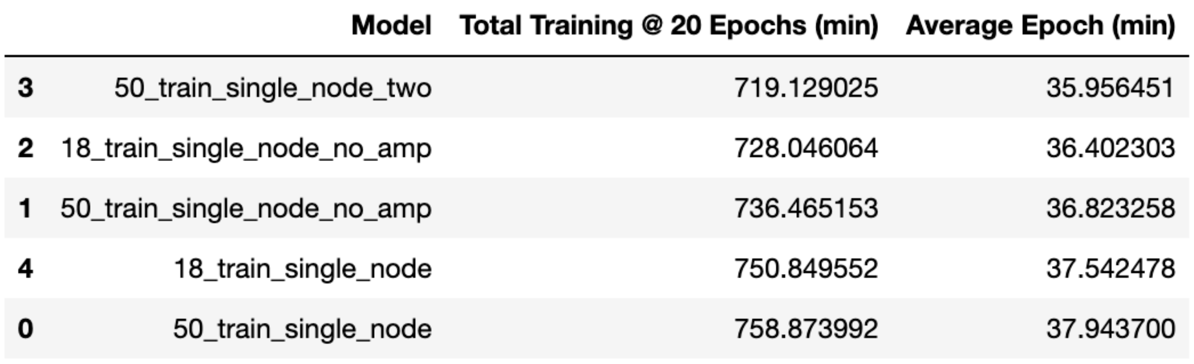
HW #9 Post Mortem:

Single Node Training:

I spent a significant amount of time running variations of the Resnet18 and Resnet50 models. I Trained each model for 20 epochs on a single node, which in retrospect was probably overkill (as evidenced by the > 12 hour run time) but I wanted to see what the performance would look like.



	Model	Total Training @ 20 Epochs (min)	Average Epoch (min)
3	50_train_single_node_two	719.129025	35.956451
2	18_train_single_node_no_amp	728.046064	36.402303
1	50_train_single_node_no_amp	736.465153	36.823258
4	18_train_single_node	750.849552	37.542478
0	50_train_single_node	758.873992	37.943700

Fig 1: Single Node Model Training Times

Metrics from Models:

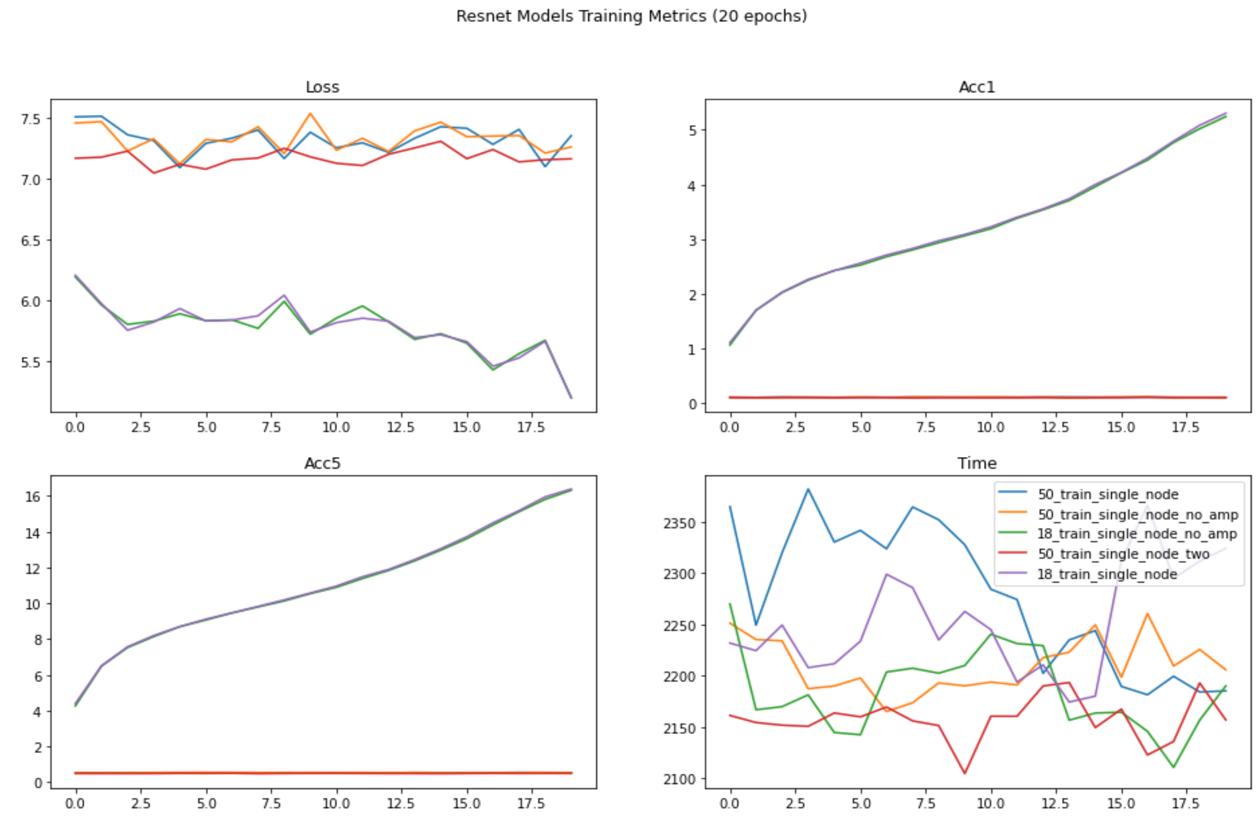


Fig 2: Single Node Model Training Metrics

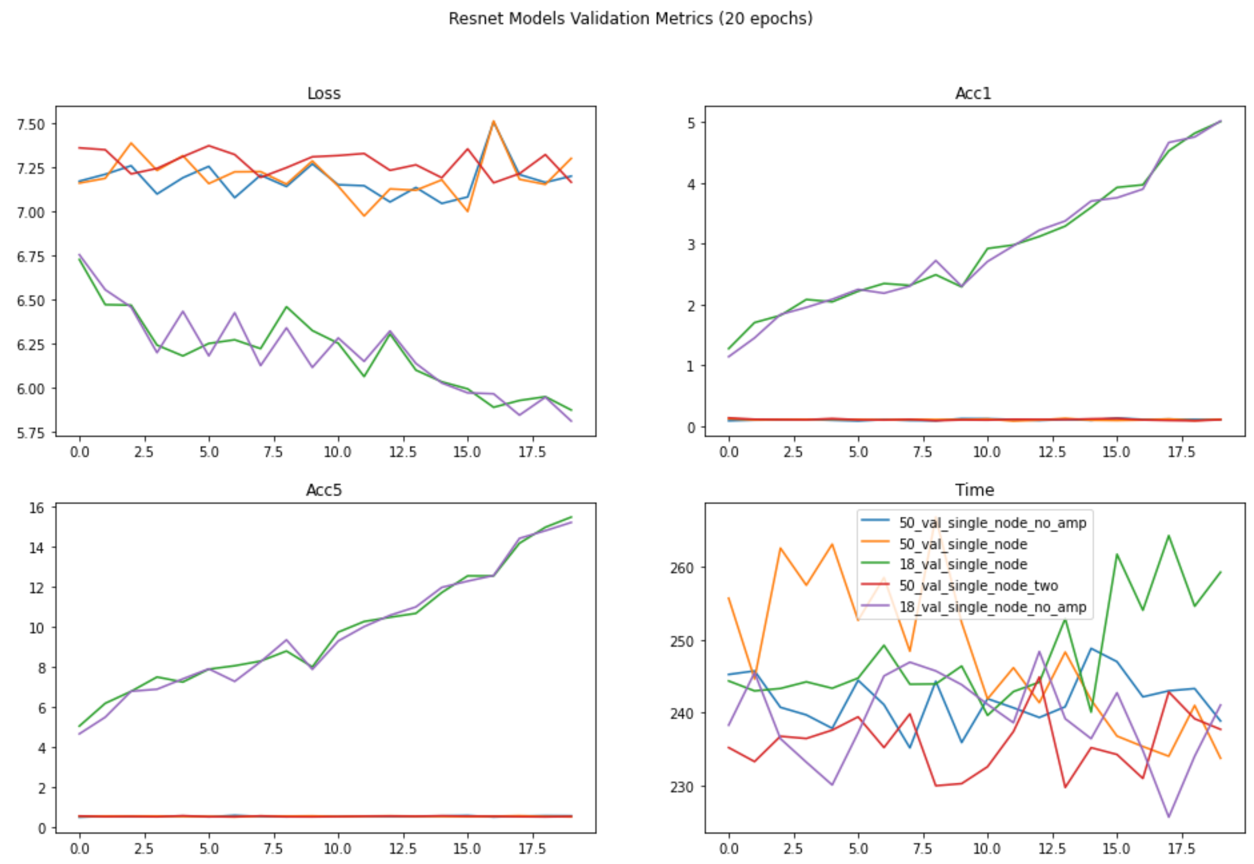
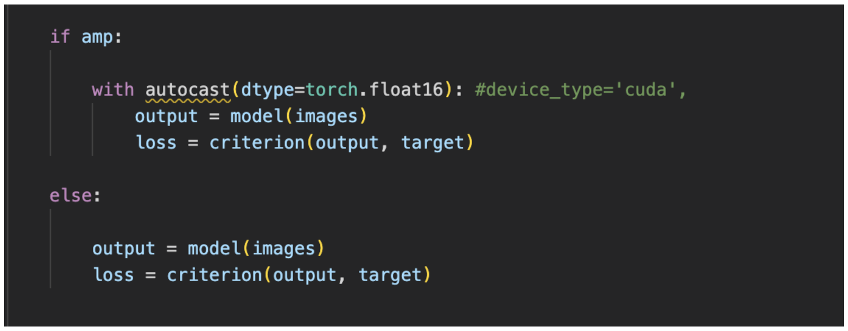


Fig 3: Single Node Model Validation Metrics

Interestingly, only the Resnet18 models showed any performance increases over epoch. It also appears that the Resnet50 model training time is not significantly different from Resnet18, I found this to be highly surprising. Maybe even more frustrating is that using Automated Mixed Precision causes speed issues, which is precisely not what AMP is meant to do.

Q: Do you have any suggestions for why this would happen? Both performance issues with Resnet 50 and AMP speed issues

Background: I instantiated the Resnet50 models identically to the Resnet18 models, I also used the following code in my training function to initiate AMP.



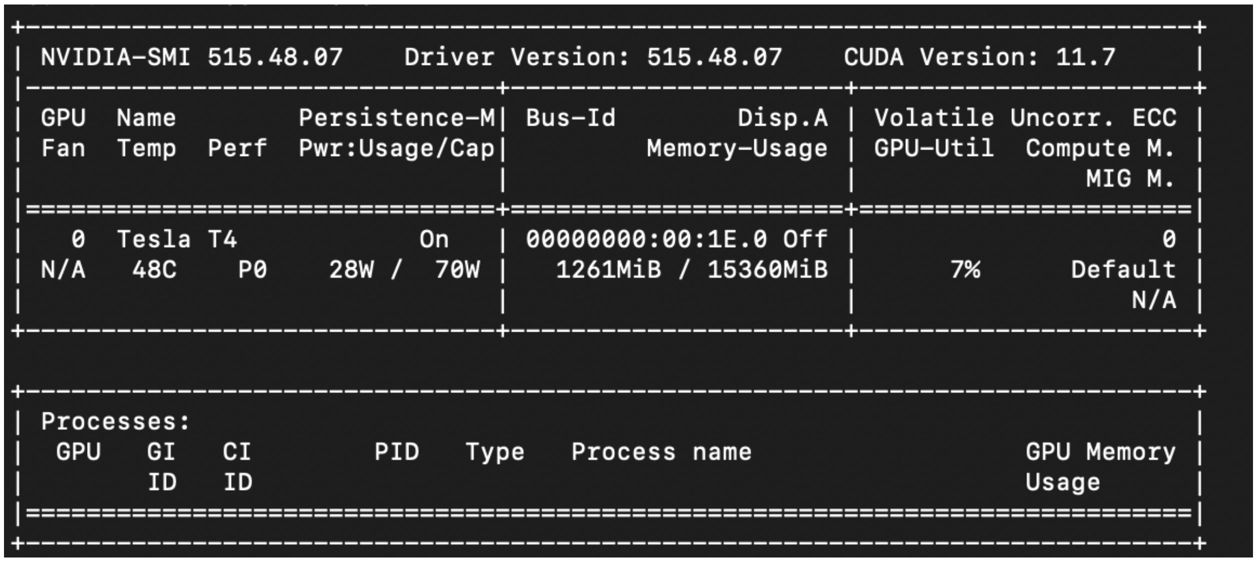
```
if amp:

    with autocast(dtype=torch.float16): #device_type='cuda',
        output = model(images)
        loss = criterion(output, target)

else:

    output = model(images)
    loss = criterion(output, target)
```

Finally, using a G4dn.xlarge (4 vCPUs) instance I was really only able to get GPU feeding to around 7%. I did see that increasing the batch size to 256 increases the speed of Resnet50 training by ~5% (figure 1). Below is an Nvidia SMI snapshot from the Resnet18 training. Looks like I was only able to get up to 7% GPU utilization which is awful. In the future I could try to push the Batch size to fit the larger GPU.



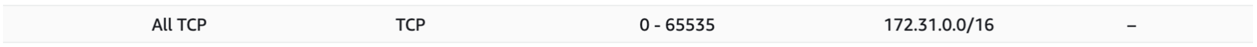
```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 515.48.07    Driver Version: 515.48.07    CUDA Version: 11.7     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            On   | 00000000:00:1E.0 Off |                    0 |
| N/A   48C    P0    28W /  70W |   1261MiB / 15360MiB |      7%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

Fig 4: Single Node Model Training, Nvidia GPU Dashboard

DISTRIBUTED DATA PARALLEL COMPUTING ISSUES:

I opened all ports for communications from my internal IP addresses as seen below:

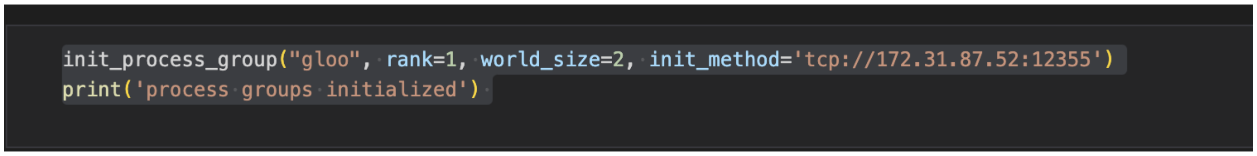


All TCP	TCP	0 - 65535	172.31.0.0/16	–

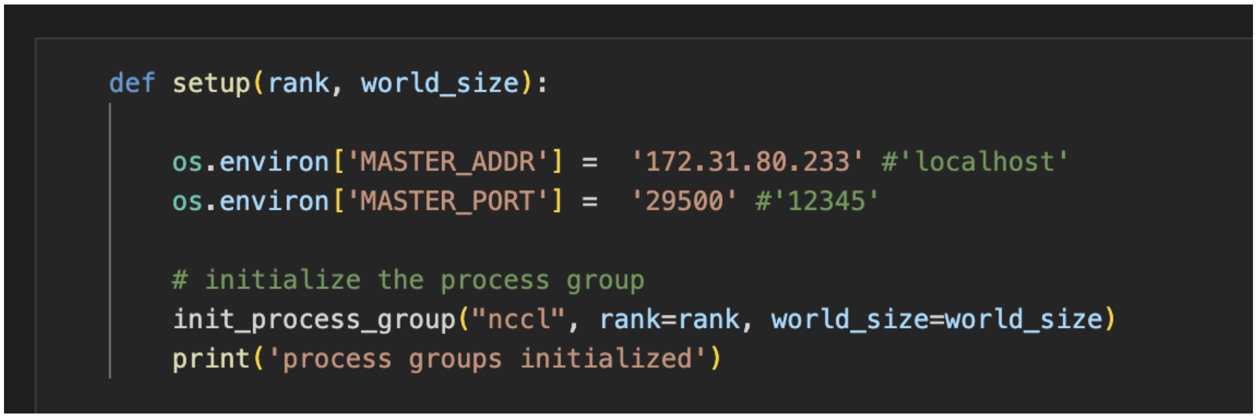
Additionally, I also made sure to map all ports to the docker containers I was running by mapping the host network to the docker network :

“docker run --net=host --gpus=all -ti nvcr.io/nvidia/pytorch:23.02-py3 bash”

I was unable to get the DDP model working. I followed along with multiple Pytorch DDP tutorials, the W251 slack, and still could not get the DDP module to work. I would really like to get this to work but I still cannot get it to work. Below are two code snippets I used to just get the two nodes to communicate - both would just process continuously without completing.



```
init_process_group("gloo", rank=1, world_size=2, init_method='tcp://172.31.87.52:12355')
print('process groups initialized')
```



```
def setup(rank, world_size):

    os.environ['MASTER_ADDR'] =  '172.31.80.233' #'localhost'
    os.environ['MASTER_PORT'] =  '29500' #'12345'

    # initialize the process group
    init_process_group("nccl", rank=rank, world_size=world_size)
    print('process groups initialized')
```