CSCI GA 3033 Cloud and Machine Learning
Homework 3: Performance study of (a small part of) Neural Network

Name: Aashiq Mohamed Baig
Net ID: amb1558

Introduction

In this report, we will be covering performance measures of Machine Learning scripts using a profiler. The 2 main metrics to measure are the time complexity via number of flops and space complexity via memory. We will be using Nvidia NCU to verify if our calculation for the flops via pen and paper is indicative of the true flops computed on the system for 1 epoch in a 2D convolution layer. Similarly, we will use PyTorch profiler to verify if the memory we calculated is indicative of the true memory used by the system. Using PyTorch profiler, we will extract logs for our Machine Learning job which we will further analyze in Tensorboard. We will write a python Machine Learning script which will run on the MNIST dataset; a dataset that has over 60,000 images of handwritten digits; and classify them into a digit from 0 to 9. There are 2 variations to our Python script; 1 for the flop calculation and one for the memory calculation i.e. NCU and PyTorch profiler.

MNIST Database

The MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. It 60,000 training images and 10,000 testing images. Each image is a grayscale image of size 28x28. Researchers have experimented with many different models and have achieved a near human level of accuracy with these models.

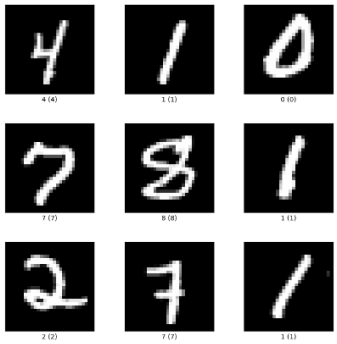


Fig 1. Sample of images from the MNIST database

Background

Before we dive into computing the time and space complexity, let us understand the terminology and the shorthand being used. Here is the representative code of the 2D convolution in the Neural Network. The 2D convolution is a simple operation; you start with a kernel, which is simply a small matrix of weights. This kernel "**slides**" over the 2D input data, performing an elementwise multiplication with the part of the input it is currently on, and then summing up the results into a single output pixel.

Conv1: nn.conv2d (1, 32, 3, 1)
Conv2: nn.conv2d (32, 64, 3, 1)

The order of the parameters passed is (input filters, output filters, kernel size, stride)
We will refer to the following parameters with the below shorthand

- input filters/channels as C_{in}
- output filters/channels as C_{out}
- kernel size as k
- stride as s

The input image size is 28x28x1. We will follow the output produced in both convolution steps. Additionally, we will be using H_{out} and W_{out} to represent the output height and width of the image after convolution.

Conv1: (1 x 28 x 28) -> (32 x 26 x 26)
Conv2: (32 x 26 x 26) -> (64 x 24 x 24)

For our calculations, we will be taking the 2nd convolution layer.

Time Complexity Pen and Paper Method

$$2 * k * k * Cin * Wout * Hout * batch\_size$$
$$= 2 * 3 * 3 * 32 * 24 * 24 * 64$$
$$= 21{,}233{,}664$$

In this method [1], we are trying to calculate the total number of additions and multiplications done to complete a single convolution. The entire number is multiplied by 2 to account for multiplications and additions since for a 3*3 kernel, we have 9 multiplications and 8 additions. This is approximated up to 2*9. The time complexity is indicated of the FLOPs measurement.

Space Complexity Pen and Paper Method

$$\big(Hout * Wout * Cout + \big((k * k * Cin) + 1\big) * Cout\big) * 4$$
$$= (Hout * Wout * Cout + Params) * 4$$
$$= (24 * 24 * 64 + 18{,}496) * 4$$
$$= 221{,}440$$

Some things to note in the following method [2]. The “Params” here refers to the weights that need to be stored for the kernels and which are used in the actual convolution. The first term is the output of the convolution. The entire sum is multiplied by 4 to account for each value being a floating-point number and each floating-point number is 4 bytes.
The approximate calculation for memory is 0.22 MB.

Machine Learning Model

In Fig 2, we can see the exact model operations being used to identify the images in the MNIST dataset. There are 6 operations in total. However, these are not the exact order of operation calls during forward propagation. This is how the operations are initialized in the init function of the Neural Network Model.

```
Net(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (dropout1): Dropout(p=0.25, inplace=False)
  (dropout2): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=9216, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=10, bias=True)
)
```

Fig 2. Machine Learning model for classifying images in MNIST

As we have seen above, conv1 and conv2 handle the actual convolutions. The dropout layer randomly sets some of the input units to 0 with a frequency of p (in Fig 2) which is 0.25 and 0.5 respectively. This is done during the training phase and it’s helps to prevent overfitting. Finally, we have 2 fully connected layers, a.k.a. linear layers, which convert the input if it is in 3D format to a 2D linear layer. This is exactly what happens in the first fully connected layer. The 2nd fully connected layer reduces the number of features. Here, we can see the 2nd layer reduces the number of features to 128.

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

Fig 3. Neural Network Class and functions for init and forward propagation

Fig 3 details the exact steps and order of all operations in the neural network. Note that there are 2 additional parameters here. We have a ReLU layer [3] which implements non-linearity and a 2D max pool layer [4] which reduces the dimensions of the matrix by 2. It does this to down sample a representation of the input such that we may still make assumptions about the features contained in the sub-regions binned [5].

Fig 4 shows us the summary of the model. It is achieved by calling summary from the “torchsummary” package. We specify the model and the dimensions of the input. It gives us a detailed view of the model, the resulting dimensions after each operation and the number of parameters required/learned/stored for that layer. It also returns the total number of trainable and non-trainable parameters as well as the size required to store the model outputs and parameters in memory/disk.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Conv2d: 1-1                            [-1, 32, 26, 26]          320
├─Conv2d: 1-2                            [-1, 64, 24, 24]          18,496
├─Dropout: 1-3                           [-1, 64, 12, 12]          --
├─Linear: 1-4                            [-1, 128]                 1,179,776
├─Dropout: 1-5                           [-1, 128]                 --
├─Linear: 1-6                            [-1, 10]                  1,290
==========================================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
Total mult-adds (M): 11.99
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.45
Params size (MB): 4.58
Estimated Total Size (MB): 5.03
==========================================================================================
```

Fig 4. Summary of the Machine Learning model

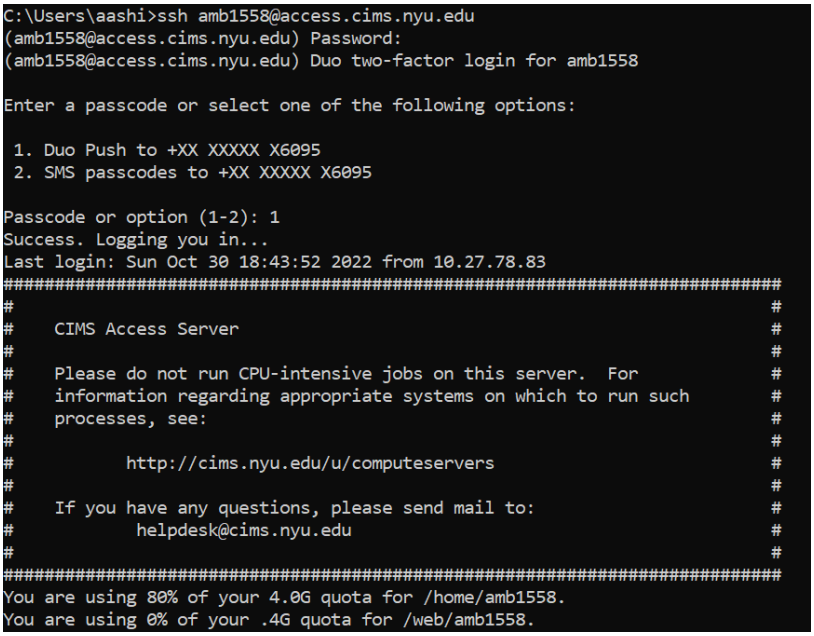
FLOPs Calculation

We will be running our code on NYU’s Greene cluster to calculate the number of floating-point operations for a single convolution layer. We will be using NVIDIA Nsight Compute to get the fadd, fmul, and ffma parameters which can be used for FLOPs calculation.

```
flop_count_sp:
smsp__sass_thread_inst_executed_op_fadd_pred_on.sum +
smsp__sass_thread_inst_executed_op_fmul_pred_on.sum +
smsp__sass_thread_inst_executed_op_ffma_pred_on.sum * 2
```

Phase 1: Setting up a container with latest dependencies

Step 1: SSH into access on the CIMS server



Step 2: SSH into the Greene cluster. This is where we will run our scripts

Step 3: Do a module load to use only use Python 3.8. Following that do an ssh burst and start a slurm interactive job on the cluster.

Step 4: Create a python virtual environment. Following that, create a Singularity container and activate the environment inside the container.

Step 5: I have created a folder called mnist with all my required files. Enter the file and install the packages inside requirements.txt.

```
(ash) Singularity> cd mnist/
(ash) Singularity> ls
log  main.py  main_ncu.py  memory_mnist.py  ncu.out  requirements.txt  test-no-ncu.sh  test-w-ncu.sh
(ash) Singularity>
```

```
(ash) Singularity> pip3.8 install -r requirements.txt
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: torch in /home/amb1558/.local/lib/python3.8/site-packages (from -r requirements.txt (line 1)) (1.13.0)
Requirement already satisfied: torchvision in /home/amb1558/.local/lib/python3.8/site-packages (from -r requirements.txt (line 2)) (0.14.0)
Requirement already satisfied: torch_tb_profiler in /home/amb1558/.local/lib/python3.8/site-packages (from -r requirements.txt (line 3)) (0.4.0)
Requirement already satisfied: torchsummary in /home/amb1558/.local/lib/python3.8/site-packages (from -r requirements.txt (line 4)) (1.5.1)
Requirement already satisfied: tensorboard-plugin-profile in /home/amb1558/.local/lib/python3.8/site-packages (from -r requirements.txt (line 5)) (2.8.0)
Requirement already satisfied: fvcore in /home/amb1558/.local/lib/python3.8/site-packages (from -r requirements.txt (line 6)) (0.1.5.post20220512)
Requirement already satisfied: nvidia-cudnn-cu11==8.5.0.96 in /home/amb1558/.local/lib/python3.8/site-packages (from torch->-r requirements.txt (line 1)) (8.5.0.96)
Requirement already satisfied: nvidia-cuda-runtime-cu11==11.7.99 in /home/amb1558/.local/lib/python3.8/site-packages (from torch->-r requirements.txt (line 1)) (11.7.99)
Requirement already satisfied: nvidia-cuda-nvrtc-cu11==11.7.99 in /home/amb1558/.local/lib/python3.8/site-packages (from torch->-r requirements.txt (line 1)) (11.7.99)
Requirement already satisfied: nvidia-cublas-cu11==11.10.3.66 in /home/amb1558/.local/lib/python3.8/site-packages (from torch->-r requirements.txt (line 1)) (11.10.3.66)
Requirement already satisfied: typing-extensions in /home/amb1558/.local/lib/python3.8/site-packages (from torch->-r requirements.txt (line 1)) (4.4.0)
Requirement already satisfied: setuptools in /usr/lib/python3/dist-packages (from nvidia-cublas-cu11==11.10.3.66->torch->-r requirements.txt (line 1)) (45.2.0)
Requirement already satisfied: wheel in /usr/lib/python3/dist-packages (from nvidia-cublas-cu11==11.10.3.66->torch->-r requirements.txt (line 1)) (0.34.2)
Requirement already satisfied: requests in /usr/lib/python3/dist-packages (from torchvision->-r requirements.txt (line 2)) (2.22.0)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.8/dist-packages (from torchvision->-r requirements.txt (line 2)) (8.4.0)
Requirement already satisfied: numpy in /home/amb1558/.local/lib/python3.8/site-packages (from torchvision->-r requirements.txt (line 2)) (1.23.4)
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.8/dist-packages (from torch_tb_profiler->-r requirements.txt (line 3)) (1.3.4)
Requirement already satisfied: tensorboard!=2.1.0,>=1.15 in /home/amb1558/.local/lib/python3.8/site-packages (from torch_tb_profiler->-r requirements.txt (line 3)) (2.10.1)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.8/dist-packages (from tensorboard-plugin-profile->-r requirements.txt (line 5)) (1.16.0)
Requirement already satisfied: protobuf>=3.12.0 in /home/amb1558/.local/lib/python3.8/site-packages (from tensorboard-plugin-profile->-r requirements.txt (line 5)) (3.19.6)
Requirement already satisfied: gviz-api>=1.9.0 in /home/amb1558/.local/lib/python3.8/site-packages (from tensorboard-plugin-profile->-r requirements.txt (line 5)) (1.10.0)
Requirement already satisfied: werkzeug>=0.11.15 in /home/amb1558/.local/lib/python3.8/site-packages (from tensorboard-plugin-profile->-r requirements.txt (line 5)) (2.2.2)
Requirement already satisfied: termcolor>=1.1 in /home/amb1558/.local/lib/python3.8/site-packages (from fvcore->-r requirements.txt (line 6)) (2.0.1)
Requirement already satisfied: tabulate in /home/amb1558/.local/lib/python3.8/site-packages (from fvcore->-r requirements.txt (line 6)) (0.9.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/lib/python3/dist-packages (from fvcore->-r requirements.txt (line 6)) (5.3.1)
Requirement already satisfied: iopath>=0.1.7 in /home/amb1558/.local/lib/python3.8/site-packages (from fvcore->-r requirements.txt (line 6)) (0.1.10)
Requirement already satisfied: tqdm in /home/amb1558/.local/lib/python3.8/site-packages (from fvcore->-r requirements.txt (line 6)) (4.64.1)
Requirement already satisfied: yacs>=0.1.6 in /home/amb1558/.local/lib/python3.8/site-packages (from fvcore->-r requirements.txt (line 6)) (0.1.8)
Requirement already satisfied: portalocker in /home/amb1558/.local/lib/python3.8/site-packages (from iopath>=0.1.7->fvcore->-r requirements.txt (line 6)) (2.6.0)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from pandas>=1.0.0->torch_tb_profiler->-r requirements.txt (line 3)) (2021.3)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.8/dist-packages (from pandas>=1.0.0->torch_tb_profiler->-r requirements.txt (line 3)) (2.8.2)
Requirement already satisfied: google-auth<3,>=1.6.3 in /home/amb1558/.local/lib/python3.8/site-packages (from tensorboard!=2.1.0,>=1.15->torch_tb_profiler->-r requirements.
txt (line 3)) (2.13.0)
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /home/amb1558/.local/lib/python3.8/site-packages (from tensorboard!=2.1.0,>=1.15->torch_tb_profiler->
-r requirements.txt (line 3)) (0.6.1)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /home/amb1558/.local/lib/python3.8/site-packages (from tensorboard!=2.1.0,>=1.15->torch_tb_profiler->-r requi
rements.txt (line 3)) (1.8.1)
```

Step 6: Verify that torch can be used inside a python script and check the version

```
>>> import torch
>>> torch.__version__
'1.13.0+cu117'
>>> torch.cuda.is_available()
True
```

Phase 2: Analyzing the FLOP count with and without NCU

Step 1: First we will run a simple MNIST script (Vanilla) which will do a simple dry run for 2 epochs. The code is not unlike to the model we have seen above. It uses the file main.py which does not have any NCU code. Using this, we can verify if our script can run on the Greene cluster.

```
#!/bin/bash

python3.8 main.py --dry-run --epochs 2
```

```
Net(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (dropout1): Dropout(p=0.25, inplace=False)
  (dropout2): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=9216, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=10, bias=True)
)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 26, 26]             320
            Conv2d-2          [-1, 64, 24, 24]          18,496
           Dropout-3          [-1, 64, 12, 12]               0
            Linear-4                 [-1, 128]       1,179,776
           Dropout-5                 [-1, 128]               0
            Linear-6                  [-1, 10]           1,290
================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.52
Params size (MB): 4.58
Estimated Total Size (MB): 5.10
----------------------------------------------------------------
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.303741

Test set: Average loss: 2.4183, Accuracy: 2356/10000 (24%)

Train Epoch: 2 [0/60000 (0%)]   Loss: 2.346994

Test set: Average loss: 2.2437, Accuracy: 2600/10000 (26%)
```

Step 2: Now that we have verified that our vanilla script runs on the cluster, we will make changes to the code to get the flop parameters from NCU. We have passed the epoch number as a parameter and are only running the NCU computation for epoch 4 over the 2nd convolution layer.

```python
def forward(self, x, epoch):
    x = self.conv1(x)
    x = F.relu(x)
    if epoch==4:
        NCU.start()
    x = self.conv2(x)
    if epoch==4:
        NCU.stop()
    x = F.relu(x)
    x = F.max_pool2d(x, 2)
    x = self.dropout1(x)
    x = torch.flatten(x, 1)
    x = self.fc1(x)
    x = F.relu(x)
    x = self.dropout2(x)
    x = self.fc2(x)
    output = F.log_softmax(x, dim=1)
    return output
```

```
(ash) Singularity> ./test-w-ncu.sh
==PROF== Target process 338988 terminated before first instrumented API call.
==PROF== Connected to process 338879 (/usr/bin/python3.8)
Net(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (dropout1): Dropout(p=0.25, inplace=False)
  (dropout2): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=9216, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=10, bias=True)
)
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.311868
==PROF== Target process 340309 terminated before first instrumented API call.

Test set: Average loss: 2.3230, Accuracy: 1010/10000 (10%)

==PROF== Target process 344172 terminated before first instrumented API call.
Train Epoch: 2 [0/60000 (0%)]   Loss: 2.517499
==PROF== Target process 344664 terminated before first instrumented API call.

Test set: Average loss: 2.2570, Accuracy: 2735/10000 (27%)

==PROF== Target process 344724 terminated before first instrumented API call.
Train Epoch: 3 [0/60000 (0%)]   Loss: 2.243627
==PROF== Target process 345216 terminated before first instrumented API call.

Test set: Average loss: 2.1056, Accuracy: 4068/10000 (41%)
```

```
==PROF== Target process 345222 terminated before first instrumented API call.
==PROF== Profiling "nchwToNhwcKernel" - 1: 0%....50%....100% - 1 pass
==PROF== Profiling "nchwToNhwcKernel" - 2: 0%....50%....100% - 1 pass
==PROF== Profiling "kernel" - 3: 0%....50%....100% - 1 pass
==PROF== Profiling "elementwise_kernel" - 4: 0%....50%....100% - 1 pass
Train Epoch: 4 [0/60000 (0%)]   Loss: 2.158825
==PROF== Target process 345767 terminated before first instrumented API call.

Test set: Average loss: 2.0066, Accuracy: 4467/10000 (45%)
```

```
==PROF== Target process 346206 terminated before first instrumented API call.
==PROF== Disconnected from process 338879
[338879] python3.8@127.0.0.1
  void cudnn::ops::nchwToNhwcKernel<float, float, float, (bool)0, (bool)1, (cudnnKernelDataType_t)2>(cudnn::ops::nchw2nhwc_params_t<T3>, const T1 *, T2 *), 2022-Oct-31 15:11
:21, Context 1, Stream 7
    Section: Command line profiler metrics
    ---------------------------------------------------------------------- -------------- ------------------------------
    smsp__sass_thread_inst_executed_op_fadd_pred_on.avg                    inst                              0
    smsp__sass_thread_inst_executed_op_fadd_pred_on.max                    inst                              0
    smsp__sass_thread_inst_executed_op_fadd_pred_on.min                    inst                              0
    smsp__sass_thread_inst_executed_op_fadd_pred_on.sum                    inst                              0
    smsp__sass_thread_inst_executed_op_ffma_pred_on.avg                    inst                              0
    smsp__sass_thread_inst_executed_op_ffma_pred_on.max                    inst                              0
    smsp__sass_thread_inst_executed_op_ffma_pred_on.min                    inst                              0
    smsp__sass_thread_inst_executed_op_ffma_pred_on.sum                    inst                              0
    smsp__sass_thread_inst_executed_op_fmul_pred_on.avg                    inst                        4005.93
    smsp__sass_thread_inst_executed_op_fmul_pred_on.max                    inst                           5184
    smsp__sass_thread_inst_executed_op_fmul_pred_on.min                    inst                           2816
    smsp__sass_thread_inst_executed_op_fmul_pred_on.sum                    inst                        1730560
    ---------------------------------------------------------------------- -------------- ------------------------------

  void cudnn::ops::nchwToNhwcKernel<float, float, float, (bool)0, (bool)1, (cudnnKernelDataType_t)2>(cudnn::ops::nchw2nhwc_params_t<T3>, const T1 *, T2 *), 2022-Oct-31 15:11
:22, Context 1, Stream 7
    Section: Command line profiler metrics
    ---------------------------------------------------------------------- -------------- ------------------------------
    smsp__sass_thread_inst_executed_op_fadd_pred_on.avg                    inst                              0
    smsp__sass_thread_inst_executed_op_fadd_pred_on.max                    inst                              0
    smsp__sass_thread_inst_executed_op_fadd_pred_on.min                    inst                              0
    smsp__sass_thread_inst_executed_op_fadd_pred_on.sum                    inst                              0
    smsp__sass_thread_inst_executed_op_ffma_pred_on.avg                    inst                              0
    smsp__sass_thread_inst_executed_op_ffma_pred_on.max                    inst                              0
    smsp__sass_thread_inst_executed_op_ffma_pred_on.min                    inst                              0
    smsp__sass_thread_inst_executed_op_ffma_pred_on.sum                    inst                              0
    smsp__sass_thread_inst_executed_op_fmul_pred_on.avg                    inst                          42.67
    smsp__sass_thread_inst_executed_op_fmul_pred_on.max                    inst                             96
    smsp__sass_thread_inst_executed_op_fmul_pred_on.min                    inst                              0
    smsp__sass_thread_inst_executed_op_fmul_pred_on.sum                    inst                          18432
    ---------------------------------------------------------------------- -------------- ------------------------------
```

```
  void xmma_cudnn::gemm::kernel<xmma_cudnn::implicit_gemm::fprop_indexed::Kernel_traits<xmma_cudnn::Ampere_hmma_tf32_traits<unsigned int, float>, xmma_cudnn::Cta_tile<xmma_c
udnn::Ampere, (int)256, (int)64, (int)32, (int)4, (int)2, (int)1, (int)1>, xmma_cudnn::implicit_gemm::fprop_indexed::Gmem_tile_a_t<xmma_cudnn::Ampere_hmma_tf32_traits<unsign
ed int, float>, xmma_cudnn::Cta_tile<xmma_cudnn::Ampere, (int)256, (int)64, (int)32, (int)4, (int)2, (int)1, (int)1>, xmma_cudnn::implicit_gemm::Input_related<(int)0, (int)0
, (int)0, (bool)0>, (int)16, (bool)0, xmma_cudnn::implicit_gemm::fprop_indexed::Gmem_tile_base_a<xmma_cudnn::Ampere_hmma_tf32_traits<unsigned int, float>, xmma_cudnn::Cta_ti
le<xmma_cudnn::Ampere, (int)256, (int)64, (int)32, (int)4, (int)2, (int)1, (int)1>, xmma_cudnn::implicit_gemm::Input_related<(int)0, (int)0, (int)0, (bool)0>, (int)16, xmma_
cudnn::Row, (int)32, (int)256>>, xmma_cudnn::implicit_gemm::fprop_indexed::Gmem_tile_c_n<xmma_cudnn::Ampere_hmma_tf32_traits<unsigned int, float>, xmma_cudnn::Cta_tile<xmma_
cudnn::Ampere, (int)256, (int)64, (int)32, (int)4, (int)2, (int)1, (int)1>, (int)4, xmma_cudnn::Fragment_c<xmma_cudnn::Ampere_hmma_tf32_traits<unsigned int, float>, xmma_cud
nn::Cta_tile<xmma_cudnn::Ampere, (int)256, (int)64, (int)32, (int)4, (int)2, (int)1, (int)1>, (bool)0>>, xmma_cudnn::implicit_gemm::Input_related<(int)0, (int)0, (int)0, (bo
ol)0>, (int)3>>(T1::Params), 2022-Oct-31 15:11:22, Context 1, Stream 7
    Section: Command line profiler metrics
    ---------------------------------------------------------------------- -------------- ------------------------------
    smsp__sass_thread_inst_executed_op_fadd_pred_on.avg                    inst                        6826.67
    smsp__sass_thread_inst_executed_op_fadd_pred_on.max                    inst                           8192
    smsp__sass_thread_inst_executed_op_fadd_pred_on.min                    inst                           4096
    smsp__sass_thread_inst_executed_op_fadd_pred_on.sum                    inst                        2949120
    smsp__sass_thread_inst_executed_op_ffma_pred_on.avg                    inst                              0
    smsp__sass_thread_inst_executed_op_ffma_pred_on.max                    inst                              0
    smsp__sass_thread_inst_executed_op_ffma_pred_on.min                    inst                              0
    smsp__sass_thread_inst_executed_op_ffma_pred_on.sum                    inst                              0
    smsp__sass_thread_inst_executed_op_fmul_pred_on.avg                    inst                        6826.67
    smsp__sass_thread_inst_executed_op_fmul_pred_on.max                    inst                           8192
    smsp__sass_thread_inst_executed_op_fmul_pred_on.min                    inst                           4096
    smsp__sass_thread_inst_executed_op_fmul_pred_on.sum                    inst                        2949120
    ---------------------------------------------------------------------- -------------- ------------------------------

  void at::native::elementwise_kernel<(int)128, (int)2, void at::native::gpu_kernel_impl<at::native::CUDAFunctor_add<float>>(at::TensorIteratorBase &, const T1 &)::[lambda(i
nt) (instance 1)]>(int, T3), 2022-Oct-31 15:11:22, Context 1, Stream 7
    Section: Command line profiler metrics
    ---------------------------------------------------------------------- -------------- ------------------------------
    smsp__sass_thread_inst_executed_op_fadd_pred_on.avg                    inst                              0
    smsp__sass_thread_inst_executed_op_fadd_pred_on.max                    inst                              0
    smsp__sass_thread_inst_executed_op_fadd_pred_on.min                    inst                              0
    smsp__sass_thread_inst_executed_op_fadd_pred_on.sum                    inst                              0
    smsp__sass_thread_inst_executed_op_ffma_pred_on.avg                    inst                        6826.67
    smsp__sass_thread_inst_executed_op_ffma_pred_on.max                    inst                          10560
    smsp__sass_thread_inst_executed_op_ffma_pred_on.min                    inst                           4160
    smsp__sass_thread_inst_executed_op_ffma_pred_on.sum                    inst                        2949120
    smsp__sass_thread_inst_executed_op_fmul_pred_on.avg                    inst                              0
    smsp__sass_thread_inst_executed_op_fmul_pred_on.max                    inst                              0
    smsp__sass_thread_inst_executed_op_fmul_pred_on.min                    inst                              0
    smsp__sass_thread_inst_executed_op_fmul_pred_on.sum                    inst                              0
    ---------------------------------------------------------------------- -------------- ------------------------------
```

The above images show the outputs from the NCU for the 2nd comvolution for epoch number 4. It is necessary to make the calculations for a single epoch.

Flop Calculation from NCU:

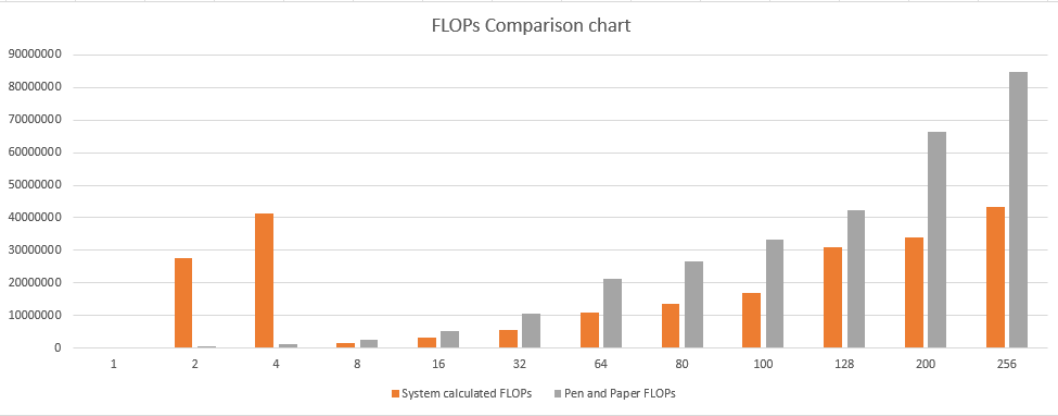
smsp__sass_thread_inst_executed_op_fadd_pred_on.sum +
smsp__sass_thread_inst_executed_op_fmul_pred_on.sum +
smsp__sass_thread_inst_executed_op_ffma_pred_on.sum * 2

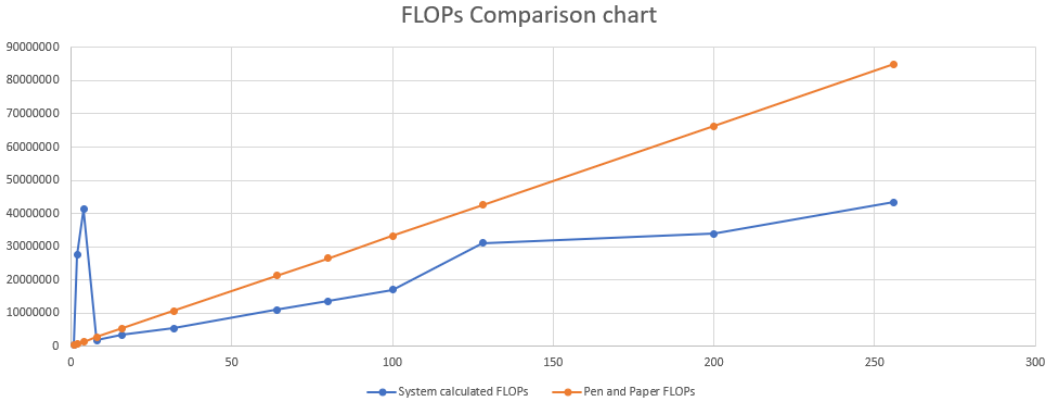
= 1384448 + 18432 + 2359296 + 2359296 + 2*2359296
= 10,840,064

Manual Flop Calculation:

$$2 * k * k * Cin * Wout * Hout * batch\_size$$
$$= 2 * 3 * 3 * 32 * 24 * 24 * 64$$
$$= 21{,}233{,}664$$

Here is a comparison graph for how the FLOPs calculation changes with different batch sizes.





In most cases, the pen and paper method for calculating the FLOPs is greater. This is because the system does some caching or optimization to reduce the number of actual operations done. There is also a huge increase in the number of computations for small sizes since GPUs are not designed to handles smaller batch sizes. We can see more standard behavior for higher batch sizes. System calculated FLOPs is measured and Pen and Paper FLOPs is estimated.

Memory Calculation

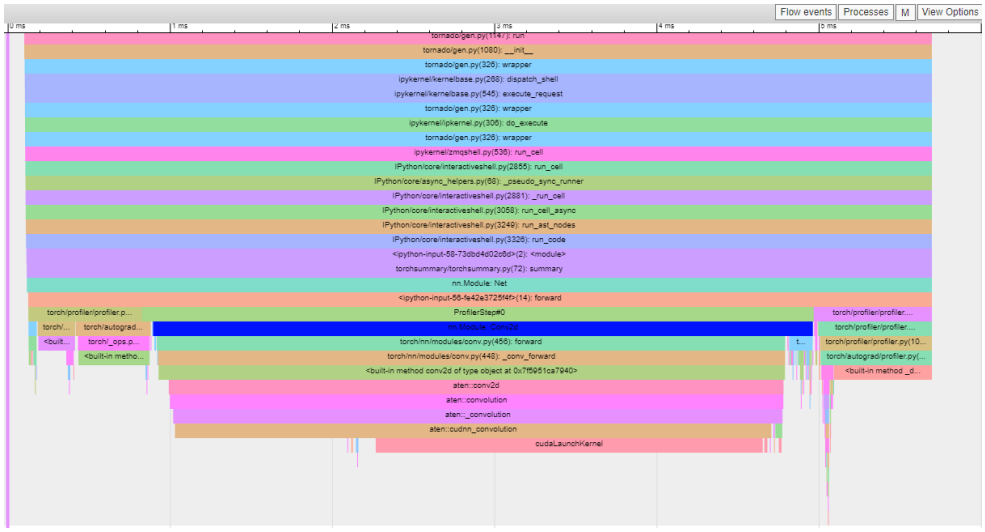
We will be using the PyTorch profiler to compute the amount of memory required for 1 convolutional layer. I have already described the method to estimate the amount of memory required in the pen and paper method for space complexity. Here, we will modify the vanilla code to insert a profiler step before and after the convolution layer. The code will generate a call stack log which can be used to analyze the code steps including the forward and backward propagation. There are many more blocks to analyze.

```
profiler = torch.profiler.profile(
      schedule=torch.profiler.schedule(wait=0, warmup=0, active=1, repeat=1),
      on_trace_ready=torch.profiler.tensorboard_trace_handler('./log/mnist'),
      record_shapes=True,
      profile_memory=True,
      with_stack=True,
      activities=[ProfilerActivity.CPU, ProfilerActivity.CUDA]
)
```

```
def forward(self, x):
    x = self.conv1(x)
    x = F.relu(x)
    profiler.start()
    x = self.conv2(x)
    profiler.step()
    profiler.stop()
    x = F.relu(x)
    x = F.max_pool2d(x, 2)
    x = self.dropout1(x)
    x = torch.flatten(x, 1)
    x = self.fc1(x)
    x = F.relu(x)
    x = self.dropout2(x)
    x = self.fc2(x)
    output = F.log_softmax(x, dim=1)
    return output
```

Snippet of the generated log:

Generated Log Trace:
We use the chrome tracer to upload the log in chrome://tracing.

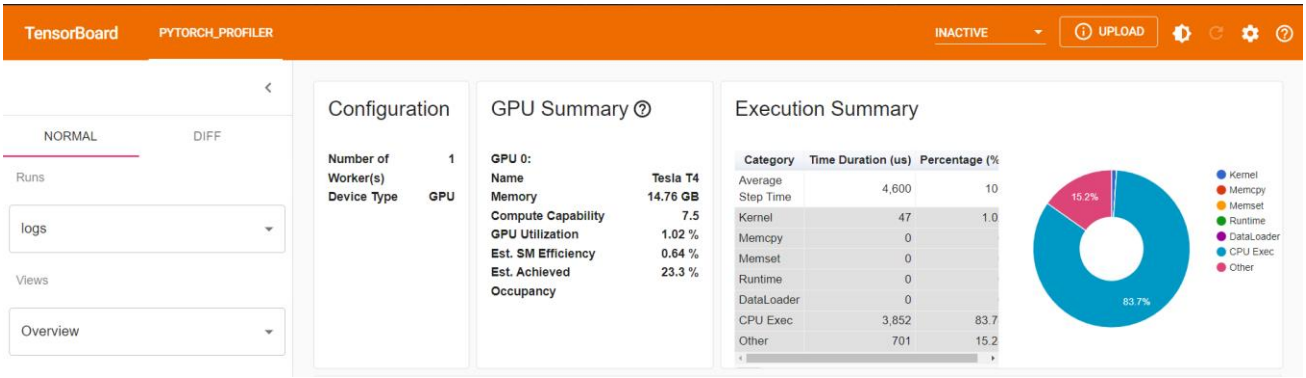


In order to better analyze the logs, we will use tensor board.

Step 1: Copy the logs to your PC and open in Jupyter Notebook

Step 2: Run the following snippet and then open it up in tensor board; localhost://6006

```
!python -m tensorboard.main --logdir=./logs
```



Step 3: Navigate to the Memory View and access the table at the bottom. This is a snippet of the memory utilization of batch size 1.

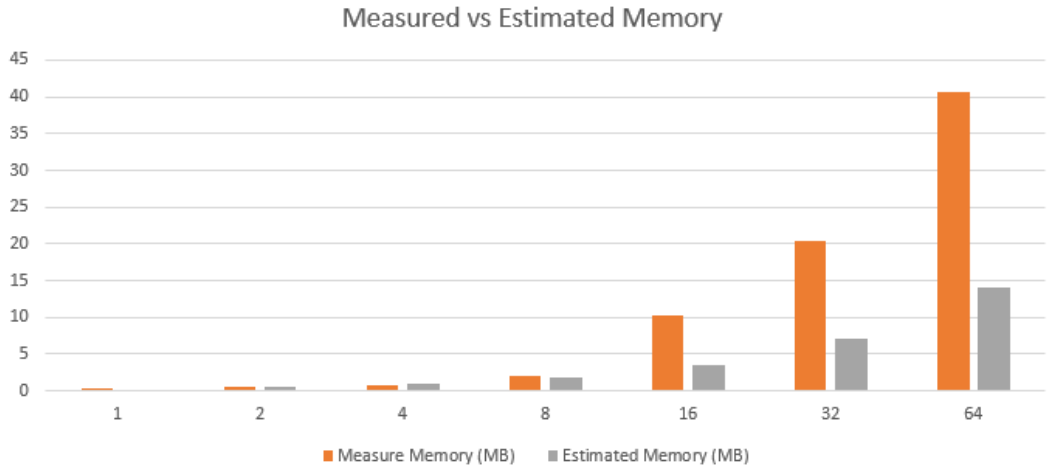
Operator Name	Calls	Size Increase (KB)	Self Size Increase (KB)	Allocation Count	Self Allocation Count	Allocation Size (KB)	Self Allocation Size (KB)
aten::empty	3	31392	31392	1	1	31392	31392
aten::_convolution	1	9216	0	2	0	40608	0
aten::conv2d	1	9216	0	2	0	40608	0
aten::convolution	1	9216	0	2	0	40608	0
aten::cudnn_convolution	1	9216	-22176	2	1	40608	9216

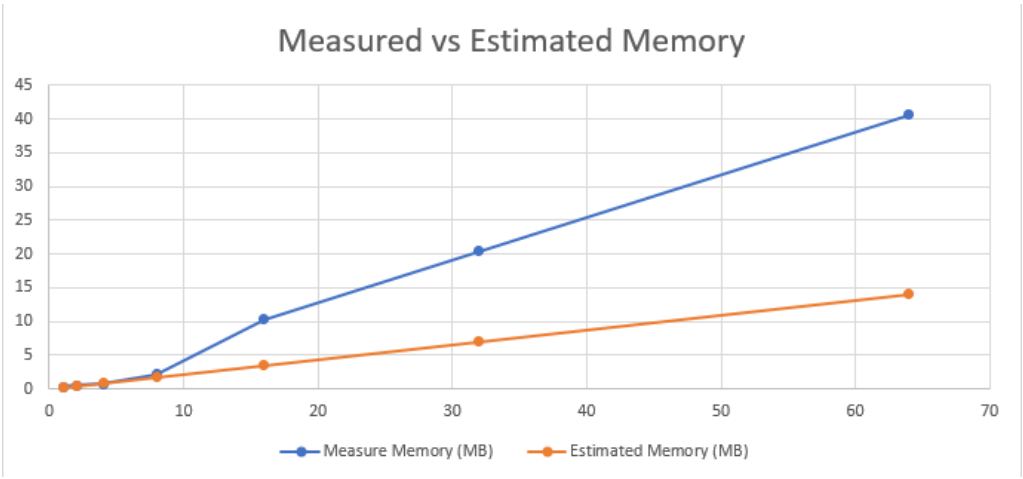
Using out pen and paper method, we estimated the required memory would be 0.22 MB. This is the calculation for a batch-size of 1. If we multiply it by the batch size, we have the memory requirement for a specific batch size.

$$
\begin{aligned}
&(Hout * Wout * Cout + ((k * k * Cin) + 1) * Cout) * 4 \\
&= (Hout * Wout * Cout + Params) * 4 \\
&= (24 * 24 * 64 + 18{,}496) * 4 \\
&= 221{,}440
\end{aligned}
$$

From the tensor board image attached above, we can see that it uses 40.6 MB for a batch size of 64. By multiplying 64 with 0.22 MB we get 14.08 MB which is the estimate for the memory computation. There is a marked difference between the measured memory and the estimated memory because there are many kinds of overheads that are actually involved when doing convolutions. The compiler uses many different temporary variables for value storage and often times, extra memory is used in PyTorch to optimize the run-time. To avoid errors, we may also need to save the values temporarily in order to recover them in case of node failure (due to multiple GPUs being used).

Here is a comparison graph for how the memory changes with different batch sizes for both measured and estimated.





Changing the Conv Layer dimensions

This is one example of changing the dimensions of the 2D convolution. Here, we are changing the output filters of the first and 2nd layer Additionally, we need to change the input parameters for the linear layer. Output of the first layer will be 26x26x20 and the output of the 2nd layer will be 24x24x40. After passing through a pooling layer, the output will become 12x12x40 which is 5760.

Below, we can also see a snippet of the training happening for this new dimensions of the model. If we do not change the linear layer dimensions it will give us a matrix multiplication error.

```
def __init__(self):
    super(Net, self).__init__()
    self.conv1 = nn.Conv2d(1, 20, 3, 1)
    self.conv2 = nn.Conv2d(20, 40, 3, 1)
    self.dropout1 = nn.Dropout(0.25)
    self.dropout2 = nn.Dropout(0.5)
    self.fc1 = nn.Linear(5760, 128)
    self.fc2 = nn.Linear(128, 10)
```

```
Train Epoch: 2 [57760/60000 (96%)]      Loss: 0.251239
Train Epoch: 2 [57920/60000 (97%)]      Loss: 0.003926
Train Epoch: 2 [58080/60000 (97%)]      Loss: 0.006728
Train Epoch: 2 [58240/60000 (97%)]      Loss: 0.007224
Train Epoch: 2 [58400/60000 (97%)]      Loss: 0.013919
Train Epoch: 2 [58560/60000 (98%)]      Loss: 0.013936
Train Epoch: 2 [58720/60000 (98%)]      Loss: 0.000084
Train Epoch: 2 [58880/60000 (98%)]      Loss: 0.000386
Train Epoch: 2 [59040/60000 (98%)]      Loss: 0.002476
Train Epoch: 2 [59200/60000 (99%)]      Loss: 0.062372
Train Epoch: 2 [59360/60000 (99%)]      Loss: 0.149094
Train Epoch: 2 [59520/60000 (99%)]      Loss: 0.000228
Train Epoch: 2 [59680/60000 (99%)]      Loss: 0.004760
Train Epoch: 2 [59840/60000 (100%)]     Loss: 0.001256
```

Conclusion

From this report, we have gone over how to PyTorch profiler to record the memory required for a convolution layer. We have also seen the usage of NVIDIA Nsight (NCU) to calculate the FLOPs for a convolution layer. We use different formulas to estimate the number of flops and the memory used for this calculation/ However, the estimated flops and memory is often different from the measured values, due to the architecture of the cloud which is not taken into consideration. There are overheads and other methods used for optimizations which can deviate from the estimated value. Overall, it is still a good method for estimation.

References

[1] https://www.thinkautonomous.ai/blog/deep-learning-optimization/#calculating-the-flops-in-a-model
[2] https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks
[3] https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/
[4] https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html
[5] https://computersciencewiki.org/index.php/Max-pooling_/_Pooling