哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

实验报告

开课学期：	2023 秋季
课程名称：	计算机体系结构
实验名称：	AI 系统实践专题
学生班级：	智能强基-计算机
学生学号：	210010101
学生姓名：	房煊梓
评阅教师：	
报告成绩：	

实验与创新实践教育中心制

2023 年 12 月

一、实验概述

1. 实验目的

a. 掌握 PyTorch 深度学习的基本实现架构及部署流程
b. 学习 SMOKE 目标检测算法的基本原理
c. 学习如何基于 PyTorch 训练 SMOKE 模型和导出权重结果
d. 掌握 C/C++/CUDA 与 PyTorch 的绑定

2. 实验内容

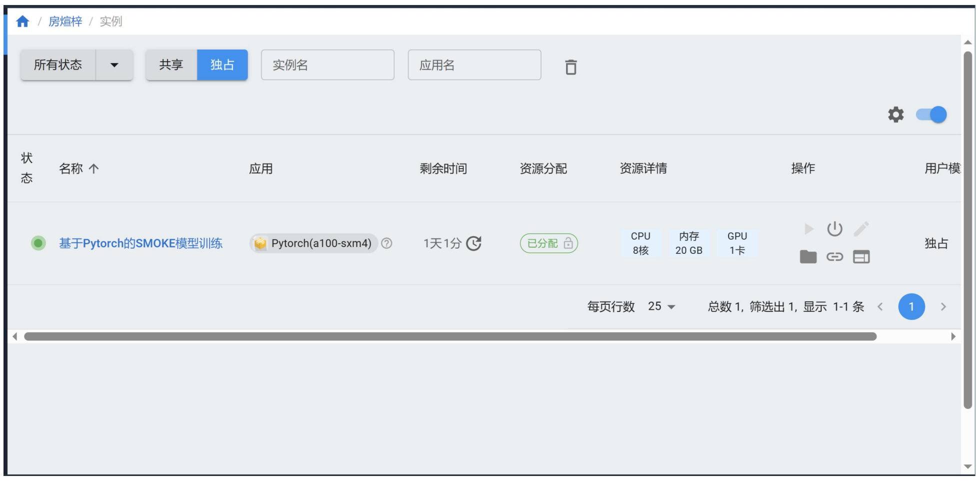
本实验旨在掌握 PyTorch 深度学习的核心实现结构以及相关的部署步骤。通过完成本实验内容，将获得 PyTorch 深度学习的基础知识和实践经验，以及掌握 SMOKE 目标检测算法的核心原理和基于 PyTorch 的模型训练、导出过程。这将为进一步深入学习和应用深度学习技术打下坚实的基础。

二、实验流程和测试结果及原理分析

步骤一：环境准备

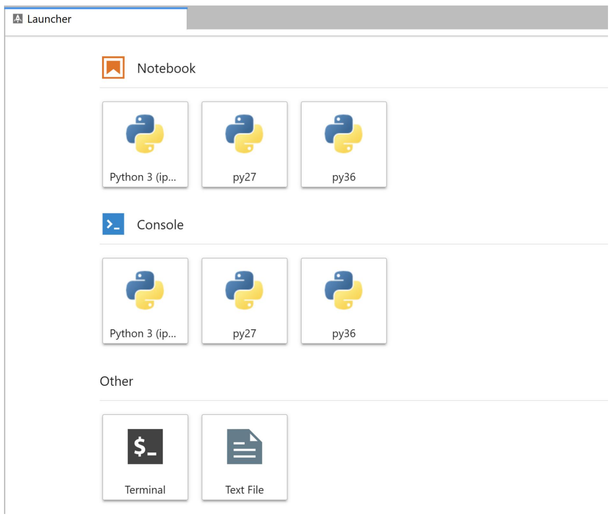
1. 创建独占实例

按照实验指导书的提示，创建并启动实例，进入用户界面。



状态	名称 ↑	应用	剩余时间	资源分配	资源详情	操作	用户模
●	基于Pytorch的SMOKE模型训练	Pytorch(a100-sxm4)	1天1分	已分配	CPU 8核 内存 20 GB GPU 1卡		独占

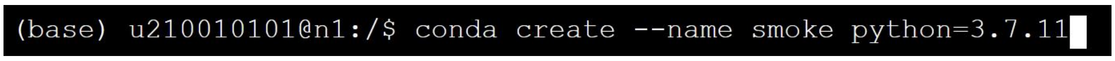
每页行数 25 总数 1，筛选出 1，显示 1-1 条 1



2. 配置 python 环境

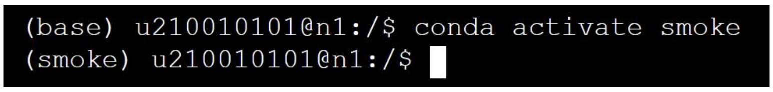
根据指导书的提示，利用命令行配置 python 环境。

(1)创建 smoke 虚拟环境



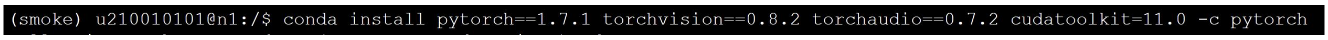
```
(base) u210010101@n1:/$ conda create --name smoke python=3.7.11
```

(2)启动虚拟环境



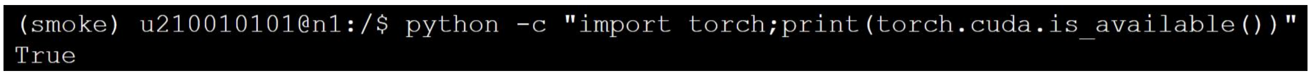
```
(base) u210010101@n1:/$ conda activate smoke
(smoke) u210010101@n1:/$
```

(3)虚拟环境下安装 pytorch



```
(smoke) u210010101@n1:/$ conda install pytorch==1.7.1 torchvision==0.8.2 torchaudio==0.7.2 cudatoolkit=11.0 -c pytorch
```

(4)检查是否安装成功

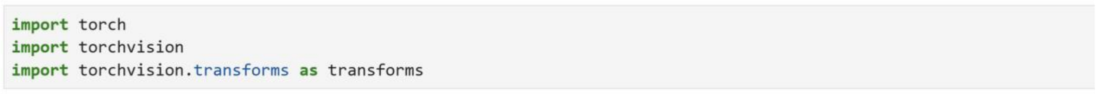


```
(smoke) u210010101@n1:/$ python -c "import torch;print(torch.cuda.is_available())"
True
```

3. 运行深度学习代码

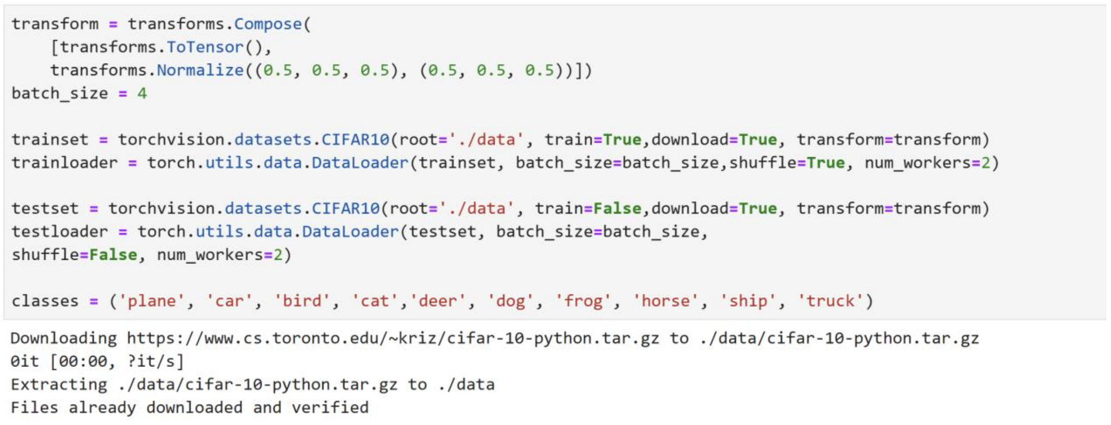
(1)下载实验数据

a.导入必要的包



```
import torch
import torchvision
import torchvision.transforms as transforms
```

b.下载数据，定义训练集和测试集

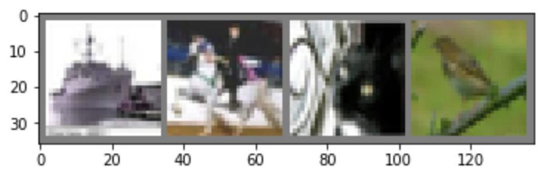


```
transform = transforms.Compose(
    [transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
batch_size = 4

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat','deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
0it [00:00, ?it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

c.查看下载好的数据

```python
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image


def imshow(img):
    img = img / 2 + 0.5 # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(trainloader)
images, labels = next(dataiter)


# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))
```



```
ship  horse cat   bird
```

(2)定义一个卷积神经网络

```python
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

# 使用 GPU 训练
device = torch.device("cuda:0");
net = Net().to(device)
```

(3)定义损失函数和优化器

```python
import torch.optim as optim
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

(4)训练网络

```
for epoch in range(2): # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data[0].to(device), data[1].to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999: # print every 2000 mini-batches
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
            running_loss = 0.0

print('Finished Training')
```

```
[1,  2000] loss: 2.258
[1,  4000] loss: 1.959
[1,  6000] loss: 1.728
[1,  8000] loss: 1.628
[1, 10000] loss: 1.546
[1, 12000] loss: 1.502
[2,  2000] loss: 1.414
[2,  4000] loss: 1.403
[2,  6000] loss: 1.381
[2,  8000] loss: 1.352
[2, 10000] loss: 1.331
[2, 12000] loss: 1.273
Finished Training
```

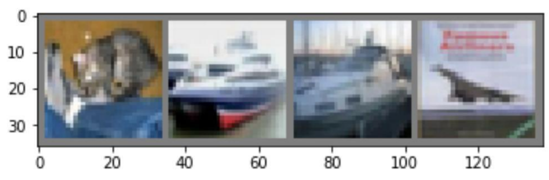
(5) 保存训练过的模型

```
PATH = './try1'
torch.save(net.state_dict(), PATH)
```

(6) 测试网络

a.查看测试集

```
dataiter = iter(testloader)
images, labels = next(dataiter)

imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join(f'{classes[labels[j]]:5s}' for j in range(4)))
```



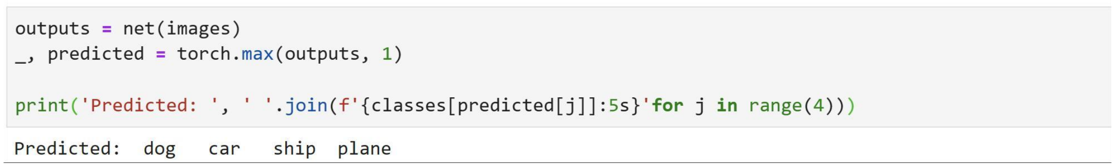
```
GroundTruth:  cat   ship  ship  plane
```

b.加载训练好的模型

```
net = Net()
net.load_state_dict(torch.load(PATH))
```

```
<All keys matched successfully>
```

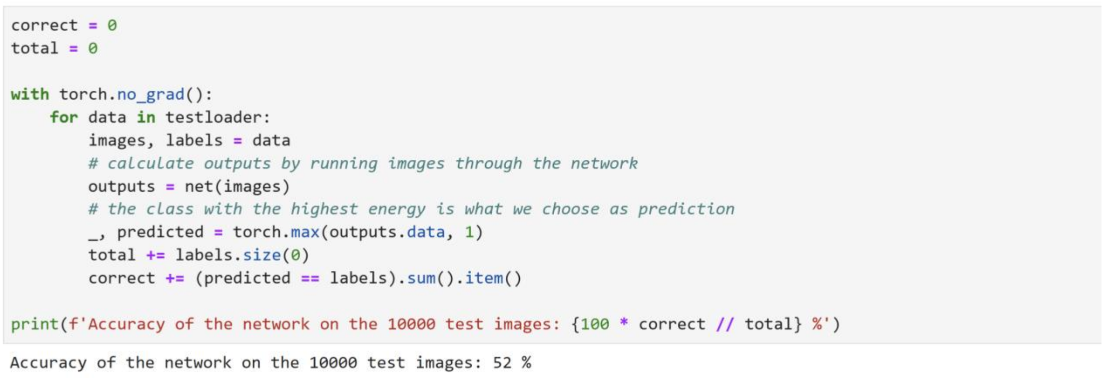
c.查看模型的训练结果



```
outputs = net(images)
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join(f'{classes[predicted[j]]:5s}'for j in range(4)))
```

```
Predicted:  dog   car   ship  plane
```

d.查看训练准确度

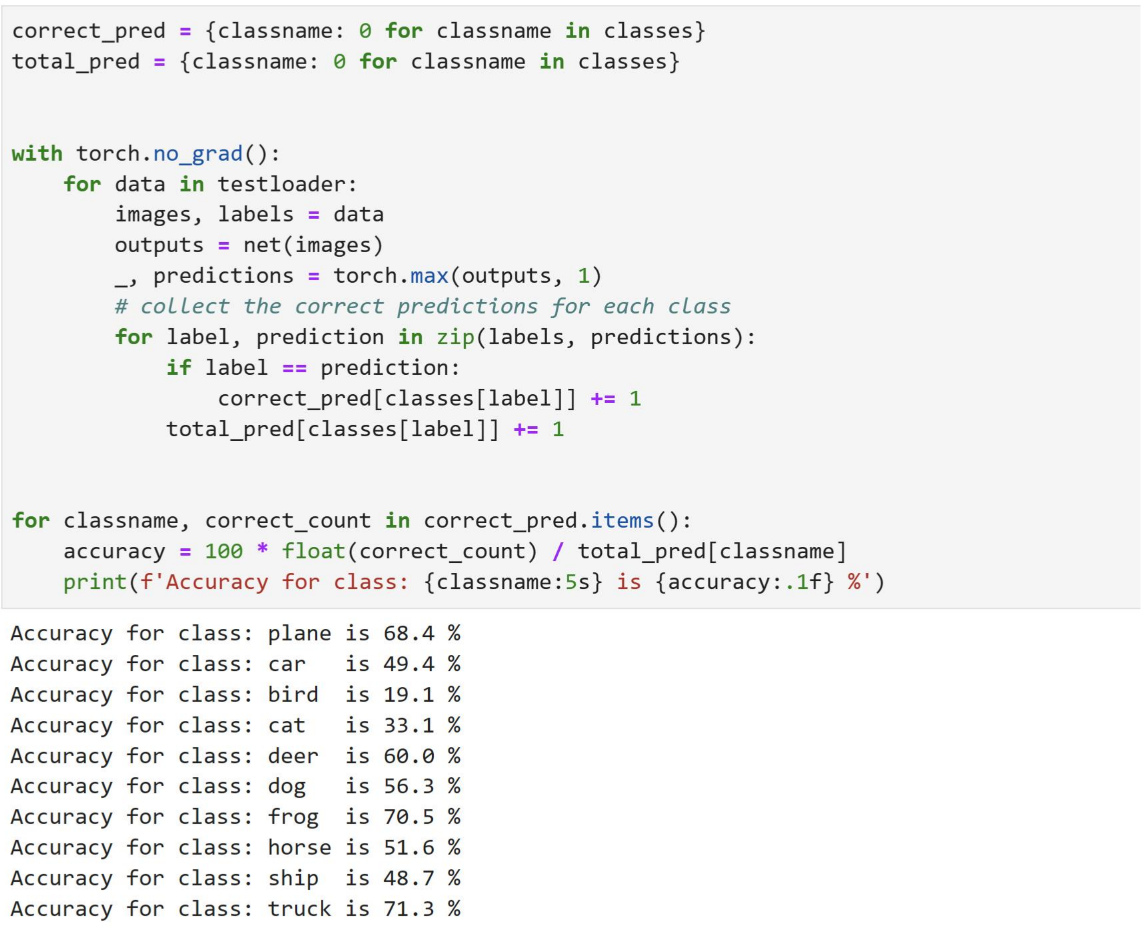


```
correct = 0
total = 0

with torch.no_grad():
    for data in testloader:
        images, labels = data
        # calculate outputs by running images through the network
        outputs = net(images)
        # the class with the highest energy is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 * correct // total} %')
```

```
Accuracy of the network on the 10000 test images: 52 %
```

结果分析：可看到训练准确度为 52%。

e.查看对种类分类的准确度



```
correct_pred = {classname: 0 for classname in classes}
total_pred = {classname: 0 for classname in classes}


with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predictions = torch.max(outputs, 1)
        # collect the correct predictions for each class
        for label, prediction in zip(labels, predictions):
            if label == prediction:
                correct_pred[classes[label]] += 1
            total_pred[classes[label]] += 1


for classname, correct_count in correct_pred.items():
    accuracy = 100 * float(correct_count) / total_pred[classname]
    print(f'Accuracy for class: {classname:5s} is {accuracy:.1f} %')
```

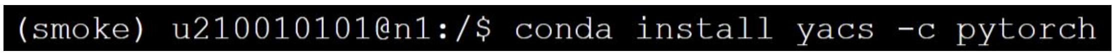
```
Accuracy for class: plane is 68.4 %
Accuracy for class: car   is 49.4 %
Accuracy for class: bird  is 19.1 %
Accuracy for class: cat   is 33.1 %
Accuracy for class: deer  is 60.0 %
Accuracy for class: dog   is 56.3 %
Accuracy for class: frog  is 70.5 %
Accuracy for class: horse is 51.6 %
Accuracy for class: ship  is 48.7 %
Accuracy for class: truck is 71.3 %
```

结果分析：对不同种类分类的准确度有较大差异，其中对 truck 分类的准确度最高，为 71.3%；对 bird 分类的准确度最低，为 19.1%。

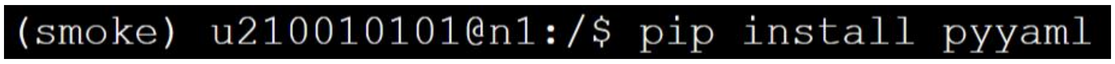
步骤二：SMOKE 环境搭建

由于步骤 2 至 4 已经在实验一完成，故直接进行第 5 步：

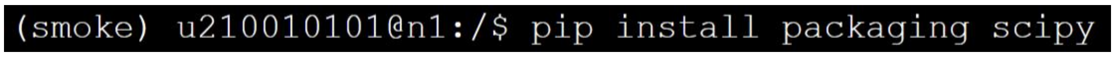
安装运行 SMOKE 模型时需要用到的包



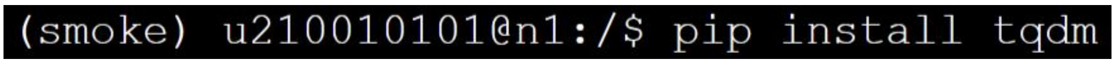
```
(smoke) u210010101@n1:/$ conda install yacs -c pytorch
```



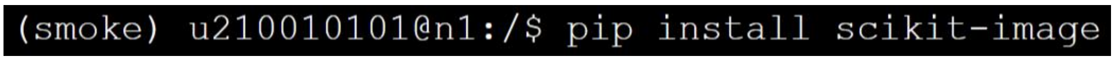
```
(smoke) u210010101@n1:/$ pip install pyyaml
```



```
(smoke) u210010101@n1:/$ pip install packaging scipy
```



```
(smoke) u210010101@n1:/$ pip install tqdm
```



```
(smoke) u210010101@n1:/$ pip install scikit-image
```

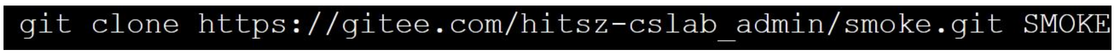
步骤三：运行 SMOKE 模型

1. 下载数据集

由于数据集文件较大，为节省时间，直接在集群的共享目录进行访问。访问路径为：/groups/public_cluster/home/share/dataset/

2. 克隆代码仓库

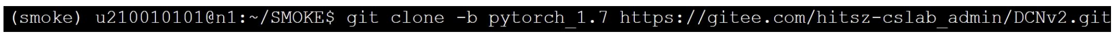
故克隆镜像仓库：



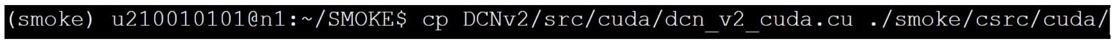
```
git clone https://gitee.com/hitsz-cslab_admin/smoke.git SMOKE
```

3. 修改代码以适配 pytorch1.7.1

在 SMOKE 目录下克隆镜像仓库并运行命令：

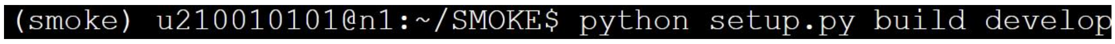


```
(smoke) u210010101@n1:~/SMOKE$ git clone -b pytorch_1.7 https://gitee.com/hitsz-cslab_admin/DCNv2.git
```



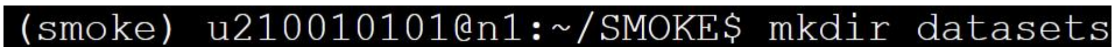
```
(smoke) u210010101@n1:~/SMOKE$ cp DCNv2/src/cuda/dcn_v2_cuda.cu ./smoke/csrc/cuda/
```

4. 编译

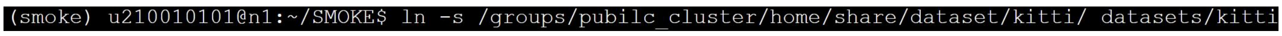


```
(smoke) u210010101@n1:~/SMOKE$ python setup.py build develop
```

5. 链接数据集



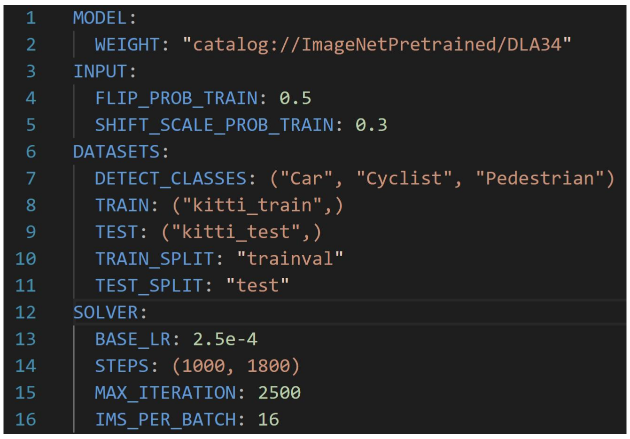
```
(smoke) u210010101@n1:~/SMOKE$ mkdir datasets
```



```
(smoke) u210010101@n1:~/SMOKE$ ln -s /groups/pubilc_cluster/home/share/dataset/kitti/ datasets/kitti
```

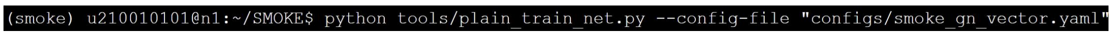
6. 运行

修改 yaml 配置文件：



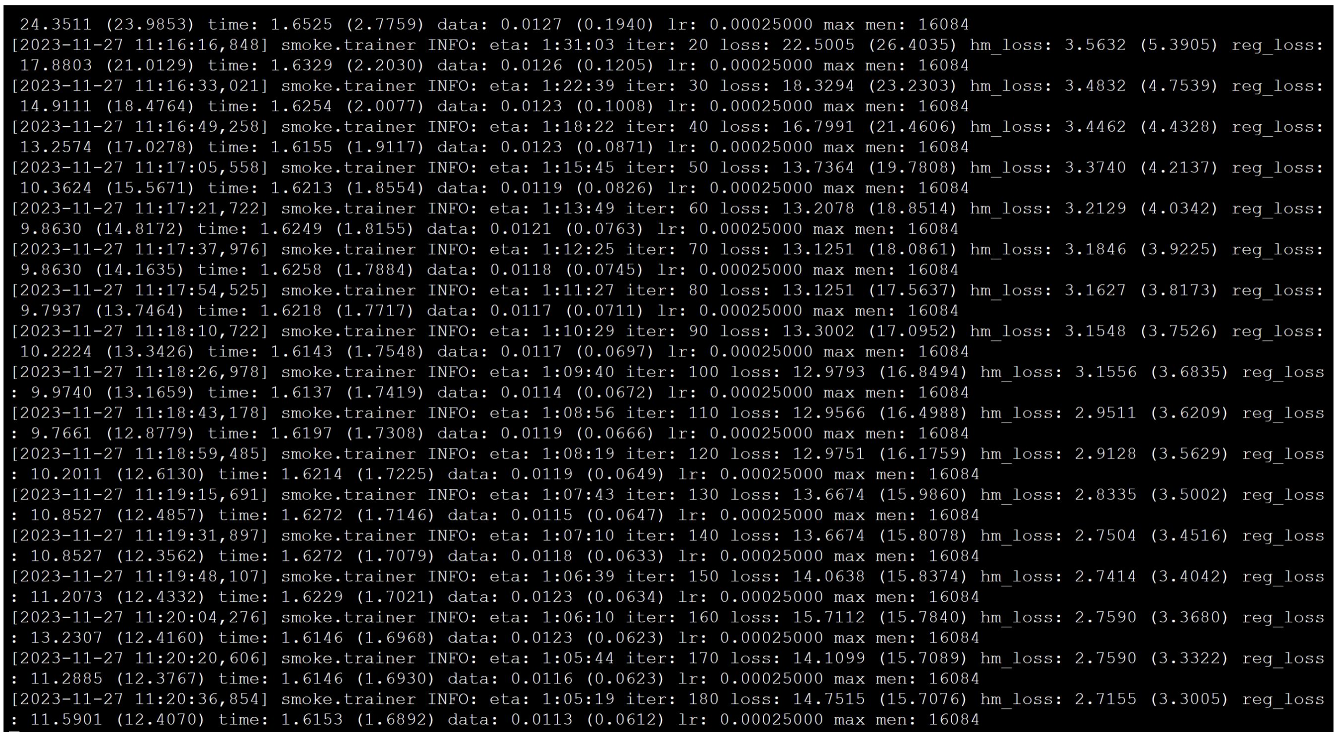
```yaml
MODEL:
  WEIGHT: "catalog://ImageNetPretrained/DLA34"
INPUT:
  FLIP_PROB_TRAIN: 0.5
  SHIFT_SCALE_PROB_TRAIN: 0.3
DATASETS:
  DETECT_CLASSES: ("Car", "Cyclist", "Pedestrian")
  TRAIN: ("kitti_train",)
  TEST: ("kitti_test",)
  TRAIN_SPLIT: "trainval"
  TEST_SPLIT: "test"
SOLVER:
  BASE_LR: 2.5e-4
  STEPS: (1000, 1800)
  MAX_ITERATION: 2500
  IMS_PER_BATCH: 16
```

单 GPU 运行：



```
(smoke) u210010101@n1:~/SMOKE$ python tools/plain_train_net.py --config-file "configs/smoke_gn_vector.yaml"
```

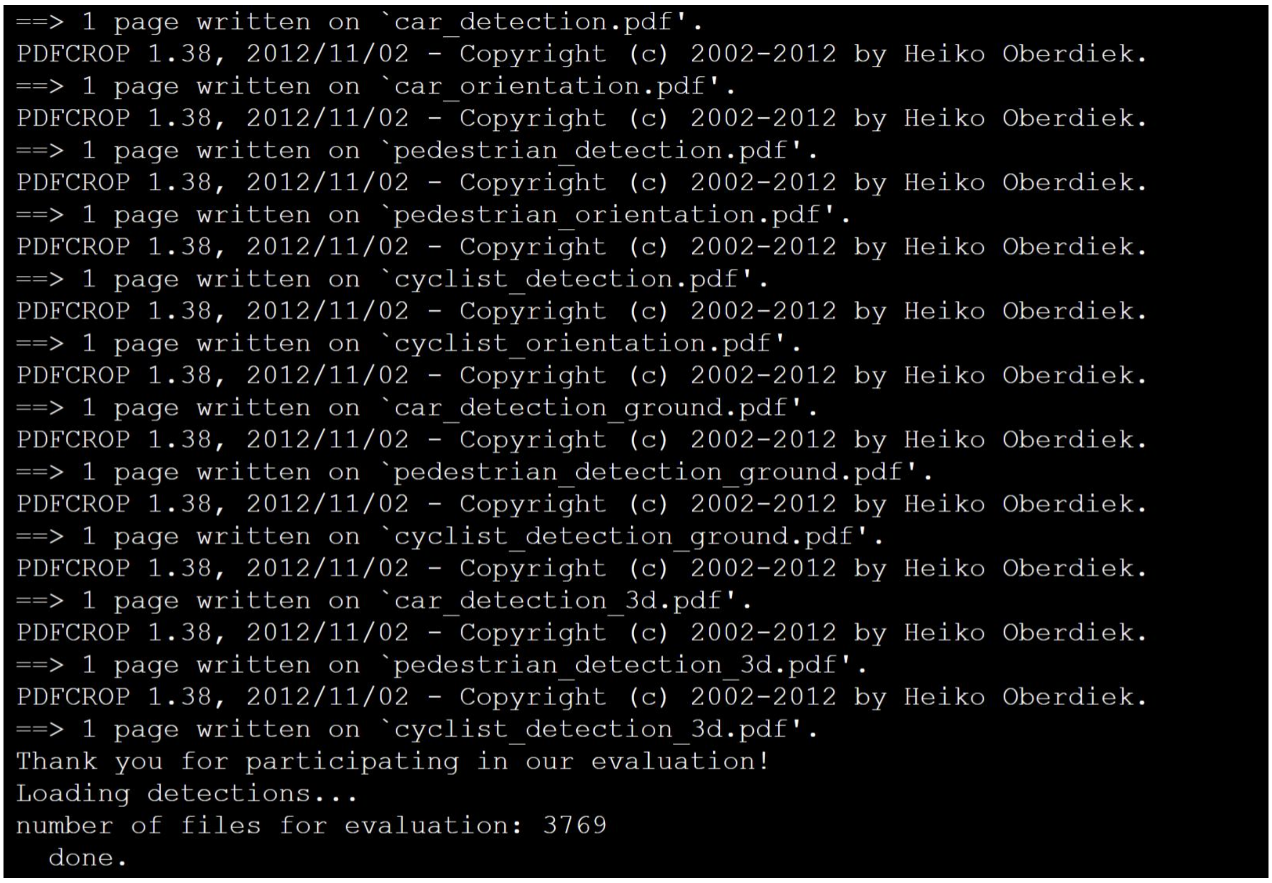
运行过程的部分输出截图如下：



```
 24.3511 (23.9853) time: 1.6525 (2.7759) data: 0.0127 (0.1940) lr: 0.00025000 max men: 16084
[2023-11-27 11:16:16,848] smoke.trainer INFO: eta: 1:31:03 iter: 20 loss: 22.5005 (26.4035) hm_loss: 3.5632 (5.3905) reg_loss:
 17.8803 (21.0129) time: 1.6329 (2.2030) data: 0.0126 (0.1205) lr: 0.00025000 max men: 16084
[2023-11-27 11:16:33,021] smoke.trainer INFO: eta: 1:22:39 iter: 30 loss: 18.3294 (23.2303) hm_loss: 3.4832 (4.7539) reg_loss:
 14.9111 (18.4764) time: 1.6254 (2.0077) data: 0.0123 (0.1008) lr: 0.00025000 max men: 16084
[2023-11-27 11:16:49,258] smoke.trainer INFO: eta: 1:18:22 iter: 40 loss: 16.7991 (21.4606) hm_loss: 3.4462 (4.4328) reg_loss:
 13.2574 (17.0278) time: 1.6155 (1.9117) data: 0.0123 (0.0871) lr: 0.00025000 max men: 16084
[2023-11-27 11:17:05,558] smoke.trainer INFO: eta: 1:15:45 iter: 50 loss: 13.7364 (19.7808) hm_loss: 3.3740 (4.2137) reg_loss:
 10.3624 (15.5671) time: 1.6213 (1.8554) data: 0.0119 (0.0826) lr: 0.00025000 max men: 16084
[2023-11-27 11:17:21,722] smoke.trainer INFO: eta: 1:13:49 iter: 60 loss: 13.2078 (18.8514) hm_loss: 3.2129 (4.0342) reg_loss:
 9.8630 (14.8172) time: 1.6249 (1.8155) data: 0.0121 (0.0763) lr: 0.00025000 max men: 16084
[2023-11-27 11:17:37,976] smoke.trainer INFO: eta: 1:12:25 iter: 70 loss: 13.1251 (18.0861) hm_loss: 3.1846 (3.9225) reg_loss:
 9.8630 (14.1635) time: 1.6258 (1.7884) data: 0.0118 (0.0745) lr: 0.00025000 max men: 16084
[2023-11-27 11:17:54,525] smoke.trainer INFO: eta: 1:11:27 iter: 80 loss: 13.1251 (17.5637) hm_loss: 3.1627 (3.8173) reg_loss:
 9.7937 (13.7464) time: 1.6218 (1.7717) data: 0.0117 (0.0711) lr: 0.00025000 max men: 16084
[2023-11-27 11:18:10,722] smoke.trainer INFO: eta: 1:10:29 iter: 90 loss: 13.3002 (17.0952) hm_loss: 3.1548 (3.7526) reg_loss:
 10.2224 (13.3426) time: 1.6143 (1.7548) data: 0.0117 (0.0697) lr: 0.00025000 max men: 16084
[2023-11-27 11:18:26,978] smoke.trainer INFO: eta: 1:09:40 iter: 100 loss: 12.9793 (16.8494) hm_loss: 3.1556 (3.6835) reg_loss
: 9.9740 (13.1659) time: 1.6137 (1.7419) data: 0.0114 (0.0672) lr: 0.00025000 max men: 16084
[2023-11-27 11:18:43,178] smoke.trainer INFO: eta: 1:08:56 iter: 110 loss: 12.9566 (16.4988) hm_loss: 2.9511 (3.6209) reg_loss
: 9.7661 (12.8779) time: 1.6197 (1.7308) data: 0.0119 (0.0666) lr: 0.00025000 max men: 16084
[2023-11-27 11:18:59,485] smoke.trainer INFO: eta: 1:08:19 iter: 120 loss: 12.9751 (16.1759) hm_loss: 2.9128 (3.5629) reg_loss
: 10.2011 (12.6130) time: 1.6214 (1.7225) data: 0.0119 (0.0649) lr: 0.00025000 max men: 16084
[2023-11-27 11:19:15,691] smoke.trainer INFO: eta: 1:07:43 iter: 130 loss: 13.6674 (15.9860) hm_loss: 2.8335 (3.5002) reg_loss
: 10.8527 (12.4857) time: 1.6272 (1.7146) data: 0.0115 (0.0647) lr: 0.00025000 max men: 16084
[2023-11-27 11:19:31,897] smoke.trainer INFO: eta: 1:07:10 iter: 140 loss: 13.6674 (15.8078) hm_loss: 2.7504 (3.4516) reg_loss
: 10.8527 (12.3562) time: 1.6272 (1.7079) data: 0.0118 (0.0633) lr: 0.00025000 max men: 16084
[2023-11-27 11:19:48,107] smoke.trainer INFO: eta: 1:06:39 iter: 150 loss: 14.0638 (15.8374) hm_loss: 2.7414 (3.4042) reg_loss
: 11.2073 (12.4332) time: 1.6229 (1.7021) data: 0.0123 (0.0634) lr: 0.00025000 max men: 16084
[2023-11-27 11:20:04,276] smoke.trainer INFO: eta: 1:06:10 iter: 160 loss: 15.7112 (15.7840) hm_loss: 2.7590 (3.3680) reg_loss
: 13.2307 (12.4160) time: 1.6146 (1.6968) data: 0.0123 (0.0623) lr: 0.00025000 max men: 16084
[2023-11-27 11:20:20,606] smoke.trainer INFO: eta: 1:05:44 iter: 170 loss: 14.1099 (15.7089) hm_loss: 2.7590 (3.3322) reg_loss
: 11.2885 (12.3767) time: 1.6146 (1.6930) data: 0.0116 (0.0623) lr: 0.00025000 max men: 16084
[2023-11-27 11:20:36,854] smoke.trainer INFO: eta: 1:05:19 iter: 180 loss: 14.7515 (15.7076) hm_loss: 2.7155 (3.3005) reg_loss
: 11.5901 (12.4070) time: 1.6153 (1.6892) data: 0.0113 (0.0612) lr: 0.00025000 max men: 16084
```

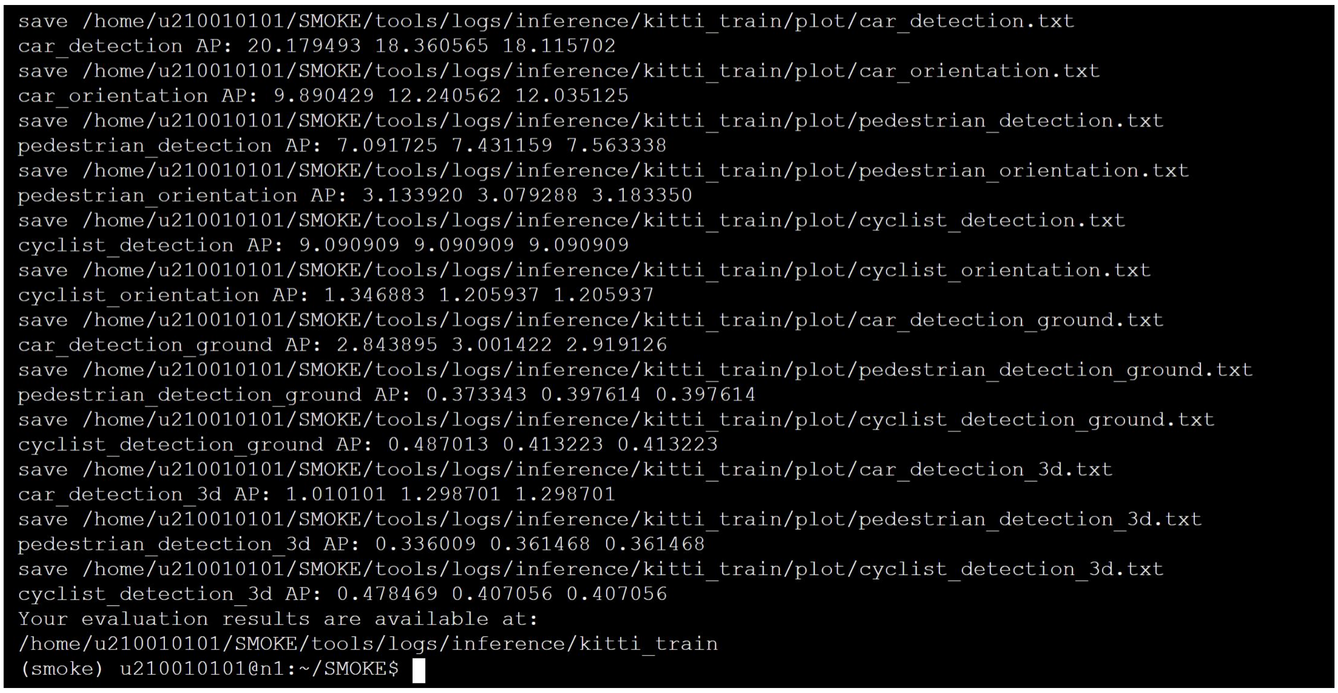
7.进行单 GPU 测试

将测试结果绘图生成 pdf 文件：



```
==> 1 page written on `car_detection.pdf'.
PDFCROP 1.38, 2012/11/02 - Copyright (c) 2002-2012 by Heiko Oberdiek.
==> 1 page written on `car_orientation.pdf'.
PDFCROP 1.38, 2012/11/02 - Copyright (c) 2002-2012 by Heiko Oberdiek.
==> 1 page written on `pedestrian_detection.pdf'.
PDFCROP 1.38, 2012/11/02 - Copyright (c) 2002-2012 by Heiko Oberdiek.
==> 1 page written on `pedestrian_orientation.pdf'.
PDFCROP 1.38, 2012/11/02 - Copyright (c) 2002-2012 by Heiko Oberdiek.
==> 1 page written on `cyclist_detection.pdf'.
PDFCROP 1.38, 2012/11/02 - Copyright (c) 2002-2012 by Heiko Oberdiek.
==> 1 page written on `cyclist_orientation.pdf'.
PDFCROP 1.38, 2012/11/02 - Copyright (c) 2002-2012 by Heiko Oberdiek.
==> 1 page written on `car_detection_ground.pdf'.
PDFCROP 1.38, 2012/11/02 - Copyright (c) 2002-2012 by Heiko Oberdiek.
==> 1 page written on `pedestrian_detection_ground.pdf'.
PDFCROP 1.38, 2012/11/02 - Copyright (c) 2002-2012 by Heiko Oberdiek.
==> 1 page written on `cyclist_detection_ground.pdf'.
PDFCROP 1.38, 2012/11/02 - Copyright (c) 2002-2012 by Heiko Oberdiek.
==> 1 page written on `car_detection_3d.pdf'.
PDFCROP 1.38, 2012/11/02 - Copyright (c) 2002-2012 by Heiko Oberdiek.
==> 1 page written on `pedestrian_detection_3d.pdf'.
PDFCROP 1.38, 2012/11/02 - Copyright (c) 2002-2012 by Heiko Oberdiek.
==> 1 page written on `cyclist_detection_3d.pdf'.
Thank you for participating in our evaluation!
Loading detections...
number of files for evaluation: 3769
  done.
```

保存模型：



```
save /home/u210010101/SMOKE/tools/logs/inference/kitti_train/plot/car_detection.txt
car_detection AP: 20.179493 18.360565 18.115702
save /home/u210010101/SMOKE/tools/logs/inference/kitti_train/plot/car_orientation.txt
car_orientation AP: 9.890429 12.240562 12.035125
save /home/u210010101/SMOKE/tools/logs/inference/kitti_train/plot/pedestrian_detection.txt
pedestrian_detection AP: 7.091725 7.431159 7.563338
save /home/u210010101/SMOKE/tools/logs/inference/kitti_train/plot/pedestrian_orientation.txt
pedestrian_orientation AP: 3.133920 3.079288 3.183350
save /home/u210010101/SMOKE/tools/logs/inference/kitti_train/plot/cyclist_detection.txt
cyclist_detection AP: 9.090909 9.090909 9.090909
save /home/u210010101/SMOKE/tools/logs/inference/kitti_train/plot/cyclist_orientation.txt
cyclist_orientation AP: 1.346883 1.205937 1.205937
save /home/u210010101/SMOKE/tools/logs/inference/kitti_train/plot/car_detection_ground.txt
car_detection_ground AP: 2.843895 3.001422 2.919126
save /home/u210010101/SMOKE/tools/logs/inference/kitti_train/plot/pedestrian_detection_ground.txt
pedestrian_detection_ground AP: 0.373343 0.397614 0.397614
save /home/u210010101/SMOKE/tools/logs/inference/kitti_train/plot/cyclist_detection_ground.txt
cyclist_detection_ground AP: 0.487013 0.413223 0.413223
save /home/u210010101/SMOKE/tools/logs/inference/kitti_train/plot/car_detection_3d.txt
car_detection_3d AP: 1.010101 1.298701 1.298701
save /home/u210010101/SMOKE/tools/logs/inference/kitti_train/plot/pedestrian_detection_3d.txt
pedestrian_detection_3d AP: 0.336009 0.361468 0.361468
save /home/u210010101/SMOKE/tools/logs/inference/kitti_train/plot/cyclist_detection_3d.txt
cyclist_detection_3d AP: 0.478469 0.407056 0.407056
Your evaluation results are available at:
/home/u210010101/SMOKE/tools/logs/inference/kitti_train
(smoke) u210010101@n1:~/SMOKE$
```

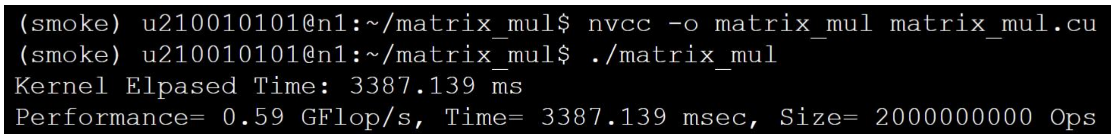
步骤四：运行 CUDA 矩阵乘法样例

1.拷贝矩阵乘法样例

拷贝矩阵乘法样例并进入 matrix_mul 文件夹。

2.编译并运行矩阵乘法样例

结果如下图：

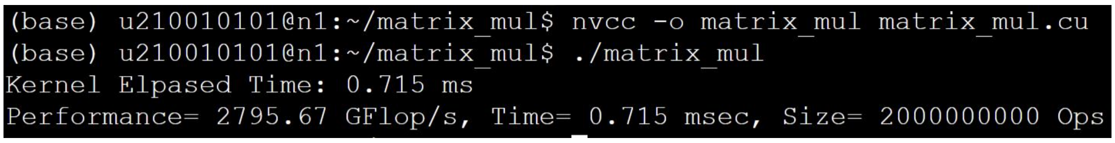


```
(smoke) u210010101@n1:~/matrix_mul$ nvcc -o matrix_mul matrix_mul.cu
(smoke) u210010101@n1:~/matrix_mul$ ./matrix_mul
Kernel Elpased Time: 3387.139 ms
Performance= 0.59 GFlop/s, Time= 3387.139 msec, Size= 2000000000 Ops
```

结果分析：观察 GFlops/s 和 Time 可知 GFlop/s 较小，运算需要的时间较长，故运算能力较差。

3.修改代码

结果如下图：

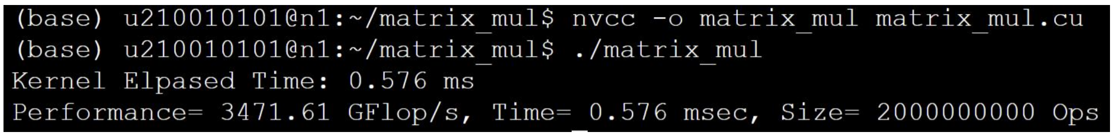


```
(base) u210010101@n1:~/matrix_mul$ nvcc -o matrix_mul matrix_mul.cu
(base) u210010101@n1:~/matrix_mul$ ./matrix_mul
Kernel Elpased Time: 0.715 ms
Performance= 2795.67 GFlop/s, Time= 0.715 msec, Size= 2000000000 Ops
```

结果分析：观察 GFlops/s 和 Time，GFlops/s 与 2 中的 GFlops/s 相比大幅增加，Time 与 2 中的 Time 相比大幅度降低，可知运算性能得到很大的提升。其原因在于初始时使用 CPU 计算实现矩阵乘法，修改代码后则是在 GPU 上完成矩阵乘法运算。

4. 使用 Shared Memory

结果如下图：

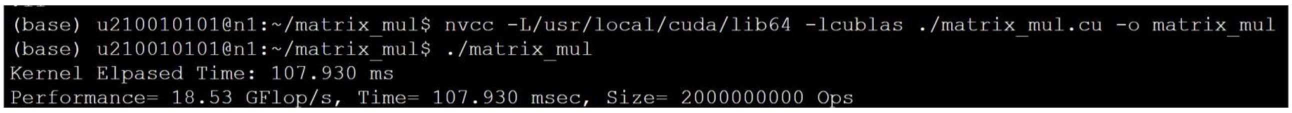


```
(base) u210010101@n1:~/matrix_mul$ nvcc -o matrix_mul matrix_mul.cu
(base) u210010101@n1:~/matrix_mul$ ./matrix_mul
Kernel Elpased Time: 0.576 ms
Performance= 3471.61 GFlop/s, Time= 0.576 msec, Size= 2000000000 Ops
```

结果分析：观察 GFlops/s 和 Time，GFlops/s 与 3 中的 GFlops/s 相比有一定的增加，Time 与 3 中的 Time 相比有一定的降低，可知运算性能得到一定的提升。其原因在于在已经实现 GPU 计算的基础上采用 shared_memory 方式实现矩阵乘法可进一步加快矩阵乘法的运算速度。

5. 使用 CublasSgemm 计算矩阵乘法

结果如下图：

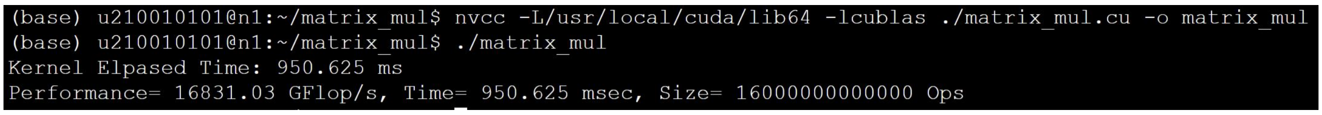


```
(base) u210010101@n1:~/matrix_mul$ nvcc -L/usr/local/cuda/lib64 -lcublas ./matrix_mul.cu -o matrix_mul
(base) u210010101@n1:~/matrix_mul$ ./matrix_mul
Kernel Elpased Time: 107.930 ms
Performance= 18.53 GFlop/s, Time= 107.930 msec, Size= 2000000000 Ops
```

结果分析：观察 GFlops/s 和 Time，发现 GFlops/s 比 2 中的 GFlops/s 大很多但比 3 中的 GFlops/s 小很多，Time 比 2 中的 Time 小很多但比 3 中的 Time 大很多，故性能比 CPU 计算实现矩阵乘法好很多，但比 GPU 计算实现矩阵乘法差很多。由此可知通过 CUDA 的 cublas 库的矩阵相乘函数提升的性能远不如使用 GPU 提升的性能。

6. 观察不同矩阵大小下 CUDA 运算结果和性能

修改矩阵大小为：20000，20000，20000。结果如下图：



```
(base) u210010101@n1:~/matrix_mul$ nvcc -L/usr/local/cuda/lib64 -lcublas ./matrix_mul.cu -o matrix_mul
(base) u210010101@n1:~/matrix_mul$ ./matrix_mul
Kernel Elapsed Time: 950.625 ms
Performance= 16831.03 GFlop/s, Time= 950.625 msec, Size= 16000000000000 Ops
```

结果分析：当矩阵规模较大时，GFlops/s 和 Time 都会较大。由此可知使用同样的方法，在计算规模较大的矩阵时需要更多的时间。

步骤五：运行 CUDA 矩阵乘法与 SMOKE 集成

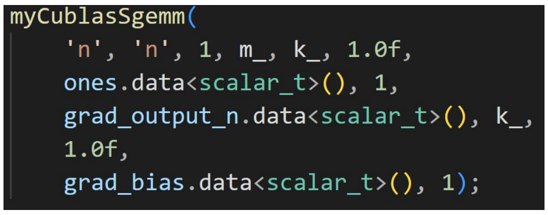
1. 激活在步骤三装好的 smoke 环境

2. 下载 CUDA 矩阵乘法代码到 SMOKE 代码目录中

进入相应文件夹并按照指导书给出的命令，下载 CUDA 矩阵乘法代码到 SMOKE 代码目录中。

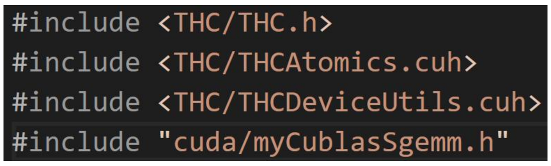
3. 修改 dcn_v2_cuda_backward 函数代码

将 dcn_v2_cuda.cu 文件中的 dcn_v2_cuda_backward 函数在最后一次调用 THCudaBlas_Sgemm 修改为 myCublasSgemm：



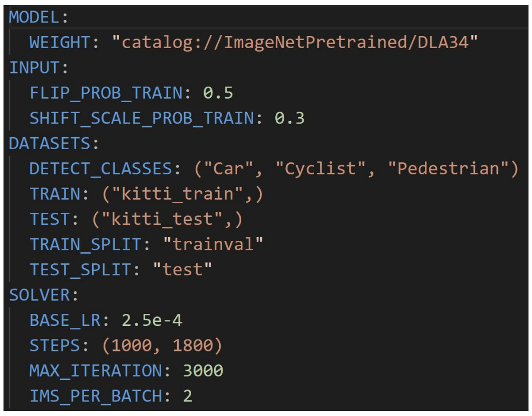
```
myCublasSgemm(
    'n', 'n', 1, m_, k_, 1.0f,
    ones.data<scalar_t>(), 1,
    grad_output_n.data<scalar_t>(), k_,
    1.0f,
    grad_bias.data<scalar_t>(), 1);
```

并且在 dcn_v2_cuda.cu 文件引入头文件：



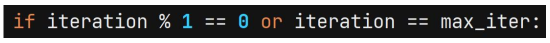
```
#include <THC/THC.h>
#include <THC/THCAtomics.cuh>
#include <THC/THCDeviceUtils.cuh>
#include "cuda/myCublasSgemm.h"
```

4. 修改 yaml 配置文件



```
MODEL:
  WEIGHT: "catalog://ImageNetPretrained/DLA34"
INPUT:
  FLIP_PROB_TRAIN: 0.5
  SHIFT_SCALE_PROB_TRAIN: 0.3
DATASETS:
  DETECT_CLASSES: ("Car", "Cyclist", "Pedestrian")
  TRAIN: ("kitti_train",)
  TEST: ("kitti_test",)
  TRAIN_SPLIT: "trainval"
  TEST_SPLIT: "test"
SOLVER:
  BASE_LR: 2.5e-4
  STEPS: (1000, 1800)
  MAX_ITERATION: 3000
  IMS_PER_BATCH: 2
```

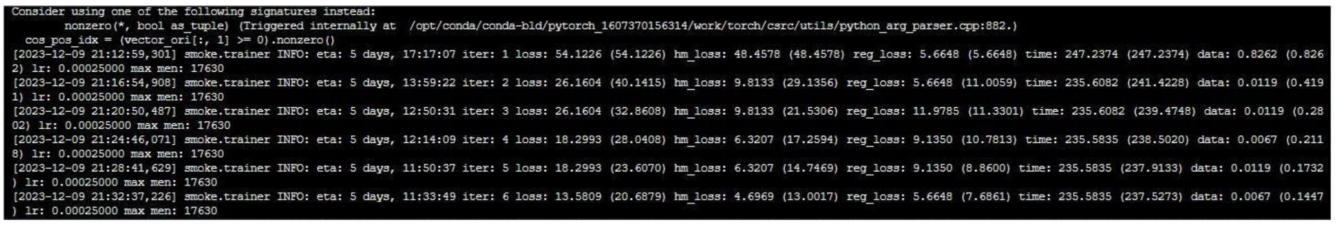
5. 修改 trainer.py 训练文件



```
if iteration % 1 == 0 or iteration == max_iter:
```

6. 重新编译和执行

部分结果截图如下：



```
Consider using one of the following signatures instead:
        nonzero(*, bool as_tuple) (Triggered internally at  /opt/conda/conda-bld/pytorch_1607370156314/work/torch/csrc/utils/python_arg_parser.cpp:882.)
  cos_pos_idx = (vector_ori[:, 1] >= 0).nonzero()
[2023-12-09 21:12:59,301] smoke.trainer INFO: eta: 5 days, 17:17:07 iter: 1 loss: 54.1226 (54.1226) hm_loss: 48.4578 (48.4578) reg_loss: 5.6648 (5.6648) time: 247.2374 (247.2374) data: 0.8262 (0.826
2) lr: 0.00025000 max men: 17630
[2023-12-09 21:16:54,908] smoke.trainer INFO: eta: 5 days, 13:59:22 iter: 2 loss: 26.1604 (40.1415) hm_loss: 9.8133 (29.1356) reg_loss: 5.6648 (11.0059) time: 235.6082 (241.4228) data: 0.0119 (0.419
1) lr: 0.00025000 max men: 17630
[2023-12-09 21:20:50,487] smoke.trainer INFO: eta: 5 days, 12:50:31 iter: 3 loss: 26.1604 (32.8608) hm_loss: 9.8133 (21.5306) reg_loss: 11.9785 (11.3301) time: 235.6082 (239.4748) data: 0.0119 (0.28
02) lr: 0.00025000 max men: 17630
[2023-12-09 21:24:46,071] smoke.trainer INFO: eta: 5 days, 12:14:09 iter: 4 loss: 18.2993 (28.0408) hm_loss: 6.3207 (17.2594) reg_loss: 9.1350 (10.7813) time: 235.5835 (238.5020) data: 0.0067 (0.211
8) lr: 0.00025000 max men: 17630
[2023-12-09 21:28:41,629] smoke.trainer INFO: eta: 5 days, 11:50:37 iter: 5 loss: 18.2993 (23.6070) hm_loss: 6.3207 (14.7469) reg_loss: 9.1350 (8.8600) time: 235.5835 (237.9133) data: 0.0119 (0.1732
) lr: 0.00025000 max men: 17630
[2023-12-09 21:32:37,226] smoke.trainer INFO: eta: 5 days, 11:33:49 iter: 6 loss: 13.5809 (20.6879) hm_loss: 4.6969 (13.0017) reg_loss: 5.6648 (7.6861) time: 235.5835 (237.5273) data: 0.0067 (0.1447
) lr: 0.00025000 max men: 17630
```

结果分析：大约每 2 分钟产生一条输出信息。观察 Time 即每一次迭代所用的时间，发现经过迭代，Time 有所降低，即训练结果速度变快了。

三、实验收获和建议

在完成本实验的过程中，我主要按照指导书的步骤进行实验，通过一步步完成本实验，我获得了 PyTorch 深度学习的基础知识和实践经验，了解了 SMOKE 目标检测算法的核心原理和基于 PyTorch 的模型训练、导出过程。对我来说，本次实验既是对深度学习知识的一次实践，又为进一步深入学习和应用深度学习技术打下坚实的基础。总之，本次实验让我在深度学习方面有了一次入门的实践。

对于本实验，我认为我对实验的代码部分的理解比较欠缺，所以我的建议是可以在各个实验步骤中涉及深度学习代码的部分添加一些说明和注释，以便让同学们对代码有更进一步的理解。