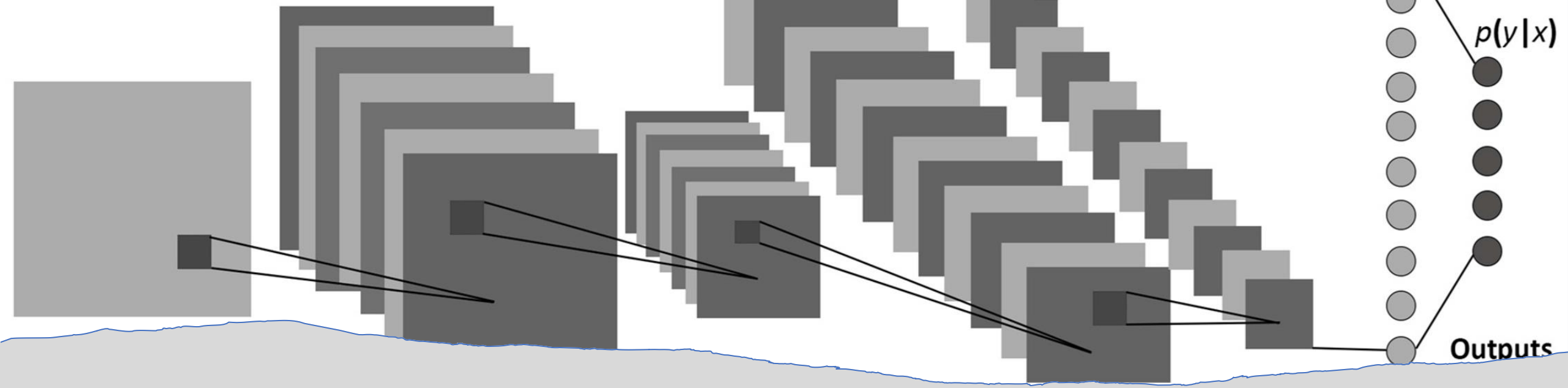


CNN Memory Profiling and Optimizations

Akash Maji	Utkarsh Sharma	Suraj Reddy	Aman Nokhwal

But Why?

CNN architecture is used in various vision tasks which require **very fast inference!**

ARVR:

Typically, 72Hz, 90Hz and 120Hz screen refresh rate.

(Roughly 13.89 ms – 8.33 ms for each frame)

VR/AR

Robotics:

For localization/action need faster prediction for better action

Graphics Rendering

Techniques such as path tracing, use denoising which have CNN (need ~30-100+ fps)

Graphics Rendering

Robotics

Problem Statement

We need to be able to do fast inference

We must consider the underlying hardware and its capability

Memory Profiling and Optimization is needed for such vision/image processing tasks

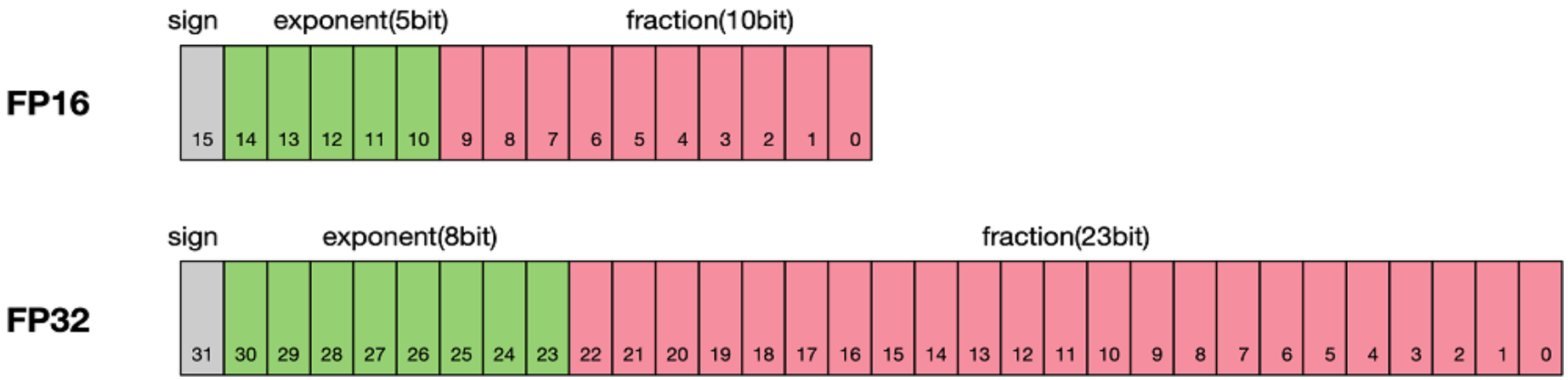
We should consider the inference time and resource utilization constraints

A careful analysis is needed before deploy critical vision/image related models

Our Methodology

- We pick a CNN model (ResNet) and do memory profiling
- We apply some heuristic to do training and then inferencing with varying criteria
- We apply various optimization techniques to improve resource usage
- We put forth various observations and comments
- Some of them are FP-16 arithmetic, Tiling, AMP + AMC, Toeplitz transformation etc.

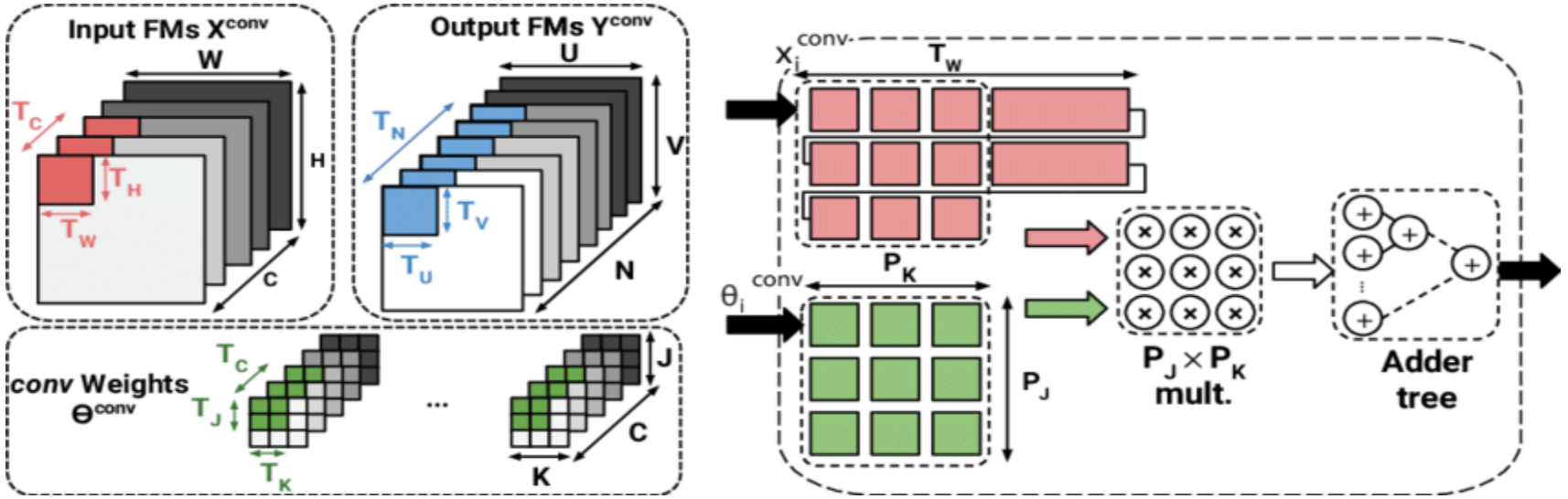
Some Optimizations: FP16



+ Faster Computation : benefits from **Tensor Core acceleration**, resulting in **faster inference** times, often **2x to 4x** speedup compared to FP32.
+ Reduced Memory Usage: FP16 reduces memory bandwidth and storage requirements, allowing larger models or batch sizes to fit into **GPU memory**

- Reduced Precision: **Lower numerical precision** can lead to **loss of accuracy** in some models, particularly those sensitive to small changes in values.
- Compatibility and Support: Not all models or operations may be optimized for FP16.

Some Optimizations: Tiling



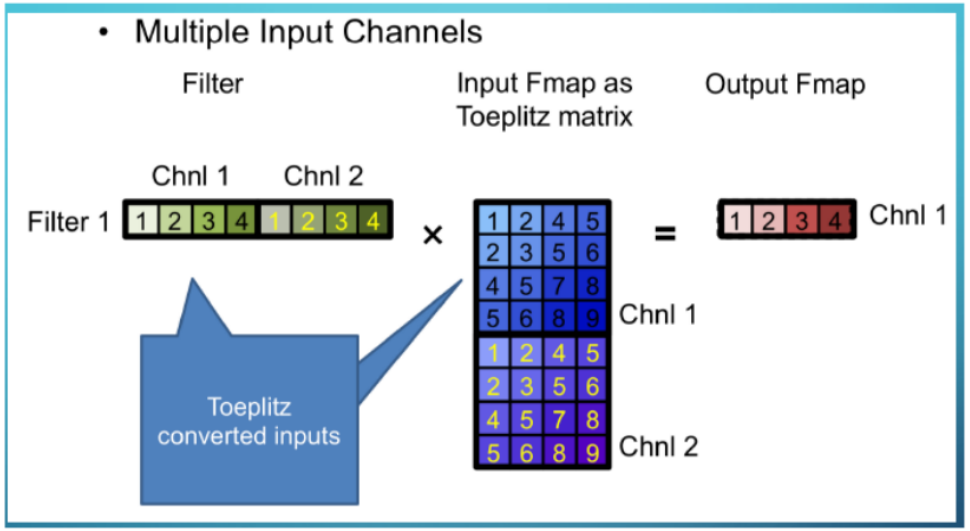
+ Improved Memory Access Efficiency: Reduces memory latency and boosts data access speed by utilizing smaller, localized tiles that fit into faster memory.

+ Optimized GPU Utilization: Enhances parallelism and load balancing, maximizing GPU usage and improving performance for large inference tasks.

- Increased Complexity: Requires careful planning of memory layouts and tile sizes, making optimization more complex.

- Limited by Memory Hierarchy: Performance gains depend on fitting tiles into fast memory, and inefficient tiling can cause memory bottlenecks and reduce performance.

Some Optimizations: Toeplitz Matrix



+ Efficient Storage: Requires less memory by storing only the first row and column, making it more space-efficient.
+ Faster Computation: Simplifies matrix operations (e.g., convolutions) and reduces computational complexity using its inherent structure.

- Limited Flexibility: Works best for problems with repetitive or structured data, limiting its use for arbitrary matrices.
- Not Ideal for All Operations: Computational benefits are mainly in specific applications, not universally applicable.

Experiment Configuration: Model and Dataset

Models used:

ResNet: Residual Network [MicroSoft Research, 2015]

Neural Network using "residual connections" or "skip connections" https://arxiv.org/pdf/1512.03385

Variants Tested:

- **ResNet-20** – 270K parameters
- **ResNet-32** – 465K parameters
- **ResNet-44** – 660K parameters
- **ResNet-56** – 855K parameters

(Implement the ResNet models, not inbuilt resnet classes from PyTorch)

- **Datasets Used:**

CIFAR-10: **10 classes, 60,000 images,** Image size: **3 × 28 × 28**

Mini ImageNet: **1,000 classes, ~38,000 images,** Image size: **3 × 224 × 224**

Experiment Configuration: System

Experiments were performed on following systems

	System-1	System-2	System-3
CPU	AMD Ryzen 7 5800H (8 cores, 16 threads)	AMD Ryzen 7 5800H (8 cores, 16 threads)	Intel Xeon (hosted environment)
GPU	NVIDIA GeForce RTX 3060 Laptop GPU (6 GB VRAM, Ampere Architecture)	NVIDIA GeForce GTX 1050 Laptop GPU (4 GB VRAM, Pascal Architecture)	NVIDIA Tesla T4 (16 GB VRAM, Turing Architecture)
RAM	16 GB	24 GB	12–16 GB (depending on session)
OS	Windows 11 24H2	Ubuntu 24.04	Ubuntu 20.04 (Cloud Environment)
Framework	PyTorch 2.6.0, CUDA 12.6	PyTorch 2.6.0, CUDA 12.7	PyTorch 2.6.0, CUDA 12.6
Python	3.11.5	3.12.7	3.11.12

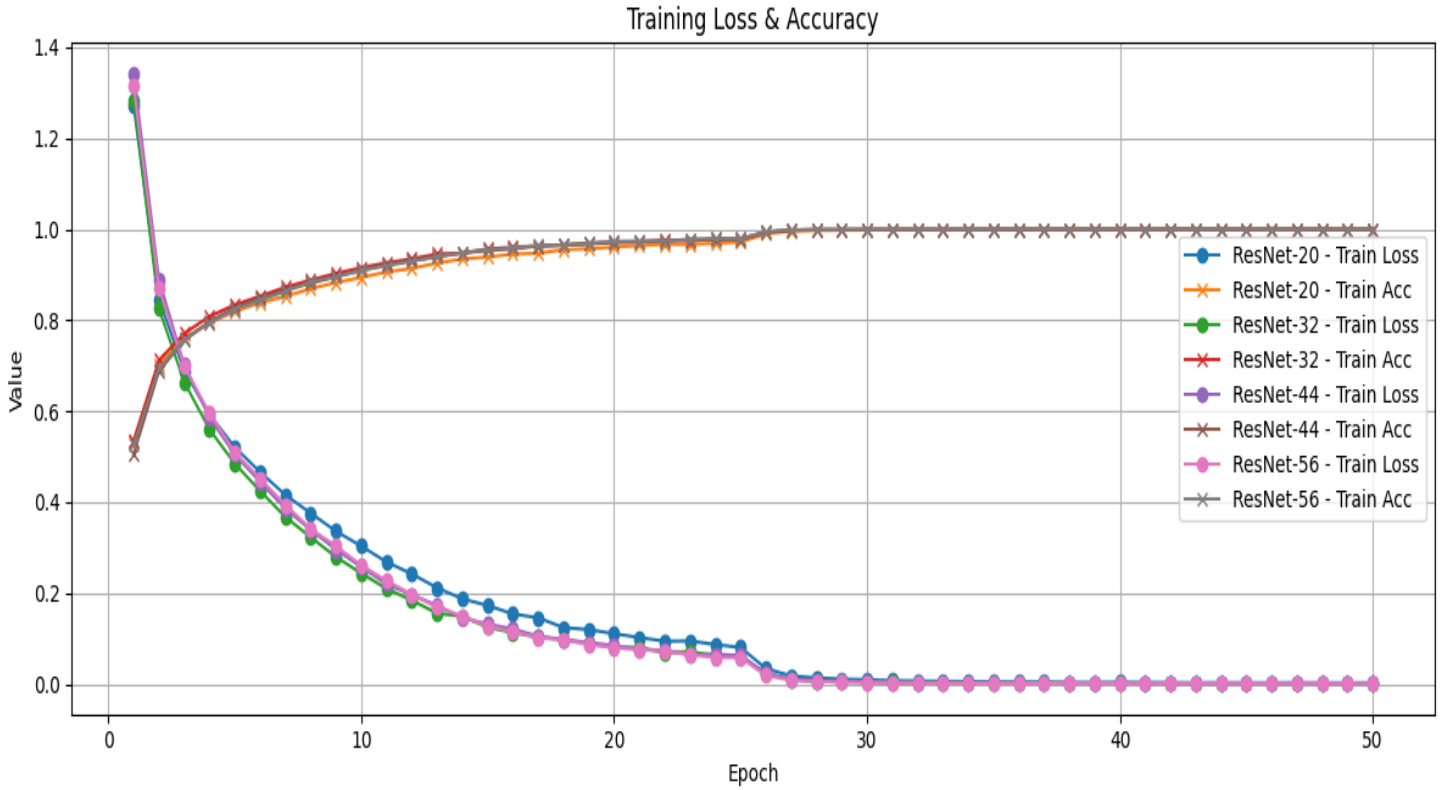
Let's Train

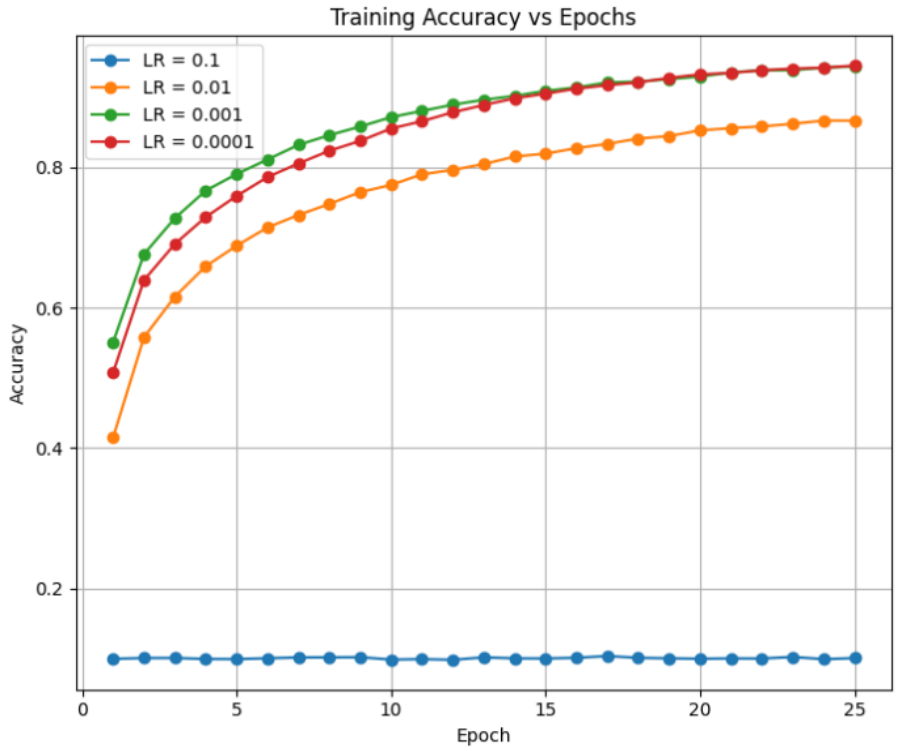
Our Heuristic

1. We observe that training loss and training accuracy saturates around 50 epochs [we fix our training for 50 epochs]
2. We observe that a learning rate of n = 0.001 gives better accuracy on test set [we fix our training for 0.001 LR]

So, we save our model parameters (weights) after training with 50 epochs

We use these saved weights to run inference (on test set)





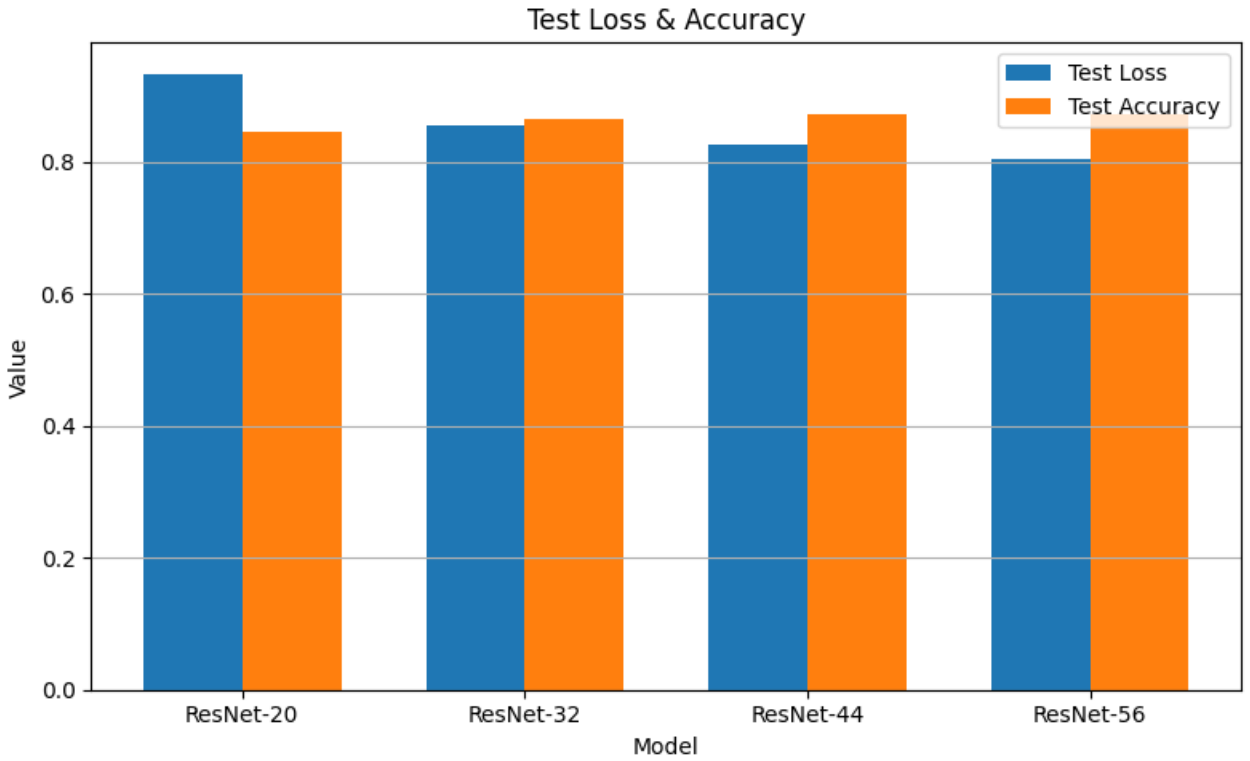
Inference Results: improving test accuracy

Observations on test inference:

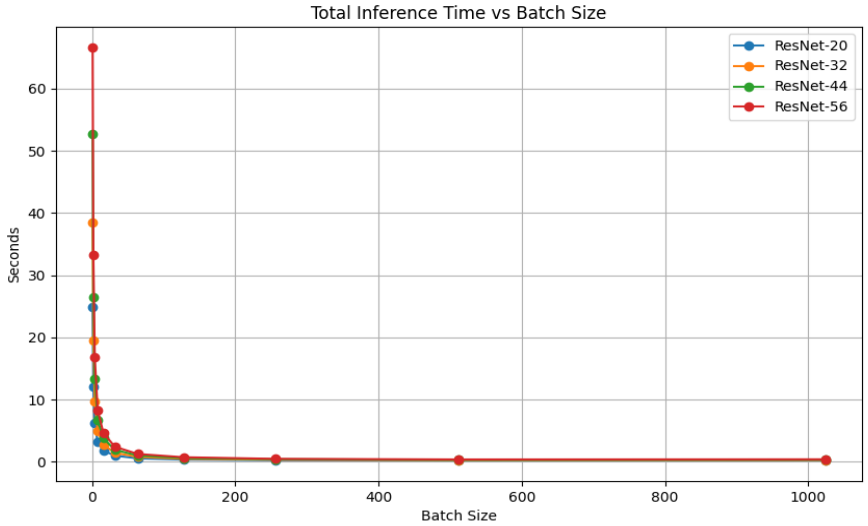
1. As we go from smaller models to bigger models (Resnet20 to ResNet56), we see decrease in test loss and increase in test accuracy.

2. The decrease in test loss and increase in test accuracy doesn't seem to happen at a higher rate

3. It indicates that training smaller models (lower training time) might be not significantly worse than bigger models in terms of inference performance, considering the time and resources it takes to train bigger models.



Inference Results: CIFAR-10 dataset



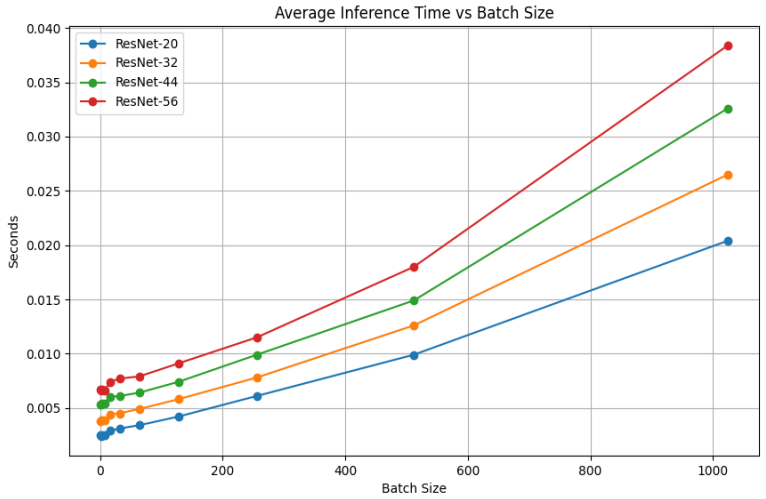
Observations:

- We see rapid decay in time as we increase the batch size for inference
- This suggests we should have larger batch size to leverage the parallelism.
- The average memory usage is higher in larger batch sizes (1024) over smaller ones,
- It shows that sometimes having larger batch-size may not be suitable for memory-constrained environments when we run multiple such models.

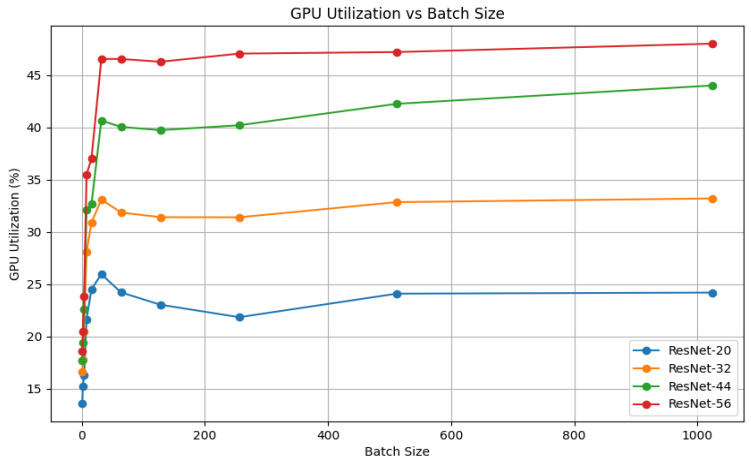
Inference Results: CIFAR-10 dataset

CIFAR-10 has 10000 images of 28x28 size for test-set

- We plot graphs for average inference time
- We see that ResNet-56 has higher avg inference time across all batch sizes
- The time it takes to do inference on batch size 1024 is higher, which is expected

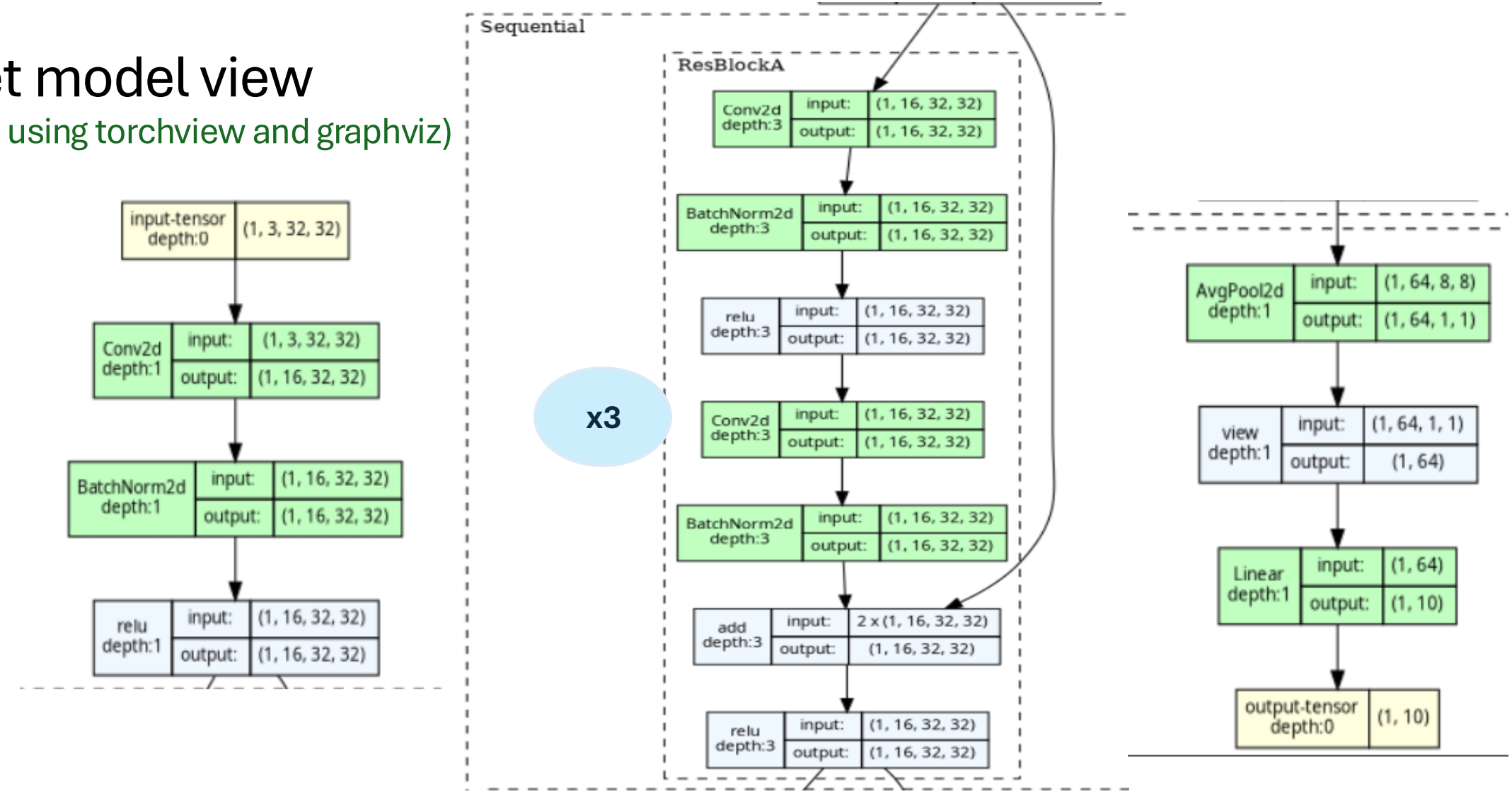


- We see that GPU utilization(percentage of time GPU is used) is highest in ResNet-56 as it will have higher operations to do.
- We see higher utilization with increasing batch size.



ResNet model view

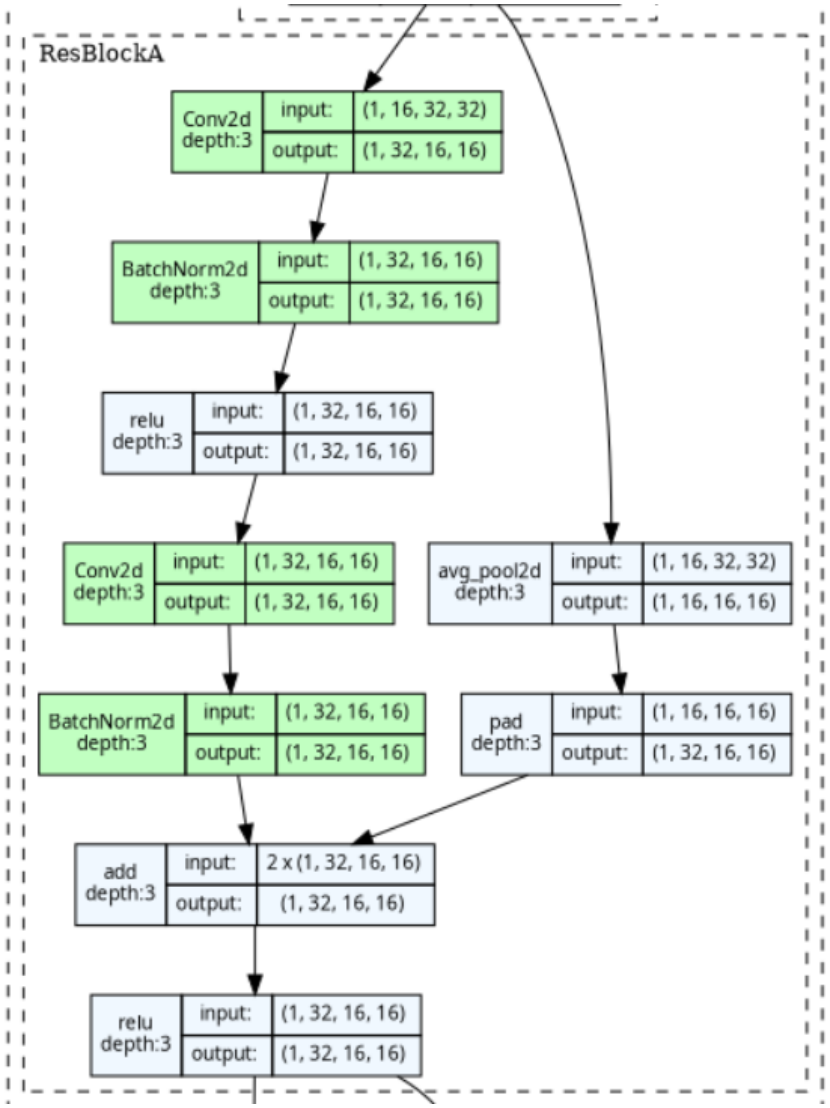
(visualized using torchview and graphviz)

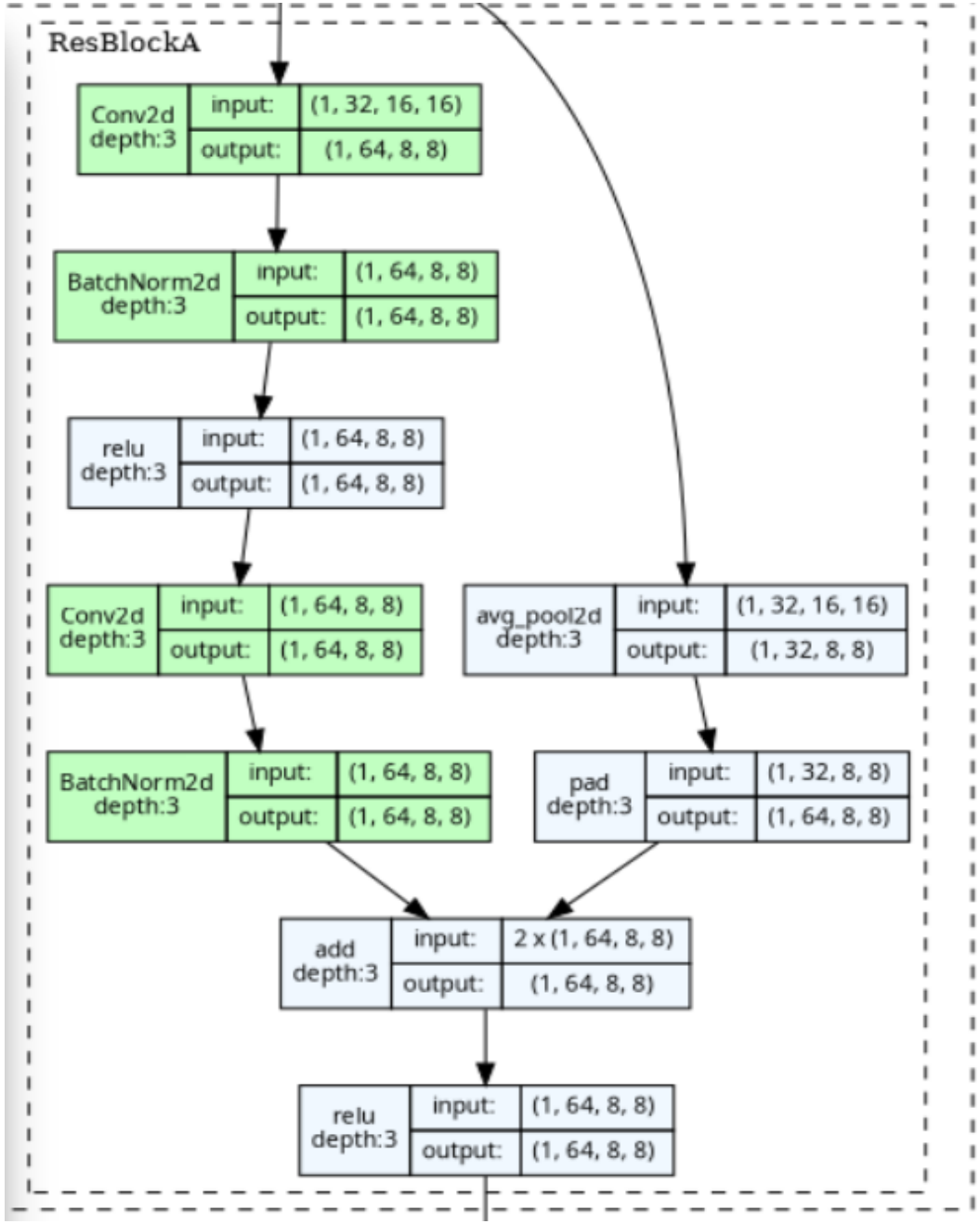



```
model_graph = draw_graph(model, input_size=(1, 3, 32, 32), expand_nested=True, roll=True)
model_graph.visual_graph.render(filename='resnet8_compact', format='png')
```

ResNet model view

ResNet8 structure [2 middle blocks]





Optimizing memory usage with FP-16

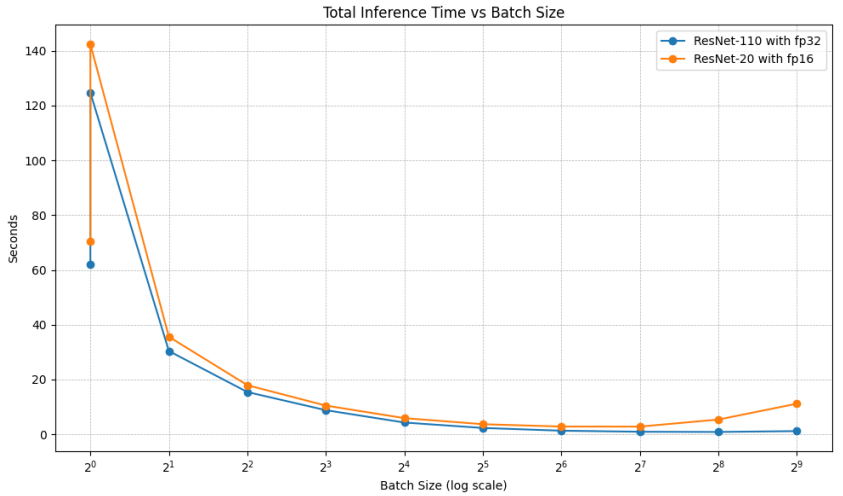
We used ResNet-110 as our reference model for this and used our pretrained weights

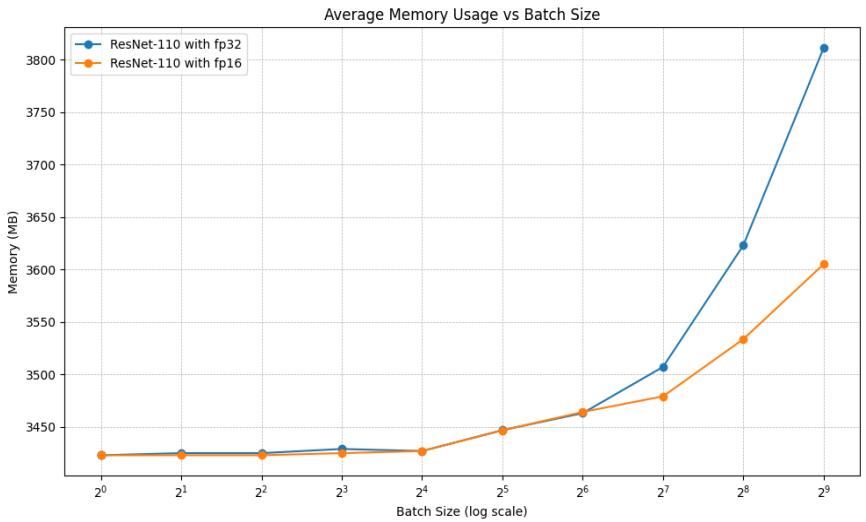
One way to optimize memory usage during inference is to use FP16 arithmetic

This is particularly useful as many modern GPUs support mixed precision arithmetic

We see slight increase in inference time per batch size. This is due to fact that for smaller image sets like 28*28 CIFAR-10, the overhead of switching to fp-16 arithmetic dominates.

However, we see improvement in average memory in higher batch sizes usage due to dynamically using fp16 arithmetic.





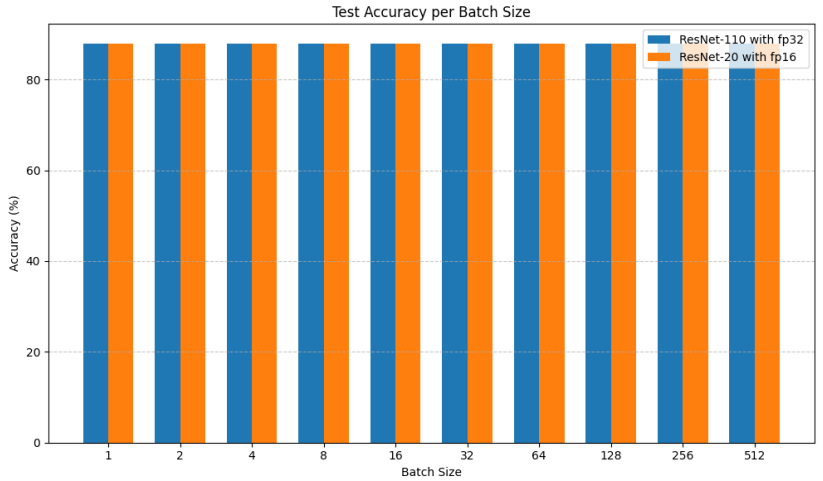
Optimizing memory usage with FP-16 inference

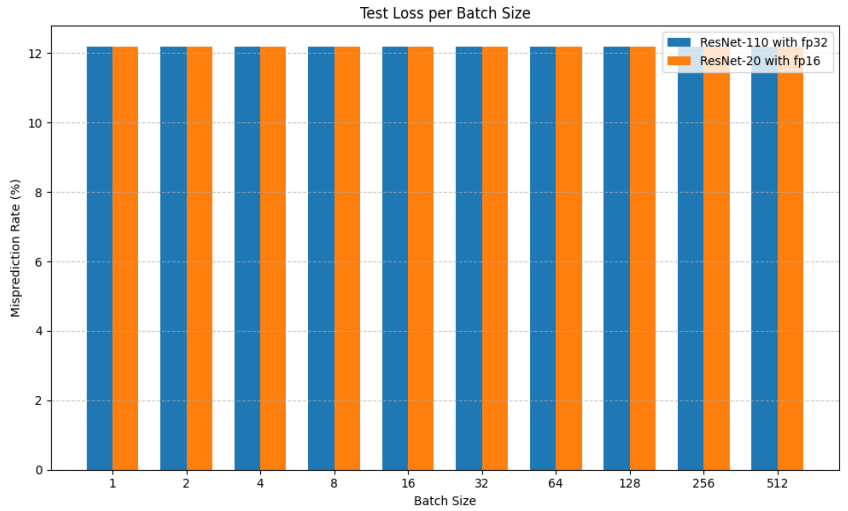
We had run our training on CIFAR-10 dataset with FP-32 weights that we had trained

We ran inferences in two ways:
FP-32 arithmetic
FP-16 arithmetic using torch.autocast()
We see same test accuracy and same test loss in both the configurations. This is good.

Observations:
Using FP-16 arithmetic for FP-32 trained weights gives almost same accuracy in inferencing for CIAFAR-10 on ResNet models
We see zero to minimal impact on test accuracy

We can use this technique to optimize GPU usage





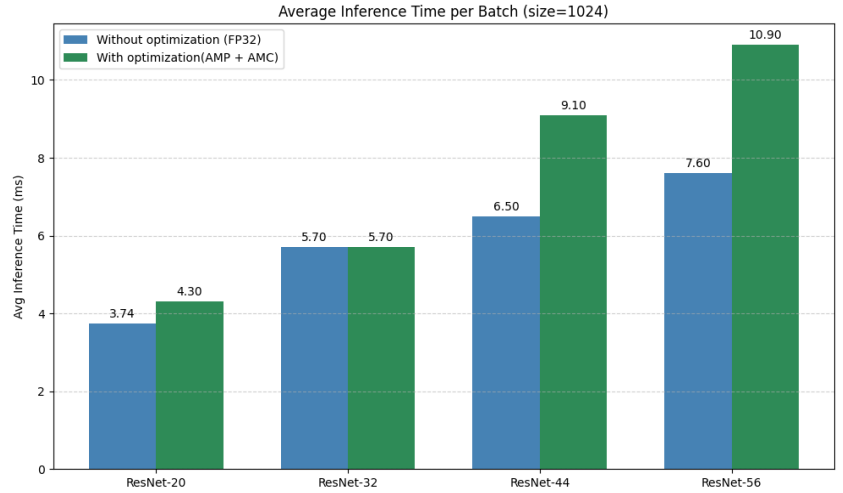
Optimizing memory usage (AMP + AMC)

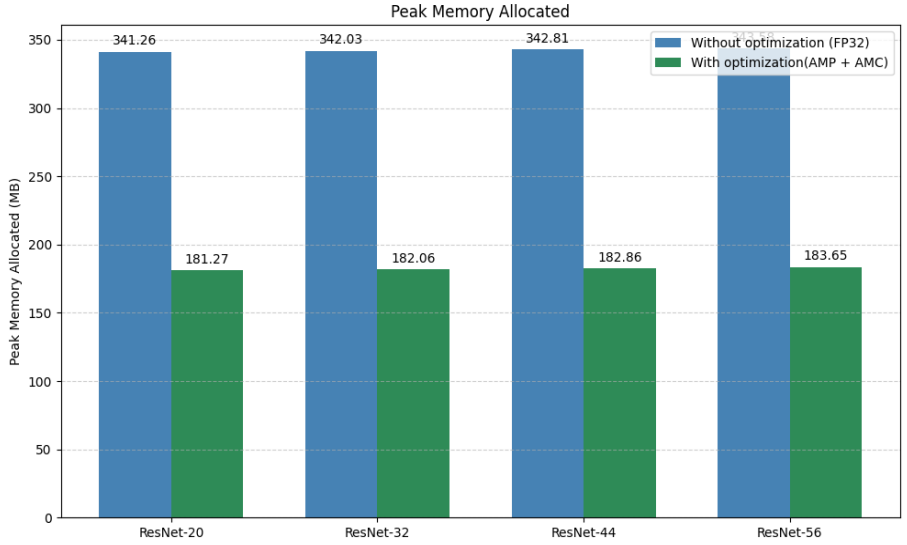
We use **AMP** = automatic mixed precision (fp16) and **AMC** = automatic memory coalescing to improve the memory usage

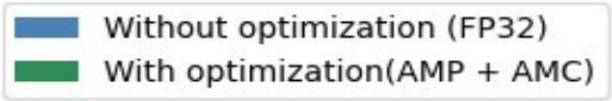
The system we used has a Tesla T4 GPU, which is based on Turing architecture (Inference Done on Cloud)

We observe slight increase in inference time due to dynamic decision of using fp16 and switching to tensor cores for fp16, instead of fp32, as earlier.
We see this overshadowing effect across all batch sizes

However, we see drastic improvement in peak memory allocated which saves GPU memory for other tasks.







Optimizing memory usage with tiling

We use ImageNet Dataset containing ~38K images (~19K train + ~19K test) for this experiment

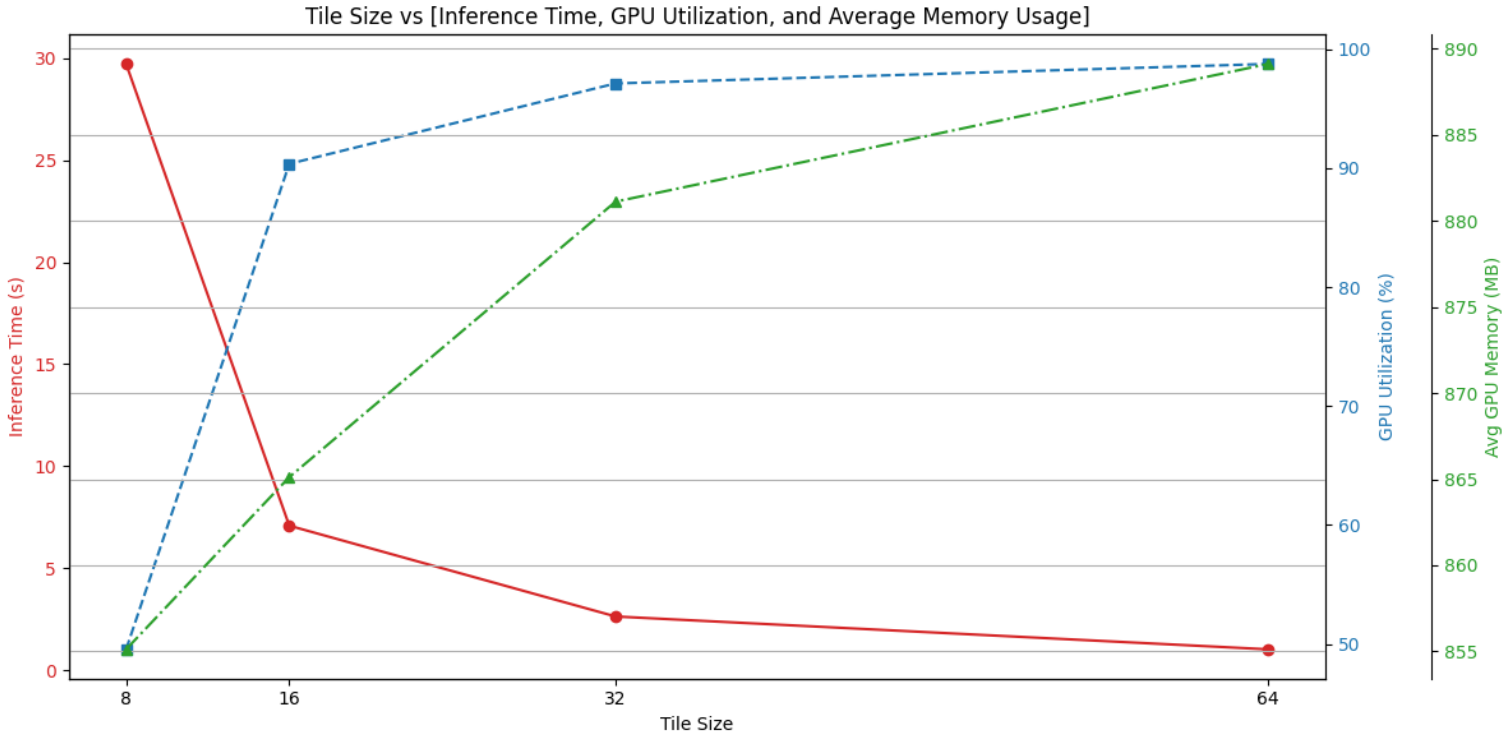
We train the ResNet-20 model using train dataset for ~10 epochs

The inference is then run on the test data set with batch size = 8

Observations:
Average GPU memory consumption is lower for smaller tile size, which is expected

Also, GPU utilization is better in higher tile sizes.

We have very high scope of analysis here



Summary

Genesis

- We saw that we trained multiple ResNet models using our heuristics
- We saw how we can improve test accuracy with deeper models
- We saw inference performance on CIFAR-10 dataset and observations
- We looked at our model's graphical implementation structure

Synthesis

- We began optimizing with FP-16 arithmetic for inference
- We saw how tiling can be used to reduce average memory usage
- We saw how we can combine AMP + AMC to improve peak memory usage
- We saw how GPU utilization varies with tiling sizes

CNN Memory Profiling and Optimizations

Akash Maji

Utkarsh Sharma

Suraj Reddy

Amandeep Nokhwal

https://github.com/akashmaji946/CNN-Memory-Profiling-And-Optimization-v2