Deep Learning Hardware Profiling and Roofline Analysis

Alif Ilham Madani

April 16, 2026

1 Chip Analysis

In this section, we collected peak TFLOPs and memory bandwidth for 5 devices from [7, 5, 1, 2, 6], as shown in Table 1. We use BF16 FLOPS values stated in the datasheet, except for NVIDIA Grace CPU, where we use FP64 since they only state this value in the datasheet. To easily compare the performance of these devices, we present the roofline model in Figure 1. The CPU performance is far below the GPUs and TPUs as it has much lower peak FLOPS, even though it has comparable memory bandwidth to TPU v4. The performance of TPUs and GPUs are comparable both in earlier generations (TPU v4 and NVIDIA A100) and later generations (TPU7x and NVIDIA H100).

Table 1: Hardware Specifications: Peak TFLOPS and Memory Bandwidth

Hardware	**Peak TFLOPS**	**Memory Bandwidth (TB/s)**
NVIDIA H100 SXM	1979.0	3.350
NVIDIA A100 SXM	624.0	2.039
Google TPU7x	2307.0	7.380
Google TPU v4	275.0	1.200
NVIDIA Grace CPU	7.1	1.000

2 DNN Compute and Memory Analysis

Figure 2 shows the compute, memory footprint and operational intensity of 10 DNN models. We analyze 9 vision models and 1 audio model (Wav2Vec2) by measuring the number of FLOPs required in each model. Then, we measure the memory footprint by summing the model parameters, input size, and activation size. In this experiment, we use a batch size of 1 as the input but we also record batch size of 64 and 256 for the next sections. To get the operational intensity, we simply divide the FLOPs with the memory footprint.

We can see in Figure 2 that higher FLOPs do not mean higher memory footprint as shown by ResNet50 which has the highest FLOPs but not the highest memory footprint. This is due to the residual network, which involves computation from previous layers. VGG16 has the highest memory footprint due to its big layer size and deep network. On the lower spectrum, MobileNetV2 has the lowest operational intensity due to low FLOPs and low memory footprint, making it suitable for mobile devices.

We then analyze the operational intensity of each model using the roofline model of CPU and GPU accessible in Google Colab (Intel Xeon 1-core and NVIDIA T4). We were able to get the GPU FLOPS and memory bandwidth data from [8]. However, for the CPU, we estimate using the following calculation: 1 core x 3.2 GHz (turbo mode) x

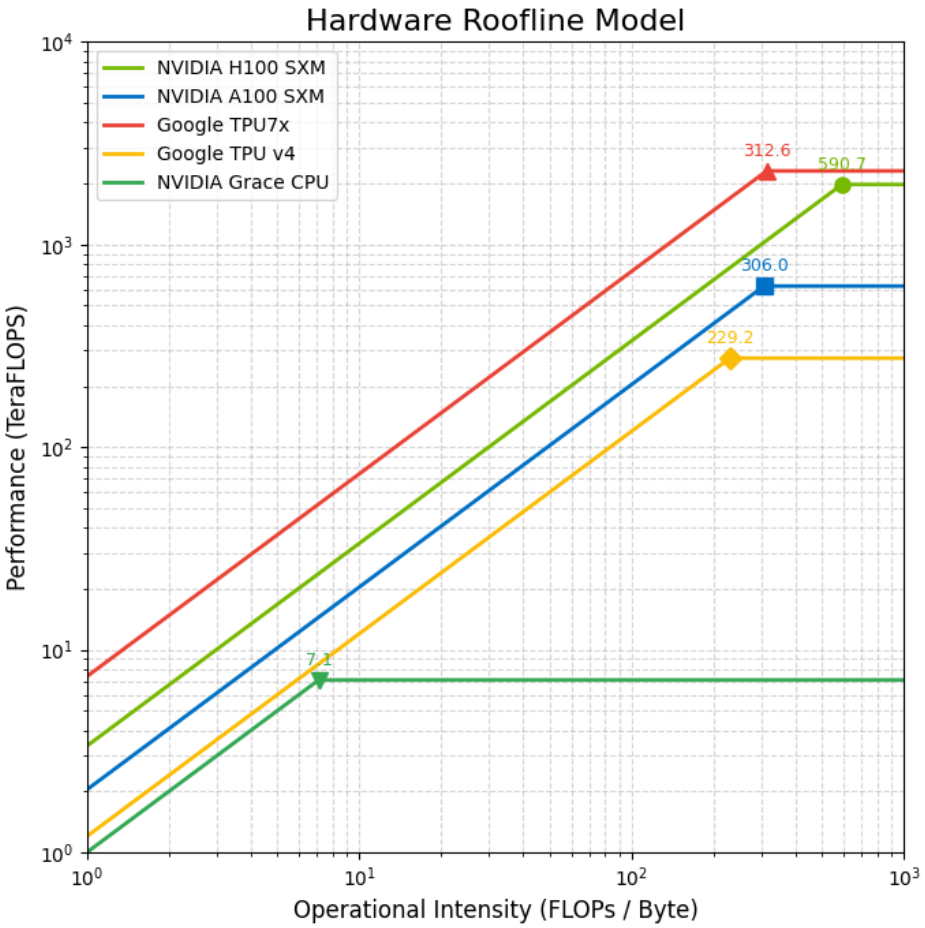

Figure 1: Roofline model of CPU, GPUs, and TPUs

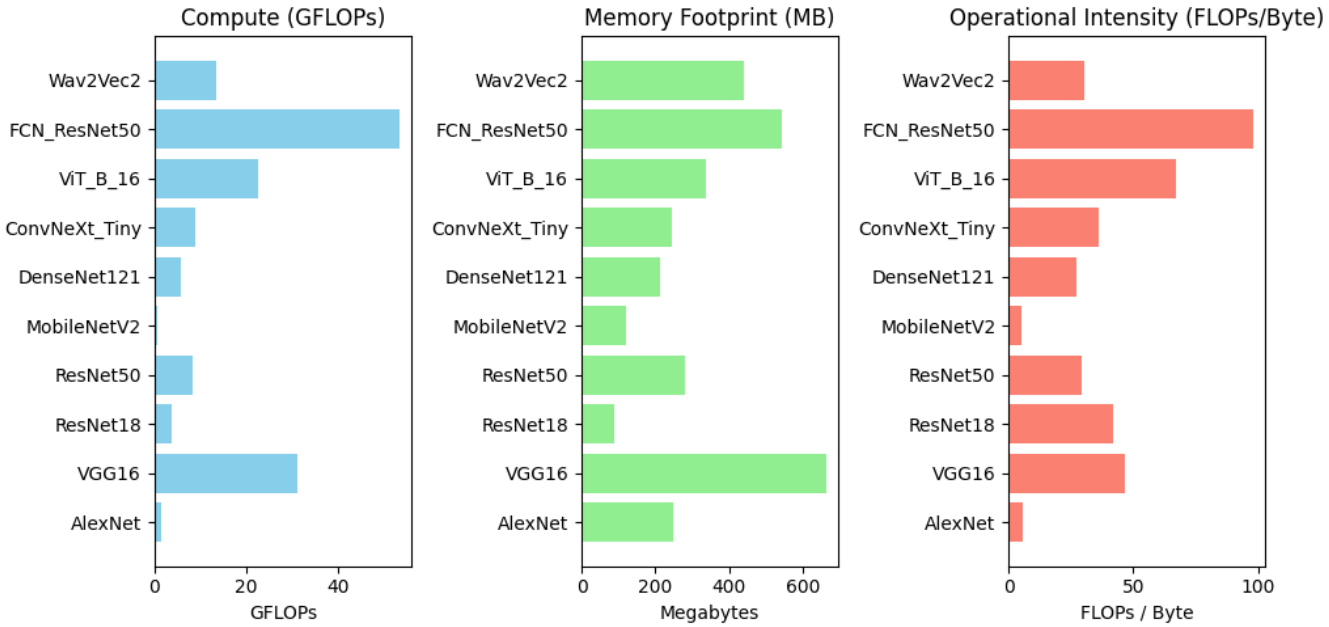

Figure 2: FLOPs, memory footprint and operational intensity of different DNN models

8 vector elements x 2 FMA/core x 2 operation/cycle = 102.4 GFLOPS peak. [4, 9] Table 2 shows the summarized specifications.

Table 2: Colab Free Tier Hardware Specifications

Hardware	**Peak GFLOPS**	**Memory Bandwidth (GB/s)**
CPU (Intel Xeon 1-core)	102.4	76.8
GPU (NVIDIA T4)	65000.0	320.0

As shown in Figure 3, all models will be compute-bound under CPU and memory-bound under GPU. This is expected since the CPU is not designed to train DNN models that have intense operations. On the GPU side, "memory-bound" means we can load more data in parallel to better utilize the device until it reaches peak performance. This can be done by increasing the batch size.

3 DNN Performance Benchmarking

In this section, we run inference on the 10 models using a batch size of 1, 64, and 256 on both CPU and GPU. Then, we measure the latency for each configuration. Figure 4 shows latency vs. FLOPs and latency vs. parameters plots. Figure 5 shows the Spearman correlation matrix between latency vs. FLOPs and latency vs. parameters. We can clearly see there is a strong correlation between CPU latency with GPU latency (0.89) and FLOPs but not with parameters (0.36). GPU latency also has strong correlation with FLOPs but not with parameters.

Based on this result, in general, FLOPs can be a good indicator to predict how fast the model runs since more computations require more time to complete. However, MobileNetV2 has higher latency with higher batch sizes

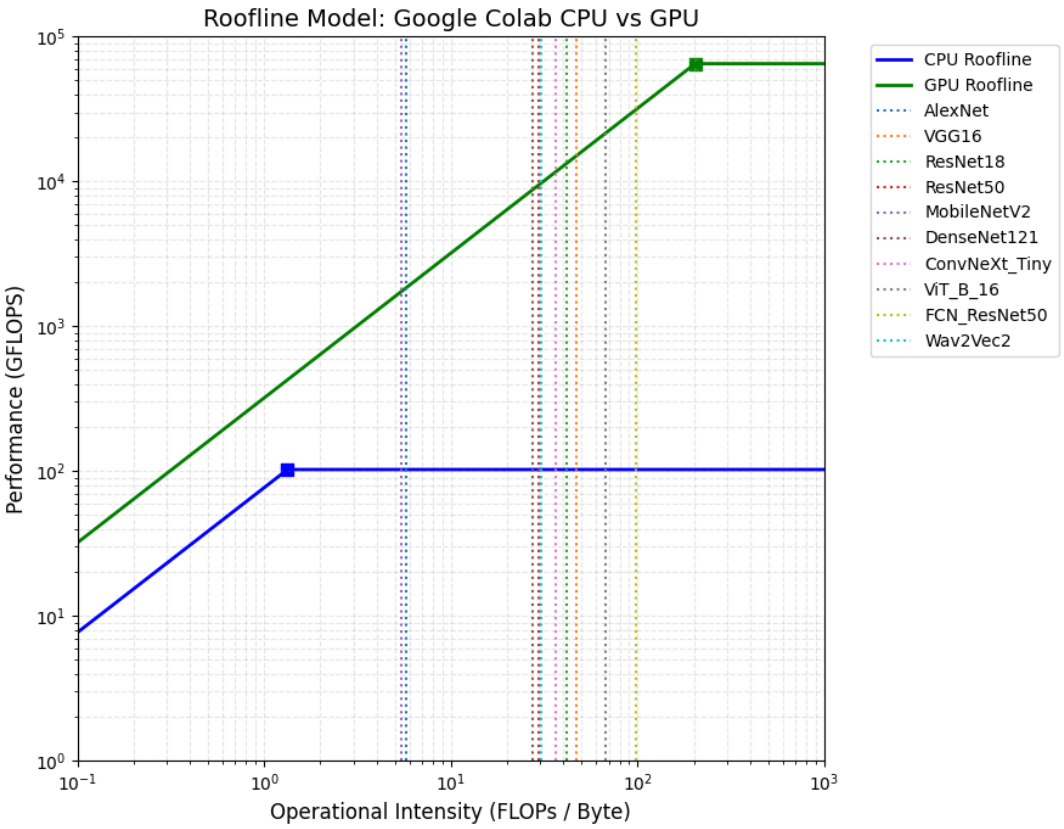

Figure 3: Roofline model of CPU and GPU along with various DNN profiles

Figure 4: Latency vs FLOP of different models

running on CPU, and it also shows a higher latency for all batch sizes running on GPU. Although the MobileNetV2 has the lowest FLOPs, it has the lowest latency on CPU only, which means it is optimized for mobile devices, which mainly run on CPU. Parameters is not a good indicator of latency since higher parameters does not mean more computation, as we discussed with residual networks. However, it can be a good indicator when all the workloads are memory-bound, meaning it takes longer time load the parameters than to complete the computations.

Figure 6 shows that throughput varies across models and batch sizes. AlexNet has higher throughput as the batch size increases on both CPU and GPU. For most vision models (VGG16, ResNet18, ResNet50, MobileNetV2, DenseNet121, ConvNeXt_Tiny, FCN_ResNet50), as the batch size increases, the throughput decreases on CPU but increases on GPU. For Wav2Vec2 model, the throughput increases on both CPU and GPU as the batch size increases. MobileNetV2 has much higher throughput on CPU using batch size of 1 because this model is optimized for mobile device inference, which usually uses a batch size of 1 running on CPU.

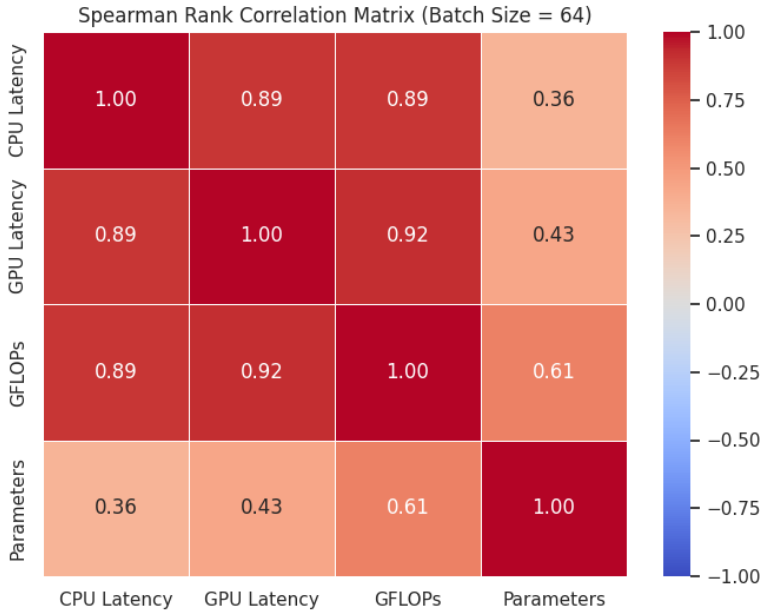

Figure 5: Correlation matrix of latency, parameters, and FLOPs

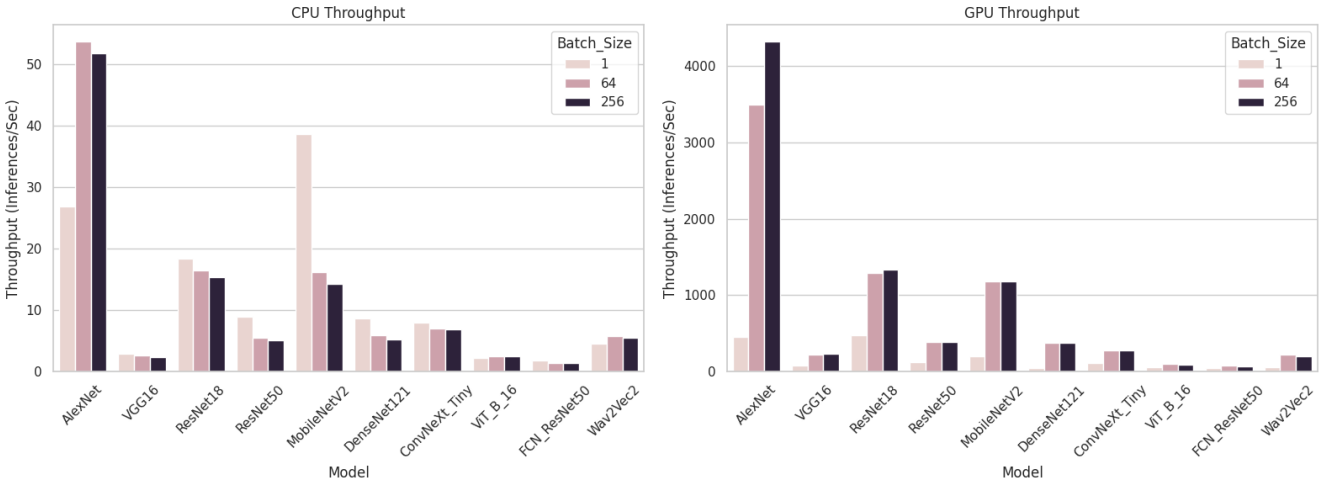

Figure 6: Throughput of various models with different batch sizes

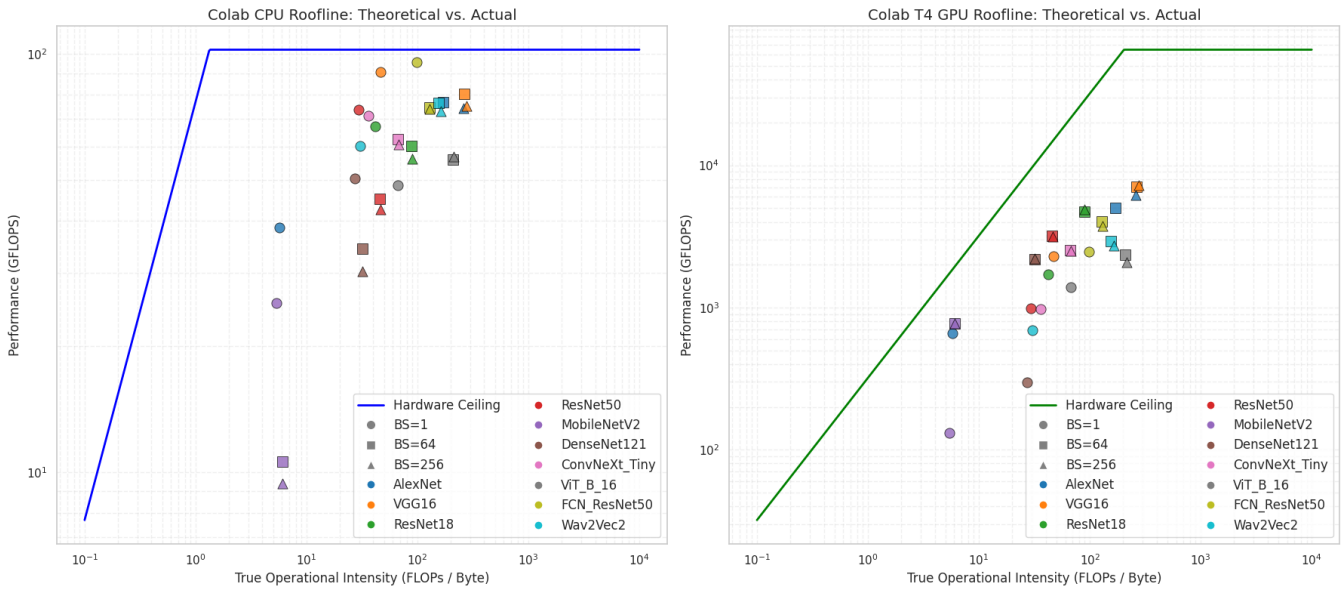

Figure 7: Roofline model showing hardware utilization of different models

4 Hardware Utilization and Peak Performance

Figure 7 shows the roofline model of all models on both CPU and GPU. To calculate the actual FLOPS, we divide the FLOPs with the actual latency for each configuration. The operational intensity is calculated by accounting for not only the model parameters but also the input and activation sizes for each batch size. For all configurations on CPU, they are compute-bound, meaning that they are limited by CPU peak FLOPS. The performance cannot reach the peak because there are other limiting factors, such as the CPU not being able to run on turbo mode, as it will increase the heat for a long period of time. "Compute bound" means that we need to upgrade the hardware that has higher computation to increase the performance.

As the batch size increases, the operational intensity tends to increase on both the CPU and the GPU. However, the performance on the GPU also increases, unlike on the CPU. This is because GPUs are designed to do parallel computations effectively. Almost all workloads are memory-bound, meaning we can still pass more data into computation to increase the performance. Aside from that, we can also reduce the model size hence increasing the operational intensity while reducing latency by computing fewer bytes.

5 Inference vs training

Table 3 shows the measured latency and FLOPs during forward and backward propagation. The latency measurement matches with the theoretical ratio of forward and backward passes (1:2). This is because when doing the backward pass, we need to calculate both the gradient of the weights and the inputs. However, the measured FLOPs do not match because the Torch profile does not count the backward FLOPs properly, as explained in [3].

Figure 8 shows the breakdown of latency, memory footprint, and FLOPs on forward and backward for AlexNet. We can see that convolution accounts for the majority (60-70%) of latency in both forward and backward propagation. This means that AlexNet heavily uses convolution in the model architecture. On the other hand, the linear layers use the most memory due to the backpropagation calculation needed to compute the gradient. The FLOPs measurement

Table 3: Model Latency and Measured FLOPs Comparison

Model	Pass Type	Latency (s)	Measured FLOPs	Ratio (F:B)
AlexNet	Forward	0.01594	91,416,125,440	**Latency**
	Backward	0.03186	15,007,219,712	1 : 2.00
				FLOPs
				1 : 0.1642
ResNet18	Forward	0.05171	232,201,388,032	**Latency**
	Backward	0.10428	131,072,000	1 : 2.02
				FLOPs
				1 : 0.0006

shows that convolution layers use the most FLOPs in the forward pass, while the linear layers use the most FLOPs in the backward pass aligned with the backpropagation calculation mentioned earlier.

Figure 9 shows the breakdown for ResNet18. ResNet18 also takes the most time on convolution layers, the same as AlexNet. This is not surprising since both are CNN models. However, we can see on the memory footprint that ResNet18 uses much less memory for linear layers. Instead, it uses the memory for convolution, activation, normalization, and pool layers because this model has residual network which require far more of these layers compared to linear layers on AlexNet. The FLOPs measurement also shows the same characteristics as AlexNet.

References

[1] Google Cloud. *Cloud TPU Architecture Documentation*. 2024. URL: https://docs.cloud.google.com/tpu/docs/tpu7x.

[2] Google Cloud. *Cloud TPU v4 Architecture*. 2023. URL: https://docs.cloud.google.com/tpu/docs/v4.

[3] Marius Hobbhahn. *How to measure FLOP/s for Neural Networks empirically?* Nov. 2021. URL: https://www.alignmentforum.org/posts/jJApGWG95495pYM7C/how-to-measure-flop-s-for-neural-networks-empirically.

[4] Intel Corporation. *Intel Xeon Processor E5-2660 v4 Product Specifications*. https://www.intel.com/content/www/us/en/products/sku/91772/intel-xeon-processor-e52660-v4-35m-cache-2-00-ghz/specifications.html. 2016.

[5] NVIDIA Corporation. *NVIDIA A100 Tensor Core GPU Datasheet*. 2020. URL: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-nvidia-us-2188504-web.pdf.

[6] NVIDIA Corporation. *NVIDIA Grace CPU Superchip Architecture In-Depth*. 2023. URL: https://developer.nvidia.com/blog/nvidia-grace-cpu-superchip-architecture-in-depth/.

[7] NVIDIA Corporation. *NVIDIA H100 Tensor Core GPU*. 2023. URL: https://www.nvidia.com/en-us/data-center/h100/.

[8] NVIDIA Corporation. *NVIDIA T4 Tensor Core GPU*. https://www.nvidia.com/en-us/data-center/tesla-t4/. 2018.

[9] Stack Overflow Contributors. *What CPU is used in Google Colab?* https://stackoverflow.com/a/68807247. 2021.

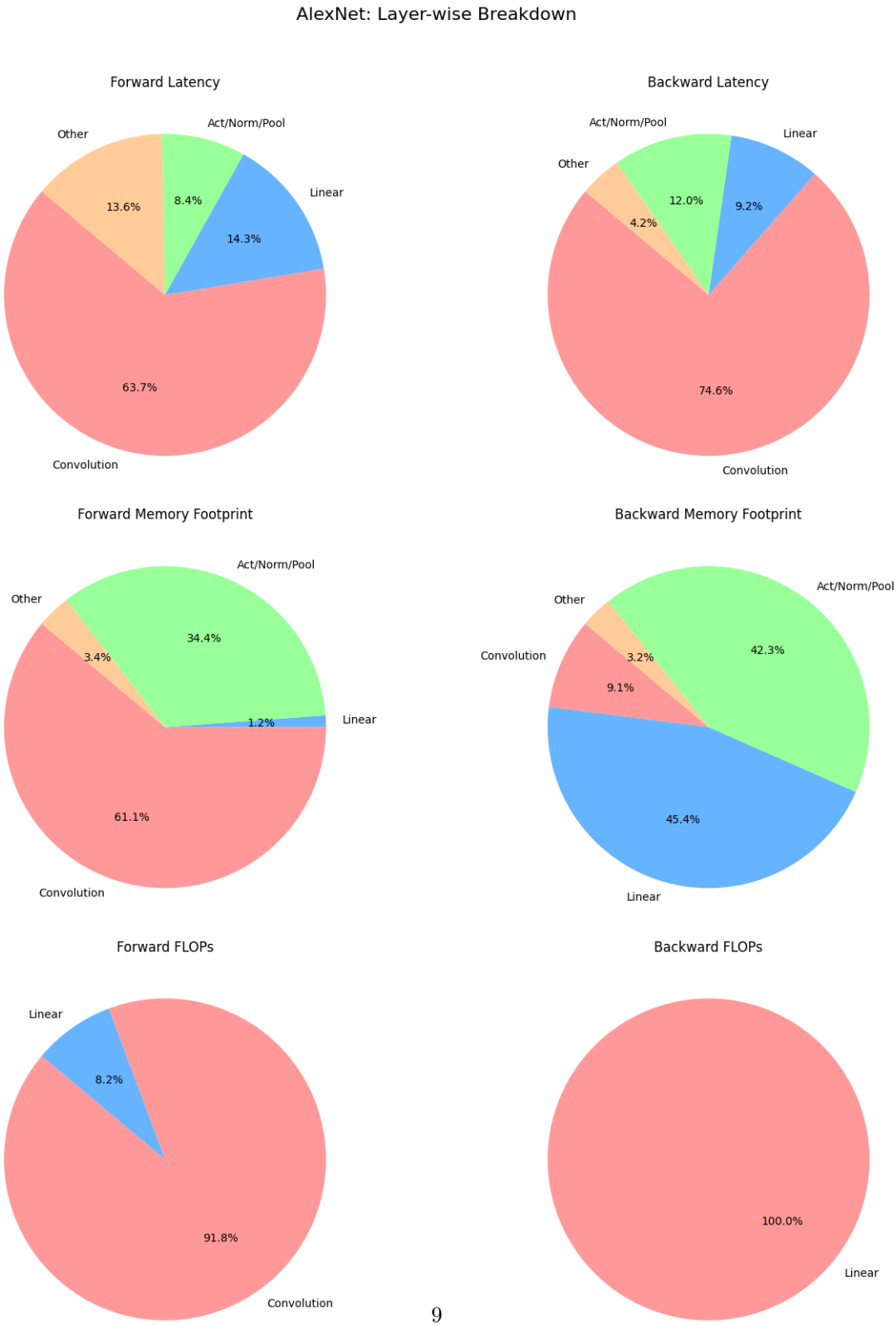

Figure 8: Breakdown of forward and backward propagation of AlexNet

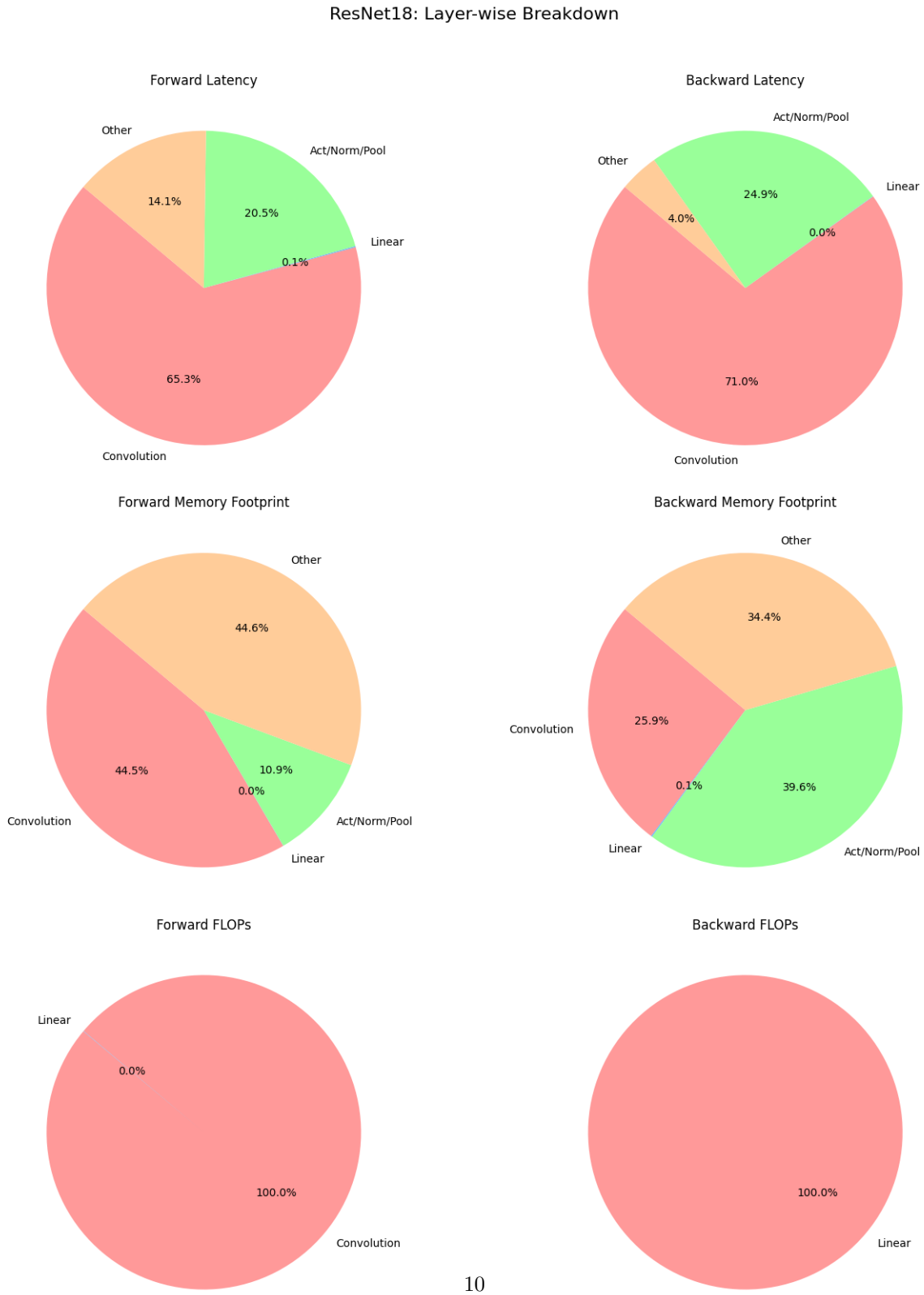

Figure 9: Breakdown of forward and backward propagation of ResNet18