Lab 10: Multi-GPU

Author: Nigel Nelson
Course: CS2300
Date: 2/19/21

Introduction:

This lab acts as an introduction to multi-GPU computing. Specifically, Horovod is utilized, which is a distributed training framework for use with common deep learning libraries. Horovod is utilized to revisit the problem of fruit identification seen in Lab 9, so that performance differences then can observed when comparing multi-GPU training to single GPU training. Furthermore, the Caltech 256 data set is revisited so that multiple GPU configurations can be used to train identical models, so that performance differences can be observed. Finally, emphasis is placed on the interpretation of the created Caltech 256 models, so generalizations about multi-GPU training can be made in order to hypothesize about GPU configurations outside the test set.

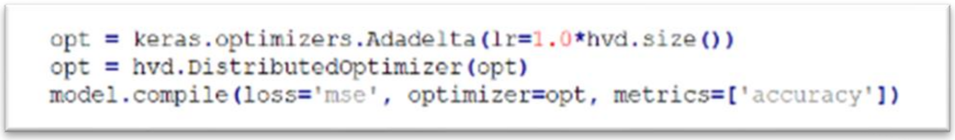
Dataset 1:

The data used to train this fresh vs. rotten fruit identifier for part 1 and part 2 is sourced from Kaggle.com and was used for a model creation competition. The original data consists of 6 different categories of color images, fresh apples, fresh oranges, fresh bananas, rotten apples, rotten oranges, and rotten bananas. The original data is 1.82 GB in size, and is divided into testing and training subsets, where the training set contains 10,901 images, and the testing set contains 2,698 images. The number of images in each image category varies, with the training set having a 700 image difference between the category with the lowest number of images to the category with the highest number of images, while the testing set has a variance of about 200 images per category. However, only a subset of this dataset is used in this lab as well as in Lab 9. This subset of data is 250 MB in size, likely with similar ratios between the 6 categories.

Part 1:

Lab 10 uses the same .py file as Lab 9, except with changes to include Horovod, so that multi-GPU training can be enabled with a distributed optimizer. Leading into this lab, students were instructed to watch a video that acted as an introduction to multi-Node scaling, in this video, there were coding examples of how to implement Horovod. However, theses examples differ from our provided lab10.py file in a couple ways, the code for both implementations is seen below:

Video code example:

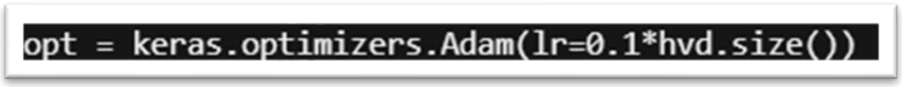


```
opt = keras.optimizers.Adadelta(lr=1.0*hvd.size())
opt = hvd.DistributedOptimizer(opt)
model.compile(loss='mse', optimizer=opt, metrics=['accuracy'])
```

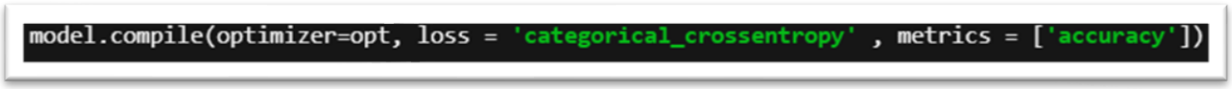
Lab10.py code:

Lab 10: Multi-GPU

Author: Nigel Nelson
Course: CS2300
Date: 2/19/21



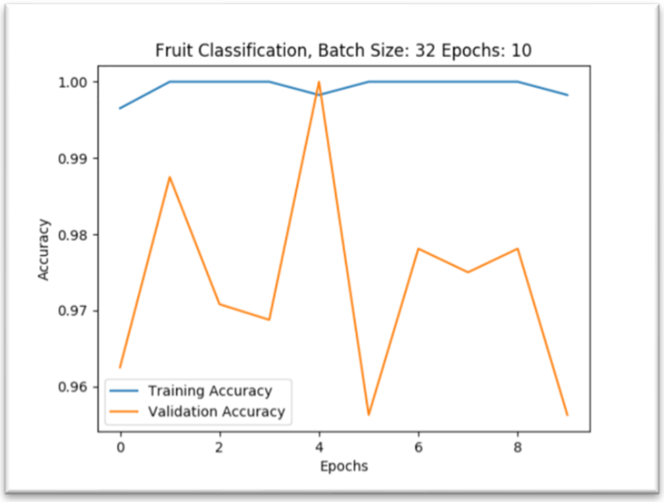
```
opt = keras.optimizers.Adam(lr=0.1*hvd.size())
```



```
model.compile(optimizer=opt, loss = 'categorical_crossentropy' , metrics = ['accuracy'])
```

The first different noticed is the scaling of learning rates. In the video example, the learning rate scaled to 1.0*(number of nodes), whereas in the lab10.py file, the learning rate is scaled to 0.1*(number of nodes). Secondly, the loss argument for the model.compile() call is set to 'mse' in the video code example, and to 'categorical_crossentrophy' in lab10.py. This difference is due to the fact that the video code example calculates loss for a regression model, whereas our model in lab10.py is responsible for classification, as such, these respective loss functions give meaning to their associated model type.

In order to compare results to Lab9, when jobs were run on a single GPU, the same hyperparameters of a batch size equal to 32, and epochs set to 10, as well as fine-tuning and data-augmentation set to true, were used to create a model that was trained on 2 GPUS. The accuracy plot for this model is shown below:



Model 1, Batch size of 32, 10 epochs, trained on 2 T4 GPUs

Interestingly, with all hyperparameters the same, *model 1* created worse results by the 10th epoch of training, with final a validation accuracy of **0.9563**. Interestingly though, both the training and validation accuracies started at a higher value when trained on 2 GPUs. The reason that the end result is ultimately worse, is likely due to the one large difference, the fact that *model 1* was trained on 2 GPUs. This likely resulted in a poorer accuracy score because of the averaging that takes place in data

Lab 10: Multi-GPU

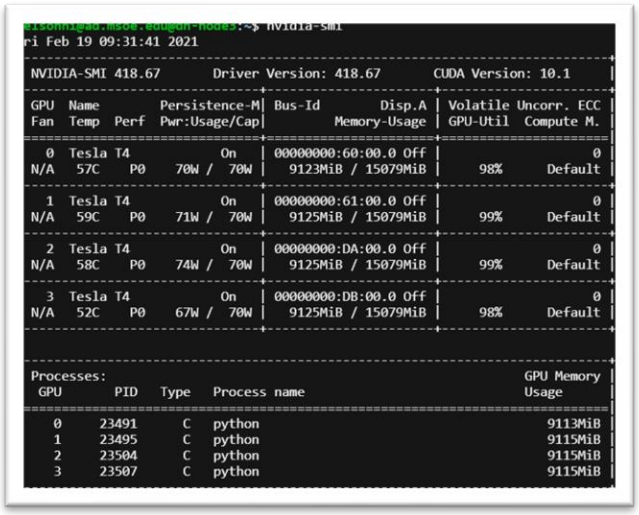
Author: Nigel Nelson
Course: CS2300
Date: 2/19/21

parallelism. Due to the fact that both GPUs receive different data, the way they create a unified model is by averaging their weights, and this averaging likely results in less precision that creates the 2% loss in accuracy when compared to an identical model trained on 1 GPU.

However, the model was able to train significantly faster. By using 2 T4 GPUs, *model 1* trained in **5:19 minutes**, with epoch turnaround times taking approximately **11 seconds** per single GPU, whereas the same model trained on 1 T4 GPU took **11:02 minutes** to train and had an approximate epoch turnaround time of **20 seconds** per GPU**.**

Part 2:

To further test the speed up that can be accomplished with multiple GPUs, a model identical to the hyperparameters in *model 1* was created, *model 2*, and trained on 4 T4 GPUs. The resulting GPU usage from training this model is seen below:



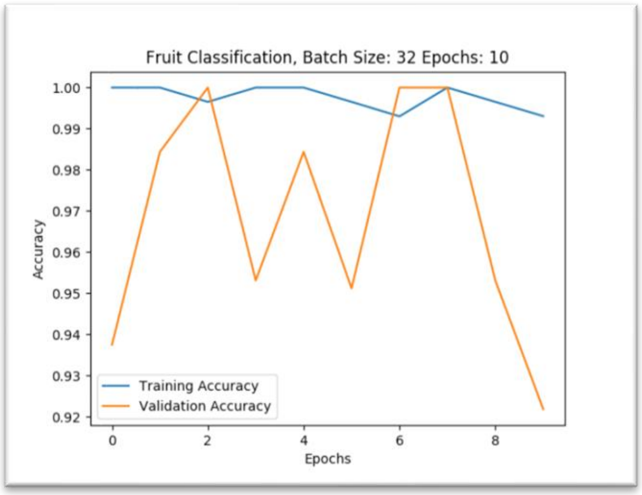
```
elsonni@ad.msoe.edu@dh-node3:~$ nvidia-smi
ri Feb 19 09:31:41 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.67       Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla T4            On   | 00000000:60:00.0 Off |                    0 |
| N/A   57C    P0    70W /  70W |   9123MiB / 15079MiB |     98%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla T4            On   | 00000000:61:00.0 Off |                    0 |
| N/A   59C    P0    71W /  70W |   9125MiB / 15079MiB |     99%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla T4            On   | 00000000:DA:00.0 Off |                    0 |
| N/A   58C    P0    74W /  70W |   9125MiB / 15079MiB |     99%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla T4            On   | 00000000:DB:00.0 Off |                    0 |
| N/A   52C    P0    67W /  70W |   9125MiB / 15079MiB |     98%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0     23491      C   python                                      9113MiB |
|    1     23495      C   python                                      9115MiB |
|    2     23504      C   python                                      9115MiB |
|    3     23507      C   python                                      9115MiB |
+-----------------------------------------------------------------------------+
```

Model 2, GPU usage

Lab 10: Multi-GPU

Author: Nigel Nelson
Course: CS2300
Date: 2/19/21



Model 2, accuracy plot

Interestingly, *model 2*, representing the same hyperparameters as *model 1* except for the fact that it was trained with 4 GPUs, has even worse results after 10th epoch of training. *Model 2* finished its training with a validation accuracy of **0.9219**. For both models, the training accuracy plummets around the 8th epoch. However, *model 1* seemed to be more stable in its training, with a more consistent range of training accuracies. Again, this trend is likely due to the averaging that is taking place between the weights of the 4 GPUs. Due to the fact that there are now 4 different weights to average together, it is likely that by averaging twice the weights as model 1, that even more precision is lost, resulting in a less accurate model.

Benchmarking:

In addition, it was discovered that the complete time to train the model was **3:25 minutes**, which is about an 87% improvement in the time to train the model for 2 times the number of GPUs. In addition, the average turnaround time for an epoch of training was **5 seconds** for model 2, a 75% decrease in the turn around time for an epoch of training on a single GPU when compared to model 1.

Dataset 2:

- For part 3 and part 4 of this lab, the Caltech 256 dataset is used. This dataset is 2.59 GB in size and contains 30,607 images which spans 257 object categories. Each category of images has a minimum of 80 unique images, a maximum of 827 unique images, and a mean of 119 unique images. Each image has a mean of 351 pixels, and the images range from a picture of an iPod, to a picture of Buddha.

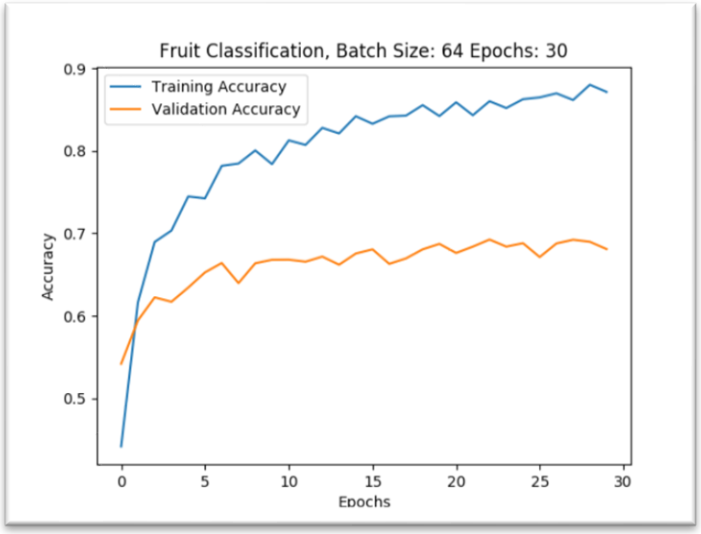
Lab 10: Multi-GPU

Author: Nigel Nelson
Course: CS2300
Date: 2/19/21

Part 3:

To use this data set, new lab10caltech256.sh and Lab10caltech256.py files are provided to utilize the caltech256 data set. Some differences to note between the lab10caltech256.sh file and the lab10.sh file, are that the hyperparameters are different. In the lab10.sh file, the models were trained for 10 epochs, whereas lab10caltech256.sh trains its models for 30 epochs, while both use a batch size of 32. In addition, lab10caltech256.sh does not fine tune its models, whereas lab10.sh does. As for the .py files, the Lab10caltech256.py has a final dense layer of 257, whereas lab10.py has a final dense layer of 6. This is due to the fact that the Caltech 256 dataset has 257 object categories, whereas the fruit classifier only has 6. In addition, the learning rate is scaled to 1/10th of the number of nodes in the Lab10.py file, whereas the Lab10caltech256.py file uses a learning rate scaled to the number of nodes. Furthermore, the Lab10caltech256.py file splits its data so that 20% is used for validation, whereas the Lab10.py file does not have to do this because the data is already split. Both use data augmentation, but implement it differently. The Lab10.py file randomly zooms in on an image up to 1%, shifts the images vertically or horizontally up to 1%, and does not flip images. Whereas the Lab10caltech256.py randomly rotates images up to 5%, zooms in on images up to 20%, shifts the images vertically or horizontally up to 20%, and will horizontally flip images. Furthermore, Lab10caltech256.py file must specify subsets of data for training and validation due to the fact that its data is not pre-split.

After looking through the associated files and understanding their nuances, lab10caltech256.sh was submitted as a batch job in order to create *model 3. Model 3* was trained on 2 T4 GPUs, with a batch size of 32, data augmentation set to true, fine tuning set to false, and trained for 30 epochs. The resulting accuracy plot is shown below.



Lab 10: Multi-GPU

Author: Nigel Nelson
Course: CS2300
Date: 2/19/21

Model 3, accuracy plot

As seen from the accuracy plot, the resulting validation accuracy is a fraction of training accuracy. In addition, there is the observable trend that training accuracy appears to continue improve, whereas validation accuracy seems to reach a plateau around the 15th epoch. For these reasons, it can be reasonably concluded that this model exhibits overfitting.

Benchmarking:

Unsurprisingly due to the larger data set and larger number of epochs, model 3 took much longer to train than *model 1* and *model 2*, with a total training time of **01:38:30 hours**. In addition, a single epoch turnaround took approximately **209 seconds** for a single GPU.

Part 4:

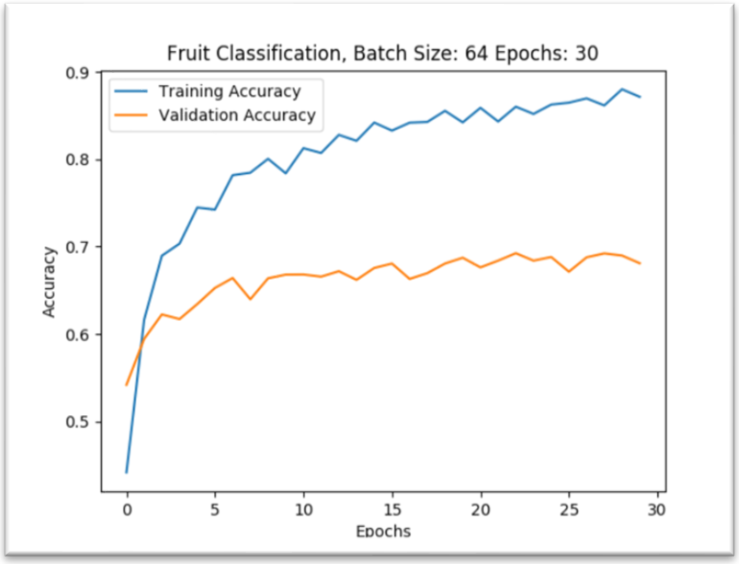
For part 4, several identical models are created as batch jobs, with the only major difference between models being the number and type of GPUs used in their training. For the following models, all were trained with a batch size of 32, data augmentation set to true, fine tuning set to false, and trained for 30 epochs. The models created are compared below.

Model 3

- Number GPUs: **2**
- GPU type: **Nvidia T4**
- Job turnaround time: **01:38:30 hours**
- Single epoch turnaround time for single GPU: **209 seconds**
- Approx. # of epochs to reach stable validation accuracy maximum: **15***
- Final training accuracy: **0.8610**
- Final validation accuracy: **0.6888**

Lab 10: Multi-GPU

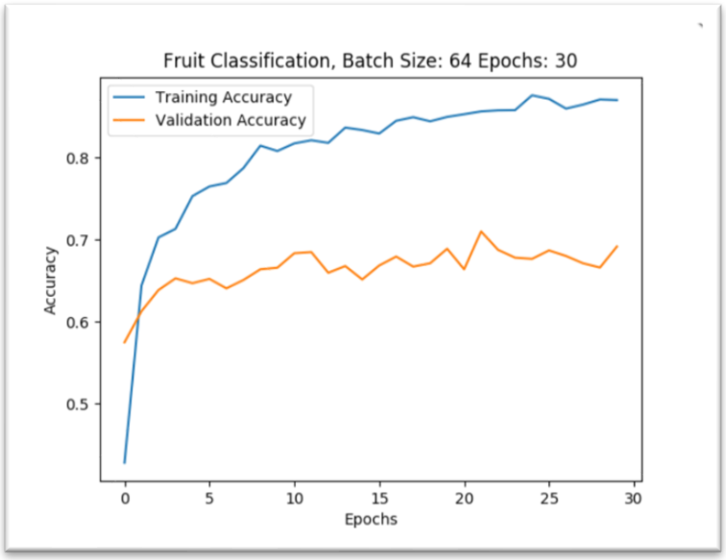
Author: Nigel Nelson
Course: CS2300
Date: 2/19/21



Model 3, accuracy plot

Model 4

- Number GPUs: **4**
- GPU type: **Nvidia T4**
- Job turnaround time: **00:43:14 hours**
- Single epoch turnaround time for single GPU: **81 seconds**
- Approx. # of epochs to reach stable validation accuracy maximum: **19***
- Final training accuracy: **0.8678**
- Final validation accuracy: **0.6760**



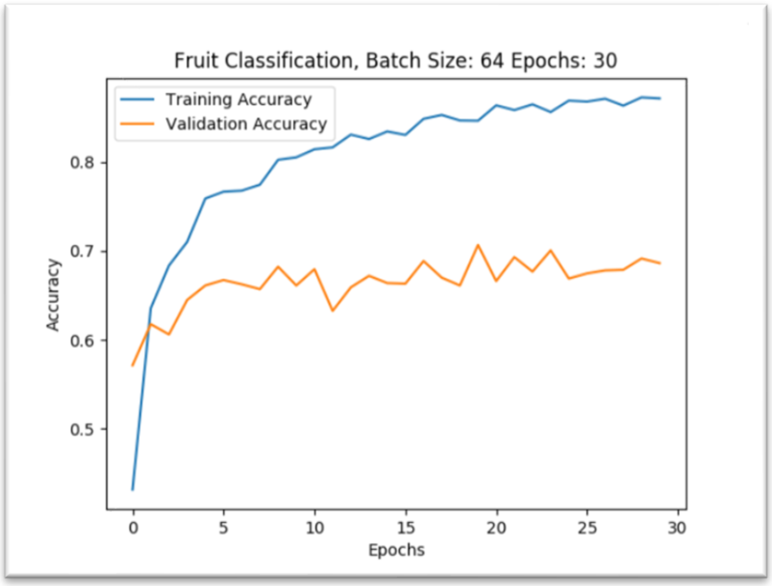
Lab 10: Multi-GPU

Author: Nigel Nelson
Course: CS2300
Date: 2/19/21

Model 4, accuracy plot

Model 5

- Number GPUs: **4**
- GPU type: **Nvidia V100**
- Job turnaround time: **00:41:42 hours**
- Single epoch turnaround time for single GPU: **80 seconds**
- Approx. # of epochs to reach stable validation accuracy maximum: **20***
- Final training accuracy: **0.8713**
- Final validation accuracy: **0.6861**



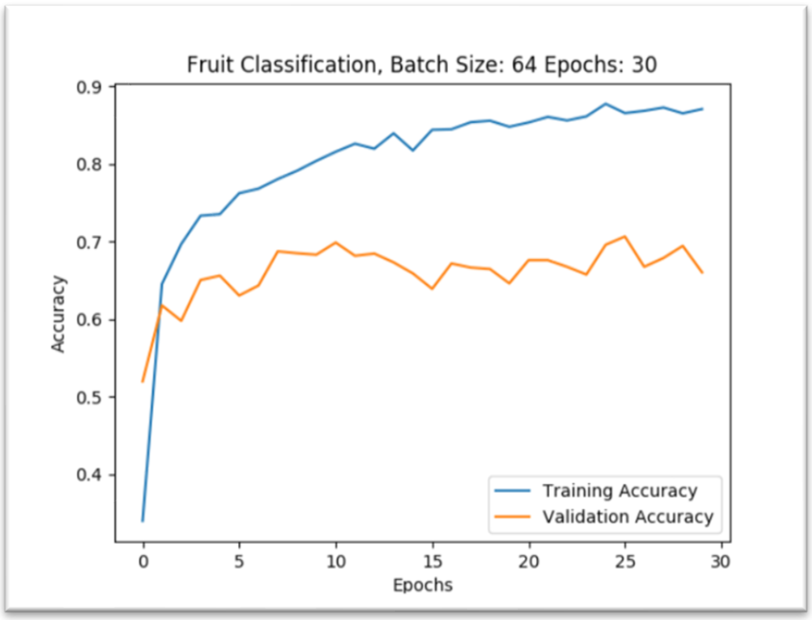
Model 5, accuracy plot

Model 6

- Number GPUs: **8**
- GPU type: **Nvidia V100**
- Job turnaround time: **00:23:07 hours**
- Single epoch turnaround time for single GPU: **44 seconds**
- Approx. # of epochs to reach stable validation accuracy maximum: **25***
- Final training accuracy: **0.8636**
- Final validation accuracy: **0.6719**

Lab 10: Multi-GPU

Author: Nigel Nelson
Course: CS2300
Date: 2/19/21



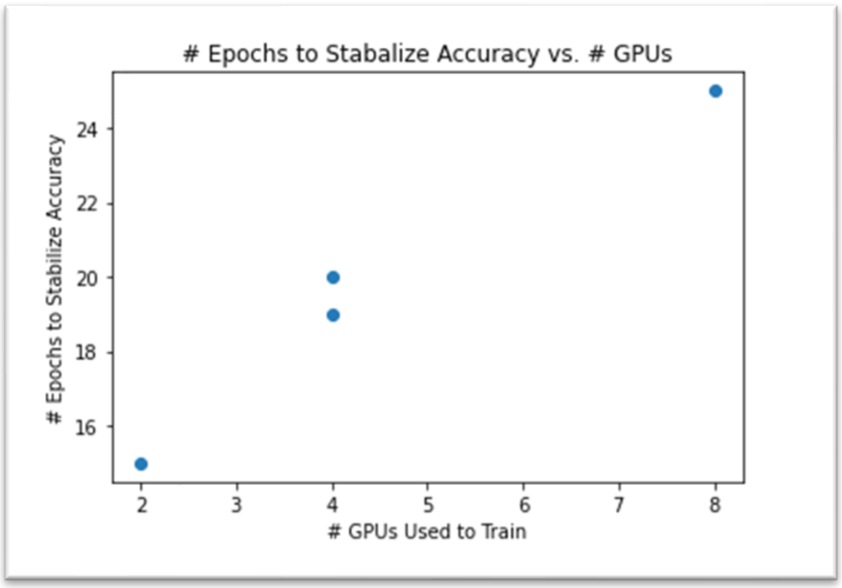
Model 6, accuracy plot

Reflection:

After the results of part 1 and part 2, it is not surprising to see that the number of GPUs used has a noticeable effect on the resulting model accuracy, even when most all other variables are controlled. Specifically, there is a noticeable difference in the number of epochs required for a relatively stable validation accuracy maximum to be established. Training a model for the Caltech 256 data set with 2 GPUs resulted in needing approximately 15* epochs to stabilize training accuracy, 4 GPUs needed approximately 20* epochs, and 8 GPUs needed approximately 25* epochs. These values have an asterisk next to them because they were arbitrarily calculated, by simply looking at the graphs and inspecting at what epoch the validation curve reached both a relative maximum, as well as stability in the scope of the entire plot. While this sample size is admittedly small, from this data one can conclude that as the number of GPUs used in training increases, the number of epochs required to reach a stable validation accuracy increases as well. This trend can be better visualized by plotting the number of GPUs against the number of epochs needed to reach a stable validation accuracy, which can be seen below.

Lab 10: Multi-GPU

Author: Nigel Nelson
Course: CS2300
Date: 2/19/21



Plot of # Epochs to Stabilize Accuracy vs # GPUs Used to Train

With the number of epochs needed to reach a stable validation accuracy in mind, it must be considered which number of GPUs offers the fastest turnaround time to get to this important value. While a lower number of GPUs had the benefit of being able to offer a stable validation accuracy earlier in terms of epochs, ultimately training the models on 8 V100 GPUs offered the quickest turnaround for this problem space. The reason for this is that while 2 GPUs is able to attain stable results 15 epochs into its training, or half way into training, half of its training time would still equate to almost 50 minutes, much more than the total training time for 8 V100 GPUs. In addition, the 4 GPU configurations attained a level of stability about 20 epochs into their training, or 2/3 of the way through. Yet again, 2/3 of their training would still take more than 25 minutes, which is still greater than the entire training time for 8 V100 GPUs. As such, of the combinations tested in part 4, 8 V100 GPUs used for training offer the quickest turn around time in terms of being able to reach a stable maximum validation accuracy.

After seeing the results for the above 4 combinations of GPUs in order to train an image classifier for the Caltech 256 data set, predictions can be made about what observations would be made if an identical model was trained on 16 Nvidia V100 GPUs. First of all, with epochs to stabilize validation accuracy in mind, it would be likely that this model would require a greater number of epochs to stabilize. Furthermore, based on the above graph which shows that the epochs required increase by 5 every time the number of GPUs is doubled, it is also likely that this hypothetical model would take 30 epochs to achieve a stable maximum validation accuracy. It is also likely that this model would take even less time to train. Eventually simply increasing the number GPUs will not correlate to a large increase in time, however, for the tests conducted, this data set seems sufficiently large to take advantage of the additional GPUs, with almost all models taking almost half the time to train for double the GPUs. Due to this, it is likely that this model would take slightly more than 11:30 minutes to train 30 epochs. For the same reasons, this model would surely need less time per single GPU to train for a single epoch, and it is likely that a single GPU would take slightly more than 22 seconds per single epoch of training based on the fact that this quantitative measure is reduced by a little less than ½ for double the GPUs used in training. Lastly, it is likely that this model would produce similar final accuracies for both training and validation. The reason for this is that throughout the 4 different GPU configurations tested, no definite correlation can be observed between the final validation and testing accuracies, with all values being approximately 1% within each other.

Conclusion:

This lab acted as an introduction to multi-GPU computing. Specifically, Horovod is utilized, which is a distributed training framework for use with common deep learning libraries. Horovod was utilized to revisit the problem of fruit identification seen in Lab 9. It was observed that increasing the number of GPUs used in training resulted in a significant decrease in training time, with doubling the GPUs used resulting in the total training time and time to train a single epoch on a single GPU to be almost cut in half. However, it was also observed that increasing GPUs resulted in a less and less accurate model, that also needed more epochs to train sufficiently. Furthermore, the Caltech 256 data set was revisited so that multiple GPU configurations could be used to train identical models. With this larger data set, it was seen that by doubling GPUs used in training, the turnaround time for training as well as time to train a single epoch on a single GPU is decreased by almost ½. However, it was identified that increasing GPUs increased the number of epochs needed to reach a stable maximum validation accuracy level. To conclude, training models with multiple GPUs can offer significant performance advantages when the data set is sufficiently large, but the tradeoffs associated with this performance increase must be considered when deciding if multi-GPU usage is appropriate for a given model.