

fast.ai 스터디 2주차 일요일

Gathering Data,
From Data to DataLoaders,
Training Your Model, and Using It to Clean Your Data

2020. 09. 06. 최민영

Gathering Data - 1

- 프로젝트를 진행하려면 데이터가 필요하다!
 - 우리가 진행할 프로젝트는 *'bear detector'*
 - 곰 사진들이 필요하다.
 - Bing에서 제공하는 API를 이용하자.

```
[ ] key = '실제 키값을 저장하면 됩니다.'
```

Or, if you're comfortable at the command line, you can set it in your terminal with:

```
export AZURE_SEARCH_KEY=your_key_here
```

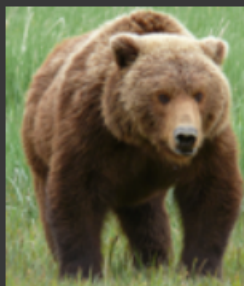
and then restart Jupyter Notebook, type this in a cell and execute it:

```
key = os.environ['AZURE_SEARCH_KEY']
```

Gathering Data - 2

- 이미지 다운로드 테스트

```
[ ] #hide  
ims = ['http://3.bp.blogspot.com/-S1scR0kI3vY/UHzY2kucsPI/AAAAAAAAA-k/YQ5UzHEm9Ss/s1600/Grizzly%2BBear%2BWildlife.jpg']  
  
[ ] dest = 'images/grizzly.jpg'  
download_url(ims[0], dest)  
  
[ ] im = Image.open(dest)  
im.to_thumb(128,128)
```



Gathering Data - 3

- Bing API 사용예제

```
bear_types = 'grizzly','black','teddy' # 튜플로 저장
path = Path('bears') # 주피터 노트북을 실행한 경로에서 bears 라는 서브 디렉토리를 나타냄

if not path.exists(): # ./bears 가 존재하지 않으면,
    path.mkdir() # ./bears 를 생성하고
    for o in bear_types: # ('grizzly','black','teddy') 를 순회한다.
        dest = (path/o) # e.g. dset == './bears/grizzly'
        dest.mkdir(exist_ok=True) # './bears/grizzly' 디렉토리 생성
        results = search_images_bing(key, f'{o} bear') # bing 검색 결과 저장
        download_images(dest, urls=results.attrgot('content_url')) # 검색 결과에서 'content_url'에 해당하는 값만 가져와 urls에 넣는다.
```

```
fns = get_image_files(path)
fns
```

```
(#438) [Path('bears/grizzly/00000000.jpg'),Path('bears/grizzly/00000001.jpg'),
```

Gathering Data - 3

- 데이터 정리

```
failed = verify_images(fns)    # 이미지 파일이 아닌 애들의 경로 리스트를 반환한다.  
failed
```

```
(#15) [Path('bears/grizzly/00000028.jpg'), Path('bears/grizzly/00000048.jpg'), Path('b
```

```
failed.map(Path.unlink);    # failed에 저장된 모든 항목에 'Path.unlink' 함수를 적용한다.
```

From Data to DataLoaders - 1

```
class DataLoaders(GetAttr):  
    def __init__(self, *loaders): self.loaders = loaders  
    def __getitem__(self, i): return self.loaders[i]  
    train, valid = add_props(lambda i, self: self[i])
```

* 알아두면 좋은 몇가지 파이썬 문법들

1. dunder
2. unpack
3. Class attribute

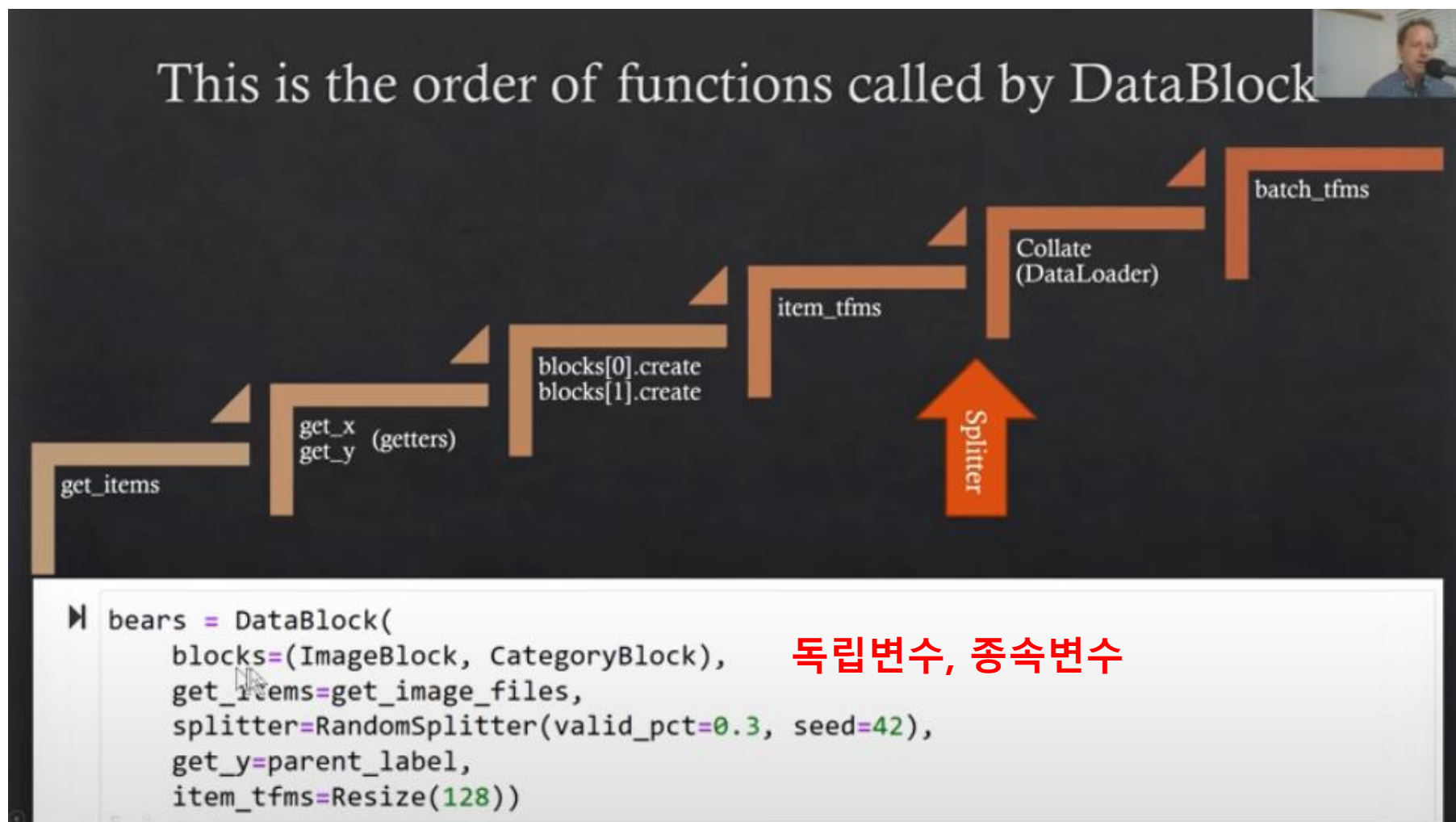
DataLoaders 클래스로 알 수 있는 것들

1. 어떤 데이터를 다룰 것인가
2. 리스트를 얻는 방법
3. 아이템에 라벨링을 하는 방법
4. Training, validation 셋을 만드는 법

From Data to DataLoaders - 2

- DataLoaders 에게 추가로 알려줘야 할 정보들
 - 어떤 데이터를 사용할지
 - 어떻게 데이터를 얻는지
 - 어떻게 데이터에 라벨링 할지
 - 어떻게 validation 셋을 정할지
- **Data block API !**
- > DataLoaders를 생성하는 각 스텝을 손쉽게 커스터마이징 할 수 있다.

From Data to DataLoaders - 3



From Data to DataLoaders - 4

- DataBlock은 DataLoaders을 만들어주는 템플릿이다.
 - DataBlock 재사용하는 예제들이 뒤에 나옴.
- 템플릿이 정의 되면, DataLoaders를 생성할 수 있다.

```
dls = bears.dataloaders(path)
```

```
dls.valid.show_batch(max_n=4, nrows=1)
```

grizzly



grizzly



teddy



grizzly



From Data to DataLoaders - 5

```
bears = bears.new(item_tfms=Resize(128, ResizeMethod.Squish))  
dls = bears.dataloaders(path)  
dls.valid.show_batch(max_n=4, nrows=1)
```



```
bears = bears.new(item_tfms=Resize(128, ResizeMethod.Pad, pad_mode='zeros'))  
dls = bears.dataloaders(path)  
dls.valid.show_batch(max_n=4, nrows=1)
```



From Data to DataLoaders - 5

- 실제로 많이 사용하는 방법

```
bears = bears.new(item_tfms=RandomResizedCrop(128, min_scale=0.3))  
dls = bears.dataloaders(path)  
dls.train.show_batch(max_n=4, nrows=1, unique=True)
```



From Data to DataLoaders - 5

- Data Augmentation
 - rotation, flipping, perspective warping, brightness changes and contrast changes.

```
bears = bears.new(item_tfms=Resize(128), batch_tfms=aug_transforms(mult=2))  
dls = bears.dataloaders(path)  
dls.train.show_batch(max_n=8, nrows=2, unique=True)
```



Training Your Model, and Using It to Clean Your Data - 1

```
: bears = bears.new(  
    item_tfms=RandomResizedCrop(224, min_scale=0.5),  
    batch_tfms=aug_transforms())  
dls = bears.dataloaders(path)
```

We can now create our Learner and fine-tune it in the usual way:

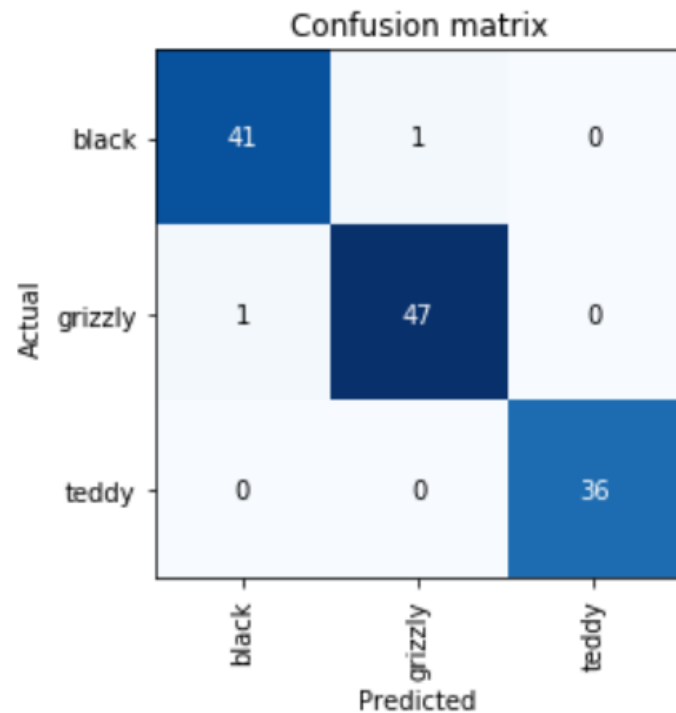
```
: learn = cnn_learner(dls, resnet18, metrics=error_rate)  
learn.fine_tune(4)
```

epoch	train_loss	valid_loss	error_rate	time
0	1.235733	0.212541	0.087302	00:05

epoch	train_loss	valid_loss	error_rate	time
0	0.213371	0.112450	0.023810	00:05
1	0.173855	0.072306	0.023810	00:06
2	0.147096	0.039068	0.015873	00:06
3	0.123984	0.026801	0.015873	00:06

Training Your Model, and Using It to Clean Your Data - 2

```
interp = ClassificationInterpretation.from_learner(learn)
interp.plot_confusion_matrix()
```



Training Your Model, and Using It to Clean Your Data - 3

```
interp.plot_top_losses(5, nrow=1)
```

Prediction/Actual/Loss/Probability

grizzly/black / 1.37 / 0.74



black/grizzly / 0.94 / 0.61



black/black / 0.56 / 0.57



grizzly/grizzly / 0.14 / 0.87



grizzly/grizzly / 0.11 / 0.90




Training Your Model, and Using It to Clean Your Data - 4

```
[ ] #hide_output
cleaner = ImageClassifierCleaner(learn)
cleaner
```

black ▼

Train ▼



<Delete> ▼ <Delete> ▼ black ▼ <Delete> ▼

◀

```
▶ #hide
for idx in cleaner.delete(): cleaner.fns[idx].unlink()
for idx,cat in cleaner.change(): shutil.move(str(cleaner.fns[idx]), path/cat)
```