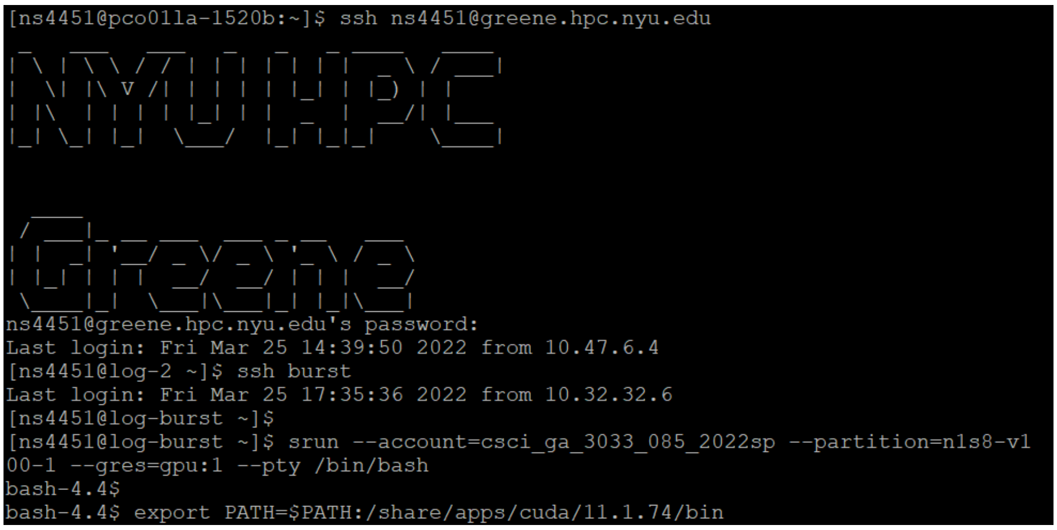
Niharika Sinha
ns4451

HW4 – Computation and Complexity Measurement for MNIST CNN

I logged onto the Greene cluster, and performed the following steps. The data and code was downloaded from git repo [1] and cloned in a directory here.

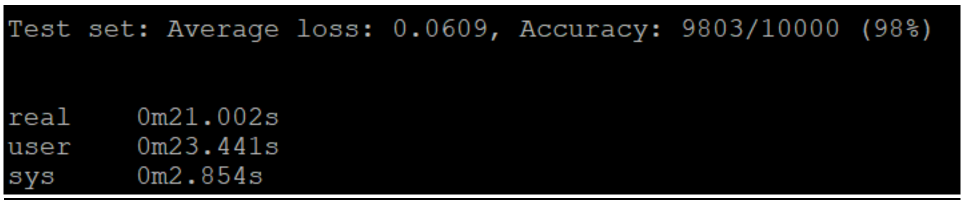


```
[ns4451@pco01la-1520b:~]$ ssh ns4451@greene.hpc.nyu.edu
 _   ___   ___   _   _   _ ____   ____
| \ | \ \ / / | | | | | | |  _ \ / ___|
|  \| |\ V /| | | | | |_| | |_) | |
| |\  | | | | |_| | |  _  |  __/| |___
|_| \_| |_|  \___/  |_| |_|_|    \____|


  ____
 / ___|_ __ ___  ___ _ __   ___
| |  _| '__/ _ \/ _ \ '_ \ / _ \
| |_| | | |  __/  __/ | | |  __/
 \____|_|  \___|\___|_| |_|\___|
ns4451@greene.hpc.nyu.edu's password:
Last login: Fri Mar 25 14:39:50 2022 from 10.47.6.4
[ns4451@log-2 ~]$ ssh burst
Last login: Fri Mar 25 17:35:36 2022 from 10.32.32.6
[ns4451@log-burst ~]$
[ns4451@log-burst ~]$ srun --account=csci_ga_3033_085_2022sp --partition=n1s8-v1
00-1 --gres=gpu:1 --pty /bin/bash
bash-4.4$
bash-4.4$ export PATH=$PATH:/share/apps/cuda/11.1.74/bin
```

Figure 1: Logging onto the Greene cluster and starting GPU

I had to install torchvision through pip (already present), ran the code and captured the elapsed time:

```
python3 main.py --dry-run
time python3 main.py –epoch 1
```



```
Test set: Average loss: 0.0609, Accuracy: 9803/10000 (98%)


real    0m21.002s
user    0m23.441s
sys     0m2.854s
```

Figure 2: The elapsed time for running MNIST CNN for 1 epoch and default batch size

FLOPS – Pen and Paper Estimation

FLOPs stands for Floating Point operations per second and is a measure of computer performance, especially those that involve floating point calculations[2]. In our CNN model, to calculate FLOPs of a particular layer we need to find the number of operations being executed in finding the CNN output, the number of multiplication operations and the number of addition operations. These are given by the formulae:

$$\#\text{operations}_{\text{CNN\_output}} = \text{channels}_{\text{output}} * \text{width}_{\text{output\_image}} * \text{height}_{\text{output\_image}}$$

$$= 64 * 24 * 24 = 36864$$

$$\#\text{multiplications} = \text{channels}_{\text{input}} * \text{kernel\_size} * \text{kernel\_size}$$

$$= 32 * 3 * 3 = 288$$

$$\#\text{additions} = \text{channels}_{\text{input}} * \text{kernel\_size} * \text{kernel\_size} - 1$$

$$= 32 * 3 * 3 - 1 = 288 - 1 = 287$$

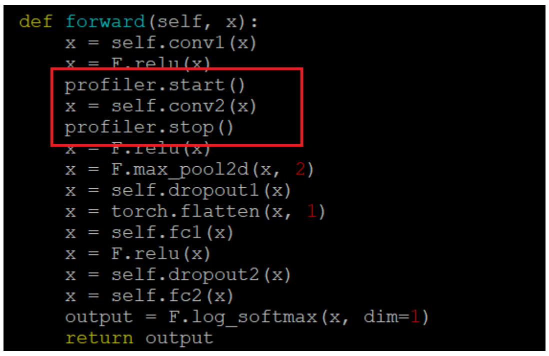
We take bias = 1

$$\text{FLOPS} = \#\text{operations}_{\text{CNN\_output}} * (\#\text{multiplications} + \#\text{additions} + \text{bias})$$

$$= 36864 * (288 + 277 + 1) = 2.1 * 10^7$$

FLOPS – Program Measurement

We profile the same code in order to measure the complexity in terms of FLOPS by making a few changes as shown in the following figure.



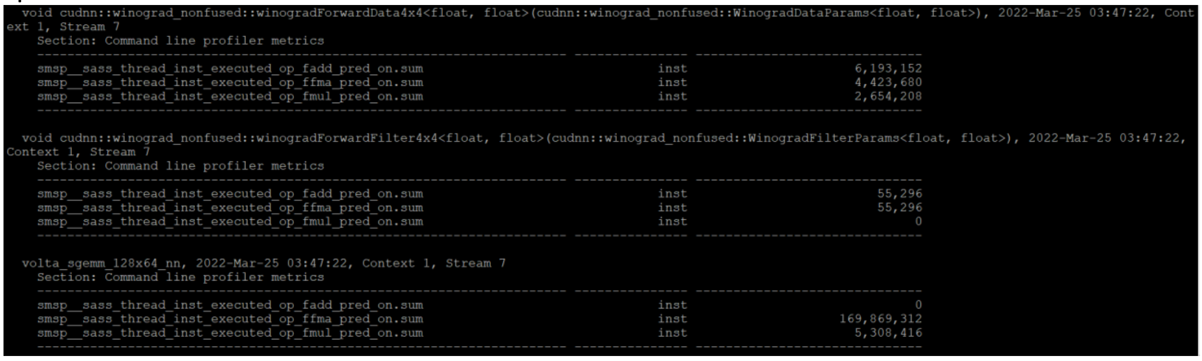
```
def forward(self, x):
    x = self.conv1(x)
    x = F.relu(x)
    profiler.start()
    x = self.conv2(x)
    profiler.stop()
    x = F.relu(x)
    x = F.max_pool2d(x, 2)
    x = self.dropout1(x)
    x = torch.flatten(x, 1)
    x = self.fc1(x)
    x = F.relu(x)
    x = self.dropout2(x)
    x = self.fc2(x)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

Figure 3: Changes in code to capture FLOPS

The command to run the profiler is:

```
ncu --log-file [LOG_FOLDER]/[LOG_FILE_NAME] --profile-from-start off --metrics 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 --target-processes all python3 [PROFILER_FILE].py --dry-run --batch-size [BATCH_SIZE] --epoch 1 --log-interval 10
```

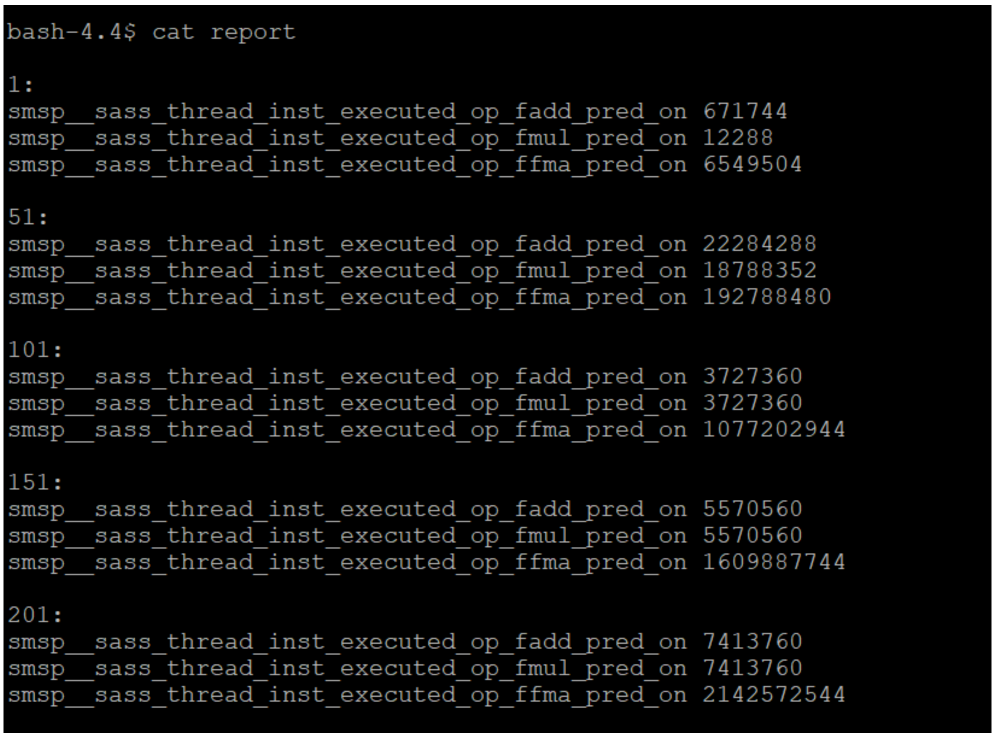
I ran this command for batch size starting from 1 till 501 at intervals of 50. The *fadd, fmul* and *ffma* information for each batch size gets captured in their corresponding log files as specified in the command.



```
  void cudnn::winograd_nonfused::winogradForwardData4x4<float, float>(cudnn::winograd_nonfused::WinogradDataParams<float, float>), 2022-Mar-25 03:47:22, Cont
ext 1, Stream 7
    Section: Command line profiler metrics
    ---------------------------------------------------------------- --------------- ------------------------------
    smsp__sass_thread_inst_executed_op_fadd_pred_on.sum                         inst                      6,193,152
    smsp__sass_thread_inst_executed_op_ffma_pred_on.sum                         inst                      4,423,680
    smsp__sass_thread_inst_executed_op_fmul_pred_on.sum                         inst                      2,654,208
    ---------------------------------------------------------------- --------------- ------------------------------

  void cudnn::winograd_nonfused::winogradForwardFilter4x4<float, float>(cudnn::winograd_nonfused::WinogradFilterParams<float, float>), 2022-Mar-25 03:47:22,
Context 1, Stream 7
    Section: Command line profiler metrics
    ---------------------------------------------------------------- --------------- ------------------------------
    smsp__sass_thread_inst_executed_op_fadd_pred_on.sum                         inst                         55,296
    smsp__sass_thread_inst_executed_op_ffma_pred_on.sum                         inst                         55,296
    smsp__sass_thread_inst_executed_op_fmul_pred_on.sum                         inst                              0
    ---------------------------------------------------------------- --------------- ------------------------------

  volta_sgemm_128x64_nn, 2022-Mar-25 03:47:22, Context 1, Stream 7
    Section: Command line profiler metrics
    ---------------------------------------------------------------- --------------- ------------------------------
    smsp__sass_thread_inst_executed_op_fadd_pred_on.sum                         inst                              0
    smsp__sass_thread_inst_executed_op_ffma_pred_on.sum                         inst                    169,869,312
    smsp__sass_thread_inst_executed_op_fmul_pred_on.sum                         inst                      5,308,416
    ---------------------------------------------------------------- --------------- ------------------------------
```

Figure 4: FLOP output for batch size = 51

I consolidated this information into a single report, referring to the code at [3].



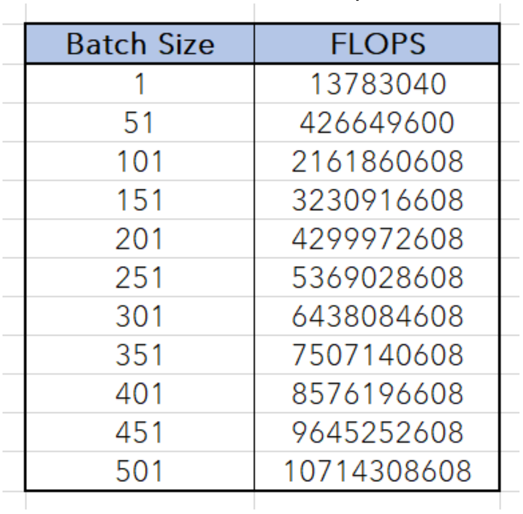
```
bash-4.4$ cat report

1:
smsp__sass_thread_inst_executed_op_fadd_pred_on 671744
smsp__sass_thread_inst_executed_op_fmul_pred_on 12288
smsp__sass_thread_inst_executed_op_ffma_pred_on 6549504

51:
smsp__sass_thread_inst_executed_op_fadd_pred_on 22284288
smsp__sass_thread_inst_executed_op_fmul_pred_on 18788352
smsp__sass_thread_inst_executed_op_ffma_pred_on 192788480

101:
smsp__sass_thread_inst_executed_op_fadd_pred_on 3727360
smsp__sass_thread_inst_executed_op_fmul_pred_on 3727360
smsp__sass_thread_inst_executed_op_ffma_pred_on 1077202944

151:
smsp__sass_thread_inst_executed_op_fadd_pred_on 5570560
smsp__sass_thread_inst_executed_op_fmul_pred_on 5570560
smsp__sass_thread_inst_executed_op_ffma_pred_on 1609887744

201:
smsp__sass_thread_inst_executed_op_fadd_pred_on 7413760
smsp__sass_thread_inst_executed_op_fmul_pred_on 7413760
smsp__sass_thread_inst_executed_op_ffma_pred_on 2142572544
```

Figure 5: Consolidated output for all batch sizes

Using the following formula,

```
(FMA*2) + FADD + FMUL
```

we can find the total FLOPS for a given batch size of the model. I have captured the effect of varying the batch size on the memory in terms of FLOPs in a tabular and graphical format. These results are calculated on one mini batch for 1 epoch.



Batch Size	FLOPS
1	13783040
51	426649600
101	2161860608
151	3230916608
201	4299972608
251	5369028608
301	6438084608
351	7507140608
401	8576196608
451	9645252608
501	10714308608

Table 1: Batch size vs FLOPS

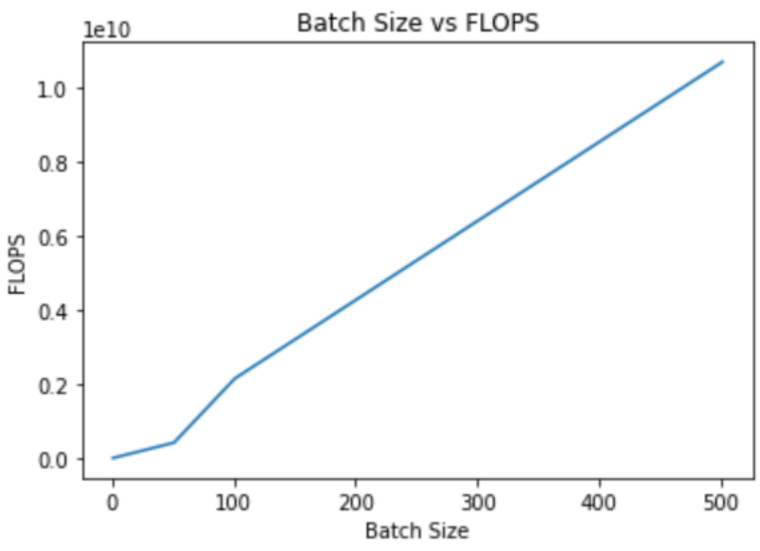


Figure 6: Batch size vs FLOPS

Memory – Pen and Paper Estimation

We use the following equation:

Convolutional layers = (Cin * Cout * K2)parameters + (Cout * out2dim)layer_outputs

Linear layers = (Cout * (Cin + 1))parameters and bias + Cout

Conv1 = $1 * 32 * 3^2 + 32 * 26^2$ = 21920 bytes = 0.16 MB

Conv2 = $32 * 64 * 3^2 + 64 * 24^2$ = 55296 bytes = 0.42 MB

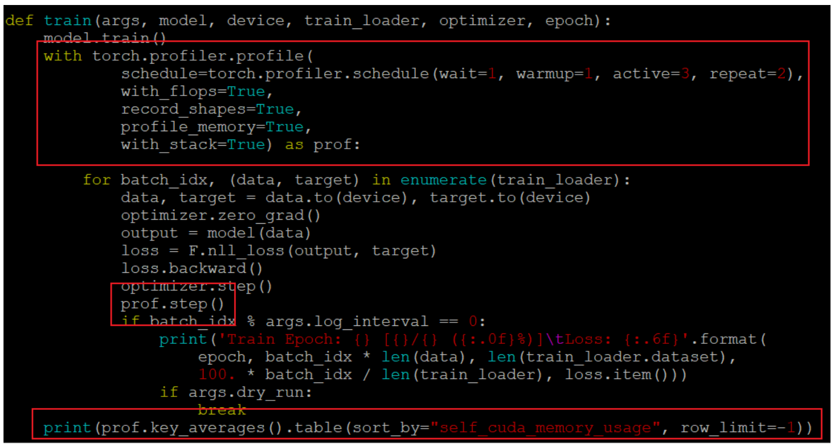
Linear1 = 9216 * (128 + 1) + 128 = 1179904 bytes = 9.0 MB

Linear2 = 10 * (128 + 1) + 10 = 1300 bytes = 0.01 MB

Total = 9.6 MB

Memory – Program Measurement

We profile the same code in order to measure the memory complexity by making a few changes shown in the following figure:



```
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    with torch.profiler.profile(
            schedule=torch.profiler.schedule(wait=1, warmup=1, active=3, repeat=2),
            with_flops=True,
            record_shapes=True,
            profile_memory=True,
            with_stack=True) as prof:

        for batch_idx, (data, target) in enumerate(train_loader):
            data, target = data.to(device), target.to(device)
            optimizer.zero_grad()
            output = model(data)
            loss = F.nll_loss(output, target)
            loss.backward()
            optimizer.step()
            prof.step()
            if batch_idx % args.log_interval == 0:
                print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                    epoch, batch_idx * len(data), len(train_loader.dataset),
                    100. * batch_idx / len(train_loader), loss.item()))
                if args.dry_run:
                    break
    print(prof.key_averages().table(sort_by="self_cuda_memory_usage", row_limit=-1))
```

Figure 7: Change in code to capture Memory

The command I used to run this memory profiler is:

```
python3 memory.py --epoch 1 --batch-size [BATCH_SIZE]
```

I ran this command for batch size starting from 1 till 201 at intervals of 50. The gives the memory utilized by all the layers of the CNN. I recorded the CUDA memory for the cudnn_convolution layer for different batch sizes and captured the same in a tabular format.

Batch Size	Memory
1	685.50 Kb
51	34.14 Mb
101	67.61 Mb
151	103.38 Mb
201	134.56 Mb

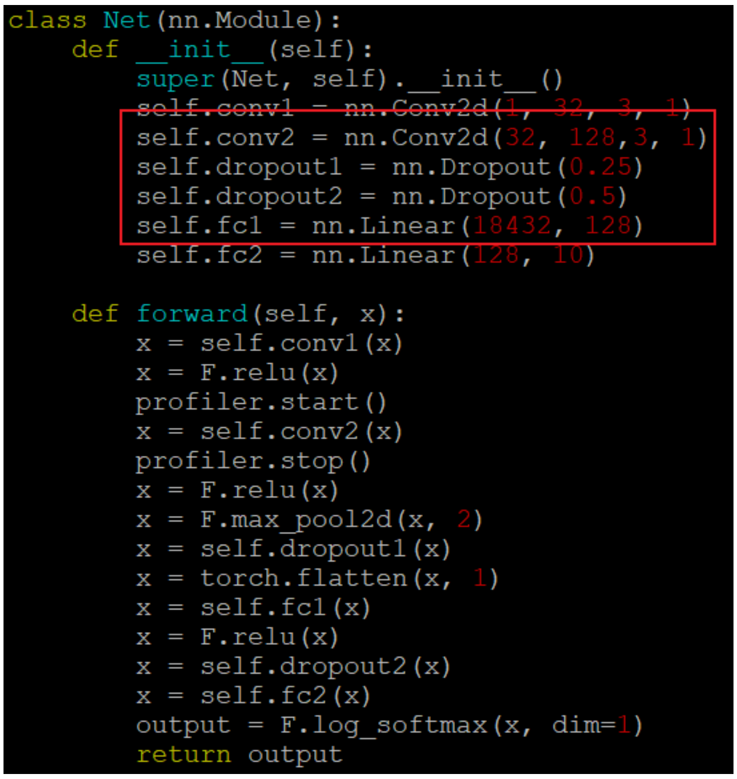
Table 2: Batch size vs Memory

<u>Difference between estimated and measured results</u>

We see a difference in the estimated and the actual performance metrics. This is because although there is an overhead involved in calculating the flops and updating the performance metrics through the program, GPU optimizations bring the time below the theoretical value.

<u>Modifying shape and dimension of Conv2d-2 layer</u>

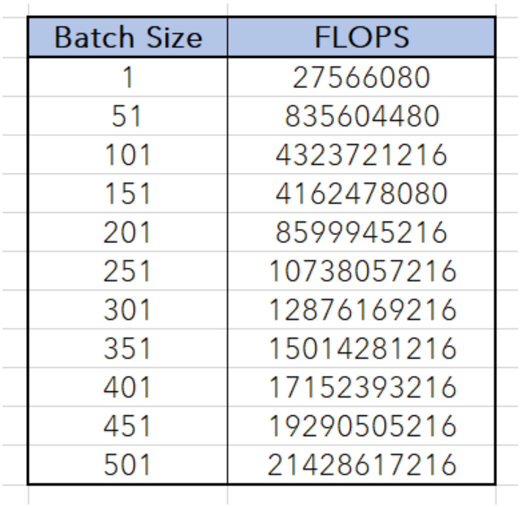
I made the following change in the CNN layers. I modified the input and output channels of the intermediate layers.



```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 128,3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(18432, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        profiler.start()
        x = self.conv2(x)
        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
```

Figure 8: Mofied CNN layers

I repeated the process to calculate and plot the FLOPS against batch size, and obtained the following results:



Batch Size	FLOPS
1	27566080
51	835604480
101	4323721216
151	4162478080
201	8599945216
251	10738057216
301	12876169216
351	15014281216
401	17152393216
451	19290505216
501	21428617216

Table 3: Batch size vs FLOPS for modified CNN

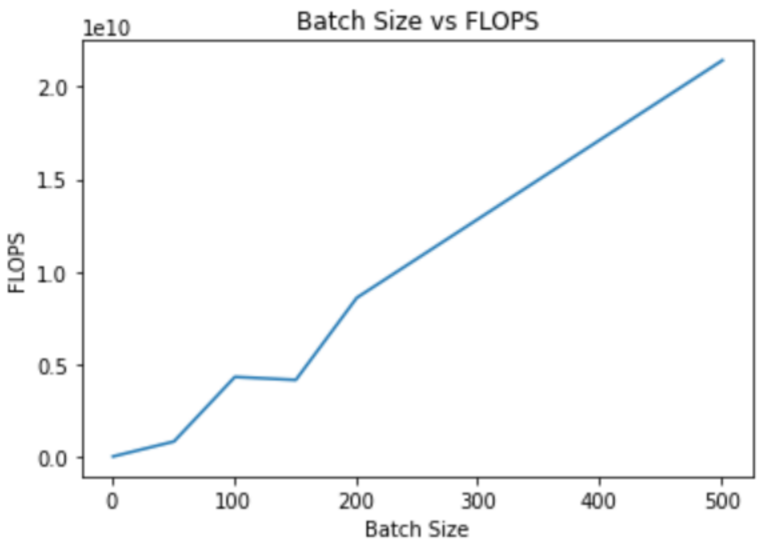


Figure 9: Batch Size vs FLOPS graph for modified CNN

References

[1] *https://github.com/pytorch/examples/tree/master/mnist*
[2] *https://en.wikipedia.org/wiki/FLOPS*
[3] *https://github.com/CharlleChen/NYU-Cloud-and-Machine-Learning/blob/main/hw3/generate_report.sh*