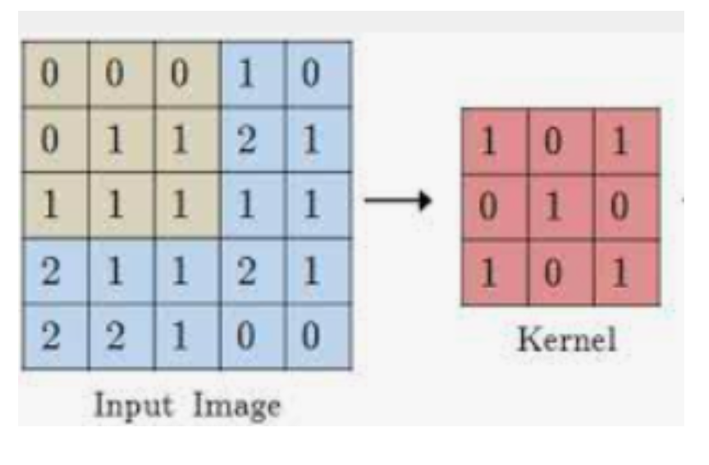
**INDIVIDUAL REPORT 21R241-SHUDHI RISHAA P**

**Aim:**

To Write a python program for convolution operation for the given pixel and kernel values



# **Software/ Packages Used:**

1. Pycharm IDE
2. Libraries used:
   * NumPy
   * opencv-python
   * matplotlib
   * tkinter
   * Sqlite 3

**INTRODUCTION:**

Convolution is a fundamental operation in image processing that involves applying a filter (kernel) to an image's pixel array. This code demonstrates a simple convolution operation using NumPy, a powerful numerical computing library for Python. The example includes a function **convolution** that performs convolution on a given pixel array and a specific kernel. Additionally, it applies this operation to a sample pixel array using an edge detection kernel.

**Theory:**

**1. Convolution Operation:** Convolution is a mathematical operation that combines two functions to produce a third. In image processing, convolution involves sliding a filter (kernel) over an image and computing the sum of element-wise products at each position. This process is commonly used for tasks such as edge detection, blurring, and sharpening.

**2. Convolution Function:**

* **Inputs:**
  + **pixel\_array**: The input 2D array representing the image.
  + **kernel**: The convolution kernel to be applied.
* **Algorithm:**
  + The function initializes an empty result array with the same shape as the input pixel array.
  + It calculates the center position of the kernel.
  + It iterates through each pixel position in the input array, extracts the region of interest (ROI) corresponding to the kernel, and performs element-wise multiplication and summation to obtain the convolution result.
  + The result array is returned.

**3. Example Pixel Array and Kernel:** The example uses a 5x5 pixel array (**pixel\_values**) and an edge detection kernel (**edge\_detection\_kernel**).

**4. Edge Detection Kernel:**

* The kernel used in this example is designed for edge detection. It assigns higher weights to the central and surrounding pixels, emphasizing differences in intensity.

**5. Convolution Result:**

* The script applies the **convolution** function to the example pixel array using the edge detection kernel.
* The original pixel array, the kernel, and the result after convolution are printed for analysis.

**PROGRAM:**

import numpy as np

def convolution(pixel\_array, kernel):

pixel\_array = np.array(pixel\_array)

kernel = np.array(kernel)

height, width = pixel\_array.shape

k\_height, k\_width = kernel.shape

# Calculate the center position of the kernel

center\_x, center\_y = k\_width // 2, k\_height // 2

# Initialize the result array

result = np.zeros\_like(pixel\_array)

# Perform convolution

for y in range(center\_y, height - center\_y):

for x in range(center\_x, width - center\_x):

# Extract the region of interest (ROI) from the pixel array

roi = pixel\_array[y - center\_y:y + center\_y + 1, x - center\_x:x + center\_x + 1]

# Perform element-wise multiplication and sum to get the convolution result

result[y, x] = np.sum(roi \* kernel)

return result

pixel\_values = np.array([[0,0,0,1,0],

[0,1,1,2,1],

[1,1,1,1,1],

[2,1,1,2,1],

[2,2,1,0,0]])

# Example kernel for edge detection

edge\_detection\_kernel = np.array([[1, 0, 1],

[0, 1, 0],

[1, 0, 1]])

# Perform convolution

convolution\_result = convolution(pixel\_values, edge\_detection\_kernel)

# Print the original pixel array and the result after convolution

print("Original Pixel Array:")

print(pixel\_values)

print("\nKernel:")

print(edge\_detection\_kernel)

print("\nResult after Convolution:")

print(convolution\_result)

# **OUTPUT:**

# Original Pixel Array:

# [[0 0 0 1 0]

# [0 1 1 2 1]

# [1 1 1 1 1]

# [2 1 1 2 1]

# [2 2 1 0 0]]

# Kernel:

# [[1 0 1]

# [0 1 0]

# [1 0 1]]

# Result after Convolution:

# [[0 0 0 0 0]

# [0 3 4 4 0]

# [0 5 7 5 0]

# [0 6 5 5 0]

# [0 0 0 0 0]]

# **Inference:**

# The convolution operation enhances features in the image based on the characteristics of the kernel.

# The edge detection kernel accentuates changes in intensity, highlighting edges in the image.

# Convolution is a versatile technique with applications in various image processing tasks.

# **Conclusion:** This code provides a concise implementation of the convolution operation using NumPy, showcasing its utility in image processing. Understanding and modifying the convolution kernel can tailor the operation for different applications, making it a crucial tool in computer vision and image analysis.

# 

# **RESULT:**

a python program for convolution operation for the given pixel and kernel values were written and verified.