



Tecnológico de Monterrey

Tarea 1: Aprender sobre CNNs

Materia:

Inteligencia artificial avanzada para la ciencia de datos II (Gpo 101)

Alumno

Gilberto Ramos Salinas

A01734128

Fecha

23 de Septiembre del 2022

Primera Parte

Para el desarrollo de la tarea me di el objetivo de implementar un código que pudiera dar la respuestas a través del corrimiento de la matriz imagen con el kernel. Esto se hace, por que la convolución es definida como la integral del producto de dos funciones después de que una se ajusta.

A continuación se mostrarán diferentes métodos para resolverlo:

	1	0	0	2
f :	1	4	1	0
	0	3	2	0
	4	0	0	5

g :	2	1	6
	1	0	0
	5	0	3

Valid Convolution:

```
def validMethod(image, kernel, bias):  
    # El tamaño de la imagen se ajusta de acuerdo al tamaño del kernel  
    image_height, image_width = image.shape  
    kernel_height, kernel_width = kernel.shape  
    # Establecer la altura de salida  
    # En este caso los datos dados definen una matriz 4x4, por lo que la salida será 2x2 para mantener el  
    mismo tamaño  
    # con la matriz original y devuelve una convolución válida  
    output_height = image_height - kernel_height + 1  
    output_width = image_width - kernel_width + 1  
    # Inicializar salida  
    output = np.zeros((output_height, output_width))  
    # Loop a través de la salida  
    for y in range(output_height):  
        for x in range(output_width):  
            # multiplicación elemento a elemento del kernel y la imagen  
            output[y,x] = (kernel * image[y: y+kernel_height, x: x+kernel_width]).sum()  
    # Agregar sesgo  
    output += bias  
    # Devolver salida  
    return output
```

```
valid Convolution
[[ 9. 31.]
 [32. 27.]]
```

Full Convolution:

```
def fullMethod(image, kernel, bias):
```

```
    # El tamaño de la imagen se ajusta de acuerdo al tamaño del kernel
```

```
    image_height, image_width = image.shape
```

```
    kernel_height, kernel_width = kernel.shape
```

```
    output_height = image_height + kernel_height - 1
```

```
    output_width = image_width + kernel_width - 1
```

```
    # Establecer la altura de salida
```

```
    # En este caso los datos dados definen una matriz 4x4, por lo que la salida será 6x6 para mantener el mismo tamaño
```

```
    #con la matriz original y devuelve una convolución completa
```

```
    # Inicializar salida
```

```
    output = np.zeros((output_height, output_width))
```

```
    # Loop a través de la salida
```

```
    #ajustar forma de la imagen
```

```
    image = np.pad(image, ((kernel_height-1, kernel_height-1), (kernel_width-1, kernel_width-1)),
'constant')
```

```
    for y in range(output_height):
```

```
        for x in range(output_width):
```

```
            # multiplicación elemento a elemento del kernel y la imagen
```

```
            output[y,x] = (kernel * image[y: y+kernel_height, x: x+kernel_width]).sum()
```

```
    # Agregar sesgo
```

```
    output += bias
```

```
    # Devolver salida
```

```
    return output
```

```
full Convolution
[[ 3.  0.  5.  6.  0. 10.]
 [ 3. 12.  9. 20.  5.  2.]
 [ 6. 10.  9. 31. 13.  4.]
 [18. 25. 32. 27.  4. 25.]
 [ 0. 18. 19.  8.  4.  5.]
 [24.  4.  8. 30.  5. 10.]]
-----
```

Segunda Parte

La vision computacional es una herramienta novedosa que se ha estado implementando a lo largo de los años en el mundo de ciencias de datos, realmente encuentro interesante como las convoluciones juegan un papel importante en esta área, al extraer datos de una imagen al procesarla desde varias perspectivas. Este conocimiento nos permite definir un set mas amplio de parámetros para entrenar una red neuronal y con ello brindar una respuesta interesante como lo fue en el caso de Ninja Warrior.