixels within the window. The window is moved to the next pixel location and the process is

$$g(x,y) = H_N(f(x,y))$$

repeated. Thus,
Where H_N is the neighborhood operator of size N and g is the output image.

कन्वोल्यूशन ऑपरेशन

इनपुट इमेज $f(x, y)$ दिए जाने पर, इमेज f में प्रत्येक पिक्सेल के स्थानीय पड़ोस N पर कुछ ऑपरेशन लागू करके आउटपुट इमेज $g(x, y)$ की गणना की जाती है। इसे इस प्रकार देखा जा सकता है: $f(x, y)$ में प्रत्येक पिक्सेल स्थान पर एक विंडो या मास्क रखा जाता है और विंडो के भीतर पिक्सेल पर कुछ ऑपरेशन किया जाता है। विंडो को अगले पिक्सेल स्थान पर ले जाया जाता है और प्रक्रिया को दोहराया जाता है। इस प्रकार,

$$g(x,y) = H_N(f(x,y))$$

जहाँ HN आकार N का पड़ोस ऑपरेटर है और g आउटपुट इमेज है।

Linear operations:

Linear operations can be represented as a convolution operation between (x, y) and a window function $w\ (x, y)$ as follows.

$$g(x,y) = \sum_{u=-a}^{a} \sum_{v=-b}^{b} w(u,v) f(x+u, y+v)$$

w is a window function which in practice is in the form of a matrix of size $s \times t$; $a = (s - 1)/2\ and\ b = (t - 1)/2$. An example of 3x3 (x, y) is shown below**.**

रैखिक संक्रियाएँ:

रैखिक संक्रियाओं को $f(x, y)$ और विंडो फ़ंक्शन $w(x, y)$ के बीच एक संवलन संक्रिया के रूप में इस प्रकार दर्शाया जा सकता है

$$g(x,y) = \sum_{u=-a}^{a} \sum_{v=-b}^{b} w(u,v) f(x+u, y+v)$$

w(-1,-1)	w(-1,0)	w(-1,1)
w(0,-1)	w(0,0)	w(0,1)
w(1,-1)	w(1,0)	w(1,1)

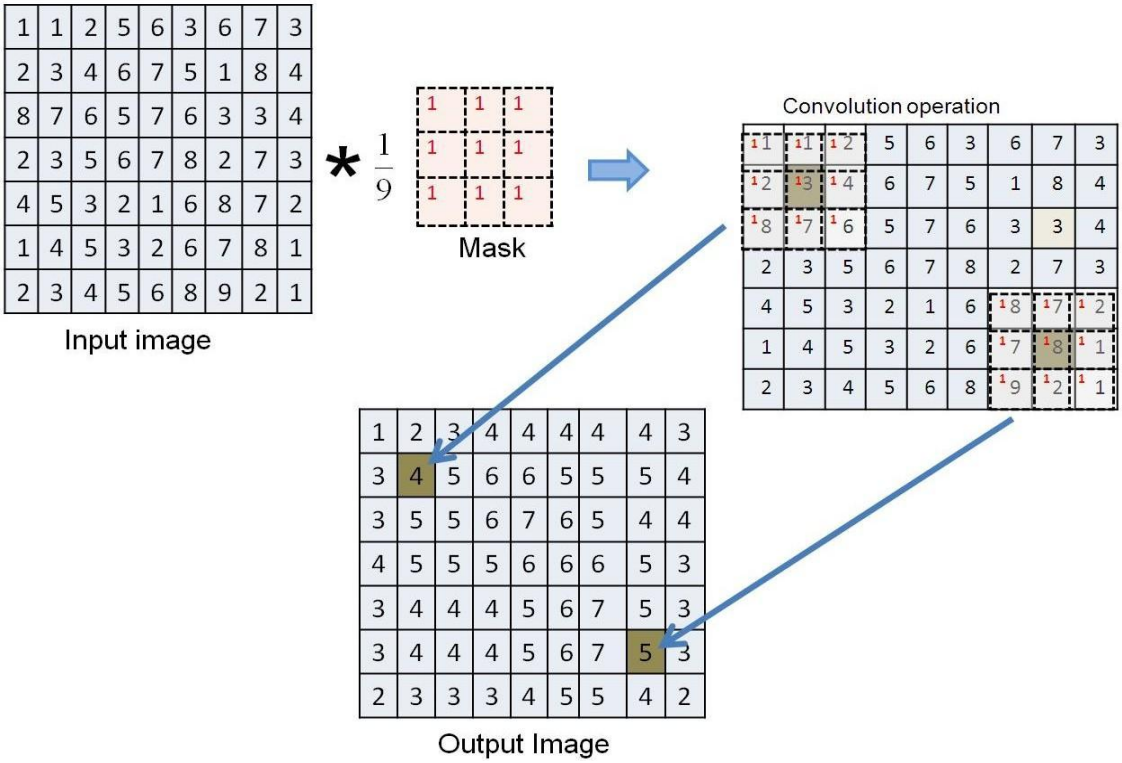
An example of convolution operation is shown below:

Note that (0,0) is the coefficient or weight of the window function at the center which is the position of the current pixel in $f(x, y)$. Varying the weights (i, j), results in

different type of linear neighborhood operations. The above operations are generally termed as filtering operations and w is referred to as a filter.

कन्वोल्यूशन ऑपरेशन का एक उदाहरण नीचे दिखाया गया है:

ध्यान दें कि $w(0,0)$ केंद्र में विंडो फ़ंक्शन का गुणांक या भार है जो $f(x, y)$ में वर्तमान पिक्सेल की स्थिति है। भार $w(i, j)$ को बदलने से विभिन्न प्रकार के रैखिक पड़ोस संचालन होते हैं। उपरोक्त संचालन को आम तौर पर फ़िल्टरिंग ऑपरेशन कहा जाता है और w को फ़िल्टर के रूप में संदर्भित किया जाता है।



A popular filter is one which performs local averaging or smoothing of the image. This is a low pass filter.

एक लोकप्रिय फ़िल्टर वह है जो छवि का स्थानीय औसत या समतलीकरण करता है। यह एक लो पास फ़िल्टर है।

Table of various popular kernels

विभिन्न लोकप्रिय कर्नेल की तालिका

Sr No.	Kernel	Effect
1	$\begin{bmatrix} 0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 0.0625 & 0.125 & 0.0625 \end{bmatrix}$	Blur
2	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$	Bottom Sobel
3	$\begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$	Emboss
4	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	Identity
5	$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$	Left Sobel
6	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -2 \\ -1 & -1 & -1 \end{bmatrix}$	Outline
7	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$	Right Sobel
8	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	Sharpen
9	$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$	Top Sobel

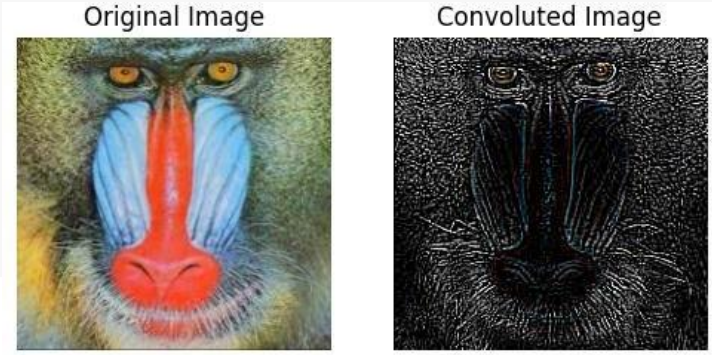
The key characteristic of this filter is that $(i, j) > 0$ for every (i, j). An additional constraint is generally imposed on the weights to sum to 1. Below are some examples.

इस फ़िल्टर की मुख्य विशेषता यह है कि प्रत्येक (i, j) के लिए $w(i, j) > 0$ होता है। आम तौर पर वज़न पर एक अतिरिक्त प्रतिबंध लगाया जाता है ताकि योग 1 हो। नीचे कुछ उदाहरण दिए गए हैं।

Code:

```
################ Convolution Operation in an Image #################### img4 =
cv2.imread("/content/Baboon.jpg")
img4 = cv2.resize(img4, (224,224))
img4 = cv2.cvtColor(img4, cv2.COLOR_BGR2RGB) from PIL import Image
import numpy as np
def apply_convolution(img:np.array, kernel:np.array):
# Get the height, width, and number of channels of the image height,width,c
=img.shape[0],img.shape[1],img.shape[2]
# Get the height, width, and number of channels of the kernel kernel_height,kernel_width =
kernel.shape[0],kernel.shape[1] # Create a new image of original img size minus the border
# where the convolution can't be applied
new_img = np.zeros((height-kernel_height+1,width-kernel_width+1,3))
# Loop through each pixel in the image
# But skip the outer edges of the image
```

```
for i in range(kernel_height//2, height-kernel_height//2-1): for j in range(kernel_width//2, width-kernel_width//2-1):
# Extract a window of pixels around the current pixel window = img[i-kernel_height//2 : i+kernel_height//2+1,j-kernel_width//2 : j+kernel_width//2+1]
# Apply the convolution to the window and set the result as the value of the current pixel in the new image
new_img[i, j, 0] = int((window[:,:,0] * kernel).sum())
new_img[i, j, 1] = int((window[:,:,1] * kernel).sum())
new_img[i, j, 2] = int((window[:,:,2] * kernel).sum())
# Clip values to the range 0-255 new_img = np.clip(new_img, 0, 255) return new_img.astype(np.uint8)
if _name_ == "_main_":
# kernel for edge detection
kernel = np.array([[-1,-1,-1], [-1,8,-1], [-1,-1,-1]])
# kernel for vertical edge detection
#kernel = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
# kernel for horizontal edge detection
# kernel = np.array([[-1,-1,-1],[0,0,0],[1,1,1]])
# Kernel for box blur
# kernel = np.array([[1/9,1/9,1/9],[1/9,1/9,1/9],[1/9,1/9,1/9]])
# Open the image and convert it to an array
# Try to put your own picture
img = Image.open('/content/Baboon.jpg') or_img = np.asarray(img)
new_img = apply_convolution(img4, kernel)

fig = plt.figure(figsize=(6, 4)) fig.add_subplot(1, 2, 1) plt.imshow(img4) plt.axis("off") plt.title("Original Image")

fig.add_subplot(1, 2, 2) plt.imshow(new_img) plt.axis("off") plt.title("Convoluted Image")
```



For different kernels convolution operation has to be performed

अलग-अलग कर्नेल के लिए कन्वोल्यूशन ऑपरेशन करना पड़ता है

Image padding:

Image padding is an essential technique in image processing which is used to maintain data consistency at the edges of an image. It involves adding layers of pixels to the image, ensuring uniform processing when applying filters and other operations. This prevents

information loss and supports the desired output dimensions, making it essential for precise image analysis and enhancement.

इमेज पैडिंग:

इमेज पैडिंग इमेज प्रोसेसिंग में एक ज़रूरी तकनीक है जिसका इस्तेमाल इमेज के किनारों पर डेटा की एकरूपता बनाए रखने के लिए किया जाता है। इसमें इमेज में पिक्सल की परतें जोड़ना, फ़िल्टर और अन्य ऑपरेशन लागू करते समय एक समान प्रोसेसिंग सुनिश्चित करना शामिल है। यह सूचना हानि को रोकता है और वांछित आउटपुट आयामों का समर्थन करता है, जिससे यह सटीक इमेज विश्लेषण और संवर्द्धन के लिए ज़रूरी हो जाता है।

Zero padding

Zero padding adds black borders around the image; black borders are achieved by adding pixels with intensity value **0** around the image. The thickness of the borders depends on the type of filter kernel used.

ज़ीरो पैडिंग

ज़ीरो पैडिंग इमेज के चारों ओर काली सीमाएँ जोड़ती है; इमेज के चारों ओर तीव्रता मान 0 वाले पिक्सल जोड़कर काली सीमाएँ प्राप्त की जाती हैं। सीमाओं की मोटाई इस्तेमाल किए गए फ़िल्टर कर्नेल के प्रकार पर निर्भर करती है।

0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	0
0	7	8	9	10	11	12	0
0	1	2	3	4	5	6	0
0	7	8	9	10	11	12	0
0	1	2	3	4	5	6	0
0	7	8	9	10	11	12	0
0	0	0	0	0	0	0	0

Zero padding with a one-pixel thick boundary

(एक पिक्सेल मोटी सीमा के साथ शून्य पैडिंग)

It's the simplest form of padding and is often used when we want to maintain the original size of the data while adding a border of uniform values. It defines a function zero padding that adds zero padding to an input array based on the specified padding size. In this case, it applies zero padding with a size of 1 to input array and prints the resulting array with zeros added around its edges.

यह पैडिंग का सबसे सरल रूप है और इसका उपयोग अक्सर तब किया जाता है जब हम समान मानों की सीमा जोड़ते हुए डेटा के मूल आकार को बनाए रखना चाहते हैं। यह एक फ़ंक्शन zero padding को परिभाषित करता है जो निर्दिष्ट पैडिंग आकार के आधार पर इनपुट सरणी में शून्य पैडिंग जोड़ता है। इस मामले में, यह input array पर 1 के आकार के साथ शून्य पैडिंग लागू करता है और परिणामी सरणी को उसके किनारों के चारों ओर शून्य जोड़कर प्रिंट करता है।

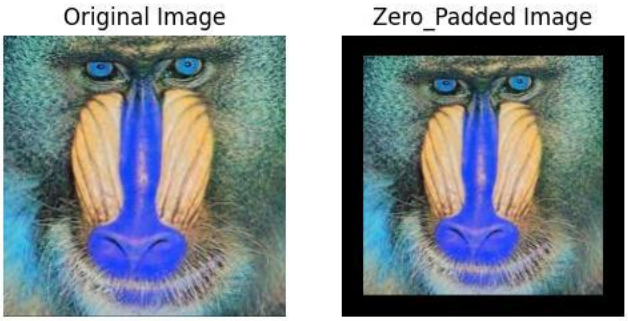
```
##### Zero Padding Operation ##### ####################
```

```
######################## Import libraries #########################
import cv2
import matplotlib.pyplot as plt import numpy as np
################## Zero padding in a 2-D array #######################
image_array = np.array ([[1,  2,  3,  4,  5],
[6,  7,  0,  1,  2],
[3,  4,  5,  6,  7],
[0,  1,  2,  3,  4],
[5,  6,  7,  0,  1]])
def zero_padding(arr, padding_size):
return np.pad(arr, pad_width=padding_size, mode='constant', constant_values=(0))
# Apply zero padding with a padding size of 1 zero_padded_arr = zero_padding(image_array, 1) print("\nZero Padded Array:")
print(zero_padded_arr)
```

```
Zero Padded Array: [[0 0 0 0 0 0 0]
                    [0 1 2 3 4 5 0]
                    [0 6 7 0 1 2 0]
                    [0 3 4 5 6 7 0]
                    [0 0 1 2 3 4 0]
                    [0 5 6 7 0 1 0]
                    [0 0 0 0 0 0 0]]
```

######################## **Zero Padding in an Image** #######################

```
# read image
img1 = cv2.imread('/content/Baboon.jpg') img1 = cv2.resize(img1, (224, 224))
# Make black border
Zero_Padded_img = cv2.copyMakeBorder(img1, 20, 20, 20, 20, cv2.BORDER_CONSTANT, None, value = 0)
fig = plt.figure(figsize=(6, 4)) fig.add_subplot(1, 2, 1) plt.imshow(img1) plt.axis("off") plt.title("Original Image")
fig.add_subplot(1, 2, 2)
plt.imshow(Zero_Padded_img) plt.axis("off")
plt.title("Zero_Padded Image")
```



Mirror padding

Mirror padding, also known as symmetric padding or reflective padding, adds a boundary around the image by mirror-reflecting the image on the original image border. The thickness of the boundary can be adjusted. The pixels near the vertical and horizontal edges act as the line of reflection that adds new boundary pixels directly above/below and left/right of the image. For the remaining boundary pixels, the corner pixels act as the line of reflection

मिरर पैडिंग

मिरर पैडिंग, जिसे सममित पैडिंग या परावर्तक पैडिंग के रूप में भी जाना जाता है, मूल छवि सीमा पर छवि को दर्पण-प्रतिबिंबित करके छवि के चारों ओर एक सीमा जोड़ता है। सीमा की मोटाई को समायोजित किया जा सकता है। ऊर्ध्वाधर

और क्षैतिज किनारों के पास के पिक्सेल प्रतिबिंब की रेखा के रूप में कार्य करते हैं जो छवि के सीधे ऊपर/नीचे और बाएं/दाएं नए सीमा पिक्सेल जोड़ते हैं। शेष सीमा पिक्सेल के लिए, कोने के पिक्सेल प्रतिबिंब की रेखा के रूप में कार्य करते हैं।

Note: The boundary pixel values become the edge over which the values are reflected to get the padding boundary values.

नोट: सीमा पिक्सेल मान वह किनारा बन जाते हैं जिस पर पैडिंग सीमा मान प्राप्त करने के लिए मान प्रतिबिंबित होते हैं।

11	10	9	10	11	12	11	10
7	6	5	6	7	8	7	6
3	2	1	2	3	4	3	2
7	6	5	6	7	8	7	6
11	10	9	10	11	12	11	10
15	14	13	14	15	16	15	14
11	10	9	10	11	12	11	10
7	6	5	6	7	8	7	6

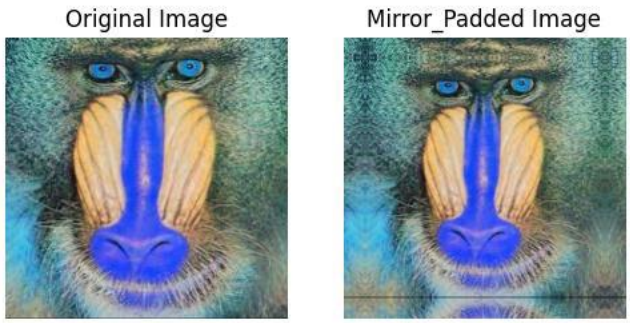
Mirror padding is applicable when the areas near the border contain important image details, and we want to extend those details. It reflects or mirrors the values at the edges to fill the padded regions. This helps in maintaining the continuity of patterns and textures at the borders. Mirror padding is useful when the border regions have intricate patterns or features that we want to extend without creating abrupt changes or artifacts.

मिरर पैडिंग तब उपयोगी होती है जब बॉर्डर के पास के क्षेत्रों में महत्वपूर्ण छवि विवरण होते हैं, और हम उन विवरणों को विस्तारित करना चाहते हैं। यह पैड किए गए क्षेत्रों को भरने के लिए किनारों पर मूल्यों को प्रतिबिंबित या प्रतिबिम्बित करता है। यह बॉर्डर पर पैटर्न और बनावट की निरंतरता को बनाए रखने में मदद करता है। मिरर पैडिंग तब उपयोगी होती है जब बॉर्डर क्षेत्रों में जटिल पैटर्न या विशेषताएं होती हैं जिन्हें हम अचानक परिवर्तन या कलाकृतियों को बनाए बिना विस्तारित करना चाहते हैं।

```
##################Mirror padding in 2-D array #####################
import numpy as np
image_array = np.array([[1, 2, 5, 9, 6],
                        [4, 7, 0, 1, 2],
                        [3, 4, 5, 6, 7],
                        [0, 1, 2, 3, 4]])
def mirror_padding(arr, padding_size):
return np.pad(arr, pad_width=padding_size, mode='reflect')
# Apply mirror padding with a padding size of 1
mirror_padded_arr = mirror_padding(image_array, 2)
print("\nMirror Padded Array:")
print(mirror_padded_arr)
```

```
Mirror Padded Array:
[[5 4 3 4 5 6 7 6 5]
[0 7 4 7 0 1 2 1 0]
[5 2 1 2 5 9 6 9 5]
[0 7 4 7 0 1 2 1 0]
[5 4 3 4 5 6 7 6 5]
[2 1 0 1 2 3 4 3 2]
[5 4 3 4 5 6 7 6 5]
[0 7 4 7 0 1 2 1 0]]
```

```
######################### Mirror Padding in an Image
#######################
# read image
img1 = cv2.imread('/content/Baboon.jpg')
img1 = cv2.resize(img1, (224, 224))
# Make black border
Mirror_Pad_img = cv2.copyMakeBorder(img1, 20, 20, 20, 20,
cv2.BORDER_REFLECT)

fig = plt.figure(figsize=(6, 4))
fig.add_subplot(1, 2, 1)
plt.imshow(img1)
```



Replicate padding:

Replicate padding adds the closest values outside the boundary, ensuring that values outside the boundary are set equal to the nearest image border value. Replicate padding is used when we want to preserve the values at the edges of our data while extending it. It effectively replicates the values of the nearest edge pixel to fill the padded regions. This type of padding is suitable when the areas near the border of the image or data have a relatively constant or uniform pattern that we want to maintain. For example, if we have an image

with a solid colour border, replicate padding would be a good choice to extend that solid colour to the padded regions, preserving the appearance of a continuous border.

प्रतिकृति पैडिंग:

प्रतिकृति पैडिंग सीमा के बाहर निकटतम मान जोड़ती है, यह सुनिश्चित करते हुए कि सीमा के बाहर के मान निकटतम छवि सीमा मान के बराबर सेट किए गए हैं। प्रतिकृति पैडिंग का उपयोग तब किया जाता है जब हम अपने डेटा को विस्तारित करते समय उसके किनारों पर मानों को संरक्षित करना चाहते हैं। यह पैड किए गए क्षेत्रों को भरने के लिए निकटतम किनारे पिक्सेल के मानों को प्रभावी ढंग से दोहराता है। इस प्रकार की पैडिंग तब उपयुक्त होती है जब छवि या डेटा की सीमा के पास के क्षेत्रों में अपेक्षाकृत स्थिर या समान पैटर्न होता है जिसे हम बनाए रखना चाहते हैं। उदाहरण के लिए, यदि हमारे पास एक ठोस रंग की सीमा वाली छवि है, तो प्रतिकृति पैडिंग उस ठोस रंग को पैड किए गए क्षेत्रों तक विस्तारित करने के लिए एक अच्छा विकल्प होगा, जो एक निरंतर सीमा की उपस्थिति को संरक्षित करता है।

1	1	1	2	3	4	4	4
1	1	1	2	3	4	4	4
1	1	1	2	3	4	4	4
5	5	5	6	7	8	8	8
1	1	1	2	3	4	4	4
5	5	5	6	7	8	8	8
5	5	5	8	9	8	8	8
5	5	5	8	9	8	8	8

################## Replicate padding #######################

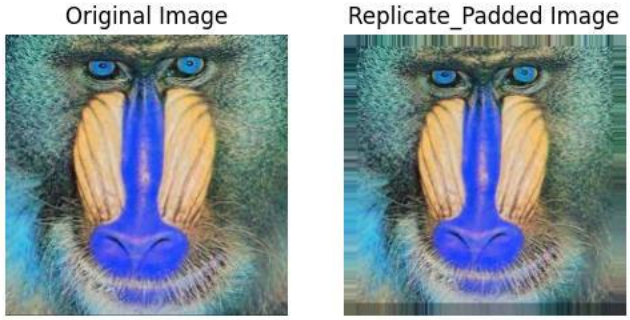
```
import numpy as np
image_array = np.array([[1, 2, 3, 4, 5],
                        [6, 7, 0, 1, 2],
                        [3, 4, 5, 6, 7],
                        [0, 1, 2, 3, 4]])
```

```
def replicate_padding(arr, padding_size):
return np.pad(arr, pad_width=padding_size, mode='edge')
# Apply replicate padding with a padding size of 1
replicate_padded_arr = replicate_padding(image_array, 1)
print("\nReplicate Padded Array:")
print(replicate_padded_arr)
Replicate Padded Array:
[[1 1 2 3 4 5 5]
```

```
[1 1 2 3 4 5 5]
[6 6 7 0 1 2 2]
[3 3 4 5 6 7 7]
[0 0 1 2 3 4 4]
[0 0 1 2 3 4 4]]
```

################### Replicate Padding in an Image ##########

```
# read image#####
img1 = cv2.imread('/content/Baboon.jpg')
img1 = cv2.resize(img1, (224, 224))
# Make black border###
Repli_pad_img = cv2.copyMakeBorder(img1, 10, 10, 10, 10, cv2.BORDER_REPLICATE)
fig = plt.figure(figsize=(6, 4))
fig.add_subplot(1, 2, 1)
plt.imshow(img1)
plt.axis("off")
plt.title("Original Image")
fig.add_subplot(1, 2, 2)
plt.imshow(Repli_pad_img)
plt.axis("off")
plt.title("Replicate_Padded Image")
```



Digital Image Processing Lab
Experiment No-6
डिजिटल इमेज प्रोसेसिंग लैब
प्रयोग संख्या-6

Aim: To study and perform non-linear neighborhood operations on images:

A. Min
B. Max
C. Median
D. With various padding operations (zero, 1's, line, mirror)

उद्देश्य: छवियों पर गैर-रैखिक पड़ोस संचालन का अध्ययन और प्रदर्शन करना:

(ए) न्यूनतम

(बी) अधिकतम

(सी) माध्यिका

(डी) विभिन्न पैडिंग संचालन (शून्य, 1, रेखा, दर्पण) के साथ

Theory:

Non-Linear Filter

Using some non-linear function from the source pixel value. The idea is to replace the target pixel value with its neighbour pixels value from some ordering mechanism or function**.**

There are many types of Non-Linear Filter but in this article, I will show you just 3 of them

- Minimum Filter
- Maximum Filter
- Median Filter

Minimum and maximum are the simplest non-linear operators. The former scans each pixel in an image and replaces it with the lowest ranking value in its neighborhood. Maximum does the same, except that the replacing value is the highest-ranking pixel in the neighborhood.

सिद्धांत:

गैर-रैखिक फ़िल्टर

स्रोत पिक्सेल मान से कुछ गैर-रैखिक फ़ंक्शन का उपयोग करना। विचार लक्ष्य पिक्सेल मान को किसी ऑर्डरिंग मैकेनिज़्म या फ़ंक्शन से उसके पड़ोसी पिक्सेल मान से बदलना है।

गैर-रैखिक फ़िल्टर के कई प्रकार हैं लेकिन इस लेख में, मैं आपको उनमें से सिर्फ़ 3 दिखाऊँगा

• न्यूनतम फ़िल्टर

• अधिकतम फ़िल्टर

• माध्य फ़िल्टर

न्यूनतम और अधिकतम सबसे सरल गैर-रैखिक ऑपरेटर हैं। पहला एक छवि में प्रत्येक पिक्सेल को स्कैन करता है और इसे उसके पड़ोस में सबसे कम रैंकिंग वाले मान से बदल देता है। अधिकतम भी यही करता है, सिवाय इसके कि प्रतिस्थापित मान पड़ोस में सबसे उच्च रैंकिंग वाला पिक्सेल होता है।

Minimum Filter

This algorithm is to select the lowest pixel value from the neighbours' pixels around the target then replace it**.**

The Procedure of minimum filter:

- The window is overlaid on the upper left corner of the image, and the minimum value is determined by sorting the pixels values (ascending order).
- This value (minimum) is put into the output image corresponding to the center location of the window
- The window is then sliding one pixel over, and the process is repeated when the end of the row is reached, the window is slide back to the left side of the image and down one row, and the process is repeated
- This process continues until the entire image has been processed**.**

.

न्यूनतम फ़िल्टर

इस एल्गोरिथ्म में लक्ष्य के आस-पास के पड़ोसी पिक्सेल से सबसे कम पिक्सेल मान का चयन करना और फिर उसे प्रतिस्थापित करना शामिल है।

न्यूनतम फ़िल्टर की प्रक्रिया:

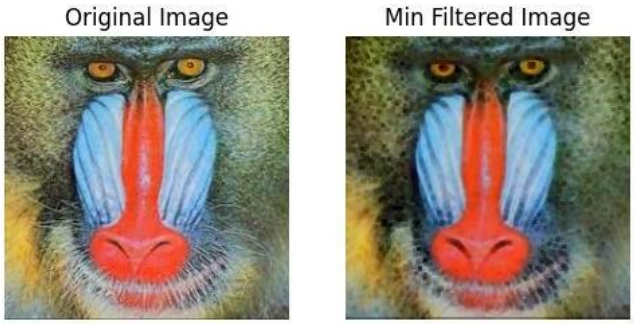
- विंडो को छवि के ऊपरी बाएँ कोने पर ओवरले किया जाता है, और पिक्सेल मानों (आरोही क्रम) को छाँटकर न्यूनतम मान निर्धारित किया जाता है।
- यह मान (न्यूनतम) विंडो के केंद्र स्थान के अनुरूप आउटपुट छवि में डाला जाता है।
- फिर विंडो एक पिक्सेल ऊपर खिसक जाती है, और पंक्ति के अंत तक पहुँचने पर प्रक्रिया दोहराई जाती है, विंडो को छवि के बाईं ओर और एक पंक्ति नीचे वापस खिसकाया जाता है, और प्रक्रिया दोहराई जाती है।
- यह प्रक्रिया तब तक जारी रहती है जब तक कि पूरी छवि संसाधित नहीं हो जाती।

Note:

the outer rows and columns are not replaced. And these “wasted” rows and columns are often filled with zeros (or cropped off the image). For example, with 3X3 mask, we lose one outer row and column, a 5X5 mask we lose two rows and columns

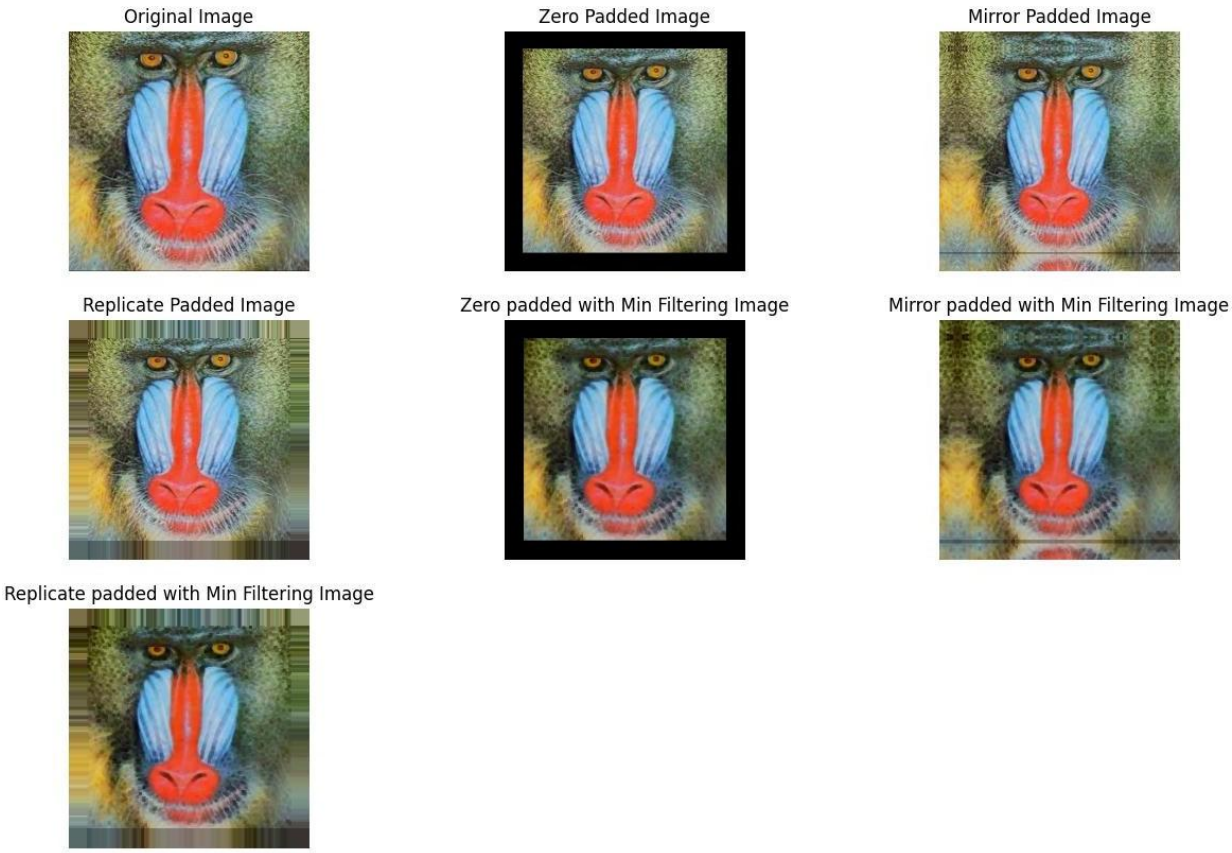
नोट: बाहरी पंक्तियों और स्तंभों को प्रतिस्थापित नहीं किया जाता है। और ये "बर्बाद" पंक्तियाँ और स्तंभ अक्सर शून्य से भरे होते हैं (या छवि से काट दिए जाते हैं)। उदाहरण के लिए, 3X3 मास्क के साथ, हम एक बाहरी पंक्ति और स्तंभ खो देते हैं, 5X5 मास्क के साथ हम दो पंक्तियाँ और स्तंभ खो देते हैं

```
############# Import Important Libraries ####### from
PIL import Image, ImageFilter
import matplotlib.pyplot as plt

#### read image using pillow library img1 =
Image.open("/content/Baboon.jpg") ####
resize image to 224x224
img1 = img1.resize((224, 224))

#### Apply min filtering
new_image1 = img1.filter(ImageFilter.MinFilter(size = 3))

#### Plot original image and min filtering
Fig = plt.figure(figsize=(6, 4))
  fig.add_subplot(1,2, 1)
   plt.imshow(img1)
plt.axis("off")
   fig.add_subplot(1, 2, 2)
plt.imshow(new_image1)
plt.axis("off")
plt.title("Min Filtered Image")
```



```
############## Minimum filtering operation with different padding##
############# Import Important Libraries #######
import cv2from PIL import Image, ImageFilter import matplotlib.pyplot as plt

#### read image using pillow library
img1 = cv2.imread("/content/Baboon.jpg")
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
#### resize image to 224x224
img1 = cv2.resize(img1, (224, 224))

## apply padding (Zero padding, Mirror padding,and Replicate paddding)##
img1_pad1 = cv2.copyMakeBorder(img1,20, 20, 20, 20, cv2.BORDER_CONSTANT,
None, value=0)
img1_pad2 = cv2.copyMakeBorder(img1,20, 20, 20, 20, cv2.BORDER_REFLECT)
img1_pad3 = cv2.copyMakeBorder(img1,20, 20, 20, 20, cv2.BORDER_REPLICATE)

########### converet array as image for the respective padded images
img1_new1 = Image.fromarray(img1_pad1)
img1_new2 = Image.fromarray(img1_pad2) img1_new3 =
Image.fromarray(img1_pad3)

### apply min filtering##
result1 = img1_new1.filter(ImageFilter.MinFilter(size = 3)) result2 =
img1_new2.filter(ImageFilter.MinFilter(size = 3)) result3 =
img1_new3.filter(ImageFilter.MinFilter(size = 3))

#### Plot all the results##
fig = plt.figure(figsize=(15, 10))
fig.add_subplot(3, 3, 1)
plt.imshow(img1)
plt.axis("off")
plt.title("Original Image")
fig.add_subplot(3, 3, 2)
plt.imshow(img1_pad1)
plt.axis("off")
```

```
plt.title("Zero Padde Image")
fig.add_subplot(3, 3, 3)
plt.imshow(img1_pad2)
plt.axis("off")
plt.title("Mirror Padde Image")
fig.add_subplot(3, 3, 4)
plt.imshow(img1_pad3)
plt.axis("off") plt.title("Replicate Padded Image")
fig.add_subplot(3, 3, 5)
plt.imshow(result1)
plt.axis("off")
plt.title("Zero padded with Min Filtering Image")
fig.add_subplot(3, 3, 6)
plt.imshow(result2)
plt.axis("off")
plt.title("Mirror padded with Min Filtering Image")
fig.add_subplot(3, 3, 7)
plt.imshow(result3)
plt.axis("off")
plt.title (“Replicate padded with mini filtering image”)
```



Maximum Filter

This algorithm also similar to minimum filter but pick the highest one.

The Procedure of MAXIMUM filter:

- The window is overlaid on the upper left corner of the image, and the maximum value is determined by sorting the pixels values (ascending order).
- This value (maximum) is put into the output image corresponding to the center location of the window.
- The window is then sliding one pixel over, and the process is repeated when the end of the row is reached, the window is slide back to the left side of the image and down one row, and the process is repeated.
- This process continues until the entire image has been processed.

अधिकतम फ़िल्टर

यह एल्गोरिथ्म भी न्यूनतम फ़िल्टर के समान है, लेकिन उच्चतम फ़िल्टर चुनें।

अधिकतम फ़िल्टर की प्रक्रिया:

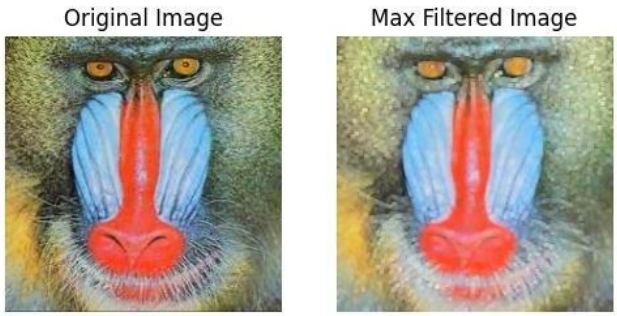
- विंडो को छवि के ऊपरी बाएँ कोने पर ओवरले किया जाता है, और पिक्सेल मानों (आरोही क्रम) को सॉर्ट करके अधिकतम मान निर्धारित किया जाता है।
- यह मान (अधिकतम) विंडो के केंद्र स्थान के अनुरूप आउटपुट छवि में डाला जाता है।

- फिर विंडो एक पिक्सेल ऊपर खिसक जाती है, और पंक्ति के अंत तक पहुँचने पर प्रक्रिया दोहराई जाती है, विंडो को छवि के बाईं ओर और एक पंक्ति नीचे वापस खिसकाया जाता है, और प्रक्रिया दोहराई जाती है।
- यह प्रक्रिया तब तक जारी रहती है जब तक कि पूरी छवि संसाधित नहीं हो जाती।

```
#### import the necesarry libraries #######
import matplotlib.pyplot as plt from PIL import Image, ImageFilter
```

############## Maximum filtering operation with different padding ###################

```
######## Read Image
##### # creating a image #
img1 = Image.open(r"/content/Baboon.jpg")
img1 = img1.resize((224, 224))

new_image = img1.filter(ImageFilter.MaxFilter(size = 3))

fig = plt.figure(figsize=(6, 4))

fig.add_subplot(1, 2, 1)
plt.imshow(img1) plt.axis("off")
plt.title("Original Image")


fig.add_subplot(1, 2, 2)
plt.imshow(new_image)
plt.axis("off")
plt.title("Max Filtered Image")
```

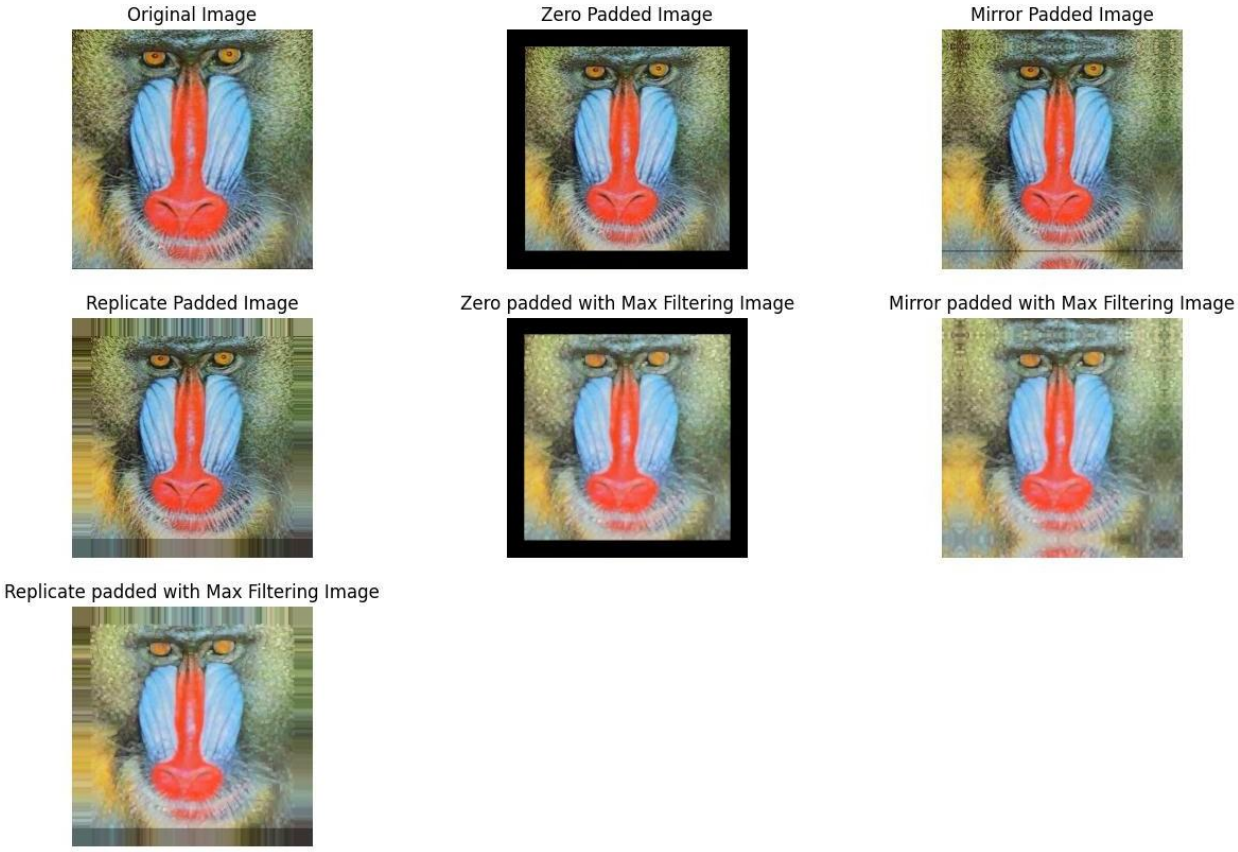


##########Maximum filtering operation with different padding ######

############# Import Important Libraries #######

```
import cv2
from PIL import Image, ImageFilter
import matplotlib.pyplot as plt
#### read image using pillow library
img1 = cv2.imread("/content/Baboon.jpg")
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
#### resize image to 224x224
img1 = cv2.resize(img1, (224, 224))
######### apply padding (Zero padding, Mirror padding, and Replicate paddding)
img1_pad1 = cv2.copyMakeBorder(img1,20, 20, 20, 20, cv2.BORDER_CONSTANT, None, value=0)
img1_pad2 = cv2.copyMakeBorder(img1,20, 20, 20, 20, cv2.BORDER_REFLECT)
img1_pad3 = cv2.copyMakeBorder(img1,20, 20, 20, 20,20 cv2.BORDER_REPLATE)
########### converet array as image for the respective padded images img1_new1 = Image.fromarray(img1_pad1)
img1_new2 = Image.fromarray(img1_pad2) img1_new3 = Image.fromarray(img1_pad3)

### apply min filtering
result1 = img1_new1.filter(ImageFilter.MaxFilter(size = 3)) result2 = img1_new2.filter(ImageFilter.MaxFilter(size = 3)) result3 = img1_new3.filter(ImageFilter.MaxFilter(size = 3))

#### Plot all the results
fig = plt.figure(figsize=(15, 10)) fig.add_subplot(3, 3, 1) plt.imshow(img1) plt.axis("off") plt.title("Original Image")

fig.add_subplot(3, 3, 2) plt.imshow(img1_pad1) plt.axis("off") plt.title("Zero Padded Image")
```

```
fig.add_subplot(3, 3, 3) plt.imshow(img1_pad2) plt.axis("off") plt.title("Mirror Padded Image")
fig.add_subplot(3, 3, 4) plt.imshow(img1_pad3) plt.axis("off") plt.title("Replicate Padded Image")
fig.add_subplot(3, 3, 5) plt.imshow(result1) plt.axis("off")
plt.title("Zero padded with Max Filtering Image")

fig.add_subplot(3, 3, 6) plt.imshow(result2) plt.axis("off")
plt.title("Mirror padded with Max Filtering Image")

fig.add_subplot(3, 3, 7)
plt.imshow(result3) plt.axis("off")
plt.title("Replicate padded with Max Filtering Image")
```



Median Filter

The median filter is actually a specific form of a rank filter, where the ith pixel intensity in the sorted list of neighborhood pixels is chosen as the output.

The Procedure of median filter:

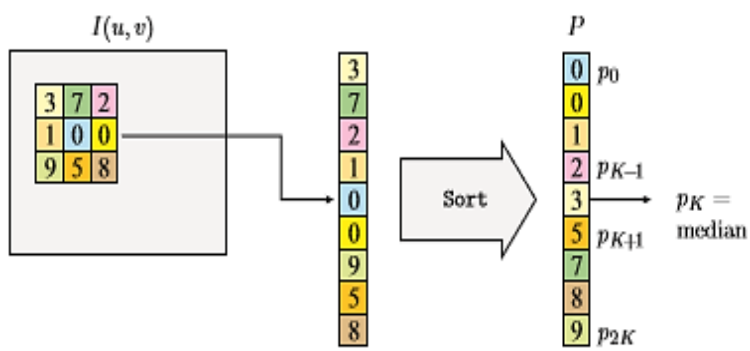
- The window is overlaid on the upper left corner of the image, and the median value is determined.

- This value (median) is put into the output image (buffer) corresponding to the center location of the window.
- The window is then sliding one pixel over, and the process is repeated when the end of the row is reached, the window is slide back to the left side of the image and down one row, and the process is repeated.
- This process continues until the entire image has been processed.

मीडियन फ़िल्टर

मीडियन फ़िल्टर वास्तव में रैंक फ़िल्टर का एक विशिष्ट रूप है, जहाँ पड़ोस पिक्सेल की सॉर्ट की गई सूची में ith पिक्सेल **तीव्रता को आउटपुट के रूप में चुना जाता है।**

- मीडियन फ़िल्टर की प्रक्रिया:
- विंडो को छवि के ऊपरी बाएँ कोने पर ओवरले किया जाता है, और मीडियन मान निर्धारित किया जाता है।
- यह मान (मीडियन) विंडो के केंद्र स्थान के अनुरूप आउटपुट छवि (बफर) में डाला जाता है।
- फिर विंडो एक पिक्सेल ऊपर खिसकती है, और पंक्ति के अंत तक पहुँचने पर प्रक्रिया दोहराई जाती है, विंडो को छवि के बाईं ओर और एक पंक्ति नीचे वापस खिसकाया जाता है, और प्रक्रिया दोहराई जाती है।
- यह प्रक्रिया तब तक जारी रहती है जब तक कि पूरी छवि संसाधित नहीं हो जाती।



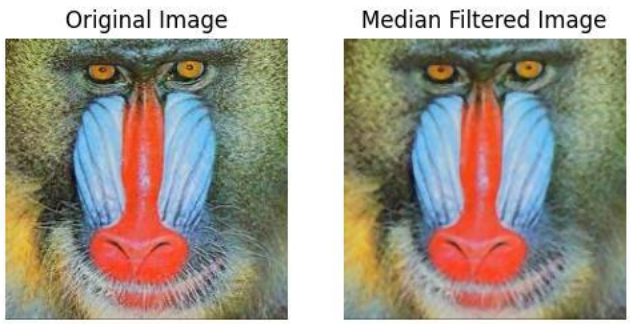
For more clarity of the advantage of this filter, here is the example that used in salt and pepper pictures.

इस फिल्टर के लाभ की अधिक स्पष्टता के लिए, यहां नमक और काली मिर्च के चित्रों में उपयोग किया गया उदाहरण दिया गया है।

Median Filtering
import the necesarry libraries

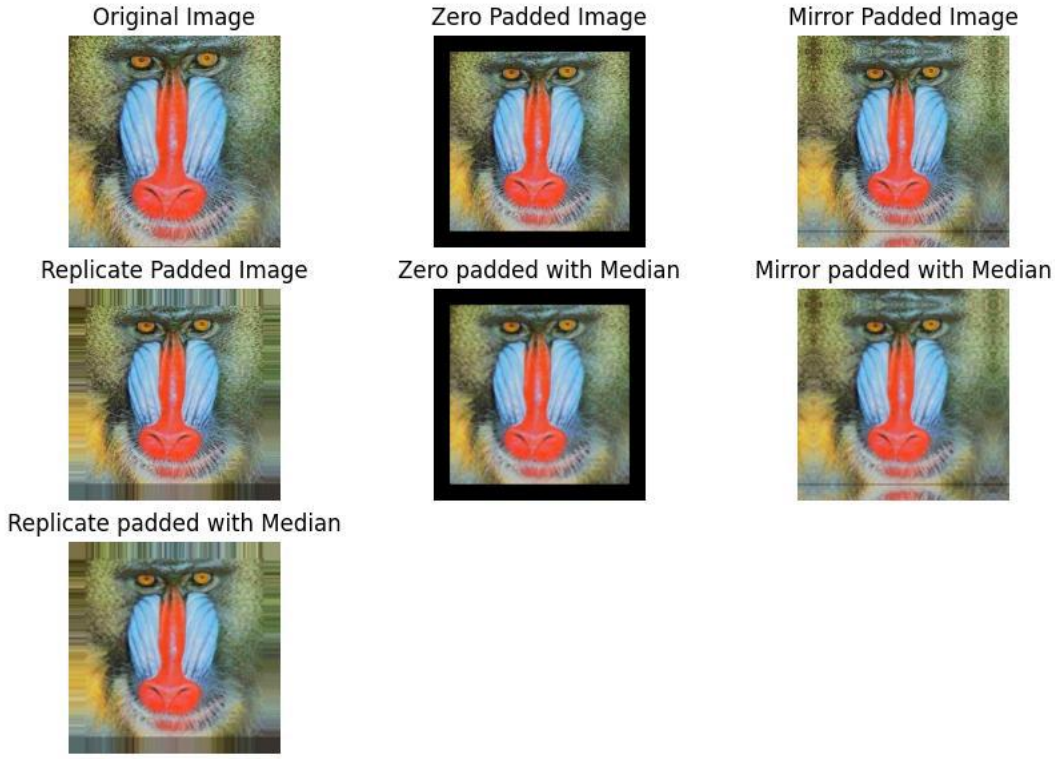
import matplotlib.pyplot as plt from PIL import Image, ImageFilter ######## Read Image #####
# creating a image object
img1 = Image.open(r"/content/Baboon.jpg") img1 = img1.resize((224, 224))
new_image = img1.filter(ImageFilter.MedianFilter(size = 3)) fig = plt.figure(figsize=(6, 4))
fig.add_subplot(1, 2, 1) plt.imshow(img1) plt.axis("off") plt.title("Original Image")

fig.add_subplot(1, 2, 2) plt.imshow(new_image) plt.axis("off")
plt.title("Median Filtered Image")



############# Import Important Libraries #######
import cv2
from PIL import Image, ImageFilter
import matplotlib.pyplot as plt
read image using pillow library
img1 = cv2.imread("/content/Baboon.jpg")
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
resize image to 224x224
img1 = cv2.resize(img1, (224, 224))
######### apply padding (Zero padding, Mirror padding, and Replicate paddding)
img1_pad1 = cv2.copyMakeBorder(img1,20, 20, 20, 20, cv2.BORDER_CONSTANT, None, value=0)
img1_pad2 = cv2.copyMakeBorder(img1,20, 20, 20, 20, cv2.BORDER_REFLECT)
img1_pad3 = cv2.copyMakeBorder(img1,20, 20, 20, 20, cv2.BORDER_REPLICATE)
########### converet array as image for the respective padded images
img1_new1 = Image.fromarray(img1_pad1)
img1_new2 = Image.fromarray(img1_pad2)
img1_new3 = Image.fromarray(img1_pad3)
apply min filtering
result1 = img1_new1.filter(ImageFilter.MedianFilter(size = 3))

```
result2 = img1_new2.filter(ImageFilter.MedianFilter(size = 3))
result3 = img1_new3.filter(ImageFilter.MedianFilter(size = 3))
#### Plot all the results
fig = plt.figure(figsize=(15, 10))
fig.add_subplot(3, 3, 1)
plt.imshow(img1)
plt.axis("off")
plt.title("Original Image")
fig.add_subplot(3, 3, 2)
plt.imshow(img1_pad1)
plt.axis("off")
plt.title("Zero Padded Image")
fig.add_subplot(3, 3, 3)
plt.imshow(img1_pad2)
plt.axis("off")
plt.title("Mirror Padded Image")
fig.add_subplot(3, 3, 4)
plt.imshow(img1_pad3)
plt.axis("off")
plt.title("Replicate Padded Image")
fig.add_subplot(3, 3, 5)
plt.imshow(result1)
plt.axis("off")
plt.title("Zero padded with Median Filtering Image")
fig.add_subplot(3, 3, 6)
plt.imshow(result2)
plt.axis("off")
plt.title("Mirror padded with Median Filtering Image")
fig.add_subplot(3, 3, 7)
plt.imshow(result3)
plt.axis("off")
plt.title("Replicate padded with Median Filtering Image")
```



The median filter is a nonlinear filter (order filter) used to remove noise from images.

❖ The median filter is also used to preserve edge properties while reducing the noise.

❖ These filters are based on as specific type of image statistics called order statistics.

❖ Typically, these filters operate on small sub image, “Window”, and replace the center pixel value (similar to the convolution process).

Digital Image Processing Lab
Experiment No-7
जिटल इमेज प्रोसेसिंग लैब
प्रयोग संख्या-7

Aim:
To study and perform spatial domain image denoising using:
Linear filters for Impulse and Gaussian noise of various densities.
Non-linear filters for Impulse and Gaussian noise of various densities.

उद्देश्य:

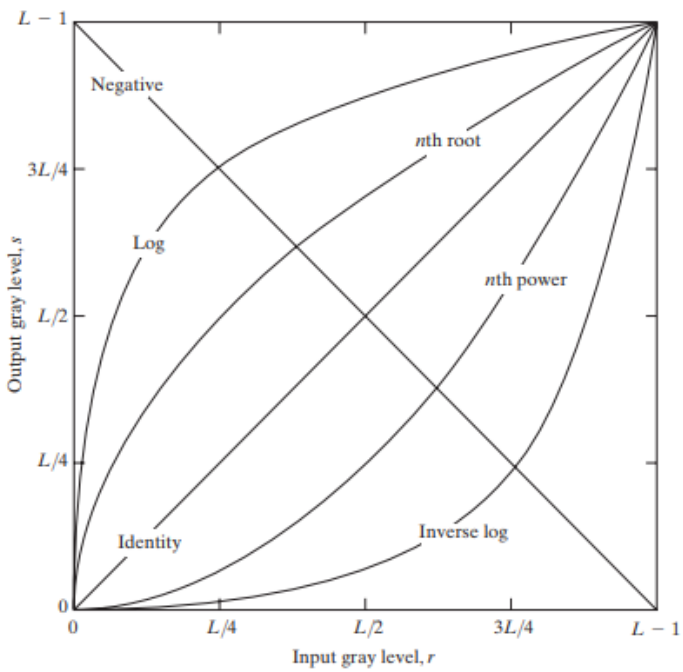
स्थानिक डोमेन छवि को शोरमुक्त करने के लिए अध्ययन और प्रदर्शन करने के लिए:
विभिन्न घनत्वों के आवेग और गौसियन शोर के लिए रैखिक फ़िल्टर।
विभिन्न घनत्वों के आवेग और गौसियन शोर के लिए गैर-रैखिक फ़िल्टर।

Theory:
Intensity transformations are applied on images for contrast manipulation or image thresholding. These are in the spatial domain, i.e. they are performed directly on the pixels of the image at hand, as opposed to being performed on the Fourier transform of the image. The following are commonly used intensity transformations:
1. Image Negatives (Linear)
2. Log Transformations
3. Power-Law (Gamma) Transformations
4. Piecewise-Linear Transformation Functions

सिद्धांत:
तीव्रता परिवर्तन छवियों पर कंट्रास्ट हेरफेर या छवि थ्रेशोल्डिंग के लिए लागू किए जाते हैं। ये स्थानिक डोमेन में हैं, यानी वे छवि के फूरियर रूपांतरण पर किए जाने के विपरीत, हाथ में मौजूद छवि के पिक्सेल पर सीधे किए जाते हैं। निम्नलिखित आमतौर पर इस्तेमाल किए जाने वाले तीव्रता परिवर्तन हैं:
1. इमेज नेगेटिव (रैखिक)
2. लॉग ट्रांसफॉर्मेशन
3. पावर-लॉ (गामा) ट्रांसफॉर्मेशन
4. टुकड़ा-रैखिक परिवर्तन फ़ंक्शन

Spatial Domain Processes – Spatial domain processes can be described using the equation:g(x, y) = T[f(x, y)]wheref(x, y)is the input image, T is an operator on f defined over a neighbourhood of the point (x, y), andg(x, y)is the output

स्थानिक डोमेन प्रक्रियाएँ - स्थानिक डोमेन प्रक्रियाओं को समीकरण का उपयोग करके वर्णित किया जा सकता है: g(x, y) = T[f(x, y)]जहाँ f(x, y)इनपुट छवि है, T बिंदु (x, y) के पड़ोस पर परिभाषित f पर एक ऑपरेटर है, और g(x, y)आउटपुट है



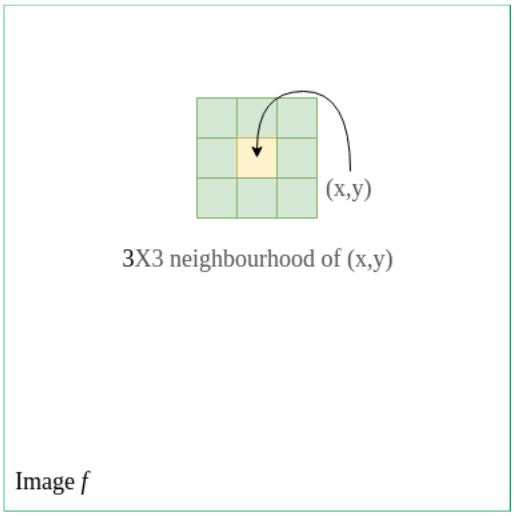


Image Negatives – Image negatives are discussed in this article. Mathematically, assume that an image goes from intensity levels 0 to (L-1). Generally, L = 256. Then, the negative transformation can be described by the expression $s = L-1-r$ where r is the initial intensity level and s is the final intensity level of a pixel. This produces a photographic negative.

इमेज नेगेटिव - इस लेख में इमेज नेगेटिव पर चर्चा की गई है। गणितीय रूप से, मान लें कि एक छवि तीव्रता स्तर 0 से (L-1) तक जाती है। आम तौर पर, L = 256। फिर, नकारात्मक परिवर्तन को अभिव्यक्ति s = L-1-r द्वारा वर्णित किया जा सकता है जहाँ r प्रारंभिक तीव्रता स्तर है और s पिक्सेल का अंतिम तीव्रता स्तर है। यह एक फोटोग्राफिक नकारात्मक उत्पन्न करता है।

Log Transformations –

Mathematically, log transformations can be expressed as s = clog(1+r). Here, s is the output intensity, r>=0 is the input intensity of the pixel, and c is a scaling constant. c is given by 255/(log (1 + m)), where m is the maximum pixel value in the image. It is done to ensure that the final pixel value does not exceed (L-1), or 255. Practically, log transformation maps a narrow range of low-intensity input values to a wide range of output values. Consider the following input image

लॉग ट्रांसफ़ॉर्मेशन -

गणितीय रूप से, लॉग ट्रांसफ़ॉर्मेशन को s = clog(1+r) के रूप में व्यक्त किया जा सकता है। यहाँ, s आउटपुट तीव्रता है, r>=0 पिक्सेल की इनपुट तीव्रता है, और c एक स्केलिंग स्थिरांक है। c 255/(log(1 + m)) द्वारा दिया जाता है, जहाँ m छवि में अधिकतम पिक्सेल मान है। यह सुनिश्चित करने के लिए किया जाता है कि अंतिम पिक्सेल मान

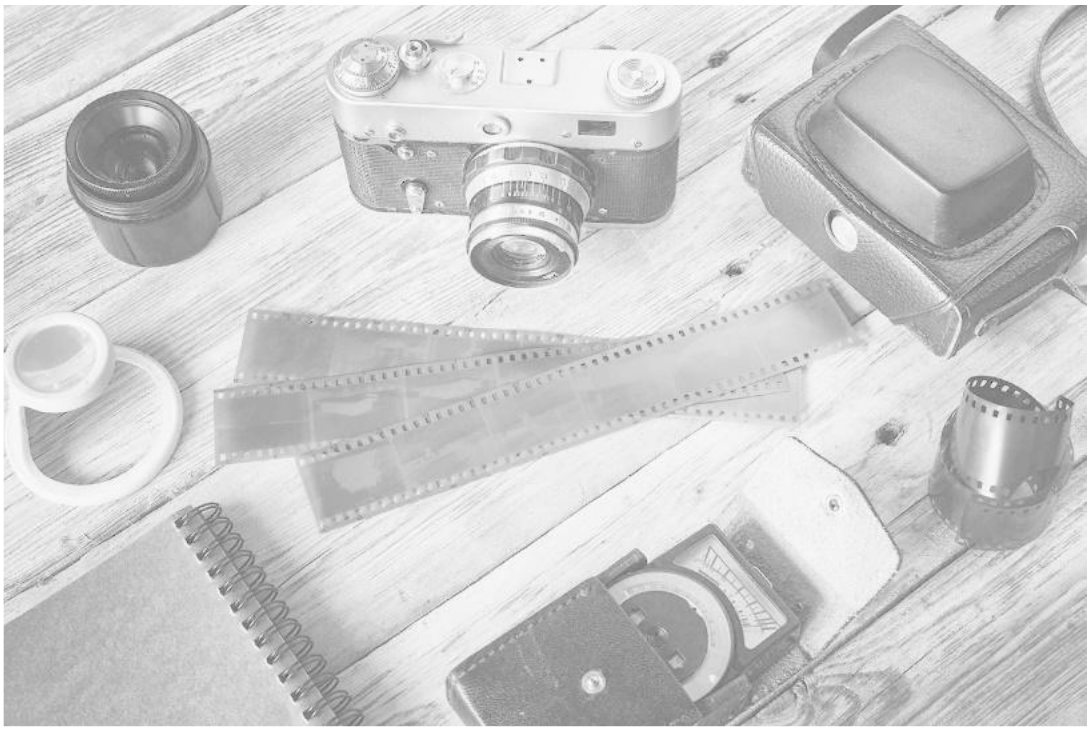
(L-1) या 255 से अधिक न हो। व्यावहारिक रूप से, लॉग ट्रांसफ़ॉर्मेशन कम तीव्रता वाले इनपुट मानों की एक संकीर्ण श्रेणी को आउटपुट मानों की एक विस्तृत श्रेणी में मैप करता है। निम्नलिखित इनपुट छवि पर विचार करें

Below is the code to apply log transformation to the image.

```
import cv2
import numpy as np
# Open the image.
img = cv2.imread('sample.jpg')
```

```
# Apply log transform.
c = 255/(np.log(1 + np.max(img)))
log_transformed = c * np.log(1 + img)
# Specify the data type.
log_transformed = np.array(log_transformed, dtype = np.uint8)
# Save the output.
cv2.imwrite('log_transformed.jpg', log_transformed)
```

Below is the log-transformed output



Power-Law (Gamma) Transformation –

Power-law (gamma) transformations can be mathematically expressed as . Gamma correction is important for displaying images on a screen correctly, to prevent bleaching or darkening of images when viewed from different types of monitors with different display settings. This is done because our eyes perceive images in a gamma-shaped curve, whereas cameras capture images in a linear fashion. Below is the Python code to apply gamma correction.

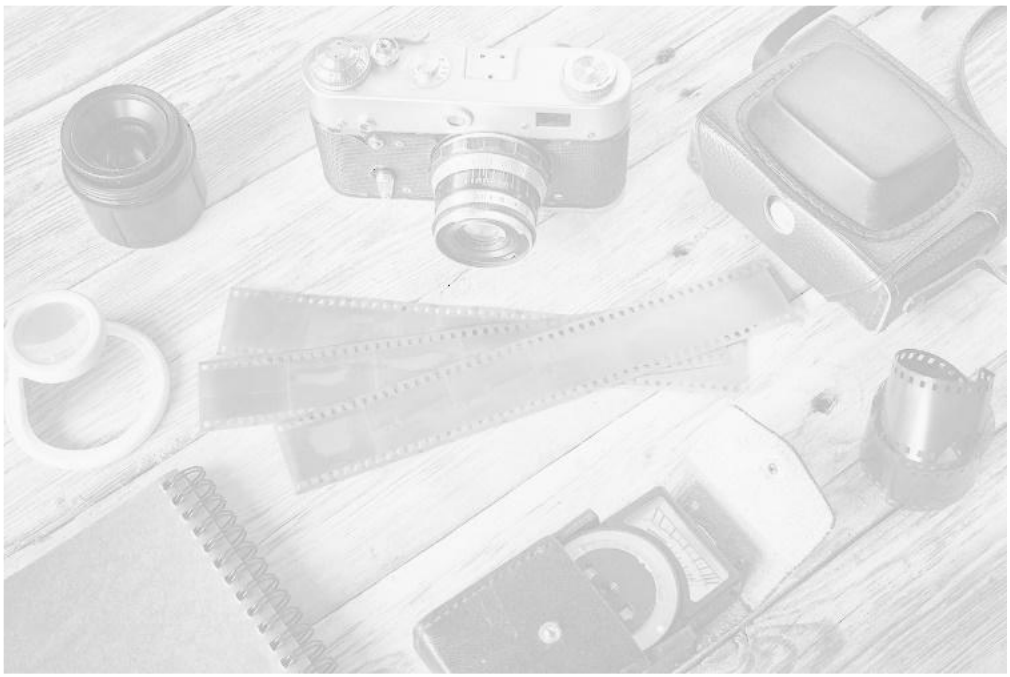
पावर-लॉ (गामा) परिवर्तन -

पावर-लॉ (गामा) परिवर्तन को गणितीय रूप से इस प्रकार व्यक्त किया जा सकता है। गामा सुधार स्क्रीन पर छवियों को सही ढंग से प्रदर्शित करने के लिए महत्वपूर्ण है, ताकि विभिन्न डिस्प्ले सेटिंग्स वाले विभिन्न प्रकार के मॉनिटर से देखने पर छवियों के विरंजन या कालेपन को रोका जा सके। ऐसा इसलिए किया जाता है क्योंकि हमारी आँखें छवियों को गामा के आकार के वक्र में देखती हैं, जबकि कैमरे छवियों को रैखिक तरीके से कैप्चर करते हैं। नीचे गामा सुधार लागू करने के लिए पायथन कोड दिया गया है

```
import cv2
import numpy as np
# Open the image.
img = cv2.imread('sample.jpg')
# Trying 4 gamma values.
for gamma in [0.1, 0.5, 1.2, 2.2]:
# Apply gamma correction.
gamma_corrected = np.array(255*(img / 255) ** gamma, dtype = 'uint8')
# Save edited images.
cv2.imwrite('gamma_transformed'+str(gamma)+'.jpg', gamma_corrected)
```

Below are the gamma-corrected outputs for different values of gamma.

Gamma = 0.1:



Gamma = 0.5

Gamma = 1.2:

Gamma = 2.2: As can be observed from the outputs as well as the graph, gamma>1 (indicated by the curve corresponding to ‘nth power’ label on the graph), the intensity of pixels decreases i.e. the image becomes darker. On the other hand, gamma<1 (indicated by the curve corresponding to 'nth root' label on the graph), the intensity increases i.e. the image becomes lighter

गामा = 2.2: जैसा कि आउटपुट और ग्राफ से देखा जा सकता है, गामा>1 (ग्राफ पर 'एनवें पावर' लेबल के अनुरूप वक्र द्वारा दर्शाया गया है), पिक्सल की तीव्रता कम हो जाती है यानी छवि गहरी हो जाती है। दूसरी ओर, गामा<1 (ग्राफ पर 'एनवें रूट' लेबल के अनुरूप वक्र द्वारा दर्शाया गया है), तीव्रता बढ़ जाती है यानी छवि हल्की हो जाती है

Piecewise-Linear Transformation Functions –

These functions, as the name suggests, are not entirely linear in nature. However, they are linear between certain x-intervals. One of the most commonly used piecewise-linear transformation functions is contrast stretching. Contrast can be defined as:

Contrast = (I_max - I_min)/(I_max + I_min)

टुकड़ा-वार-रैखिक परिवर्तन कार्य -

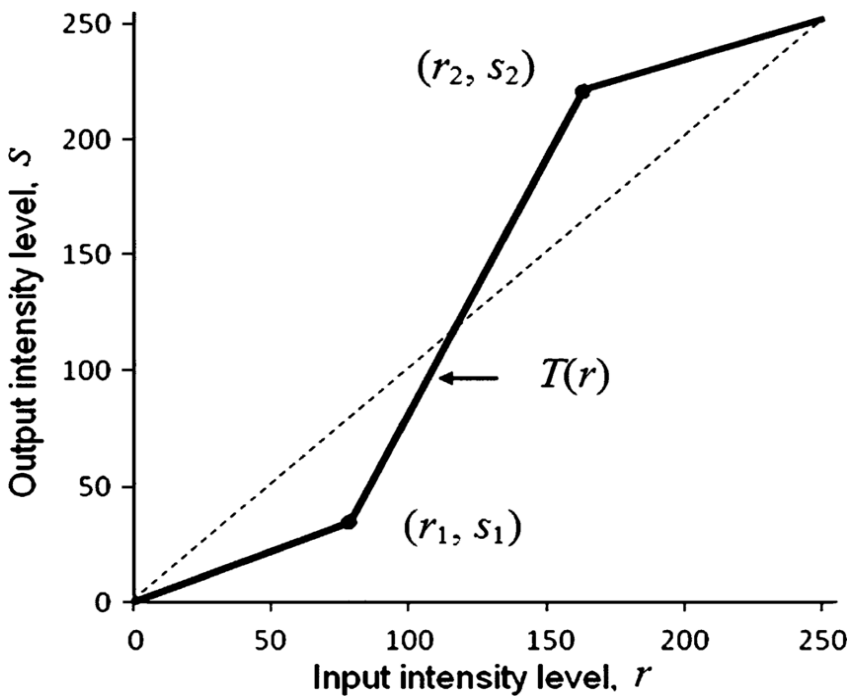
जैसा कि नाम से पता चलता है, ये कार्य पूरी तरह से रैखिक प्रकृति के नहीं हैं। हालाँकि, वे कुछ x-अंतरालों के बीच रैखिक होते हैं। सबसे अधिक इस्तेमाल किए जाने वाले टुकड़ा-वार-रैखिक परिवर्तन कार्यों में से एक कंट्रास्ट स्ट्रेचिंग है। कंट्रास्ट को इस प्रकार परिभाषित किया जा सकता है:

कंट्रास्ट = (I_max - I_min)/(I_max + I_min)

This process expands the range of intensity levels in an image so that it spans the full intensity of the camera/display. The figure below shows the graph corresponding to the contrast stretching.

With (r1, s1), (r2, s2) as parameters, the function stretches the intensity levels by essentially decreasing the intensity of the dark pixels and increasing the intensity of the light pixels. If r1 = s1 = 0 and r2 = s2 = L-1, the function becomes a straight dotted line in the graph (which gives no effect). The function is monotonically increasing so that the order of intensity levels between pixels is preserved. Below is the Python code to perform contrast stretching.

यह प्रक्रिया किसी छवि में तीव्रता के स्तरों की सीमा का विस्तार करती है ताकि यह कैमरा/डिस्प्ले की पूरी तीव्रता को कवर कर सके। नीचे दिया गया चित्र कंट्रास्ट स्ट्रेचिंग के अनुरूप ग्राफ दिखाता है। (r1, s1), (r2, s2) को पैरामीटर के रूप में इस्तेमाल करते हुए, फ़ंक्शन डार्क पिक्सल की तीव्रता को अनिवार्य रूप से कम करके और लाइट पिक्सल की तीव्रता को बढ़ाकर तीव्रता के स्तरों को फैलाता है। यदि r1 = s1 = 0 और r2 = s2 = L-1 है, तो फ़ंक्शन ग्राफ में एक सीधी बिंदीदार रेखा बन जाता है (जिससे कोई प्रभाव नहीं पड़ता)। फ़ंक्शन एकरस रूप से बढ़ रहा है ताकि पिक्सल के बीच तीव्रता के स्तरों का क्रम संरक्षित रहे। कंट्रास्ट स्ट्रेचिंग करने के लिए नीचे पायथन कोड दिया गया है




```
import cv2
import numpy as np
# Function to map each intensity level to output intensity level.
def pixelVal(pix, r1, s1, r2, s2):
if (0 <= pix and pix <= r1):
return (s1 / r1)*pix
elif (r1 < pix and pix <= r2):
return ((s2 - s1)/(r2 - r1)) * (pix - r1) + s1
else:
return ((255 - s2)/(255 - r2)) * (pix - r2) + s2
```

```
# Open the image.
img = cv2.imread('sample.jpg')
# Define parameters.
r1 = 70
s1 = 0
r2 = 140
s2 = 255
# Vectorize the function to apply it to each value in the Numpy array.
pixelVal_vec = np.vectorize(pixelVal)
# Apply contrast stretching.
contrast_stretched = pixelVal_vec(img, r1, s1, r2, s2)
# Save edited image.
cv2.imwrite('contrast_stretch.jpg', contrast_stretched)
```

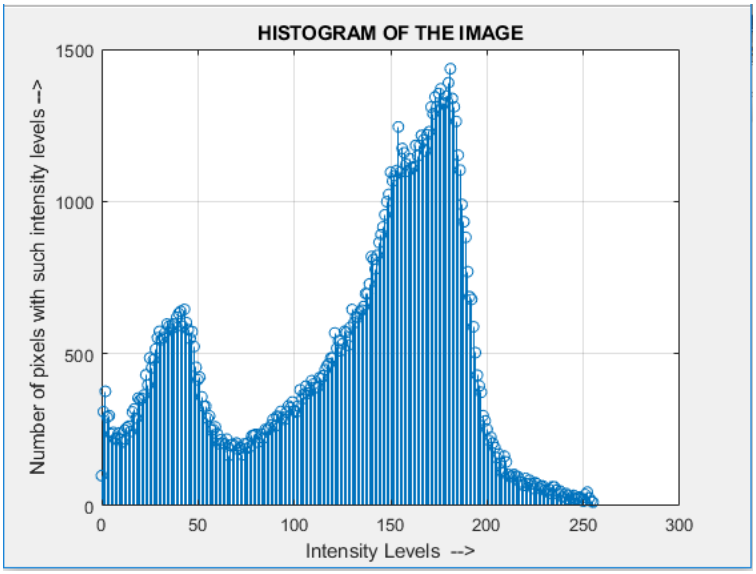
Output:

Digital Image Processing Lab
Experiment No-8
डिजिटल इमेज प्रोसेसिंग लैब
प्रयोग संख्या-8

Aim:
To study and perform Image Binarization and mask formation using different threshold values. Further application of mask using:
Multiplication operation
Addition operation
AND operation
OR operation

उद्देश्य: विभिन्न थ्रेशोल्ड मानों का उपयोग करके इमेज बाइनरीकरण और मास्क निर्माण का अध्ययन और प्रदर्शन करना। मास्क का आगे का अनुप्रयोग:
गुणन ऑपरेशन
जोड़ ऑपरेशन
और ऑपरेशन
या ऑपरेशन

Theory:
Thresholding
Thresholding is one of the segmentation techniques that generates a binary image (a binary image is one whose pixels have only two values – 0 and 1 and thus requires only one bit to store pixel intensity) from a given grayscale image by separating it into two regions based on a threshold value. Hence pixels having intensity values greater than the said threshold will be treated as white or 1 in the output image and the others will be black or 0.
सिद्धांत:
थ्रेशोल्डिंग: थ्रेशोल्डिंग सेगमेंटेशन तकनीकों में से एक है जो किसी दिए गए ग्रेस्केल इमेज से बाइनरी इमेज (बाइनरी इमेज वह होती है जिसके पिक्सल में केवल दो मान होते हैं - 0 और 1 और इस प्रकार पिक्सल तीव्रता को संग्रहीत करने के लिए केवल एक बिट की आवश्यकता होती है) उत्पन्न करती है, इसे थ्रेशोल्ड मान के आधार पर दो क्षेत्रों में विभाजित करके। इसलिए उक्त थ्रेशोल्ड से अधिक तीव्रता मान वाले पिक्सल को आउटपुट इमेज में सफ़ेद या 1 माना जाएगा और अन्य काले या 0 होंगे।



Suppose the above is the histogram of an image f(x,y). We can see one peak near level 40 and another at 180. So there are two major groups of pixels – one group consisting of pixels having a darker shade and the others having a lighter shade. So there can be an object of interest set in the background. If we use an appropriate threshold value, say 90, will divide the entire image into two distinct regions.
In other words, if we have a threshold T, then the segmented image g(x,y) is computed as shown below:
So the output segmented image has only two classes of pixels – one having a value of 1 and others having a value of 0. If the threshold T is constant in processing over the entire image region, it is said to be global thresholding. If T varies over the image region, we say it is variable thresholding.

मान लीजिए कि ऊपर एक छवि f(x,y) का हिस्टोग्राम है। हम एक चोटी को स्तर 40 के पास और दूसरी को 180 पर देख सकते हैं। इसलिए पिक्सेल के दो प्रमुख समूह हैं - एक समूह में गहरे शेड वाले पिक्सेल होते हैं और अन्य में हल्के शेड होते हैं। इसलिए पृष्ठभूमि में रुचि की कोई वस्तु सेट की जा सकती है। यदि हम एक उपयुक्त थ्रेशोल्ड मान का उपयोग करते हैं, जैसे कि 90, तो यह पूरी छवि को दो अलग-अलग क्षेत्रों में विभाजित करेगा। दूसरे शब्दों में, यदि हमारे पास एक थ्रेशोल्ड T है, तो खंडित छवि g(x,y) की गणना नीचे दिखाए अनुसार की जाती है: इसलिए आउटपुट खंडित छवि में पिक्सेल के केवल दो वर्ग हैं - एक का मान 1 है और अन्य का मान 0 है। यदि थ्रेशोल्ड T पूरे छवि क्षेत्र में प्रसंस्करण में स्थिर है, तो इसे वैश्विक थ्रेशोल्डिंग कहा जाता है। यदि T छवि क्षेत्र में भिन्न होता है, तो हम कहते हैं कि यह परिवर्तनशील थ्रेशोल्डिंग है

Multiple-thresholding classifies the image into three regions – like two distinct objects on a background. The histogram in such cases shows three peaks and two valleys between them. The segmented image can be completed using two appropriate thresholds T1 and T2.
where a, b and c are three distinct intensity values
मल्टीपल-थ्रेशोल्डिंग छवि को तीन क्षेत्रों में वर्गीकृत करती है - जैसे पृष्ठभूमि पर दो अलग-अलग ऑब्जेक्ट। ऐसे मामलों में हिस्टोग्राम तीन चोटियों और उनके बीच दो घाटियों को दर्शाता है। खंडित छवि को दो उपयुक्त थ्रेसहोल्ड T1 और T2 का उपयोग करके पूरा किया जा सकता है।
जहाँ a, b और c तीन अलग-अलग तीव्रता मान हैं
From the above discussion, we may intuitively infer that the success of intensity thresholding is directly related to the width and depth of the valleys separating the histogram modes. In turn, the key factors affecting the properties of the valleys are the separation between peaks, the noise content in the image, and

the relative sizes of objects and backgrounds. The more widely the two peaks in the histogram are separated, the better thresholding and hence image segmenting algorithms will work. Noise in an image often degrades this widely-separated two-peak histogram distribution and leads to difficulties in adequate thresholding and segmenting. When noise is present, it is appropriate to use some filter to clean the image and then apply segmentation. The relative object sizes play a role in determining the accuracy of segmentation

उपरोक्त चर्चा से, हम सहज रूप से यह अनुमान लगा सकते हैं कि तीव्रता सीमा की सफलता सीधे हिस्टोग्राम मोड को अलग करने वाली घाटियों की चौड़ाई और गहराई से संबंधित है। बदले में, घाटियों के गुणों को प्रभावित करने वाले प्रमुख कारक चोटियों के बीच अलगाव, छवि में शोर सामग्री और वस्तुओं और पृष्ठभूमि के सापेक्ष आकार हैं। हिस्टोग्राम में दो चोटियों को जितना अधिक अलग किया जाता है, उतनी ही बेहतर सीमा और इसलिए छवि सेगमेंटिंग एल्गोरिदम काम करेंगे। एक छवि में शोर अक्सर इस व्यापक रूप से अलग किए गए दो-शिखर हिस्टोग्राम वितरण को खराब करता है और पर्याप्त सीमा और सेगमेंटिंग में कठिनाइयों का कारण बनता है। जब शोर मौजूद होता है, तो छवि को साफ करने के लिए कुछ फ़िल्टर का उपयोग करना और फिर विभाजन लागू करना उचित होता है। सापेक्ष वस्तु आकार विभाजन की सटीकता निर्धारित करने में एक भूमिका निभाते हैं

Global Thresholding

वैश्विक सीमा

Original Image

Thresholded and segmented Image

When the intensity distribution of objects and background are sufficiently distinct, it is possible to use a single or global threshold applicable over the entire image. The basic global thresholding algorithm iteratively finds the best threshold value so segmenting

The algorithm is explained below.

1. Select an initial estimate of the threshold T.
2. Segment the image using T to form two groups G1 and G2: G1 consists of all pixels with intensity values > T, and G2 consists of all pixels with intensity values $\leq$ T.
3. Compute the average intensity values m1 and m2 for groups G1 and G2.σ

4. Compute the new value of the threshold T as T = (m1 + m2)/2
5. Repeat steps 2 through 4 until the difference in the subsequent value of T is smaller than a pre-defined value δ.
6. Segment the image as g(x,y) = 1 if f(x,y) > T and g(x,y) = 0 if f(x,y) $\leq$ T.

This algorithm works well for images that have a clear valley in their histogram. The larger the value of δ, the smaller will be the number of iterations. The initial estimate of T can be made equal to the average pixel intensity of the entire image

The above simple global thresholding can be made optimum by using Otsu's method. Otsu's method is optimum in the sense that it maximizes the between-class variance. The basic idea is that well-thresholded classes or groups should be distinct with respect to the intensity values of their pixels and conversely, a threshold giving the best separation between classes in terms of their intensity values would be the best or optimum threshold.

जब वस्तुओं और पृष्ठभूमि का तीव्रता वितरण पर्याप्त रूप से अलग होता है, तो संपूर्ण छवि पर लागू एकल या वैश्विक सीमा का उपयोग करना संभव होता है। बुनियादी वैश्विक सीमा निर्धारण एल्गोरिथ्म पुनरावृत्तिपूर्वक सर्वोत्तम सीमा मान ढूँढता है, इसलिए सेगमेंटिंग

एल्गोरिदम को नीचे समझाया गया है।

1. सीमा T का प्रारंभिक अनुमान चुनें।

2. दो समूह G1 और G2 बनाने के लिए T का उपयोग करके छवि को विभाजित करें: G1 में वे सभी पिक्सेल शामिल हैं जिनका तीव्रता मान > T है, और G2 में वे सभी पिक्सेल शामिल हैं जिनका तीव्रता मान $\leq$ T है।

3. समूह G1 और G2 के लिए औसत तीव्रता मान m1 और m2 की गणना करें।σ

4. थ्रेशोल्ड T का नया मान T = (m1 + m2)/2 के रूप में परिकलित करें

5. चरण 2 से 4 को तब तक दोहराएँ जब तक कि T के बाद के मान में अंतर पूर्व-निर्धारित मान δ से छोटा न हो जाए। 6. छवि को g(x,y) = 1 के रूप में विभाजित करें यदि f(x,y) > T और g(x,y) = 0 यदि f(x,y) $\leq$ T। यह एल्गोरिथ्म उन छवियों के लिए अच्छी तरह से काम करता है जिनके हिस्टोग्राम में एक स्पष्ट घाटी होती है। δ का मान जितना बड़ा होगा, पुनरावृत्तियों की संख्या उतनी ही कम होगी। T का प्रारंभिक अनुमान पूरी छवि की औसत पिक्सेल तीव्रता के बराबर बनाया जा सकता है। उपरोक्त सरल वैश्विक थ्रेशोल्डिंग को ओत्सु की विधि का उपयोग करके इष्टतम बनाया जा सकता है। ओत्सु की विधि इस अर्थ में इष्टतम है कि यह वर्ग के बीच के विचरण को अधिकतम करती है। मूल विचार यह है कि अच्छी तरह से थ्रेशोल्ड किए गए वर्ग या समूह अपने पिक्सेल के तीव्रता मूल्यों के संबंध में अलग-अलग होने चाहिए और इसके विपरीत, उनके तीव्रता मूल्यों के संदर्भ में वर्गों के बीच सबसे अच्छा अलगाव देने वाली सीमा सबसे अच्छी या इष्टतम सीमा होगी।

Variable Thresholding

There are broadly two different approaches to local thresholding. One approach is to partition the image into non-overlapping rectangles. Then the techniques of global thresholding or Otsu's method are applied to each of the sub-images. Hence in the image partitioning technique, the methods of global thresholding are applied to each sub-image rectangle by assuming that each such rectangle is a separate image in itself. This approach is justified when the sub-image histogram properties are suitable (have two peaks with a wide valley in between) for the application of thresholding techniques but the entire image histogram is corrupted by noise and hence is not ideal for global thresholding.

The other approach is to compute a variable threshold at each point from the neighborhood pixel properties. Let us say that we have a neighborhood Sxy of a pixel having coordinates (x,y). If the mean and standard deviation of pixel intensities in this neighborhood be mxy and σxy , then the threshold at each point can be computed as:

where a and b are arbitrary constants. The above definition of the variable threshold is just an example. Other definitions can also be used according to the need.

The segmented image is computed as:

Moving averages can also be used as thresholds. This technique of image thresholding is the most general one and can be applied to widely different cases.

परिवर्तनशील थ्रेशोल्डिंग

स्थानीय थ्रेशोल्डिंग के लिए मोटे तौर पर दो अलग-अलग दृष्टिकोण हैं। एक दृष्टिकोण छवि को गैर-अतिव्यापी आयतों में विभाजित करना है। फिर वैश्विक थ्रेशोल्डिंग या ओत्सु की विधि की तकनीकों को प्रत्येक उप-छवि पर लागू किया जाता है। इसलिए छवि विभाजन तकनीक में, वैश्विक थ्रेशोल्डिंग की विधियों को प्रत्येक उप-छवि आयत पर लागू किया जाता है, यह मानकर कि प्रत्येक ऐसी आयत अपने आप में एक अलग छवि है। यह दृष्टिकोण तब उचित है जब उप-छवि हिस्टोग्राम गुण थ्रेशोल्डिंग तकनीकों के अनुप्रयोग के लिए उपयुक्त हों (बीच में एक विस्तृत घाटी के साथ दो चोटियाँ हों) लेकिन संपूर्ण छवि हिस्टोग्राम शोर से दूषित हो और इसलिए वैश्विक थ्रेशोल्डिंग के लिए आदर्श न हो।

दूसरा तरीका पड़ोस पिक्सेल गुणों से प्रत्येक बिंदु पर एक चर सीमा की गणना करना है। मान लें कि हमारे पास निर्देशांक (x, y) वाले पिक्सेल का एक पड़ोस Sxy है। यदि इस पड़ोस में पिक्सेल तीव्रता का माध्य और मानक विचलन mxy और σxy है, तो प्रत्येक बिंदु पर सीमा की गणना इस प्रकार की जा सकती है:

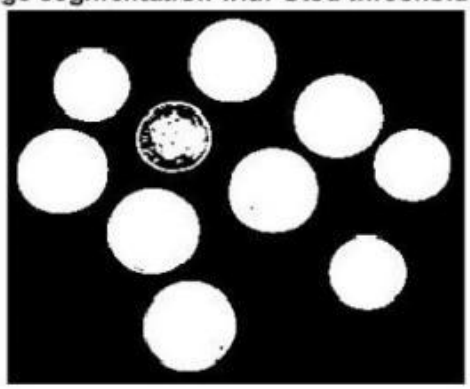
जहाँ a और b मनमाने स्थिरांक हैं। चर सीमा की उपरोक्त परिभाषा केवल एक उदाहरण है। आवश्यकता के अनुसार अन्य परिभाषाओं का भी उपयोग किया जा सकता है

खंडित छवि की गणना इस प्रकार की जाती है:

चलती औसत का उपयोग थ्रेसहोल्ड के रूप में भी किया जा सकता है। छवि थ्रेशोल्डिंग की यह तकनीक सबसे सामान्य है और इसे व्यापक रूप से अलग-अलग मामलों में लागू किया जा सकता है।

Original image.

Image segmentation with Otsu thresholding.



Digital Image Processing Lab
Experiment No-9
डिजिटल इमेज प्रोसेसिंग लैब
प्रयोग संख्या-9

Aim:
To study and perform morphological operations on images:
Erosion
Dilation
Opening
Closing

उद्देश्य: छवियों पर रूपात्मक संचालन का अध्ययन और निष्पादन करना:

क्षरण

फैलाव

खोलना

बंद करना

Theory:
Morphological operations based on OpenCV are as follows:
Erosion
Dilation
Opening
Closing

For all the above techniques the two important requirements are the binary image and a kernel structuring element that is used to slide across the image.
Images used for demonstration:

सिद्धांत:
OpenCV पर आधारित रूपात्मक संचालन इस प्रकार हैं:
क्षरण
फैलाव
खोलना
बंद करना
उपरोक्त सभी तकनीकों के लिए दो महत्वपूर्ण आवश्यकताएँ हैं बाइनरी इमेज और कर्नेल संरचना तत्व जिसका उपयोग इमेज पर स्लाइड करने के लिए किया जाता है।
प्रदर्शन के लिए उपयोग की गई छवियाँ:

Images used

Images used

Erosion

Erosion primarily involves eroding the outer surface (the foreground) of the image. As binary images only contain two pixels 0 and 255, it primarily involves eroding the foreground of the image and it is suggested to

have the foreground as white. The thickness of erosion depends on the size and shape of the defined kernel. We can make use of NumPy's ones () function to define a kernel. There are a lot of other functions like NumPy zeros, customized kernels, and others that can be used to define kernels based on the problem in hand.

Code:

- Import the necessary packages as shown
- Read the image
- Binarize the image.
- As it is advised to keep the foreground in white, we are performing OpenCV's invert operation on the binarized image to make the foreground as white
- We are defining a 5×5 kernel filled with ones
- Then we can make use of Opencv erode() function to erode the boundaries of the image.
- • Python3

क्षरण

क्षरण में मुख्य रूप से छवि की बाहरी सतह (अग्रभूमि) का क्षरण शामिल है। चूंकि बाइनरी छवियों में केवल दो पिक्सेल 0 और 255 होते हैं, इसलिए इसमें मुख्य रूप से छवि के अग्रभूमि का क्षरण शामिल है और अग्रभूमि को सफ़ेद रखने का सुझाव दिया जाता है। क्षरण की मोटाई परिभाषित कर्नेल के आकार और आकार पर निर्भर करती है। हम कर्नेल को परिभाषित करने के लिए NumPy के वन () फ़ंक्शन का उपयोग कर सकते हैं। NumPy शून्य, अनुकूलित कर्नेल और अन्य जैसे कई अन्य फ़ंक्शन हैं जिनका उपयोग हाथ में समस्या के आधार पर कर्नेल को परिभाषित करने के लिए किया जा सकता है।

कोड:

• दिखाए गए अनुसार आवश्यक पैकेज आयात करें

• छवि पढ़ें

• छवि को बाइनरीकृत करें।

• चूंकि अग्रभूमि को सफ़ेद रखने की सलाह दी जाती है, इसलिए हम अग्रभूमि को सफ़ेद बनाने के लिए बाइनरीकृत छवि पर OpenCV का उलटा ऑपरेशन कर रहे हैं

• हम 5×5 कर्नेल को परिभाषित कर रहे हैं जो कि वन से भरा हुआ है

• फिर हम इमेज की सीमाओं को मिटाने के लिए Opencv erode() फ़ंक्शन का उपयोग कर सकते हैं

• पायथन3

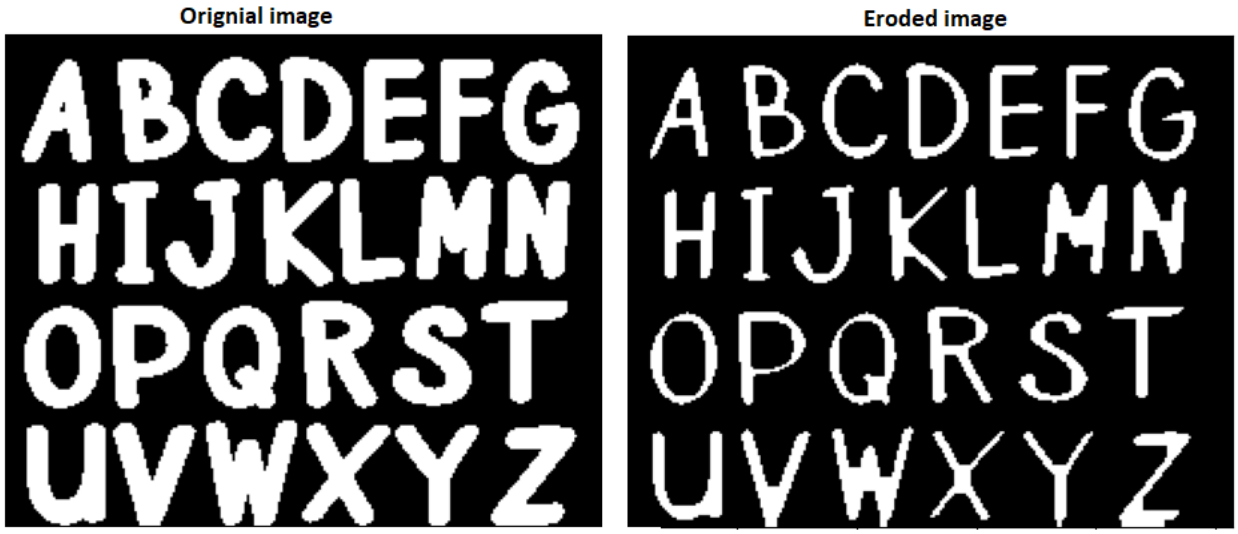
```
# import the necessary packages
import cv2
import numpy as np
import matplotlib.pyplot as plt
# read the image
img = cv2.imread(r"Downloads\test (2).png", 0)
# binarize the image
binr = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
# define the kernel
kernel = np.ones((5, 5), np.uint8)
# invert the image
invert = cv2.bitwise_not(binr)
```

```
# erode the image
erosion = cv2.erode(invert, kernel,
iterations=1)
# print the output
plt.imshow(erosion, cmap='gray')
```

Output:

The output should be a thinner image than the original one.

आउटपुट: आउटपुट मूल छवि की तुलना में पतली छवि होनी चाहिए।



Erosion

Dilation

Dilation involves dilating the outer surface (the foreground) of the image. As binary images only contain two pixels 0 and 255, it primarily involves expanding the foreground of the image and it is suggested to have the foreground as white. The thickness of erosion depends on the size and shape of the defined kernel. We can make use of NumPy's ones () function to define a kernel. There are a lot of other functions like NumPy zeros, customized kernels, and others that can be used to define kernels based on the problem at hand. It is exactly opposite to the erosion operation

फैलाव

फैलाव में छवि की बाहरी सतह (अग्रभूमि) को फैलाना शामिल है। चूंकि बाइनरी छवियों में केवल दो पिक्सेल 0 और 255 होते हैं, इसलिए इसमें मुख्य रूप से छवि के अग्रभूमि का विस्तार करना शामिल है और अग्रभूमि को सफ़ेद रखने का सुझाव दिया जाता है। क्षरण की मोटाई परिभाषित कर्नेल के आकार और आकार पर निर्भर करती है। हम कर्नेल को परिभाषित करने के लिए NumPy के वन () फ़ंक्शन का उपयोग कर सकते हैं। NumPy शून्य, अनुकूलित कर्नेल और अन्य जैसे कई अन्य फ़ंक्शन हैं जिनका उपयोग समस्या के आधार पर कर्नेल को परिभाषित करने के लिए किया जा सकता है। यह क्षरण ऑपरेशन के बिल्कुल विपरीत है

Code:

• Import the necessary packages as shown
• Read the image
• Binarize the image.
• As it is advised to keep the foreground in white, we are performing OpenCV's invert operation on the binarized image to make the foreground white.
• We are defining a 3×3 kernel filled with ones
• Then we can make use of the Opencv dilate() function to dilate the boundaries of the image.

• Python3
कोड:
• दिखाए गए अनुसार आवश्यक पैकेज आयात करें
• छवि पढ़ें
• छवि को बाइनरी करें।
• जैसा कि अग्रभूमि को सफ़ेद रखने की सलाह दी जाती है, हम अग्रभूमि को सफ़ेद बनाने के लिए बाइनरीकृत छवि पर OpenCV का उलटा ऑपरेशन कर रहे हैं।
• हम 3×3 कर्नेल को वन से भरकर परिभाषित कर रहे हैं
• फिर हम छवि की सीमाओं को फैलाने के लिए Opencv dilate() फ़ंक्शन का उपयोग कर सकते हैं।

• Python3

```
import cv2
# read the image
img = cv2.imread(r"path to image", 0)
# binarize the image
binr = cv2.threshold(img, 0, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)[1]
# define the kernel
kernel = np.ones((3, 3), np.uint8)
# invert the image
invert = cv2.bitwise_not(binr)
# dilate the image
dilation = cv2.dilate(invert, kernel, iterations=1)
# print the output
plt.imshow(dilation, cmap='gray')
```

Output:
The output should be a thicker image than the original one.
आउटपुट: आउटपुट मूल छवि से अधिक मोटी होनी चाहिए।

Dilated image

Opening

Opening involves erosion followed by dilation in the outer surface (the foreground) of the image. All the above-said constraints for erosion and dilation applies here. It is a blend of the two prime methods. It is generally used to remove the noise in the image.

Code:

• Import the necessary packages as shown

• Read the image

• Binarize the image.

• We are defining a 3×3 kernel filled with ones

• Then we can make use of the Opencv cv.morphologyEx() function to perform an Opening operation on the image.

खोलना

खोलने में छवि की बाहरी सतह (अग्रभूमि) में क्षरण के बाद फैलाव शामिल है। क्षरण और फैलाव के लिए ऊपर बताई गई सभी बाधाएँ यहाँ लागू होती हैं। यह दो प्रमुख विधियों का मिश्रण है। इसका उपयोग आम तौर पर छवि में शोर को हटाने के लिए किया जाता है।

कोड:

• दिखाए गए अनुसार आवश्यक पैकेज आयात करें

• छवि पढ़ें

• छवि को बाइनरी करें।

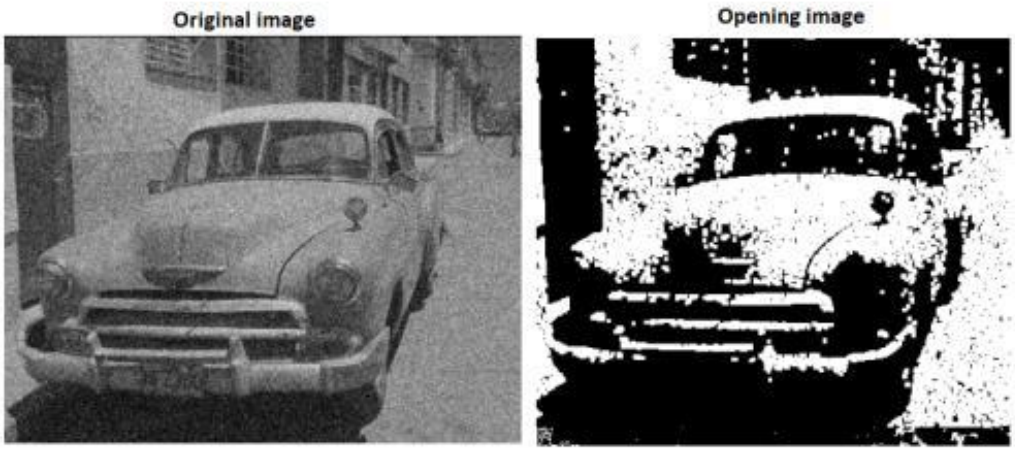
• हम एक 3×3 कर्नेल को परिभाषित कर रहे हैं जो कि वन से भरा हुआ है

• फिर हम छवि पर ओपनिंग ऑपरेशन करने के लिए Opencv cv.morphologyEx() फ़ंक्शन का उपयोग कर सकते हैं।

Python3

```
# import the necessary packages
import cv2
# read the image
img = cv2.imread(r"\noise.png", 0)
# binarize the image
binr = cv2.threshold(img, 0, 255,
cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
# define the kernel
kernel = np.ones((3, 3), np.uint8)
# opening the image
opening = cv2.morphologyEx(binr, cv2.MORPH_OPEN,
kernel, iterations=1)
# print the output
plt.imshow(opening, cmap='gray')
```

Output:



Opening Image

Closing

Closing involves dilation followed by erosion in the outer surface (the foreground) of the image. All the above-said constraints for erosion and dilation applies here. It is a blend of the two prime methods. It is generally used to remove the noise in the image.

Code:

- Import the necessary packages as shown
- Read the image
- Binarize the image.
- We are defining a 3×3 kernel filled with ones
- Then we can make use of the Opencv cv.morphologyEx() function to perform a Closing operation on the image
- Python3
- **क्लोजिंग**

- क्लोजिंग में छवि की बाहरी सतह (अग्रभूमि) में फैलाव के बाद क्षरण शामिल है। क्षरण और फैलाव के लिए ऊपर बताई गई सभी बाध्यताएँ यहाँ लागू होती हैं। यह दो प्रमुख विधियों का मिश्रण है। इसका उपयोग आम तौर पर छवि में शोर को हटाने के लिए किया जाता है।

कोड:

• दिखाए गए अनुसार आवश्यक पैकेज आयात करें

• छवि पढ़ें

• छवि को बाइनरी करें।

•हम 3×3 कर्नेल को परिभाषित कर रहे हैं जो कि वन से भरा हुआ है

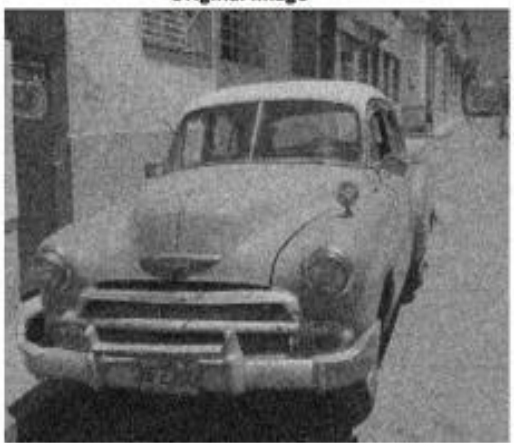
• फिर हम छवि पर क्लोजिंग ऑपरेशन करने के लिए Opencv cv.morphologyEx() फ़ंक्शन का उपयोग कर सकते हैं

- Python3

```
# import the necessary packages
import cv2
# read the image
img = cv2.imread(r"\Images\noise.png", 0)
# binarize the image
binr = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
# define the kernel
kernel = np.ones((3, 3), np.uint8)
# opening the image
closing = cv2.morphologyEx(binr, cv2.MORPH_CLOSE, kernel, iterations=1)
# print the output
plt.imshow(closing, cmap='gray')
```

Output:

Original image



Closing image

Digital Image Processing Lab
ECE-324
Experiment No-10
डिजिटल इमेज प्रोसेसिंग लैब
प्रयोग संख्या-10

Aim:
Perform Edge Detection on an image using techniques like:

- **Sobel Operator**
- **Laplacian Operator**
- **Canny Edge Detection**

उद्देश्य:

निम्नलिखित तकनीकों का उपयोग करके छवि पर एज डिटेक्शन करें:

- सोबेल ऑपरेटर
- लाप्लासियन ऑपरेटर
- कैनी एज डिटेक्शन

Theory:

- **Sobel Operator** is useful for detecting directional edges.
- **Laplacian Operator** is sensitive to noise but can provide better edge maps for sharp transitions.
- **Canny Edge Detection** is more robust and widely used due to its effectiveness in real-world applications.

लिखित:

• सोबेल ऑपरेटर दिशात्मक किनारों का पता लगाने के लिए उपयोगी है।

• लाप्लासियन ऑपरेटर शोर के प्रति संवेदनशील है, लेकिन तीखे बदलावों के लिए बेहतर एज मैप प्रदान कर सकता है।

• कैनी एज डिटेक्शन वास्तविक दुनिया के अनुप्रयोगों में अपनी प्रभावशीलता के कारण अधिक मजबूत और व्यापक रूप से उपयोग किया जाता है।

Explanation

- **Image Loading**

    - The image is loaded using `cv2.imread()` in grayscale mode.
- **Sobel Operator**
    - `cv2.Sobel()` detects horizontal (`dx=1, dy=0`) and vertical (`dx=0, dy=1`) edges using a kernel of size 3.
    - `cv2.magnitude()` computes the gradient magnitude to combine both Sobel results.
- **Laplacian Operator**
    - `cv2.Laplacian()` detects edges by calculating second-order derivatives, highlighting sharper transitions.
- **Canny Edge Detection**
    - `cv2.Canny()` is a multi-stage edge detection algorithm with two thresholds (50 and 150).
    - It detects strong edges and suppresses weak edges using non-maximum suppression.
- **Display Results**
    - Matplotlib is used to display the original and edge-detected images.

स्पष्टीकरण

- छवि लोड करना

छवि को ग्रेस्केल मोड में `cv2.imread()` का उपयोग करके लोड किया जाता है।

- सोबेल ऑपरेटर

`cv2.Sobel()` आकार 3 के कर्नेल का उपयोग करके क्षैतिज (`dx=1, dy=0`) और ऊर्ध्वाधर (`dx=0, dy=1`) किनारों का पता लगाता है।

`cv2.magnitude()` दोनों सोबेल परिणामों को संयोजित करने के लिए ग्रेडिएंट परिमाण की गणना करता है।

- लाप्लासियन ऑपरेटर

`cv2.Laplacian()` दूसरे क्रम के व्युत्पन्नों की गणना करके किनारों का पता लगाता है, जो तीखे संक्रमणों को उजागर करता है।

- कैनी एज डिटेक्शन

`cv2.Canny()` दो थ्रेसहोल्ड (50 और 150) के साथ एक बहु-चरणीय एज डिटेक्शन एल्गोरिदम है।

यह मजबूत किनारों का पता लगाता है और गैर-अधिकतम दमन का उपयोग करके कमजोर किनारों को दबाता है।

- परिणाम प्रदर्शित करें

Matplotlib का उपयोग मूल और किनारे से पहचानी गई छवियों को प्रदर्शित करने के लिए किया जाता है।

Code

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

# Load the image in grayscale
image = cv2.imread('/content/image.jpg', cv2.IMREAD_GRAYSCALE)

# Apply Sobel Operator
sobel_x  =  cv2.Sobel(image,  cv2.CV_64F,  1,  0,  ksize=3)    #
Horizontal edges
sobel_y  =  cv2.Sobel(image,  cv2.CV_64F,  0,  1,  ksize=3)    #
Vertical edges
sobel_combined = cv2.magnitude(sobel_x, sobel_y)

# Apply Laplacian Operator
laplacian = cv2.Laplacian(image, cv2.CV_64F)

# Apply Canny Edge Detection
canny_edges = cv2.Canny(image, 50, 150)

# Convert results to displayable format
sobel_x = cv2.convertScaleAbs(sobel_x)
sobel_y = cv2.convertScaleAbs(sobel_y)
sobel_combined = cv2.convertScaleAbs(sobel_combined)
laplacian = cv2.convertScaleAbs(laplacian)

# Plot the Results
plt.figure(figsize=(12, 8))

plt.subplot(2, 3, 1)
plt.imshow(image, cmap='gray')
plt.title("Original Image")
plt.axis("off")

plt.subplot(2, 3, 2)
plt.imshow(sobel_x, cmap='gray')
plt.title("Sobel X (Horizontal Edges)")
plt.axis("off")
```

```
plt.subplot(2, 3, 3)
plt.imshow(sobel_y, cmap='gray')
plt.title("Sobel Y (Vertical Edges)")
plt.axis("off")

plt.subplot(2, 3, 4)
plt.imshow(sobel_combined, cmap='gray')
plt.title("Sobel Combined")
plt.axis("off")

plt.subplot(2, 3, 5)
plt.imshow(laplacian, cmap='gray')
plt.title("Laplacian")
plt.axis("off")

plt.subplot(2, 3, 6)
plt.imshow(canny_edges, cmap='gray')
plt.title("Canny Edge Detection")
plt.axis("off")

plt.tight_layout()
plt.show()
```