Capas de Pooling

Explicación de las Capas de Pooling

Objetivo de las Capas de Pooling

Las capas de pooling en las redes neuronales convolucionales (CNN) tienen como objetivo reducir las dimensiones espaciales (ancho y alto) de los mapas de características, manteniendo la información más relevante. Esto ayuda a disminuir el número de parámetros y la carga computacional del modelo, además de controlar el sobreajuste.

Tipos de Pooling

1. **Max Pooling**:
   - Selecciona el valor máximo en cada región de la imagen.
   - Ayuda a resaltar las características más prominentes.
   - Ejemplo:

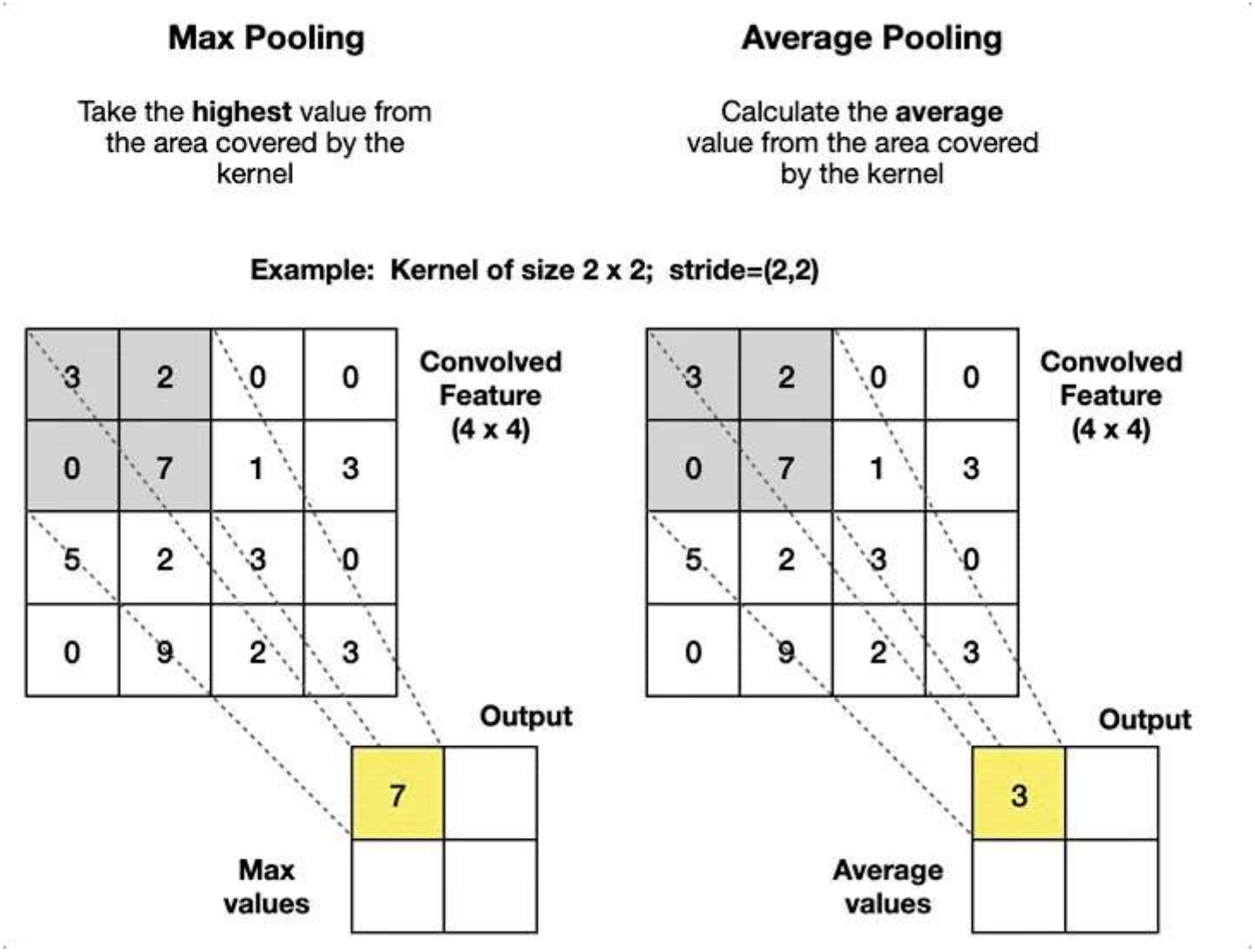
     ```
     Entrada (2x2):
     1  3
     2  4

     Max Pooling: 4
     ```

2. **Average Pooling**:
   - Calcula el promedio de los valores en cada región de la imagen.
   - Mantiene una representación más suave de la imagen.
   - Ejemplo:

     ```
     Entrada (2x2):
     1  3
     2  4

     Average Pooling: (1+3+2+4) / 4 = 2.5
     ```



Ventajas de las Capas de Pooling

1. **Reducción de Dimensiones**:
   - Reduce el tamaño de los mapas de características, lo que disminuye el número de parámetros y la complejidad computacional.
2. **Control del Sobreajuste**:
   - Al reducir el número de parámetros, se disminuye el riesgo de sobreajuste, lo que mejora la generalización del modelo.

3. **Invariancia a Traslaciones**:

    - Las capas de pooling proporcionan una mayor invariancia a pequeñas traslaciones en la entrada, lo que significa que el modelo puede reconocer características similares independientemente de su ubicación exacta en la imagen.

Ejemplo de Implementación en Python

In [10]:

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Definir la imagen de entrada
input_image = np.array([
    [1, 3, 2, 1],
    [4, 6, 6, 5],
    [7, 8, 9, 2],
    [1, 2, 3, 0]
])

# Definir la función de max pooling
def max_pooling(image, size=2, stride=2):
    h, w = image.shape
    output_h = (h - size) // stride + 1
    output_w = (w - size) // stride + 1
    output = np.zeros((output_h, output_w))

    for i in range(0, h - size + 1, stride):
        for j in range(0, w - size + 1, stride):
            output[i // stride, j // stride] = np.max(image[i:i+size, j:j+size])

    return output

# Aplicar max pooling a la imagen de entrada
output_image = max_pooling(input_image)

# Visualizar la imagen de entrada y la imagen de salida
fig, axs = plt.subplots(1, 2, figsize=(10, 5))

sns.heatmap(input_image, annot=True, fmt="d", cmap="Blues", ax=axs[0])
axs[0].set_title('Imagen de Entrada')

sns.heatmap(output_image, annot=True, fmt=".1f", cmap="Blues", ax=axs[1])
axs[1].set_title('Imagen de Salida (Max Pooling)')

plt.show()
```

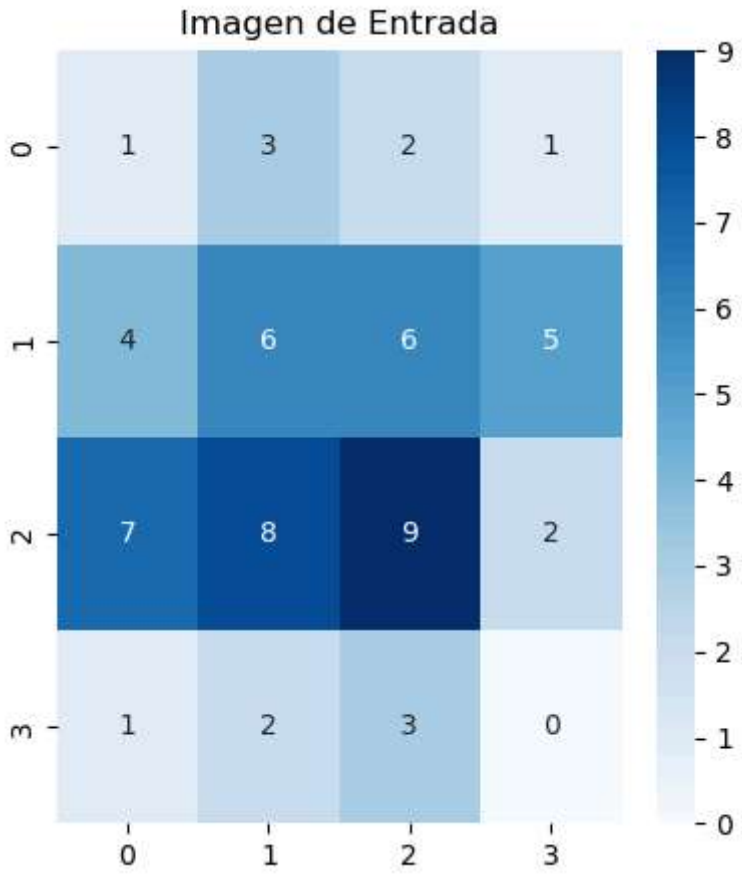


Imagen de Entrada
0
1
2
3
1 3 2 1
4 6 6 5
7 8 9 2
1 2 3 0
0 1 2 3
9 8 7 6 5 4 3 2 1 0

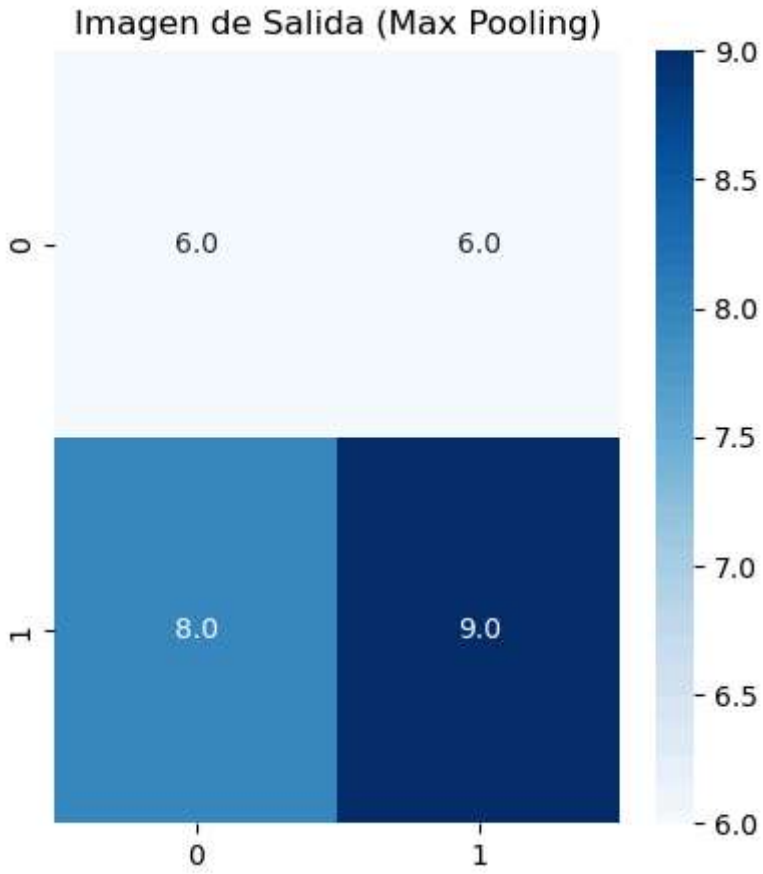


Imagen de Salida (Max Pooling)
0
1
6.0 6.0
8.0 9.0
0 1
9.0 8.5 8.0 7.5 7.0 6.5 6.0

In [ ]: