pytorch - GPU

Lecture 24

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:

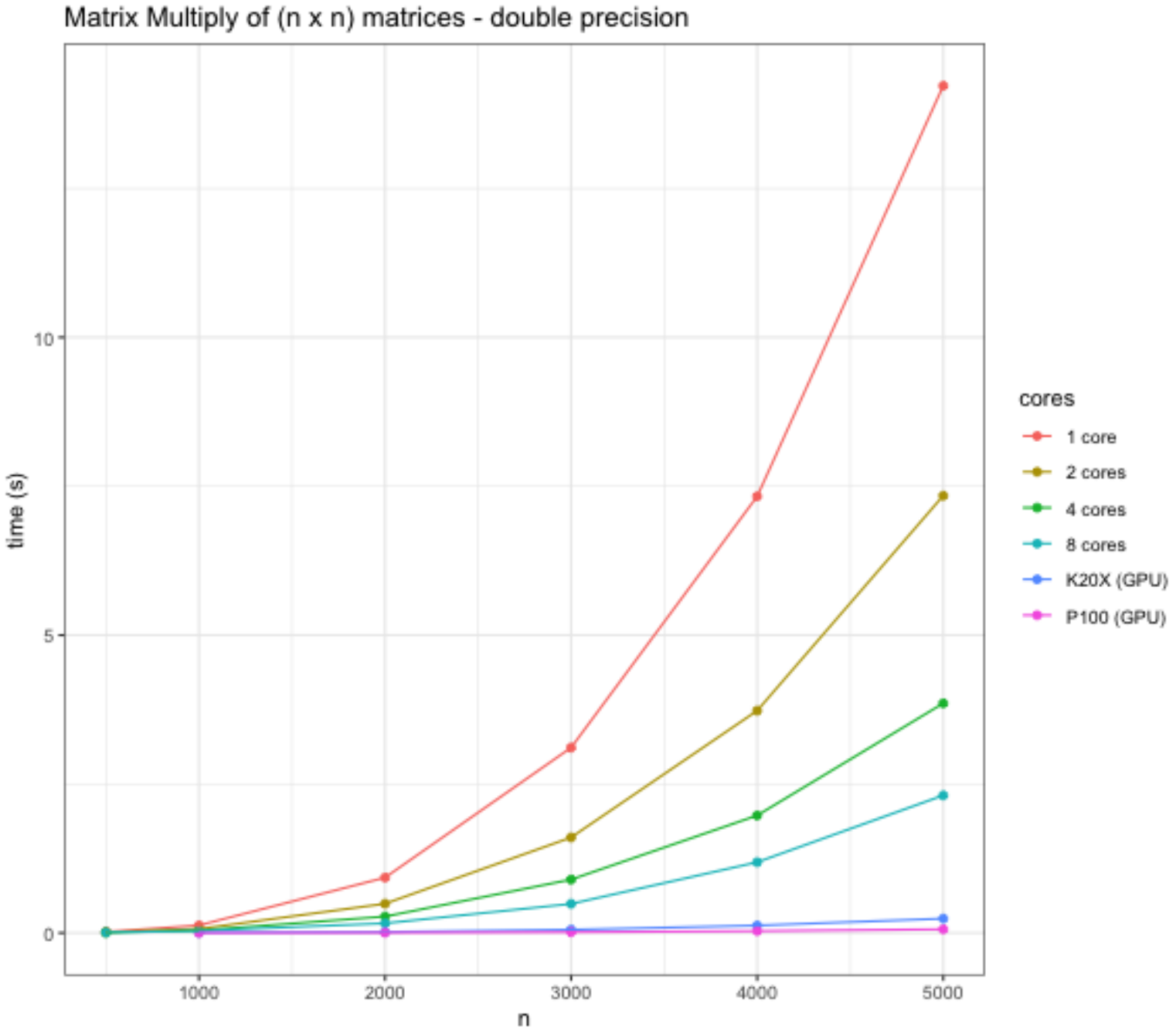
- cuBLAS
- cuSOLVER
- cuSPARSE
- cuFFT
- cuTENSOR
- cuRAND
- Thrust
- cuDNN

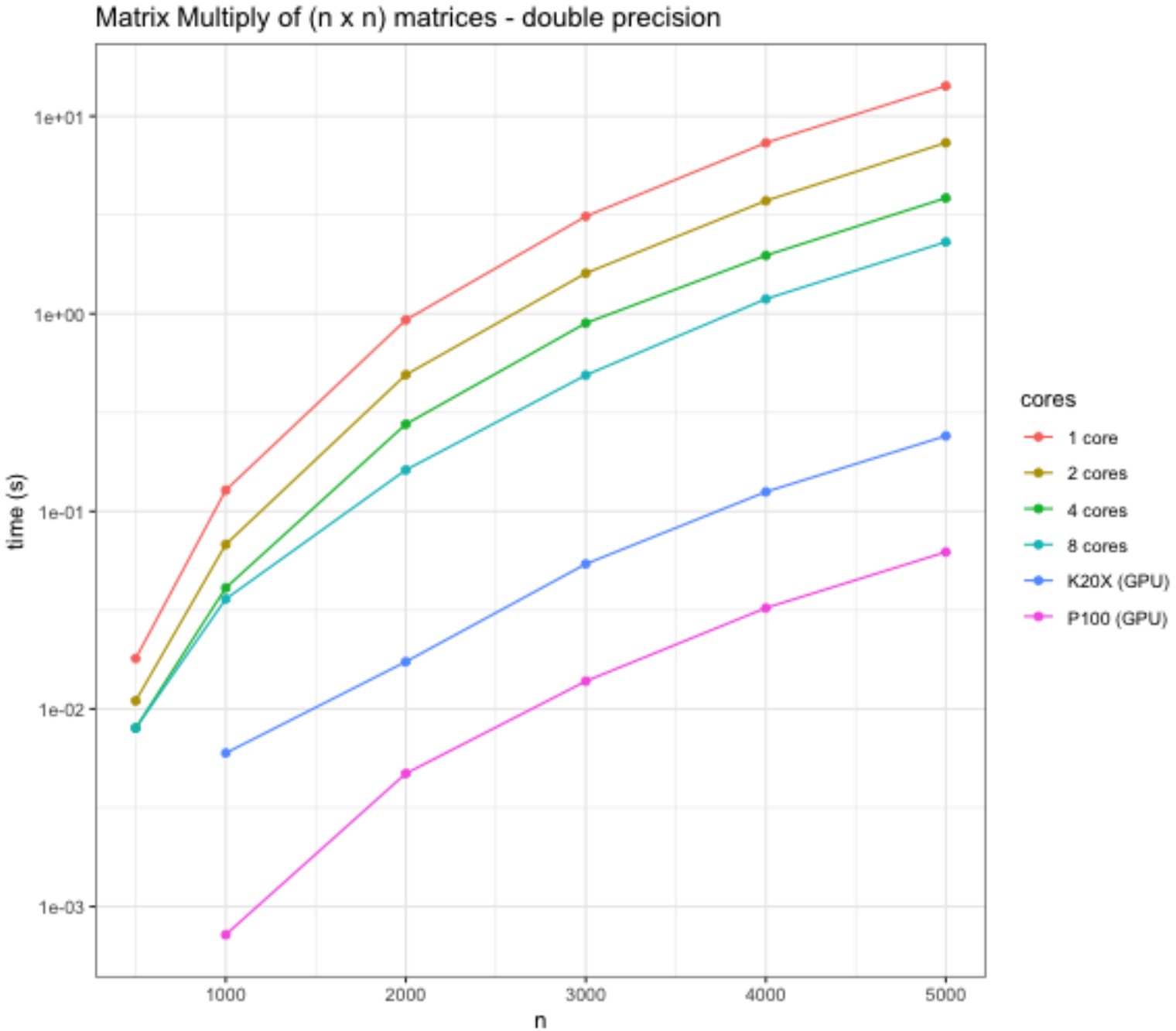
CUDA Kernels

```
// 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)
10
5
0
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

```
1 nvidia-smi
```

```
Tue Apr 11 22:04:42 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 530.30.02              Driver Version: 530.30.02    CUDA Version: 12.1     |
|-----------------------------------------+----------------------+----------------------+
| 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-16GB           Off| 00000000:02:00.0 Off |                    0 |
| N/A   38C    P0               25W / 250W|      0MiB / 16384MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   1  Tesla P100-PCIE-16GB           Off| 00000000:03:00.0 Off |                    0 |
| N/A   39C    P0               27W / 250W|      0MiB / 16384MiB |      2%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
```

Torch GPU Information

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

```
True
```

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

```
2
```

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

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

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

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

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

```
_CudaDeviceProperties(name='Tesla P100-PCIE-16GB', major=6, minor=0, total_memory=16276MB, multi_processor_c
```

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

```
_CudaDeviceProperties(name='Tesla P100-PCIE-16GB', major=6, minor=0, total_memory=16276MB, 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.

```
1 cpu = torch.device('cpu')
2 cuda0 = torch.device('cuda:0')
3 cuda1 = torch.device('cuda:1')
4
5 x = torch.linspace(0,1,5, device=cuda0); x
```

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

```
1 y = torch.randn(5,2, device=cuda0); y
```

```
tensor([[ 0.2451,  2.1824],
        [ 0.1801, -0.6749],
        [-1.3846,  1.8360],
        [ 0.5597, -1.0635],
        [ 0.1923,  0.4840]], device='cuda:0')
```

```
1 z = torch.rand(2,3, device=cpu); z
```

```
tensor([[0.1953, 0.2696, 0.8139],
        [0.7407, 0.7084, 0.5559]])
```

```
1 x @ y
```

```
tensor([-0.0352,  0.4357], device='cuda:0')
```

```
1 y @ z
```

```
Error: RuntimeError: Expected all tensors to be on th
```

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

```
tensor([[ 1.6644,  1.6121,  1.4127],
        [-0.4647, -0.4295, -0.2286],
        [ 1.0895,  0.9274, -0.1063],
        [-0.6784, -0.6025, -0.1356],
        [ 0.3961,  0.3947,  0.4256]], device='cuda:0'
```

NN Layers + GPU

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

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

```
1 nn(X)
```

```
Error: RuntimeError: Expected all tensors to be on the same device, but found at
```

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

```
tensor([[-0.0210, -0.8098,  0.4232,  0.
         [ 0.3679,  0.5455, -0.4389, -0.
         [ 0.2806, -0.1451,  0.1253, -1.
         [ 0.1966,  0.0387, -0.0724,  0.
         [-0.0910,  0.2663,  0.0824, -0.
         [-0.0415,  0.2805, -0.2223, -0.
         [-0.7177,  0.6184,  0.1609,  0.
         [ 0.0736,  0.5034, -0.1955,  0.
         [ 0.2303, -0.4893,  0.0740,  0.
         [ 0.5738, -0.8393,  0.0200,  0.
       grad_fn=<AddmmBackward0>)
```

```
1 nn.to(device="cuda")(X)
```

```
tensor([[-0.0210, -0.8098,  0.4232,  0.
         [ 0.3679,  0.5455, -0.4389, -0.
         [ 0.2806, -0.1451,  0.1253, -1.
         [ 0.1966,  0.0387, -0.0724,  0.
         [-0.0910,  0.2663,  0.0824, -0.
         [-0.0415,  0.2805, -0.2223, -0.
         [-0.7177,  0.6184,  0.1609,  0.
         [ 0.0736,  0.5034, -0.1955,  0.
         [ 0.2303, -0.4893,  0.0740,  0.
         [ 0.5738, -0.8393,  0.0200,  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.model = torch.nn.Sequential(
          torch.nn.Unflatten(1, (1,8,8)),
          torch.nn.Conv2d(
            in_channels=1, out_channels=8,
            kernel_size=3, stride=1, padding=1
          ),
          torch.nn.ReLU(),
          torch.nn.MaxPool2d(kernel_size=2),
          torch.nn.Flatten(),
          torch.nn.Linear(8 * 4 * 4, 10)
        ).to(device=self.device)

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

    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 = []
```

CPU vs Cuda

```
m = mnist_conv_model(device="cpu")
loss = m.fit(X_train, y_train, n=1000)
loss[-1]
```

```
0.03284111246466637
```

```
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=
loss[-1]
```

```
0.040549129247665405
```

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

```
tensor(0.9861, device='cuda:0')
```

Performance

CPU performance:

```
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) / 1000)
```

```
2.515770263671875
```

GPU performance:

```
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=
end.record()

torch.cuda.synchronize()
print(start.elapsed_time(end) / 1000)
```

```
2.3513505859375
```

Profiling CPU

```
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
---------------------------------  ------------  ------------  ------------  ------------  ------------  ---
         aten::mkldnn_convolution        50.03%       1.500ms        50.73%       1.521ms       1.521ms
    aten::max_pool2d_with_indices        28.89%     866.000us        28.89%     866.000us     866.000us
                 aten::clamp_min        12.27%     368.000us        12.27%     368.000us     368.000us
                     aten::addmm         2.84%      85.000us         3.64%     109.000us     109.000us
                 aten::unflatten         0.83%      25.000us         1.27%      38.000us      38.000us
---------------------------------  ------------  ------------  ------------  ------------  ------------  ---
Self CPU time total: 2.998ms
```

Profiling GPU

```
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  C
-------------------------------------------------  ------------  ------------  ------------  ------------  -
                         aten::cudnn_convolution        83.89%       1.797ms        85.15%       1.824ms
                                      aten::addmm         3.73%      80.000us         4.90%     105.000us
                                cudaLaunchKernel         3.08%      66.000us         3.08%      66.000us
                                     aten::_convolution         1.45%      31.000us        88.38%       1.893ms
                   aten::max_pool2d_with_indices         1.21%      26.000us         1.63%      35.000us
-------------------------------------------------  ------------  ------------  ------------  ------------  -
Self CPU time total: 2.142ms
```

CIFAR10

homepage

Loading the data

```
import torchvision

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

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

CIFAR10 data

```
1 training_data.classes
```

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

```
1 training_data.data.shape
```

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

```
1 test_data.data.shape
```

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

```
1 training_data[0]
```

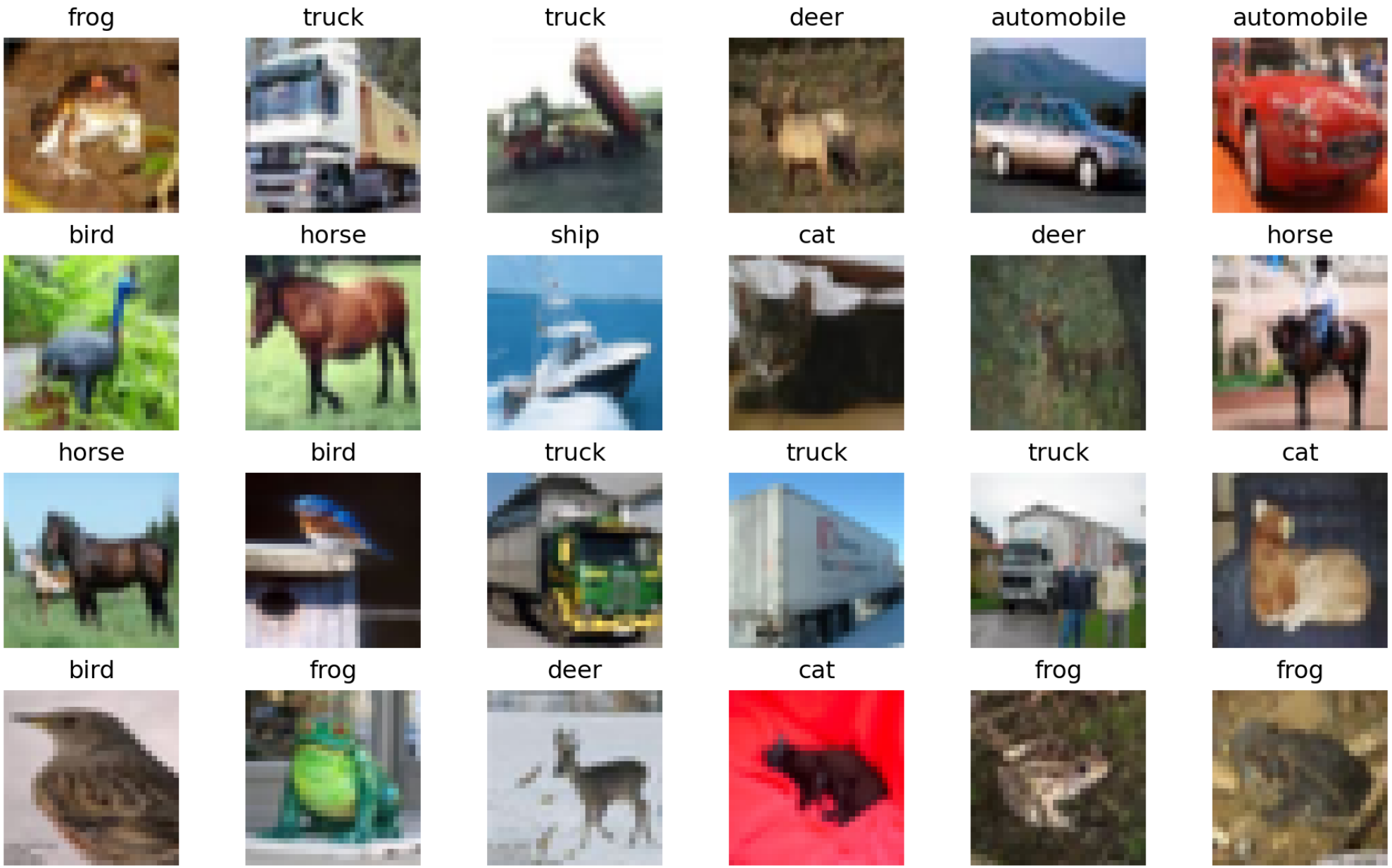
```
(tensor([[[0.2314, 0.1686, 0.1961,  ..., 0.6196, 0.5961, 0.5804],
         [0.0627, 0.0000, 0.0706,  ..., 0.4824, 0.4667, 0.4784],
         [0.0980, 0.0627, 0.1922,  ..., 0.4627, 0.4706, 0.4275],
         ...,
         [0.8157, 0.7882, 0.7765,  ..., 0.6275, 0.2196, 0.2078],
         [0.7059, 0.6784, 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],
         [0.0784, 0.0000, 0.0314,  ..., 0.3451, 0.3255, 0.3412],
         [0.0941, 0.0275, 0.1059,  ..., 0.3294, 0.3294, 0.2863],
         ...,
         [0.6667, 0.6000, 0.6314,  ..., 0.5216, 0.1216, 0.1333],
         [0.5451, 0.4824, 0.5647,  ..., 0.5804, 0.2431, 0.2078],
```

```
 [0.5647, 0.5059, 0.5569,  ..., 0.7216, 0.4627, 0.3608]],

[[0.2471, 0.1765, 0.1686,  ..., 0.4235, 0.4000, 0.4039],
 [0.0784, 0.0000, 0.0000,  ..., 0.2157, 0.1961, 0.2235],
 [0.0824, 0.0000, 0.0314,  ..., 0.1961, 0.1961, 0.1647],
 ...,
 [0.3765, 0.1333, 0.1020,  ..., 0.2745, 0.0275, 0.0784],
 [0.3765, 0.1647, 0.1176,  ..., 0.3686, 0.1333, 0.1333],
```

Example data



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
)
```

Loader generator

```
1 training_loader
```

```
<torch.utils.data.dataloader.DataLoader object at 0x7fb8542fce10>
```

```
1 X, y = next(iter(training_loader))
2 X.shape
```

```
torch.Size([100, 3, 32, 32])
```

```
1 y.shape
```

```
torch.Size([100])
```

CIFAR CNN

```
class cifar_conv_model(torch.nn.Module):
    def __init__(self, device):
        super().__init__()
        self.device = torch.device(device)
        self.epoch = 0
        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)

```

CNN Performance - CPU (1 step)

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

m_cpu = cifar_conv_model(device="cpu")
tmp = m_cpu(X)

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    #
---------------------------------  ------------  ------------  ------------  ------------  ------------  ---
         aten::mkldnn_convolution        60.41%       2.100ms        60.99%       2.120ms       1.060ms
    aten::max_pool2d_with_indices        26.73%     929.000us        26.73%     929.000us     464.500us
                     aten::addmm         4.78%     166.000us         5.58%     194.000us      64.667us
                  aten::clamp_min         3.60%     125.000us         3.60%     125.000us      31.250us
                       aten::relu         0.78%      27.000us         4.37%     152.000us      38.000us
---------------------------------  ------------  ------------  ------------  ------------  ------------  ---
Self CPU time total: 3.476ms
```

CNN Performance - GPU (1 step)

```
m_cuda = cifar_conv_model(device="cuda")
Xc, yc = X.to(device="cuda"), y.to(device="cuda")
tmp = m_cuda(Xc)

with torch.autograd.profiler.profile(with_stack=True) as prof_cuda:
    tmp = m_cuda(Xc)
```

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

```
-------------------------------------------------  ------------  ------------  ------------  ------------  -
                                             Name    Self CPU %      Self CPU   CPU total %     CPU total  C
-------------------------------------------------  ------------  ------------  ------------  ------------  -
                         aten::cudnn_convolution        68.77%       1.394ms        70.45%       1.428ms
                                       cudaMalloc         6.41%     130.000us         6.41%     130.000us
                                       aten::addmm         5.97%     121.000us         7.75%     157.000us
                                  cudaLaunchKernel         4.93%     100.000us         4.93%     100.000us
                                       aten::clamp_min         2.91%      59.000us        10.56%     214.000us
-------------------------------------------------  ------------  ------------  ------------  ------------  -
Self CPU time total: 2.027ms
```

CNN Performance - CPU (1 epoch)

```
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 to
-------------------------------------------------------  ------------  ------------  ------------  ---------
                          aten::convolution_backward        29.40%     987.971ms        29.75%     999.90
                            aten::mkldnn_convolution        17.97%     604.031ms        18.45%     619.98
                       aten::max_pool2d_with_indices        10.16%     341.493ms        10.18%     342.25
                             Optimizer.step#SGD.step         6.21%     208.598ms         9.66%     324.58
enumerate(DataLoader)#_MultiProcessingDataLoaderIter...         5.47%     183.792ms         5.50%     184.85
-------------------------------------------------------  ------------  ------------  ------------  ---------
Self CPU time total: 3.361s
```

CNN Performance - GPU (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 to
-------------------------------------------------------  ------------  ------------  ------------  ---------
enumerate(DataLoader)#_MultiProcessingDataLoaderIter...        12.77%     343.852ms        12.82%     345.32
                                         cudaLaunchKernel       11.54%     310.888ms        11.55%     311.13
                                Optimizer.step#SGD.step         9.49%     255.614ms        12.55%     338.03
                                             aten::addmm         5.55%     149.386ms         7.58%     204.09
                           aten::convolution_backward        4.76%     128.087ms         8.27%     222.63
-------------------------------------------------------  ------------  ------------  ------------  ---------
Self CPU time total: 2.693s
```

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 predicti
            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 continue fitting with the existing model,

```
m.fit(training_loader, epochs=10, n_report=500)
## [Epoch 11, Minibatch  500] loss: 0.885
## [Epoch 12, Minibatch  500] loss: 0.853
## [Epoch 13, Minibatch  500] loss: 0.839
## [Epoch 14, Minibatch  500] loss: 0.828
## [Epoch 15, Minibatch  500] loss: 0.817
## [Epoch 16, Minibatch  500] loss: 0.806
## [Epoch 17, Minibatch  500] loss: 0.798
## [Epoch 18, Minibatch  500] loss: 0.787
## [Epoch 19, Minibatch  500] loss: 0.780
## [Epoch 20, 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 21, Minibatch  500] loss: 0.764
## [Epoch 22, Minibatch  500] loss: 0.756
## [Epoch 23, Minibatch  500] loss: 0.748
## [Epoch 24, Minibatch  500] loss: 0.739
## [Epoch 25, Minibatch  500] loss: 0.733
## [Epoch 26, Minibatch  500] loss: 0.726
## [Epoch 27, Minibatch  500] loss: 0.718
## [Epoch 28, Minibatch  500] loss: 0.710
## [Epoch 29, Minibatch  500] loss: 0.702
## [Epoch 30, 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 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(),
            torch.nn.Linear(512,10)
        ]

        return torch.nn.Sequential(*layers).to(self.device)

    def __init__(self, device):
        super().__init__()
```


Based on code from pytorch-cifar, original paper

Model

```
1 VGG16("cpu").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)
```

VGG16 performance - CPU

```
X, y = next(iter(training_loader))
m_cpu = VGG16(device="cpu")
tmp = m_cpu(X)

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    #
-------------------------------  ------------  ------------  ------------  ------------  ------------  ---
        aten::mkldnn_convolution        84.43%     172.734ms        84.68%     173.247ms      13.327ms
         aten::native_batch_norm         8.11%      16.593ms         8.23%      16.830ms       1.295ms
   aten::max_pool2d_with_indices         5.24%      10.725ms         5.24%      10.725ms       2.145ms
               aten::clamp_min_         1.21%       2.476ms         1.21%       2.476ms     190.462us
                    aten::empty         0.31%     628.000us         0.31%     628.000us       4.831us
-------------------------------  ------------  ------------  ------------  ------------  ------------  ---
Self CPU time total: 204.599ms
```

VGG16 performance - GPU

```
m_cuda = VGG16(device="cuda")
Xc, yc = X.to(device="cuda"), y.to(device="cuda")
tmp = m_cuda(Xc)

with torch.autograd.profiler.profile(with_stack=True) as prof_cuda:
    tmp = m_cuda(Xc)
```

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

```
-------------------------------------------------  ------------  ------------  ------------  ------------  -
                                             Name    Self CPU %      Self CPU   CPU total %     CPU total  C
-------------------------------------------------  ------------  ------------  ------------  ------------  -
                         aten::cudnn_convolution        36.58%       2.866ms        51.90%       4.066ms
                          aten::cudnn_batch_norm        20.74%       1.625ms        25.35%       1.986ms
                                       cudaMalloc        13.85%       1.085ms        13.85%       1.085ms
                                  cudaLaunchKernel         7.14%     559.000us         7.14%     559.000us
                                       aten::add_         3.83%     300.000us         5.58%     437.000us
-------------------------------------------------  ------------  ------------  ------------  ------------  -
Self CPU time total: 7.834ms
```

VGG16 performance - Apple M1 GPU (mps)

```
m_mps = VGG16(device="mps")
Xm, ym = X.to(device="mps"), y.to(device="mps")

with torch.autograd.profiler.profile(with_stack=True) as prof_mps:
    tmp = m_mps(Xm)
```

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

```
-------------------------------  ------------  ------------  ------------  ------------  ------------  ----
                          Name    Self CPU %      Self CPU   CPU total %     CPU total  CPU time avg    #
-------------------------------  ------------  ------------  ------------  ------------  ------------  ----
         aten::native_batch_norm        35.71%       3.045ms        35.71%       3.045ms     234.231us
          aten::_mps_convolution        19.67%       1.677ms        19.88%       1.695ms     130.385us
    aten::_batch_norm_impl_index        11.92%       1.016ms        36.02%       3.071ms     236.231us
                     aten::relu_        11.29%     963.000us        11.29%     963.000us      74.077us
                      aten::add_        10.40%     887.000us        10.44%     890.000us      68.462us
-------------------------------  ------------  ------------  ------------  ------------  ------------  ----
Self CPU time total: 8.526ms
```

Fitting w/ `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
```

Fitting w/ `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
```

Some state-of-the-art examples

Hugging Face

This is an online community and platform for sharing machine learning models (architectures and weights), data, and related artifacts. They also maintain a number of packages and related training materials that help with building, training, and deploying ML models.

Some notable resources,

- `transformers` - APIs and tools to easily download and train state-of-the-art (pretrained) transformer based models
- `diffusers` - provides pretrained vision and audio diffusion models, and serves as a modular toolbox for inference and training
- `timm` - a library containing SOTA computer vision models, layers, utilities, optimizers, schedulers, data-loaders, augmentations, and training/evaluation scripts

Stable Diffusion

```
from diffusers import StableDiffusionPipeline

pipe = StableDiffusionPipeline.from_pretrained(
  "stabilityai/stable-diffusion-2-1-base", torch_dtype=torch.float16
).to("cuda")
```

...

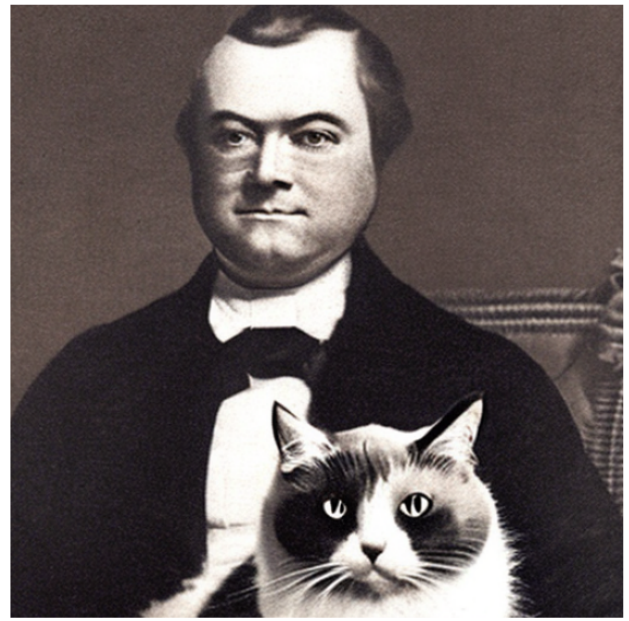
```
prompt = "a picture of thomas bayes with a cat on his lap"
generator = [torch.Generator(device="cuda").manual_seed(i) for i in range(6)]
fit = pipe(prompt, generator=generator, num_inference_steps=20, num_images_per_prompt=6)
```

...

```
fit.images
```

```
[<PIL.Image.Image image mode=RGB size=512x512 at 0x7FB991D3B910>, <PIL.Image.Image image mode=RGB size=512x5
   0x7FB993365850>, <PIL.Image.Image image mode=RGB size=512x512 at 0x7FB9933D0D90>, <PIL.Image.Image image
   size=512x512 at 0x7FB993094110>, <PIL.Image.Image image mode=RGB size=512x512 at 0x7FB991D82410>, <PIL.Im
   image mode=RGB size=512x512 at 0x7FBB53A9C6D0>]
```

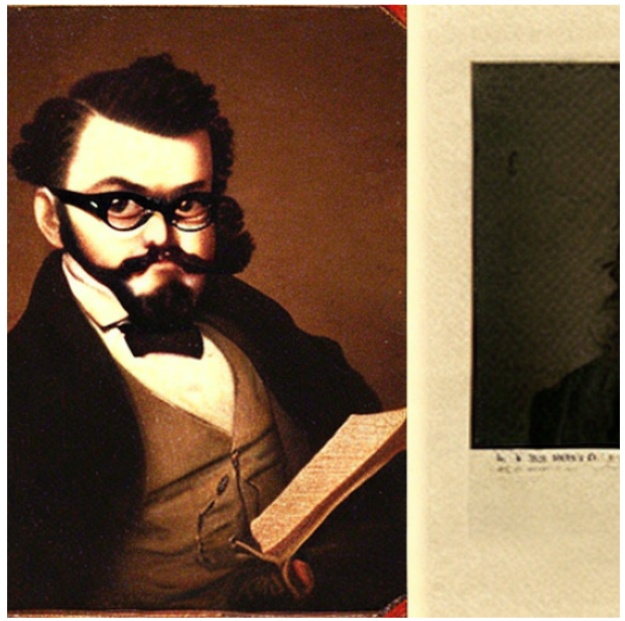





Customizing prompts

```
prompt = "a picture of thomas bayes with a cat on his lap"
prompts = [
  prompt + t for t in
  ["in the style of a japanese wood block print",
   "as a hipster with facial hair and glasses",
   "as a simpsons character, cartoon, yellow",
   "in the style of a vincent van gogh painting",
   "in the style of a picasso painting",
   "with flowery wall paper"
  ]
]

generator = [torch.Generator(device="cuda").manual_seed(i) for i in range(6)]
fit = pipe(prompts, generator=generator, num_inference_steps=20, num_images_per_prompt=1)
```



Increasing inference steps

```
generator = [torch.Generator(device="cuda").manual_seed(i) for i in range(6)]
fit = pipe(prompts, generator=generator, num_inference_steps=50, num_images_per_prompt=1)
```

Alpaca LoRA

```python
from transformers import GenerationConfig, LlamaTokenizer, LlamaForCausalLM

tokenizer = LlamaTokenizer.from_pretrained("chainyo/alpaca-lora-7b")

model = LlamaForCausalLM.from_pretrained(
    "chainyo/alpaca-lora-7b",
    load_in_8bit=True,
    torch_dtype=torch.float16,
    device_map="auto",
)

generation_config = GenerationConfig(
    temperature=0.2,
    top_p=0.75,
    top_k=40,
    num_beams=4,
    max_new_tokens=128,
)
```

Based on chainyo/alpaca-lora-7b, see also alpaca lora, stanform alpaca & llama

Generate a prompt

```
instruction = "Write a short childrens story about Thomas Bayes and his pet cat"
input_ctxt = None
prompt = generate_prompt(instruction, input_ctxt)
print(prompt)
```

```
Below is an instruction that describes a task. Write a response that appropriately completes the request.

### Instruction:
Write a short childrens story about Thomas Bayes and his pet cat

### Response:
```

Running the model

```python
input_ids = tokenizer(prompt, return_tensors="pt").input_ids.to(model.device)

with torch.no_grad():
    outputs = model.generate(
        input_ids=input_ids,
        generation_config=generation_config,
        return_dict_in_generate=True,
        output_scores=True,
    )

response = tokenizer.decode(outputs.sequences[0], skip_special_tokens=True)
print(response)
```

```
Below is an instruction that describes a task. Write a response that appropriately completes the request.

### Instruction:
Write a short childrens story about Thomas Bayes and his pet cat

### Response:
Once upon a time, there was a little boy named Thomas Bayes. He had a pet cat named Fluffy, and
they were the best of friends. One day, Thomas and Fluffy decided to go on an adventure. They
traveled far and wide, exploring new places and meeting new people. Along the way, Thomas and
Fluffy learned many valuable lessons, such as the importance of friendship and the joy of discovery.
Eventually, Thomas and Fluffy made their way back home, where they were welcomed with open arms.
Thomas and Fluffy had a wonderful time.
```