Visualizing Deep Learning using PyTorch and PowerAI

Dustin VanStee
Data Scientists
IBM Worldwide Client Experience Centers

2019 Yale University
Dec 3 | New Haven, CT

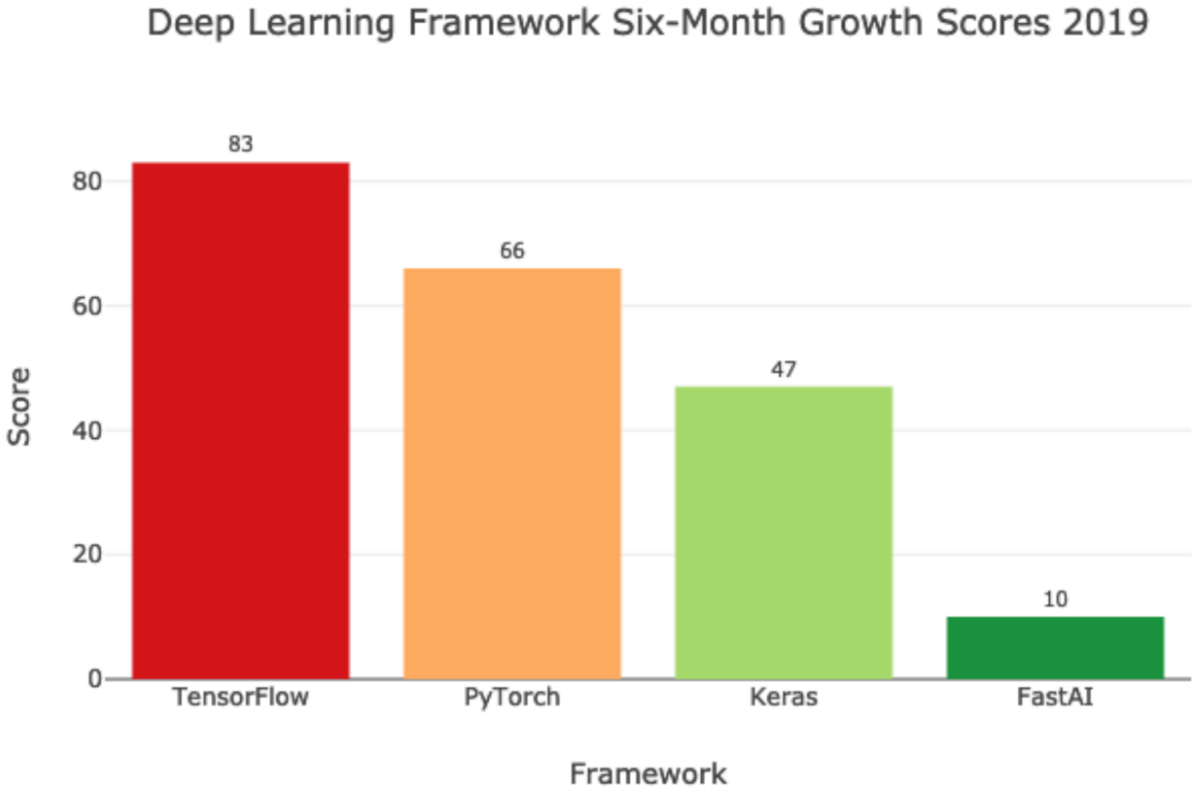
Agenda

- **Pytorch on PowerAI**
  - **What is Pytorch?**
  - **Where does it fit in deep learning frameworks**
  - **Where can I get it?**

  - **Some examples**
  - **GPU access**
  - **Demo**
  - **Lab !**

Deep Learning Frameworks

Deep Learning Framework Popularity

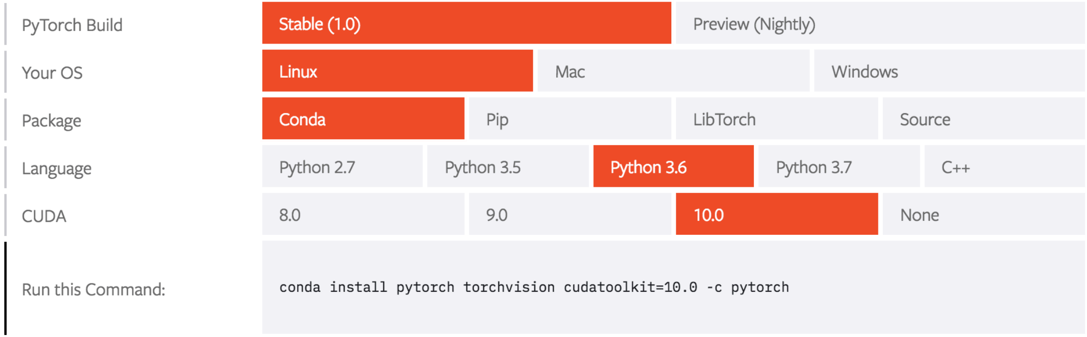


- https://towardsdatascience.com/which-deep-learning-framework-is-growing-fastest-3f77f14aa318

PYTORCH – a quick review

Facebook’s framework for research

- ✓ Cousin of LUA based Torch framework, but was rewritten to be tailored to Python frontend
- ✓ Gaining popularity quickly for its ease of use in R&D
- ✓ Supports dynamic computation graphs !
- ✓ Based on Python with Numpy compatibility
- ✓ Multi-GPU
- ✓ Easy to use, and supports standard debug tools

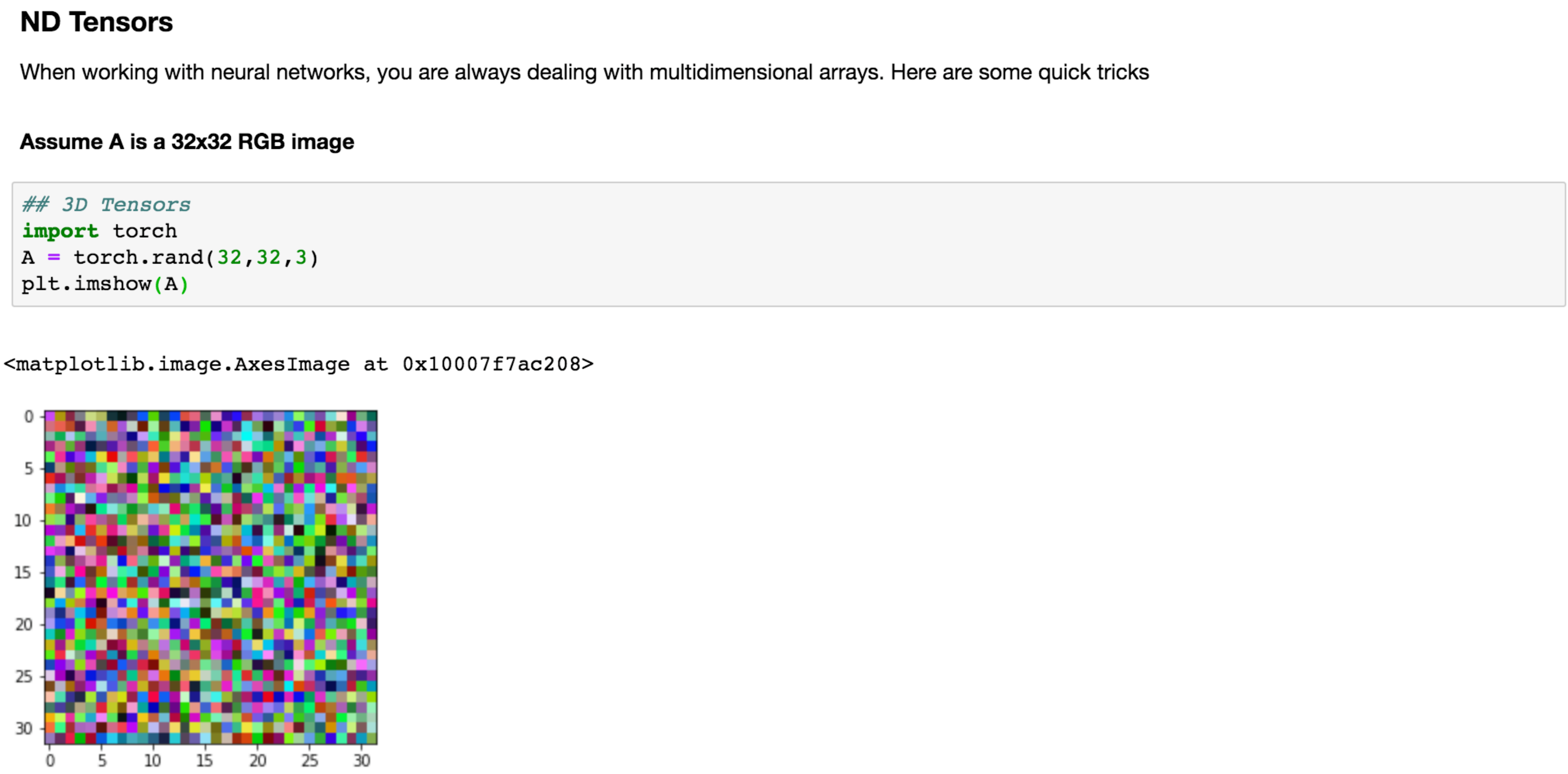


PyTorch Build	[✓ Stable (1.0)]	Preview (Nightly)			
Your OS	[✓ Linux]	Mac	Windows		
Package	[✓ Conda]	Pip	LibTorch	Source	
Language	Python 2.7	Python 3.5	[✓ Python 3.6]	Python 3.7	C++
CUDA	8.0	9.0	[✓ 10.0]	None	

Run this Command:

```
conda install pytorch torchvision cudatoolkit=10.0 -c pytorch
```

PyTorch Basics – ND arrays



ND Tensors

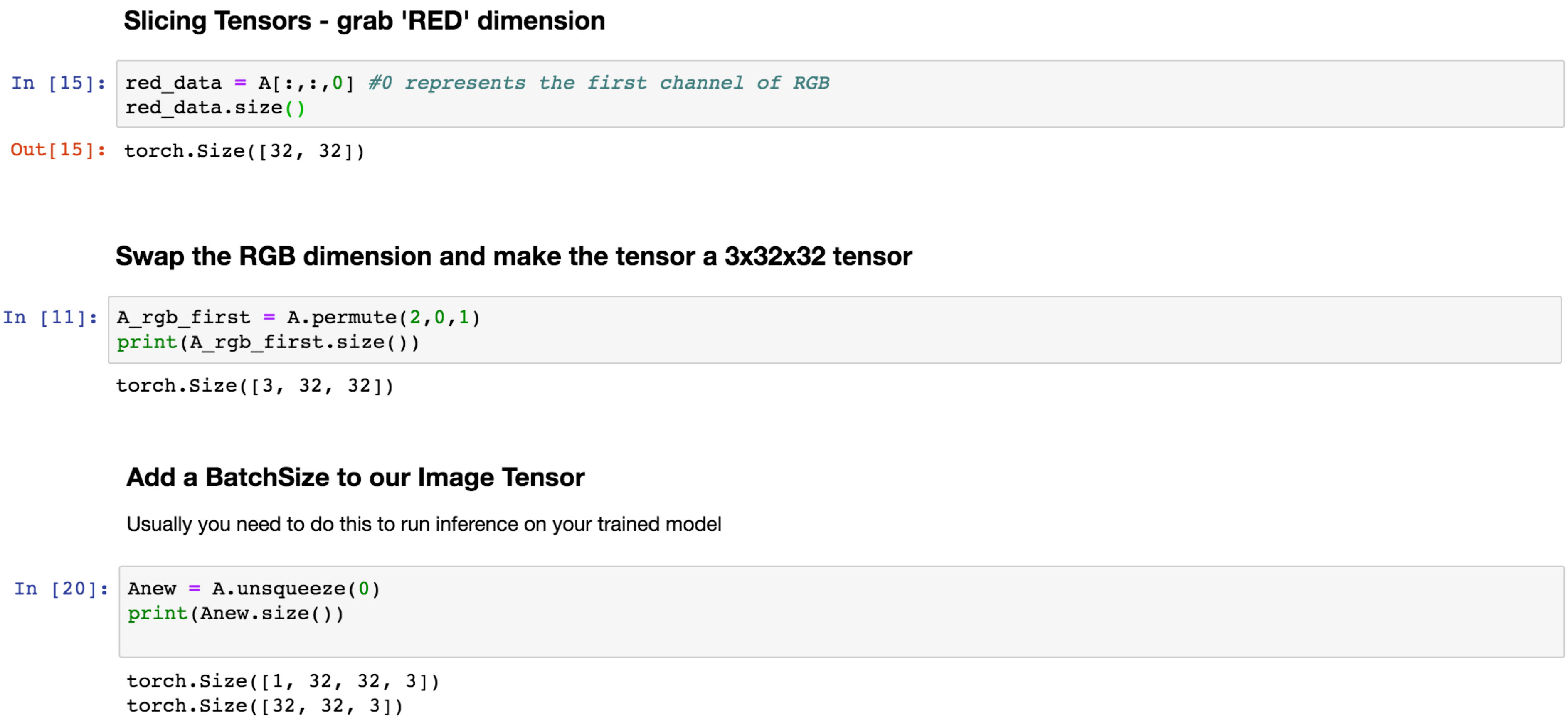
When working with neural networks, you are always dealing with multidimensional arrays. Here are some quick tricks

Assume A is a 32x32 RGB image

```
## 3D Tensors
import torch
A = torch.rand(32,32,3)
plt.imshow(A)
```

```
<matplotlib.image.AxesImage at 0x10007f7ac208>
```

PyTorch Basics – Basic Slicing /Dimension Operations



Slicing Tensors - grab 'RED' dimension

```
In [15]: red_data = A[:,:,0] #0 represents the first channel of RGB
         red_data.size()

Out[15]: torch.Size([32, 32])
```

Swap the RGB dimension and make the tensor a 3x32x32 tensor

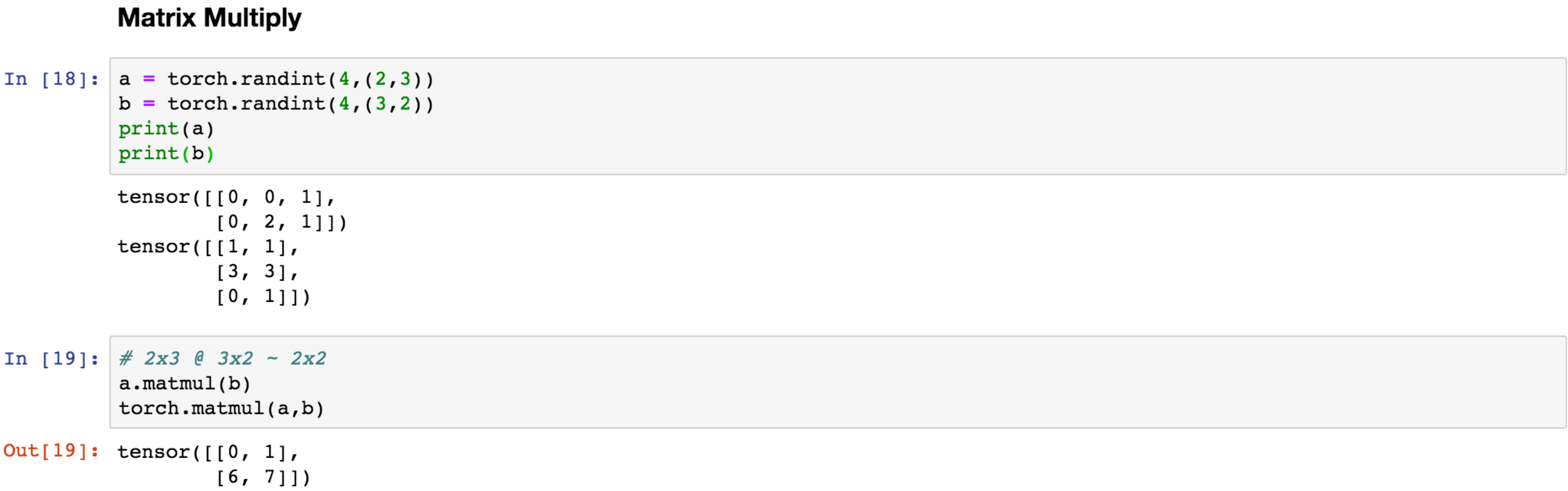
```
In [11]: A_rgb_first = A.permute(2,0,1)
         print(A_rgb_first.size())

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

Add a BatchSize to our Image Tensor

Usually you need to do this to run inference on your trained model

```
In [20]: Anew = A.unsqueeze(0)
         print(Anew.size())

         torch.Size([1, 32, 32, 3])
         torch.Size([32, 32, 3])
```

PyTorch Basics – Basic Matrix Operations

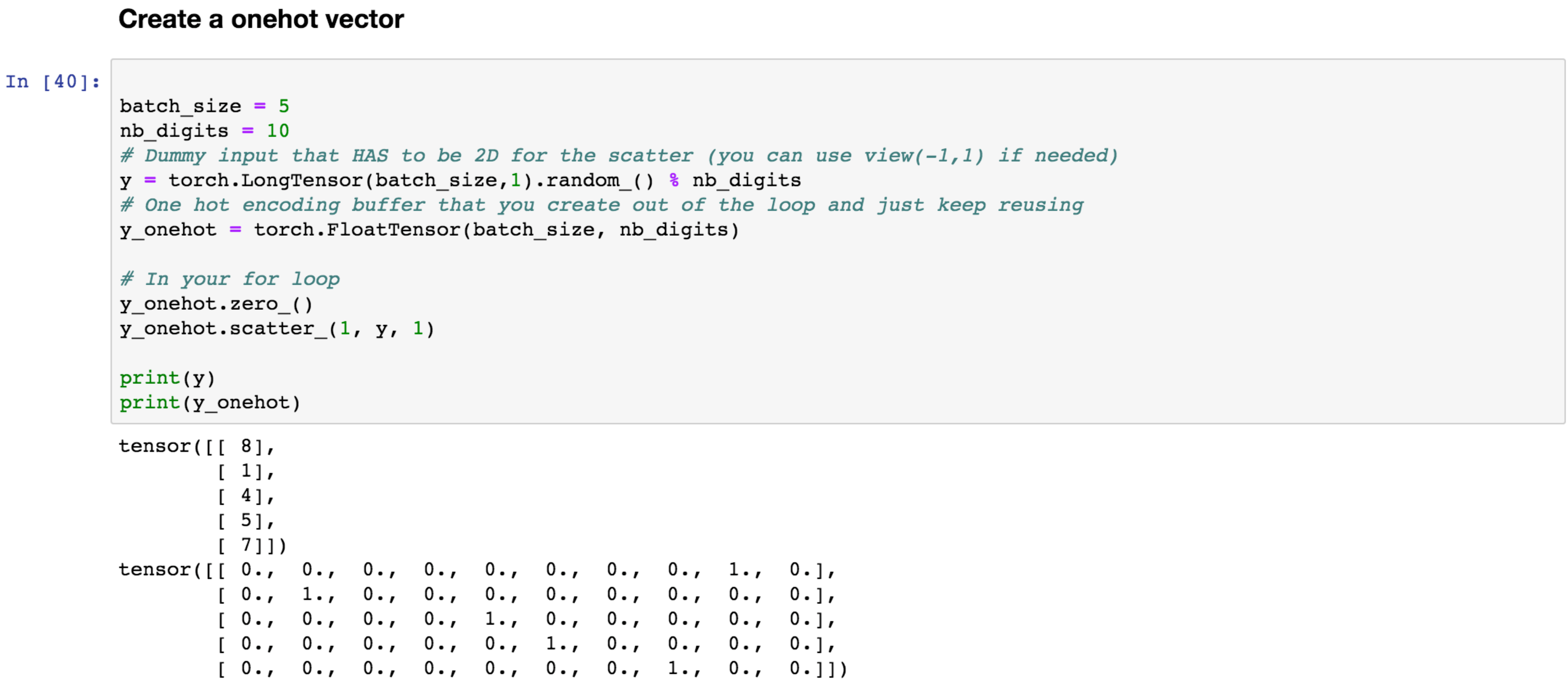


Matrix Multiply

```
In [18]: a = torch.randint(4,(2,3))
         b = torch.randint(4,(3,2))
         print(a)
         print(b)

         tensor([[0, 0, 1],
                 [0, 2, 1]])
         tensor([[1, 1],
                 [3, 3],
                 [0, 1]])
```

```
In [19]: # 2x3 @ 3x2 ~ 2x2
         a.matmul(b)
         torch.matmul(a,b)

Out[19]: tensor([[0, 1],
                 [6, 7]])
```

PyTorch Basics – Basic Index Operations



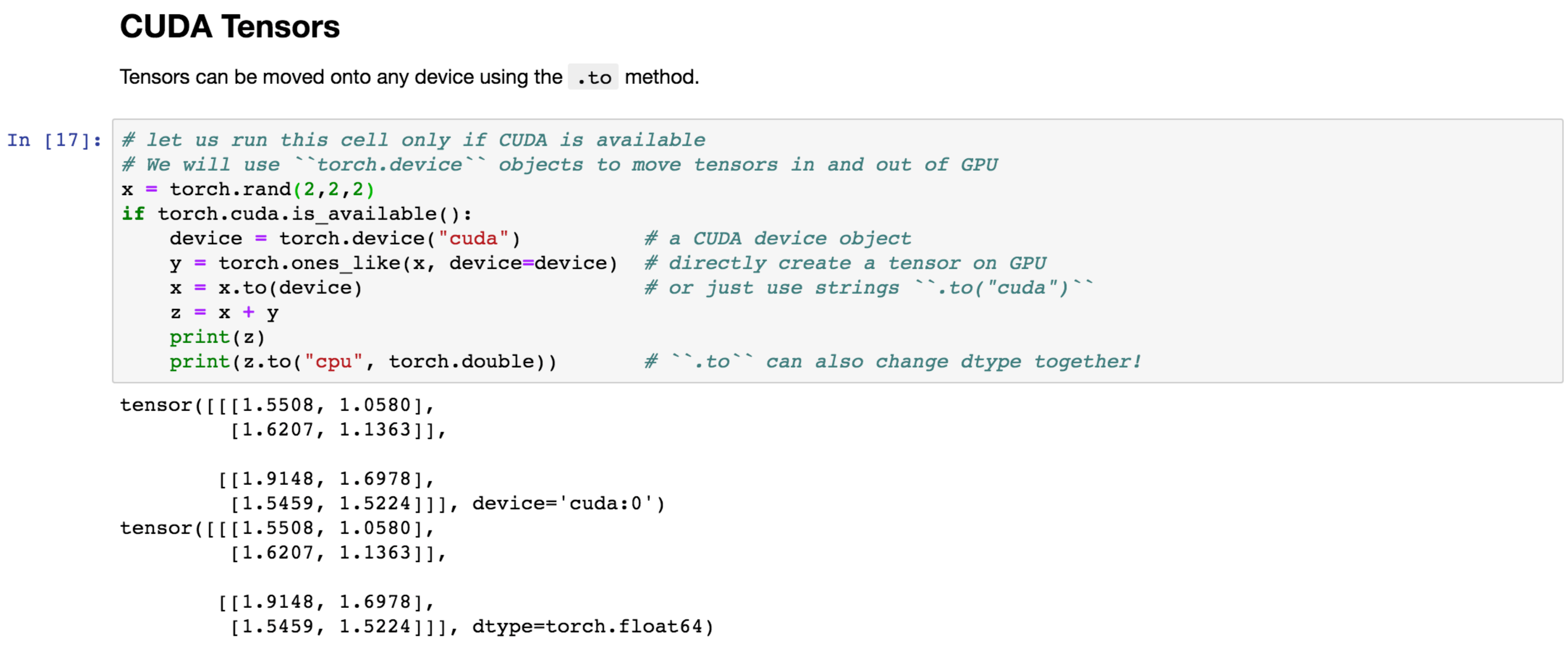
Create a onehot vector

In [40]:

```
batch_size = 5
nb_digits = 10
# Dummy input that HAS to be 2D for the scatter (you can use view(-1,1) if needed)
y = torch.LongTensor(batch_size,1).random_() % nb_digits
# One hot encoding buffer that you create out of the loop and just keep reusing
y_onehot = torch.FloatTensor(batch_size, nb_digits)

# In your for loop
y_onehot.zero_()
y_onehot.scatter_(1, y, 1)

print(y)
print(y_onehot)
```

```
tensor([[ 8],
        [ 1],
        [ 4],
        [ 5],
        [ 7]])
tensor([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.],
        [ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
        [ 0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.],
        [ 0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.],
        [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.]])
```

PyTorch: Using GPU’s



CUDA Tensors

Tensors can be moved onto any device using the `.to` method.

In [17]:

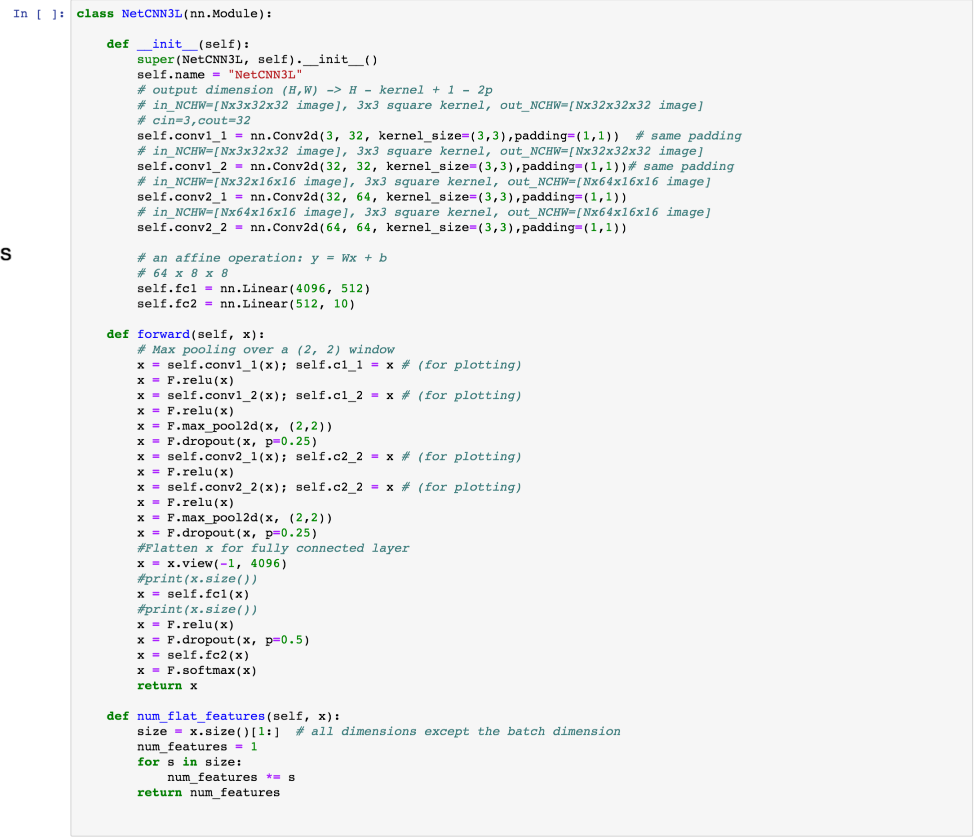
```
# let us run this cell only if CUDA is available
# We will use ``torch.device`` objects to move tensors in and out of GPU
x = torch.rand(2,2,2)
if torch.cuda.is_available():
    device = torch.device("cuda")          # a CUDA device object
    y = torch.ones_like(x, device=device)  # directly create a tensor on GPU
    x = x.to(device)                       # or just use strings ``.to("cuda")``
    z = x + y
    print(z)
    print(z.to("cpu", torch.double))       # ``.to`` can also change dtype together!
```

```
tensor([[[1.5508, 1.0580],
         [1.6207, 1.1363]],

        [[1.9148, 1.6978],
         [1.5459, 1.5224]]], device='cuda:0')
tensor([[[1.5508, 1.0580],
         [1.6207, 1.1363]],

        [[1.9148, 1.6978],
         [1.5459, 1.5224]]], dtype=torch.float64)
```

PyTorch: Build a model

To build a neural network using pytorch, you need to perform three main steps

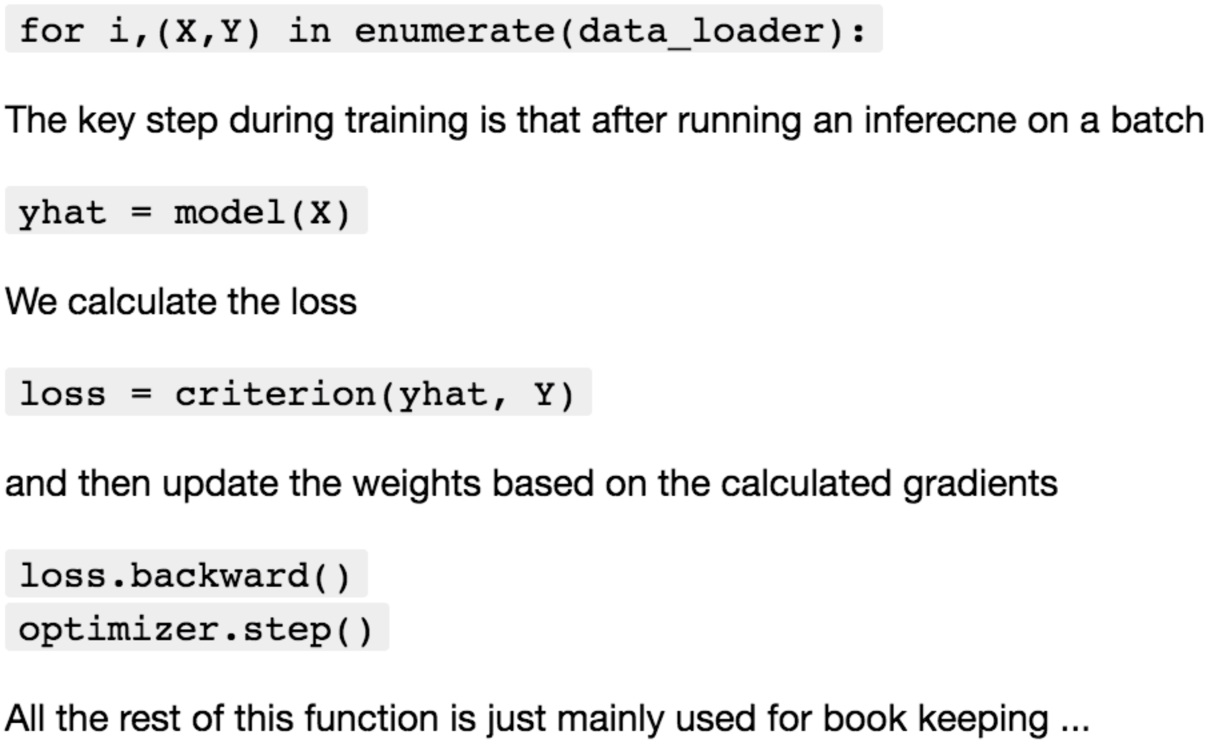
- extend nn.Module
- define **init** function
- define forward function



In [ ]:

```
class NetCNN3L(nn.Module):

    def __init__(self):
        super(NetCNN3L, self).__init__()
        self.name = "NetCNN3L"
        # output dimension (H,W) -> H - kernel + 1 - 2p
        # in_NCHW=[Nx3x32x32 image], 3x3 square kernel, out_NCHW=[Nx32x32x32 image]
        # cin=3,cout=32
        self.conv1_1 = nn.Conv2d(3, 32, kernel_size=(3,3),padding=(1,1))  # same padding
        # in_NCHW=[Nx3x32x32 image], 3x3 square kernel, out_NCHW=[Nx32x32x32 image]
        self.conv1_2 = nn.Conv2d(32, 32, kernel_size=(3,3),padding=(1,1))# same padding
        # in_NCHW=[Nx32x16x16 image], 3x3 square kernel, out_NCHW=[Nx64x16x16 image]
        self.conv2_1 = nn.Conv2d(32, 64, kernel_size=(3,3),padding=(1,1))
        # in_NCHW=[Nx64x16x16 image], 3x3 square kernel, out_NCHW=[Nx64x16x16 image]
        self.conv2_2 = nn.Conv2d(64, 64, kernel_size=(3,3),padding=(1,1))

        # an affine operation: y = Wx + b
        # 64 x 8 x 8
        self.fc1 = nn.Linear(4096, 512)
        self.fc2 = nn.Linear(512, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = self.conv1_1(x); self.c1_1 = x # (for plotting)
        x = F.relu(x)
        x = self.conv1_2(x); self.c1_2 = x # (for plotting)
        x = F.relu(x)
        x = F.max_pool2d(x, (2,2))
        x = F.dropout(x, p=0.25)
        x = self.conv2_1(x); self.c2_2 = x # (for plotting)
        x = F.relu(x)
        x = self.conv2_2(x); self.c2_2 = x # (for plotting)
        x = F.relu(x)
        x = F.max_pool2d(x, (2,2))
        x = F.dropout(x, p=0.25)
        #Flatten x for fully connected layer
        x = x.view(-1, 4096)
        #print(x.size())
        x = self.fc1(x)
        #print(x.size())
        x = F.relu(x)
        x = F.dropout(x, p=0.5)
        x = self.fc2(x)
        x = F.softmax(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features
```

PyTorch: Train a model



```
for i,(X,Y) in enumerate(data_loader):
```

The key step during training is that after running an inferecne on a batch

```
yhat = model(X)
```

We calculate the loss

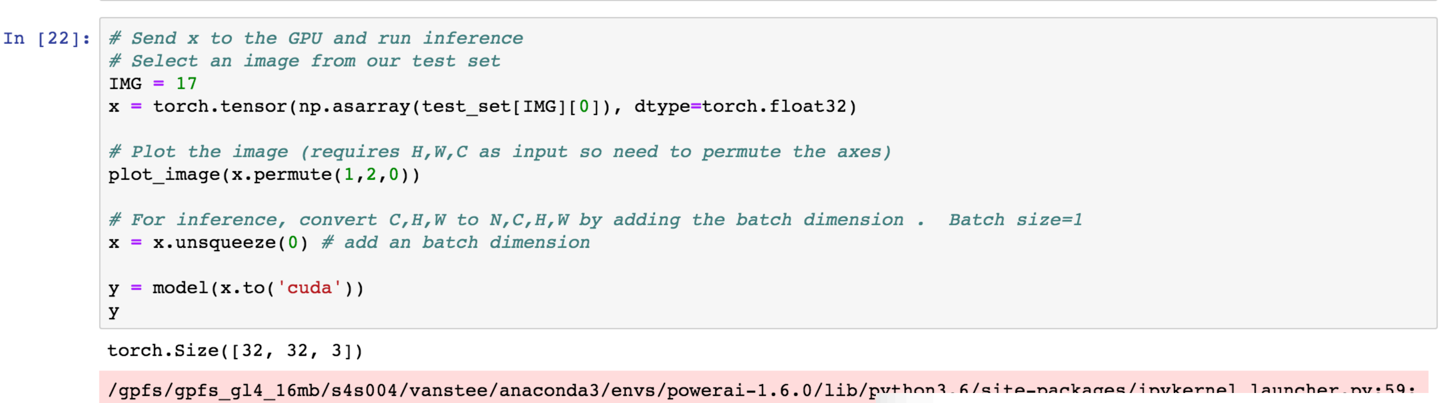
```
loss = criterion(yhat, Y)
```

and then update the weights based on the calculated gradients

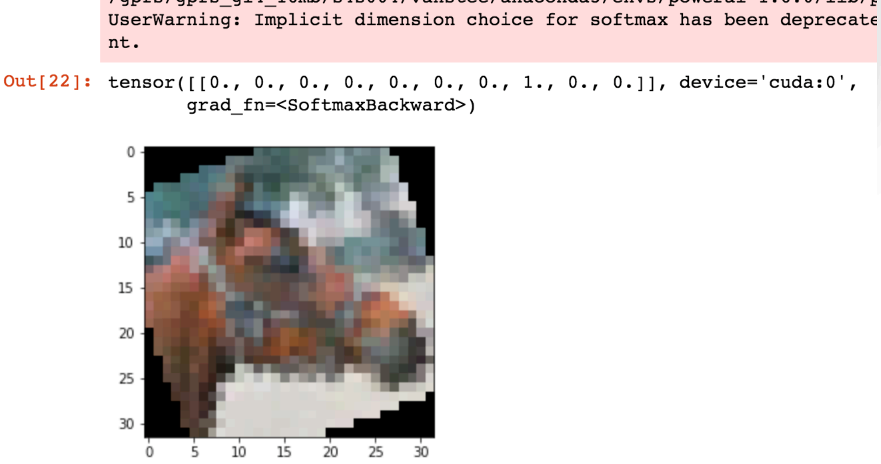
```
loss.backward()
optimizer.step()
```

All the rest of this function is just mainly used for book keeping ...

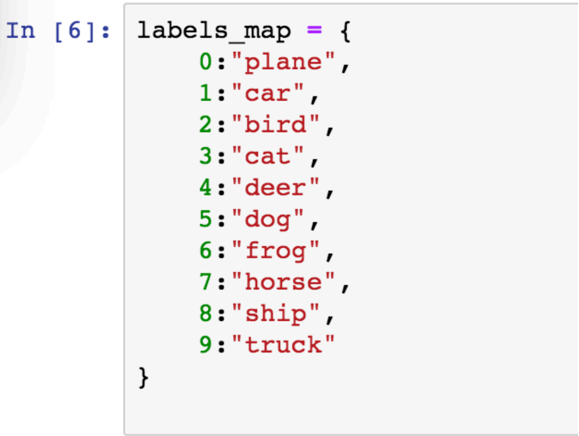
PyTorch: Inference on a model



```
In [22]: # Send x to the GPU and run inference
         # Select an image from our test set
         IMG = 17
         x = torch.tensor(np.asarray(test_set[IMG][0]), dtype=torch.float32)

         # Plot the image (requires H,W,C as input so need to permute the axes)
         plot_image(x.permute(1,2,0))

         # For inference, convert C,H,W to N,C,H,W by adding the batch dimension .  Batch size=1
         x = x.unsqueeze(0) # add an batch dimension

         y = model(x.to('cuda'))
         y

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

         /gpfs/gpfs_gl4_16mb/s4s004/vanstee/anaconda3/envs/powerai-1.6.0/lib/python3.6/site-packages/ipykernel_launcher.py:59:
         UserWarning: Implicit dimension choice for softmax has been deprecate
         nt.
```



```
Out[22]: tensor([[0., 0., 0., 0., 0., 0., 0., 1., 0., 0.]], device='cuda:0',
                grad_fn=<SoftmaxBackward>)
```



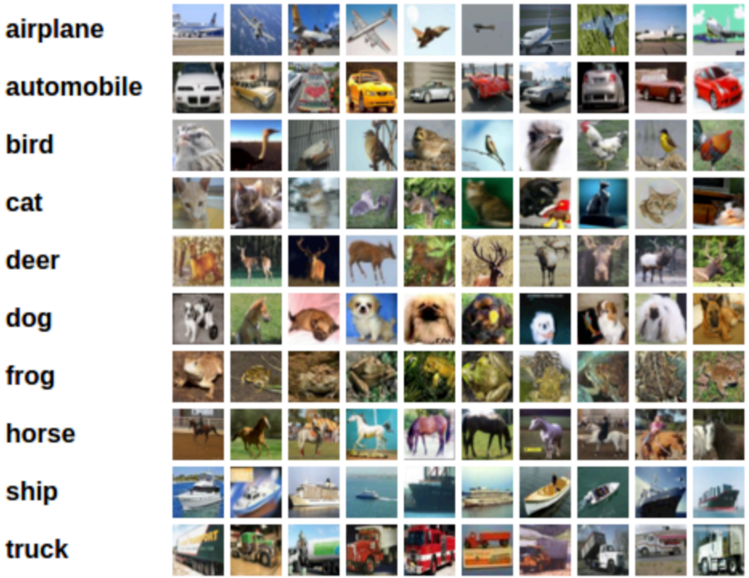
```
In [6]: labels_map = {
            0:"plane",
            1:"car",
            2:"bird",
            3:"cat",
            4:"deer",
            5:"dog",
            6:"frog",
            7:"horse",
            8:"ship",
            9:"truck"
        }
```

PyTorch : TorchVision Library

Torchvision : An image library that contains

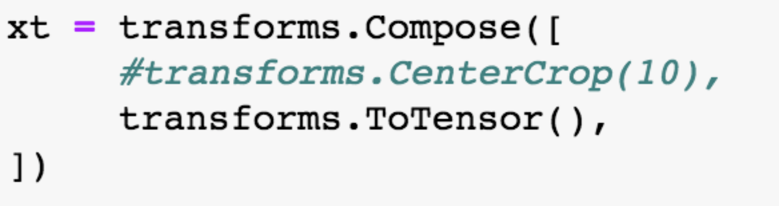
- Popular Image Datasets
- Popular NN Models (pretrained)
- Transformers

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3



CIFAR 10 Dataset

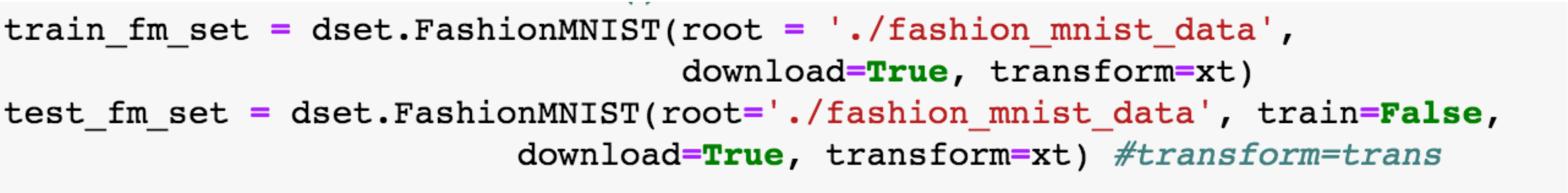
PyTorch : Vision Dataset API Example



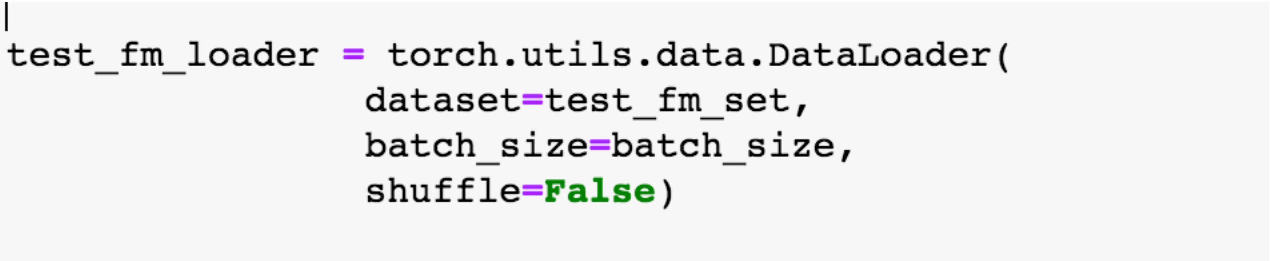
```
xt = transforms.Compose([
     #transforms.CenterCrop(10),
     transforms.ToTensor(),
])
```

Torchvision : An image library that contains

- Popular Image Datasets
- Popular NN Models (pretrained)
- Transformers



```
train_fm_set = dset.FashionMNIST(root = './fashion_mnist_data',
                                 download=True, transform=xt)
test_fm_set = dset.FashionMNIST(root='./fashion_mnist_data', train=False,
                         download=True, transform=xt) #transform=trans
```



```
test_fm_loader = torch.utils.data.DataLoader(
                dataset=test_fm_set,
                batch_size=batch_size,
                shuffle=False)
```

Pytorch PowerAI LAB –

Dustin VanStee
Bob Chesebrough
IBM WW Client Centers

Lab Code Here :

Code is here
https://github.com/dustinvanstee/pytorch-examples