In [1]:
```
import numpy as np
```

In [2]:
```
def ReLU(z):
    return np.maximum(z,0)

def d_ReLU(z):
    return 1.0 * (z > 0)

def sigmoid(z):
    return 1.0/(1.0 + np.exp(-z))

def d_sigmod(z):
    return sigmoid(z)*(1-sigmoid(z))

# Using log10
def binary_cross_entropy(y_hat, y_pred):
    return -1*(y_hat*np.log10(y_pred) + (1-y_hat)*np.log10(1-y_pred))

def d_binary_cross_entrpoy(y_hat, y_pred):
    return -1*(y_hat/y_pred - (1-y_hat)/(1-y_pred))

def max_pool(input, kernel, stride):
    z = np.zeros(((input.shape[0] - kernel) // stride + 1 , (input.shape[1]
    idx = []

    for x in range(z.shape[0]):
        for y in range(z.shape[1]):
            i = x*kernel
            j = y*kernel
            block = input[i : i+kernel, j : j+kernel]
            z[x,y] = np.max(block)
            index = np.add(np.unravel_index(block.argmax(), block.shape), (i
            idx.append(index)
    return z, idx

# Not using stride or padding for simplicity
def convolution(input, kernel, bias):
    output = np.zeros((input.shape[0] - kernel.shape[0] + 1, input.shape[1]
    for i in range(output.shape[0]):
        for j in range(output.shape[1]):
            output[i, j] = np.sum(input[i:i+kernel.shape[0], j:j+kernel.shap
    return output

# Simple visualization function
def printMatrix(A, text):
    print(text)
    print(A)
    print("---")
```

Forward pass

In [3]:
```
input = np.array([[1,2,0,2,1],
                  [0,1,1,2,1],
                  [1,2,0,1,0],
                  [1,2,1,2,1],
                  [0,0,3,2,3]])

kernel = np.array([[1,-1],
                   [-1,1]])
bias_conv = 1.0
```

```
printMatrix(input, "Input image")
printMatrix(kernel, "Kernel")
```

```
Input image
[[1 2 0 2 1]
 [0 1 1 2 1]
 [1 2 0 1 0]
 [1 2 1 2 1]
 [0 0 3 2 3]]
---
Kernel
[[ 1 -1]
 [-1  1]]
---
```

In [4]:
```
z_conv = convolution(input, kernel, bias_conv)

a_relu = ReLU(z_conv)

a_max_pool, idx_max = max_pool(a_relu, kernel.shape[0], stride=2)

a_flatten = a_max_pool.flatten()

printMatrix(z_conv, "Conv")
printMatrix(a_relu, "ReLU")
printMatrix(a_max_pool, "Max Pool")
printMatrix(a_flatten.reshape(1,4), "Flatten")
```

```
Conv
[[ 1.  3.  0.  1.]
 [ 1. -1.  1.  1.]
 [ 1.  2.  1.  1.]
 [ 0.  5. -1.  3.]]
---
ReLU
[[1. 3. 0. 1.]
 [1. 0. 1. 1.]
 [1. 2. 1. 1.]
 [0. 5. 0. 3.]]
---
Max Pool
[[3. 1.]
 [5. 3.]]
---
Flatten
[[3. 1. 5. 3.]]
---
```

Fully Connected layer

In [5]:
```
weight_fcnn = np.array([1, 1, -1, 1])
bias_fcnn = 0

z_fcnn = weight_fcnn.dot(a_flatten) + bias_fcnn
a_fcnn = sigmoid(z_fcnn)
loss = binary_cross_entropy(1, a_fcnn)
print(f"Activation FCNN: {a_fcnn}")
print(f"Binary cross-entropy loss: {loss}")
```

```
Activation FCNN: 0.8807970779778823
Binary cross-entropy loss: 0.05512413479491803
```

In [6]:
```
printMatrix(bias_conv, "Bias Conv")
```

```
printMatrix(kernel, "Kernel")
printMatrix(weight_fcnn, "Weight FCNN")
printMatrix(bias_fcnn, "Bias FCNN")
```

```
Bias Conv
1.0
---
Kernel
[[ 1 -1]
 [-1  1]]
---
Weight FCNN
[ 1  1 -1  1]
---
Bias FCNN
0
---
```

Backward Pass

In [7]:
```
learning_rate = 0.1
```

In [8]:
```
delta = d_binary_cross_entrpoy(1, a_fcnn) * d_sigmod(z_fcnn)
print(f"dL / d(bias_fcnn) = {delta}")
bias_fcnn = bias_fcnn - learning_rate * delta

delta = delta * a_flatten
print(f"dL / d(weight_fcnn) = {delta}")
weight_fcnn = weight_fcnn - learning_rate * delta

delta = delta.reshape(a_max_pool.shape)
printMatrix(delta, "Reshaped derivative")
```

```
dL / d(bias_fcnn) = -0.11920292202211769
dL / d(weight_fcnn) = [-0.35760877 -0.11920292 -0.59601461 -0.35760877]
Reshaped derivative
[[-0.35760877 -0.11920292]
 [-0.59601461 -0.35760877]]
---
```

In [9]:
```
calculate_delta = np.zeros(a_relu.shape)

for idx, grad in zip(idx_max, delta.flatten()):
    calculate_delta[idx[0],idx[1]] = grad

delta = calculate_delta
printMatrix(a_relu, "a_relu")
printMatrix(delta, "dL/d(a_relu)")

delta = d_ReLU(z_conv) * delta
printMatrix(delta, "dL/d(z_conv)")

grad_b1 = delta.sum()
print(f"dL / d(grad_b1) = {grad_b1}")
bias_conv = bias_conv - learning_rate * grad_b1
```

```
a_relu
[[1. 3. 0. 1.]
 [1. 0. 1. 1.]
 [1. 2. 1. 1.]
 [0. 5. 0. 3.]]
---
dL/d(a_relu)
[[ 0.         -0.35760877  0.         -0.11920292]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.         -0.59601461  0.         -0.35760877]]
---
dL/d(z_conv)
[[ 0.         -0.35760877  0.         -0.11920292]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.         -0.59601461  0.         -0.35760877]]
---
dL / d(grad_b1) = -1.4304350642654122
```

In [10]:

```
grad_kernel = np.zeros(kernel.shape)

grad_kernel[0,0] = np.sum(input[0:4, 0:4] * delta)
grad_kernel[0,1] = np.sum(input[0:4, 1:5] * delta)
grad_kernel[1,0] = np.sum(input[1:5, 0:4] * delta)
grad_kernel[1,1] = np.sum(input[1:5, 1:5] * delta)

kernel = kernel - learning_rate * grad_kernel
printMatrix(grad_kernel, "dL/d(grad_kernel)")
```

```
dL/d(grad_kernel)
[[-2.86087013 -1.0728263 ]
 [-1.31123214 -3.33768182]]
---
```

Updated values

In [11]:

```
printMatrix(weight_fcnn, "Updated Weight FCNN")
printMatrix(bias_fcnn, "Updated Bias FCNN")
printMatrix(kernel, "Updated Kernel")
printMatrix(bias_conv, "Updated Bias Conv")
```

```
Updated Weight FCNN
[ 1.03576088  1.01192029 -0.94039854  1.03576088]
---
Updated Bias FCNN
0.011920292202211769
---
Updated Kernel
[[ 1.28608701 -0.89271737]
 [-0.86887679  1.33376818]]
---
Updated Bias Conv
1.1430435064265412
---
```