Python programming

Welcome to PyTorch Tutorials

What's new in PyTorch tutorials?

- Data Loading Optimization in PyTorch
- Distributed Training with Ray Train
- Serve PyTorch models at scale with Ray Serve
- Hyperparameter tuning using Ray Tune
- Memory Profiling with Mosaic
- Using Variable Length Attention in PyTorch
- DebugMode: Recording Dispatched Operations and Numerical Debugging

https://docs.pytorch.org/tutorials/index.html

pip install numpy pandas matplotlib
pip install torch
pip install torchvision

-Quick start: https://docs.pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html

-CNN: https://docs.pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#define-a-convolutional-neural-network

-DCGAN: https://docs.pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

```
#import modules
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import v2


# Download training data from open datasets.
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=v2.Compose([v2.ToImage(),
v2.ToDtype(torch.float32, scale=True)]),
)

# Download test data from open datasets.
test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=v2.Compose([v2.ToImage(),
v2.ToDtype(torch.float32, scale=True)]),
)
```

```
#data loading
batch_size = 64

# Create data loaders.
train_dataloader = DataLoader(training_data,
batch_size=batch_size)
test_dataloader = DataLoader(test_data,
batch_size=batch_size)

for X, y in test_dataloader:
    print(f"Shape of X [N, C, H, W]: {X.shape}")
    print(f"Shape of y: {y.shape} {y.dtype}")
    break
```

```
device = torch.accelerator.current_accelerator().type if
torch.accelerator.is_available() else "cpu"
print(f"Using {device} device")

# Define model
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10)
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = NeuralNetwork().to(device)
print(model)
```

← Creating model

← See the dimension of input and output

#parameter setting2

```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-
3)
```

#training

```
def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if batch % 100 == 0:
            loss, current = loss.item(), (batch + 1) * len(X)
            print(f"loss: {loss:>7f}
[{current:>5d}/{size:>5d}]")
```

#test

```
def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) ==
y).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: ₩n Accuracy: {(100*correct):>0.1f}%,
Avg loss: {test_loss:>8f} ₩n")
```

#training

```
epochs = 5
for t in range(epochs):
    print(f"Epoch {t+1}₩n-------------------------------")
    train(train_dataloader, model, loss_fn, optimizer)
    test(test_dataloader, model, loss_fn)
print("Done!")
```

#I/O

```
torch.save(model.state_dict(), "model.pth")
print("Saved PyTorch Model State to model.pth")


model = NeuralNetwork().to(device)
model.load_state_dict(torch.load("model.pth",
weights_only=True))
```

for fun

```
classes = [
    "T-shirt/top",
    "Trouser",
    "Pullover",
    "Dress",
    "Coat",
    "Sandal",
    "Shirt",
    "Sneaker",
    "Bag",
    "Ankle boot",
]

model.eval()
x, y = test_data[0][0], test_data[0][1]
with torch.no_grad():
    x = x.to(device)
    pred = model(x)
    predicted, actual = classes[pred[0].argmax(0)],
classes[y]
    print(f'Predicted: "{predicted}", Actual: "{actual}"')
```

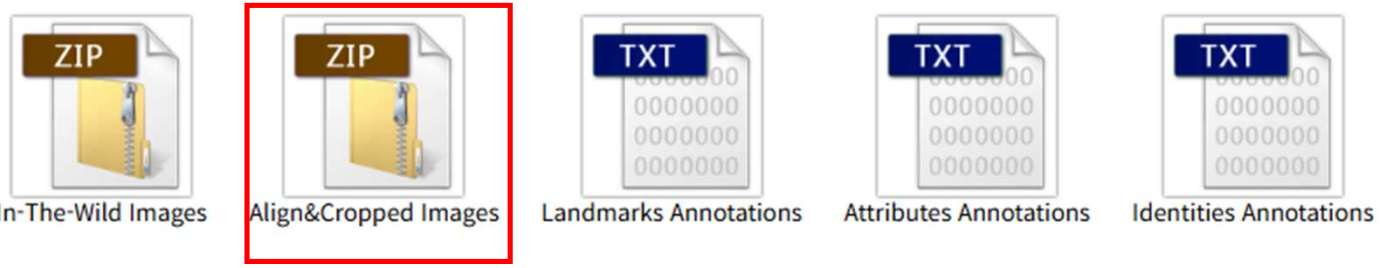
#CNN data download

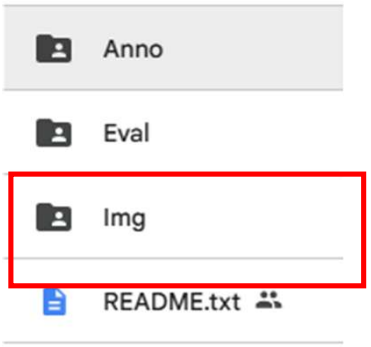
```
transform = v2.Compose([
    v2.ToImage(),
    v2.ToDtype(torch.float32, scale=True),
    v2.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

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')
```

Data

In this tutorial we will use the Celeb-A Faces dataset which can be downloaded at the linked site, or in Google Drive. The dataset will download as a file named `img_align_celeba.zip`. Once downloaded, create a directory named `celeba` and extract the zip file into that directory. Then, set the `dataroot` input for this notebook to the `celeba` directory you just created. The resulting directory structure should be:

Downloads





#DCGAN 폴더 구조

/path/to/celeba
-> img_align_celeba
-> 188242.jpg
-> 173822.jpg
-> 284702.jpg
-> 537394.jpg
...