

Applied Accelerated Artificial Intelligence
Dr. Satyajit Das
Department of Computer Science and Engineering
Indian Institute of Technology, Palakkad

Introduction to Deep Learning
Lecture - 22
Profiling with DLProf PyTorch Catalyst Part - 1

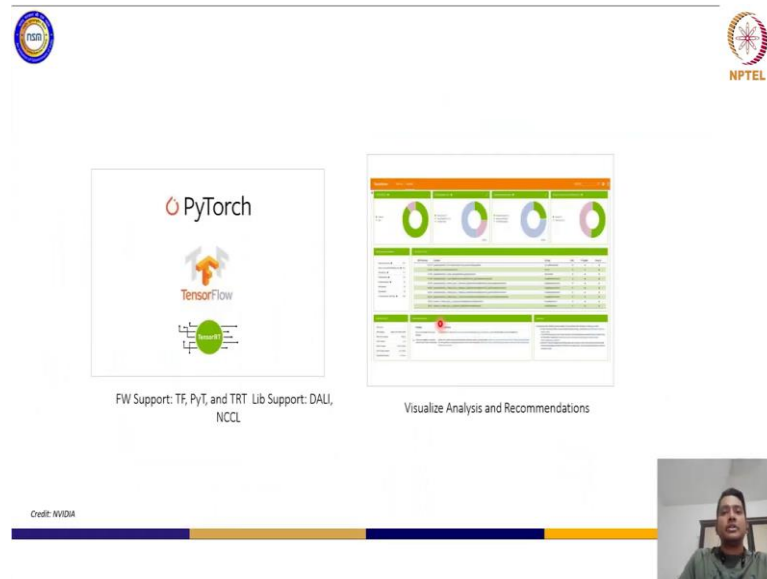
Good evening everybody. So, today we will start with Profiling with DLProf and then we will look into the PyTorch Catalyst. So, far what we have done? So far we have looked into how to make the pipeline for training in deep neural networks. So, deep learning algorithms that we have developed so far mostly in the domain of a classification, but the same pipeline will be followed for other kind of problems also.

So, but in tensor board in the last class that we whatever we have seen is how we can actually track or how we can see the performance metrics and maybe in the graphs for the computational graph for the algorithm and so. But here in DLProf what we will see is that we will profile the resource usage, the utilization, the recommendation from the from the profiler. So, all these things we will see in DLProf.

So, this is our new dimension of profiling that we will see today. And data scientists must be aware of such profiling because this will help essentially to improve the training performance in the end. So, in today's session, we will see one of such customization which will improve our performance of training module that we will run, ok. And we will see how much performance gain we are achieving or what are the improvements we are doing. So, all these things we will analyze.

And also in the next session, we will see the PyTorch catalyst. So, this is as name suggests, it is accelerated framework for PyTorch. This will give you the flexibility to efficiently wrap up everything whatever you want to do it in PyTorch in very simple terms, ok. So, that also we will see. So, that way we will get the enhancement of the training modules. So, all these things will be very modular and it will be easy to define the entire pipeline and also reproducibility will be very very high.

(Refer Slide Time: 02:54)

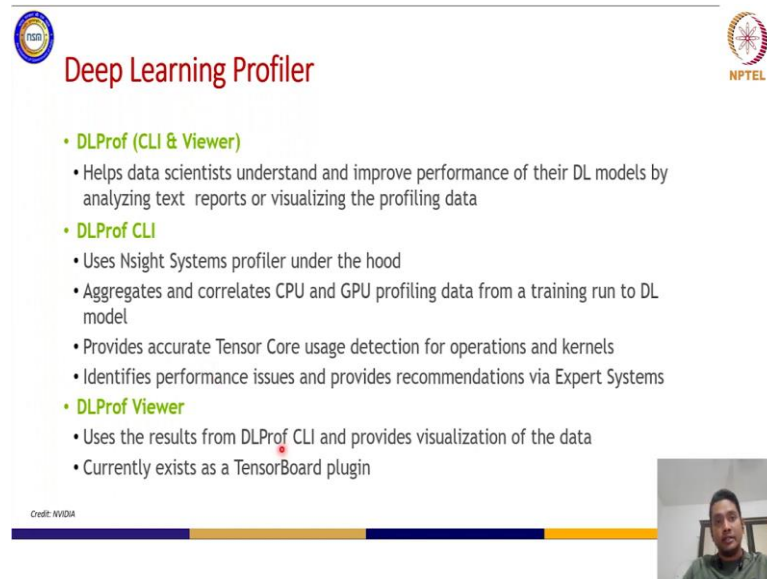


So, we will start with DLProf. So, let me; so as I was mentioning the DLProf will help you to visualize, analyze and recommend, ok. So, what kind of recommendations that we will see, as well as we will see what kind of profile that are coming from let us say the CPU usage the GPU usage. Also, inside the GPUs what are the cores that are being used how much time they are being used, and how you can improve the performance.

I mean the framework point of view this DLProf supports also by tensor flow you can work with TensorFlow and DLProf, you can work with PyTorch, TRT, also several libraries supported by it because you need to improve some performance and support some background as well. So, DALI and NCCL, so all these libraries are also supported.

So, today we will see how to use PyTorch modules to be profiled with the DLProf profile, ok.

(Refer Slide Time: 04:06)



The slide features the NVIDIA logo on the top left and the NPTEL logo on the top right. The title "Deep Learning Profiler" is centered at the top in red. Below the title, there are three main sections, each with a green bullet point:

- **DLProf (CLI & Viewer)**
 - Helps data scientists understand and improve performance of their DL models by analyzing text reports or visualizing the profiling data
- **DLProf CLI**
 - Uses Nsight Systems profiler under the hood
 - Aggregates and correlates CPU and GPU profiling data from a training run to DL model
 - Provides accurate Tensor Core usage detection for operations and kernels
 - Identifies performance issues and provides recommendations via Expert Systems
- **DLProf Viewer**
 - Uses the results from DLProf CLI and provides visualization of the data
 - Currently exists as a TensorBoard plugin

At the bottom left, it says "Credit: NVIDIA". On the bottom right, there is a small video inset showing a man speaking.

And so, before that we will need to understand and a few things basically what is DLProf it helps us to understand and improve the performance of the DL pipeline. And mostly the trained pipeline because it will be iterated several times as you have seen loop or several loops will be running based on your target one or target achievement.

Now, it will also generate reports or visualize the profile profiles. Now, DLProf CLI what it uses the Nsight systems profiler. So, we will enable this profiler to input the profiling data and it will make the database. Also, it will correlate the CPU usage and the GPU usage, because the both will be working in tandem and some data, it will be shared from or accessed from you and it will be shared through your GPU and GPU will aggregate or train your model depending on the data which is available.

So, the correlation between the CPU and GPU will be important and that we will directly get the data from here. As well as, it provides to track your tensor cores usage. Now, this is very very important because finally, we are talking about the deep learning modules for deep learning, training in particular. So, that means, like tensor cores must be used, right because tensor cores are specifically designed for your deep learning computational acceleration, right.

So, usage of tensor cores how it is being monitored or how it is being used by your algorithm that usage detection must be there. So, we will see how to track this. As well as, so some library that we will be using or enabling the tensor core usage.

Now, to profile or to see the recommendations you need to have one recommended system, right. So, the expert system provided by DLProf CLI will provide the recommendations. So, if let us say your tensor cores are underutilized based on the other course that are being utilized inside your GPU.

So, this expert recommended system will tell you that this these are not utilized. So, you can use this libraries to use to enhance the usage or maybe let us say you have some data bottleneck or memory bottleneck, so what kind of library you can use to remove that or eradicate the bottleneck. So, all these expert suggestions that also will be given by the recommended system.

And the viewer will essentially help you to visualize profiles and also you can log the reports and save them for your later analysis. And also, currently tensor board plugin is also available, so you can use tensor board assume to see or visualize the profiles, ok. So, either you use DLProf viewer or you use tensor board plugin, ok.

So, we will help you to see more, but I mean there will be some instructions to how the how you can get the profiles in tensor board and how you can create the profiles in viewer. So, all these things we will, ok.

(Refer Slide Time: 08:08)

GETTING STARTED

TensorFlow

1. TensorFlow and TRT require no additional code modification
2. Profile using DLProf CLI - prepend with **dlprof**
3. Visualize results with DLProf Viewer

PyTorch

1. Add few lines of code to your training script to enable **nvidia_dlprof_pytorch_nvtx** module
2. Profile using DLProf CLI - prepend with **dlprof**
3. Visualize with DLProf Viewer

Profile → Visualize in viewer

Add few lines of code → Profile → Visualize in viewer

Credit: NVIDIA

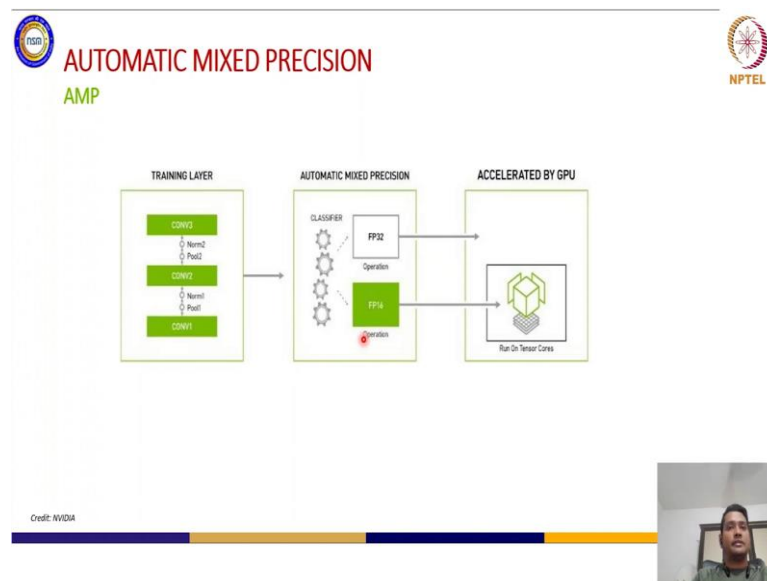
So, as I was mentioning that it supports tensor flow tensor RTs for these two frameworks, you do not take any additional code for modification to be used in DLProf. So, it will directly profile you the program and it will help you to visualize.

For PyTorch, you need to activate this module because this NVIDIA's profiler will be written. So, basically you will enable first and then you use your training pipeline or your multi-stage pipeline or whatever you want to use for your deep learning training.

Now, as I was mentioning in the previous slide the CLI will process this with DLProf and that will help you to visualize.

So, very simple few lines of code you need to add for PyTorch, but that is very minimal. Just one minute, ok.

(Refer Slide Time: 09:09)



Now, in the previous section I was mentioning that DLProf, the relationship between DLProf and this automatic precision, mix precision library is very impact because the proper tensor cores. So, the profiler actually tracks the usage or utilization of your tensor cores that are available inside your GPU.

So, that means, you want to have some computations to be offloaded into the tensor cores and automatic precision library will help you to do that, ok. So, that is why if you are not using automatic mixed precision library, then the tensor cores will be underutilized.

So, your training layer will have this pipeline. Let us say I have 2 convolution layers, 3 convolution layers here and normalization, pooling and so on. So, all these layers up there from input 1. So, this is one forward pass and this will be iterated several times.

Now, what I want to do? I want to convert or skills some of the parameters into some into some data types that is supported inside the tensor because all the data types which are let us say floating point single precision data type which are supported in the CUDA cores that are not supported in your tensor cores, ok.

So, for the tensor cores you need to scale your data which you use inside your training model 2 FP16, ok. So, basically, makes precision you will do a computing, ok. You will use both the data types inside your training to be able to optimize and also gets better results, in that right.

So, some of the data will be processed in the rest of the course and the scaled data which is FP16 will be run on the tensor cores. So, that is the purpose of your automatic mixed precision library.

Now, automatic mixed precision library has two sources. If you are using NVIDIA, so basically you will be using NVIDIA apex and for generalized use of mixed precision library you can use native PyTorch amp also. So, there are two sources you can use either you can use NVIDIA apex or native PyTorch amp, ok.

So, this is going to be interesting because you will see that some of the I mean most of the computations which will be run on your tensor cores to convert them or to skill them, you will need very minimal product. So, for use of automatic mixed precision library, it is also very easy.

(Refer Slide Time: 12:13)

BEFORE YOU PROFILE

Do:

- make sure your code runs without an issue
- make a habit of using profiler when you make changes to your code
- Observe if changes you made improve the training performance
- get familiar with the optional arguments DLProf provides
- Iteration range, delay, duration and etc.,

Don't:

- profile for extended periods of time. It will take very long to profile
- DL training is repetitive and you only need a couple minutes to profile to learn
- try to open DLProf database with your TensorBoard
- You need NVIDIA TB GPU plugin to visualize DLProf event files

Credit: NVIDIA

Before going into profiler you need to know certain things to do and to follow, not to follow because people often spin or some resources because once you start profiling many resources will be used, ok.

So, what to do? Make sure that your code runs without an issue because we just check it before running into the into the DLprofiler. We just check it whether your training module is working perfectly or not. Just one run will be will suffice that without any error it is trained.

And also you need to make a habit of using profiler when you make changes in it because each change can replicate or it can offer you a cascading effect which will enhance the either the performance or it can degrade the performance, ok.

So, several order of magnitude of your performance can increase or decrease. So, how your change in the code helps either increasing or decreasing the performance that you need to check everything, and this is just one rule. If you know that this is going to increase your performance that is good.

But in several cases you will see when you are working with a big pipeline, you will see maintaining certain aspects of speedup will not be actually happening as prescribed by the library. So, they will say that this library will help you to enhance the performance, but in some cases it will not happen because of that where you have defined the

performance. So, basically, you need to check every time whether each change is giving you other than your desired performance training.

And also you need to get familiarized with some arguments in the DLProf which we provided. And of course, the iteration range, delay, duration etc also you need to get familiar because all these aspects; so basically iteration range, how this will affect your training, how delay, how much duration you are you are actually spending for making certain modules run. So, all these things you need to be, at least in some abstract way you need to do, ok.

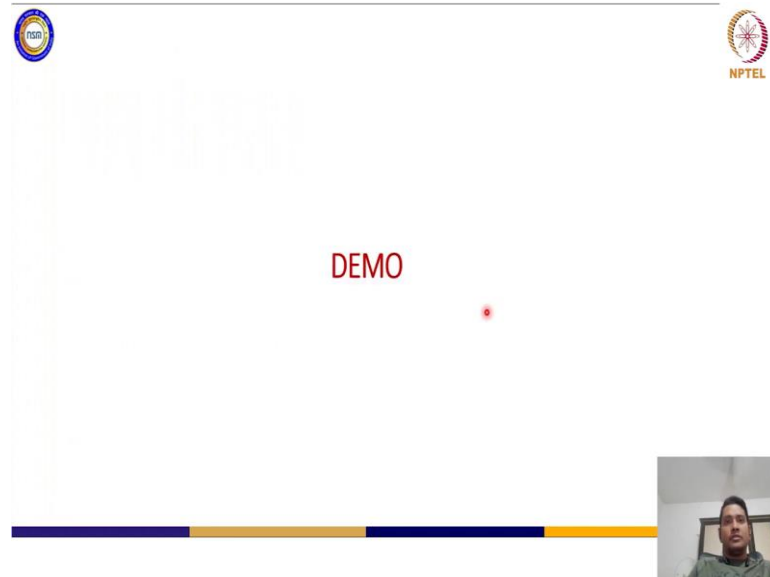
And of course, you should not profile for extended period of time because you should not train the entire thing to get correct profile for your training because training module is essentially one iterative process, right. And the same thing will be repeated every time.

So, few iterations if you check for your training model that will give you overall idea that how it is going to perform. You do not need to go through the entire training process for all the epochs or for all the iterations you do not go. So, few of iterations will help you understand that, to be frank it is going to be it makes some modifications or needs an update.

DL training is repetitive as I was mentioning, and that is why you need to spend a couple of minutes to profile it and that will suffice. Also, try to open DLProf database with your tensor board or DLProf viewer.

And so, basically this the logs will be saved as database and the recommended systems are tensor board and profiler. So, you can use other yeah. So, it will help to visualize the DL profiler event files. So, basically NVIDIA test bench GPU plugin you need to use, ok.

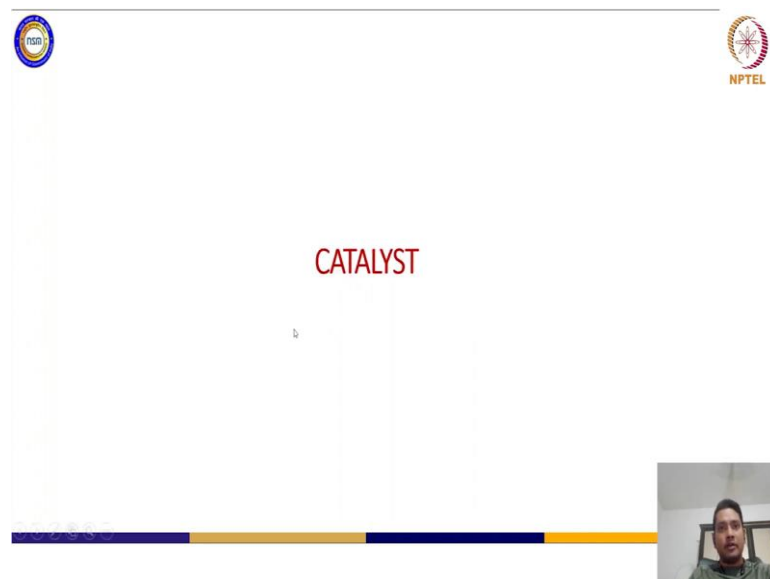
(Refer Slide Time: 16:20)



The slide features a white background with a blue and yellow decorative bar at the bottom. In the top left corner is a circular logo with the text 'NPTEL'. In the top right corner is the NPTEL logo, which includes a stylized figure and the text 'NPTEL'. The word 'DEMO' is written in red in the center of the slide. A small red dot is positioned to the right of the word. In the bottom right corner, there is a small video inset showing a man with dark hair and a beard, wearing a green shirt, looking towards the camera.

So, let us go to the demo section now.

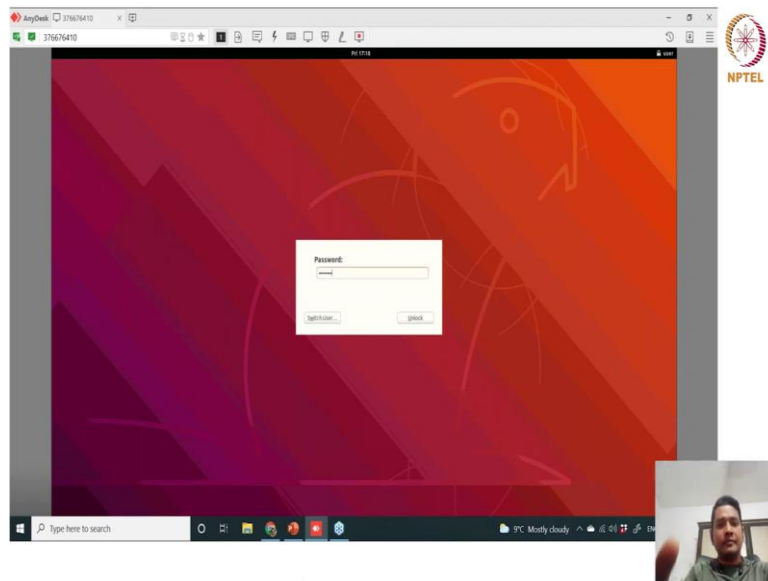
(Refer Slide Time: 16:25)



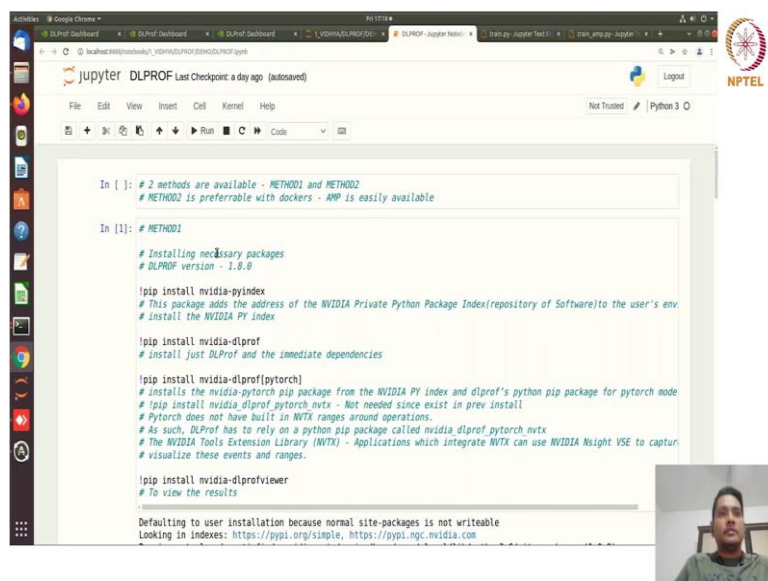
The slide features a white background with a blue and yellow decorative bar at the bottom. In the top left corner is a circular logo with the text 'NPTEL'. In the top right corner is the NPTEL logo, which includes a stylized figure and the text 'NPTEL'. The word 'CATALYST' is written in red in the center of the slide. In the bottom right corner, there is a small video inset showing the same man from the previous slide, wearing a green shirt, looking towards the camera.

To demonstrate the DLProf, how to do that and then we will come to explanation, ok.

(Refer Slide Time: 16:32)

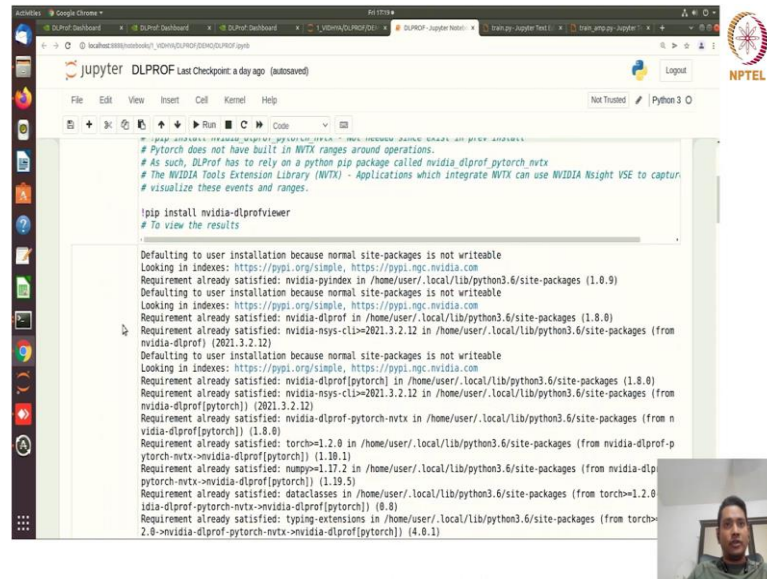


(Refer Slide Time: 16:44)



So, now let me do this. Let me explain. So, basically there are two methods to run DLProf, one is through your notebook and through the command. So, if you have installed any you know or if you have not installed it or if you have to use the DLProf in Jupyter notebook or any other notebook you can use.

(Refer Slide Time: 17:17)

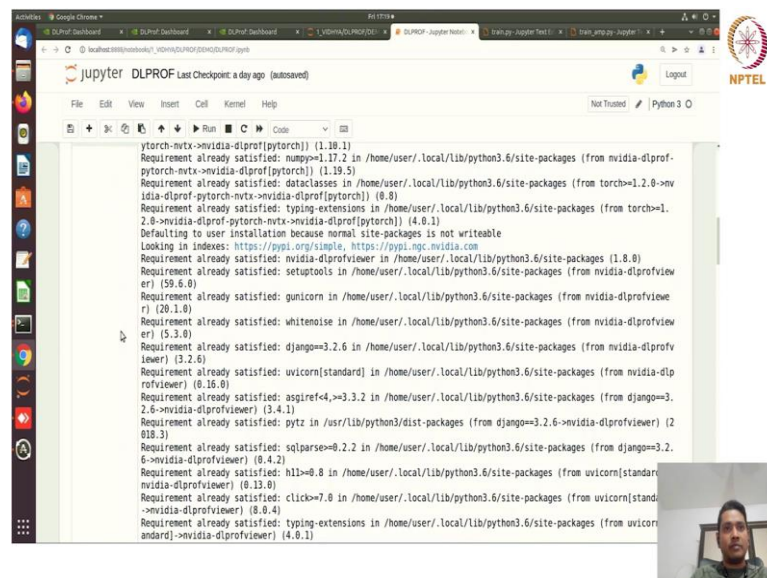


```
!pip install nvidia-dlprof
# To view the results

Defaulting to user installation because normal site-packages is not writable
Looking in indexes: https://pypi.org/simple, https://pypi.ngc.nvidia.com
Requirement already satisfied: nvidia-pyindex in /home/user/.local/lib/python3.6/site-packages (1.0.9)
Defaulting to user installation because normal site-packages is not writable
Looking in indexes: https://pypi.org/simple, https://pypi.ngc.nvidia.com
Requirement already satisfied: nvidia-dlprof in /home/user/.local/lib/python3.6/site-packages (1.8.0)
Requirement already satisfied: nvidia-sys-cti>=2021.3.2.12 in /home/user/.local/lib/python3.6/site-packages (from nvidia-dlprof) (2021.3.2.12)
Defaulting to user installation because normal site-packages is not writable
Looking in indexes: https://pypi.org/simple, https://pypi.ngc.nvidia.com
Requirement already satisfied: nvidia-dlprof in /home/user/.local/lib/python3.6/site-packages (1.8.0)
Requirement already satisfied: nvidia-sys-cti>=2021.3.2.12 in /home/user/.local/lib/python3.6/site-packages (from nvidia-dlprof[pytorch]) (2021.3.2.12)
Requirement already satisfied: nvidia-dlprof-pytorch-nvtx in /home/user/.local/lib/python3.6/site-packages (from nvidia-dlprof[pytorch]) (1.18.1)
Requirement already satisfied: torch>=1.2.0 in /home/user/.local/lib/python3.6/site-packages (from nvidia-dlprof-pytorch-nvtx->nvidia-dlprof[pytorch]) (1.10.1)
Requirement already satisfied: numpy>=1.17.2 in /home/user/.local/lib/python3.6/site-packages (from nvidia-dlprof-pytorch-nvtx->nvidia-dlprof[pytorch]) (1.19.5)
Requirement already satisfied: typing-extensions in /home/user/.local/lib/python3.6/site-packages (from torch>=1.2.0->nvidia-dlprof-pytorch-nvtx->nvidia-dlprof[pytorch]) (4.8.1)
Requirement already satisfied: dataclasses in /home/user/.local/lib/python3.6/site-packages (from torch>=1.2.0->nvidia-dlprof-pytorch-nvtx->nvidia-dlprof[pytorch]) (0.8)
Requirement already satisfied: typing-extensions in /home/user/.local/lib/python3.6/site-packages (from torch>=1.2.0->nvidia-dlprof-pytorch-nvtx->nvidia-dlprof[pytorch]) (4.8.1)
```

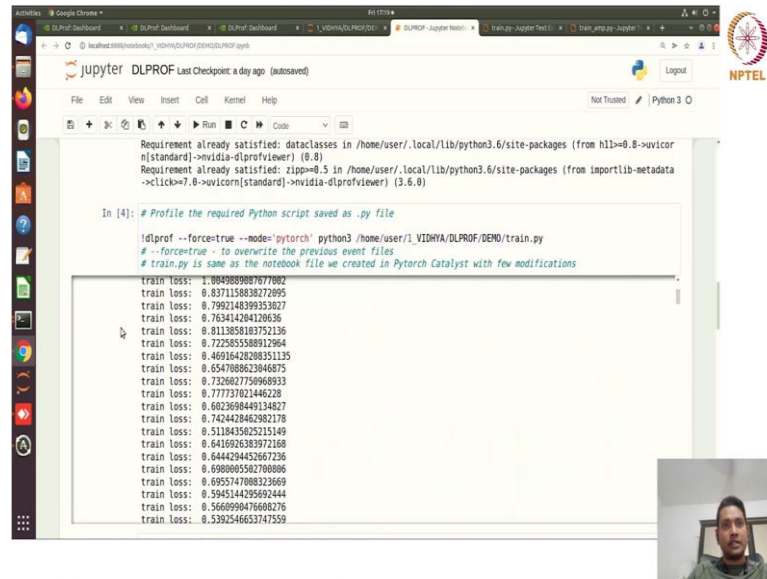
So, the method 1 is essentially the first method which is used in the notebook. We will not go into details. You can see the same details in the next section as well.

(Refer Slide Time: 17:26)



```
Requirement already satisfied: typing-extensions in /home/user/.local/lib/python3.6/site-packages (from torch>=1.2.0->nvidia-dlprof-pytorch-nvtx->nvidia-dlprof[pytorch]) (4.8.1)
Requirement already satisfied: dataclasses in /home/user/.local/lib/python3.6/site-packages (from torch>=1.2.0->nvidia-dlprof-pytorch-nvtx->nvidia-dlprof[pytorch]) (0.8)
Defaulting to user installation because normal site-packages is not writable
Looking in indexes: https://pypi.org/simple, https://pypi.ngc.nvidia.com
Requirement already satisfied: nvidia-dlprof in /home/user/.local/lib/python3.6/site-packages (1.8.0)
Requirement already satisfied: setuptools in /home/user/.local/lib/python3.6/site-packages (from nvidia-dlprofviewer) (59.5.0)
Requirement already satisfied: gunicorn in /home/user/.local/lib/python3.6/site-packages (from nvidia-dlprofviewer) (20.1.0)
Requirement already satisfied: whitenoise in /home/user/.local/lib/python3.6/site-packages (from nvidia-dlprofviewer) (5.3.0)
Requirement already satisfied: django=3.2.6 in /home/user/.local/lib/python3.6/site-packages (from nvidia-dlprofviewer) (3.2.6)
Requirement already satisfied: uvicorn[standard] in /home/user/.local/lib/python3.6/site-packages (from nvidia-dlprofviewer) (0.16.0)
Requirement already satisfied: asgiref<4, >=3.3.2 in /home/user/.local/lib/python3.6/site-packages (from django=3.2.6->nvidia-dlprofviewer) (3.4.1)
Requirement already satisfied: pytz in /usr/lib/python3/dist-packages (from django=3.2.6->nvidia-dlprofviewer) (2018.3)
Requirement already satisfied: sqlparse>=0.2.2 in /home/user/.local/lib/python3.6/site-packages (from django=3.2.6->nvidia-dlprofviewer) (0.4.2)
Requirement already satisfied: h11>=0.8 in /home/user/.local/lib/python3.6/site-packages (from uvicorn[standard]->nvidia-dlprofviewer) (0.13.8)
Requirement already satisfied: click>=7.0 in /home/user/.local/lib/python3.6/site-packages (from uvicorn[standard]->nvidia-dlprofviewer) (8.0.4)
Requirement already satisfied: typing-extensions in /home/user/.local/lib/python3.6/site-packages (from uvicorn[standard]->nvidia-dlprofviewer) (4.8.1)
```

(Refer Slide Time: 17:27)



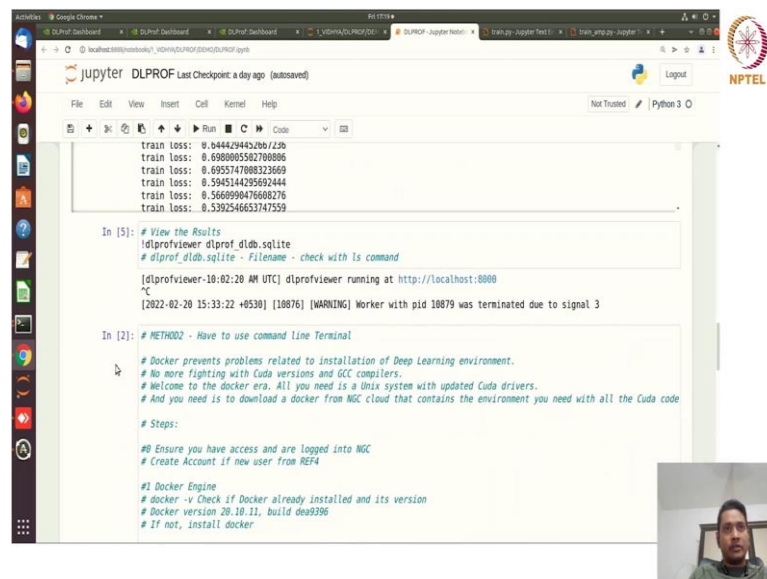
The screenshot shows a JupyterLab interface with a terminal window. The terminal displays the following text:

```
Requirement already satisfied: dataclasses in /home/user/.local/lib/python3.6/site-packages (from h11=>0.8-suvicorn[standard]->midia-dlprofviewer) (0.8)
Requirement already satisfied: zipp=>0.5 in /home/user/.local/lib/python3.6/site-packages (from importlib-metadata->click=>7.0-suvicorn[standard]->midia-dlprofviewer) (3.6.0)

In [4]: # Profile the required Python script saved as .py file
!dlprof --force=true --mode='pytorch' python3 /home/user/1_VIDHYA/DLPROF/DEND/train.py
# --force=true - to overwrite the previous event files
# train.py is same as the notebook file we created in Pytorch Catalyst with few modifications

train loss: 1.0849898987677802
train loss: 0.8371158838272095
train loss: 0.7992148399352027
train loss: 0.76341204120626
train loss: 0.8113858183752136
train loss: 0.722585588912964
train loss: 0.46916428208351135
train loss: 0.6547080623046075
train loss: 0.722621759606933
train loss: 0.777737021446228
train loss: 0.6623698449134827
train loss: 0.742442846392178
train loss: 0.511843502215149
train loss: 0.6416926383972168
train loss: 0.6444294452667236
train loss: 0.698005502708086
train loss: 0.6955147088323669
train loss: 0.5945144295692444
train loss: 0.566090476608276
train loss: 0.5392546653747559
```

(Refer Slide Time: 17:28)



The screenshot shows a JupyterLab interface with a terminal window. The terminal displays the following text:

```
train loss: 0.6444294452667236
train loss: 0.698005502708086
train loss: 0.6955147088323669
train loss: 0.5945144295692444
train loss: 0.566090476608276
train loss: 0.5392546653747559

In [5]: # View the Results
!dlprofviewer dlprof.dlprof.sqlite
# dlprof.dlprof.sqlite - Filename - check with ls command

[dlprofviewer-10:02:20 AM UTC] dlprofviewer running at http://localhost:8080
^C
[2022-02-20 15:33:22 +0530] [10876] [WARNING] Worker with pid 10879 was terminated due to signal 3

In [2]: # METHOD02 - Have to use command line Terminal

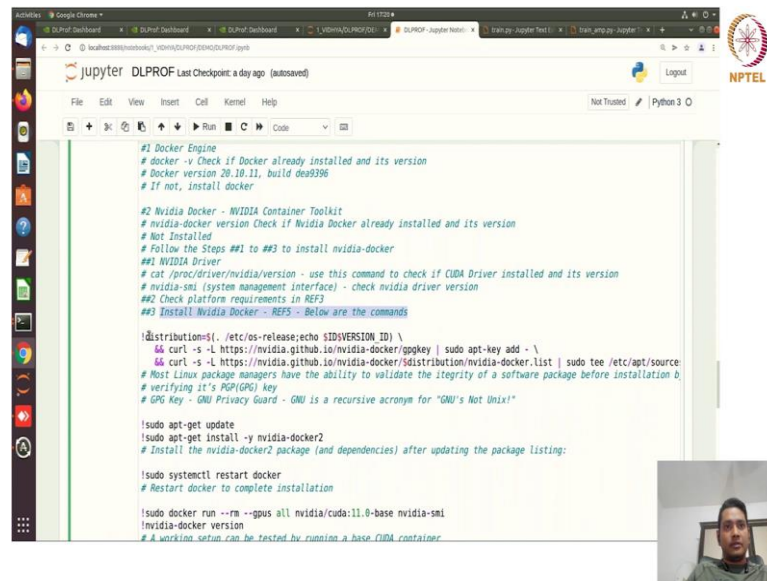
# Docker prevents problems related to installation of Deep Learning environment.
# No more fighting with Cuda versions and GCC compilers.
# Welcome to the docker era. All you need is a Unix system with updated Cuda drivers.
# And you need is to download a docker from NGC cloud that contains the environment you need with all the Cuda code

# Steps:

#0 Ensure you have access and are logged into NGC
# Create Account if new user from REF4

#1 Docker Engine
# docker -v Check if Docker already installed and its version
# docker version 20.10.11, build dea9396
# If not, install docker
```

(Refer Slide Time: 17:30)



```
#1 Docker Engine
# docker -y Check if Docker already installed and its version
# Docker version 20.10.11, build dea9396
# If not, install docker

#2 Nvidia Docker - NVIDIA Container Toolkit
# nvidia-docker version Check if Nvidia Docker already installed and its version
# Not Installed
# Follow the Steps #1 to #3 to install nvidia-docker

#1 NVIDIA Driver
# cat /proc/driver/nvidia/version - use this command to check if CUDA Driver installed and its version
# nvidia-smi (system management interface) - check nvidia driver version
#2 Check platform requirements in REF3
#3 Install Nvidia Docker - REFS - Below are the commands

!distribution=$(cat /etc/os-release; echo ${IDVERSION_ID}) \
  && curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add - \
  && curl -s -L https://nvidia.github.io/nvidia-docker/nvidia-docker.list | sudo tee /etc/apt/sources.list
# Most Linux package managers have the ability to validate the integrity of a software package before installation b
# verifying it's PGP(GPG) key
# GPG Key - GNU Privacy Guard - GNU is a recursive acronym for "GNU's Not Unix!"

!sudo apt-get update
!sudo apt-get install -y nvidia-docker2
# Install the nvidia-docker2 package (and dependencies) after updating the package listing:

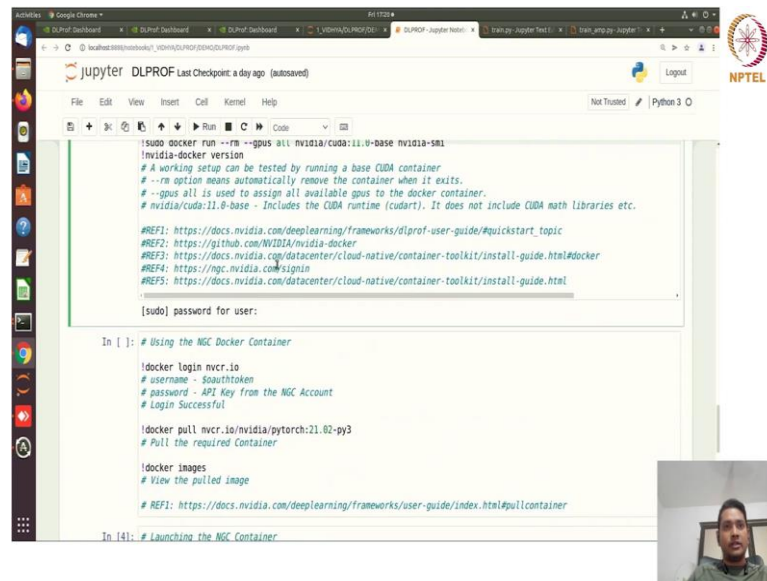
!sudo systemctl restart docker
# Restart docker to complete installation

!sudo docker run --rm --gpus all nvidia/cuda:11.0-base nvidia-smi
!nvidia-docker version
# A working setup can be tested by running a base CUDA container.
```

So, in the method 2, we will discuss about how to use in command line. So, basically you need to make sure that you may have access in NGC. You need to create one account for that and you need to have Docker Engine installed.

And if you do not have you can check these commands to install it. And then you can install the container toolkit and you can start using the check the whether necessary drivers are there or not and then we install the NVIDIA Docker for the. So, this is the distribution that we want to make available inside my Docker. So, basically you just install it with this command.

(Refer Slide Time: 18:22)



The screenshot shows a Jupyter Notebook interface with a terminal window. The terminal output includes the following code and comments:

```
!sudo docker run --rm --gpus all nvidia/cuda:11.0-base nvidia-smi
!nvidia-docker version
# A working setup can be tested by running a base CUDA container
# --rm option means automatically remove the container when it exits.
# --gpus all is used to assign all available gpus to the docker container.
# nvidia/cuda:11.0-base - Includes the CUDA runtime (cudart). It does not include CUDA math libraries etc.

#REF1: https://docs.nvidia.com/deeplearning/frameworks/dlprof-user-guide/#quickstart_topic
#REF2: https://github.com/NVIDIA/nvidia-docker
#REF3: https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html#docker
#REF4: https://ngc.nvidia.com/signup
#REF5: https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html

[sudo] password for user:

In [ ]: # Using the NGC Docker Container
!docker login nvr.io
# username - Southtoken
# password - API Key from the NGC Account
# Login Successful

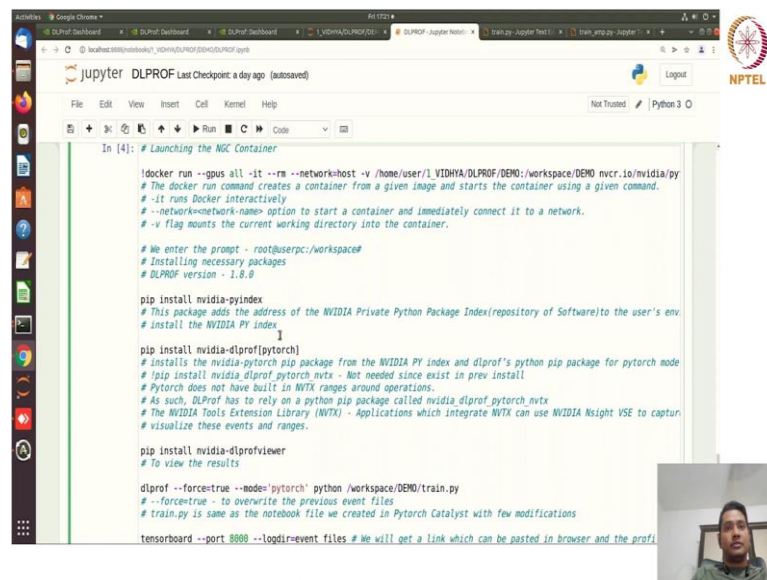
!docker pull nvr.io/nvidia/pytorch:21.02-py3
# Pull the required Container

!docker images
# View the pulled image
# REF: https://docs.nvidia.com/deeplearning/frameworks/user-guide/index.html#pullcontainer

In [1]: # Launching the NGC Container
```

And then once it is installed, so basically, you need to install this package for your use and then you restart the Docker, and then you start it and just check the version. So, this is the complete installation of your Docker and check the version. And once it is successfully installed, then using this NGC Docker then we will start using that.

(Refer Slide Time: 18:46)



The screenshot shows a Jupyter Notebook interface with a terminal window. The terminal output includes the following code and comments:

```
In [4]: # Launching the NGC Container
!docker run --gpus all -it --rm --network=host -v /home/user1/VIDHYA/DLPROF/DEMO:/workspace/DEMO nvr.io/nvidia/py
# The docker run command creates a container from a given image and starts the container using a given command.
# -it runs Docker interactively
# --network=<network-name> option to start a container and immediately connect it to a network.
# -v flag mounts the current working directory into the container.

# We enter the prompt - root@userpc:/workspace#
# Installing necessary packages
# DLPROF version - 1.8.0

pip install nvidia-pyindex
# This package adds the address of the NVIDIA Private Python Package Index(repository of Software)to the user's env
# Install the NVIDIA PY index

pip install nvidia-dlprof[pytorch]
# Installs the nvidia-pytorch pip package from the NVIDIA PY index and dlprof's python pip package for pytorch mode
# !pip install nvidia_dlprof_pytorch nvtx - Not needed since exist in prev install
# Pytorch does not have built in NVTX ranges around operations.
# As such, DLPROF has to rely on a python pip package called nvidia_dlprof_pytorch nvtx
# The NVIDIA Tools Extension Library (NVTX) - Applications which integrate NVTX can use NVIDIA Nsight VSE to capture
# visualize these events and ranges.

pip install nvidia-dlprofviewer
# To view the results

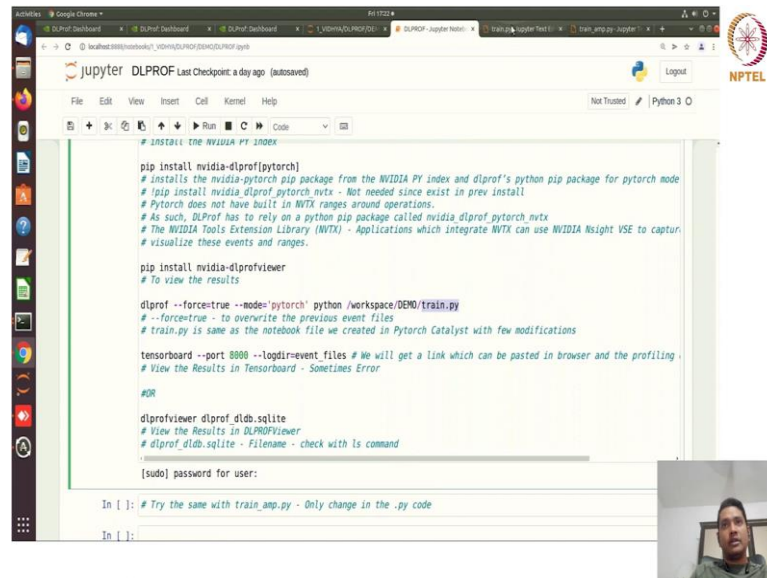
dlprof --force=true --mode='pytorch' python /workspace/DEMO/train.py
# --force=true - to overwrite the previous event files
# train.py is same as the notebook file we created in Pytorch Catalyst with few modifications

tensorboard --port 8000 --logdir=event_files # We will get a link which can be pasted in browser and the profi
```

So, basically you need to pull the Docker image, ok. So, this is the NGC Docker container or usage. So, once it is pulled then we can just run this comment to launch the NGC container now.

And once container is launched then you can install the necessary pack. So, what are the necessary packages for here? So, basically, I need to install nvidia-pyindex, nvidia-dlprof with pytorch. You can this also tensor flow as I was mentioning.

(Refer Slide Time: 19:30)



```
# Install the NVIDIA PY index
pip install nvidia-dlprof[pytorch]
# Installs the nvidia-pytorch pip package from the NVIDIA PY index and dlprof's python pip package for pytorch mode
# pip install nvidia_dlprof_pytorch_nvtx - Not needed since exist in prev install
# Pytorch does not have built in NVTX ranges around operations.
# As such, DLPROF has to rely on a python pip package called nvidia dlprof pytorch nvtx
# The NVIDIA Tools Extension Library (NVTX) - Applications which integrate NVTX can use NVIDIA Nsight VSE to capture
# visualize these events and ranges.

pip install nvidia-dlprofviewer
# To view the results

dlprof --force=true --mode='pytorch' python /workspace/DENO/train.py
# --force=true - to overwrite the previous event files
# train.py is same as the notebook file we created in Pytorch Catalyst with few modifications

tensorboard --port 8000 --logdir=event_files # We will get a link which can be pasted in browser and the profiling
# View the Results in Tensorboard - Sometimes Error

#OR

dlprofviewer dlprof_dldb.sqlite
# View the Results in DLPROFViewer
# dlprof_dldb.sqlite - Filename - check with ls command

[sudo] password for user:

In [ ]: # Try the same with train_amp.py - Only change in the .py code

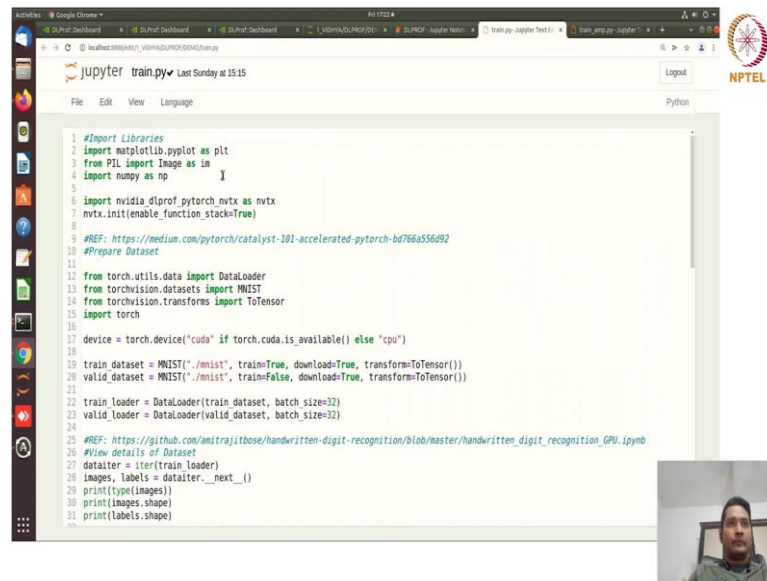
In [ ]:
```

So, here we are using PyTorch and also you need to install nvidia-dlprofviewer because we are using dlprofviewer to visualize the profiles, ok.

Now, once you have successfully completed up to this. So, basically you have installed all the packages, now you can use your dlprof in the next column. But before that you need to have the training.py because the training module that I want to profile using this dlprof just targeted prospects.

Now, once you have this training.py, so basically let us see what is in the training.py.

(Refer Slide Time: 20:13)

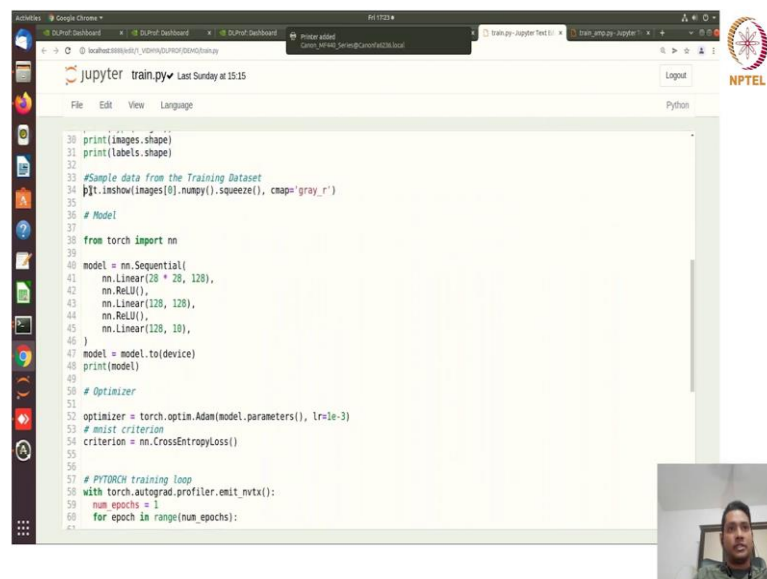


```
1 #Import Libraries
2 import matplotlib.pyplot as plt
3 from PIL import Image as im
4 import numpy as np
5
6 import nvidia.dlprof_pytorch.nvtx as nvtx
7 nvtx.init(enable_function_stack=True)
8
9 #REF: https://medium.com/pytorch/catalyst-101-accelerated-pytorch-bd766a556d92
10 #Prepare Dataset
11
12 from torch.utils.data import DataLoader
13 from torchvision.datasets import MNIST
14 from torchvision.transforms import ToTensor
15 import torch
16
17 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
18
19 train_dataset = MNIST("./mnist", train=True, download=True, transform=ToTensor())
20 valid_dataset = MNIST("./mnist", train=False, download=True, transform=ToTensor())
21
22 train_loader = DataLoader(train_dataset, batch_size=32)
23 valid_loader = DataLoader(valid_dataset, batch_size=32)
24
25 #REF: https://github.com/amitrajitbose/handwritten-digit-recognition/blob/master/handwritten_digit_recognition_GPU.ipynb
26 #View details of Dataset
27 dataloader = iter(train_loader)
28 images, labels = dataloader.__next__()
29 print(type(images))
30 print(images.shape)
31 print(labels.shape)
32
```

So, training.py is essentially the pipeline for training learning models. Now, the some of the packages that you have seen so far because we have used them, so basically these imports for your torch, torchvision that you have seen, few others that we need some libraries for images and plots, so numpy, PIL and matplotlib.

And also you need to import the nvtx, so because this will enable this stack tracers for the profiles that will generate from PyTorch training run, ok. So, this is the only one line you need to add to re-enable the profile from this training model, ok.

(Refer Slide Time: 21:04)

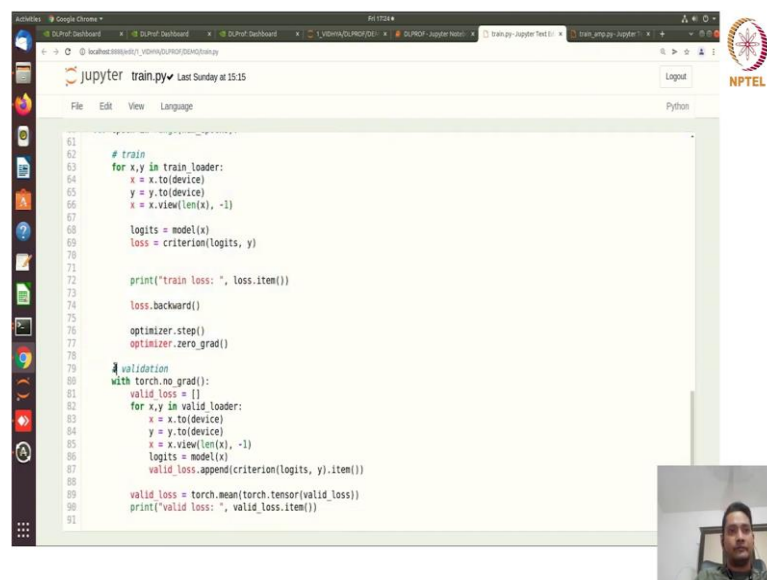


```
30 print(images.shape)
31 print(labels.shape)
32
33 #Sample data from the Training Dataset
34 plt.imshow(images[0].numpy().squeeze(), cmap='gray_r')
35
36 # Model
37
38 from torch import nn
39
40 model = nn.Sequential(
41     nn.Linear(28 * 28, 128),
42     nn.ReLU(),
43     nn.Linear(128, 128),
44     nn.ReLU(),
45     nn.Linear(128, 10),
46 )
47 model = model.to(device)
48 print(model)
49
50 # Optimizer
51
52 optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
53 # mnist criterion
54 criterion = nn.CrossEntropyLoss()
55
56
57 # PYTORCH training loop
58 with torch.autograd.profiler.emit_nvtx():
59     num_epochs = 1
60     for epoch in range(num_epochs):
61
```


So, now, once we have all the necessary packages imported, you just specify the device. These are as usual, nothing to nothing new. So, all the PyTorch things, train set data set, valid data set, so here we are doing training and validation. So, we are just using the train loader and valid loader with it this number of batch size.

Now, notice that we have not enabled the number of workers here, ok. So, we want to see what happens, ok. Now, once you have defined the train loader and data loader, you just you can print some images, you can see, ok. So, these are the things that we want to do before going into the training part.

(Refer Slide Time: 21:56)



```
61
62 # train
63 for x,y in train_loader:
64     x = x.to(device)
65     y = y.to(device)
66     x = x.view(len(x), -1)
67
68     logits = model(x)
69     loss = criterion(logits, y)
70
71
72     print("train loss: ", loss.item())
73
74     loss.backward()
75
76     optimizer.step()
77     optimizer.zero_grad()
78
79 # validation
80 with torch.no_grad():
81     valid_loss = []
82     for x,y in valid_loader:
83         x = x.to(device)
84         y = y.to(device)
85         x = x.view(len(x), -1)
86         logits = model(x)
87         valid_loss.append(criterion(logits, y).item())
88
89     valid_loss = torch.mean(torch.tensor(valid_loss))
90     print("valid loss: ", valid_loss.item())
91
```

So, this is the class where we are defining the model. So, this is the model definition and we are transferring the model to the device. We are defining the optimizer, the criterion for the loss and then this is the training loop that we have defined, ok.

And with torch.autograd.profiler, now we are starting the profiling or emitting the stresses from here because. Before that we do not need all this. So, for this loop only I need to profile, because this is the loop for your train.

Now, how many epochs? You just define it. So, for as I was mentioning you do not need to run the training for all the loops, just only one single run will suffice or maybe 2, 3 runs. If you want to have some more satisfaction you can do 2, 3 runs of inter loop and

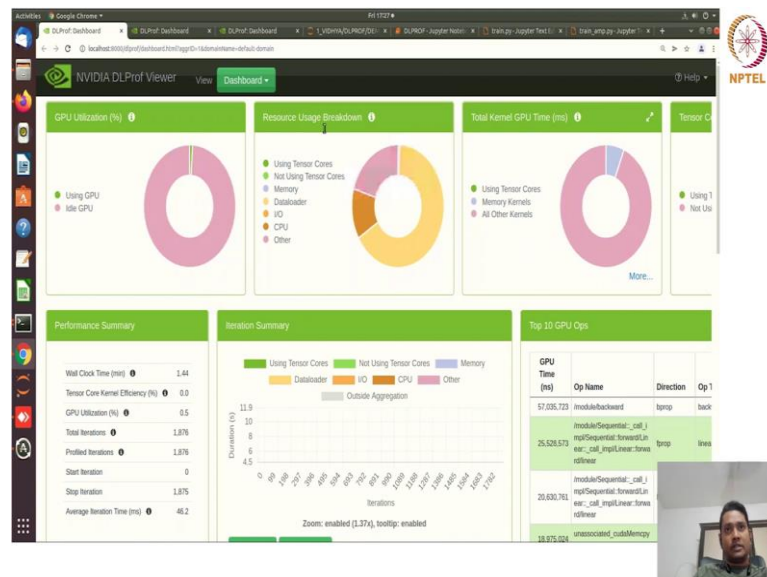
you will get the idea. And this is the validation, ok. So, this is the entire training module which I have written.

So, now we will go to the code. So, basically, here to view the results you need to run dlprof with this training dot. So, this is the comment to export the process into your databases.

Then, once you run that you will see that you have, so basically what we have done this we have; yeah. So, we have mentioned tensor stack, function stack equal to true and it will write all these data into this current directory, and so from the event files you can see inside the tensor board, ok. And if you have mentioned the targeted files in any elsewhere you can mention the directory path here, ok.

Now, if you are using dlprof, so in our case we are using dlprof. So, the database we have to create in to the current directory which is dlprof underscore dldb. So, our deployment database dot sqlite. So, this sqlite database has been created and it is recommended that you open this using DL profile viewer only, ok. So, because all the necessary encoding and decoding is supporting the dlprofviewer.

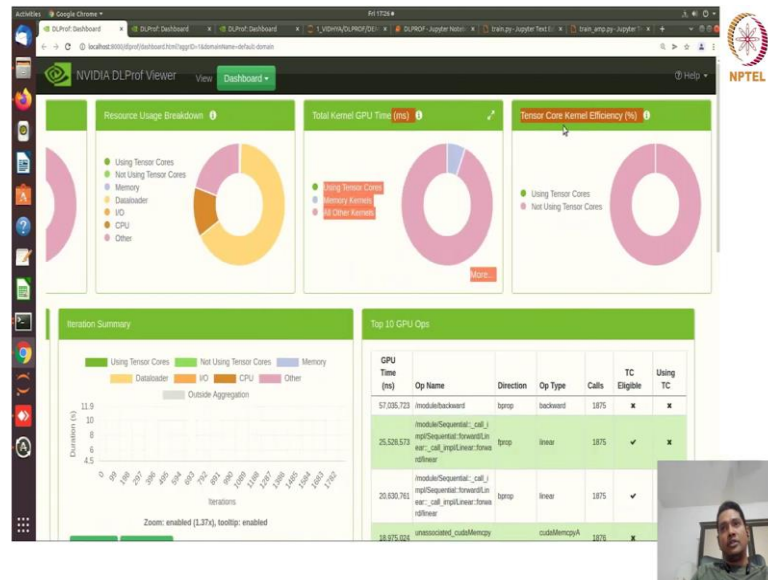
(Refer Slide Time: 24:30)



So, once you open this, open opening your profiler into dlprof, then you will see the window, ok. So, basically, you can see that this is loaded from your DLProf profiler into your local host and now this is the dashboard for your profile.

So, you can see GPU utilization and resource usage breakdown and total kernel time GPUs in milliseconds, and total core kernel efficiency, so how much percentage of core kernel has been used.

(Refer Slide Time: 24:56)



So, now as I was mentioning that these are the usage and let us say let us take this resource usage background breakdown. So, 20 percent resources were been used from elsewhere and data loader is using 65.81 percentage of resources. CPU is using 13.23 and your GPU is very narrow. So, not using tensor cores, because we have not actually used the amp library for enabling the tensor cores, ok.

(Refer Slide Time: 25:45)

The screenshot shows the NVIDIA DLProf Viewer interface. The **Performance Summary** section on the left displays the following metrics:

- Wall Clock Time (ms): 1.44
- Tensor Core Kernel Efficiency (%): 0.0
- CPU Utilization (%): 0.5
- Total Iterations: 1,876
- Profiled Iterations: 1,876
- Start Iteration: 0
- Stop Iteration: 1,876

The **Iteration Summary** section in the center features a bar chart showing the distribution of operations across iterations. The legend includes: Using Tensor Cores (green), Not Using Tensor Cores (light green), Memory (blue), Dataloader (orange), IO (red), CPU (purple), Other (pink), and Outside Aggregation (grey). The y-axis is labeled 'Duration (ns)' and ranges from 4.5 to 11.8. The x-axis is labeled 'Iterations' and ranges from 0 to 1,700. Below the chart, it indicates 'Zoom: enabled (1.37x), tooltip: enabled' and provides 'Reset zoom' and 'Toggle Tooltip' buttons.

The **Top 10 GPU Ops** section on the right lists the following operations:

GPU Time (ns)	Op Name	Direction	Op 1
57,035,723	inplaceBackward	trp	back
25,528,573	inplaceSequential_forwardLin	trp	line
20,630,761	inplaceSequential_forwardLin	trp	line
18,975,014	unassociated_cudaMemory	out	sync
18,510,228	inplaceSequential_forwardLin	trp	line
17,714,649	inplaceSequential_forwardLin	trp	line
17,538,043	inplaceSequential_forwardLin	trp	line

(Refer Slide Time: 25:45)

The screenshot shows the NVIDIA DLProf Viewer interface with the **System Config** and **Expert Systems** sections highlighted.

The **System Config** section displays the following system information:

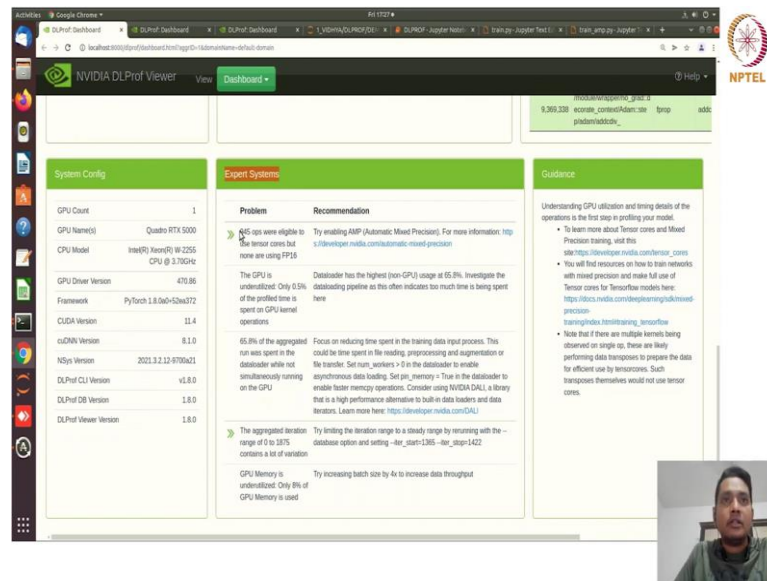
- GPU Count: 1
- GPU Name(s): Quadro RTX 5000
- CPU Model: Intel(R) Xeon(R) W-2555 CPU @ 3.10GHz
- GPU Driver Version: 470.86
- Framework: PyTorch 1.8.0a0-526a372
- CUDA Version: 11.4
- cuDNN Version: 8.1.0
- NSys Version: 2021.3.12-8709a21
- DLProf CLI Version: v1.8.0
- DLProf DB Version: 1.8.0
- DLProf Viewer Version: 1.8.0

The **Expert Systems** section provides diagnostic information:

- Problem:** 945 ops were eligible to use tensor cores but none are using FP16. The CPU is underutilized: Only 0.5% of the profiled time is spent on GPU kernel operations.
- Recommendation:** Try enabling AMP (Automatic Mixed Precision). For more information: <https://developer.nvidia.com/automated-mixed-precision>. The Dataloader has the highest (non-GPU) usage at 65.8%. Investigate the dataloading pipeline as this often indicates too much time is being spent here. Focus on reducing time spent in the training data input process. This could be time spent in the reading, preprocessing and augmentation or file transfer. Set `num_workers > 0` in the dataloader to enable asynchronous data loading. Set `pin_memory = True` in the dataloader to enable faster memory operations. Consider using NVIDIA DALI, a library that is a high performance alternative to built-in data loaders and data iterators. Learn more here: <https://developer.nvidia.com/DALI>.

The **Guidance** section offers advice on GPU utilization and timing details, including links to resources on training networks with mixed precision and Tensor Cores.

(Refer Slide Time: 25:47)



So, if you go to the recommended system, so expert system in the bottom you will see expert system, and here some recommendations you can see now. So, the recommendations are 945 operations were eligible to use tensor core, but are none are using FP16 because you have not used amp library. So, the recommendation is try enabling amp library for more information improvements.

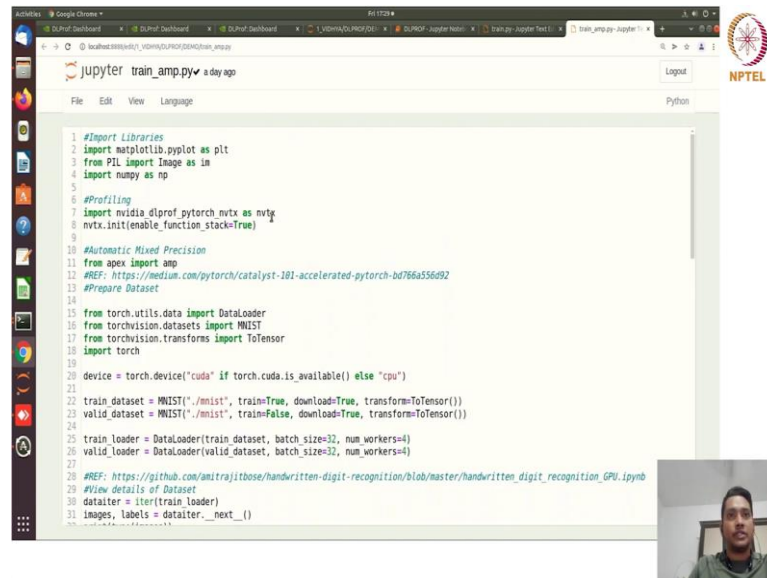
And GPU is underutilized, only 5 percent, 0.5 percent is profiled that is fine. And also, it has recommended that 65.8 percent of your aggregated run was spent in the data loader while not simultaneously running on the GPU, right because it is waiting for the data because you are not using multiple cores for loading the data, right.

So, you can either use num code greater than 0, ok. So, these are the recommendations that you are getting or use NVIDIA DALI which is the data loader library for from NVIDIA itself. So, you can use either of them. So, we will see one of them and to you. So, and also what kind of systems you are using, and what are their versions, and so on and so forth, all the information we keep. The guidance here.

Since, average iteration time is 42 milliseconds, this I mean you cannot see the graph here, but if you run from for more time like duration in seconds you can see. So, if you run in seconds, you will see what are the use of tensor cores, not using tensor cores, how much memory is being used, dataloader, IO and CPU others.

So, this is these are the performance summary you will get. And I want to improve the average iteration time in the end because with some performance tweaking and introducing the amp library, I want to use the tensor cores and in the end I want to use or I want to enhance the average time.

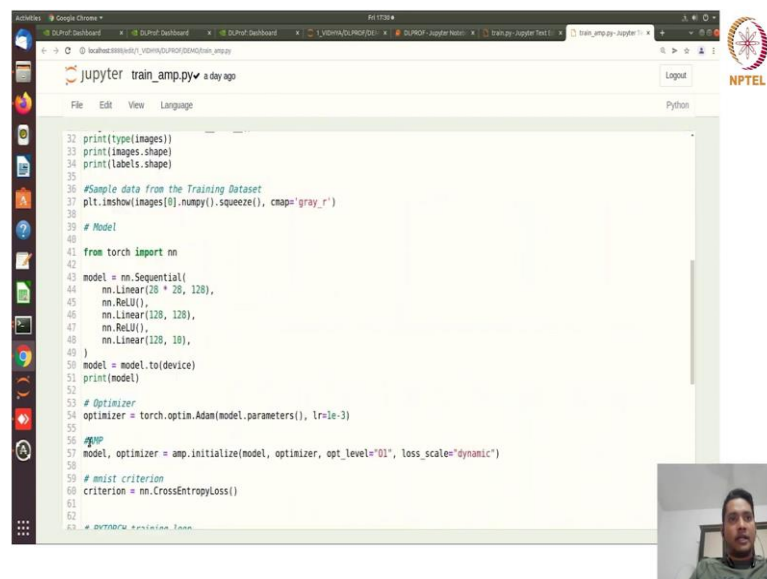
(Refer Slide Time: 27:55)



```
1 #Import Libraries
2 import matplotlib.pyplot as plt
3 from PIL import Image as im
4 import numpy as np
5
6 #Profiling
7 import nvidia_diagnostics_pytorch as nvtx
8 nvtx.init(enable_function_stack=True)
9
10 #Automatic Mixed Precision
11 from apex import amp
12 #REF: https://medium.com/pytorch/catalyst-101-accelerated-pytorch-bd76a556d92
13 #Prepare Dataset
14
15 from torch.utils.data import DataLoader
16 from torchvision.datasets import MNIST
17 from torchvision.transforms import ToTensor
18 import torch
19
20 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
21
22 train_dataset = MNIST("./mnist", train=True, download=True, transform=ToTensor())
23 valid_dataset = MNIST("./mnist", train=False, download=True, transform=ToTensor())
24
25 train_loader = DataLoader(train_dataset, batch_size=32, num_workers=4)
26 valid_loader = DataLoader(valid_dataset, batch_size=32, num_workers=4)
27
28 #REF: https://github.com/amitrajitbose/handwritten-digit-recognition/blob/master/handwritten_digit_recognition_GPU.ipynb
29 #View details of Dataset
30 dataloader = iter(train_loader)
31 images, labels = dataloader.next_()
```

So, next we will use the train_amp.py. So, this training module is using now the amp library which is from the apex. So, automatically expression now we are using from apex, right in media apex. So, you can go to this reference, ok.

(Refer Slide Time: 28:17)

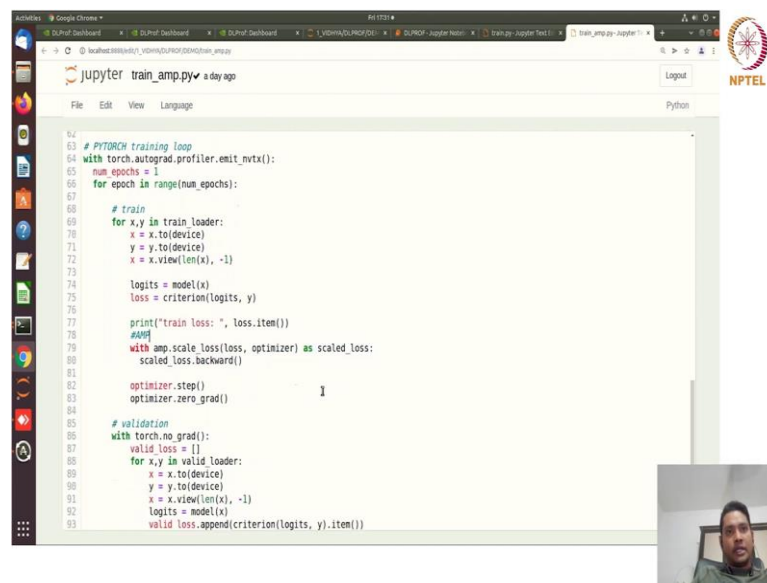


```
32 print(type(images))
33 print(images.shape)
34 print(labels.shape)
35
36 #Sample data from the Training Dataset
37 plt.imshow(images[0].numpy().squeeze(), cmap='gray_r')
38
39 # Model
40
41 from torch import nn
42
43 model = nn.Sequential(
44     nn.Linear(28 * 28, 128),
45     nn.ReLU(),
46     nn.Linear(128, 128),
47     nn.ReLU(),
48     nn.Linear(128, 10),
49 )
50 model = model.to(device)
51 print(model)
52
53 # Optimizer
54 optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
55
56 #AMP
57 model, optimizer = amp.initialize(model, optimizer, opt_level="O1", loss_scale="dynamic")
58
59 # mnist criterion
60 criterion = nn.CrossEntropyLoss()
61
62
63 #optimizer.optimizer.zero_grad_()
```

So, the usual python imports and then defining the device, defining the data loader, train loader and everything is same. Only thing is that you need to define this model and optimizer for your from your amp because now the model should be compatible to use the scaled database, ok and also optimizer. Because optimizer is essentially computing the gradients and then it is being updated with the gradients.

So, optimizer and model initialization with the amp that is important to use amp library. So, we have imported that before just one more time. So, that import was done before.

(Refer Slide Time: 29:05)

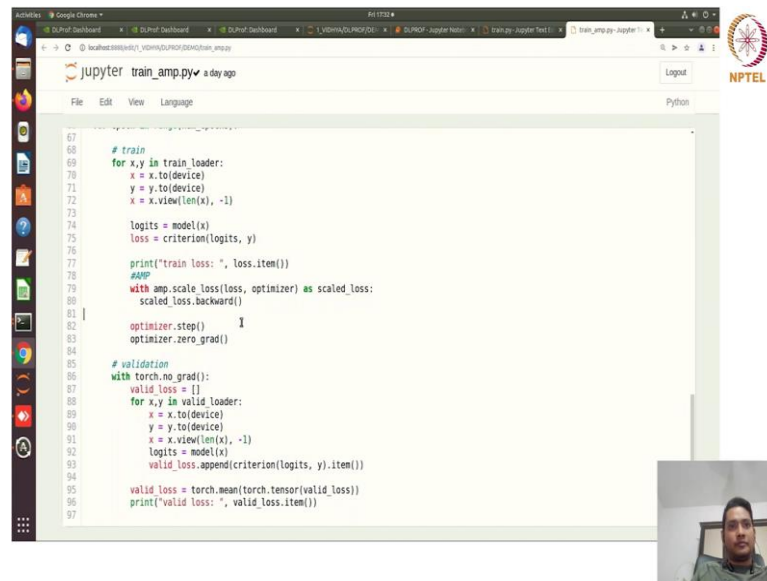


```
63 # PYTORCH training loop
64 with torch.autograd.profiler.emit_nvtx():
65     num_epochs = 1
66     for epoch in range(num_epochs):
67
68         # train
69         for x,y in train_loader:
70             x = x.to(device)
71             y = y.to(device)
72             x = x.view(len(x), -1)
73
74             logits = model(x)
75             loss = criterion(logits, y)
76
77             print("train loss: ", loss.item())
78             #AMP
79             with amp.scale_loss(loss, optimizer) as scaled_loss:
80                 scaled_loss.backward()
81
82             optimizer.step()
83             optimizer.zero_grad()
84
85         # validation
86         with torch.no_grad():
87             valid_loss = []
88             for x,y in valid_loader:
89                 x = x.to(device)
90                 y = y.to(device)
91                 x = x.view(len(x), -1)
92                 logits = model(x)
93                 valid_loss.append(criterion(logits, y).item())
```

And next we will go to the defining of the criterion, the training loop, now this is where we are starting the profiler and then number of epochs we are defining as 1. The same criterion we are keeping to see what kind of improvement we are getting with the profile, ok.

So, x.to(device), so data we are transferring to your device, levels we are transferring to your device, the modules, and it and then we are computing the loss. So, basically, while computing the loss we are now scaling the losses and optimize gradients to your amp library. So, basically, now we are using the mixed precision computer.

(Refer Slide Time: 29:59)



```
67 # train
68 for x,y in train_loader:
69     x = x.to(device)
70     y = y.to(device)
71     x = x.view(len(x), -1)
72
73     logits = model(x)
74     loss = criterion(logits, y)
75
76     print("train loss: ", loss.item())
77
78     #AMP
79     with amp.scale_loss(loss, optimizer) as scaled_loss:
80         scaled_loss.backward()
81
82     optimizer.step()
83     optimizer.zero_grad()
84
85 # validation
86 with torch.no_grad():
87     valid_loss = []
88     for x,y in valid_loader:
89         x = x.to(device)
90         y = y.to(device)
91         x = x.view(len(x), -1)
92         logits = model(x)
93         valid_loss.append(criterion(logits, y).item())
94
95     valid_loss = torch.mean(torch.tensor(valid_loss))
96     print("valid loss: ", valid_loss.item())
97
```

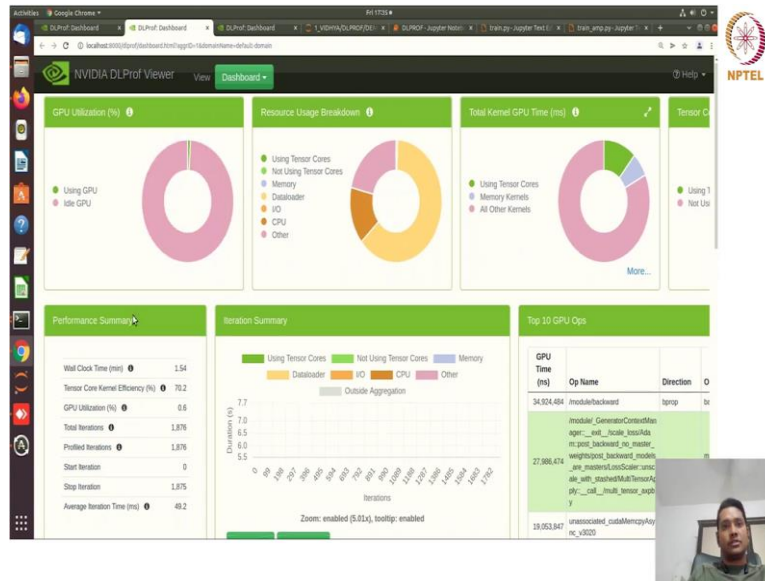
Now, the mixed precision computing uses, so basically if you see here after the scaled loss and optimizer definition, we will just do the backward pass which will compute the gradients and then we will update that and initialize then to 0. So, that is the entire pipeline that we have defined now, ok.

So, this is the entire definition. We just give you one more explanation of this optimization level, ok. So, when we are optimizing, ok there are several data types that we have mentioned, right. So, we mentioned that we can use either FP32, FP16 or mixed up the two. Now, what will be used inside your training that will be defined by this optimization level inside the initializer.

Now, what will be the criterion? I want to have mixed precision computing. So, so there are 4 levels of optimization o 0, o 1, o 2, o 3.

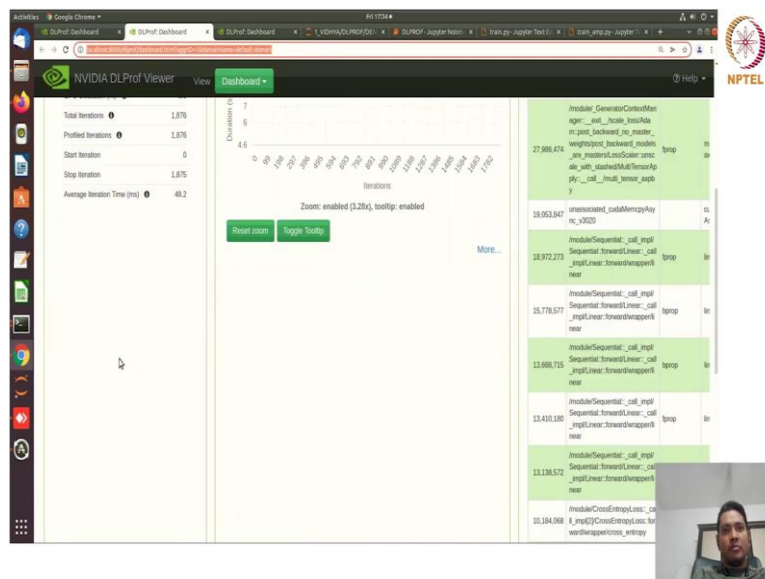
Now, o 0 is for use o 0 and o 3 is without mixed precision. So, o 0 is with FP32 and o 3 is with FP16 and o 1 and o 2 is with FP32 and FP16. And based on their usage in framing you will have two models of mix position, so one and more. So, which will be better for your training that you need to actually analyze by maybe 1 or 2 dots, ok. So, you use o1 first then see the profile, and you use o 2 and see the profile which model of mixed precision computing is helping your case that you can see from the book.

(Refer Slide Time: 32:08)

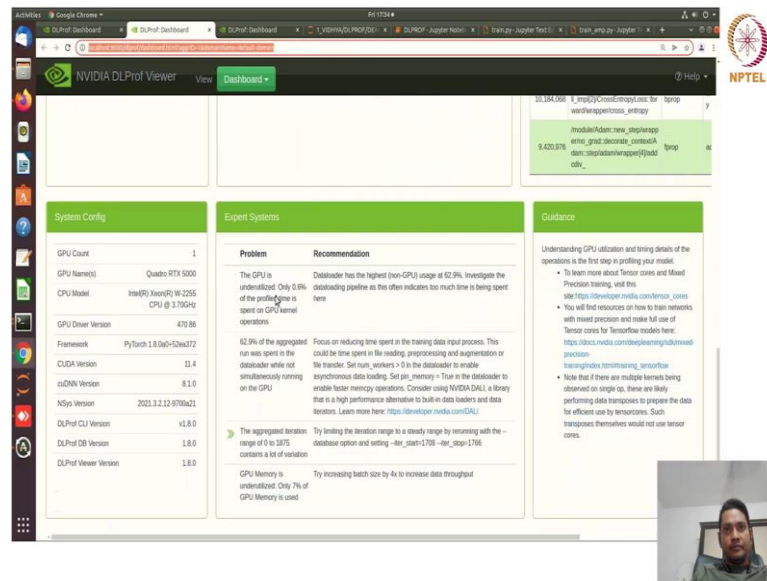


So, that is all about from the profiler, and let us go to the profiles after using the amp, ok.

(Refer Slide Time: 32:15)



(Refer Slide Time: 32:17)

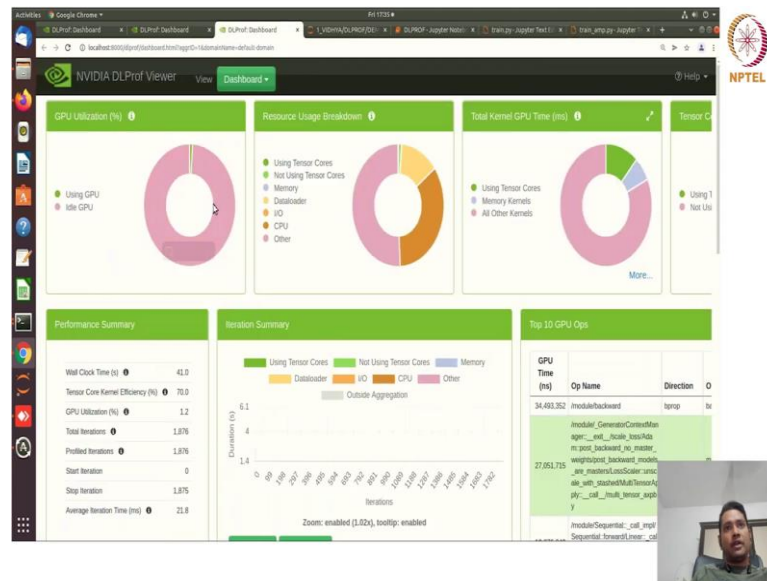


So, after using the amp you can see that go to the recommended system. Now, it is not saying the previous message, right. So, it was showing that tensor cores were not being used, now tensor cores are being utilized and it has increased the utilization from 0.5 to 0.6 percent. So, basically this was just one run, right not multiple runs.

And data loader has the highest usage, still power data loader has highest usage. So, still we have run that with without any workers and that is why you can see here the data loader is using most of the resources and also tensor cores are being used. So, using GPU 1; because we are now using tensor cores using the amp library. So, the usage of GPU utilization has been increased.

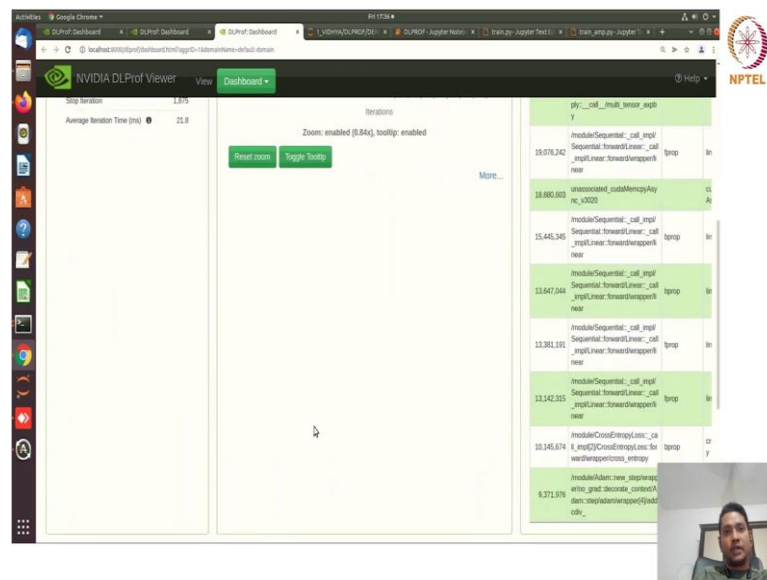
But you can see that average utilization time also have increased and that is only because your data loader, because of this data loader bottleneck you are actually increasing the average iteration time and though you are using amp library. So, after introducing the in inside your train dot train dot underscore amp dot py; so here now after introducing the number of workers equal to 4 in the data loader and valid loader, we will see that so this was the profiler after that.

(Refer Slide Time: 33:52)



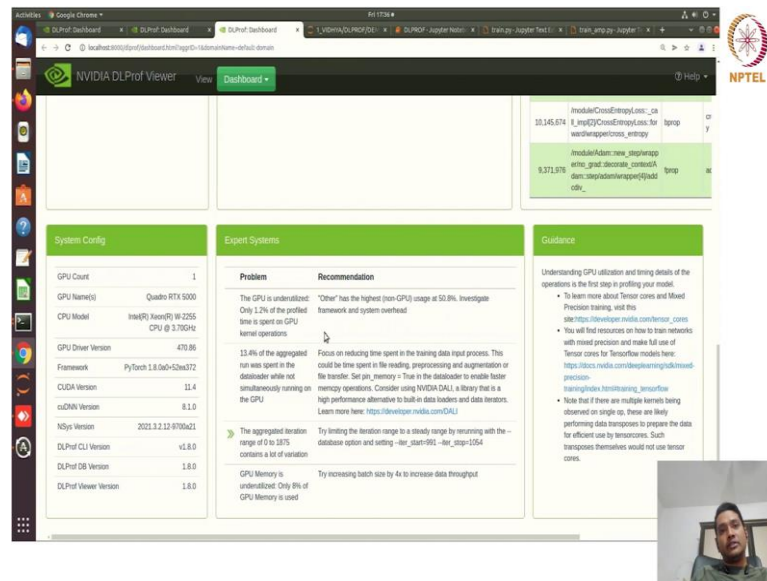
So, now you will see that data loader is now 13 percent only where others and CPU usage has increased, ok.

(Refer Slide Time: 34:03)



And now if you see the average iteration time, it has been decreased I mean almost half. So, just introducing one trick, it has increased the performance by 2 times.

(Refer Slide Time: 34:17)



And you can see that GPU is underutilized, but it has almost doubled the utilization because now it is getting data simultaneously. So, basically 1.2 percent is the double of 0.6 which was before.

But still we have some bottleneck still because this is a very small network. And so, this is how actually you will try to assess, analyze, after visualizing you change your code with some other libraries, what is recommended by your recommended system and you increase the usage of your tensor cores that are available inside your GPU.

So, let us go back to our slide now. So, that is all about from DLProf today.