Deep Learning

Lab Assignment -6

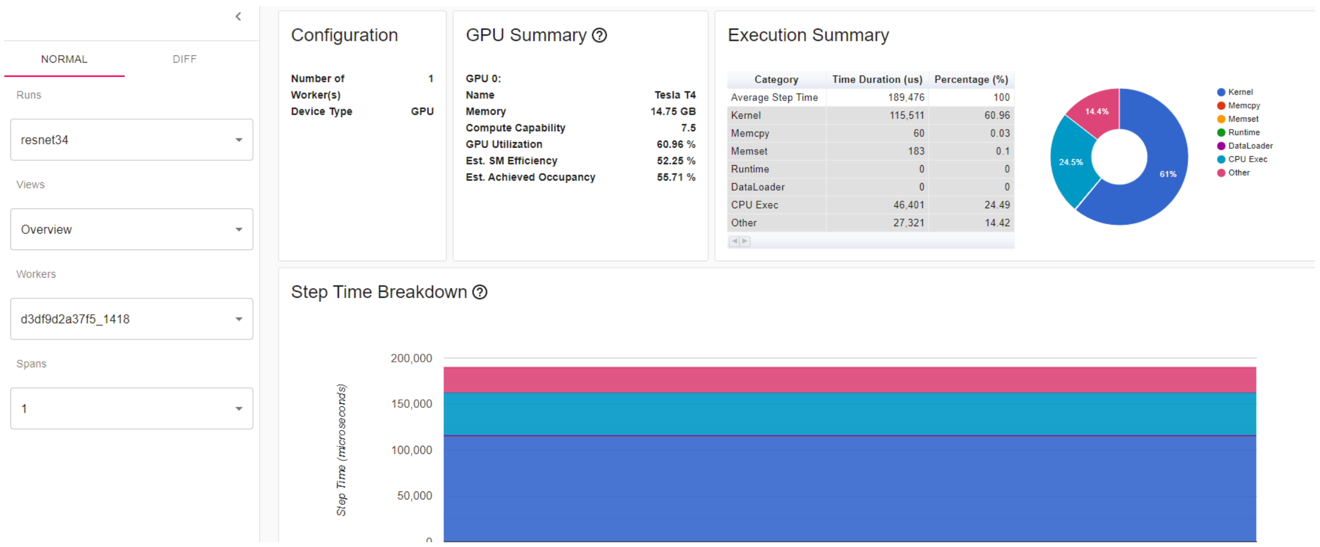
Priya Sahu (B20AI031)

- Import all the necessary libraries.
- Then loading the CIFAR100 dataset. Transforming it to tensor and normalizing it. Making the train and test loader with batch_size=16.
- I have trained 4 models:
  1) Resnet34 2) DenseNet121 3) EfficientNet_B0 4) ConvNeXt-T

In each case, I loaded the pretrained model and then changed the last layer of the models to output 100 classes. Then trained the model on the train data for 10 epochs. Now in order to do profiling during training I put the train function inside the profiler function. So it recorded the different usages for 10 epochs.

For each model, I used Adam optimiser and learning rate=0.001.

Visualization of Profiled metrics in tensorBoard :



Configuration

Number of Worker(s)	1
Device Type	GPU

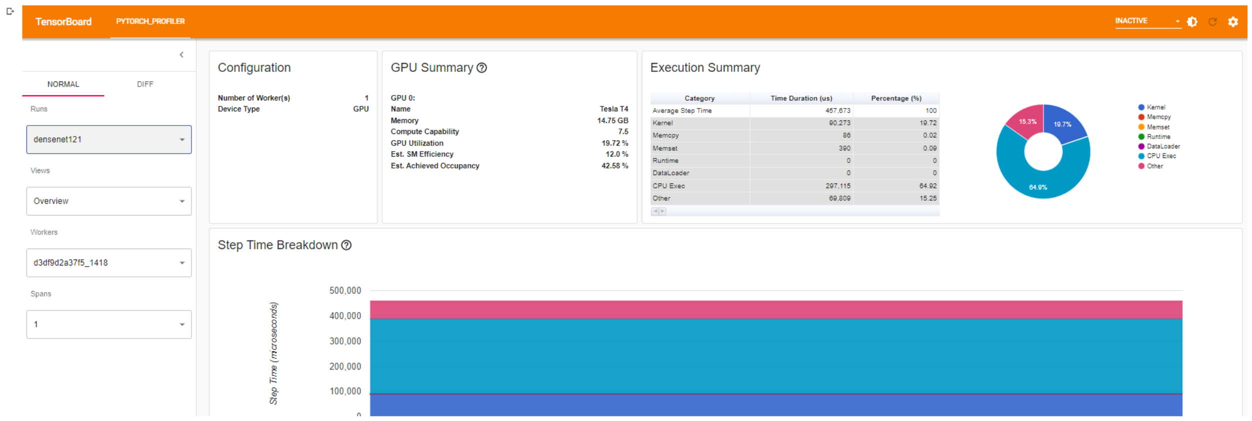
GPU Summary

GPU 0:	
Name	Tesla T4
Memory	14.75 GB
Compute Capability	7.5
GPU Utilization	60.96 %
Est. SM Efficiency	52.25 %
Est. Achieved Occupancy	55.71 %

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	189,476	100
Kernel	115,511	60.96
Memcpy	60	0.03
Memset	183	0.1
Runtime	0	0
DataLoader	0	0
CPU Exec	46,401	24.49
Other	27,321	14.42

Step Time Breakdown



Runs: densenet121
Views: Overview
Workers: d3df9d2a37f5_1418
Spans: 1

Configuration

Number of Worker(s)	1
Device Type	GPU

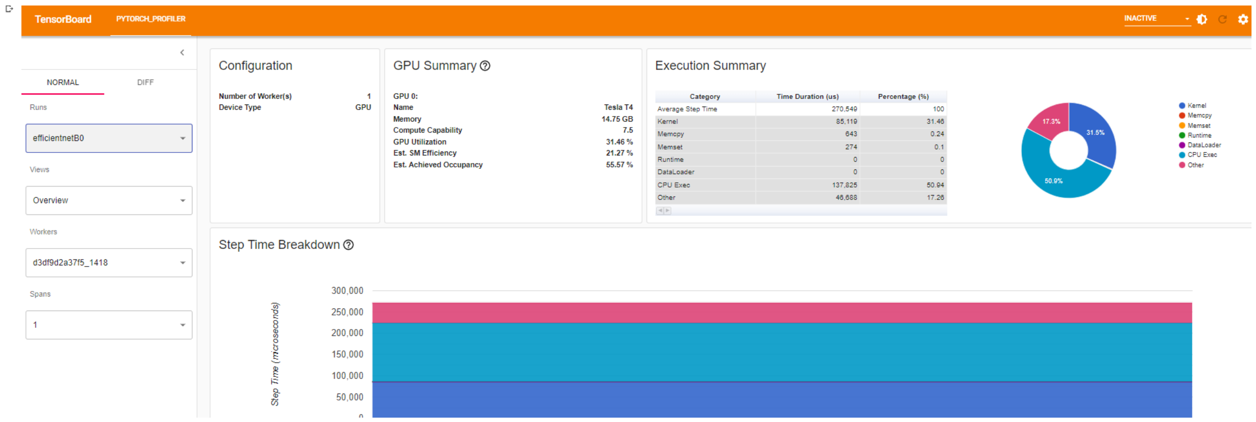
GPU Summary

GPU 0:	
Name	Tesla T4
Memory	14.75 GB
Compute Capability	7.5
GPU Utilization	19.72 %
Est. SM Efficiency	12.0 %
Est. Achieved Occupancy	42.58 %

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	457,673	100
Kernel	90,273	19.72
Memcpy	86	0.02
Memset	390	0.09
Runtime	0	0
DataLoader	0	0
CPU Exec	297,115	64.92
Other	69,809	15.25

Step Time Breakdown



Runs: efficientnetB0
Views: Overview
Workers: d3df9d2a37f5_1418
Spans: 1

Configuration

Number of Worker(s)	1
Device Type	GPU

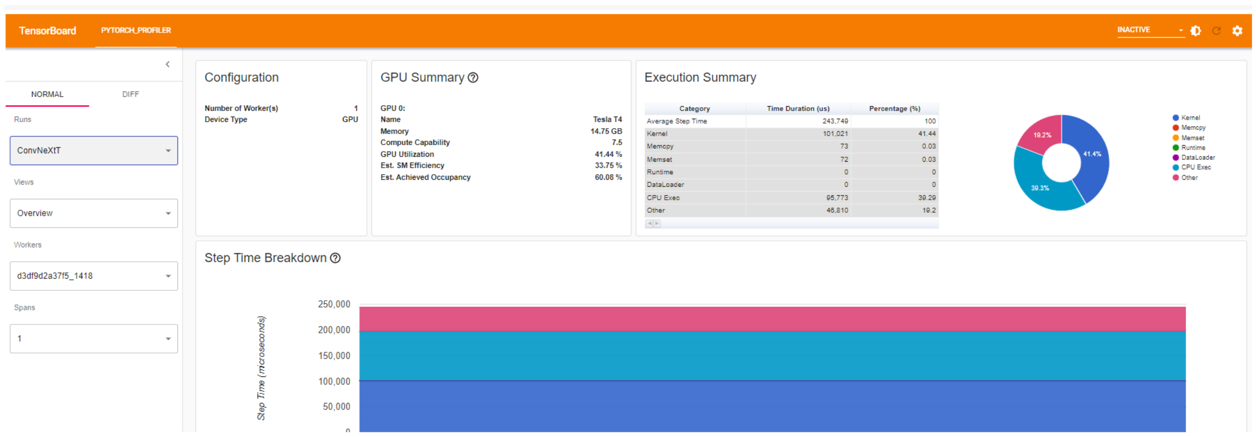
GPU Summary

GPU 0:	
Name	Tesla T4
Memory	14.75 GB
Compute Capability	7.5
GPU Utilization	31.46 %
Est. SM Efficiency	21.27 %
Est. Achieved Occupancy	55.57 %

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	270,549	100
Kernel	85,119	31.46
Memcpy	643	0.24
Memset	274	0.1
Runtime	0	0
DataLoader	0	0
CPU Exec	137,825	50.94
Other	46,688	17.26

Step Time Breakdown



Runs: ConvNeXtT
Views: Overview
Workers: d3df9d2a37f5_1418
Spans: 1

Configuration

Number of Worker(s)	1
Device Type	GPU

GPU Summary

GPU 0:	
Name	Tesla T4
Memory	14.75 GB
Compute Capability	7.5
GPU Utilization	41.44 %
Est. SM Efficiency	33.75 %
Est. Achieved Occupancy	60.08 %

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	243,749	100
Kernel	101,021	41.44
Memcpy	73	0.03
Memset	72	0.03
Runtime	0	0
DataLoader	0	0
CPU Exec	95,773	39.29
Other	46,810	19.2

Step Time Breakdown

For each model we can see different kernel utilization.

ResNet-34 with its deeper layers and residual connections may require more computations per kernel than DenseNet-121, leading to higher kernel utilization.

Similarly, we can say for Efficientnet_B0 and ConvvNeXt-T. Additionally, the specific hyperparameters and training settings used for each model can also influence their kernel utilization and overall performance. The exact kernel utilization for each model will depend on various factors, such as the batch size, image resolution, and optimization settings used during training.

Torchscript inferencing technique : TorchScript inference can be used to accelerate the execution of your PyTorch models and reduce the latency and memory footprint of your applications.

I have performed inference on all four models. And also performed further optimisation of torchscript models.
ONNX ; ONNX Quantized technique : ONNX enables developers to train and deploy models across different platforms and devices by providing a standard way to represent models that can be imported and exported from a variety of deep learning frameworks.

ONNX Quantization is a technique that allows you to convert a floating-point model into a fixed-point representation, which can be more efficient and faster to execute on hardware devices that support fixed-point arithmetic.

Firstly, we did the ONNX export and got a model. Then we optimized that ONNX model and optimized the model. Then we further quantized that model and got the quantized model.

Now comparing the model size and average execution time of all the models before and after the two inference techniques.

Note: all models are in GPU runtime as we trained the models in a colab using CUDA.

Average execution time of each model is calculated on test data. Basically how much time it takes to process one single image.

		Average execution time (in ms)	Size (MB)
Resnet34	Pytorch (simple)	0.8199	81.51
	Torchscript	0.75391	81.56
	Torchscript_optimised	0.73502	81.38
	onnx	3.29990	81.37
	onnx_optimized	3.37029	81.37
	onn_opt_quant	9.41698	20.43
Densenet121	Pytorch (simple)	2.14729	27.48
	Torchscript	1.750165	27.66
	Torchscript_optimised	4.849095	27.40
	onnx	2.13725	27.29
	onnx_optimized	2.15341	27.29
	onn_opt_quant	6.58827	7.56
EfficientNet_B0	Pytorch (simple)	1.33890	16.05
	Torchscript	1.470169	16.25
	Torchscript_optimised	1.080628	15.74
	onnx	1.05452	15.76
	onnx_optimized	1.06781	15.76
	onn_opt_quant	8.43563	4.27

ConvNeXt-T	Pytorch (simple)	0.97990	106.47
	Torchscript	0.82445	106.53
	Torchscript_optimised	0.95628	106.47
	onnx	4.31173	106.54
	onnx_optimized	4.20252	106.52
	onn_opt_quant	13.53869	27.01

Observations :

- Torchscript and its optimized model, and ONNX and its optimized model did not reduce the size of the model. You can see that size remained nearly unchanged.
- But when we did ONNX quantization, the size of the model was significantly reduced. Nearly the size reduced 4x times for each model.
- Average execution time increased after ONNX inferencing. ONNX and and ONN_optimised took nearly the same time but Quantised models take much more execution time.
- So, ONNX is more useful when needed to reduce the size of the model but execution time can increase.
- Ideally, execution time should decrease after torchscript and its optimisation but when we observe the results, it is not happening for each model.
- For resnet34, the ideal case happened.
- For densenet121, the torscript version took less time than pytorch but optimized torscript took more time. It is possible that the model has been over-optimized, which can lead to performance degradation.
- For efficientNet-B0, the optimized torchscript model took less time than the other two There can be multiple factors for this. .

- For ConvNeXt-T , the torscript version took less time than pytorch but optimized torscript took more time than torchscript model. It is possible that the model has been over-optimized, which can lead to performance degradation.

As a whole, we can see that ONNX_Quantization can be used for size reduction and torchscript can be used for execution time reduction in general. But execution time will be reduced, it is not necessary.