Lab Tutorials

Ngọc Thuận - IPSAL LAB

24, December 2022

Tóm tắt nội dung

Đây là những ghi chú, lưu ý của tôi trong quá trình làm Lab-Tutorials, mục đích là để ghi lại những ý chính nhất, phục vụ cho việc nghiên cứu sau này!

1 Lec1_Tuto1_Lab01_dataset

Dataset là một trong những đối tượng cơ bản, nhưng vô cùng quan trọng trong Trí tuệ nhân tạo nói chung, và các nhóm lĩnh vực nhỏ hơn của nó nói riêng. Ta đã biết một số Dataset có sẵn được lưu trữ trong các thư viện phổ biến như: MNIST, Iris, CIFAR, ... Tuy nhiên thực tế lại không xoay quanh những dữ liệu có sẵn, bài này sẽ giới thiệu về cách sử dụng các cơ sở dữ liệu có sẵn và cách để tự tạo một cơ sở dử liệu của riêng mình!

1.1 Sử dụng các Dataset của torchvision

Đơn giản nhất ta làm như sau, ví dụ với CIFAR10:

```
from torchvision.datasets import CIFAR10
data_train = CIFAR10('../cifar10_data2', train = True, download = True)
```

Khi đó, *data_ train* sẽ là tập dữ liệu huấn luyện được *import* từ *torchvision.datasets*.
Các *elements* của tập dữ liệu huấn luyện được biểu diễn dưới dạng *tuple*: *(image, label)*.
Các lệnh hay đối số khác, ngoài ví dụ trên sẽ tự nhiên cứu lại nếu cần thiết.
Link: DATASETS - PyTorch.

1.2 Tự tạo Datasets cho riêng mình, sử dụng ImageFolder Class.

Bỏ qua những râu ria, ta những gì ta có thể tự làm không cần code thì nên chú ý để thực hiện một cách nhanh chóng và hiệu quả.
Sau khi làm bài này, tôi nhận thấy, để tạo một cơ sở dữ liệu, ta sẽ làm các bước tiền xử lí bên ngoài như sau:

1. Đầu tiên tạo một *Folder* riêng cho *Dataset* đó.
2. Bên trong tạo các *Folder* con, đó chính là nơi lưu các lớp dữ liệu (*labels*).
3. Cuối cùng là lưu các ảnh vào đúng các lớp tương ứng.

Chú ý: Không nên lẫn lộn nhiều kiểu dữ liệu, và các tên nên đặt thân thuộc, có chỉ số, để phục vụ cho việc import ảnh.

Đến đây để có thể biến cơ sở dữ liệu trên giống như các cơ sở dữ liệu có sẵn thì ta sử dụng *ImageFolder*, ví dụ như sau:

```
data_train = torchvision.datasets.ImageFolder(data_tuto1_root)
```

Với đối số và *root* của cơ sở dữ liệu vừa tạo.
Sau đó thì mọi thứ y hệt cơ sở dữ liệu có sẵn rồi!

1.3 Data Loading in Pytorch - DataLoader

Chỗ này tôi chưa hiểu lắm nên không thể ghi lại rõ ràng được.
Nhưng tôi biết đây là bước tiền xử lí số liệu, chẳng hạn như việc tách tập huấn luyện thành các tập nhỏ, tránh quá khớp chẳng hạn, hay đơn giản là muốn thử nghiệm trên số lượng nhỏ.
Có thể gộp nhưng tôi sẽ làm tách riêng, để rõ ràng trong cả trường hợp với cơ sở dữ liệu tự tạo. Ví dụ:

```
data_train = MNIST(' /mnist_data', train = True, download = True, transform = transforms.Compose([transforms.ToTensor()])
data_train_batch_2 = torch.utils.data.DataLoader(data_train, batch_size = 200, shuffle = False)
```

Đến đấy *data_train_batch_2* khá giống bên trên, đều chứa dữ liệu và nhãn. Tuy nhiên để sử dụng phải dùng đến *Variable* của *module torch.autograd*. Dù chưa hiểu nhưng ta cứ xem ví dụ sau:

```
from torch.autograd import Variable
for i, (images, labels) in enumerate(data_train_batch_2):
    images = Variable(images)
    labels = Variable(labels)
```

Sau đó thì làm bình thường, tuy nhiên chú ý rằng một số *Dataset* lưu ảnh xám dưới dạng mảng 3 chiều, đơn kênh. Khi đó chú ý *reshape* ảnh hoặc sử dụng *squeeze method*.

2 Lec1_Tuto1_Lab01_dataset_custom

Với bài này ta hiểu rằng mục đích chính là định dạng các *Dataset* theo ý mình. Tức ta sẽ tạo ra một *class* để làm điều ấy, và *class* ấy tối thiểu cần làm được 3 nhiệm vụ sau:

1. nhập dữ liệu từ các *Dataset* gốc. (thông qua *root* là chủ yếu).
2. có thể truy cập vào ra tập dữ liệu đã được nhập.
3. trả lại số lượng của tập dữ liệu.

Thật vậy, ứng với nó ta cần tối thiểu 3 hàm lần lượt là: *__init__* (inherits từ *torch.utils.data.Dataset*), *__getitem__(self, idx)* và *__len__*.

2.1 Nhắc lại về Inheration

Các lớp trong *Python* có tính kế thừa nhau nếu ta gán cho chúng các tập bố. Khi đó các tập con sẽ hưởng hết tất cả tính chất, phương thức của hàm bố nếu không có gì thay đổi.
Để khởi tạo lại cho tập con, buộc phải có dữ liệu dành cho tập bố. Khi đó ta sử dụng *super().__init__(*args, **kwargs)*. Với các hàm và phương thức, nếu thay đổi trên tập con thì khi gọi tập con, kết quả sẽ theo hàm, phương thức được gọi trên tập con.
Dựa vào hiểu biết này, ta có thể tận dụng lớp *Dataset* có sẵn, xịn sò của *PyTorch* và chỉ cần thay đổi những cái cơ bản như trên thỏa mãn ý của chúng ta là được!
Ví dụ:

```
class ImageDataset_segment(torch.utils.data.Dataset): # inheritation.
    def __init__(self, root, N, *args, **kwargs):
        super().__init__(*args, **kwargs)
```

2.2 Các chú ý khác

Bài này, những phần trên ta có thể xem lại nhưng một số cái ta phải nhớ!

2.2.1 Contour cho phân đoạn ảnh

Khá bất ngờ, đơn giản ta chỉ việc vẽ ảnh gốc và ảnh đã phân đoạn chồng lên nhau thôi.

2.2.2 Lưu dữ liệu dưới dạng .npy hay .pt

Cần phải tạo ra các tập dữ liệu để chứa chúng, kênh đầu tiên là số lượng dữ liệu, các kênh sau tùy thuộc vào *numpy* hay *torch* mà xử lí *np.save* hay *torch.save*.

3 Link Colab

1. Tuto 1
2. Tuto 2