Kartik kanotra

KK5243

**Performance Study of a MNIST CNN 2D Layer**

**MEMORY**

Calculated:

Manually calculated value for Conv2d will include sum of two memories:

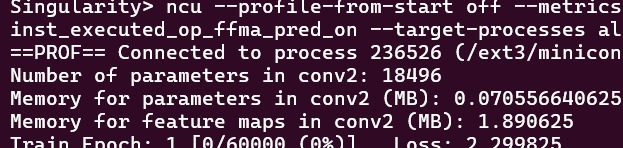
1. Fixed Memory taken by Parameters (Weights , Bias ,Kernal size)
2. Memory taken by Feature Maps

Space taken by Parameters can be calculated as :

**Memory (in bytes)=(Number of Filters∗Kernel Width∗Kernel Height∗Number of Input Channels+Number of Filters)∗4**

Space taken by memory maps can be calculated as :

**Activation Memory = (batch\_size × channels\_out × height\_out × width\_out) × 4**

****

**Comparing the calculated and actual values**

**BATCH SIZE ESTIMATED ACTUAL**

**16 9.76 4.50**

**32 17.36 9.44**

**64 32.48 18**

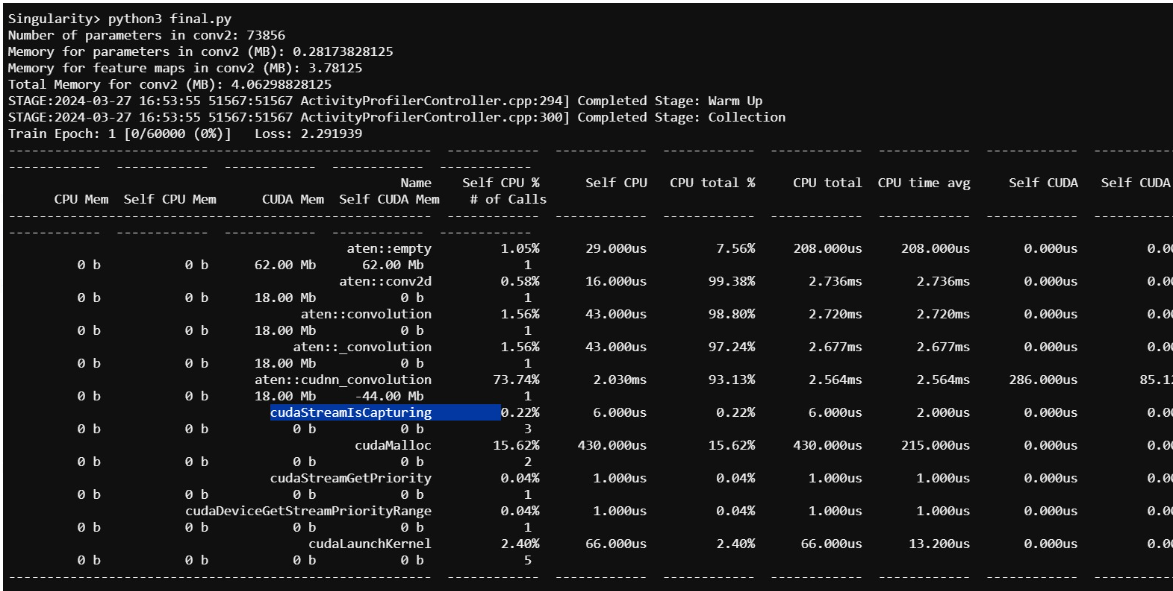
**128 62.752 36**

**256 123.2 72**

**512 244 144**

**ACTUAL MEMORY**

It is calculated by running the attached code main2.py



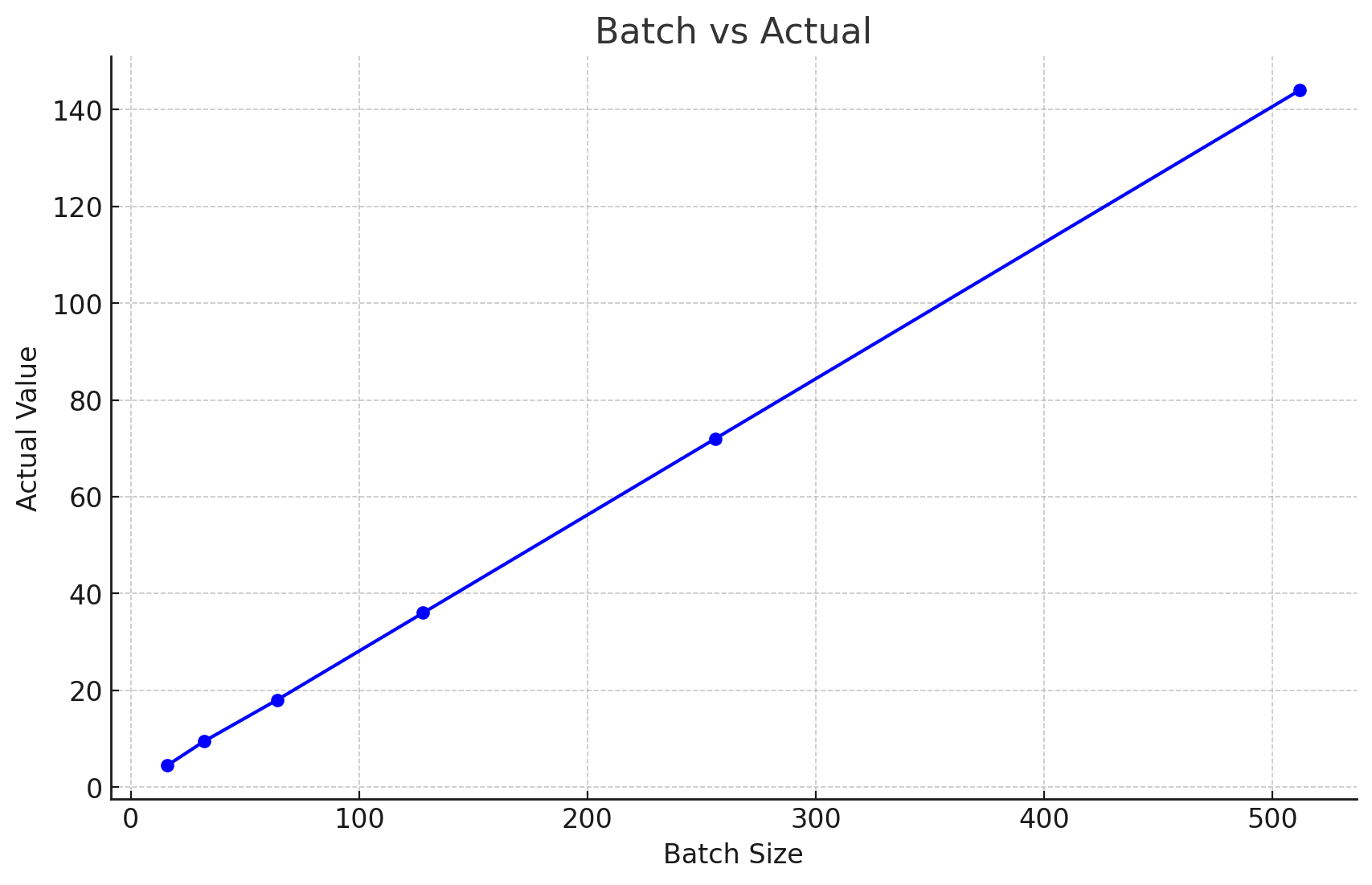
There is a difference between the estimated and actual value is because of the optimizations done by pytorch and gpu , like it may do some in place calculations instead of using different spaces for each calculation.

From the provided data, it's apparent that the space complexity of our implemented CNN Model is lower than what is predicted by theoretical methods (using equations). This discrepancy can be attributed to several factors:

a) Theoretical models typically assume each parameter and activation to have its own dedicated memory space. However, practical implementations can see a reduction in actual memory usage thanks to optimization techniques like in-place operations.

b) During operations, particularly in the inference phase where gradients are unnecessary, the model may employ techniques to share memory among certain parameters or operations.

c) Theoretical estimations often overlook the efficiency gains made possible through optimizations at both the PyTorch framework and hardware level, which can lead to less memory consumption.

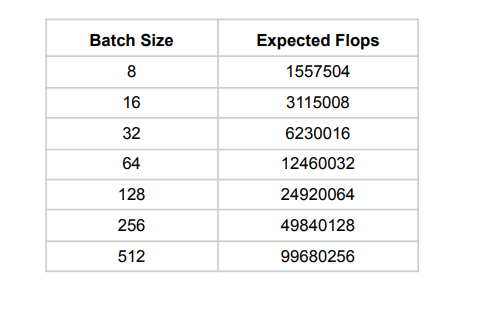


**TIME COMPLEXITY**

**Theoretical calculation (Estimated Values)**

To estimate the time complexity of the second convolutional layer (**conv2**) in the neural network defined in the provided code, we need to **identify** and **multiply** the following parameters:

* �*N*: Batch size (given as 64)
* ����*Cout*​: Number of output channels (filters)
* ����*Hout*​ and ����*Wout*​: Height and width of the output feature map, respectively
* ���*Cin*​: Number of input channels
* �*K*: Kernel size

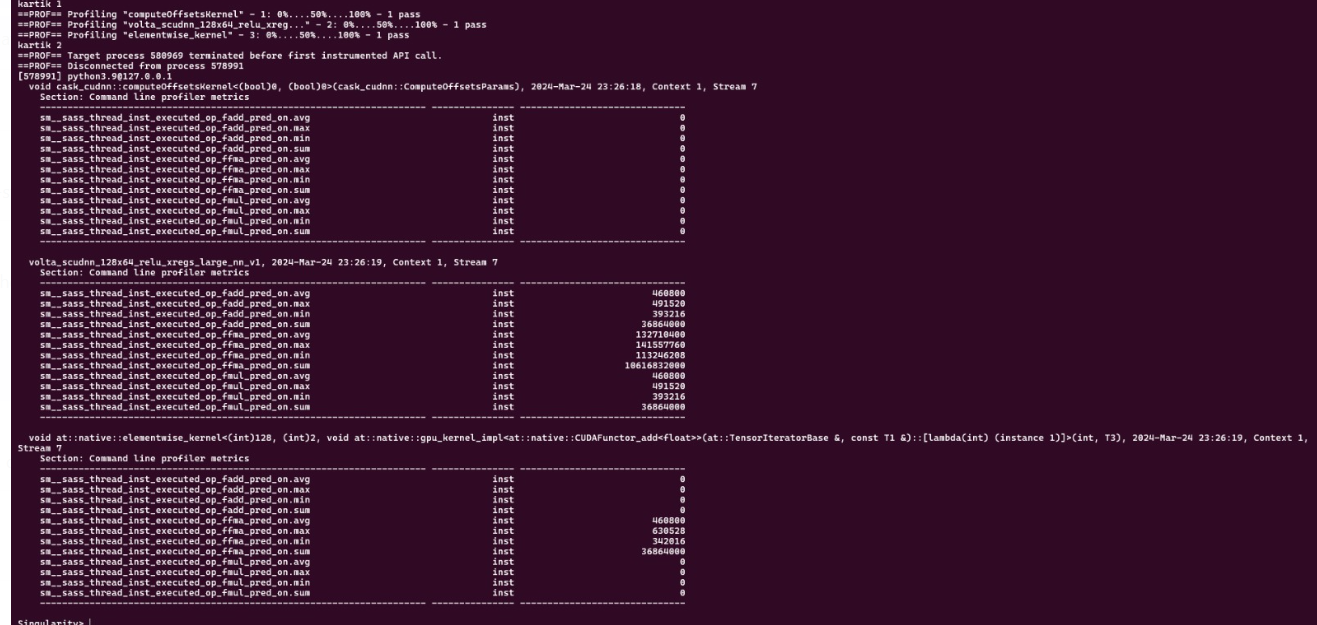


**Actual Value**

Attached file main1.png on running this command on terminal gives the result

In the current scenario, we have the following input parameters and based on them we will calculate the flops for different batch sizes. 1. Current batch size:- 64 2. Image size - 28 \* 28 3. Filter size - 3 4. Stride - 1 5. Conv2D layer 1 - (1, 32, 3, 1) 6. Conv2D layer 2 - (32, 64, 3, 1) 7. Fully connected layer 1 - (9216, 128) 8. Fully connected layer 2 - (128, 10)

ncu --profile-from-start off --metrics sm\_sass\_thread\_inst\_executed\_op\_fadd\_pred\_on,smsass\_thread\_inst\_executed\_op\_fmul\_pred\_on,sm\_sass\_thread\_inst\_executed\_op\_ffma\_pred\_on --target-processes all python3 main.py --batch-size 64 --epochs 1 --dry-run



For fadd (addition): 4696080

For fmul (multiplication): 49152

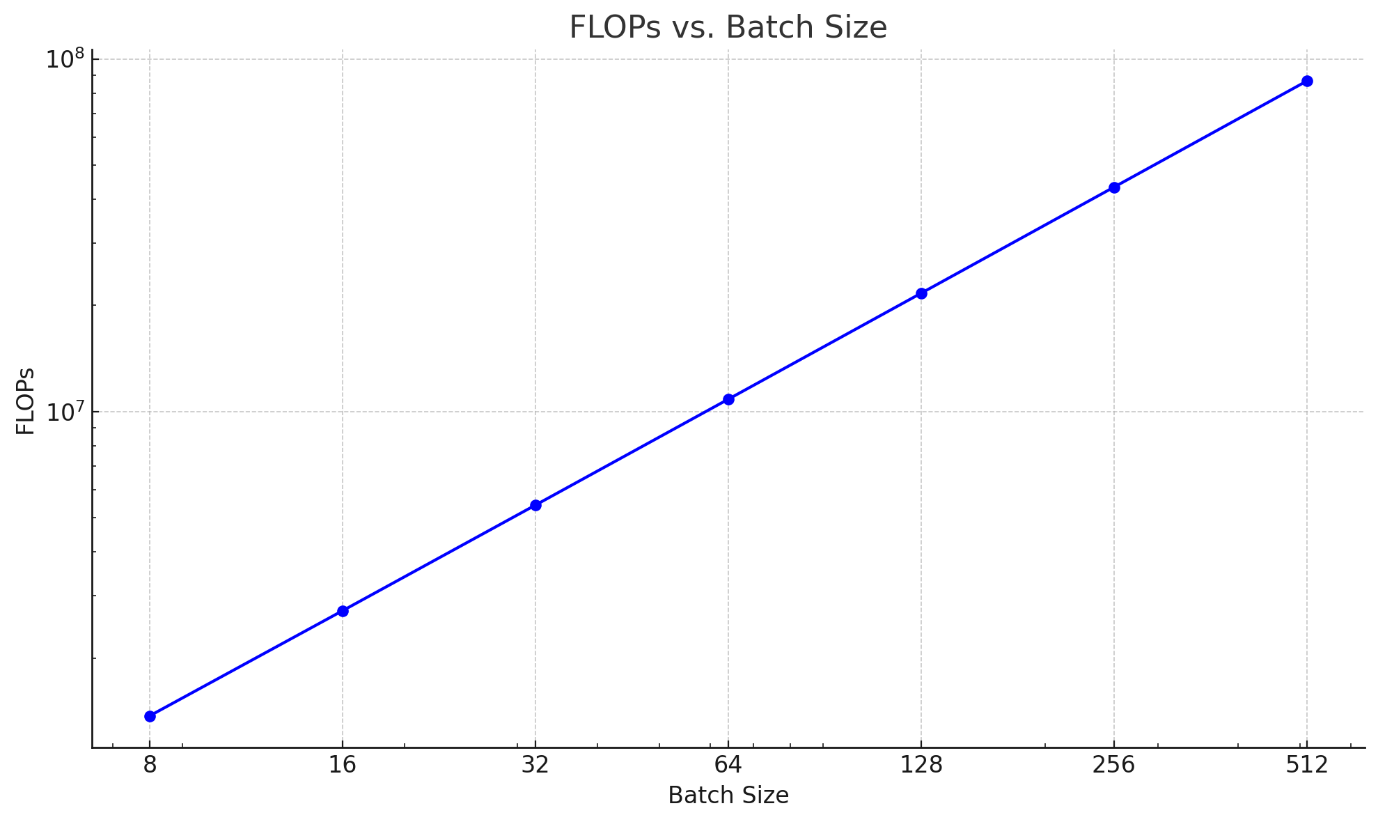
For ffma (fused multiply-add): 3689316

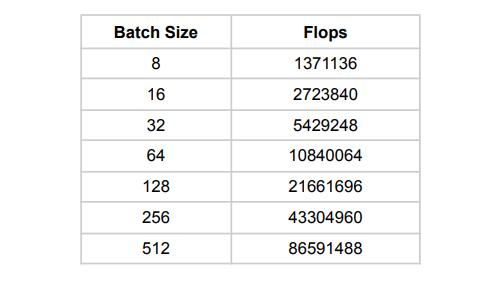
And the formula to calculate flop\_count\_sp is:

flop\_count\_sp = (fadd + fmul + 2 \* ffma)

flop\_count\_sp = 4696080 (fadd) + 49152 (fmul) + 2 \* 3689316 (ffma)

The calculated flop\_count\_sp using the metrics from the NSight Compute output is 12,123,864 single-precision floating-point operations.



****