PROJECT 1 - IMAGENET ANALYSIS

1. INTRODUCTION

As demand and innovation of deep learning models grows, we can see a growing trend of training these models on bigger datasets. Training of such deep models on such big datasets often takes days and is also very resource heavy. However, inefficient training and model implementations also adds to this.
Profiling and roofline modeling are invaluable tools.Roofline Modeling helps in understanding the limits and potential of systems.By plotting performance against hardware capabilities,engineers can identify bottlenecks and optimize code accordingly.This report,created for *Project-1 of Cloud & Machine Learning* course, aims to analyze the performance of ImageNet on various GPU's with different model architectures(VGG family of networks).

2. ROOFLINE MODELLING

Roofline modeling is a performance analysis methodology used to visualize and analyze the performance of models on a given hardware or system. The roofline model combines the performance of a computation with the capabilities of the hardware. The roofline represents the performance benchmark dictated by the hardware's peak capabilities.Roofline modeling helps identify performance bottlenecks and understand the efficiency of the deep learning models.
In a roofline model (*Figure 1*) the performance is plotted on the y-axis, and arithmetic intensity (FLOP per byte) is plotted on the x-axis. Arithmetic Intensity is defined as the ratio of total floating point operations to the total data movement.

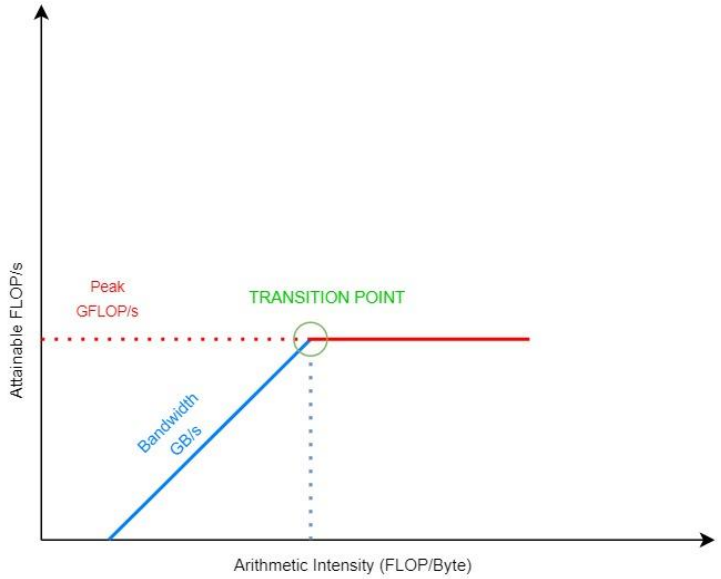


Figure 1: Roofline Model

$$Arithmetic\ Intensity = \frac{\#\ of\ Floating\ point\ operation}{Total\ Data\ Movement}$$

In *Figure 1*, the transition point is where the Peak GFLOP/s or performance is equal to AI * Peak GB/s. In such a situation the machine is said to be balanced.If the application's performance (FLOPs/second) is close to the peak performance indicated by the roofline (the upper bound), it is considered compute-bound. Here,the computation capability of the hardware is the limiting factor, and the application is making effective use of the available computational resources.If the application's performance is limited by memory bandwidth, it is considered memory-bound.This happens when the arithmetic intensity is low, meaning the application performs fewer FLOPs per byte of data moved between the memory and the processor.In a memory-bound, improving the algorithm's computational intensity or optimizing memory access patterns may enhance performance.

3. VGG FAMILY OF NETWORKS

The VGG (Visual Geometry Group) family of networks consists of convolutional neural networks built for image classification. VGG was first introduced at the University of Oxford in 2014. These networks are known for its simple architecture and uniformity over layers. VGG networks consist of multiple convolutional layers where the kernel size is 3x3. When such a kernel is applied again and again, the region in the input image that a particular layer is sensitive to remains constant. This allows the network to learn both global and local features. A small filter of size 3x3 also enables parameter sharing. Since the same filter is applied across different spatial locations, the network learns to detect similar patterns in various parts of the image.

a) VGG11 consists of 11 layers- 8 convolutional layers and 3 fully connected layers
b) VGG13 consists of 13 layers- 10 convolutional layers and 3 fully connected layers
c) VGG19 consists of 19 layers- 16 convolutional layers and 3 fully connected layers

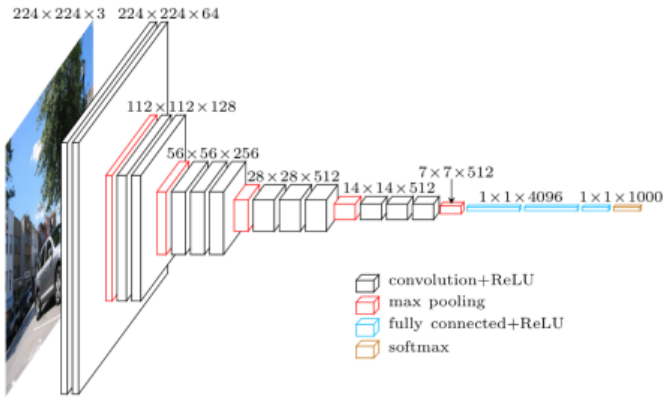
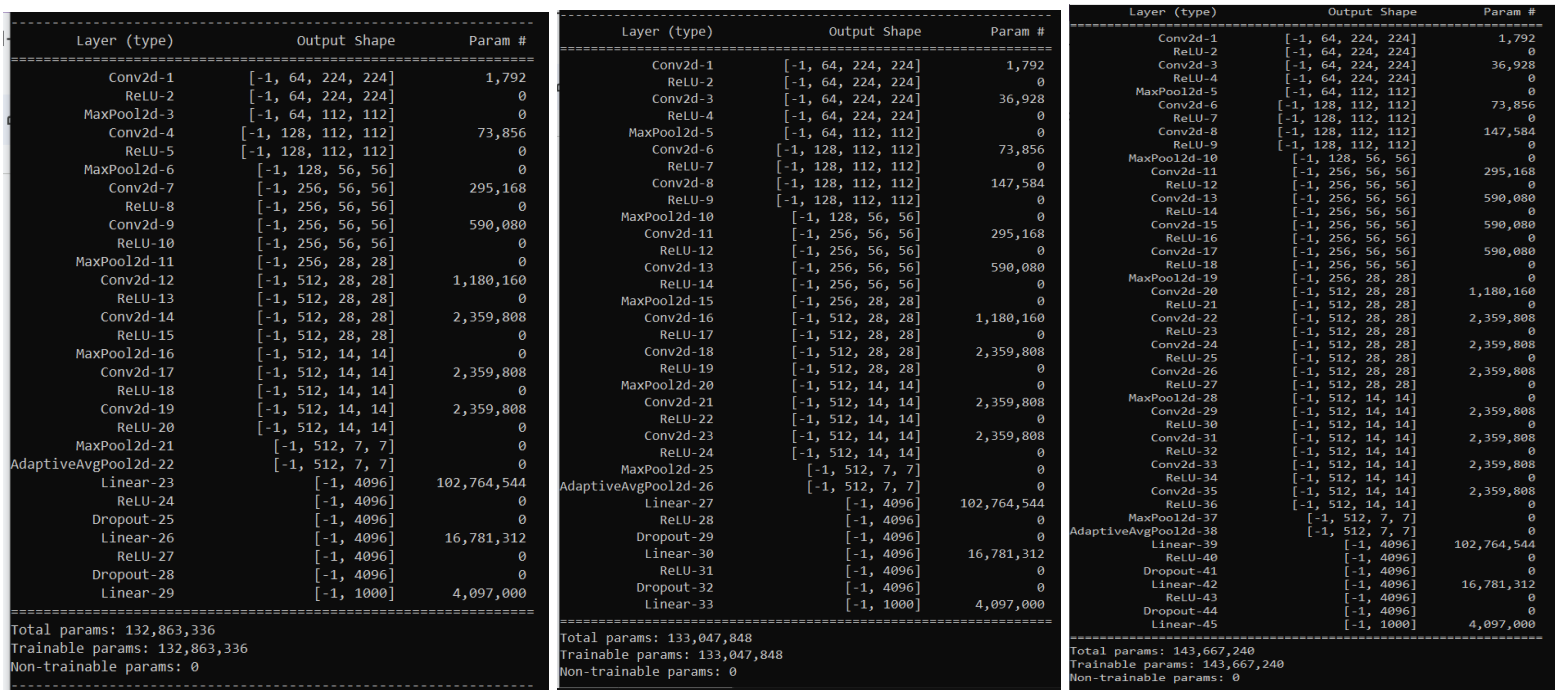


Figure 2: VGG11 model[8]



Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 224, 224]	1,792
ReLU-2	[-1, 64, 224, 224]	0
MaxPool2d-3	[-1, 64, 112, 112]	0
Conv2d-4	[-1, 128, 112, 112]	73,856
ReLU-5	[-1, 128, 112, 112]	0
MaxPool2d-6	[-1, 128, 56, 56]	0
Conv2d-7	[-1, 256, 56, 56]	295,168
ReLU-8	[-1, 256, 56, 56]	0
Conv2d-9	[-1, 256, 56, 56]	590,080
ReLU-10	[-1, 256, 56, 56]	0
MaxPool2d-11	[-1, 256, 28, 28]	0
Conv2d-12	[-1, 512, 28, 28]	1,180,160
ReLU-13	[-1, 512, 28, 28]	0
Conv2d-14	[-1, 512, 28, 28]	2,359,808
ReLU-15	[-1, 512, 28, 28]	0
MaxPool2d-16	[-1, 512, 14, 14]	0
Conv2d-17	[-1, 512, 14, 14]	2,359,808
ReLU-18	[-1, 512, 14, 14]	0
Conv2d-19	[-1, 512, 14, 14]	2,359,808
ReLU-20	[-1, 512, 14, 14]	0
MaxPool2d-21	[-1, 512, 7, 7]	0
AdaptiveAvgPool2d-22	[-1, 512, 7, 7]	0
Linear-23	[-1, 4096]	102,764,544
ReLU-24	[-1, 4096]	0
Dropout-25	[-1, 4096]	0
Linear-26	[-1, 4096]	16,781,312
ReLU-27	[-1, 4096]	0
Dropout-28	[-1, 4096]	0
Linear-29	[-1, 1000]	4,097,000

Total params: 132,863,336
Trainable params: 132,863,336
Non-trainable params: 0

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 224, 224]	1,792
ReLU-2	[-1, 64, 224, 224]	0
Conv2d-3	[-1, 64, 224, 224]	36,928
ReLU-4	[-1, 64, 224, 224]	0
MaxPool2d-5	[-1, 64, 112, 112]	0
Conv2d-6	[-1, 128, 112, 112]	73,856
ReLU-7	[-1, 128, 112, 112]	0
Conv2d-8	[-1, 128, 112, 112]	147,584
ReLU-9	[-1, 128, 112, 112]	0
MaxPool2d-10	[-1, 128, 56, 56]	0
Conv2d-11	[-1, 256, 56, 56]	295,168
ReLU-12	[-1, 256, 56, 56]	0
Conv2d-13	[-1, 256, 56, 56]	590,080
ReLU-14	[-1, 256, 56, 56]	0
MaxPool2d-15	[-1, 256, 28, 28]	0
Conv2d-16	[-1, 512, 28, 28]	1,180,160
ReLU-17	[-1, 512, 28, 28]	0
Conv2d-18	[-1, 512, 28, 28]	2,359,808
ReLU-19	[-1, 512, 28, 28]	0
MaxPool2d-20	[-1, 512, 14, 14]	0
Conv2d-21	[-1, 512, 14, 14]	2,359,808
ReLU-22	[-1, 512, 14, 14]	0
Conv2d-23	[-1, 512, 14, 14]	2,359,808
ReLU-24	[-1, 512, 14, 14]	0
MaxPool2d-25	[-1, 512, 7, 7]	0
AdaptiveAvgPool2d-26	[-1, 512, 7, 7]	0
Linear-27	[-1, 4096]	102,764,544
ReLU-28	[-1, 4096]	0
Dropout-29	[-1, 4096]	0
Linear-30	[-1, 4096]	16,781,312
ReLU-31	[-1, 4096]	0
Dropout-32	[-1, 4096]	0
Linear-33	[-1, 1000]	4,097,000

Total params: 133,047,848
Trainable params: 133,047,848
Non-trainable params: 0

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 224, 224]	1,792
ReLU-2	[-1, 64, 224, 224]	0
Conv2d-3	[-1, 64, 224, 224]	36,928
ReLU-4	[-1, 64, 224, 224]	0
MaxPool2d-5	[-1, 64, 112, 112]	0
Conv2d-6	[-1, 128, 112, 112]	73,856
ReLU-7	[-1, 128, 112, 112]	0
Conv2d-8	[-1, 128, 112, 112]	147,584
ReLU-9	[-1, 128, 112, 112]	0
MaxPool2d-10	[-1, 128, 56, 56]	0
Conv2d-11	[-1, 256, 56, 56]	295,168
ReLU-12	[-1, 256, 56, 56]	0
Conv2d-13	[-1, 256, 56, 56]	590,080
ReLU-14	[-1, 256, 56, 56]	0
Conv2d-15	[-1, 256, 56, 56]	590,080
ReLU-16	[-1, 256, 56, 56]	0
Conv2d-17	[-1, 256, 56, 56]	590,080
ReLU-18	[-1, 256, 56, 56]	0
MaxPool2d-19	[-1, 256, 28, 28]	0
Conv2d-20	[-1, 512, 28, 28]	1,180,160
ReLU-21	[-1, 512, 28, 28]	0
Conv2d-22	[-1, 512, 28, 28]	2,359,808
ReLU-23	[-1, 512, 28, 28]	0
Conv2d-24	[-1, 512, 28, 28]	2,359,808
ReLU-25	[-1, 512, 28, 28]	0
Conv2d-26	[-1, 512, 28, 28]	2,359,808
ReLU-27	[-1, 512, 28, 28]	0
MaxPool2d-28	[-1, 512, 14, 14]	0
Conv2d-29	[-1, 512, 14, 14]	2,359,808
ReLU-30	[-1, 512, 14, 14]	0
Conv2d-31	[-1, 512, 14, 14]	2,359,808
ReLU-32	[-1, 512, 14, 14]	0
Conv2d-33	[-1, 512, 14, 14]	2,359,808
ReLU-34	[-1, 512, 14, 14]	0
Conv2d-35	[-1, 512, 14, 14]	2,359,808
ReLU-36	[-1, 512, 14, 14]	0
MaxPool2d-37	[-1, 512, 7, 7]	0
AdaptiveAvgPool2d-38	[-1, 512, 7, 7]	0
Linear-39	[-1, 4096]	102,764,544
ReLU-40	[-1, 4096]	0
Dropout-41	[-1, 4096]	0
Linear-42	[-1, 4096]	16,781,312
ReLU-43	[-1, 4096]	0
Dropout-44	[-1, 4096]	0
Linear-45	[-1, 1000]	4,097,000

Total params: 143,667,240
Trainable params: 143,667,240
Non-trainable params: 0

Figure 2: (Left to Right) vgg11, vgg13 an vgg19 model summary on input of size (3,224,224)

4. METHODOLOGY

The methodology I implemented is as follows:

I. Create a small dataset of ImageNet Data. (This data consists of a subset of train and validation data where each image is of size 3x224x224)
II. Select Models for Analysis (The models selected are vgg11,vgg13 and vgg19)
III. Pen & Paper Estimation of FLOP and memory for selected networks.
IV. Profile using NCU to measure FLOP and DRAM memory.
V. Profile using NVIDIA Nsight to measure time taken for application/kernel run.
Log Files are as follows:

```
|--A100 (consists of logs of experiments run on a100 gpu)
        |--Vgg11
                |--ncu-final-metric-vgg11.txt (Sum of metrics)
                |--ncu-log-vgg11.txt ( ncu log file)
                |--nsys-final-metric-vgg11.txt ( nsys total time)
                |-- tmp_nsys_gputrace.csv (nsys trace)
        |--Vgg13
        |-- Vgg19
|--V100
        |--Vgg11
        |--Vgg13
        |--Vgg19
```

VI. Roofline Modelling

5. PEN & PAPER ESTIMATION

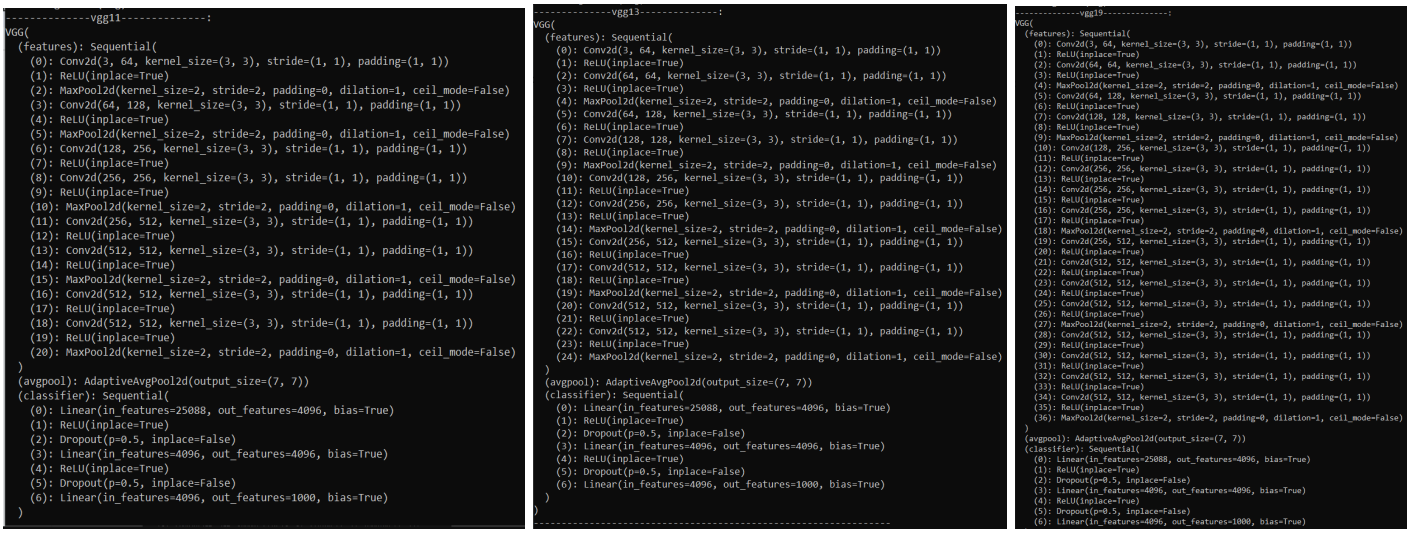


Figure 3: Network Layers Summary (Left to right: vgg11,vgg13 an vgg19)

We can consider the Convolution layers and linear layers for FLOP estimation[4]:

$$FLOP_{conv} = output\_image\_shape * kernel\ shape * input\_channels * output\_channels$$

$$FLOP_{linear} = input\_features * output\_features$$

$$FLOP_{total} = Batch\ size * (\Sigma FLOP_{linear} + \Sigma FLOP_{conv})$$

A) VGG11

$$Total\ FLOP = 121745440768 \sim 121.17 * 10^9$$

Layer	Input channels	Output channels	Kernel shape	Output image shape	FLOP
Conv2d-1	3	64	3	224	173408256
Conv2d-4	64	128	3	112	1849688064
Conv2d-7	128	256	3	56	1849688064
Conv2d-9	256	256	3	56	3699376128
Conv2d-12	256	512	3	28	1849688064
Conv2d-14	512	512	3	28	3699376128
Conv2d-17	512	512	3	14	924844032
Conv2d-19	512	512	3	14	924844032

Linear-23	25088	4096	1	1	205520896
Linear-26	4096	4096	1	1	33554432
Linear-29	4096	1000	1	1	8192000

Table 1: Vgg11 pen & paper estimation

B) VGG 13

$$Total\ FLOP = 58856046592 \sim 58.85 * 10^9$$

Layer	Input channels	Output channels	Kernel shape	Output image shape	FLOP
Conv2d-1	3	64	3	224	86704128
Conv2d-3	64	64	3	112	462422016
Conv2d-6	64	128	3	56	231211008
Conv2d-8	128	128	3	56	462422016
Conv2d-11	128	256	3	28	231211008
Conv2d-13	256	256	3	28	462422016
Conv2d-16	256	512	3	14	231211008
Conv2d-18	512	512	3	14	462422016
Conv2d-21	512	512	3	14	462422016
Conv2d-23	512	512	3	14	462422016
Linear-27	25088	4096	1	1	102760448
Linear-30	4096	4096	1	1	16777216
Linear-33	4096	1000	1	1	4096000

Table 2: Vgg13 pen & paper estimation

C) **VGG 19**

$$Total\ FLOP = 314112999424 \sim 314.11 * 10^9$$

Layer	Input channels	Output channels	Kernel shape	Output image shape	FLOP
Conv2d-1	3	64	3	224	86704128
Conv2d-3	64	64	3	224	1849688064
Conv2d-6	64	128	3	112	924844032
Conv2d-8	128	128	3	112	1849688064
Conv2d-11	128	256	3	56	924844032
Conv2d-13	256	256	3	56	1849688064
Conv2d-15	256	256	3	56	1849688064
Conv2d-17	256	256	3	56	1849688064
Conv2d-20	256	512	3	28	924844032
Conv2d-22	512	512	3	28	1849688064
Conv2d-24	512	512	3	28	1849688064
Conv2d-26	512	512	3	28	1849688064
Conv2d-29	512	512	3	14	462422016
Conv2d-31	512	512	3	14	462422016
Conv2d-33	512	512	3	14	462422016
Conv2d-35	512	512	3	14	462422016
Linear-39	25088	4096	1	1	102760448
Linear-42	4096	4096	1	1	16777216
Linear-45	4096	1000	1	1	4096000

Table 3: Vgg19 pen & paper estimation

6. ANALYSIS

$$Arithmetic\ Intensity = \frac{\#\ of\ Floating\ point\ operation}{Total\ Data\ Movement}$$

$$Performance = \frac{\#\ of\ Floating\ point\ operation}{Time}$$

Using the measured metrics, we can calculate the arithmetic intensity and the performance for each network and both the hardware(V100 and A100).

GPU: V100					
MODEL	***FLOP***	***MEMORY (Bytes)***	***TIME (nanosecond)***	***ARITHMETIC INTENSITY (FLOP/Bytes)***	***PERFORMANCE (GFLOP/s)***
VGG11	221705941507	16568700000	163316785	13.381	1357.521
VGG13	390284311131	24944187520	182278578	15.646	2141.142
VGG19	602786338165	34168748020	223432104	17.641	2697.852

Table 4: V100 Profiling

We can see here that the measured FLOP is comparable to the pen and paper estimated FLOP that can be seen in *Table 1,2 and 3*.

GPU: A100					
MODEL	***FLOP***	***MEMORY (Bytes)***	***TIME (nanosecond)***	***ARITHMETIC INTENSITY (FLOP/Bytes)***	***PERFORMANCE (GFLOP/s)***
VGG11	11771359163	9790409380	141306033	1.202	83.304
VGG13	12628475297	16365861630	153713319	0.77	82.156
VGG19	13162802109	19771592960	176424599	0.6657	74.60

Table 5: A100 Profiling

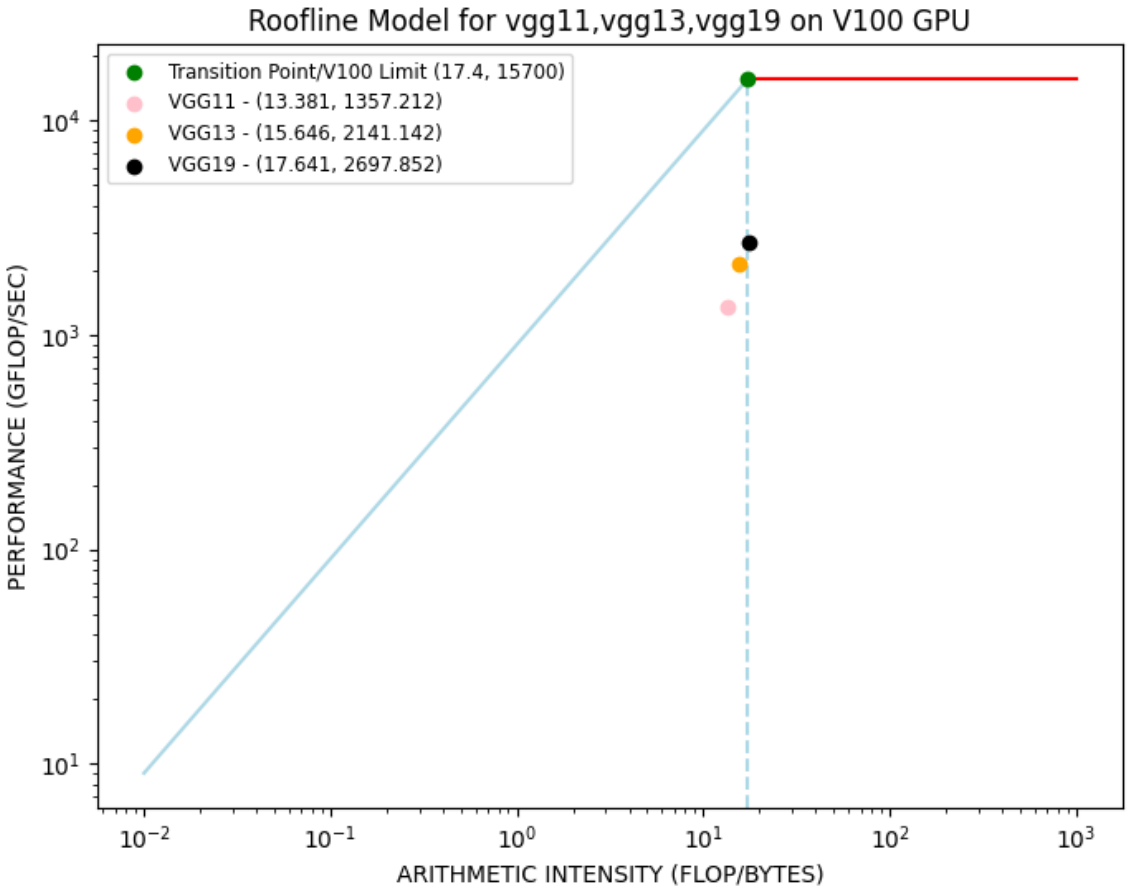
Using the NVIDIA Specification sheet of the GPU, we can get the peak Performance and bandwidth. The Peak arithmetic intensity can also be calculated.

Metric	**V100**	**A100**
Single-Precision Performance	15.7 TFLOPs	19.5 TFLOPs
Memory Bandwidth	900 GB/s	1555 GB/s
Peak AI: (FLOP/Byte) *(Peak Performance/Peak Memory Bandwidth)*	17.4	12.5

Table 6: V100 & A100 GPU Specification

A) V100 Roofline Model

From *Graph 1*, we can see that VGG11,VGG13 and VGG19 all lie near the vertical line which passes through the transition point. VGG19 lies on this vertical line. These points are just below the transition point and are not very far from it. This suggests that VGG19 and VGG13 achieve a good balance between computation and memory access. These models still lie below the peak performance and bandwidth of the V100 machine but this graph shows that they are generally performing very well . VGG11 on the other hand is more memory bound than balanced which suggests that the memory access patterns of this application can be optimized so that it is closer to the peak performance and peak bandwidth of the V100 GPU.

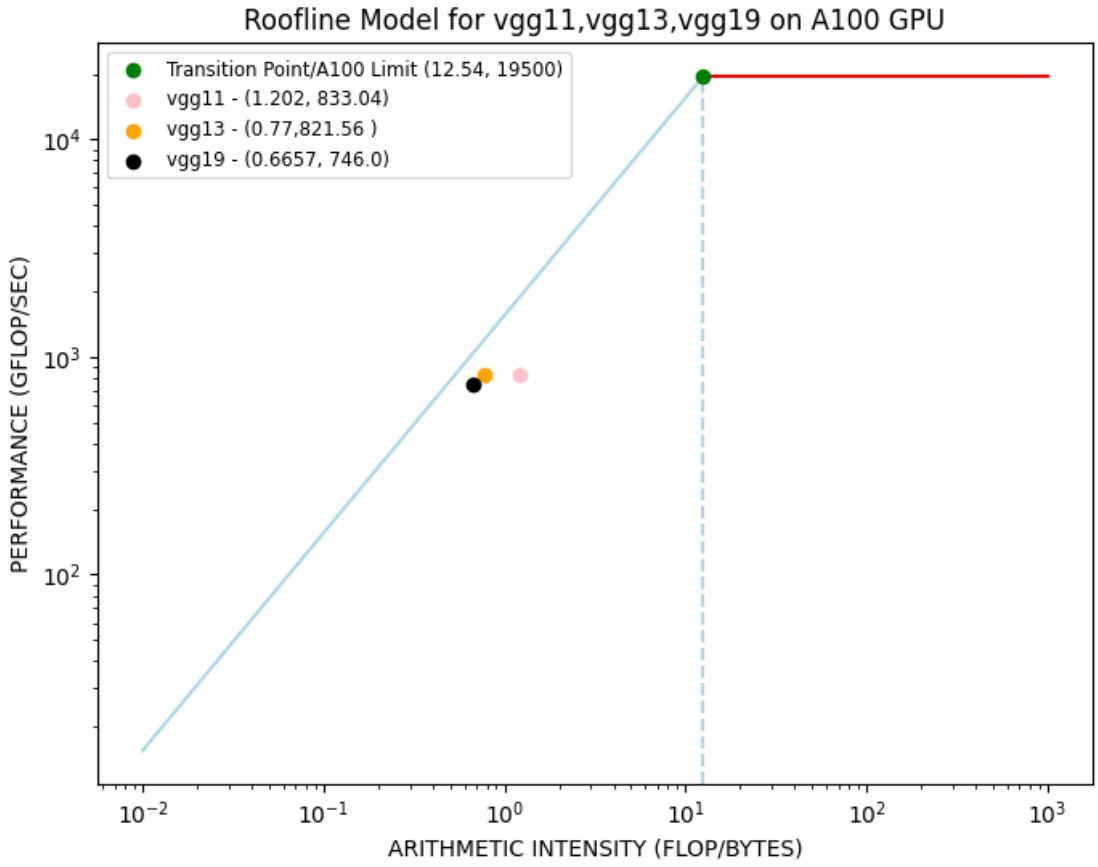


Graph 1: V100 Roofline Model

B) A100 Roofline Model

From Graph 2, we can see that points for all three architectures lie very close to each other. All these points are close to the line which has slope equal to the Peak Bandwidth of A100 GPU. This means that each of vgg11,vgg13 and vgg19 are memory bound on A100. This means that the arithmetic intensity of these applications is low and that they are not performing as many floating point operations per byte of data moved compared to the peak performance of A100. Since all these applications are memory bound, to improve performance, one can optimize the memory access patterns and reduce the amount of data transfer between the memory and the processor. VGG13 and VGG19 are closer to the peak bandwidth boundary than the VGG11 application which means that they do more transfers between memory and the processor. This might be true due to the fact that VGG19 and VGG13 are deeper networks with more convolutional, maxpool and ReLU layers. To increase performance here, more parallelization can also be

introduced. This would reduce the amount of memory access patterns done by the VGG models.



Graph 2: A100 Roofline Model

A100 v/s V100

For the selected models, V100 gives a much more balanced performance as compared to A100 even though A100 has better specifications which can be seen in Table 6. These applications, i.e vgg11,vgg13 and vgg19 can't make use of the high peak performance of the A100 GPU. Introducing data parallelization can help with this. For other architectures, it is possible that A100 is more balanced than V100. The roofline model is very specific to the application under consideration. When the VGG family of networks is considered, V100 gives a more balanced performance and should be preferred over A100.

7. CONCLUSION

In conclusion, for the selected family of networks V100 seems to perform better and is more balanced under the roofline model as compared to A100 GPU. This is specific to the models and the chosen batch size of 16. For VGG, V100 should be preferred. Roofline modeling thus is a very important tool which helps us analyze the performance of our deep learning models in comparison to the peak performance of the hardware. Roofline modeling can help indicate if the application should be parallelized or it should be optimized in terms of computations.

8. REFERENCES

1. *Pytorch Examples/Imagenet Repository* (https://github.com/pytorch/examples/tree/main/imagenet)
2. *NVIDIA V100 Specification Sheet* (https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf)
3. *NVIDIA A100 Specification Sheet* https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf
4. *Calculating MAC & FLOPS* (https://medium.com/@pashashaik/a-guide-to-hand-calculating-flops-and-macs-fa5221ce5ccc)
5. *Nvidia NCU* (https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html)
6. Very *Deep Convolutional Networks for Large-Scale Image Recognition (*https://arxiv.org/abs/1409.1556)
7. *Understanding Roofline Models* (https://dando18.github.io/posts/2020/04/02/roofline-model)
8. *Roofline: An Insightful Visual Performance Model for Floating-Point Programs and MulticoreArchitectures(*https://people.eecs.berkeley.edu/~kubitron/cs252/handouts/papers/RooflineVyNoYellow.pdf)