Data Loader

&

Make NN

발표 주제

이 동헌

2018.03.02.

matplotlib

```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable
from torch.utils.data import Dataset, DataLoader

import numpy as np

import torchvision
import torchvision.transforms as transforms
```

- IPython에서 제공하는 Rich output(그림, 소리, 에니메이션)의 표현방식
- pycharm을 사용하면 %matplolib inline을 지우고 도표를 출력하고 싶은 행 밑에 plt.show()를 넣어주어야 한다.

torchvision

- Datasets
  - MNIST
  - EMNIST
  - COCO
  - CIFAR
  - ImageFolder
  - SVHN
  - 등

torchvision

```
class torchvision.transforms.Compose(transforms)
```

- 선처리를 하기 위한 함수
- 선처리를 해야할 데이터가 많을 수 있는데 이를 한 곳으로 모아 주는 역할을 한다.

```
class torchvision.transforms.ToTensor
```

- numpy.ndarray(HxWxC)인 PIL이미지를 Tensor(CxHxW)형태로 바꾸어 준다.

```
class torchvision.transforms.Normalize(mean, std)
```

```
input[channel] = (input[channel] - mean[channel]) / std[channel]
```

- 위와 같은 연산을 하여 데이터를 정규화한다.

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

- 타겟들을 모아서 텐서형태로 변형시키고, 정규화를 해주겠다는 문장

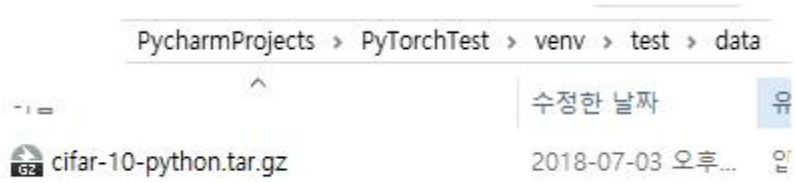
torchvision

```
class torchvision.datasets.CIFAR10(root, train=True, transform=None,
target_transform=None, download=False)
```

- root : 데이터가 어느 디렉토리에 있는지
- train : train = True면 training set, train = False면 test set
- transform : PIL 이미지를 받아 변환된 버전을 반환하는 함수
- target_transform : 타겟을 가져와 변환하는 함수
- download : download = True면 웹에서 디렉토리로 다운받는다.

```
transform = transfroms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,0.5,0.5), (0.5,0.5,0.5))])
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transfrom=transform)
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
```

- transform의 형태로 변형을 해주고, 현재 디렉토리에 data라는 폴더를 만들어(원래 존재하면 만들지 않음)다운로드를 받는다.



DataLoader

- 어떤 데이터를 학습하기 위해서 몇가지 정해주어야 하는 것들이 있는데 그 옵션들을 설정해주기 위하여 데이터를 DataLoader로 감싸주는것이다.

```
class torch.utils.data.DataLoader(dataset, batch_size=1, shuffle=False,
sampler=None, batch_sampler=None, num_workers=0, collate_fn=<function
default_collate>, pin_memory=False, drop_last=False, timeout=0,
worker_init_fn=None)  [source]
```

- dataset : 어떤 데이터를 로드할 것인지
- batch_size : 데이터를 학습하기 위해서는 batch_size를 정해주어야 하는데 8이라 정하게 되면 네트워크에 넣어주는 이미지의 크기는 8(batch) x 3(RGB) x 32 x 32(image size)로 input으로 들어가게 된다.
- shuffle : 데이터셋이 여러개일때 True이면 무작위로 input으로 넣어준다.
- num_workers : batch_size가 커지면 이미지를 가져올때 걸리는 시간이 증가한다. 이는 CPU가 처리를 하는데 CPU프로세서를 몇개를 사용할 것인지에 대한 파라미터

```
trainloader = DataLoader(trainset, batch_size=8, shuffle=True, num_workers=2)
tesetloader = DataLoader(testset, batch_size=8, shuffle=False, num_workers=2)
```

- batch_szie를 8로 해주고 이미지를 무작위로 넣어주고 프로세서는 2개를 사용한다.

Code

```
#CIFAR10에서 제공하는 클래스
classes = ('plane','car', 'bird', 'cat', 'deer', 'dpog', 'frog', 'horse', 'ship', 'truck')

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

    print(np_img.shape)
    print((np.transpose(np_img,(1,2,0))).shape)
```

- np.transpose() : img($C_{(0)}xH_{(1)}xW_{(2)}$)를 $H_{(1)}xW_{(2)}xC_{(0)}$형태로 바꾸어준다.

```
dataiter = iter(trainloader)
images, labels = iter(trainloader).next()
#아래의 형태를 더 많이 쓴다.
# for n, (img, labels) in enumerate(trainloader):
#     print(n, img.shapge, labels.shape)
```

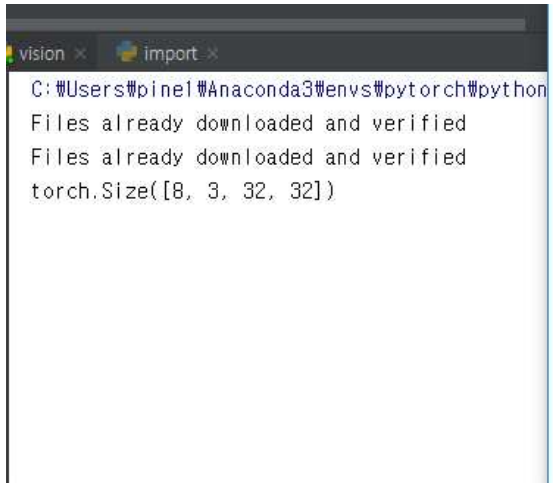
- trainloder(DataLoder)를 iterator로 변환하여 images와 labels로 담는다.
- 만약 dataiter = iter()부분에서 broken pipe가 발생한다면 num_worker를 0으로 설정해주시면 됩니다.

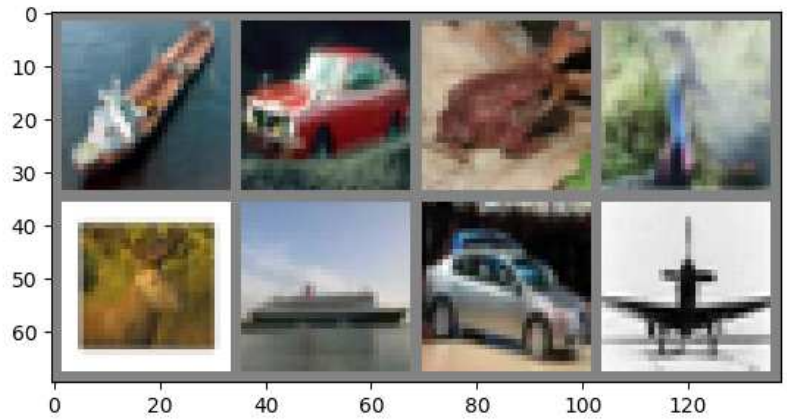
Code

```
print(images.shape)
imshow(trochvision.utils.make_grid(images, nrow=4))
print(images.shape)
print( (torchvision.utils.make_grid(images)).shape )
print(''.join('%5s ' %claseese[labels[j]] for j in range(8)))
```

torchvision.utils.make_grid(*tensor, nrow=8, padding=2, normalize=False, range=None, scale_each=False, pad_value=0*) [source]

- grid 이미지를 만들어준다.
- nrow : 출력할 그리드 이미지의 각 행에 대한 이미지 수





Code

```
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,0.5,0.5), (0.5,0.5,0.5))])
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

trainloader = DataLoader(trainset, batch_size=8, shuffle=True, num_workers=2)
testloader = DataLoader(testset, batch_size=8, shuffle=False, num_workers=2)

#CIFAR10에서 제공하는 클래스
classes = ('plane','car', 'bird', 'cat', 'deer', 'dpog', 'frog', 'horse', 'ship', 'truck')

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

    print(np_img.shape)
    print((np.transpose(np_img,(1,2,0))).shape)

dataiter = iter(trainloader)
images, labels = iter(trainloader).next()
#아래의 형태를 더 많이 쓴다.
# for n, (img, labels) in enumerate(trainloader):
#     print(n, img.shapge, labels.shape)

print(images.shape)
imshow(trochvision.utils.make_grid(images, nrow=4))
print(images.shape)
print( (torchvision.utils.make_grid(images)).shape )
print(''.join('%5s ' %claseese[labels[j]] for j in range(8)))
```

Using my data

- 자신의 데이터를 사용하고 싶다면 해당 폴더에 사진들을 저장을 하고

```
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,0.5,0.5), (0.5,0.5,0.5))])
trainset = torchvision.datasets.ImageFolder(root='./data', transform=trans)
```

- 위 코드와 같이 ImageFolder를 사용해주면 된다.
  - root : 사진 혹은 사진들이 저장되어있는 폴더가 위치한 경로
  - transform : 데이터를 선처리할 함수

Network

- Class를 선언하여 사용하고자 하는 Neural Network를 직접 구현할 수 있다.
- Calss 선언시 필수 요소
- class my_network(nn.Module)
  - def __init__(self):
    - super(class_name, self).__init__()

      (사용할 함수들을 정의할 장소)
  - def forward(self, variable):
    - 함수들을 사용하여 Network의 forward를 정의할 장소

Model

- 모델을 만들기 전에 Input을 무엇을 넣을 것인지를 확인하여야 한다.
- 8(batch)x3(RGB)x32(h)x32(w)인 이미지를 nn.Conv2d에 넣는다면, Conv2d는 in_channels과 out_channels, kernel_size를 요구한다.
  - in_channels : 3RGB을 넣어주면 된다.
  - out_channels : 사용할 filter의 갯수로 5개를 사용하고자하면 5를 입력해주면 된다.
  - kernel_size : 사용할 filter의 크기 5x5를 사용하고자하면 5 입력.

Code

```
class my_network(nn.Module):
    def __init__(self):
        super(my_network, self).__init__()
        self.net_1 = nn.Conv2d(3,5,5)
        self.net_2 = nn.Conv2d(5,10,5)

    def forward(self, x):
        x = self.net_1(x)
        x = self.net_2(x)
        return x
imgs = 0
for n, (img, labels) in enumerate(trainloader):
    print(n, img.shape, labels.shape)
    imgs = img
    break
```

- 8x3x32x32에 5x5 filter를 적용시키면 8x5x28x28이 된다. 즉, batch는 유지되고 채널이 filter의 갯수만큼 바뀌고 크기는 flter의 크기가 5x5이므로 32-4=28이 된다.
- 다음 레이어의 input은 채널이 5가 되었으므로 5가되고 그 뒤는 동일하다,

```
0 torch.Size([8, 3, 32, 32]) torch.Size([8])
torch.Size([8, 5, 28, 28])
torch.Size([8, 10, 24, 24])
```

Code

```
class my_network(nn.Module):
    def __init__(self):
        super(my_network, self).__init__()
        self.net_1 = nn.Conv2d(3,5,5)
        self.net_2 = nn.Conv2d(5,10,5)

    def forward(self, x):
        x = self.net_1(x)
        x = self.net_2(x)
        return x
imgs = 0
for n, (img, labels) in enumerate(trainloader):
    print(n, img.shape, labels.shape)
    imgs = img
    break

my_net = my_network()
out = my_net(Variable(imgs))
print(out.shape)
```

```
Files already downloaded and verified
Files already downloaded and verified
0 torch.Size([8, 3, 32, 32]) torch.Size([8])
torch.Size([8, 10, 24, 24])
```