Digital ASIC Design Lab (EEP7120)

Name: - Suraj Prajapati

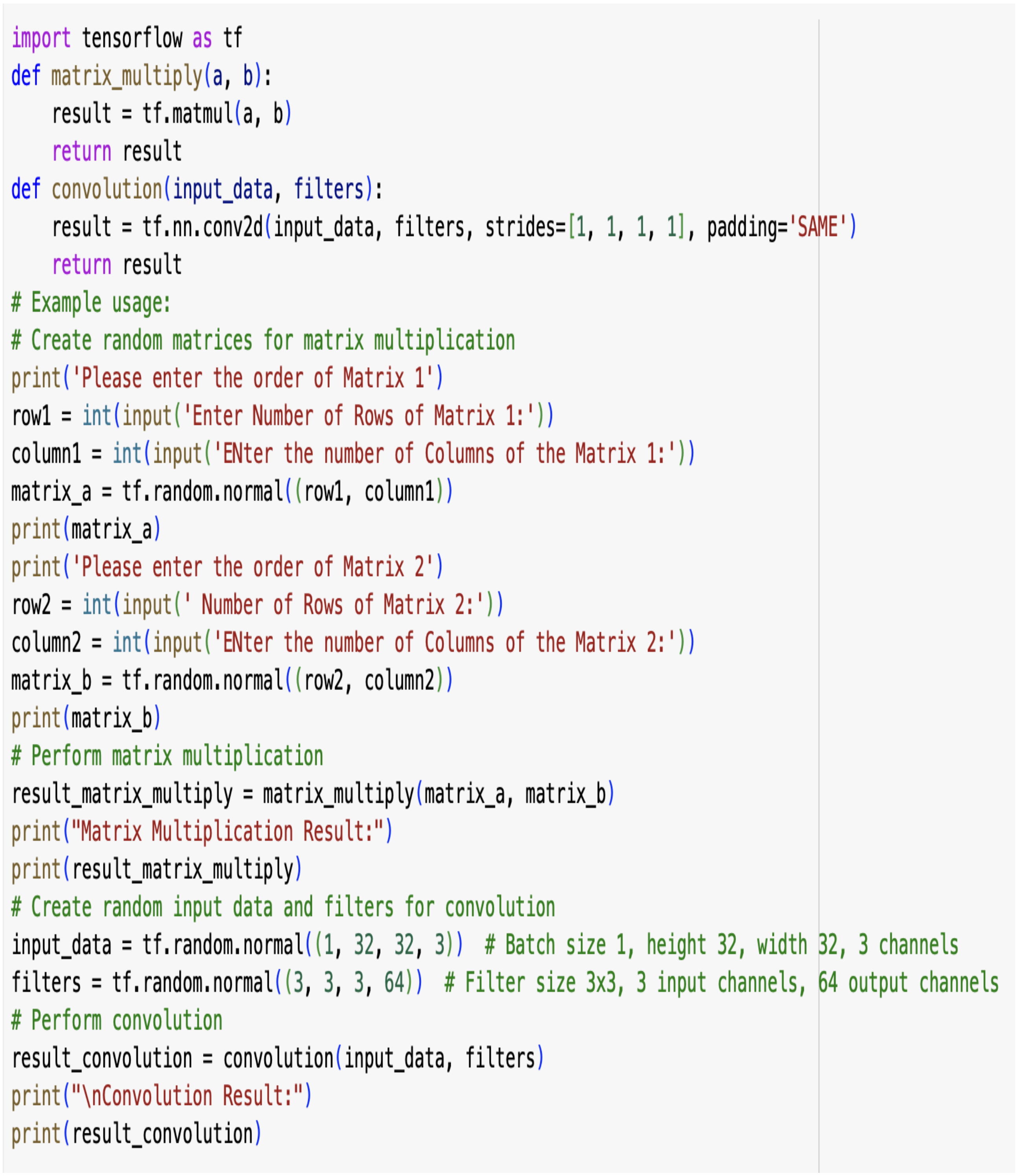
Roll Number: -M23EEV019

Branch: - MTech [Intelligent VLSI System]

Guided By: - Binod Kumar

Project Work: - Design of a specific Deep Learning ALU (DLALU) that does matrix multiplication and convolution in a generalized manner (i.e., can work for any size of inputs)

CODE:



```python
import tensorflow as tf
def matrix_multiply(a, b):
    result = tf.matmul(a, b)
    return result
def convolution(input_data, filters):
    result = tf.nn.conv2d(input_data, filters, strides=[1, 1, 1, 1], padding='SAME')
    return result
# Example usage:
# Create random matrices for matrix multiplication
print('Please enter the order of Matrix 1')
row1 = int(input('Enter Number of Rows of Matrix 1:'))
column1 = int(input('ENter the number of Columns of the Matrix 1:'))
matrix_a = tf.random.normal((row1, column1))
print(matrix_a)
print('Please enter the order of Matrix 2')
row2 = int(input(' Number of Rows of Matrix 2:'))
column2 = int(input('ENter the number of Columns of the Matrix 2:'))
matrix_b = tf.random.normal((row2, column2))
print(matrix_b)
# Perform matrix multiplication
result_matrix_multiply = matrix_multiply(matrix_a, matrix_b)
print("Matrix Multiplication Result:")
print(result_matrix_multiply)
# Create random input data and filters for convolution
input_data = tf.random.normal((1, 32, 32, 3))  # Batch size 1, height 32, width 32, 3 channels
filters = tf.random.normal((3, 3, 3, 64))  # Filter size 3x3, 3 input channels, 64 output channels
# Perform convolution
result_convolution = convolution(input_data, filters)
print("\nConvolution Result:")
print(result_convolution)
```

Output

```
Please enter the order of Matrix 1
Enter Number of Rows of Matrix 1:4
Enter the number of Columns of the Matrix 1:4
tf.Tensor(
[[-1.9550899  -0.6615045  -0.17301504 -1.4537385 ]
 [ 0.9602983   0.53852403  1.2942957  -1.1768743 ]
 [-0.6322282   1.1238395  -0.7924927  -0.3326921 ]
 [-0.9217373   0.9921181  -1.2199097  -0.5794317 ]], shape=(4, 4), dtype=float32)
Please enter the order of Matrix 2
 Number of Rows of Matrix 2:4
ENter the number of Columns of the Matrix 2:4
tf.Tensor(
[[ 0.39186504 -0.10162803 -0.30515978  0.0719901 ]
 [ 0.29991332 -0.16130887  0.65133756  0.5462831 ]
 [ 2.959181    2.9364083   1.6853344   0.61103547]
 [ 0.3937203   0.47612345 -0.9265713  -2.1125429 ]], shape=(4, 4), dtype=float32)
Matrix Multiplication Result:
tf.Tensor(
[[-2.0488746  -0.8948033   1.2211561   2.4632506 ]
 [ 3.9045138   3.0557811   3.3294954   3.6403766 ]
 [-2.3868105  -2.6025178  -0.10242283  0.7870056 ]
 [-3.9017148  -3.924397   -0.5915899   0.9542877 ]], shape=(4, 4), dtype=float32)

Convolution Result:
tf.Tensor(
[[[[  1.4157667    0.63428587  -4.730832   ...  -3.0514514
      5.511379     1.5699707 ]
   [ -1.7081138    0.49599886   1.1604731  ...   5.3561473
     -1.397914     0.17930993]
   [ -0.44528294  -4.9298196   -1.2798276  ...  -1.9306091
      2.6491084    5.0893145 ]
   ...
   [  6.7958245    7.344742     7.8486333  ...  -3.4401426
      5.3059645   -3.769426  ]
   [  6.857298     2.6103785   -3.9850779  ...   0.69314
     -7.5260787   -2.7760303 ]
   [ -3.0515106   -5.997098     0.23167624 ...   9.059984
     -3.407778     5.064493  ]]

  [[  5.842323    -2.1220953   -2.715623   ...  -5.7332306
      4.7317057    6.020149  ]
   [  8.206702    -4.16124      3.456823   ...  -0.74585027
     -9.049442     0.9108057 ]
   [ -1.3780054    5.6948414   -5.589194   ...   1.2058738
      1.9555043    2.167886  ]
   ...
   [ 11.135162    13.113119     3.2825618  ...  -6.7004414
      8.191703    -3.0277212 ]
   [ 14.593546     1.186505    -9.159704   ...  -5.1622267
      3.331447    -6.3948426 ]
   [  2.6753535   -3.086483    -8.926995   ...  -2.4411125
     -2.2164295    2.6593626 ]]

  [[ -7.376831     6.4107223    1.4965842  ...  -6.3741717
      8.681341    -5.632304  ]
[  9.6307535   -5.1902294    4.1505766  ...   1.9031935
  -5.676099     3.0087845 ]
 [ -8.284287     5.7168503   -1.8960723  ...  -3.608437
   8.262001    -4.3654556 ]
 ...
 [ 11.678753     3.183202    -2.85781     ...  -1.0348102
  -1.295573     6.1932755 ]
 [ -0.21368839   8.846865   -13.257348   ... -14.465094
  13.491926   -15.335605  ]
 [  4.7172756   -2.681489     0.5073908  ...   1.1452614
   8.996224     2.2337408 ]]

...

[[  4.0120783    8.532055    -7.1371994  ...  -1.0148253
  -2.9946225   -2.9118059 ]
 [  4.8362203   -3.0868409    5.8659654  ...   2.2293587
   2.7457669    7.63498   ]
 [ -6.7131805   -1.7391285   -5.690233   ...  -6.558873
   9.773755    -1.1803416 ]
 ...
 [ -1.8073671   -0.40979773   0.20222338 ...   9.624478
 -11.798201     3.3919969 ]
 [-12.528884     6.221901    -2.0782313  ...   5.708394
  -0.92647064  -4.9351997 ]
 [ -4.6459107   -4.9436607    1.2817852  ...   1.5287503
   1.7302891   -1.7014318 ]]

[[  3.8525405   -3.5362256   -2.5923262  ...  -6.819538
   5.6194444    1.1395267 ]
 [  3.0405612   -6.6915507   -0.78557676 ...  -0.04374173
  -1.9726145    1.4684846 ]
 [  1.6982852   -0.24714664  -2.3925369  ...  -0.18630348
   5.302333     6.246687  ]
 ...
 [  5.141237    -4.8669944   -5.1044497  ...   0.12844898
   0.75085795   2.025838  ]
 [ -2.7329812   -3.7322478    0.65695786 ...   6.6781845
  -2.773612     4.518417  ]
 [ -8.430515    -2.0513887   -3.8302348  ...   6.696033
   2.1617987   -2.9271686 ]]

[[  6.065751    -3.917849     0.06232031 ...  -5.974893
  -1.6196606    1.8861483 ]
 [  4.7385406   -2.504249    -2.8209991  ...  -1.2568916
  -1.436336     5.2392473 ]
 [ -4.302264     4.1093316   -1.7903796  ...   3.2138712
  -1.6977818    0.6564146 ]
 ...
 [ -4.5799074    0.07154182  -2.8630593  ...  -3.893609
   4.0565104   -3.080218  ]
 [ -0.32320136  -4.964287     5.410898   ...   7.143473
  -3.1782925    4.6919527 ]
 [ -6.0203304    1.6232651    1.2760688  ...   6.1563864
  -1.4465138   -0.17733559]]]], shape=(1, 32, 32, 64), dtype=float32)
```

Explanation of this code:

```
import tensorflow as tf
```

This line imports the TensorFlow library and gives it the alias tf. This is a common convention in the TensorFlow community to make the code shorter and more readable. Now, after executing this line, you have access to the TensorFlow library and its functionalities through the tf alias.

```
def matrix_multiply(a, b):
    result = tf.matmul(a, b)
    return result
```

1. matrix_multiply is the name of the function, and it takes two arguments a and b. These are assumed to be matrices (two-dimensional tensors).
2. tf.matmul(a, b) is a TensorFlow function that performs matrix multiplication between matrices a and b. Matrix multiplication is a mathematical operation where each element of the resulting matrix is obtained by multiplying elements of a row from the first matrix with corresponding elements of a column from the second matrix and summing up the products.
3. The result of the matrix multiplication is stored in the variable result.
4. Finally, the function returns the result of the matrix multiplication

```
def convolution(input_data, filters):
result = tf.nn.conv2d(input_data, filters, strides=[1, 1, 1, 1], padding='SAME')
return result
```

This code defines a convolution function using TensorFlow. The function takes two parameters: `input_data` (the input tensor or image) and `filters` (the convolutional filters or kernels). It then applies a 2D convolution operation using `tf.nn.conv2d` with a stride of 1 in all dimensions and 'SAME' padding. The result of the convolution operation is returned. This function is commonly used in convolutional neural networks for feature extraction from input data using convolutional filters.

```
matrix_a = tf.random.normal((3, 4))
matrix_b = tf.random.normal((4, 5))
result_matrix_multiply = matrix_multiply(matrix_a, matrix_b)
print("Matrix Multiplication Result:")
print(result_matrix_multiply)
```

This code uses TensorFlow to perform matrix multiplication. It generates two random matrices, `matrix_a` with shape (3, 4) and `matrix_b` with shape (4, 5). Then, it calls a function `matrix_multiply` (which is assumed to be defined elsewhere) to multiply these matrices. Finally, it prints the result of the matrix multiplication. Note that the specific implementation of the `matrix_multiply` function is not provided in the given code snippet.

```
input_data = tf.random.normal((1, 32, 32, 3))
filters = tf.random.normal((3, 3, 3, 64))
result_convolution = convolution(input_data, filters)
print("\nConvolution Result:")
print(result_convolution)
```

This code appears to be a snippet using TensorFlow, a popular deep learning library in Python. The code is performing a convolution operation on a randomly generated 4-dimensional tensor (`input_data`) with a set of randomly generated filters (`filters`).

Here's a breakdown:

1. **Import TensorFlow**: The code assumes that TensorFlow is imported as `tf`.

2. **Generate Random Input Data**: `input_data` is a randomly generated tensor of shape `(1, 32, 32, 3)`. This is likely an image with dimensions 32x32 pixels and 3 color channels (RGB).

3. **Generate Random Filters**: `filters` is a randomly generated tensor of shape `(3, 3, 3, 64)`. This represents a set of 64 filters, each with a size of 3x3 and 3 input channels. The 64 filters are expected to produce 64 output channels.

4. **Perform Convolution Operation**: The `convolution` function is assumed to be defined elsewhere in the code or imported from a library like TensorFlow. This function takes the input data (`input_data`) and filters (`filters`) and performs a convolution operation.

5. **Print Convolution Result**: The result of the convolution operation is stored in `result_convolution`, and it is printed to the console.

Note: The code assumes the existence of a function `convolution` that is not provided. This function is presumably responsible for applying the convolution operation using the input data and filters. The details of this function would determine the specific type of convolution (e.g., valid or same padding, strides, etc.).