DLOps Assignment 3:

TRANSFORMERS AND OPTIMIZATION

Ritam Sharma
20BB030

Q1

Question Overview

- ❖ The main aim is to perform image classification on selected classes[1,3,5,7,9] of CIFAR-10 dataset using the vision transformer model with the following variations:

  1. cosine positional embedding with six encoders and decoder layers with eight
- ❖ heads. Use relu activation in the intermediate layers.

- ❖ 2. learnable positional encoding with four encoder and decoder layers with six heads.
- ❖ Use relu activation in the intermediate layers.

- ❖ 3. changing the activation function in the intermediate layer from relu to
- ❖ tanh and comparing the performance

Metrics

Accuracy

Accuracy is a commonly used metric for evaluating the performance of image classification models, including those based on vision transformers. Vision transformers are a type of deep learning model that have shown promising results in various computer vision tasks, including image classification.

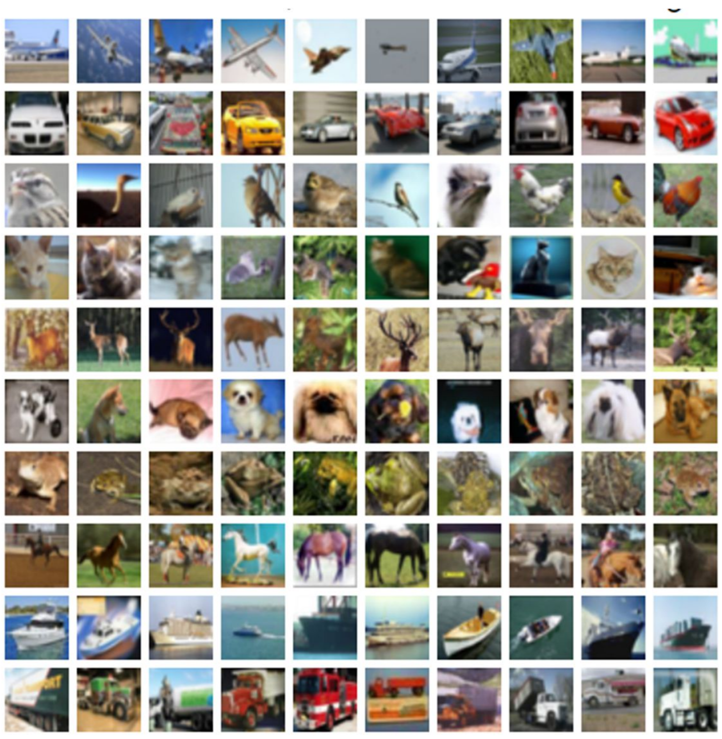
In image classification, accuracy measures the percentage of correctly classified images out of the total number of images in the dataset. To calculate accuracy, the model is trained on a set of labeled images and then tested on a separate set of images with known labels. The accuracy of the model is then calculated by dividing the number of correctly classified images by the total number of images in the test set.

Analysis

Data Exploration

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class



CFIAR10 Dataset

Methodology

Data Pre-processing:

- First, out of the 10 classes, only those images and label belonging to the required classes are filtered using a def filter() function
- Then these labels are mapped to the value 0 ,1,2,3 and 4 so that the labels don't go out of bound
- Normalizing of data is done to have a more stable convergence

Model

- Input Preparation: The input to the model is a batch of images, where each image has dimensions (C, H, W), where C is the number of channels, H is the height of the image, and W is the width of the image. The first step in the vision transformer is to divide each image into smaller patches of size (P, P), where P is decided by us

- Linear Mapper: The patches obtained from the previous step are then passed through a linear layer to map each patch to a higher-dimensional space. The output of this layer has dimensions (N, PP, D), where N is the batch size, PP is the number of patches in each image, and D is the dimension of the output feature space.

- Classification Token: In order to enable the model to classify the input image, a learnable classification token is added to the sequence of patches. This token is represented by a learnable vector of dimension D and is concatenated at the beginning of the sequence of patches.

- Positional Embedding: Since the patches do not contain any spatial information, the model needs to be provided with positional information to understand the relative location of different patches. This is done by adding a fixed-length positional embedding vector to each patch.

- Transformer Encoder Blocks: The patches with positional embeddings and the classification token are then passed through a series of transformer encoder blocks, each consisting of multiple self-attention and feedforward layers. These layers help the model learn the relationships between different patches and use this information to make predictions about the input image.

- Classification MLP: Finally, the output of the last transformer encoder block is passed through a multilayer perceptron (MLP) to obtain a classification for the input image. The MLP can have one or more fully connected layers, followed by a softmax activation function, which maps the output to a probability distribution over the classes.

- The MyViT class defined implements the above-mentioned steps to create a basic vision transformer model. The class takes input parameters such as the shape of the input image, the number of patch, the number of transformer encoder blocks, the number of attention heads, and the output dimension for classification. The class also defines the necessary layers such as the linear mapper, classification token, positional embedding, transformer encoder blocks, and MLP for classification. The forward method of the class executes the forward pass of the input through the network and returns the output of the MLP.

Loss Function used : Cross Entropy

The basic idea behind cross-entropy is to measure the difference between the predicted probability distribution and the true probability distribution of the labels. The loss function aims to minimize this difference, which is often referred to as the "cross-entropy loss".
The cross-entropy loss is defined as follows:

$$L = -1/N * \Sigma(y * log(y\_hat) + (1-y) * log(1-y\_hat))$$

where:

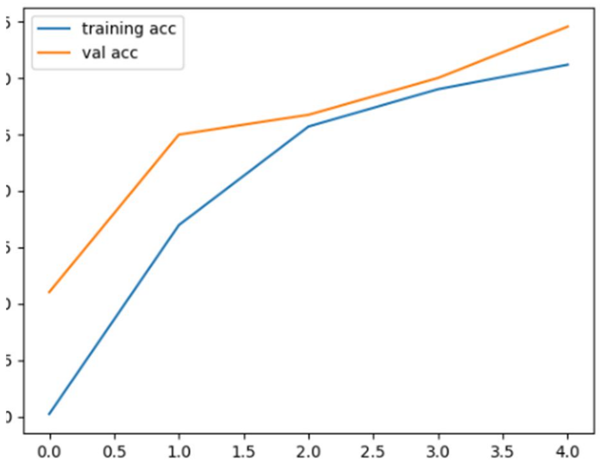
- L is the cross-entropy loss
- N is the number of samples in the dataset
- y is the true label (0 or 1)
- y_hat is the predicted probability of the label being 1

The cross-entropy loss penalizes the model more heavily when it makes confident incorrect predictions. For example, if the true label is 0 and the model predicts a probability of 0.9 for label 1, the loss will be larger than if the model predicts a probability of 0.6 for label 1.

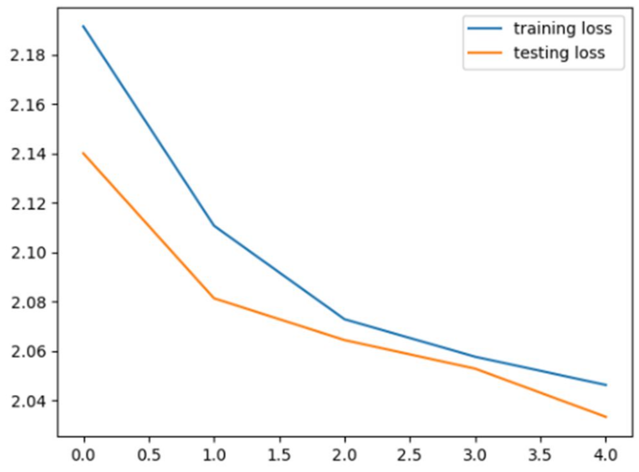
Results

Cosine positional embeddings with 8 heads and ReLu in intermediate layers

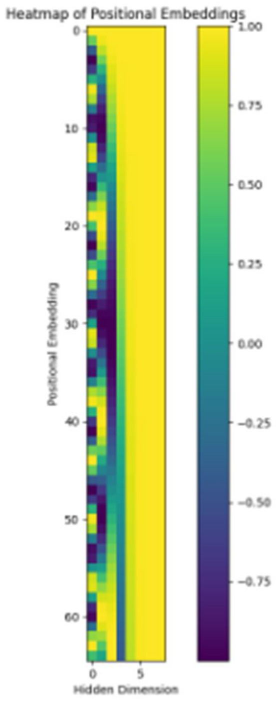
Accuracy curve



Loss curve



Heatmap (positional embedding)



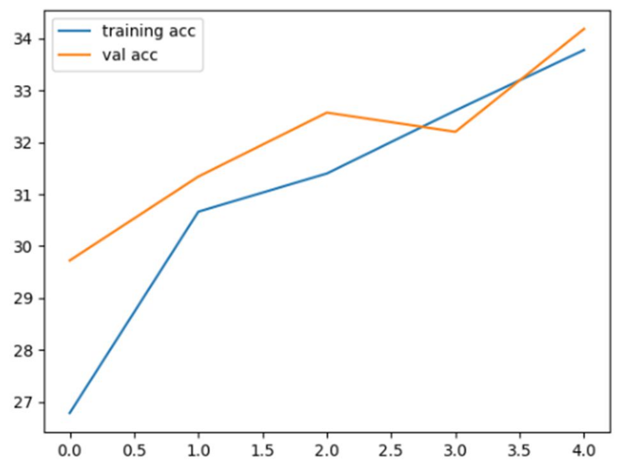
Result Table

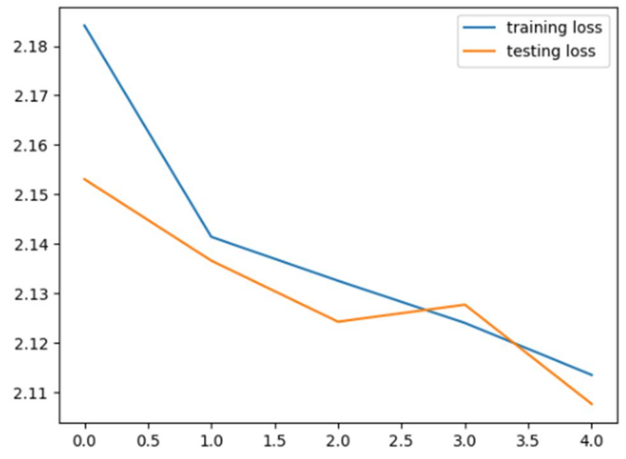
Metric	**Value**
Test loss	2.03
Test accuracy	42.29%

Learnable positional encoding with four encoder and decoder layers with six heads. ReLu in intermediate layers

Accuracy curve

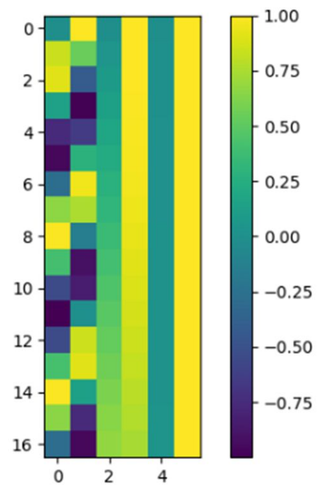
Loss curve





Heatmap (positional embedding)

Result

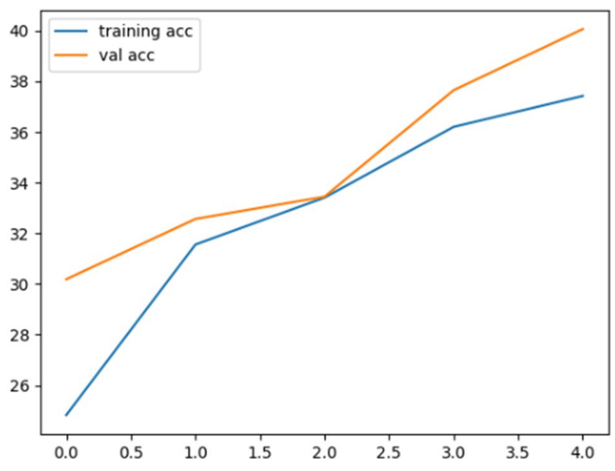


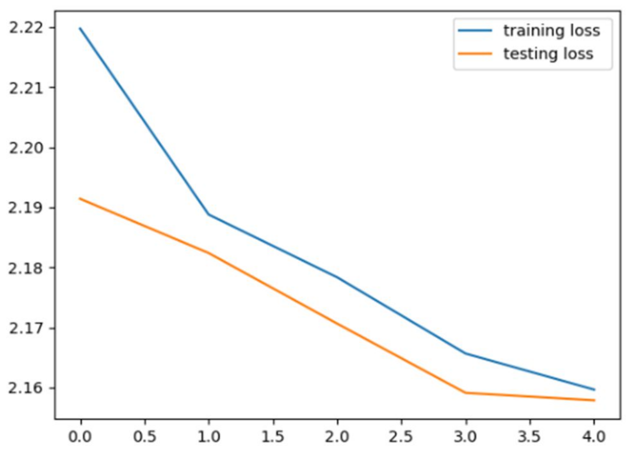
Metric	Value
Test loss	2.11
Test accuracy	34.19%

Cosine positional embeddings with 8 heads and Tanh in intermediate layers

Accuracy curve

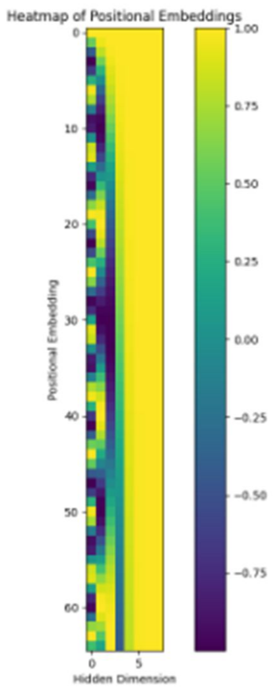
Loss curve





Heatmap (positional embedding)

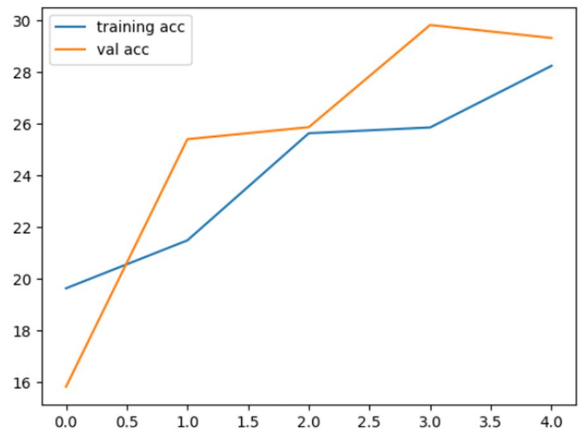
Result Table



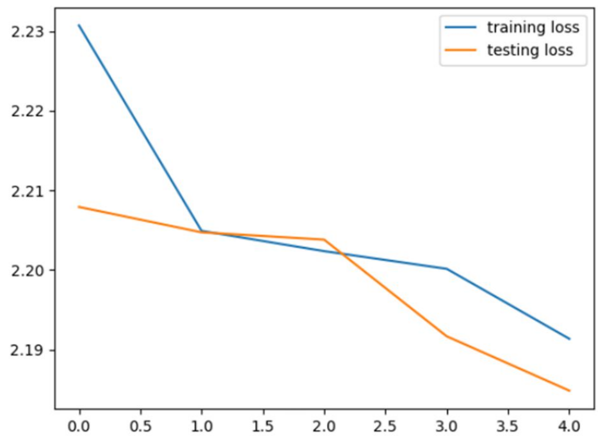
Metric	**Value**
Test loss	2.16
Test accuracy	40.07

Learnable positional encoding with four encoder and decoder layers with six heads. Tanh in intermediate layers

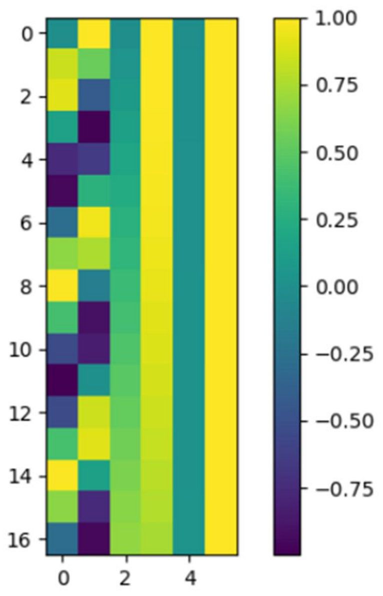
Accuracy curve



Loss curve



Heatmap (positional embedding)



Result

Metric	**Value**
Test loss	2.18
Test accuracy	29.33%

Comparison between Tanh and ReLU

With ReLu	With Tanh	Description

Loss: 2.03	Loss: 2.11	Cosine embeddings
Accuracy: 42.29%	Accuracy: 34.19%	
Loss: 2.11	Loss: 2.11	Learnable embeddings
Accuracy: 34.19%	Accuracy: 34.9%	

Conclusion

Overall, ReLu performed much better than Tanh. This is because one of the key advantages of ReLU over other activation functions such as Tanh is that it helps to alleviate the vanishing gradient problem.

The vanishing gradient problem is a common issue that can arise when training deep neural networks. It occurs when the gradients of the loss function with respect to the weights in the early layers of the network become very small as they propagate backward through the layers. This can lead to the early layers of the network learning very slowly or not at all, which can impact the overall performance of the network.

Tanh activation function, on the other hand, suffers from the saturation problem, which means that the function's output saturates to 1 or -1 for large inputs, causing the gradient to be close to zero. This means that the gradients in the earlier layers of the network may also become very small, making it difficult for the network to learn effectively.

In terms of a vision transformer, ReLU may perform better than Tanh because transformers typically use multi-layer perceptron (MLP) networks as part of their architecture. The MLP networks in transformers are similar to fully connected layers in CNNs, and ReLU has been shown to be effective in improving the performance of fully connected layers.

[9] Refences used :-

https://towardsdatascience.com/Loss-in-language-models-87a196019a94#:~:text=Loss%20can%20also%20be%20defined,based%20on%20the%20cross%2Dentropy%3F

Question 2

Question Overview

The main aim is to train the following models for profiling them using during the training step

- ❖ Conv -> Conv -> Maxpool (2,2) -> Conv -> Maxpool(2,2) -> Conv -> Maxpool(2,2)
- ❖ VGG16

After profiling the model, we need to figure out the minimum change in the architecture that would lead to a gain in performance and a decrease training time on the CIFAR10 dataset as compared to one achieved before.

Data Exploration, Augmentation, and Normalization

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class

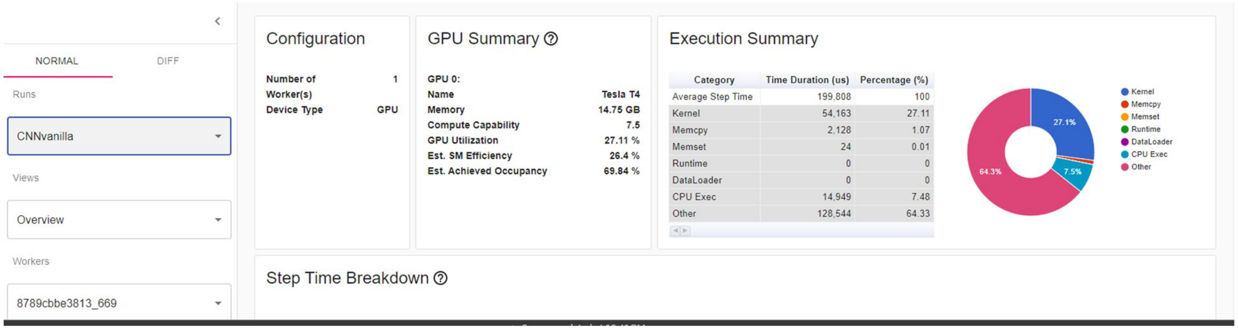
Data processing

The following is done data augmentation and Normalization:

- ❖ transforms.RandomCrop(32, padding=4),
- ❖ transforms.RandomHorizontalFlip(),
- ❖ transforms.ToTensor(),
- ❖ transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]

Results and discussion

CNN Vanilla



Runs: CNNvanilla
Views: Overview
Workers: 8789cbbe3813_669

Configuration

Number of Worker(s)	1
Device Type	GPU

GPU Summary

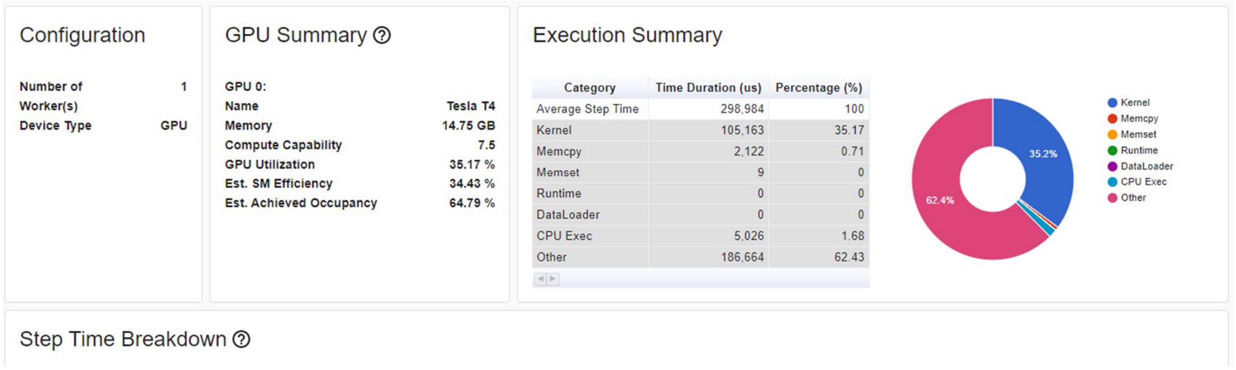
GPU 0:	
Name	Tesla T4
Memory	14.75 GB
Compute Capability	7.5
GPU Utilization	27.11 %
Est. SM Efficiency	26.4 %
Est. Achieved Occupancy	69.84 %

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	199,808	100
Kernel	54,163	27.11
Memcpy	2,128	1.07
Memset	24	0.01
Runtime	0	0
DataLoader	0	0
CPU Exec	14,949	7.48
Other	128,544	64.33

```
MyNet(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc): Linear(in_features=2048, out_features=10, bias=True)
)
```

CNN5



Configuration

Number of Worker(s)	1
Device Type	GPU

GPU Summary

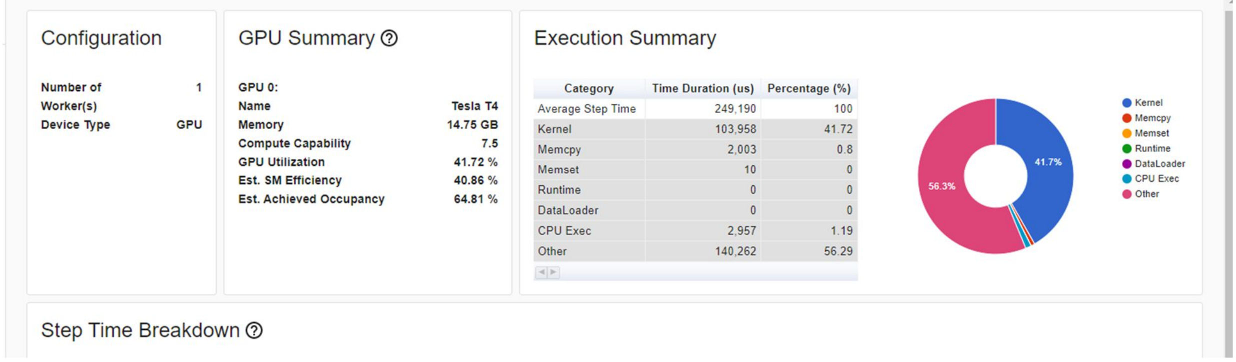
GPU 0:	
Name	Tesla T4
Memory	14.75 GB
Compute Capability	7.5
GPU Utilization	35.17 %
Est. SM Efficiency	34.43 %
Est. Achieved Occupancy	64.79 %

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	298,984	100
Kernel	105,163	35.17
Memcpy	2,122	0.71
Memset	9	0
Runtime	0	0
DataLoader	0	0
CPU Exec	5,026	1.68
Other	186,664	62.43

Step Time Breakdown

CNN4



Configuration

Number of Worker(s)	1
Device Type	GPU

GPU Summary

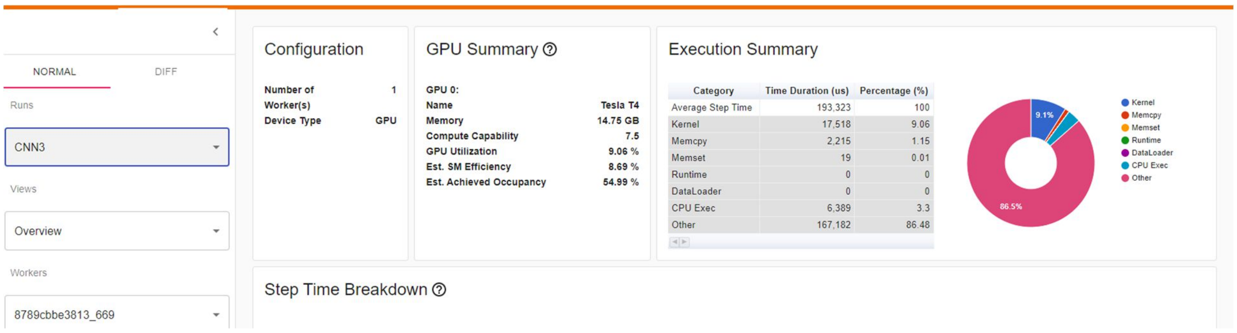
GPU 0:	
Name	Tesla T4
Memory	14.75 GB
Compute Capability	7.5
GPU Utilization	41.72 %
Est. SM Efficiency	40.86 %
Est. Achieved Occupancy	64.81 %

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	249,190	100
Kernel	103,958	41.72
Memcpy	2,003	0.8
Memset	10	0
Runtime	0	0
DataLoader	0	0
CPU Exec	2,957	1.19
Other	140,262	56.29

Step Time Breakdown

CNN3



Configuration

Number of Worker(s)	1
Device Type	GPU

GPU Summary

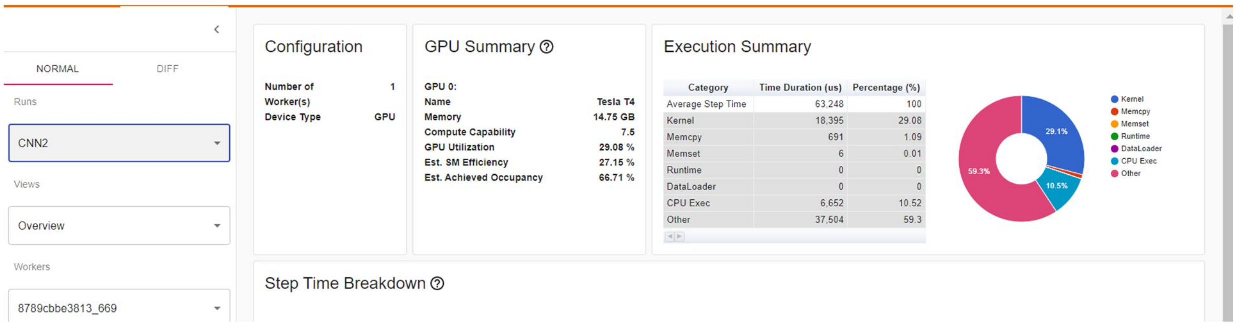
GPU 0:	
Name	Tesla T4
Memory	14.75 GB
Compute Capability	7.5
GPU Utilization	9.06 %
Est. SM Efficiency	8.69 %
Est. Achieved Occupancy	54.99 %

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	193,323	100
Kernel	17,518	9.06
Memcpy	2,215	1.15
Memset	19	0.01
Runtime	0	0
DataLoader	0	0
CPU Exec	6,389	3.3
Other	167,182	86.48

Step Time Breakdown

CNN2



Configuration

Number of Worker(s)	1
Device Type	GPU

GPU Summary

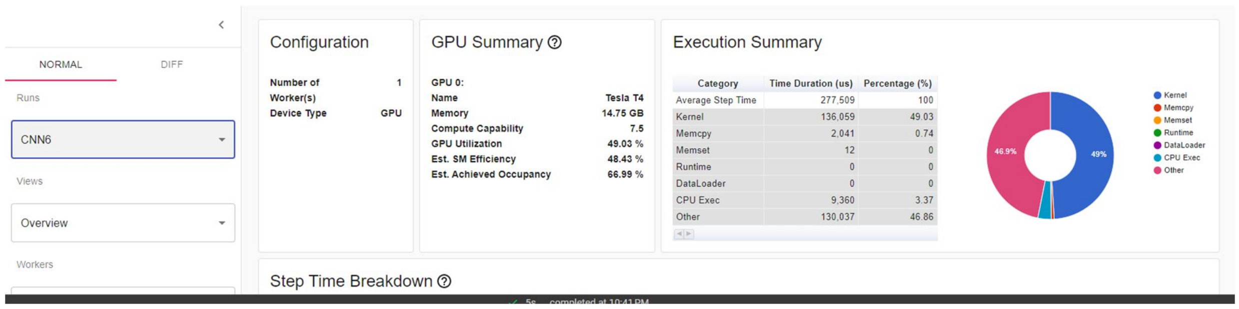
GPU 0:	
Name	Tesla T4
Memory	14.75 GB
Compute Capability	7.5
GPU Utilization	29.08 %
Est. SM Efficiency	27.15 %
Est. Achieved Occupancy	66.71 %

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	63,248	100
Kernel	18,395	29.08
Memcpy	691	1.09
Memset	6	0.01
Runtime	0	0
DataLoader	0	0
CPU Exec	6,652	10.52
Other	37,504	59.3

Step Time Breakdown

CNN 6



NORMAL DIFF

Runs: CNN6

Views: Overview

Workers

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	49.03 %
Est. SM Efficiency	48.43 %
Est. Achieved Occupancy	66.99 %

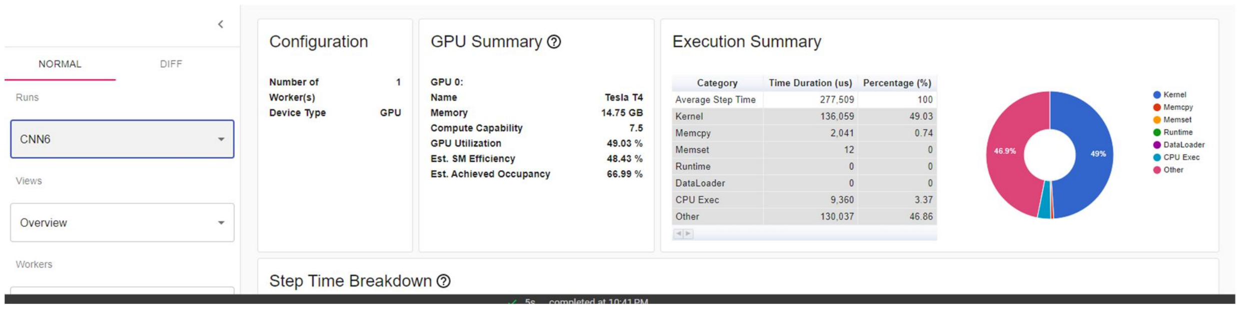
Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	277,509	100
Kernel	136,059	49.03
Memcpy	2,041	0.74
Memset	12	0
Runtime	0	0
DataLoader	0	0
CPU Exec	9,360	3.37
Other	130,037	46.86

Step Time Breakdown

5s completed at 10:41 PM

CNN7



NORMAL DIFF

Runs: CNN6

Views: Overview

Workers

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	49.03 %
Est. SM Efficiency	48.43 %
Est. Achieved Occupancy	66.99 %

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	277,509	100
Kernel	136,059	49.03
Memcpy	2,041	0.74
Memset	12	0
Runtime	0	0
DataLoader	0	0
CPU Exec	9,360	3.37
Other	130,037	46.86

Step Time Breakdown

5s completed at 10:41 PM

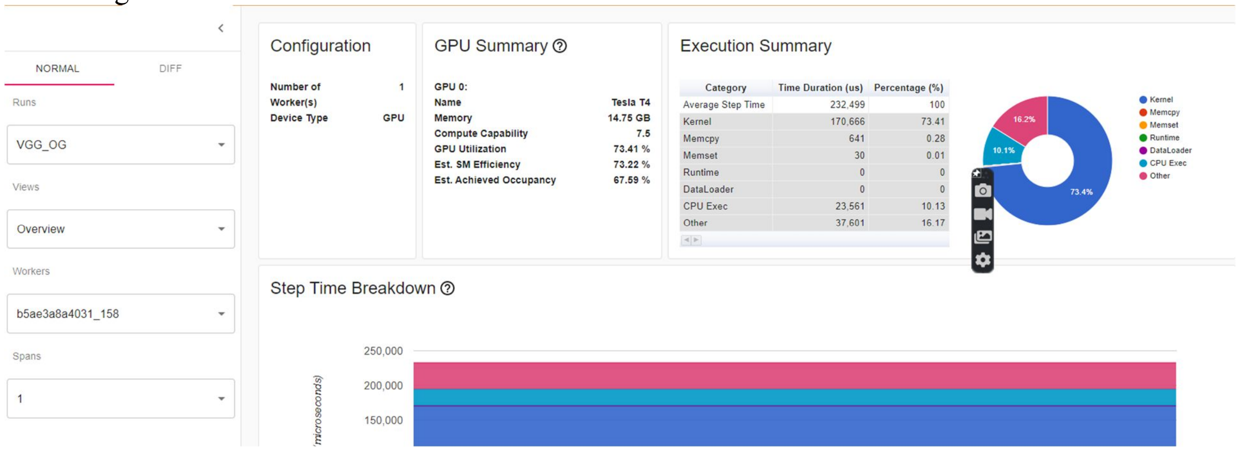
```
MyNet(
  (conv2): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv4): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc): Linear(in_features=8192, out_features=10, bias=True)
)
```

Discussion:

CNN7 gave the best result and should be used

VGG

VGG Original



NORMAL DIFF

Runs: VGG_OG

Views: Overview

Workers: b5ae3a8a4031_158

Spans: 1

Configuration

Number of Worker(s)	1
Device Type	GPU

GPU Summary

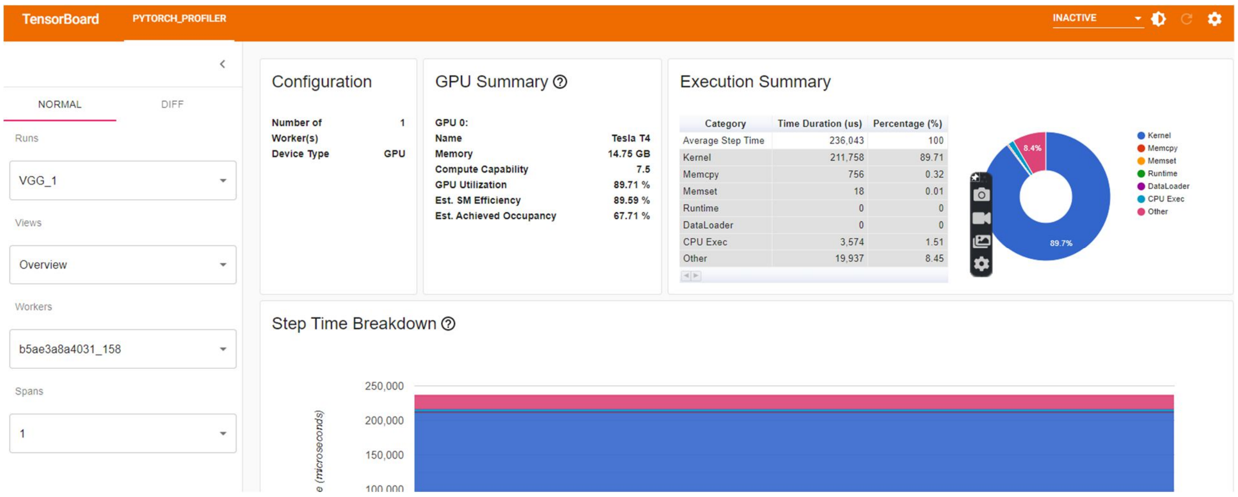
GPU 0:	
Name	Tesla T4
Memory	14.75 GB
Compute Capability	7.5
GPU Utilization	73.41 %
Est. SM Efficiency	73.22 %
Est. Achieved Occupancy	67.59 %

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	232,499	100
Kernel	170,666	73.41
Memcpy	641	0.28
Memset	30	0.01
Runtime	0	0
DataLoader	0	0
CPU Exec	23,561	10.13
Other	37,601	16.17

Step Time Breakdown

VGG1



Configuration

Number of Worker(s)	1
Device Type	GPU

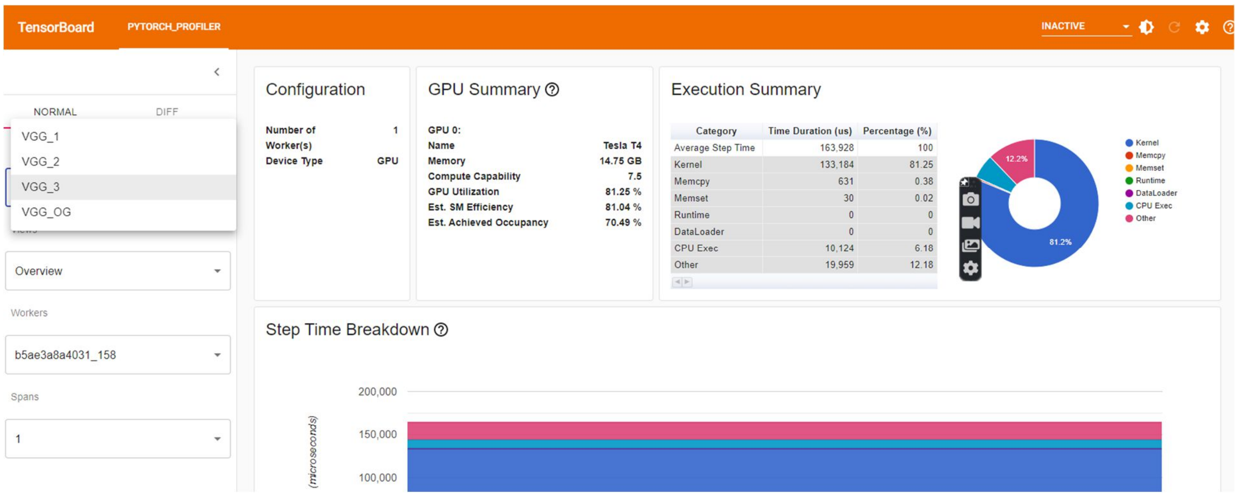
GPU Summary

GPU 0:	
Name	Tesla T4
Memory	14.75 GB
Compute Capability	7.5
GPU Utilization	89.71 %
Est. SM Efficiency	89.59 %
Est. Achieved Occupancy	67.71 %

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	236,043	100
Kernel	211,758	89.71
Memcpy	756	0.32
Memset	18	0.01
Runtime	0	0
DataLoader	0	0
CPU Exec	3,574	1.51
Other	19,937	8.45

VGG3



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	81.25 %
Est. SM Efficiency	81.04 %
Est. Achieved Occupancy	70.49 %

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	163,928	100
Kernel	133,184	81.25
Memcpy	631	0.38
Memset	30	0.02
Runtime	0	0
DataLoader	0	0
CPU Exec	10,124	6.18
Other	19,959	12.18

Discussion

VGG 1 gave the best Result where the first layer was removed