

# Analysis of Neural Network Models on ImageNet Dataset and Kernel Performance Comparison through Roofline Modelling

Niharika Sinha  
ns4451@nyu.edu

**Abstract:** The goal of this project is to understand the difference in performance on 2 different bare metal environments, when we train 3 different Neural network models on the ImageNet dataset. We run and analyze the following Neural network architectures - Resnet18, Resnet34 and Resnet50. We execute these models on V100 and A100 GPU. The aim is to understand profiling of the 3 models on a subset of training data. I also present a roofline analysis for the 2 GPUs, followed by a discussion on the differences in performances and the factors that affect this.

## 1. Roofline Modelling

Roof lining helps in profiling and optimizing the kernels we are working on. It is a visual performance model, that gives performance estimates of a given compute kernel. It combines locality, bandwidth and various parallelisation techniques into a single performance figure, and due to this it is emerging as a strong alternative to percent of peak performance estimate. [1]

Roofline analysis collects metrics to measure the kernel's compute and memory access activities. The ratio of compute to memory access for a kernel is called the arithmetic or operational intensity. Apart from this, we calculate an estimated FLOP's performance value for the kernel. Keeping the arithmetic intensity on the x-axis, and FLOPs on the y-axis, we plot a point for the kernel. The straight lines in the graph are called 'roofs', from where this methodology takes its name.

### a. NAÏVE ROOFLINE

Naïve roofline has 2 parameters - the peak performance and the peak bandwidth of the specific kernel, and 1 variable, the arithmetic intensity. Let us look at the 2 parameters:

- Peak Performance (GFLOPS) – derived from architectural manuals
- Peak Bandwidth – refers to the peak DRAM bandwidth that can be obtained by benchmarking

$$P = \min(\pi, \beta \times I), \text{ where}$$

P – attainable performance,  $\pi$  – peak performance,  $\beta$  – peak bandwidth, I – arithmetic intensity

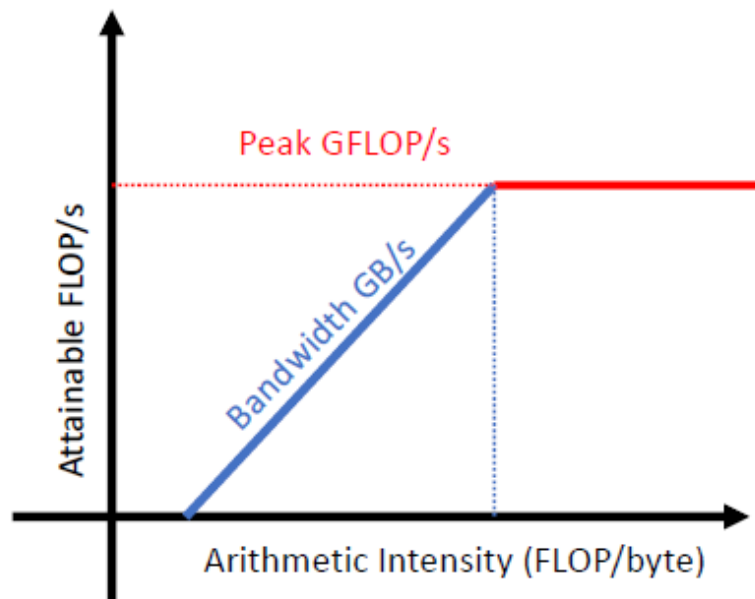


Figure 1 Roofline modelling for a system (Snapshot take from Lecture 7 slides)

The diagonal roof indicates a limit on a kernel's peak flops imposed by memory bandwidth for a given operational intensity. The horizontal roof indicates hardware compute limitations for the device. The point where the diagonal roof and horizontal roof meet is called the transition or ridge point. This is the point that indicates performance saturation (with peak performance  $\pi$ ), that is, the minimum arithmetic intensity required to achieve peak performance. Theoretically, we can give an estimate of the effort that will go into achieving the peak performance.

The entire visual representation of the kernel shows any hardware limitations we may be bumping up against. For example, increasing the operational intensity enables us to reach higher FLOPs before hitting memory bandwidth limits. Once we hit the compute roof, we need to think of other ways to boost the performance even further. (Double precision arithmetic has lower compute performance peak than single precision).

Roofline modelling is an emergent methodology that is becoming an important piece of the performance optimization puzzle.

## 2. Experiment Design

Having understood the roofline modelling, I wanted to design an experiment wherein I run the following three Neural network models on the A100 and V100 GPUs.

- Resnet18
- Resnet34
- Resnet50

I chose these 3 models belonging to the same family of neural networks because I specifically wanted to check how running the same model with different layers are plotted in the roofline graphs of NVIDIA A100 and NVIDIA V100.

### a. GPU SPECIFICATIONS

	NVIDIA V100 [2]	NVIDIA A100 [3]
Peak computation speed (FP32)	15.7 TFLOPS	19.5 TFLOPS
Peak Memory Bandwidth	0.9 TB/sec	1.555 GB/sec
Minimum Operational Intensity (As discussed, this is the ratio of the peak computation speed to the peak memory bandwidth)	<b>17.4 FLOP /Byte</b>	<b>12.5 FLOP /Byte</b>

### b. COMPLEXITY ESTIMATION

I referred to this git repo [4] to get the pre-computed theoretical values for the ResNet architecture. It provides very good information about estimates of memory consumption and FLOP counts for several convolutional neural networks.

We could also calculate the computations in each layer manually for the three models, following the formulae we used in HW4 [5]

$$\#operations_{SCNN\_output} = channels_{output} * width_{output\_image} * height_{output\_image}$$

$$\#multiplications = channels_{input} * kernel\_size * kernel\_size$$

$$\#additions = channels_{input} * kernel\_size * kernel\_size - 1$$

$$FLOPS = \#operations_{SCNN\_output} * (\#multiplications + \#additions + bias)$$

However as the layers increase, the computations get bulky and it gets tricky to manually compute the numbers by hand. Hence I referred to a site that already had this information, and I could get more time to experiment with profiling of the models.

The images in the ImageNet dataset contain images of dimension =

$$224 \times 224 \times 3 \text{ (3 corresponding to the RGB channels)}$$

One element takes 8 bits of memory, so memory taken by one image =

$$224 \times 224 \times 3 \times 1 \text{ byte} \sim 150 \text{ MB}$$

The input image size considered for the three models mentioned on [4] match our requirements here. The GFLOPS mentioned for the three models are as below:

	GFLOPS
Resnet18	2
Resnet34	4
Resnet50	4

Using this information and the formula for calculating the operational intensity:

$$\text{Operational Intensity} = \text{Compute} / \text{Memory}$$

We get the following operational intensity for the three models:

	Compute	Memory (Image size)	Operational Intensity
Resnet18	2 GFLOPS	150 MB	<b>13.3 FLOPS/Byte</b>
Resnet34	4 GFLOPS	150 MB	<b>26.6 FLOPS/Byte</b>
Resnet50	4 GFLOPS	150 MB	<b>25.3 FLOPS/Byte</b>

Keeping the expected operational intensities of the three models and the GPUs in mind, I drew the following experimental hypothesis.

### c. EXPERIMENT HYPOTHESIS

#### i. NVIDIA V100 vs NVIDIA A100:

A100 will see better performance of the three models than its counterpart V100, because the operational intensity to achieve maximum performance is significantly lower in A100 as compared to V100. It is easier for the models to meet this threshold since it is a lower value, and they will be able to utilize the A100 system in a more efficient way.

#### ii. V100:

The minimum operational intensity to achieve peak performance on V100 is 17.4 FLOPS/Byte. The operational intensity of resnet18 is lower than this, so it will not perform well on the V100 kernel. I expect the other 2 models to achieve good (at least better than resnet18) performance on this system.

#### iii. A100:

The minimum operational intensity to achieve peak performance on V100 is 12.5 FLOPS/Byte. Since all three models have a higher operational intensity than this threshold value, I expect them to have a comparable performance.

### d. EXPERIMENT SETUP AND EXECUTION PROCEDURES

I executed the three models on the 2 GPUs on the Greene HPC cluster of NYU.

ssh ns4451@gw.hpc.nyu.edu

ssh ns4451@greene.hpc.nyu.edu

ssh burst

I started the following two interactive GPU nodes using the following commands: (as shared by the TA)

```
srun --account=csci_ga_3033_085_2022sp --partition=n1s8-v100-1 --gres=gpu:v100:1 --pty /bin/bash
```

```
srun --account=csci_ga_3033_085_2022sp --partition=c12m85-a100-1 --gres=gpu:a100:1 --pty /bin/bash
```

I cloned the git repo [6] in a folder /proj1, exported the CUDA setting into the \$PATH variable and invoked the singularity container using the following command (as shared by the TA)

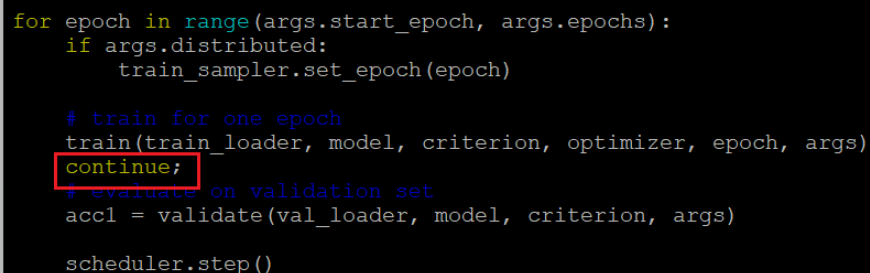
*(--nv is required to access the GPU inside the singularity container)*

```
git clone https://github.com/pytorch/examples
```

```
export PATH=$PATH:/share/apps/cuda/11.1.74/bin
```

```
singularity exec --bind /share/apps/cuda --overlay /share/apps/datasets/imagenet/imagenet-train.sqf:ro --overlay /share/apps/datasets/imagenet/imagenet-val.sqf:ro --nv /share/apps/images/cuda11.0-cudnn8-devel-ubuntu18.04.sif /bin/bash
```

I navigated to the imagenet folder inside the cloned /examples folder. This has the main.py – the python script that can accept the model it wants to run as input and starts training the model. I had to change the code in order to skip the validation process entirely, and to training only on a subset of the training dataset. I ran the code for 1 epoch at seed 1024.



```
for epoch in range(args.start_epoch, args.epochs):
    if args.distributed:
        train_sampler.set_epoch(epoch)

    # train for one epoch
    train(train_loader, model, criterion, optimizer, epoch, args)
    continue;
    # evaluate on validation set
    acc1 = validate(val_loader, model, criterion, args)

    scheduler.step()
```

Figure 2 Adding continue to skip validation set

```
for i, (images, target) in enumerate(train_loader):
    # measure data loading time
    data_time.update(time.time() - end)

    if args.gpu is not None:
        images = images.cuda(args.gpu, non_blocking=True)
    if torch.cuda.is_available():
        target = target.cuda(args.gpu, non_blocking=True)

    # compute output
    output = model(images)
    loss = criterion(output, target)

    # measure accuracy and record loss
    acc1, acc5 = accuracy(output, target, topk=(1, 5))
    losses.update(loss.item(), images.size(0))
    top1.update(acc1[0], images.size(0))
    top5.update(acc5[0], images.size(0))

    # compute gradient and do SGD step
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # measure elapsed time
    batch_time.update(time.time() - end)
    end = time.time()

    if i % args.print_freq == 0:
        progress.display(i)
        break;
```

Figure 3 Adding break to skip training on the entire training set

I executed the following command to run the python script main.py. \$MODEL is the model I want to run that I pass as an argument.

```
python3 main.py /imagenet --arch $MODEL --epochs 1 --batch-size 10 --print-freq 10 --seed 1024
```

### 3. Complexity Estimation and Measurement

#### a. COMPLEXITY ESTIMATION

I have touched upon complexity estimation in part 2b.

#### b. COMPLEXITY MEASUREMENT

For this, I used the ncu tool to profile the total FLOPS and memory throughput, using which I can calculate the Operational Intensity.

```
dram__sectors_write.sum
dram__bytes_write.sum.per_second
dram__sectors_read.sum
dram__bytes_read.sum.per_second
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
```

With these figures in hand, I was able to calculate the operational intensity and the attainable FLOPS per second using the following formula:

$$\text{flop\_count\_sp} = \text{smsp\_sass\_thread\_inst\_executed\_op\_fadd\_pred\_on.sum} + \\ \text{smsp\_sass\_thread\_inst\_executed\_op\_fmul\_pred\_on.sum} + \\ \text{smsp\_sass\_thread\_inst\_executed\_op\_ffma\_pred\_on.sum} * 2$$

$$\text{Operational Intensity} = \text{flop\_count\_sp} / ((\text{dram\_read\_transactions} + \text{dram\_write\_transactions}) * 32)$$

$$\text{Attainable flop/s} = \text{flop\_count\_sp} / \text{gpu\_runtime}$$

I used the nsys tool to profile the total model runtime.

#### c. RESULTS

To revise, I have also placed the theoretical values we had got for the three models in the following table.

I got the following results for the three models:

##### ***V100 GPU Results***

Model	Operational Intensity (FLOPS /Byte)	Attainable Computation (TFLOPS / sec)	Theoretical Operational Intensity (FLOPS /Byte)
resnet18	24.53	6.51	13.3
resnet34	27.67	6.25	26.6
resnet50	18.78	6.43	25.3

Figure 4 V100 results for the 3 models

***A100 GPU Results***

Model	Operational Intensity (FLOPS /Byte)	Attainable Computation (TFLOPS / sec)	Theoretical Operational Intensity (FLOPS /Byte)
resnet18	22.13	6.04	13.3
resnet34	23.89	6.05	26.6
resnet50	18.37	7.3	25.3

*Figure 5: A100 results for the three models***d. OBSERVATIONS*****V100 Observations***

The operational intensity we got for resnet18 and resnet34 is higher than the calculated estimated theoretical value. The actual operational intensity that I observed is higher than the system threshold. This implies that there may be some overhead that we are not accounting for during our calculations. Resnet34 has the maximum operational intensity, and it is higher than its theoretical value as well. Resnet50 has given a lower operational intensity than its theoretical expected value. A possible reason I can attribute this to is that the architecture of resnet34 is a complicated one requiring special computations in a few layers. This might be getting handled by some optimizations in the GPU. It is noteworthy that its operational intensity has not fallen below the V100's threshold value, indicating good performance. In terms of maximum attainable computation, resnet18 achieves the highest value.

***A100 Observations***

My hypothesis with regard to A100 GPU was a good comparable performance shown by all the models. Again, resnet18 achieves much higher operational intensity than its theoretical value. The reason would be the same as stated for the V100 system. Similar to the case of V100, resnet34 achieves the maximum operational intensity of the three models. The theoretical value of resnet50 is higher than what is actually observed, and the reason could be GPU optimizations involved in its complex architecture as discussed in the case of V100. In terms of maximum attainable computation, for A100 GPU, resnet50 achieves the highest value. Overall, the maximum computation attained by the three models is higher in the case of the A100 GPU.



### e. ROOFLINE GRAPHS

As discussed above, roofline graphs typically consist of performance related “roofs”, derived from the processor’s peak performance and memory bandwidth. Both the axes in the graph are taken in the logarithmic scale. I have used the Desmos Calculator[7] to plot values for GPUs and the three models. The roofline graphs for the 3 models as observed on NVIDIA V100 and NVIDIA A100 GPUs is as follows:

#### i. Roofline Graph for the 3 models on NVIDIA V100

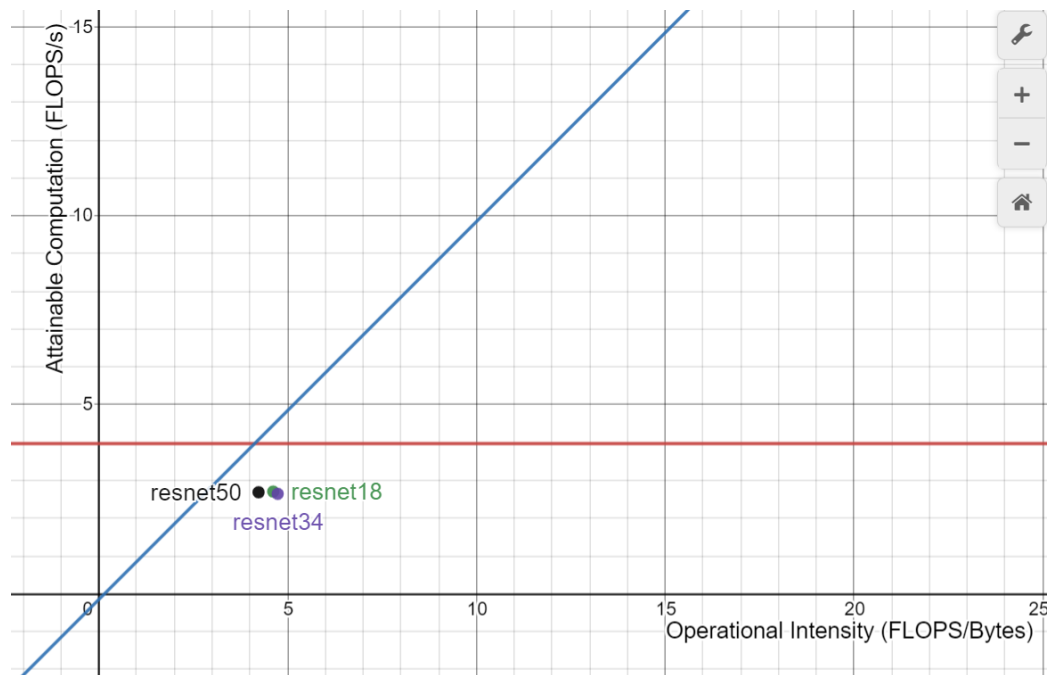


Figure 6: Roofline graph for Resnet18, Resnet34 and Resnet50 on NVIDIA V100

As observed in the table, Resnet50 had the lowest operational intensity. All three models achieve operational intensities above the threshold, but the resnet50 model has a value very close to the threshold. As we can see in the graph, it has the poorest performance of the three models. The performance of resnet34 and resnet18 is comparable, since both are above the system threshold and have a similar maximum attainable computation value.

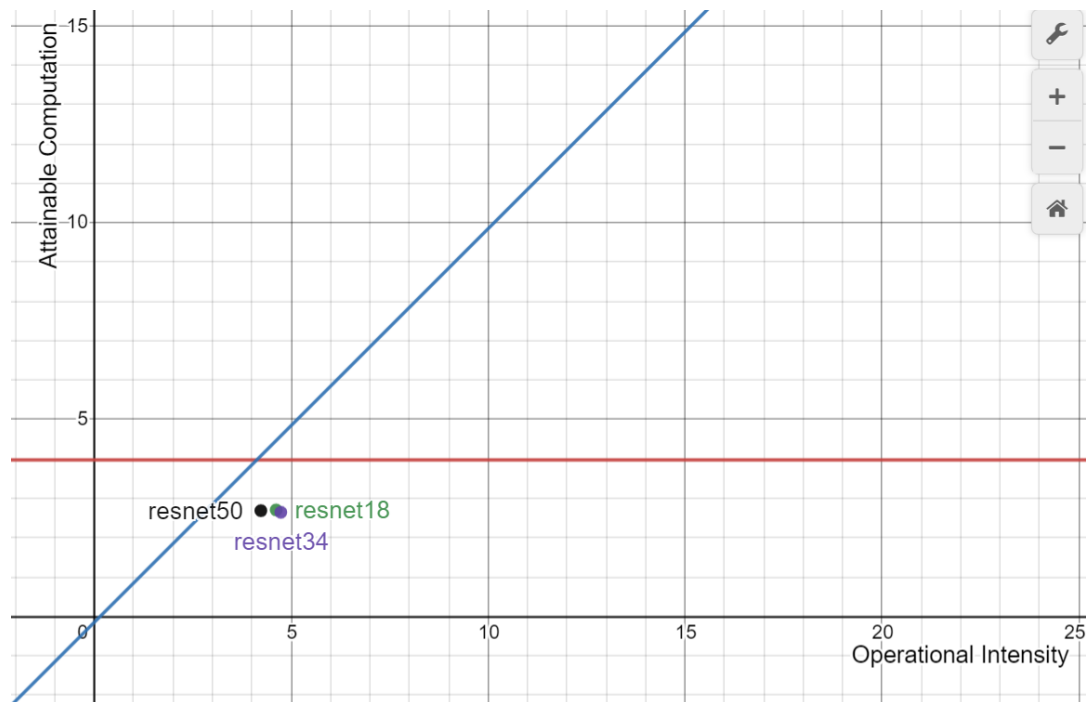
**ii. Roofline Graph for the 3 models on NVIDIA A100**

Figure 7: Roofline graph for Resnet18, Resnet34 and Resnet50 on NVIDIA A100

All the three models have operational intensity above the threshold frequency, hence they are attaining comparable performance. The low threshold of NVIDIA A100 allows resnet50 to not be restricted by the memory bandwidth, as opposed to V100 case. The trend we see on the 2 GPUs with respect to the attainable computation remains the same for all the three models.

#### 4. Discussion

The roofline model gives us a good estimate of performance on a specific machine. It gives an opportunity early in the code implementation stage to take advantage of the system performance on which we will run the code. Similarly, even in this experiment, I first found the threshold operational intensity of both the GPUs. This denoted the minimum operational intensity that any model should have in order for the model to fully exploit the system resources. For this, I first calculated the threshold operational intensities of both the GPUs. Then I calculated the expected operational intensities of the three models (using pen and paper method). Based on the OIs, I presented a hypothesis and gave an estimation of the models I thought would achieve better performance on both the GPUs.

From my observations, in a few cases, the actual Operational Intensity obtained by profiling is lower than the operational intensity observed (resnet18 on both the GPUs). I attributed this behaviour to some kind of overhead that was not accounted for during the theoretical calculations. In this case, the GPU optimizations is not able to handle this overcharge. On the contrary, there are a few cases where the theoretical value is greater than the actual one achieved (like resnet50 on both GPUs). This can be because of the complicated architecture of resnet50 layers which the GPU automatically handles because of the optimizations in it.

The values for TF32 as observed in V100 and A100 systems were very small. They seemed insignificant as a measure basis on which we can perform our analysis. I got the FP32 values more easily for both the GPU systems, and hence have carried out my analysis based on that.

The operational intensity of any model should ideally depend on just the model architecture. However, the ridge point or the transition point that we get on the roofline graphs of GPUs affect the performance of the models and cause them to differ on different systems. Hence, it is because of the ridge point that we see a difference in the value of operational intensity and maximum attainable compute of different models on 2 separate GPUs. This furthermore increases my confidence in roofline modelling being a good performance indicator, and surpassing its alternatives (like percent of peak theoretical).

## 5. References

- [1] [https://en.wikipedia.org/wiki/Roofline\\_model](https://en.wikipedia.org/wiki/Roofline_model)
- [2] <https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf>
- [3] <https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf>
- [4] <https://github.com/albanie/convnet-burden>
- [5] <https://towardsdatascience.com/understanding-and-calculating-the-number-of-parameters-in-convolution-neural-networks-cnns-fc88790d530d>
- [6] <https://github.com/pytorch/examples>
- [7] <https://www.desmos.com/calculator>