Lec 24 - pytorch - GPU

Statistical Computing and Computation

Sta 663 | Spring 2022

Dr. Colin Rundel

CUDA

> CUDA (or Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) that allows software to use certain types of graphics processing unit (GPU) for general purpose processing, an approach called general-purpose computing on GPUs (GPGPU). CUDA is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels.

Core libraries:

- cuBLAS
- cuSOLVER
- cuSPARSE
- cuFFT
- cuTENSOR
- cuRAND
- Thrust
- cuDNN

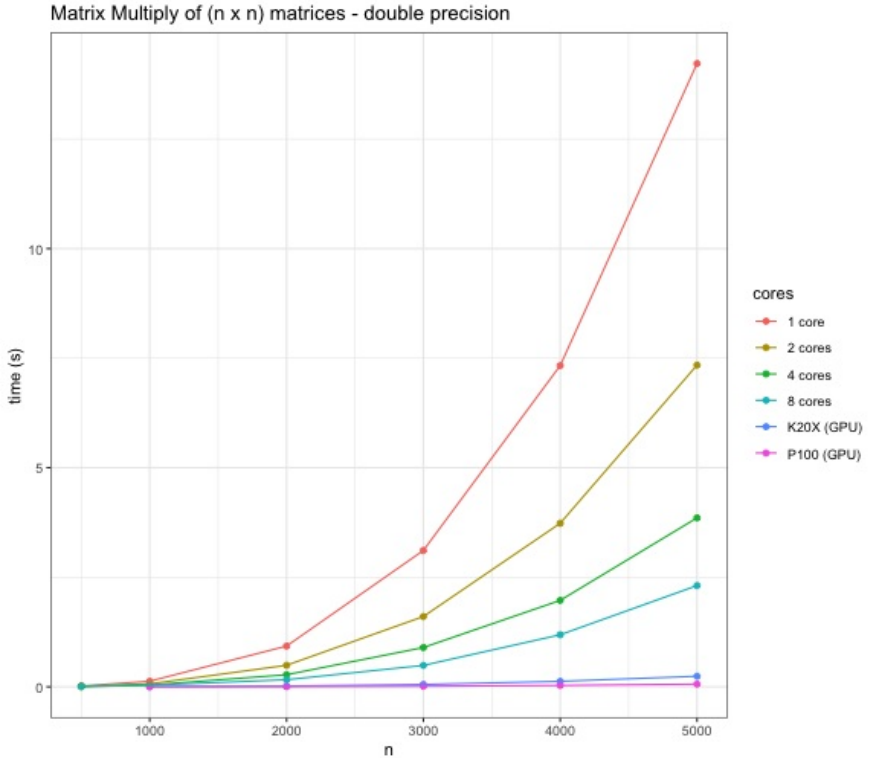
Source

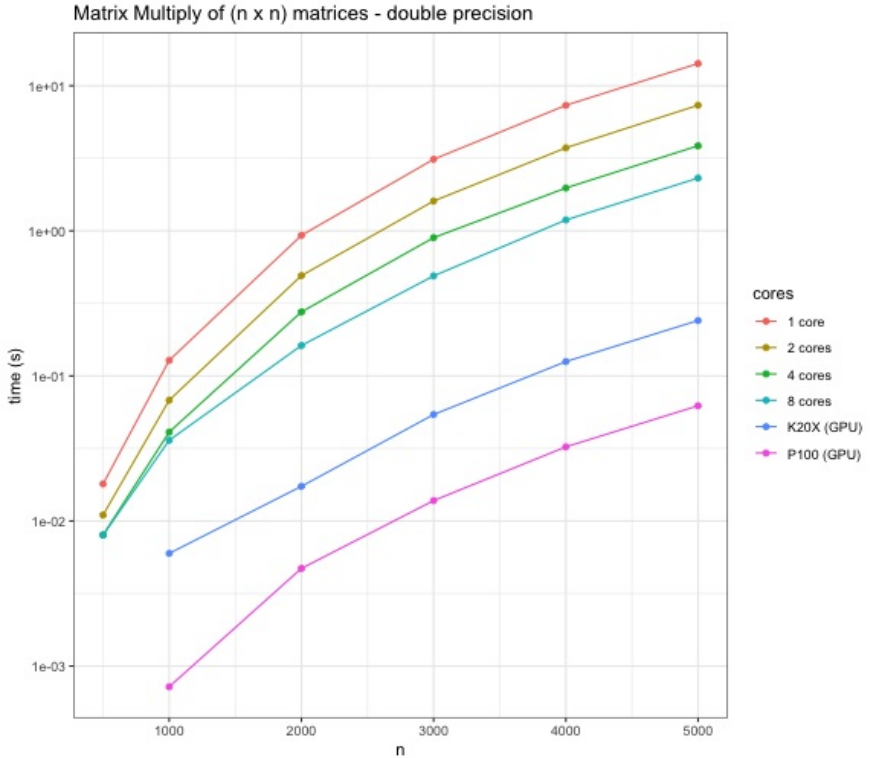
CUDA Kernels

```
// Kernel - Adding two matrices MatA and MatB
__global__ void MatAdd(float MatA[N][N], float MatB[N][N], float MatC[N][N])
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < N && j < N)
        MatC[i][j] = MatA[i][j] + MatB[i][j];
}

int main()
{
    ...
    // Matrix addition kernel launch from host code
    dim3 threadsPerBlock(16, 16);
    dim3 numBlocks(
        (N + threadsPerBlock.x -1) / threadsPerBlock.x,
        (N+threadsPerBlock.y -1) / threadsPerBlock.y
    );

    MatAdd<<<numBlocks, threadsPerBlock>>>(MatA, MatB, MatC);
    ...
}
```


Matrix Multiply of (n x n) matrices - double precision
time (s)
0
5
10
1000
2000
3000
4000
5000
n
cores
1 core
2 cores
4 cores
8 cores
K20X (GPU)
P100 (GPU)


Matrix Multiply of (n x n) matrices - double precision
time (s)
1e+01
1e+00
1e-01
1e-02
1e-03
1000
2000
3000
4000
5000
n
cores
1 core
2 cores
4 cores
8 cores
K20X (GPU)
P100 (GPU)

GPU Status

```
nvidia-smi
```

```
## Wed Apr  6 10:22:09 2022
## +-----------------------------------------------------------------------------+
## | NVIDIA-SMI 470.103.01   Driver Version: 470.103.01   CUDA Version: 11.4     |
## |-------------------------------+----------------------+----------------------+
## | 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 P100-PCIE...  Off  | 00000000:02:00.0 Off |                    0 |
## | N/A   42C    P0    33W / 250W |   1521MiB / 16280MiB |      0%      Default |
## |                               |                      |                  N/A |
## +-------------------------------+----------------------+----------------------+
## |   1  Tesla P100-PCIE...  Off  | 00000000:03:00.0 Off |                    0 |
## | N/A   39C    P0    27W / 250W |      2MiB / 16280MiB |      0%      Default |
## |                               |                      |                  N/A |
## +-------------------------------+----------------------+----------------------+
##
## +-----------------------------------------------------------------------------+
## | Processes:                                                                  |
## |  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
## |        ID   ID                                                   Usage      |
## |=============================================================================|
```

Torch GPU Information

```
torch.cuda.is_available()
```

```
## True
```

```
torch.cuda.device_count()
```

```
## 2
```

```
torch.cuda.get_device_name("cuda:0")
```

```
## 'Tesla P100-PCIE-16GB'
```

```
torch.cuda.get_device_name("cuda:1")
```

```
## 'Tesla P100-PCIE-16GB'
```

```
torch.cuda.get_device_properties(0)
```

```
## _CudaDeviceProperties(name='Tesla P100-PCIE-16GB', major=6, minor=0, total_memory=16280MB, multi_processor_c
```

```
torch.cuda.get_device_properties(1)
```

```
## _CudaDeviceProperties(name='Tesla P100-PCIE-16GB', major=6, minor=0, total_memory=16280MB, multi_processor_c
```

GPU Tensors

Usage of the GPU is governed by the location of the Tensors - to use the GPU we allocate them on the GPU device.

```
cpu = torch.device('cpu')
cuda0 = torch.device('cuda:0')
cuda1 = torch.device('cuda:1')

x = torch.linspace(0,1,5, device=cuda0)
y = torch.randn(5,2, device=cuda0)
z = torch.rand(2,3, device=cpu)
```

```
x
```

```
## tensor([0.0000, 0.2500, 0.5000, 0.7500, 1.0000]
```

```
y
```

```
## tensor([[ 0.6870,  2.4937],
##         [ 0.1939, -0.1403],
##         [ 0.5835, -0.7860],
##         [-0.4810, -0.0132],
##         [-1.8345, -1.3653]], device='cuda:0')
```

```
x @ y
```

```
## tensor([-1.8550, -1.8033], device='cuda:0')
```

```
y @ z
```

```
## RuntimeError: Expected all tensors to be on the same
```

```
y @ z.to(cuda0)
```

```
## tensor([[ 2.6112,  1.5216,  1.7083],
##         [ 0.0228,  0.0241,  0.0287],
##         [-0.2391, -0.1022, -0.1090],
##         [-0.3623, -0.2332, -0.2653],
##         [-2.4942, -1.5211, -1.7181]], device='cuda:0
```

NN Layers + GPU

NN layers (parameters) also need to be assigned to the GPU to be used with GPU tensors,

```
nn = torch.nn.Linear(5,5)
X = torch.randn(10,5).cuda()
```

```
nn(X)
```

```
## RuntimeError: Expected all tensors to be on the same device, but found at least two devices, cpu and cuda:0!
```

```
nn.cuda()(X)
```

```
## tensor([[ 0.5100,  0.7798,  0.8372,  0.3515, -0.1180],
##         [ 0.7318,  0.0803,  0.5940,  0.2220,  0.0618],
##         [-0.4500, -0.4220,  0.8975, -0.3160,  0.4002],
##         [ 0.1245, -0.1168, -0.0518,  0.2124,  0.1446],
##         [ 0.2666, -0.1466,  0.2908, -0.0510,  0.3981],
##         [ 0.2873, -0.0099,  0.3003, -0.1141, -0.0164],
##         [-0.1170, -0.2701,  0.2517,  0.1761, -0.1481],
##         [ 0.3376, -0.1155,  0.0512,  0.2084,  0.0318],
##         [-0.0619, -0.0692,  0.6493,  0.1803, -0.0687],
##         [-0.4949, -1.0181,  0.0118, -0.1059, -0.0487]], device='cuda:0',
##        grad_fn=<AddmmBackward0>)
```

Back to MNIST

Same MNIST data from last time (1x8x8 images),

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

digits = load_digits()
X, y = digits.data, digits.target

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, shuffle=True, random_state=1234
)

X_train = torch.from_numpy(X_train).float()
y_train = torch.from_numpy(y_train)
X_test = torch.from_numpy(X_test).float()
y_test = torch.from_numpy(y_test)
```

To use the GPU for computation we need to copy these tensors to the GPU,

```
X_train_cuda = X_train.to(device=cuda0)
y_train_cuda = y_train.to(device=cuda0)
X_test_cuda = X_test.to(device=cuda0)
y_test_cuda = y_test.to(device=cuda0)
```

Convolutional NN

```
class mnist_conv_model(torch.nn.Module):
    def __init__(self, device):
        super().__init__()
        self.device = torch.device(device)
        self.cnn  = torch.nn.Conv2d(
          in_channels=1, out_channels=8,
          kernel_size=3, stride=1, padding=1
        ).to(device=self.device)
        self.relu = torch.nn.ReLU().to(device=self.device)
        self.pool = torch.nn.MaxPool2d(kernel_size=2).to(device=self.device)
        self.lin  = torch.nn.Linear(8 * 4 * 4, 10).to(device=self.device)

    def forward(self, X):
        out = self.cnn(X.view(-1, 1, 8, 8))
        out = self.relu(out)
        out = self.pool(out)
        out = self.lin(out.view(-1, 8 * 4 * 4))
        return out

    def fit(self, X, y, lr=0.001, n=1000, acc_step=10):
      opt = torch.optim.SGD(self.parameters(), lr=lr, momentum=0.9)
      losses = []
      for i in range(n):
          opt.zero_grad()
```

CPU vs Cuda

```
m = mnist_conv_model(device="cpu")
loss = m.fit(X_train, y_train, n=1000)
loss[-5:]
```

```
## [0.04613681882619858, 0.046090248972177505, 0.0
```

```
m.accuracy(X_test, y_test)
```

```
## tensor(0.9778)
```

```
m_cuda = mnist_conv_model(device="cuda")
loss = m_cuda.fit(X_train_cuda, y_train_cuda, n=1
loss[-5:]
```

```
## [0.036959268152713776, 0.036920323967933655, 0.03688
```

```
m_cuda.accuracy(X_test_cuda, y_test_cuda)
```

```
## tensor(0.9750, device='cuda:0')
```

```
m_cuda = mnist_conv_model(device="cuda")

start = torch.cuda.Event(enable_timing=True)
end = torch.cuda.Event(enable_timing=True)

start.record()
loss = m_cuda.fit(X_train_cuda, y_train_cuda, n=1000)
end.record()

torch.cuda.synchronize()
print(start.elapsed_time(end))
```

```
## 2772.14794921875
```

```
m = mnist_conv_model(device="cpu")

start = torch.cuda.Event(enable_timing=True)
end = torch.cuda.Event(enable_timing=True)

start.record()
loss = m.fit(X_train, y_train, n=1000)
end.record()

torch.cuda.synchronize()
print(start.elapsed_time(end))
```

```
## 8505.6484375
```

CPU vs GPU Profiles

```
m_cuda = mnist_conv_model(device="cuda")
with torch.autograd.profiler.profile(with_stack=True) as prof_cuda:
    tmp = m_cuda(X_train_cuda)
```

```
print(prof_cuda.key_averages().table(sort_by='self_cpu_time_total', row_limit=5))
```

```
## ---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
##                               Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    # of Calls
## ---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
##           aten::cudnn_convolution        72.08%     821.000us        76.21%     868.000us     868.000us             1
##                 cudaLaunchKernel         6.06%      69.000us         6.06%      69.000us       9.857us             7
##                      aten::addmm         3.60%      41.000us         5.27%      60.000us      60.000us             1
##                   aten::clamp_min         2.19%      25.000us         6.50%      74.000us      37.000us             2
##                        aten::add_         2.11%      24.000us         2.63%      30.000us      30.000us             1
## ---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
## Self CPU time total: 1.139ms
```

```
m = mnist_conv_model(device="cpu")
with torch.autograd.profiler.profile(with_stack=True, profile_memory=True) as prof_cpu:
    tmp = m(X_train)
```

```
print(prof_cpu.key_averages().table(sort_by='self_cpu_time_total', row_limit=5))
```

```
## ---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  -------
##                               Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg       CPU Mem  Self CPU Mem    # of
## ---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  -------
##         aten::mkldnn_convolution        41.88%       3.086ms        42.05%       3.098ms       3.098ms       2.81 Mb           0 b
##    aten::max_pool2d_with_indices        41.86%       3.084ms        41.86%       3.084ms       3.084ms       2.10 Mb       2.10 Mb
##                   aten::clamp_min        13.15%     969.000us        26.25%       1.934ms     967.000us       5.61 Mb       2.81 Mb
```

CIFAR10

Loading the data

```
import torchvision

training_data = torchvision.datasets.CIFAR10(
    root="/data",
    train=True,
    download=True,
    transform=torchvision.transforms.ToTensor()
)
```

```
## Files already downloaded and verified
```

```
test_data = torchvision.datasets.CIFAR10(
    root="/data",
    train=False,
    download=True,
    transform=torchvision.transforms.ToTensor()
)
```

```
## Files already downloaded and verified
```

CIFAR10 data

```
training_data.classes
```

```
## ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

```
training_data.data.shape
```

```
## (50000, 32, 32, 3)
```

```
test_data.data.shape
```

```
## (10000, 32, 32, 3)
```

```
training_data[0]
```

```
## (tensor([[[0.2314, 0.1686, 0.1961,  ..., 0.6196, 0.5961, 0.5804],
##          [0.0627, 0.0000, 0.0706,  ..., 0.4824, 0.4667, 0.4784],
##          [0.0980, 0.0627, 0.1922,  ..., 0.4627, 0.4706, 0.4275],
##          ...,
##          [0.8157, 0.7882, 0.7765,  ..., 0.6275, 0.2196, 0.2078],
##          [0.7059, 0.6784, 0.7294,  ..., 0.7216, 0.3804, 0.3255],
##          [0.6941, 0.6588, 0.7020,  ..., 0.8471, 0.5922, 0.4824]],
##
##         [[0.2431, 0.1804, 0.1882,  ..., 0.5176, 0.4902, 0.4863],
```


frog
truck
truck
deer
automobile
automobile
bird
horse
ship
cat
deer
horse
horse
bird
truck
truck
truck
cat
bird
frog
deer
cat
frog
frog

Data Loaders

```
batch_size = 100

training_loader = torch.utils.data.DataLoader(
    training_data,
    batch_size=batch_size,
    shuffle=True,
    num_workers=4,
    pin_memory=True
)

test_loader = torch.utils.data.DataLoader(
    test_data,
    batch_size=batch_size,
    shuffle=True,
    num_workers=4,
    pin_memory=True
)
```

```
training_loader
```

```
## <torch.utils.data.dataloader.DataLoader object at 0x7f586c2e8280>
```

```
X, y = next(iter(training_loader))
```

```
class cifar_conv_model(torch.nn.Module):
    def __init__(self, device):
        super().__init__()
        self.device = torch.device(device)
        self.model = torch.nn.Sequential(
            torch.nn.Conv2d(3, 6, kernel_size=5),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2, 2),
            torch.nn.Conv2d(6, 16, kernel_size=5),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2, 2),
            torch.nn.Flatten(),
            torch.nn.Linear(16 * 5 * 5, 120),
            torch.nn.ReLU(),
            torch.nn.Linear(120, 84),
            torch.nn.ReLU(),
            torch.nn.Linear(84, 10)
        ).to(device=self.device)

    def forward(self, X):
        return self.model(X)

    def fit(self, loader, epochs=10, n_report=250, lr=0.001):
        opt = torch.optim.SGD(self.parameters(), lr=lr, momentum=0.9)

        for epoch in range(epochs):
            running_loss = 0.0
            for i, (X, y) in enumerate(loader):
                X, y = X.to(self.device), y.to(self.device)
                opt.zero_grad()
                loss = torch.nn.CrossEntropyLoss()(self(X), y)
                loss.backward()
                opt.step()
                # print statistics
```

Based on source

Forward step performance

```
m_cuda = cifar_conv_model(device="cuda")
X, y = next(iter(training_loader))
with torch.autograd.profiler.profile(with_stack=True) as prof_cuda:
    X, y = X.to(device="cuda"), y.to(device="cuda")
    tmp = m_cuda(X)
```

```
print(prof_cuda.key_averages().table(sort_by='self_cpu_time_total', row_limit=5))
```

```
## ---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
##                               Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    # of Calls
## ---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
##                          aten::to        38.77%     725.000us        53.21%     995.000us     497.500us             2
##                 cudaLaunchKernel         8.56%     160.000us         8.56%     160.000us       8.889us            18
##           aten::cudnn_convolution         6.63%     124.000us        10.37%     194.000us      97.000us             2
##             cudaStreamSynchronize         6.26%     117.000us         6.26%     117.000us      58.500us             2
##                        aten::addmm         6.26%     117.000us         9.14%     171.000us      57.000us             3
## ---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
## Self CPU time total: 1.870ms
```

```
m_cpu = cifar_conv_model(device="cpu")
X, y = next(iter(training_loader))
with torch.autograd.profiler.profile(with_stack=True) as prof_cpu:
    tmp = m_cpu(X)
```

```
print(prof_cpu.key_averages().table(sort_by='self_cpu_time_total', row_limit=5))
```

```
## ---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
##                               Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    # of Calls
## ---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
```

Fit - 1 epoch

```
m_cuda = cifar_conv_model(device="cuda")
with torch.autograd.profiler.profile(with_stack=True) as prof_cuda:
    m_cuda.fit(loader=training_loader, epochs=1, n_report=501)
```

```
print(prof_cuda.key_averages().table(sort_by='self_cpu_time_total', row_limit=5))
```

```
## -------------------------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
##                                                   Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    # of Calls
## -------------------------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
##                                         cudaLaunchKernel        15.50%     619.786ms        15.50%     619.786ms      12.790us         48460
##                                  Optimizer.step#SGD.step        11.77%     470.570ms        22.73%     908.671ms       1.817ms           500
## enumerate(DataLoader)#_MultiProcessingDataLoaderIter...         8.34%     333.381ms         8.43%     337.105ms     672.864us           501
##                                              aten::add_         7.75%     309.836ms        12.29%     491.312ms      30.745us         15980
##                           Optimizer.zero_grad#SGD.zero_grad         2.99%     119.458ms         7.02%     280.481ms     560.962us           500
## -------------------------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
## Self CPU time total: 3.998s
```

```
m_cpu = cifar_conv_model(device="cpu")
with torch.autograd.profiler.profile(with_stack=True) as prof_cpu:
    m_cpu.fit(loader=training_loader, epochs=1, n_report=501)
```

```
print(prof_cpu.key_averages().table(sort_by='self_cpu_time_total', row_limit=5))
```

```
## -------------------------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
##                                                   Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    # of Calls
## -------------------------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
##                                 aten::mkldnn_convolution        31.96%       2.457s        32.11%       2.468s       2.468ms          1000
##                               aten::convolution_backward        29.23%       2.247s        29.31%       2.253s       2.253ms          1000
##                            aten::max_pool2d_with_indices        14.50%       1.114s        14.50%       1.114s       1.114ms          1000
```

Fit - 2 epochs

```
m_cuda = cifar_conv_model(device="cuda")
with torch.autograd.profiler.profile(with_stack=True) as prof_cuda:
    m_cuda.fit(loader=training_loader, epochs=2, n_report=501)
```

```
print(prof_cuda.key_averages().table(sort_by='self_cpu_time_total', row_limit=5))
```

```
## -------------------------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
##                                                   Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    # of Calls
## -------------------------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
##                                        cudaLaunchKernel        14.73%        1.221s        14.73%        1.221s      12.593us         96960
##                                  Optimizer.step#SGD.step        11.35%      940.821ms        21.78%        1.805s       1.805ms          1000
## enumerate(DataLoader)#_MultiProcessingDataLoaderIter...         9.87%      818.526ms        10.02%      830.834ms      829.176us          1002
##                                             aten::add_         7.30%      604.797ms        11.52%      955.289ms       29.871us         31980
##                          Optimizer.zero_grad#SGD.zero_grad         3.02%      250.166ms         7.08%      586.920ms      586.920us          1000
## -------------------------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
## Self CPU time total: 8.289s
```

```
m_cpu = cifar_conv_model(device="cpu")
with torch.autograd.profiler.profile(with_stack=True) as prof_cpu:
    m_cpu.fit(loader=training_loader, epochs=2, n_report=501)
```

```
print(prof_cpu.key_averages().table(sort_by='self_cpu_time_total', row_limit=5))
```

```
## -------------------------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
##                                                   Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    # of Calls
## -------------------------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
##                                aten::mkldnn_convolution        29.41%        4.998s        29.61%        5.032s       2.516ms          2000
##                              aten::convolution_backward        28.05%        4.766s        28.19%        4.790s       2.395ms          2000
##                             aten::max_pool2d_with_indices        15.23%        2.588s        15.23%        2.588s       1.294ms          2000
```

Loaders & Accuracy

```
def accuracy(model, loader, device):
    total, correct = 0, 0
    with torch.no_grad():
        for X, y in loader:
            X, y = X.to(device=device), y.to(device=device)
            pred = model(X)
            # the class with the highest energy is what we choose as prediction
            val, idx = torch.max(pred, 1)
            total += pred.size(0)
            correct += (idx == y).sum().item()

    return correct / total
```

Model fitting

```
m = cifar_conv_model("cuda")
m.fit(training_loader, epochs=10, n_report=500, lr=0.01)

## [Epoch 1, Minibatch  500] loss: 2.098
## [Epoch 2, Minibatch  500] loss: 1.692
## [Epoch 3, Minibatch  500] loss: 1.482
## [Epoch 4, Minibatch  500] loss: 1.374
## [Epoch 5, Minibatch  500] loss: 1.292
## [Epoch 6, Minibatch  500] loss: 1.226
## [Epoch 7, Minibatch  500] loss: 1.173
## [Epoch 8, Minibatch  500] loss: 1.117
## [Epoch 9, Minibatch  500] loss: 1.071
## [Epoch 10, Minibatch  500] loss: 1.035
```

```
accuracy(m, training_loader, "cuda")
## 0.63444

accuracy(m, test_loader, "cuda")
## 0.572
```

More epochs

If we use fit with the existing model we continue fitting,

```
m.fit(training_loader, epochs=10, n_report=500)

## [Epoch 1, Minibatch  500] loss: 0.885
## [Epoch 2, Minibatch  500] loss: 0.853
## [Epoch 3, Minibatch  500] loss: 0.839
## [Epoch 4, Minibatch  500] loss: 0.828
## [Epoch 5, Minibatch  500] loss: 0.817
## [Epoch 6, Minibatch  500] loss: 0.806
## [Epoch 7, Minibatch  500] loss: 0.798
## [Epoch 8, Minibatch  500] loss: 0.787
## [Epoch 9, Minibatch  500] loss: 0.780
## [Epoch 10, Minibatch  500] loss: 0.773
```

```
accuracy(m, training_loader, "cuda")
## 0.73914
accuracy(m, test_loader, "cuda")
## 0.624
```

More epochs (again)

```
m.fit(training_loader, epochs=10, n_report=500)

## [Epoch 1, Minibatch  500] loss: 0.764
## [Epoch 2, Minibatch  500] loss: 0.756
## [Epoch 3, Minibatch  500] loss: 0.748
## [Epoch 4, Minibatch  500] loss: 0.739
## [Epoch 5, Minibatch  500] loss: 0.733
## [Epoch 6, Minibatch  500] loss: 0.726
## [Epoch 7, Minibatch  500] loss: 0.718
## [Epoch 8, Minibatch  500] loss: 0.710
## [Epoch 9, Minibatch  500] loss: 0.702
## [Epoch 10, Minibatch  500] loss: 0.698
```

```
accuracy(m, training_loader, "cuda")
## 0.76438
accuracy(m, test_loader, "cuda")
## 0.6217
```

The VGG16 model

```
class VGG16(torch.nn.Module):
    def __init__(self, device):
        super().__init__()
        self.device = torch.device(device)
        self.model = self.make_layers()

    def forward(self, X):
        return self.model(X)

    def make_layers(self):
        cfg = [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, 512, 'M']
        layers = []
        in_channels = 3
        for x in cfg:
            if x == 'M':
                layers += [torch.nn.MaxPool2d(kernel_size=2, stride=2)]
            else:
                layers += [torch.nn.Conv2d(in_channels, x, kernel_size=3, padding=1),
                           torch.nn.BatchNorm2d(x),
                           torch.nn.ReLU(inplace=True)]
                in_channels = x
        layers += [
            torch.nn.AvgPool2d(kernel_size=1, stride=1),
            torch.nn.Flatten(),
```

Based on code from pytorch-cifar, original paper

```
VGG16("cuda").model
```

```
## Sequential(
##   (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
##   (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
##   (2): ReLU(inplace=True)
##   (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
##   (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
##   (5): ReLU(inplace=True)
##   (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
##   (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
##   (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
##   (9): ReLU(inplace=True)
##   (10): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
##   (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
##   (12): ReLU(inplace=True)
##   (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
##   (14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
##   (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
##   (16): ReLU(inplace=True)
##   (17): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
##   (18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
##   (19): ReLU(inplace=True)
##   (20): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
##   (21): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
##   (22): ReLU(inplace=True)
##   (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
##   (24): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
##   (25): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
##   (26): ReLU(inplace=True)
##   (27): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
##   (28): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
##   (29): ReLU(inplace=True)
##   (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
##   (31): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

Minibatch performance

```
m_cuda = VGG16(device="cuda")
X, y = next(iter(training_loader))
with torch.autograd.profiler.profile(with_stack=True) as prof_cuda:
    X, y = X.to(device="cuda"), y.to(device="cuda")
    tmp = m_cuda(X)
```

```
print(prof_cuda.key_averages().table(sort_by='self_cpu_time_total', row_limit=5))
```

```
## ---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
##                               Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    # of Calls
## ---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
##                          aten::to        69.03%      51.397ms        69.38%      51.653ms      25.826ms             2
##            aten::cudnn_batch_norm        12.10%       9.007ms        15.14%      11.273ms     867.154us            13
##                        cudaMalloc        11.64%       8.664ms        11.64%       8.664ms     866.400us            10
##     aten::max_pool2d_with_indices         1.41%       1.050ms         2.93%       2.182ms     436.400us             5
##                aten::_convolution         1.20%     897.000us        11.12%       8.282ms     637.077us            13
## ---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
## Self CPU time total: 74.451ms
```

```
m_cpu = VGG16(device="cpu")
X, y = next(iter(training_loader))
with torch.autograd.profiler.profile(with_stack=True) as prof_cpu:
    tmp = m_cpu(X)
```

```
print(prof_cpu.key_averages().table(sort_by='self_cpu_time_total', row_limit=5))
```

```
## ---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
##                               Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    # of Calls
## ---------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
```

Fitting

```
def fit(model, loader, epochs=10, n_report=250, lr = 0.01):
    opt = torch.optim.SGD(model.parameters(), lr=lr, momentum=0.9)

    for epoch in range(epochs):
        running_loss = 0.0
        for i, (X, y) in enumerate(loader):
            X, y = X.to(model.device), y.to(model.device)
            opt.zero_grad()
            loss = torch.nn.CrossEntropyLoss()(model(X), y)
            loss.backward()
            opt.step()

            running_loss += loss.item()
            if i % n_report == (n_report-1):
                print(f'[Epoch {epoch + 1}, Minibatch {i + 1:4d}] loss: {running_loss / n_report:.3f}')
                running_loss = 0.0
```

lr = 0.01

```
m = VGG16(device="cuda")
fit(m, training_loader, epochs=10, n_report=500, lr=0.01)

## [Epoch 1, Minibatch  500] loss: 1.345
## [Epoch 2, Minibatch  500] loss: 0.790
## [Epoch 3, Minibatch  500] loss: 0.577
## [Epoch 4, Minibatch  500] loss: 0.445
## [Epoch 5, Minibatch  500] loss: 0.350
## [Epoch 6, Minibatch  500] loss: 0.274
## [Epoch 7, Minibatch  500] loss: 0.215
## [Epoch 8, Minibatch  500] loss: 0.167
## [Epoch 9, Minibatch  500] loss: 0.127
## [Epoch 10, Minibatch  500] loss: 0.103
```

```
accuracy(model=m, loader=training_loader, device="cuda")
## 0.97008
accuracy(model=m, loader=test_loader, device="cuda")
## 0.8318
```

lr = 0.001

```
m = VGG16(device="cuda")
fit(m, training_loader, epochs=10, n_report=500, lr=0.001)

## [Epoch 1, Minibatch  500] loss: 1.279
## [Epoch 2, Minibatch  500] loss: 0.827
## [Epoch 3, Minibatch  500] loss: 0.599
## [Epoch 4, Minibatch  500] loss: 0.428
## [Epoch 5, Minibatch  500] loss: 0.303
## [Epoch 6, Minibatch  500] loss: 0.210
## [Epoch 7, Minibatch  500] loss: 0.144
## [Epoch 8, Minibatch  500] loss: 0.108
## [Epoch 9, Minibatch  500] loss: 0.088
## [Epoch 10, Minibatch  500] loss: 0.063
```

```
accuracy(model=m, loader=training_loader, device="cuda")
## 0.9815
accuracy(model=m, loader=test_loader, device="cuda")
## 0.7816
```

Report

```
from sklearn.metrics import classification_report

def report(model, loader, device):
    y_true, y_pred = [], []
    with torch.no_grad():
        for X, y in loader:
            X = X.to(device=device)
            y_true.append( y.cpu().numpy() )
            y_pred.append( model(X).max(1)[1].cpu().numpy() )

    y_true = np.concatenate(y_true)
    y_pred = np.concatenate(y_pred)

    return classification_report(y_true, y_pred, target_names=loader.dataset.classes)
```

```
print(report(model=m, loader=test_loader, device="cuda"))
```

```
##               precision    recall  f1-score   support
##
##     airplane       0.82      0.88      0.85      1000
##   automobile       0.95      0.89      0.92      1000
##         bird       0.85      0.70      0.77      1000
##          cat       0.68      0.74      0.71      1000
##         deer       0.84      0.83      0.83      1000
##          dog       0.81      0.73      0.77      1000
##         frog       0.83      0.92      0.87      1000
##        horse       0.87      0.87      0.87      1000
##         ship       0.89      0.92      0.90      1000
##        truck       0.86      0.93      0.89      1000
##
##     accuracy                           0.84     10000
##    macro avg       0.84      0.84      0.84     10000
## weighted avg       0.84      0.84      0.84     10000
```