Deep Learning Classification on Chest X-ray Images with CNN, ResNet parallelized on Multiple GPUs

Team 15: Zifeng Jiang

## Introduction

Pneumonia is an infection of one or both of the lungs caused by bacteria or viruses. It is a serious infection in which the air sacs fill with pus and other liquid. Pneumonia accounts for 14% of all deaths of children under 5 years old, killing 740 180 children in 2019. So, it is a very severe disease which may cause people die, so it is very important to discover it before it’s too late.



The normal chest X-ray (left panel) depicts clear lungs without any areas of abnormal opacification in the image. Bacterial pneumonia (middle) typically exhibits a focal lobar consolidation, in this case in the right upper lobe (white arrows), whereas viral pneumonia (right) manifests with a more diffuse ‘‘interstitial’’ pattern in both lungs.

In recent years, the field of computer vision has become more and more popular. As a direct result of these advancements, it has become possible for computer vision models to surpass humans in efficiently solving different problems related to image recognition, object detection, face recognition, image classification, etc.

Machine learning and deep learning has a phenomenal range of applications, including in health and diagnostics. In order to reduce the burden on doctors and make the computers be able to automatically identify whether a human has pneumonia or the human is healthy by chest X-ray images, I built this project with 2 deep learning models with parallel methods to help the chest X-ray images classification mission. But for massive images, the computation is very time consuming which means we need to do parallel.

The goal of this project is to do various comparison between models, parallel strategy and parallel method. In the end, I will compare the execution time between dask and NumPy array calculation, multiprocessing with multiple CPUs, different GPU parallel strategy and multiple GPUs.

## Methodology

1. Data Preprocessing Parallel

First, I combine the validation folder and test folder into one test folder. Then, I read all the images from folders. Next, resize all the images’ size to 128x128(Every image is larger than 128x128). And I did this process in my local computer machine and uploaded the both the cleaned dataset and the original dataset to discovery. This part of code can be seen in ResizeImages.ipynb file.

Graphical user interface, application

Description automatically generated

Next, from discovery or OOD, loading all the cleaned images into an array with labels. In this process, I use multiprocessing to compare the elapsed time when using different number of CPUs (1, 2, 4, 8).

A picture containing text, keyboard, indoor, electronics

Description automatically generated

Multiprocessing is the use of two or more central processing units (CPUs) within a single computer system. The term also refers to the ability of a system to support more than one processor or the ability to allocate tasks between them. In Python, multiprocessing package supports spawning processes using an API similar to the threading module.

The NumPy arrays are mainly used to load the pixel data from the images so that I can use dask array to optimize it. The NumPy operation in the serial mode are compared with parallelism by using dask arrays when I normalize image array pixel values between 0 and 1.

Dask Array implements a subset of the NumPy array interface using blocked algorithms, cutting up the large array into many small arrays. This lets us compute on arrays larger than memory using all of our cores.

Moreover, before I use the pretrained ResNet model, I also change the grey scale images into RGB 3 channel format in order to fit in the pretrained ResNet50 model. The procedure of doing this is to repeat the grey scale value three times in order to get the RGB value of the same grey scale image. In this preprocessing part, I also use NumPy array and dask array to compare the speed of this process.

1. Deep Learning Models and Parallelism

In this project, I built 2 deep learning models and parallel them using TensorFlow and Keras. The first model is a VGG type CNN model with kernel size 3x3 and 32, 64, 128 filters on different layers. The pooling method I use is max pooling because average pooling method smooths out the image and hence the sharp features may not be identified when this pooling method is used. Max pooling selects the brighter pixels from the image. It is useful when the background of the image is dark and we are interested in only the lighter pixels of the image, which suits my X-ray images data.

A picture containing chart

Description automatically generated

The second one is a pretrained ResNet50 model with initial weights selected from ImageNet dataset. ResNet-50 is a convolutional neural network with bottleneck shortcuts that is 50 layers deep.

Table

Description automatically generated

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset. Training deep learning neural network models on more data can result in more skillful models, and the augmentation techniques can create variations of the images that can improve the ability of the fit models to generalize what they have learned to new images.

Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

All the models will be trained by augmented images with batch normalization.

The hyper parameters are as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | CNN-1GPU | CNN-2GPUs | ResNet-1GPU | ResNet-2GPUs |
| Epoch | 20 | 20 | 10 | 10 |
| Batch size | 128 | 128 | 32 | 32 |
| Optimizer | Adam | Adam | Adam | Adam |
| Learning rate | 0.00001 | 0.00001 | 0.00001 | 0.00001 |

I just run few epochs for each model because the purpose of this project is not to pursue the most accurate classification model. And these few epochs are obvious enough to show us the parallel results.

On the OOD, I mainly compared the elapsed time of the same model using different TensorFlow strategies. The results and code can be seen in Final\_Project\_Zifeng\_Jiang\_OOD.ipynb file. There are two TensorFlow strategies I used. First one is the One Device Strategy. One Device Strategy is a strategy to place all variables and computation on a single specified device. This strategy is distinct from the Default Strategy in a number of ways. In the Default Strategy, the variable placement logic remains unchanged when compared to running TensorFlow without any distribution strategy. But when using One Device Strategy, all variables created in its scope are explicitly placed on the specified device. Moreover, any functions called by One Device Strategy will also be placed on the specified device. Input distributed through this strategy will be prefetched to the specified device. In the Default Strategy, there is no input distribution. Similar to the Default Strategy, this strategy could also be used to test your code before switching to other strategies which actually distribute to multiple devices/machines. Second one is Mirrored Strategy. Mirrored Strategy supports synchronous distributed training on GPUs on one machine. It creates one replica per GPU device. Each variable in the model is mirrored across all the replicas. Together, these variables form a single conceptual variable called Mirrored Variable. These variables are kept in sync with each other by applying identical updates. It also can be used in one GPU. So, on the OOD, I used one k80 GPU to compare those two strategies.

On discovery, I mainly compared the elapsed time of the same model using 2 Tesla V100 GPUs. For both CNN and ResNet, I used Mirrored Strategy for one GPU and 2 GPUs. Then, compare the time of training epoch of each model. The code and can be seen in Final\_Project\_Zifeng\_Jiang\_MultiGPU.py file.

## Description of Dataset

The dataset is organized into 3 folders (train, test, validation) and contains subfolders for each image category (Pneumonia/Normal). There are 5,863 X-Ray images (JPEG) and 2 categories (Pneumonia/Normal). Chest X-ray images were selected from retrospective cohorts of pediatric patients of one to five years old from Guangzhou Women and Children’s Medical Center, Guangzhou. All chest X-ray imaging was performed as part of patients’ routine clinical care.

A picture containing text

Description automatically generated

For the analysis of chest x-ray images, all chest radiographs were initially screened for quality control by removing all low quality or unreadable scans. The diagnoses for the images were then graded by two expert physicians before being cleared for training the AI system. In order to account for any grading errors, the evaluation set was also checked by a third expert. So the data is well labeled.

Graphical user interface, text, application, email

Description automatically generated

The size of the dataset is 1.15GB.

## Result and Analysis

1. Multiprocessing of Data Loading Parallel

In the process of loading all the cleaned images into an array with labels from discovery or OOD, I use multiprocessing.pool to compare the elapsed time when using different number of CPUs on discovery. The environment I use for this part is discovery with 8 CPUs. The result sheet is as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 CPU | 2 CPUs | 4 CPUs | 8 CPUs |
| time | 5.32 seconds | 4.44 seconds | 4.46 seconds | 4.48 seconds |

And the result plot is as follows:

Chart, line chart

Description automatically generated

This plot shows us that when number of CPUs is less than or equal to 2, the elapsed time is decreasing. But when number of CPUs is more than 4, the elapsed time is greater than number of CPUs is 2 because there is much overhead when using more CPUs.

1. Dask and NumPy Comparison

First, I compared the processing speed of NumPy array and dask array when I normalize array pixel values to be between 0 and 1. The elapsed time for NumPy array is 0.2415 seconds. And the elapsed time for dask array is 0.0033 seconds.



Then, before I use the pretrained ResNet model, I also change the grey scale images into RGB 3 channel format in order to fit in the pretrained ResNet50 model. The result is the elapsed time for NumPy array is 1.4278 seconds. And the elapsed time for dask array is 0.0074 seconds.

As we can see above, all the processes using dask arrays are faster than NumPy arrays.

1. Strategies Comparison

For the strategies comparison part, I use OOD with k80 GPU and GPU version of TensorFlow 2.4, CUDA 11.4 to compare the two different TensorFlow strategies. The time for training a CNN model with One Device Strategy is totally about 301 seconds. The time for training each epoch is about 15 seconds. The screenshot of the result is as follows:

Table

Description automatically generated

The time for training a CNN model with Mirrored Strategy using only one GPU is totally about 299 seconds. The time for training each epoch is also about 15 seconds. The screenshot of the result is as follows:

A picture containing text

Description automatically generated

The time for training a ResNet model with One Device Strategy is totally about 506 seconds. The time for training each epoch is about 49 seconds. The screenshot of the result is as follows:

Text

Description automatically generated

The time for training a ResNet model with Mirrored Strategy is totally about 549 seconds. The time for training each epoch is about 52 seconds. The screenshot of the result is as follows:

Text

Description automatically generated with medium confidence

Therefore, for CNN using only one GPU, different strategies will not have very large impact on the speed of the training. But for ResNet on one GPU, the One Device Strategy is better than Mirrored Strategy. This shows that when we are using larger and deeper models and just have one GPU, it is better to use One Device Strategy.

1. Multiple GPUs Comparison

For multiple GPU parallel part, I compare the Mirrored Strategy with one GPU and 2 GPUs on discovery d1013 node and the execution time of CNN for each epoch is 15 seconds and 8 seconds respectively. As for ResNet, the execution time for each epoch is about 52 seconds and 35 seconds respectively. I did not apply for 4 GPUs because the GPU resources is very hard to apply and if I apply for more than 2 GPUs, I needed to wait for a long time. So, I just apply for 2 GPUs in multiple GPU part of my project.

Here is the screen shot of using one GPU to train CNN:

Table

Description automatically generated

Here is the screen shot of using 2 GPUs to train CNN:

Text

Description automatically generated

Here is the screen shot of using one GPU to train ResNet:

Table

Description automatically generated

Here is the screen shot of using 2 GPUs to train ResNet:

Graphical user interface, text

Description automatically generated with medium confidence

As we can see, when I use 2 GPUs, the execution time is nearly half of the single GPU. But the time is a litter bit longer than that because there is some overhead. What’s more, CNN parallelism speedup is 15/8 (1.875). ResNet parallelism speedup is 52/35(1.486).

So the CNN parallelism speedup is greater than ResNet parallelism speedup.

Table

Description automatically generated with low confidence

Moreover, we can calculate the partition of parallel using the speedup rate and Amdahl's law.

A picture containing diagram

Description automatically generated

For CNN, the parallel partition is 14/15(93.3%). For ResNet, the parallel partition is 17/26(65.4%). As we can see above, CNN is a much simpler model that can be easy parallelized than ResNet. One of the reasons is that the ResNet model I use is a pretrained model, which means that the model or the data input to the model is less flexible to parallel.

## Conclusion

In this project, I tried four different ways to compare the speedup. First, I use multiprocessing to read the data from folder using different number of CPUs and compare the time. It turns out that 2 CPUs is the most optimized choice. And if we use more than 2 CPUs, there will be large overhead.

Second, I use dask array to calculate the result compared to NumPy array. The results shows that the dask arrays are faster than NumPy arrays in both of the two processes.

Third, I compared the execution time for different strategies within a model. When using only one GPU, it is better to use One Device Strategy. And this will have more impact on ResNet than CNN.

In the end, I compared the single GPU and multiple GPUs processing. The execution time of 2 GPUs is approximately half of the singe GPU execution time. Because there are still some overheads, so the execution time for 2 GPUs is a little bit longer than half of the time when using only one GPU to do the model training.

Thus, from all the parallel method I used, I learned that deep learning parallel is a very useful and multiple GPUs increase both memory and compute available for training a DNN which will save a lot of time. Especially for this project, saving time for doctors may save more lives.

## References

1. This data source comes from the " Chest X-Ray Images (Pneumonia) " dataset on Kaggle. Its link is as follows:

<https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia>

2. The original version of this dataset can be found at this link:

<https://data.mendeley.com/datasets/rscbjbr9sj/2>

3. <https://www.tensorflow.org/guide/distributed_training>

4.<https://towardsdatascience.com/train-a-neural-network-on-multi-gpu-with-tensorflow-42fa5f51b8af>

5.<https://rc-docs.northeastern.edu/en/latest/using-discovery/workingwithgpu.html#using-gpus-with-tensorflow>

6.<https://en.wikipedia.org/wiki/Multiprocessing#:~:text=Multiprocessing%20is%20the%20use%20of,to%20allocate%20tasks%20between%20them>.

7. <https://docs.dask.org/en/stable/array.html>

8.<https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/>

9.<https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/>

10. <https://www.hopkinsmedicine.org/health/conditions-and-diseases/pneumonia>

11. <https://www.who.int/news-room/fact-sheets/detail/pneumonia>

12.<https://towardsdatascience.com/scalable-deep-learning-on-parallel-and-distributed-infrastructures-e5fb4a956bef>