EECE 5640 – High Performance Computing

Final Project Report

Qiulin Luo

Introduction:

From the initial MNIST for handwritten digit classification, to the larger CIFAR-10, CIFAR-100, and up to the ImageNet task of 1000 classification, the image classification task has been increasing with the size of the dataset, which requires a high performance runtime environment for the image classification task in order to obtain better results and a more acceptable runtime speed. The dataset that my project is based on is CIFAR-10, and I have implemented both VGG and GoogleNet models in a high performance graphics environment. I will practice and analyze two different workloads for these two models.

Dataset:

Cifar-10[1] is a dataset for pervasive object recognition collected by Alex Krizhevsky, Ilya Sutskever. Bengio and his students received a small amount of funding from Cifar in 2004 to build the Neural Computation and Adaptive Perception project. This project brought together a number of computer scientists, biologists, electrical engineers, neuroscientists, physicists, psychologists to accelerate the process of Deep Learning, which emphasizes adaptive perception and artificial intelligence at the intersection of computer and neuroscience.

CIFAR-10[1] is a scaling problem in machine learning classification problem. The goal is to classify RGB 32*32 images into 10 categories, which are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. As can be seen, compared to the mature face recognition, the challenge of pervasive object recognition is huge, the data contains a large number of features, noise, and the proportion of recognized objects varies[2]. Thus, Cifar-10 is quite challenging compared to traditional image recognition datasets.

Workload 1

CIFAR 10 Classification Using 2 CV-Nets

(Impact of Network Model)

Architecture:

In my VGG implementation[3], I used three 3×3 convolutional kernels instead of 7×7 convolutional kernels, and in the meantime, two 3×3 convolutional kernels instead of 5×5 convolutional kernels. Three 3×3 successive convolutions are equivalent to one 7×7 convolution. The main purpose of this is to improve the depth of the network while ensuring the same perceptual field, which improves the neural network to some extent. The specific VGG network is as described in the paper[3].

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

VGG16 contains 16 hidden layers (13 convolutional layers and 3 fully connected layers), as shown in column D of the above photo. VGG19 contains 19 hidden layers (16 convolutional layers and 3 fully connected layers), as shown in column E of the above photo. The structure of the VGG network is very consistent, using 3×3 convolution and 2×2 max pooling from start to finish.

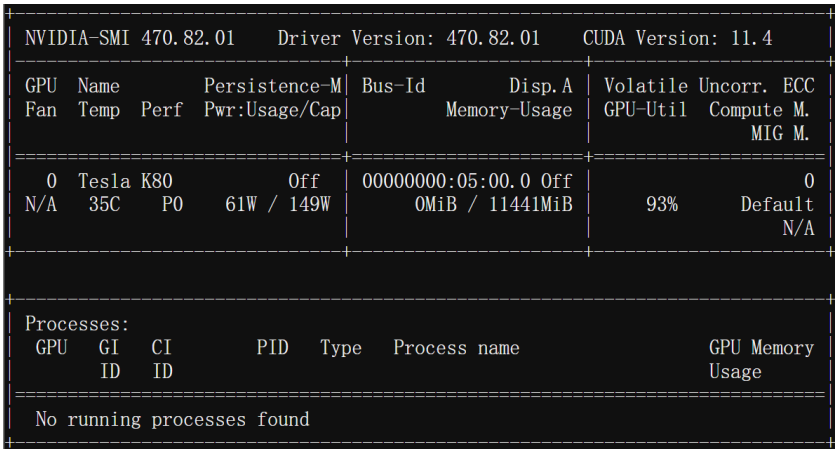
GoogleNet's architecture is slightly more complex than VGG's implementation, which has bothered me for a long time. According to the paper[4], GoogleNet introduces the Inception structure, which not only fuses feature information at different scales but also uses a 1×1 convolutional kernel for dimensionality reduction and mapping. This

comes from the idea[4] that biological nervous systems are sparsely connected and that if the probability distribution of a data set can be described by a large and very sparse neural network, then an optimal network topology can be constructed layer by layer by analyzing the statistical properties associated with the activation values of the preceding layers and clustering the neurons whose outputs are highly correlated. The specific VGG network is as described in the paper[4].

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Training:

System: Discovery Cluster GPU Nodes .



Requirment: Python 3, PyTorch 1.0.0 +, TorchVision, TensorboardX.

The default learning rate is 0.001, and the streamlined default epoch is 20, the training section batch size and the testing section batch size are all 100 by default.

Results:

(GoogleNet)

```
Epoch: 1 ===== Loss: 1.2294 | Acc: 54.898%
```

```
Epoch: 2 ===== Loss: 0.7538 | Acc: 73.548%
Epoch: 3 ===== Loss: 0.5848 | Acc: 79.630%
Epoch: 4 ===== Loss: 0.4832 | Acc: 83.356%
Epoch: 5 ===== Loss: 0.4155 | Acc: 85.584%
Epoch: 6 ===== Loss: 0.3583 | Acc: 87.616%
Epoch: 7 ===== Loss: 0.3203 | Acc: 88.878%
Epoch: 8 ===== Loss: 0.2872 | Acc: 90.094%
Epoch: 9 ===== Loss: 0.2511 | Acc: 91.218%
Epoch: 10 ===== Loss: 0.2200 | Acc: 92.310%
```

(VGG)

```
Epoch: 1 ===== Loss: 1.7690 | Acc: 30.518%
Epoch: 2 ===== Loss: 1.1273 | Acc: 58.872%
Epoch: 3 ===== Loss: 0.8659 | Acc: 69.304%
Epoch: 4 ===== Loss: 0.7291 | Acc: 74.896%
Epoch: 5 ===== Loss: 0.6250 | Acc: 79.096%
Epoch: 6 ===== Loss: 0.5508 | Acc: 81.686%
Epoch: 7 ===== Loss: 0.4834 | Acc: 84.100%
Epoch: 8 ===== Loss: 0.4259 | Acc: 85.990%
Epoch: 9 ===== Loss: 0.3868 | Acc: 87.450%
Epoch: 10 ===== Loss: 0.3557 | Acc: 88.422%
```

Analyzation:

In terms of running speed, VGG is much faster than GoogleNet because of its simpler structure. It can be seen that the training accuracy of GoogleNet is higher than that of VGG on the same classification task. This may be due to the fact that the entire network of VGG is a tandem construction of different neural modules[3]. In tandem networks, the convolutional kernels at each level are of fixed size, and only fixed-scale features can be extracted. The models constructed based on such a single-scale feature map are not robust and have poor generalization ability.

Although the stacking of multiple layers of small convolutional kernels to obtain large convolutional kernels can reduce the number of parameters, it is a drop in the bucket. The deeper the network is, the more likely it is that the gradient will disappear, making

it difficult to train the network[5]. As far as I'm know, the underlying model of faster R-CNN officially supports VGG and ZF, also in the case of GPU K80, ZF is about 8 FPS speed, while VGG is about 3 FPS, which undoubtedly shows that VGG is somewhat heavy. Although some layers are added on top of VGG in the detection task, it is enough to show that GoogleNet is more lightweight compared to VGG in any case. But GoogleNet also has its own problems. In order to prevent the gradient from disappearing, it adds two loss functions, SoftMax0 and SoftMax1[4], to the front layers, and it is these two loss functions that cause the scalability of GoogleNet to be not as strong as VGG.

Workload 2

CIFAR 10 Classification Using VGG
(Impact of GPU Models)

Training:

System: Discovery Cluster GPU Node.

Requirment: Python 3.6, PyTorch 0.4.0 +, TorchVision 0.2.0, NumPy.

Machine learning related training parameters and workload 1 are the same, trained on P100 and K40m respectively.

Results:

(200 epoch) P100: 20min 38s

(200 epoch) K40m: 43min 12s

Analyzation:

Obviously using P100 to complete the task is much faster than K40m. First, the single-precision performance and double-precision performance as well as the memory bandwidth P100 exceed K40m[6]. The single-precision performance of P100 is 9519 GFLOPS, double-precision performance is 4760 GFLOPS, and memory bandwidth is 732 GB/s, while the single-precision performance of K40m is 4291 GFLOPS, double-precision performance is 1430 GFLOPS, and memory bandwidth is 288 GFLOPS, which means that P100 has great ability to improve VGG's operation[6][7].

By an example that can be extended to VGG's operational model. For a 3×3 convolution operation with a stride of 1, assume that the input data plane is 64×64. For simplicity, assume that both the input and output features are 1. At this point, a total of 62×62 convolution operations are required, and each convolution requires $3 \times 3 = 9$ multiplications and additions, so the total number of computations is 34596, and the amount of data is (assuming that both data and If we switch to 1×1 convolution, the total number of calculations becomes $64 \times 64 = 4096$, and the amount of data required is 8194. Obviously, switching to 1×1 convolution reduces the computation volume by a factor of nearly 9, but also reduces the computation intensity to 0.5, which means that the memory bandwidth requirement also increases by a factor of nearly 9. Therefore, if

the memory bandwidth is not sufficient for 1×1 convolutional computation, switching to 1×1 convolutional computation reduces the computation volume by a factor of 9, but does not increase the computation speed by a factor of 9. As we can see, there are two bottlenecks in deep learning computing devices, one is the processor computing power, and the other is the computing bandwidth[8]. That's why P100 beats K40m.

References

[1]https://www.tensorflow.org/datasets/catalog/cifar10

[2]Krizhevsky A, Hinton G. Learning multiple layers of features from tiny images[J]. 2009.

[3]Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.

[4] Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2015: 1-9.

[5]Swapna M, Kumar, Y, Prasadh B. CNN Architectures: AlexNet, LeNet, VGG, GoogleNet, ResNet[J]. IJRTE, issn: 2277-3878, volume-8, issue-6. 2020.

[6] NVIDIA Tesla P100 Whitepaper

[7]NVIDIA Tesla K40 Whitepaper

[8] https://en.wikipedia.org/wiki/Roofline