

**CSYE 7105**

**Parallel Machine Learning and AI (Fall 2020)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Image Classification on Medical Image Dataset:**

**Melanoma Detection**

**Team 9**

**Abhishek Dabas (001302629)**

**Email:** [**dabas.a@northeastern.edu**](mailto:dabas.a@northeastern.edu)

**Table of Contents:**

1. **Introduction………………………………………………………… 3**

* **Background**
* **Objective**

1. **Hardware Specifications for Discovery…………………. 5**

* **Cluster**
* **GPU**

1. **Dataset Specification……………………………………………. 7**
2. **Methodology……………………………………….……………… 7**

* **Architecture**
* **Block Diagram**
* **Resize Image**

1. **Experiments ……………………………………….……………… 11**

* **Experiment 1**
* **Experiment 2**
* **Experiment 3**
* **Experiment 4**

1. **Analysis……………………………………….……………………… 25**
2. **Results………………………………………………………………… 35**
3. **References……………………………………….………………… 36**
4. **Introduction:**
   1. **Background:**

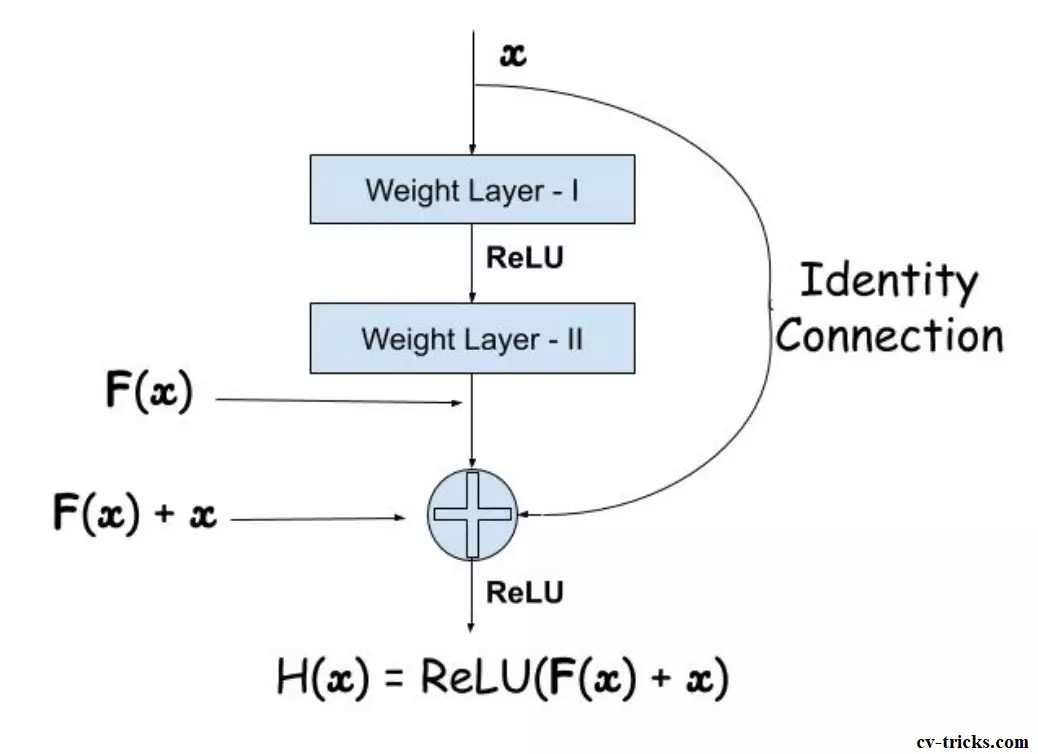
*Object Classification* is a very old and classic problem, and its still a great problem to make models ***Robust***. Deep Learning community had achieved groundbreaking results during the year 2012 when **AlexNet** was introduced to solve the ImageNet classification challenge. With all the Research and Development in Deep learning, AI now plays an important role in Disease Diagnosis and treatment, health Management, drug research and development. Recently Deep mind’s program “*AlphaFold*” outperformed in determining a protein’s 3D shape from amino-acid sequence. This shows how AI can transform the Healthcare Industry. With this motivation I picked up the topic for ***Melanona Classification****.*

As the leading healthcare organization for informatics in medical imaging, the *Society for Imaging Informatics in Medicine (SIIM)*, made the Melanoma dataset public, where one must identify melanoma in an images of skin lesions.

[*Melanoma*](https://en.wikipedia.org/wiki/Melanoma) is a serious type of skin cancer. Specifically, it is responsible for 75% of skin cancer deaths, despite being the least common skin cancer. Melanoma is a deadly disease, but if caught early, most melanomas can be cured with minor surgery. The American Cancer Society estimates over *100,000 new melanoma cases* will be diagnosed in 2020. A better detection of melanoma has the opportunity to positively impact millions of people.

To solve this problem, we will try using *ResNet152*, which is the deepest model of the *ResNet* family, and has the best performance out of all. It is one of the types of Artificial neural network of a kind that builds on constructs known from pyramidal cells in the cerebral cortex.

A *Deep Residual Network* is similar to a network which has convolutions, pooling, activation and fully connected layers stacked upon each other. The difference in a residual network is an identity connection as a curved arrow from the input to the end of residual network.



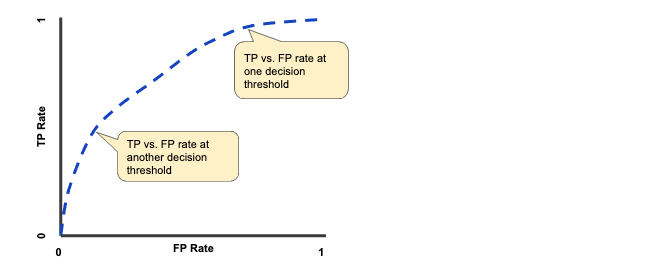
**A Residual Function**: It is the difference between the input and output of the residual block under question. Residual mapping is the value that will be added to the input to approximate the final function (A1, A2, A3, …. An) of the block. You can also assume that the residual mapping is the amount of error which can be added to input so as to reach the final destination i.e. to approximate the final function. It acts as a bridge between the input and the output of the block. We want the layers not to learn to approximate **h(x),** we are letting the layers to approximate a residual function i.e. F(x) = H(x) – x.

There are many variants of ResNet architecture i.e. same concept but with a different number of layers. We have **ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-110, ResNet-152, ResNet-164, ResNet-1202.** We select *ResNet152*becauseit achieves best accuracy amongst *ResNet* family.

* 1. **Objective:**

The objective here is to identify melanoma in images of skin lesions. In particular, we need to use images within the same patient and determine which are likely to represent a melanoma. In other words, we need to create a model which should predict the probability whether the lesion in the image is malignant or benign. The value 0 denotes benign, and 1 indicates malignant.

**The metric used:** An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives.



Modes of Parallelism:Neural networks are inherently parallel algorithms. By running on GPU or Multi GPU we can take advantage of this parallelism.

1. **Hardware Specifications for Discovery**

Discovery is a high-performance computing (HPC) resource for the Northeastern University research community

**2.1 Cluster**

**Name:** Discovery High-Performance Computing Cluster

**Reservation Name**: CSYE7105-gpu

***Commands:***

``srun -p gpu --gres=gpu:1 --pty /bin/bash``

``srun -p reservation --reservation=csye7105-gpu --gres=gpu:1  --mem=100G  --pty /bin/bash``

``srun -p reservation --reservation=csye7105-gpu --gres=gpu:4  --mem=120G  --pty /bin/bash``

* 1. **GPU**

**GPU Name:** Tesla V100-SXM2

**Memory per GPU:** 32510Mib

**Number of Nodes/GPU**: 24 nodes with 4 GPUs each

**CPU Type**: Intel Gold [6132@2.60Ghz](mailto:6132@2.60Ghz)

**RAM per Node:** 187GB

**GPU Architecture:** NVIDIA Volta

**NVIDIA CUDA Cores**: 5,120

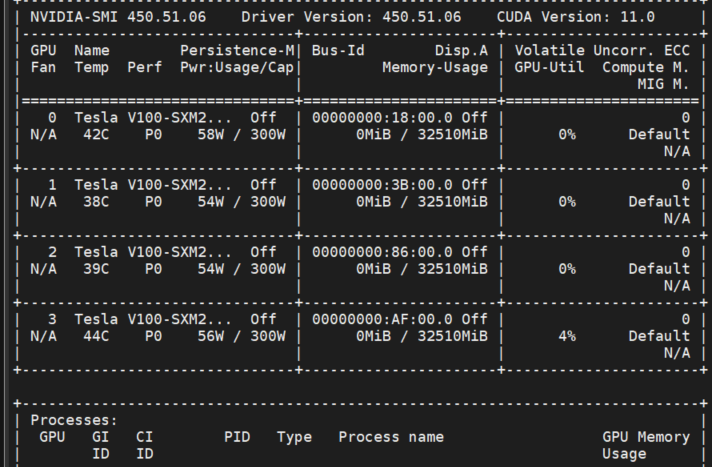
Tesla V100 is the flagship product of Tesla data center computing platform for deep learning, HPC, and graphics. Tesla V100 offers performance upto 100 CPU’s, enabling the scientist and researchers to takle challenges that were once impossible.

* Equipped with 640 Tensor Cores, Tesla V100 delivers 125 teraFLOPS of deep learning performance
* Tesla V100 runs at peak processing efficiency, providing up to 80% of the performance at half the power consumption.

**NVIDIA System Management Interface** (nvidia-smi) is command line utility based on top of nvidia management library (NVML) intended to aid the management and monitoring of Nvidia GPU devices.

**Command** 🡪

``nvidia-smi`` - check the details about the current nvidia gpu



* *Transfering the local cuda env to jupyter notebook*

``python3 -m ipykernel install --user --name=name\_of\_the\_env``

* Conda activate environment
* ``conda create -n myenv python=3.7``;
* ``Conda Activate myenv``

1. **Dataset Specification:**

The images are in **JPEG** format.

What am I predicting **:** I am predicting a binary target for each image. In Target values, **0** denotes **benign**, and **1** indicates **malignant**.

The images are stored on the “Scratch” directory on discovery.

* **Size of the Image folder:**
  + **23.9 GB**
* **No of Images we have:**
  + **Train data**: 33126 Images
  + **Test data:** 10982 Images
* “train.csv” **-** the training dataset
  + **Columns:**

*image\_name* - unique identifier, points to filename of related DICOM image

*patient\_id* - unique patient identifier

sex - the sex of the patient (when unknown, will be blank)

*age\_approx* - approximate patient age at time of imaging

*anatom\_site\_general\_challenge* - location of imaged site

*diagnosis* - detailed diagnosis information (train only)

*benign\_malignant* - indicator of malignancy of imaged lesion

*target* - binarized version of the target variable

1. **Methodology:**

**ResNet152 :** To solve the problem of vanishing/exploding gradients, a skip / shortcut connection is added to add the input x to the output after few weight layers. Hence, the output

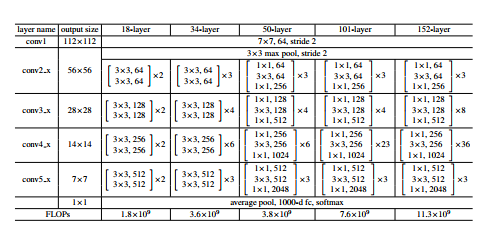
H(x)= F(x) + x. The weight layers actually is to learn a kind of residual mapping: F(x)=H(x)-x.

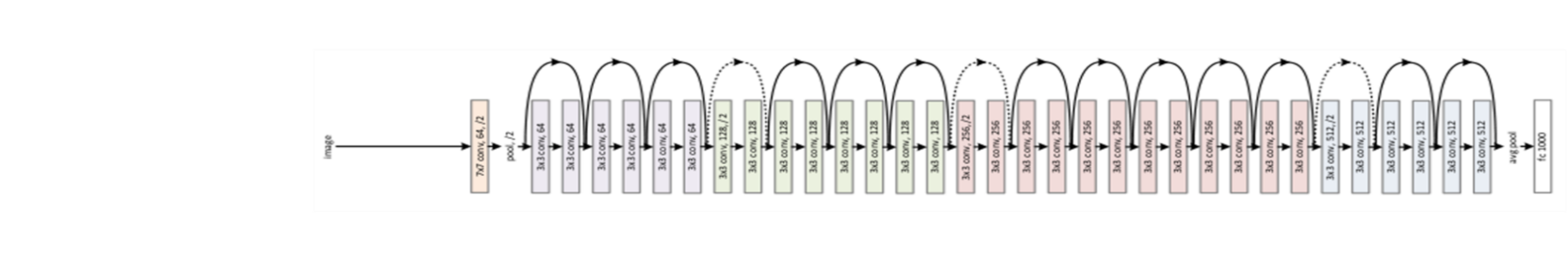
**Total parameters:** 30.2 Million

**Top accuracy:** 95.51%

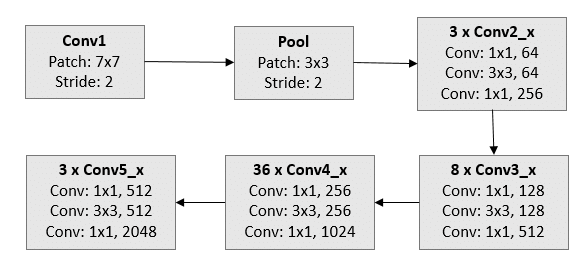
**FLOPS:** 11B

* 1. **Architecture**





* 1. **Block Diagram**



Resnet152: For one image, we extract a 2048-

dimensional feature from the last fully-pooling layer

Resnet152: For one image, we extract a 2048-

dimensional feature from the last fully-pooling layer

Resnet152: For one image, we extract a 2048-

dimensional feature from the last fully-pooling layer

As we can see the architecture above, there are some important points to be mentioned.

* ResNet architecture performs the initial convolution and max pooling using 7×7 and 3×3 kernel sizes respectively
* Stage 1 of the network starts and it has 3 Residual blocks containing 3 layers each. The size of kernels used to perform the convolution operation in all 3 layers of the block of stage 1 are 64, 64 and 256 respectively
* The dashed connection arrow refers to the convolutional operation, performed with a stride of 2, which reduced the size of the input.
* we extract 2048- dimensional features from the last fully- pooling layer (Conv5x layer)
* The network has an average Pooling layer followed by fully connected layer, which then outputs the number of classes we want to classify.
  1. **Resize the Image**

The network can take the input image having height, width as multiples of 32 and 3 as channel width. So we will have to convert the size of the image to appropriate size which could be (224 x 224 x 3) or (512 x 512 x 3).

**Original Image size** = (4000\*6000)

**Original Image size** = (4000\*6000\*3)

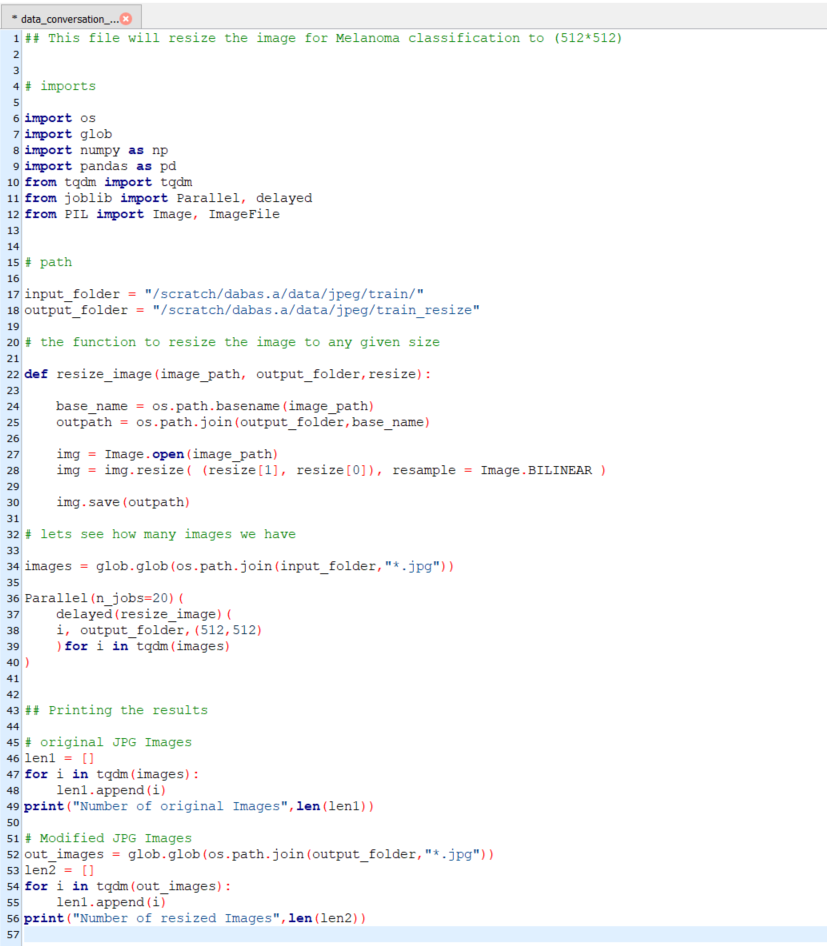
We will now have to resize the shape of the image to required shape.

**Resized Image** = (512\*512)

**Resized Image** = (512\*512\*3)

**Procedure***: “Joblib”* provides a simple helper class to write parallel for loops using multiprocessing. The core idea is to write the code to be executed as a generator expression, and convert it to parallel computing. We will be running the job to resize the image in Parallel using this “**Joblib**” Library

* **Parallel(n\_jobs=20)**



1. **Experiments:**

The key to train large scale CNN models with multiple GPUs is how to divide tasks between different GPUs. We will be using PyTorch to access multiple GPU’s and then run our model. With PyTorch, we can use package “torch.cuda” which supports CUDA tensor types, which implements the same function as CPU tensors but utilizing GPU’s for computations.

We can run commands like :

import torch

**# current device selected**

torch.cuda.current\_device()

**# total device count**

torch.cuda.device\_count()

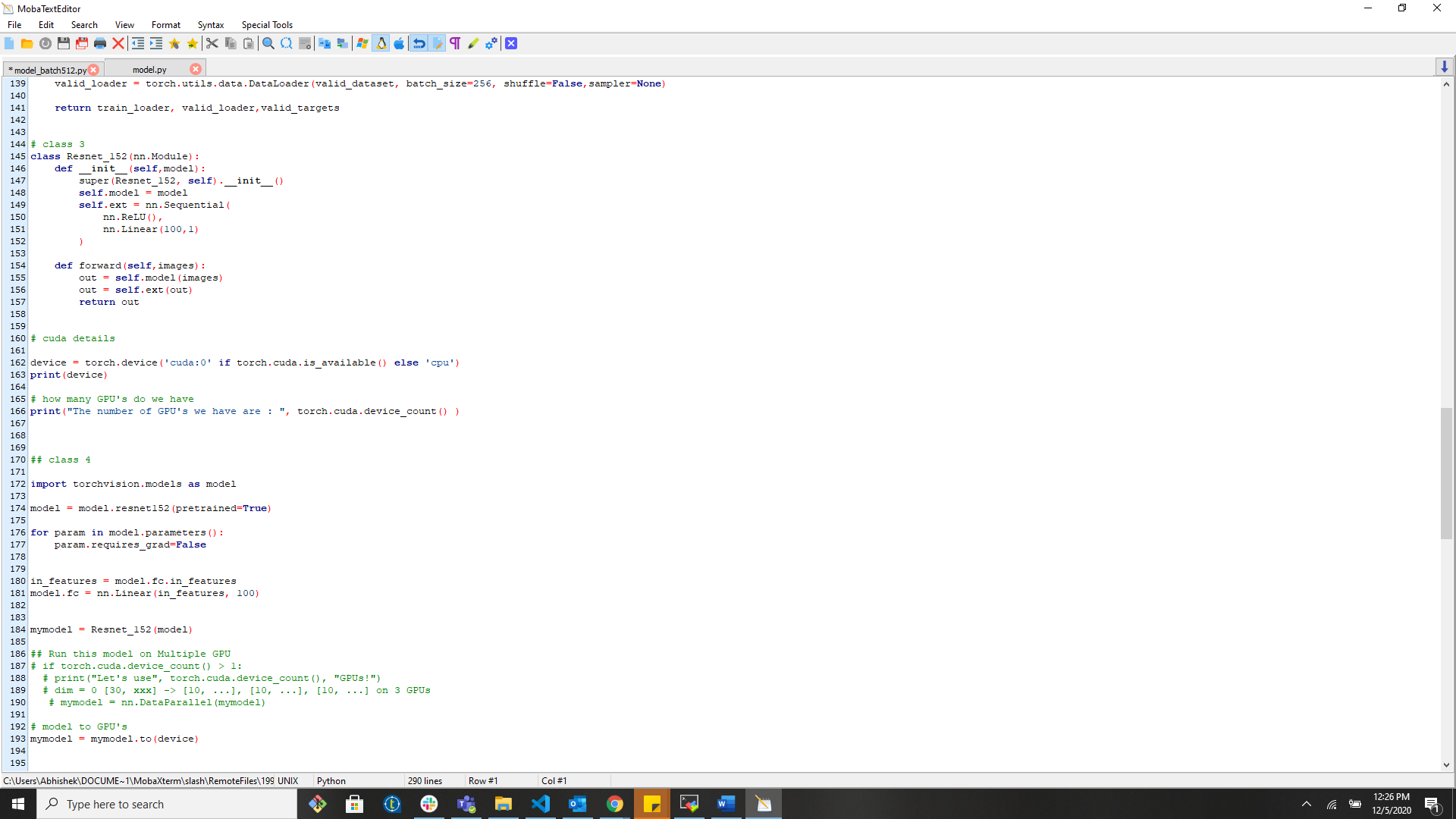
**# device name of 0 device**

torch.cuda.get\_device\_name(0)

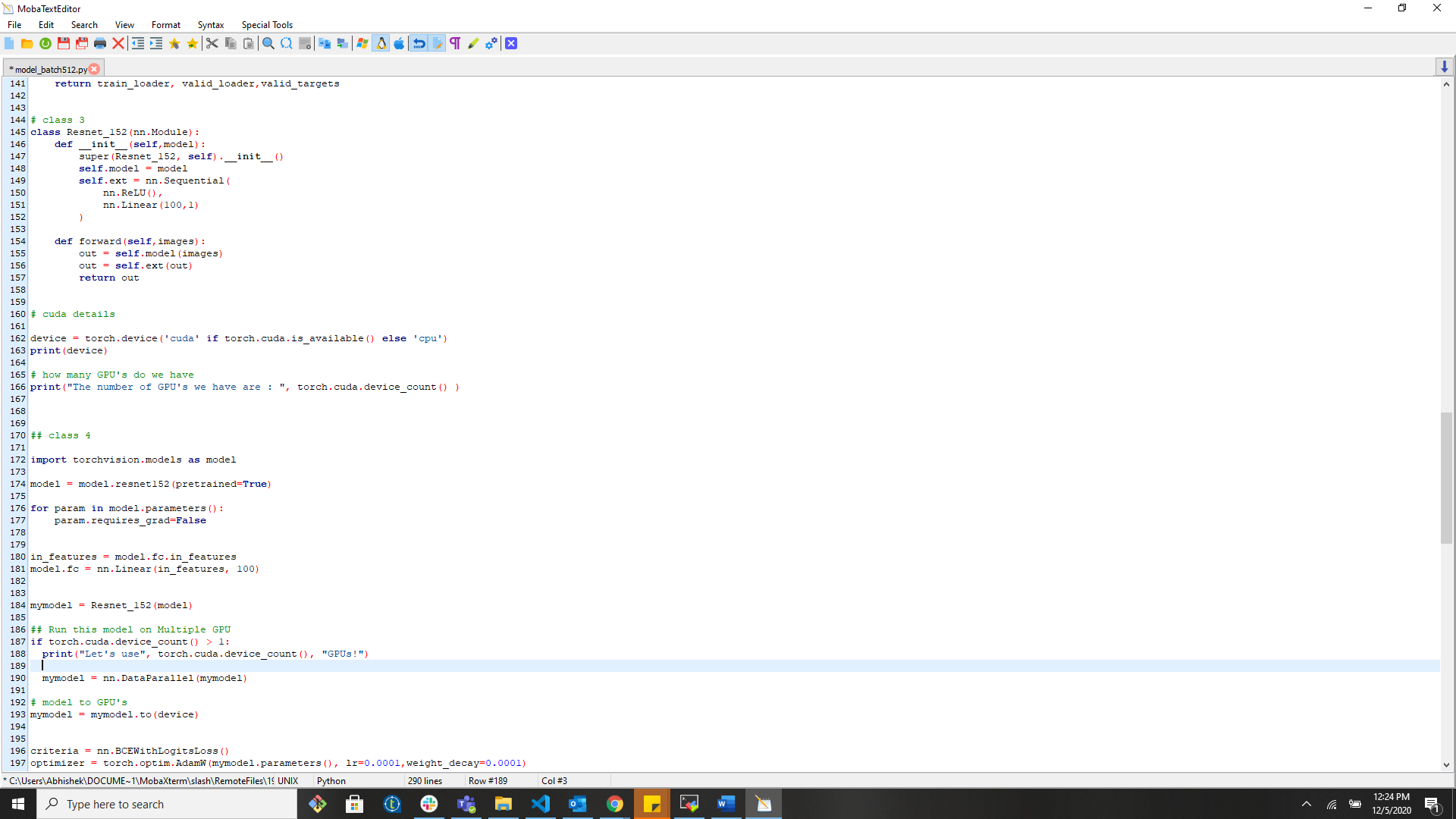
**# if gpu is available**

torch.cuda.is\_available()

Single GPU:



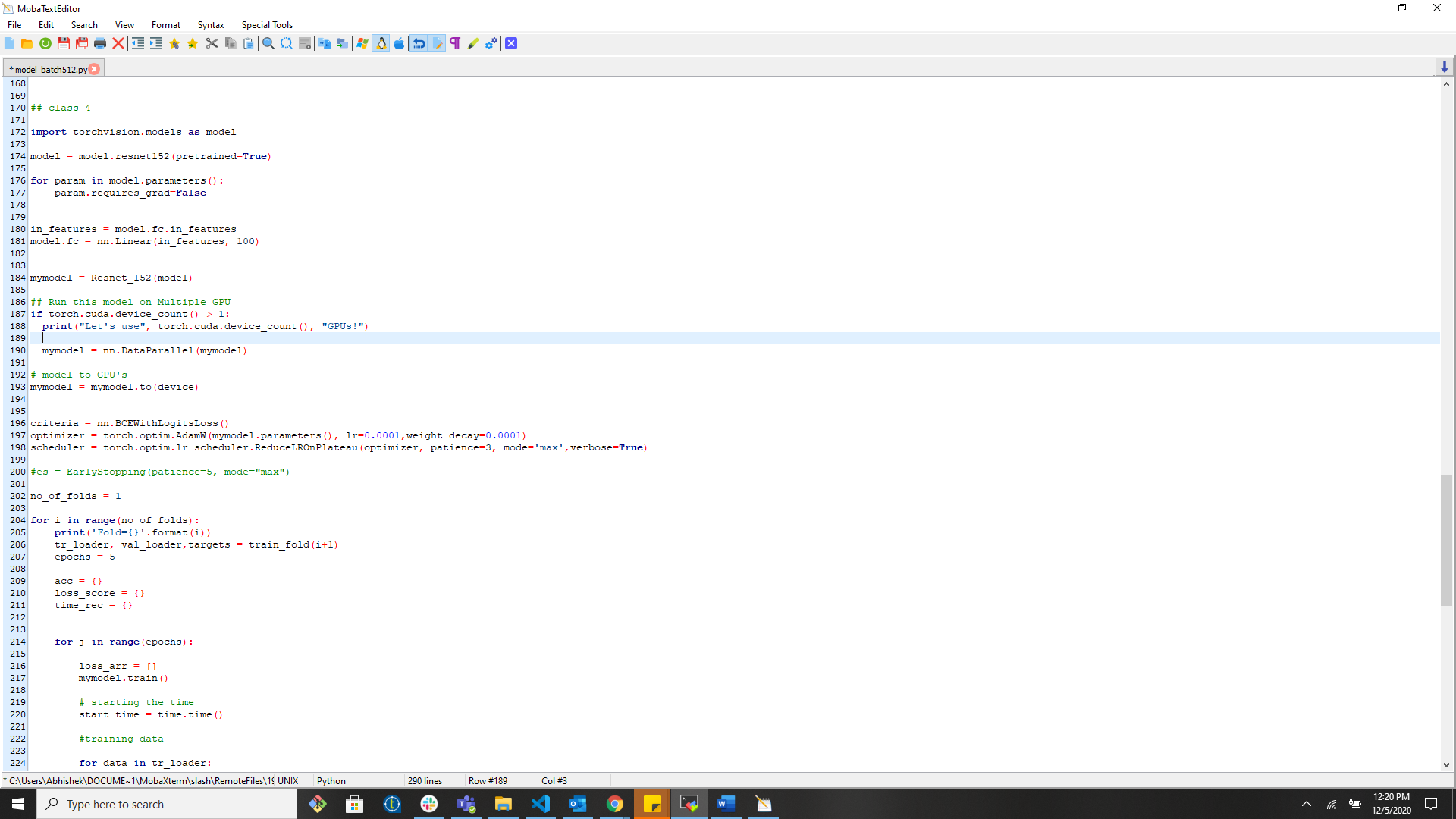
Multiple GPU’s



* **Data Parallelism:**

Data parallelism is a way of performing parallel execution of an application on multiple processors. In data parallel, there is no synchronization between GPUs in forward computing, because each GPU has a fully copy of the model, including the deep net structure and parameters. But the parameter gradients computed from different GPUs must be synchronized in BP. In data parallel, there is no synchronization between GPUs in forward computing, because each GPU has a fully copy of the model, including the deep net structure and parameters. But the parameter gradients computed from different GPUs must be synchronized in BP. Data Parallel splits your data automatically and sends job orders to multiple models on several GPUs. After each model finishes their job, DataParallel collects and merges the results before returning it to you. PyTorch “nn.parallel” primitive can be used independently. It’s natural to execute your forward, backward propagations on multiple GPUs. However, Pytorch will only use one GPU by default. You can easily run your operations on multiple GPUs by making your model run parallelly using “DataParallel”

**So it “replicated” a module on multiple devices, distributes the input in the first dimension.** Gathers and concatenates the input in the first dimension. This basically applied already distributed inputs to a set of already distributed models.



* **We will be performing 4 experiments**

Experiment 1: Multiple GPU’s

Experiment 2: Multiple EPOCHS

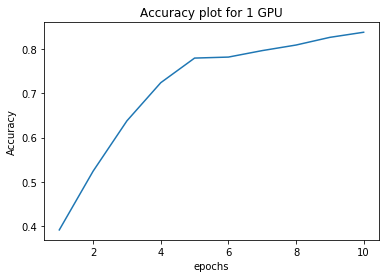
Experiment 3: Multiple Batch SIZE

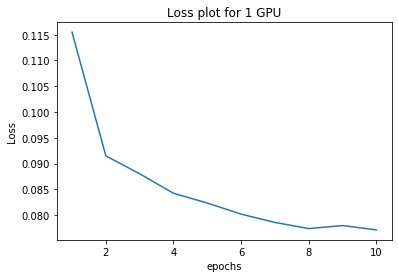
Experiment 4: GPU and CPU

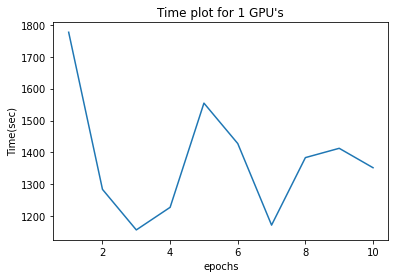
* **Experiment 1**

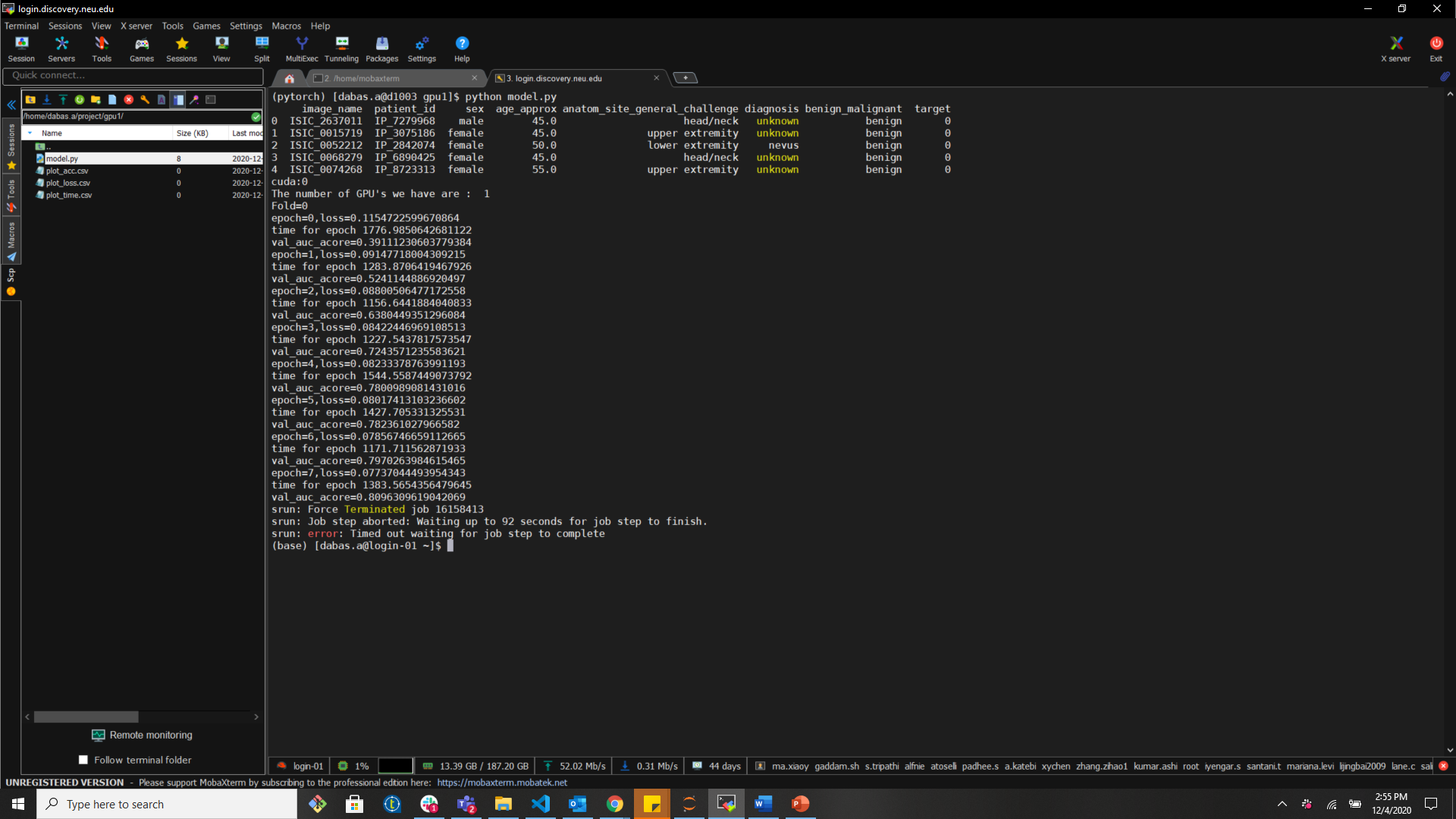
In this experiment we will testing the ResNet152 model with multiple GPU’s and performing Data Parallelism when the GPU count>1

* EPOCH: 10
* Batch size: 256
  1. **With GPU’s #1 and Data Parallelism: No**

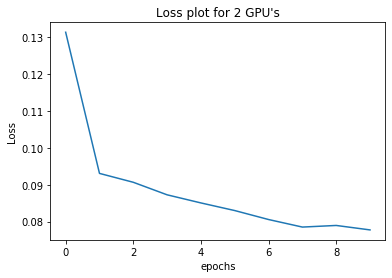
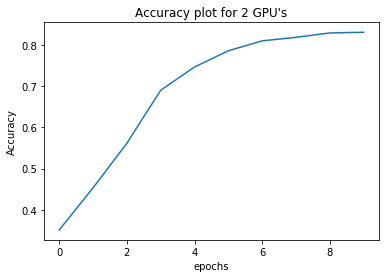
****

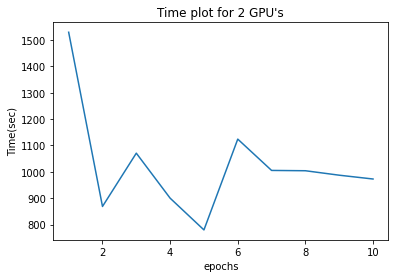
****

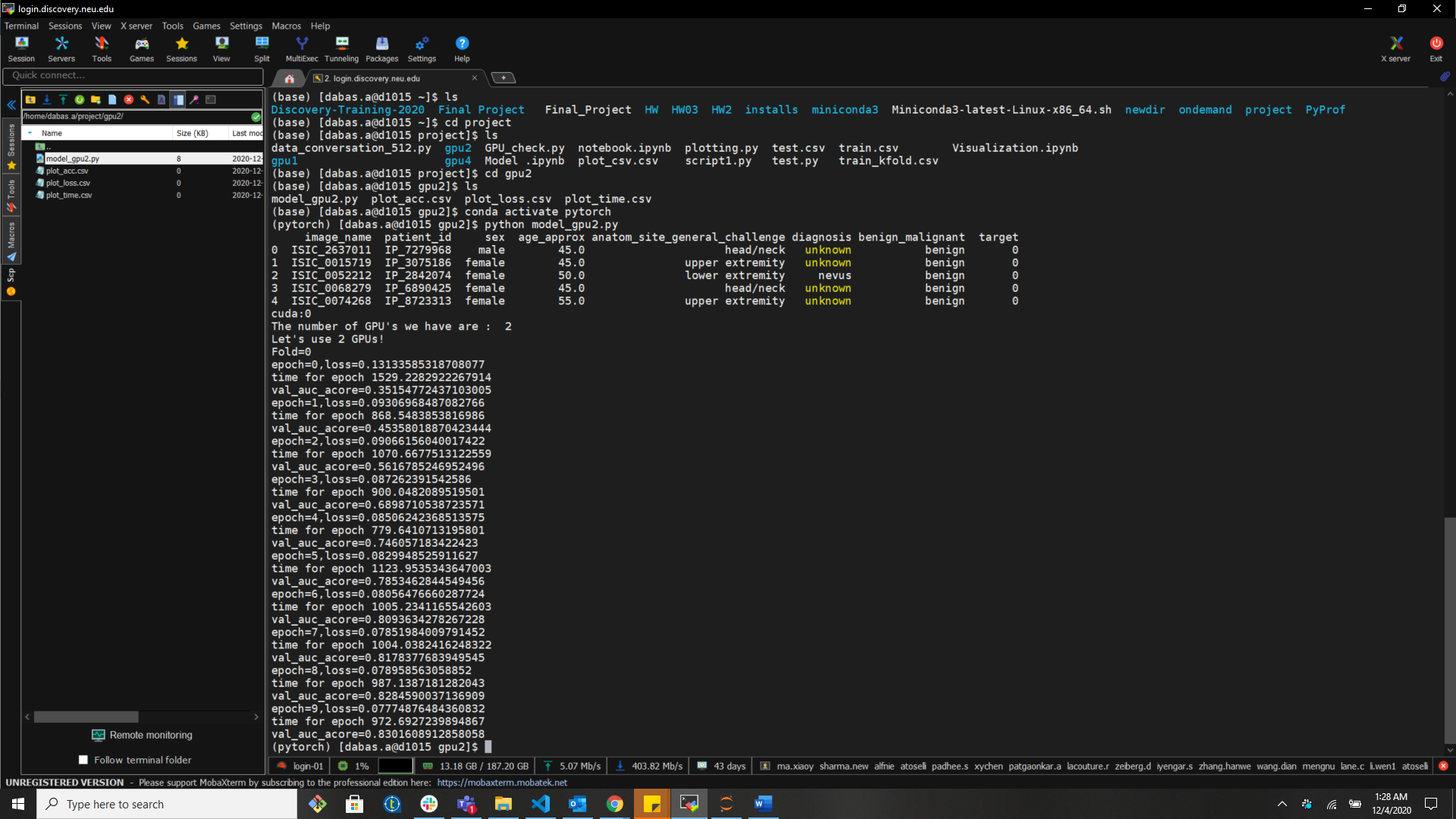
****



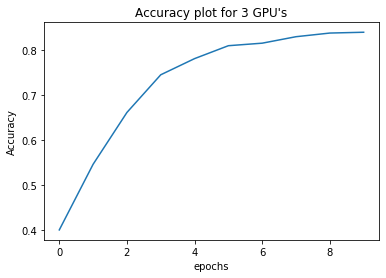
* 1. **With GPU’s #2 and Data Parallelism: Yes**

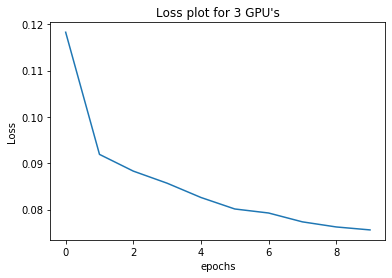
****

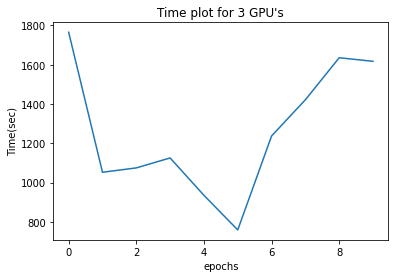
****

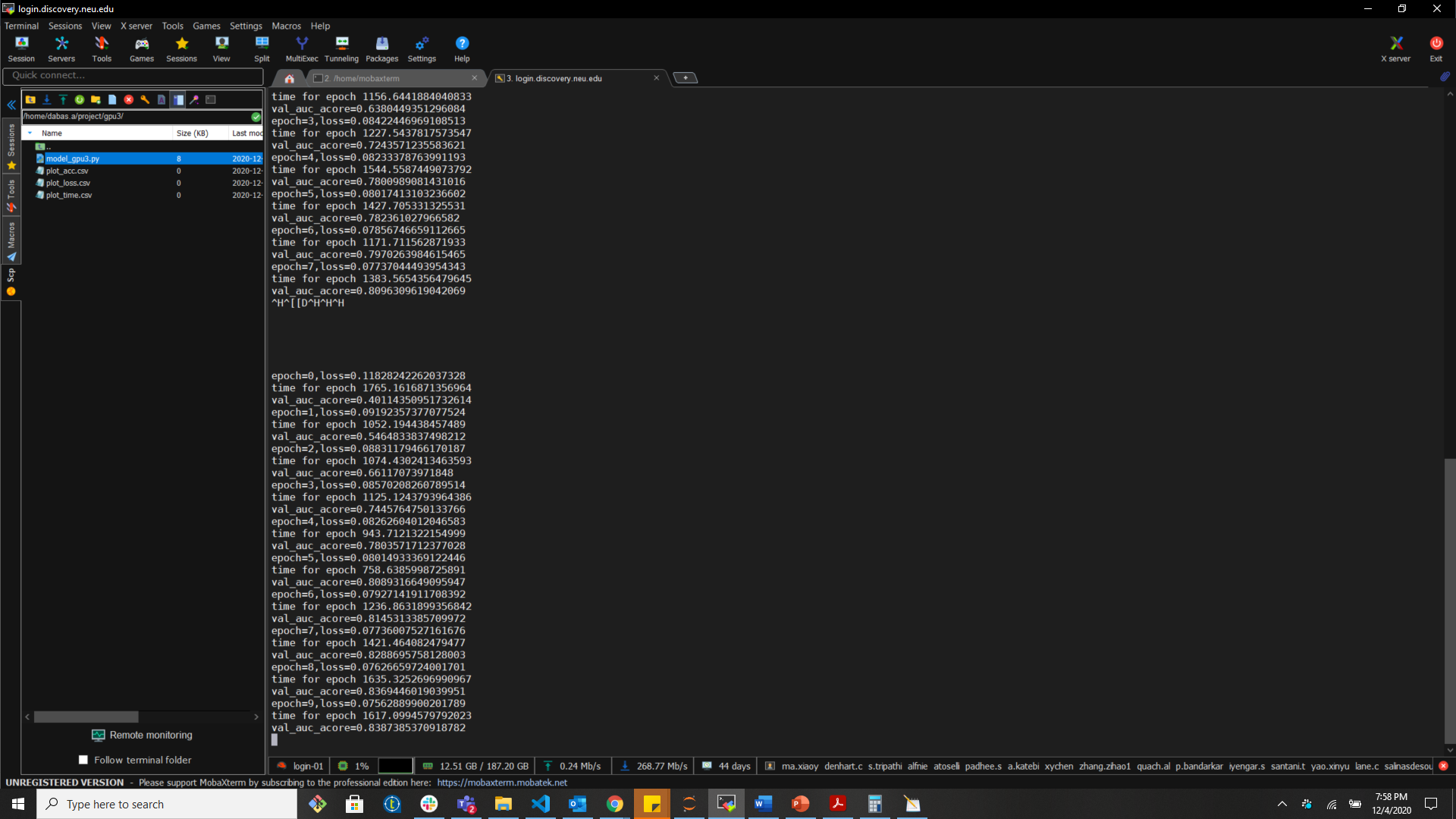


* 1. **With GPU’s #3 and Data Parallelism: Yes**

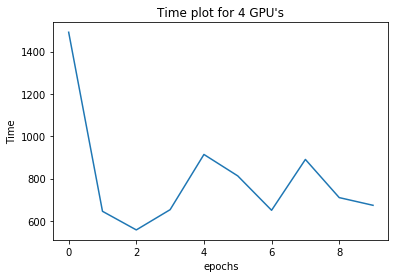
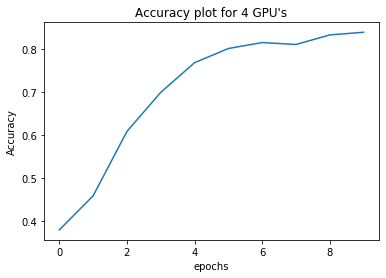
****

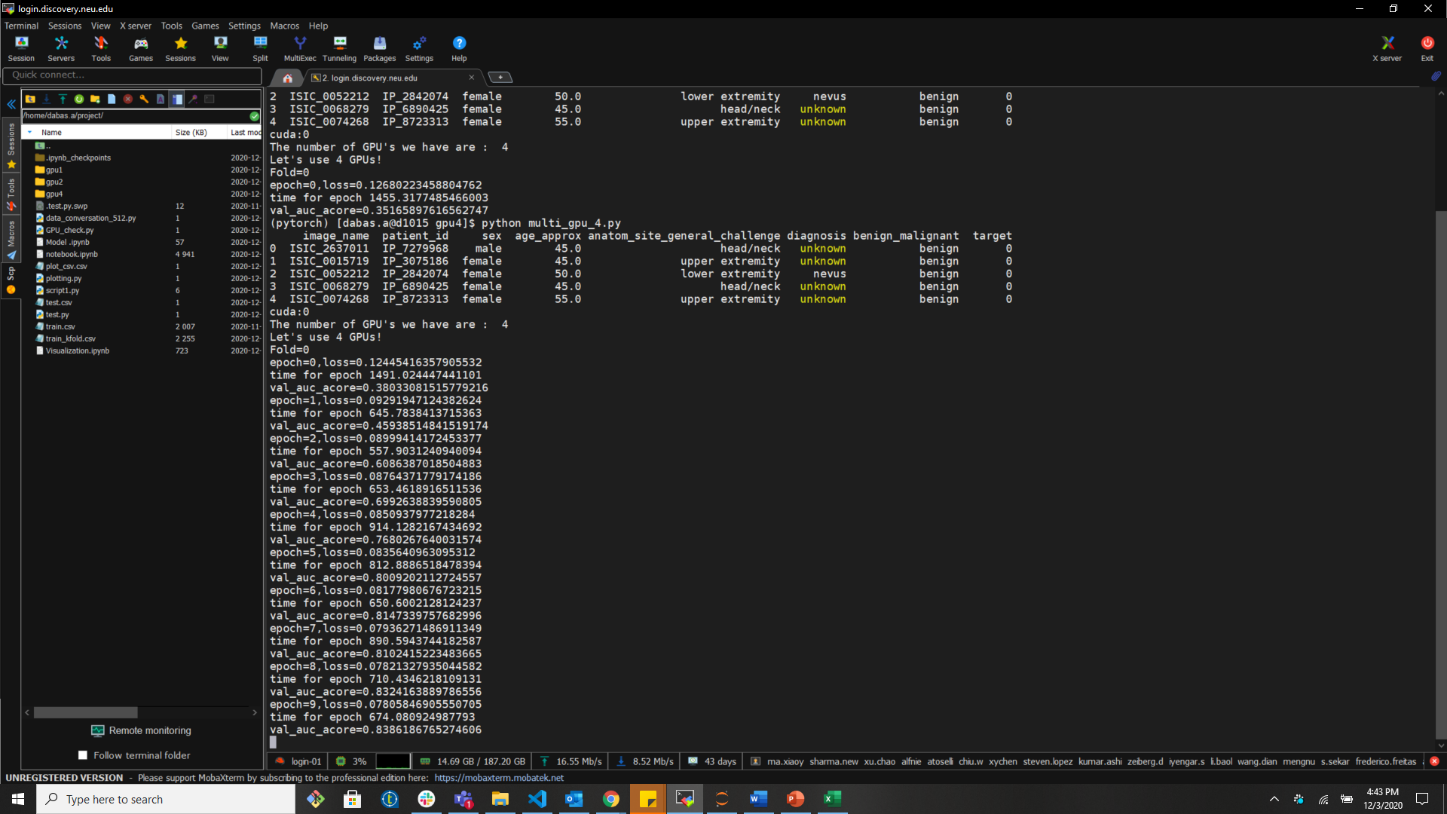
****

****



* **We can observe the results after running the script file above.** 
  1. **With GPU’s #4 and Data Parallelism: Yes**

****



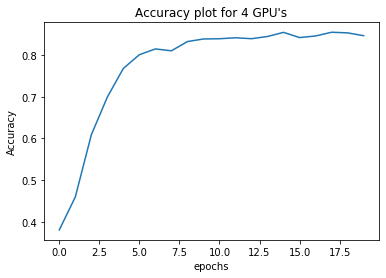
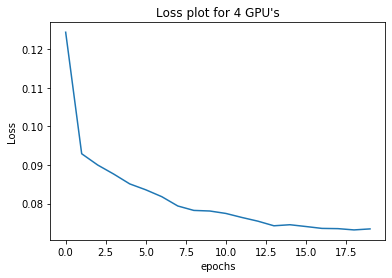
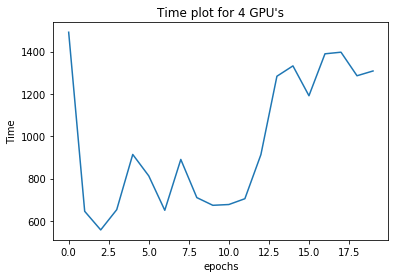
**• We can observe the results after running the script file above.**

* **Experiment 2**

In this experiment we will be running the same ResNet152 Model and increasing the number of epochs and with same batch size.

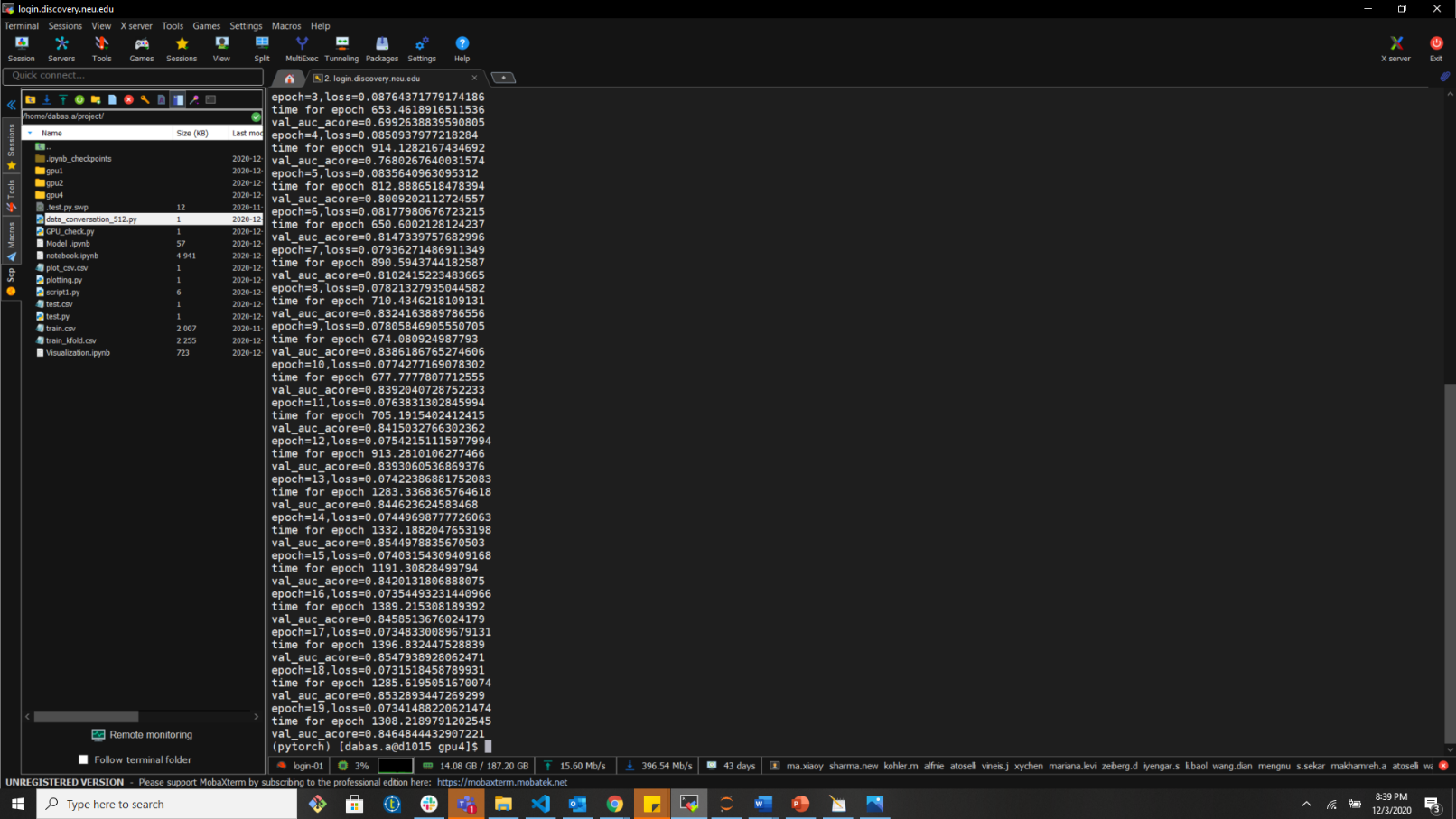
* **EPOCH = 20**
* **Batch Size = 256**

**5.5 With GPU #4 and Model Parallelism: Yes**

**  **

* **AVG Time per Epoch: 974.193**

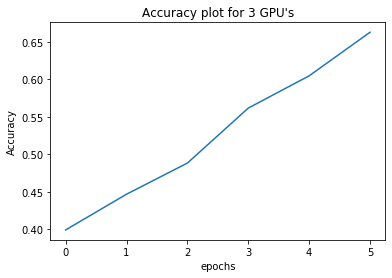
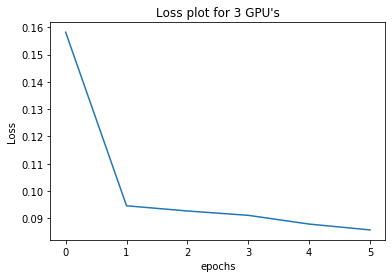
**• We can observe the results after running the script file above.**



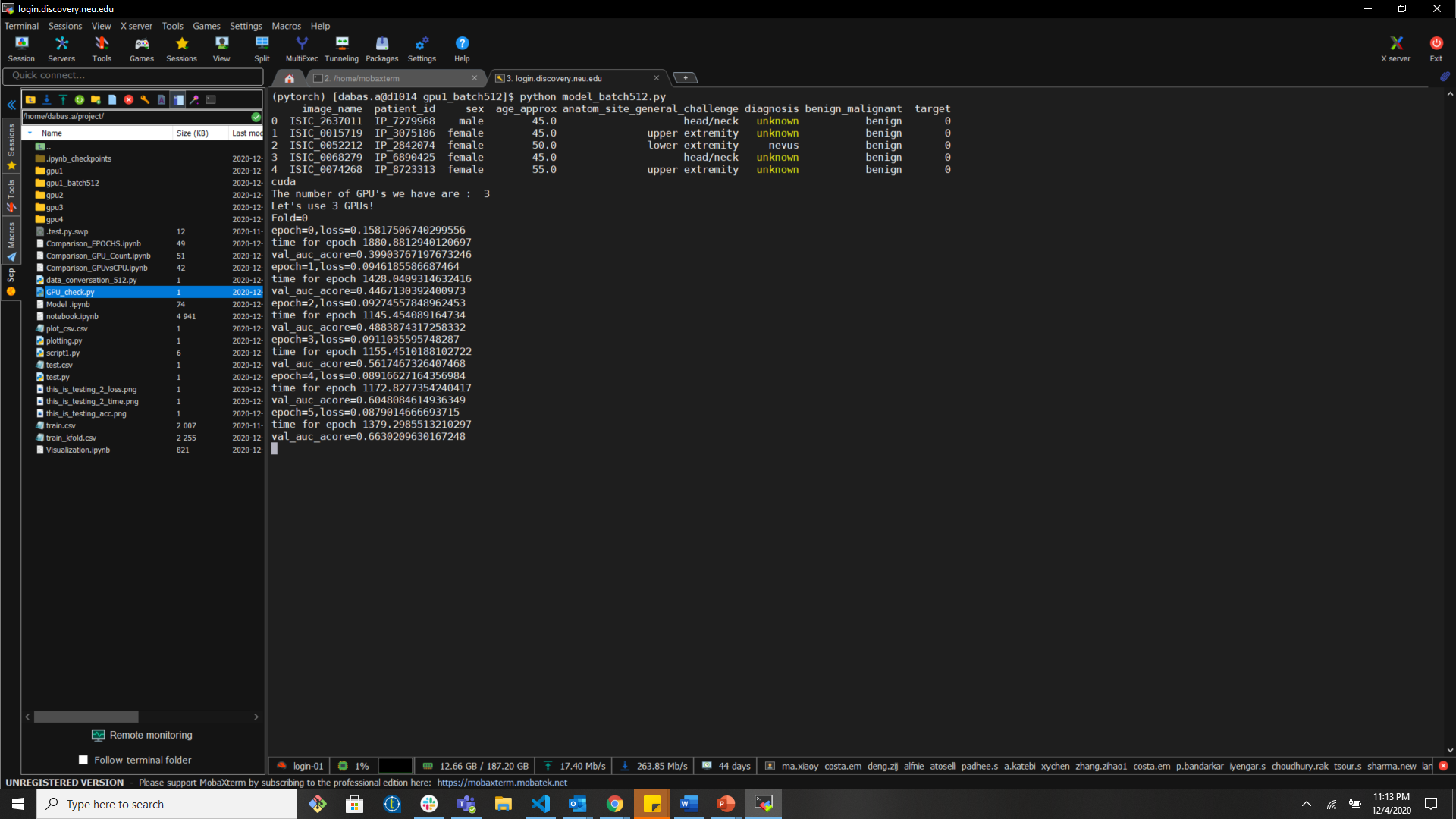
* **Experiment 3**

In this experiment we will be running the model on a different batch size to compare the performance of the GPU with different batch sizes.

* + EPOCH= 10
  + Batch size= 128,256,512
  1. **With GPU #3 and Model Parallelism: Yes, Batch size 512**

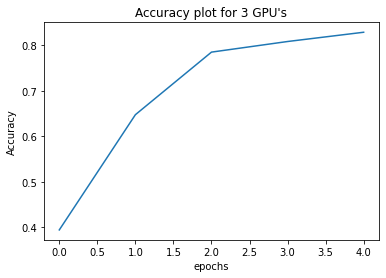
**  **

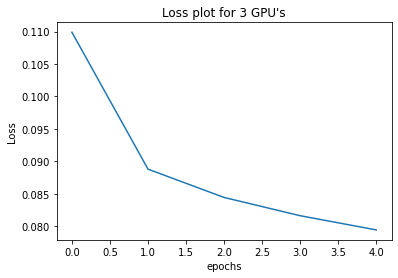
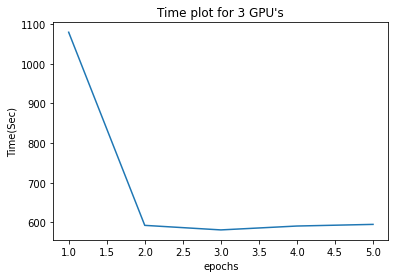
* **Avg Time: 1360.3255**

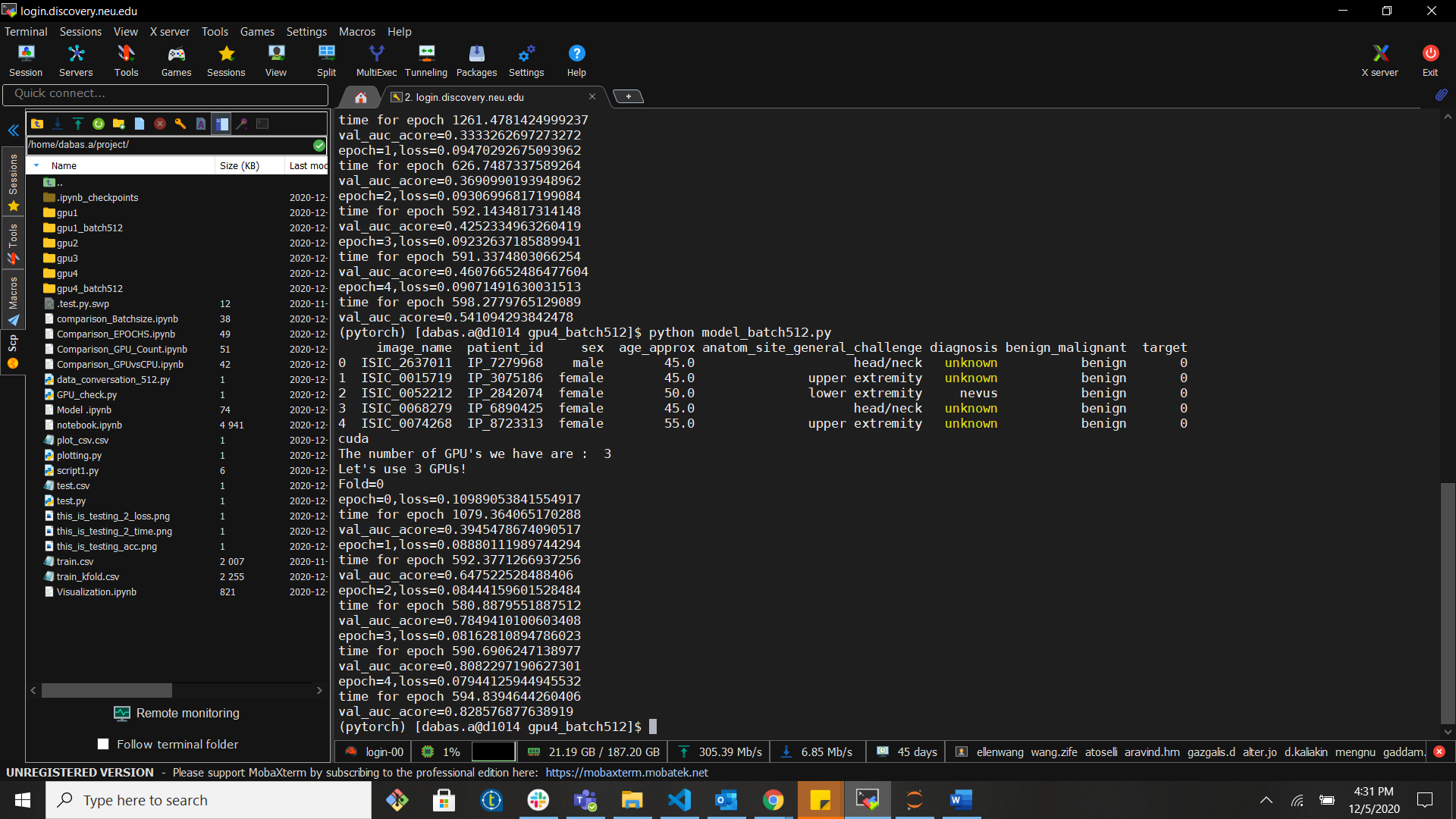


**• We can observe the results after running the script file above.**

* 1. **With GPU #3 and Model Parallelism: Yes, Batch size 128**

****

** **



* **Experiment 4**

In this experiment I am running the model on CPU and running it for 1 EPOCH, just to test and compare the performance of running the model on CPU vs GPU.

* + CPU
  + EPOCH =2
  + Batch Size= 256

**5.7 With CPU and Model Parallelism: No**

In Discovery Cluster we had High power CPUs as well. Its important to know and understand the difference between the GPU and CPU. Hence, we will run the script on CPU as well jus to analyze the difference between the CPU and the GPU.

**CPU**:

loss = 0.384948913540159

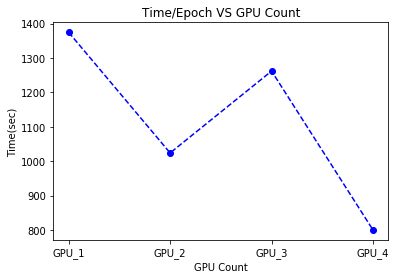
the exec time = 5279.649239778519

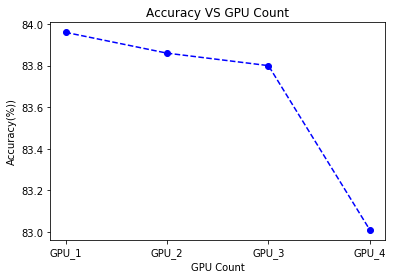
val\_auc\_acore = 0.279610829103214

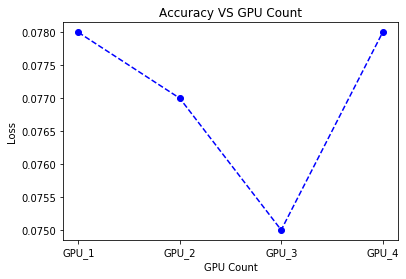
1. **Analysis:**

* **Analysis 1 - GPU Count vs Performance**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **GPU (count)** | **EPOCH** | **Model (ResNet152)** | **Execution time** | **Data Parallelism** | **Accuracy** | **Loss** |
| **1** | **10** | **Yes** | **1374.71** | **No** | **0.8395** | **0.078** |
| **2** | **10** | **Yes** | **1024.11** | **Yes** | **0.8386** | **0.077** |
| **3** | **10** | **Yes** | **1262.101** | **Yes** | **0.83873** | **0.075** |
| **4** | **10** | **Yes** | **800.09** | **Yes** | **0.8301** | **0.078** |

****

****

****

**Observation:**

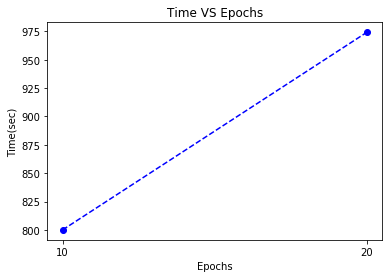
We can see that the time decreases drastically, when we shift from one GPU to another. Whereas there is not much change in the accuracy and the loss. The accuracy difference is approx. 0.8 % and difference in the loss is approx. 0.10%.

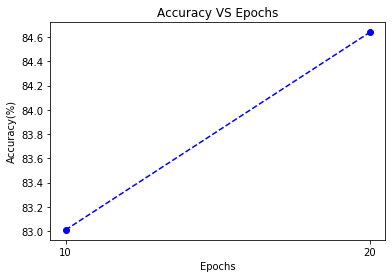
**Conclusion**:

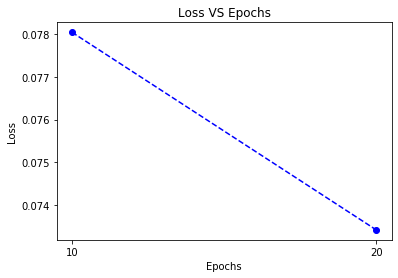
1. Increasing the number of GPU’S doesn’t necessarily increases the performance (accuracy).
2. Using more GPU’s speeds up the computations and saves time.
3. Adding more GPU’s after a certain point does not increase performance and doesn’t decrease the performance time as well.

* **Analysis 2 – (EPOCHS vs Performance)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **GPU Count** | **EPOCHS** | **Model Parallelism** | **TIME** | **Acc** | **Loss** |
| **4** | **10** | **yes** | **800.0900** | **0.8301** | **0.78** |
| **4** | **20** | **yes** | **974.19** | **0.8464** | **0.07341** |

****

****

****

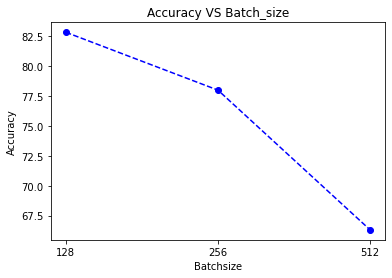
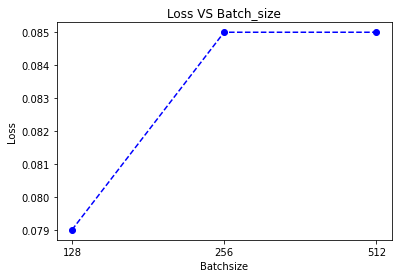
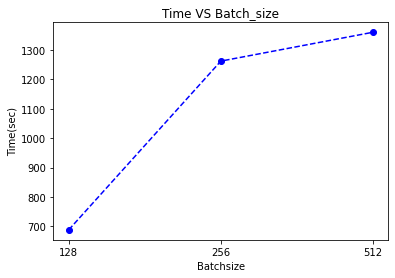
**Observation:**

We can see and compare the results from the plot above. As we increase the number of epochs the TIME/Epochs also increase, the accuracy increased by approx. 1.6% and the loss difference is approx. 0.4%

**Conclusion:**

* Increasing the EPOCHS doesn’t increase the performance of the model drastically.
* The time increases per EPOCH, and the performance doesn’t increase much.
* **Analysis 3 – Different Batch size**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **GPU Count** | **Batch Size** | **EPOCHS** | **Model Parallelism** | **TIME** | **Acc** | **Loss** |
| **3** | **128** | **5** | **yes** | **687.631** | **0.8285** | **0.079** |
| **3** | **256** | **5** | **Yes** | **1125.121** | **0.7803** | **0.085** |
| **3** | **512** | **5** | **Yes** | **1360.3255** | **0.6630** | **0.085** |

****

**Observation:**

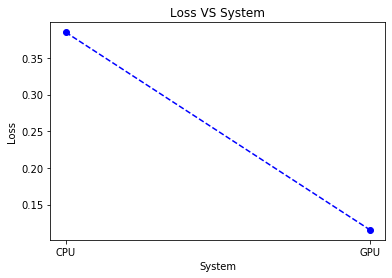
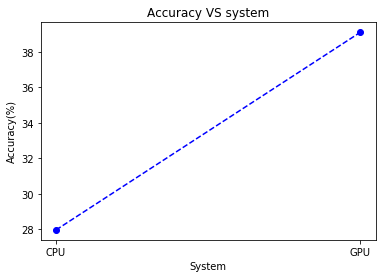
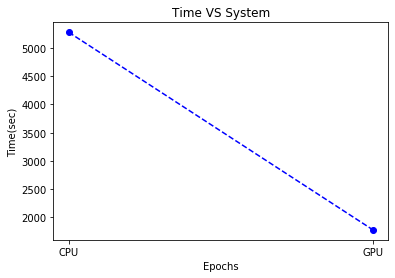
We can see and compare the results from the plot above. As we increase the number of epochs the TIME/Epochs also increase, the accuracy increased by approx. 1.6% and the loss difference is approx. 0.4%

**Conclusion:**

Increasing the batch size doesn’t increase the performance. There is a tradeoff between the number of batch size and the performance of the model. Lower number of batch size works best in our case.

* **Analysis 4 - CPU vs GPU Performance:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Batch Size** | **EPOCHS** | **Model Parallelism** | **TIME** | **Acc Loss** | **Loss** |
| **GPU -1** | **256** | **2** | **NO** | **1776.9850** | **0.3911** | **0.1154** |
| **CPU** | **256** | **2** | **NO** | **5279.6492** | **0.2796** | **0.3849** |



**Observation**: We can see and compare the results from the plot above. As the execution time decreases when we move from CPU to a GPU. The time difference is more than 3000 sec. The accuracy is also more by approx. 10%, and the loss in GPU is also less by approx. 20%. CPU takes ~ 91 min for 1 epoch whereas the GPU is taking

**Conclusion**:

* GPU is very fast comparative to the CPU.
* Performance drastically increases when we move to a GPU. The computation in GPU is much faster for our model. Hence for CNN’s GPU is much better than CPU.
* It is effective to use a GPU when using Neural Networks, as it takes less time, so we can perform more computations on a GPU as to a CPU.

**7) Results:**

* Increasing the number of GPU’S doesn’t always help in the performance. The is a tradeoff between the number of GPU’s and the performance we can achieve through it.
* Increasing the number of EPOCH’s doesn’t change the performance drastically. The accuracy increases till a point after which there is not much increase in the performance. Hence the **accuracy and loss** doesn’t differ much after a certain number of EPOCHS. There is a tradeoff between the number of EPOCHS and the number of GPU’s.
* Increasing the batch size doesn’t increase the performance hence there is a tradeoff between the number of batch sizes and the best performance. Lower batch size works best for us.
* There is a drastic increase in the performance when using a GPU as compared to a CPU. There is a huge difference between the execution time of the two. A GPU can render images very quickly, hence it works better for complex computational problems and works great with CNN’s.
* Looking at all the results we can say that the model works best with 3 GPU’s and Data Parallelism for the given computations in the problem, for EPOCH 10 and Batch Size 128.

1. **References:**
2. [**https://cv-tricks.com/keras/understand-implement-resnets/**](https://cv-tricks.com/keras/understand-implement-resnets/)
3. [**https://www.researchgate.net/figure/The-basic-architecture-of-Resnet152\_fig2\_324961229**](https://www.researchgate.net/figure/The-basic-architecture-of-Resnet152_fig2_324961229)
4. **Resnet 50 -** [**http://ethereon.github.io/netscope/#/gist/db945b393d40bfa26006**](http://ethereon.github.io/netscope/#/gist/db945b393d40bfa26006)
5. **Deep Residual Learning for Image Recognition:** <https://arxiv.org/pdf/1512.03385.pdf>
6. **Tesla Volta V100 -** <http://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf>
7. **Data Parallelism :** <https://www.sciencedirect.com/topics/computer-science/data-parallelism#:~:text=Data%20parallelism%20means%20that%20each,deep%20net%20structure%20and%20parameters>.
8. **PyTorch Documentation:** <https://pytorch.org/docs/stable/index.html>
9. **Joblib documentation:** [**https://joblib.readthedocs.io/en/latest/parallel.html**](https://joblib.readthedocs.io/en/latest/parallel.html)
10. **Cuda :**  <https://pytorch.org/docs/stable/cuda.html>
11. **Discovery Northeastern Documentation :** <https://rc-docs.northeastern.edu/en/latest/using-discovery/workingwithgpu.html#specifying-a-gpu-type>