KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
AN AUTONOMOUS INSTITUTION - ACCREDITED BY NAAC WITH 'A' GRADE
Narayanaguda, Hyderabad.

Deep Learning

CNN

EXERCISE-1

18-11-2024

BY
ASHA

EXERCISE1: 3* 3 convolution on 5* 5 image with and without padding

In this task, you will apply a 3x3 convolution on a 5x5 image matrix using a given filter. This will involve performing the convolution operation with and without padding to understand how padding affects the output dimensions.

Follow the steps below to complete the task:

1.Import necessary libraries (Numpy)

2.Define the 5x5 image matrix

3.Define the 3x3 filter (kernel) matrix

4.Convolution Operation

5.Convolution Operation Without padding

6.Convolution Operation With padding of 1

2.Define the 5x5 image matrix

The 5×5 image matrix is a numerical representation of an image where each value corresponds to the intensity of a pixel.

$$\text{image} = \begin{bmatrix} 1 & 2 & 3 & 0 & 1 \\ 4 & 5 & 6 & 1 & 2 \\ 7 & 8 & 9 & 2 & 3 \\ 1 & 2 & 3 & 0 & 1 \\ 4 & 5 & 6 & 1 & 2 \end{bmatrix}$$

```
# Define the 5x5 image matrix
image = np.array([
    [1, 2, 3, 0, 1],
    [4, 5, 6, 1, 2],
    [7, 8, 9, 2, 3],
    [1, 2, 3, 0, 1],
    [4, 5, 6, 1, 2]
])
```

3.Define the 3x3 filter (kernel) matrix

The 3×3 filter (kernel) matrix is used for feature extraction in an image. It works by performing a convolution operation, where it slides over an image and calculates a weighted sum of pixel intensities at each position.

$$\text{filter\_kernel} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

```python
# Define the 3x3 filter (kernel) matrix
filter_kernel = np.array([
    [1, 0, -1],
    [1, 0, -1],
    [1, 0, -1]
])
```

4.Convolution Operation

Write a Python function named convolve to perform a 2D convolution operation on a grayscale image. The function should:

Take three arguments:

image: A 2D NumPy array representing a grayscale image.
kernel: A 2D NumPy array representing the convolution filter (kernel).
padding: An integer specifying the number of zero-padded rows/columns to add around the image (default is 0).
Return:

A 2D NumPy array representing the convolved output.

Function Signature:

```
def convolve(image, kernel, padding=0):
    #your code here
    return output
```

Convolution function steps

1. Add padding to the image if needed
2. Dimensions of the image and kernel
3. Calculate output dimensions
4. Initialize the output matrix
5. Perform convolution
6. return output

The function should implement a **convolution operation** on a 2D image matrix using a given kernel (filter).

1. Input Parameters

1.image: A 2D array representing the input image.

- Example: A 5×55 \times 55×5 matrix of pixel values.

2.kernel: A smaller 2D array representing the filter to be applied.

- Example: A 3×33 \times 33×3 matrix for edge detection.

3.padding: Specifies how many layers of zero-padding to add around the image.

- Default is 0 (no padding).

```
# Convolution function
def convolve(image, kernel, padding=0):
```

2. Steps in the Function

Step 1: Add Padding

•**Purpose**: Padding adds a border around the original image, allowing the kernel to process edge pixels.

•**np.pad**:

- •Adds padding layers of zeros around the image.
- •The mode='constant' ensures zeros are added.

```
if padding > 0:
    image = np.pad(image, ((padding, padding), (padding, padding)), mode='constant')
```

Step 2: Extract Dimensions

•Extract the height and width of the image and kernel to calculate the output dimensions.

```
# Dimensions of the image and kernel
image_height, image_width = image.shape
kernel_height, kernel_width = kernel.shape
```

Step 3: Calculate Output Dimensions

- **Formula**

$$\text{Output Height} = \text{Image Height} - \text{Kernel Height} + 1$$

$$\text{Output Width} = \text{Image Width} - \text{Kernel Width} + 1$$

- **Why?**: The kernel only fits within certain positions in the image, reducing the output size.
- **With Padding**: If padding is applied, it compensates for the size reduction.

```
# Calculate output dimensions
output_height = image_height - kernel_height + 1
output_width = image_width - kernel_width + 1
```

Step 4: Initialize the Output Matrix

- Creates a zero matrix with the calculated output dimensions.
- This matrix will store the result of the convolution operation.

```
# Initialize the output matrix
output = np.zeros((output_height, output_width), dtype=int)
```

Step 5: Perform Convolution

1.Loop Over the Output Matrix:

- i and j iterate over each position in the output matrix.

2.Extract a Region:

- region = image[i:i+kernel_height, j:j+kernel_width]: Extracts a submatrix of the image the same size as the kernel.

3.Element-Wise Multiplication:

- region * kernel: Multiplies each element of the kernel with the corresponding element of the region.

4.Sum the Result:

- np.sum(region * kernel): Sums all the products to produce a single value for the current position.

```
# Perform convolution
for i in range(output_height):
    for j in range(output_width):
        region = image[i:i+kernel_height, j:j+kernel_width]
        output[i, j] = np.sum(region * kernel)
```

Kmit

```python
# Convolution function
def convolve(image, kernel, padding=0):
    # Add padding to the image if needed
    if padding > 0:
        image = np.pad(image, ((padding, padding), (padding, padding)), mode='constant')
    #
    # Dimensions of the image and kernel
    image_height, image_width = image.shape
    kernel_height, kernel_width = kernel.shape

    # Calculate output dimensions
    output_height = image_height - kernel_height + 1
    output_width = image_width - kernel_width + 1

    # Initialize the output matrix
    output = np.zeros((output_height, output_width), dtype=int)

    # Perform convolution
    for i in range(output_height):
        for j in range(output_width):
            region = image[i:i+kernel_height, j:j+kernel_width]
            output[i, j] = np.sum(region * kernel)

    return output
```

5.Convolution Operation Without padding

compute the **convolution** of the given 5×5 image with the 3×3 filter (kernel) **without padding**.

```
# Without padding
output_without_padding = convolve(image, filter_kernel, padding=0)
print("Output without padding:")
print(output_without_padding)
```

6.Convolution Operation With padding of 1

```
# With padding of 1
output_with_padding = convolve(image, filter_kernel, padding=1)
print("\nOutput with padding:")
print(output_with_padding)
```