1 Convolution from Scratch

In this exercise, you will explore the fundamentals of convolution, a vital operation in various fields such as signal processing, image processing, and deep learning. Convolution is a mathematical operation that combines two functions to produce a third, representing how one function modifies the other. To gain a deeper understanding of convolution, you will perform a manual convolution operation.

1.1 Subtask 1: Importing Essential Packages

Preparing for Manual Convolution🚀

1. **Set Up Your Coding Environment**

   Before delving into manual convolution, ensure you have a suitable coding environment ready. If you already have one, excellent! If not, now is the time to set it up.

2. **Identify Required Packages**

   Our journey requires a couple of trusty packages. Specifically, we need NumPy and SciPy. NumPy excels in numerical operations and array manipulations, while SciPy empowers you with advanced scientific computing capabilities. So, equip yourself with these powerful tools.

3. **Import Packages**

   With your treasures located, let's summon them into your coding realm with the following incantations:

```
import numpy as np  #NumPy for arrays and math
from scipy import signal  #SciPy for scientific
    computing
```

   Pro Tip: If you're feeling creative, give your packages cool nicknames (like "np"for NumPy) to make your code even more magical and easy to type.

3... 2... 1... Abracadabra! You now have the mystical powers of NumPy and SciPy at your fingertips. You're all set to conquer the world of manual convolution!

1.2 Subtask 2: Loading and Preparing Data

Data Preparation for Manual Convolution

1. **Load Data**

   Let's begin by loading our starship data, the image, from the enigmatic `sample.npy` file using NumPy's magic:

```
img = np.load("sample.npy")
```

2. **Data Transformation**

   Prepare the image by rounding down its values with the "np.floor()"command and save it back to `img`:

```
 img = np.floor(img / 2)
```

3. **Load the Filter**

   But wait, we also need a top-secret filter to navigate through the cosmos! Load it from the "filter.npy"file with NumPy:

```
fltr = np.load("filter.npy")
```

4. **Data Verification**

   Before our journey commences, inspect the first few elements of `img` and `fltr` by printing them:

```
print("Image data:")
print(img[:5])}  #Print the first 5 elements of img

print("Filter data:")
print(fltr[:5])  #Print the first 5 elements of fltr
```

5. **Ready to Roll**

   With our data loaded, transformed, and the filter in hand, we're all set to kick off the thrilling manual convolution process. Get ready to explore the universe of convolution in Subtask 3!

Exciting times await in the universe of manual convolution.

1.3 Subtask 3: Implementing Manual Convolution

Navigating the Galaxy of Image Processing

1. **The Manual Convolution Function**

   It's time to implement a custom convolution function named `Myconv2d`. Let's break it down step by step:

```
def Myconv2d(a, b):
```

   This function takes two inputs, `a` and `b`, representing your data and filter.

2. **Convert to NumPy Arrays**

   To prepare your data and filter for the convolution process, convert `a` and `b` into NumPy arrays:

```
    a = np.array(a)  #Convert data to a NumPy array
    b = np.array(b)  #Convert filter to a NumPy array
```

3. **Determine Data and Filter Shapes**

   Understand the dimensions of your data and filter by determining their shapes:

```
    a_shape = np.shape(a)  #Get the shape of data
    b_shape = np.shape(b)  #Get the shape of the
       filter
```

4. **Calculate Result Dimensions**

   Calculate the dimensions of the result, "res"based on the differences in dimensions:

```
    res_shape1 = np.abs(a_shape[0] - b_shape[0]) + 1
        #Calculate the first dimension of the result
    res_shape2 = np.abs(a_shape[1] - b_shape[1]) + 1
        #Calculate the second dimension of the result
```

5. **Create an Empty Result Matrix**

   Prepare a matrix, "res"to store the convolution results:

```
    res = np.zeros([res_shape1, res_shape2])  #Create
        a matrix to store convolution results
```

6. **Perform Manual Convolution**

   Traverse through the data to calculate the convolution result:

```
    for i in range(res_shape1):
        for j in range(res_shape2):
            res[i, j] = np.sum(
                np.multiply(
                    np.flip(b), a[i:i + b_shape[0], j
                       :j + b_shape[1]]
                )
            )
```

7. **Return the Resulting Matrix**

   Your manual convolution function is complete. Now, return the resulting matrix, ”res”:

```
    return res  #Return the convolution result
```

   You’ve successfully created your manual convolution function! Get ready to apply it to your data and explore the fascinating world of convolution in action!

May the convolution be with you!

1.4 Subtask 4: Checking the Result of Your Implementation

Comparing Manual Convolution to Python Library Implementation

1. **Calculate the Manual Convolution Result**

   Your custom convolution function Myconv2d is ready. Put it to the test:

```
Result_Imp = Myconv2d(img, fltr)  # Calculate the
    convolution result using your implementation
```

2. **Calculate the Python Library Convolution Result**

   Don’t forget Python’s library magic! Use the signal.convolve2d function from SciPy to calculate the convolution result:

```
Result_Python = signal.convolve2d(img, fltr, mode="
    valid")  # Calculate the convolution result using
    the Python library
```

3. **Compare the Results**

   It’s time for a grand comparison. Calculate the error value by comparing the results from your implementation and the Python library:

```
checkup = np.sum(np.abs(Result_Imp - Result_Python))
     # Calculate the error value by comparing the two
    results
```

4. **Display the Error Value**

   Display the error value to gauge how closely your implementation matches the Python library:

```
print("The Error Value is:", checkup)  # Display the
    error value to assess the closeness of your
    results to the Python library
```

5. **The Verdict**

   Results checked! If the error value is close to zero, your manual convolution implementation is performing admirably. The smaller the error, the closer your results align with the Python library’s results.

You’re now ready to evaluate your manual convolution implementation. May the results be in your favor! ⚖