🧑‍🏫 DataBlock and Dataloaders

What are Dataloaders?

> Dataloaders are replacements for `Pytorchs` DataLoader Class, but with more functionality and flexibility. They help you to investigate, clean, change and prepare you data.
>
> - You need a DataLoader to build your `Fastai` model
> - needs at least 1 argument: `source="path_to_dataset`
> - think of it as a recipe about the steps you took from getting your data till using your data
>   - often: new data will be used in the future to improve the model
>   - a DataLoader is then like a pipeline you can throw in your data without needing to remember every step
> - DataLoader requires DataBlocks 👇

> class DataLoader(**dataset**=None, **bs**=None, **num_workers**=0, **pin_memory**=False, **timeout**=0, **batch_size**=None, **shuffle**=False, **drop_last**=False, **indexed**=None, **n**=None, **device**=None, **persistent_workers**=False, **wif**=None, **before_iter**=None, **after_item**=None, **before_batch**=None, **after_batch**=None, **after_iter**=None, **create_batches**=None, **create_item**=None, **create_batch**=None, **retain**=None, **get_idxs**=None, **sample**=None, **shuffle_fn**=None, **do_batch**=None) :: GetAttrclass

✏️ source=*"path_to_dataset"* (if not already set in DataBlock)

What are Datablocks?

> DataBlocks are blueprints on how to assemble data.
>
> - necessary for building DataLoaders
> - consist of at least 4 parts:
>   1. data type (e.g. images or categorical data)
>   2. location (e.g. file path)
>   3. feature and labels (how to retrieve input data and labels)
>   4. validation set (percentage of images held back for validation)

> class DataBlock(**blocks**=None, **dl_type**=None, **getters**=None, **n_inp**=None, **item_tfms**=None, **batch_tfms**=None, **get_items**=None, **splitter**=None, **get_y**=None, **get_x**=None)

✏️ blocks ✏️ get_items ✏️ get_y ✏️ splitter

Why do we need DataLoaders?

> It is said: up to 90% of coding time belongs to data cleaning
>
> - more data handling functionalities → less time for cleaning the data
> - Fast.ai requires it

Workflow for Datasetup:

1️⃣ `Import Libraries` → 2️⃣ `Download/Source your data` → 3️⃣ `DataBlock` → 4️⃣ `DataLoader` → 5️⃣ `Data Munging` → 6️⃣ `Build Model` → 7️⃣ `Data Munging again` ... 🚀 *Deploy/Play/Use*

🎬 Simple Use Case: ugly cat classifier 🙀🐈

1️⃣ Import Libraries

Import the usual suspects ... 👻🧛‍♀️🧟

```
# Standard library imports
import os

# Third party imports
# -

# Model specific imports
# I assume you have fastai installed (if not: uncomment the next line or check o
# You also need to do this in GoogleColab
#!pip install -Uqq fastbook

from fastbook import * #all fastai and more
from fastai.vision.widgets import * #ImageCleaner
# Local application imports
# -

# Global notebook settings
%matplotlib inline
```

```
     |████████████████████████████████| 727kB 9.1MB/s
     |████████████████████████████████| 51kB 8.0MB/s
     |████████████████████████████████| 204kB 19.4MB/s
     |████████████████████████████████| 1.2MB 19.2MB/s
```

```
|████████████████████████████████| 61kB 8.9MB/s
```

And some new friends ... 🧞‍♀️🧚🦄

```
# You probably haven't DuckDuckGoImages (Source: https://pypi.org/project/DuckDu
#So just pip-install it by uncommenting the next line:
#!pip install DuckDuckGoImages
#import DuckDuckGoImages as ddg
import torch.cuda #for checking GPU available
```

If running in `Google Colab` you might want to connect to your `Google Drive`:

```
# UNCOMMMENT TO MOUNT YOUR GOOGLE DRIVE
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

2️⃣ Download Data

> Create folders with the name of the labels and save/download the corresponding images into these folders. If you run this notebook on `Google Colab` replace all paths with `content/drive/MyDrive/downloads/cats` or something similar.

```
# Create a constant to the path to our data
#PATH_TO_DATA = "downloads/cats"
PATH_TO_DATA = "/content/drive/MyDrive/fastai/unpackai/cats" #within GoogleColab
```

```
# Make directories for images
! mkdir -p "downloads/cats/prettycats"
! mkdir -p "downloads/cats/uglycats"
```

```
%%time
# Download Images from DuckDuckGo Image Search
ddg.download('pretty cat', max_urls=400, folder=PATH_TO_DATA+"/prettycats")
ddg.download('ugly cat', max_urls=400, folder=PATH_TO_DATA+"/uglycats")
```

```
CPU times: user 46.6 s, sys: 4.78 s, total: 51.4 s
Wall time: 17min 57s
370
```

```
print("pretty cat files: ", len(os.listdir(PATH_TO_DATA+"/prettycats")))
print("ugly cat files: ", len(os.listdir(PATH_TO_DATA+"/uglycats")))
```

```
pretty cat files:  335
ugly cat files:  369
```

3️⃣ Build DataBlock

```
# Create DataBlock
cats = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    splitter=RandomSplitter(valid_pct=0.2, seed=30), # 20% validation set / seed
    get_y=parent_label,
    item_tfms=Resize(128))
```

```
# Summary of DataBlock
cats.summary(PATH_TO_DATA)
```

```
Setting-up type transforms pipelines
Collecting items from /content/drive/MyDrive/fastai/unpackai/cats
Found 698 items
2 datasets of sizes 559,139
Setting up Pipeline: PILBase.create
Setting up Pipeline: parent_label -> Categorize -- {'vocab': None, 'sort': Tru

Building one sample
  Pipeline: PILBase.create
    starting from
      /content/drive/MyDrive/fastai/unpackai/cats/prettycats/34679cb825c1460d9
    applying PILBase.create gives
      PILImage mode=RGB size=480x480
  Pipeline: parent_label -> Categorize -- {'vocab': None, 'sort': True, 'add_
    starting from
      /content/drive/MyDrive/fastai/unpackai/cats/prettycats/34679cb825c1460d9
    applying parent_label gives
      prettycats
    applying Categorize -- {'vocab': None, 'sort': True, 'add_na': False} give
      TensorCategory(0)

Final sample: (PILImage mode=RGB size=480x480, TensorCategory(0))


Collecting items from /content/drive/MyDrive/fastai/unpackai/cats
Found 698 items
2 datasets of sizes 559,139
Setting up Pipeline: PILBase.create
Setting up Pipeline: parent_label -> Categorize -- {'vocab': None, 'sort': Tru
Setting up after_item: Pipeline: Resize -- {'size': (128, 128), 'method': 'cro
Setting up before_batch: Pipeline:
Setting up after_batch: Pipeline: IntToFloatTensor -- {'div': 255.0, 'div_mas

Building one batch
Applying item_tfms to the first sample:
  Pipeline: Resize -- {'size': (128, 128), 'method': 'crop', 'pad_mode': 'refl
    starting from
      (PILImage mode=RGB size=480x480, TensorCategory(0))
    applying Resize -- {'size': (128, 128), 'method': 'crop', 'pad_mode': 'ref
      (PILImage mode=RGB size=128x128, TensorCategory(0))
    applying ToTensor gives
      (TensorImage of size 3x128x128, TensorCategory(0))
```

```
Adding the next 3 samples

No before_batch transform to apply

Collating items in a batch

Applying batch_tfms to the batch built
  Pipeline: IntToFloatTensor -- {'div': 255.0, 'div_mask': 1}
    starting from
      (TensorImage of size 4x3x128x128, TensorCategory([0, 0, 0, 0], device='c
    applying IntToFloatTensor -- {'div': 255.0, 'div_mask': 1} gives
      (TensorImage of size 4x3x128x128, TensorCategory([0, 0, 0, 0], device='c
```

4️⃣ Build DataLoader

```
# Create DataLoader
dls = cats.dataloaders(source=PATH_TO_DATA)
```

5️⃣ Data Munging I: Investigate, Clean, Change the Data

Investigate Data

Find and eliminate broken Images

It is very likely that you downloaded some broken image files without knowing in. The next line of code helps you to identify it:

```
%%time
#check for broken images and delete if necessary
fns = get_image_files(PATH_TO_DATA)
failed = verify_images(fns)
print(failed) #show broken images
```

```
[]
CPU times: user 240 ms, sys: 124 ms, total: 364 ms
Wall time: 3.1 s
```

```
failed.map(Path.unlink) #unlink broken images
```

```
(#17) [None,None,None,None,None,None,None,None,None,None...]
```

This actually does not delete the images. It unlinks the images from the `DataLoader`. For this reason we need to run the `DataLoader` again:

```
# Rerun DataLoader after unlinking broken images
dls = cats.dataloaders(source=PATH_TO_DATA)
```

Show Images in batches

```
dls.train.show_batch(max_n=6, nrows=1)
```

```
dls.valid.show_batch(max_n=6, nrows=1)
```

Show Labels

```
#Show labels
dls.vocab
```

```
['prettycats', 'uglycats']
```

Show Paths

```
# investigate training/validation dataset
print("Train:\n=============\n",dls.train.dataset)
print("\nValid:\n=============\n", dls.valid.dataset)
print("\n2nd Image in Train:\n=============\n", dls.train.dataset[1])
```

```
Train:
=============
 (#559) [(PILImage mode=RGB size=480x480, TensorCategory(0)),(PILImage mode=RG

Valid:
=============
 (#139) [(PILImage mode=RGB size=800x548, TensorCategory(1)),(PILImage mode=RG

2nd Image in Train:
```

```
=============
 (PILImage mode=RGB size=1004x938, TensorCategory(0))
```

```
# Print path of images (here in validation set)
dls.valid.items[0:5]
```

```
[Path('/content/drive/MyDrive/fastai/unpackai/cats/uglycats/9a385a25eb5e467198
 Path('/content/drive/MyDrive/fastai/unpackai/cats/uglycats/877c14baf6904855b3
 Path('/content/drive/MyDrive/fastai/unpackai/cats/uglycats/ea9c41f724994a83b6
 Path('/content/drive/MyDrive/fastai/unpackai/cats/prettycats/9e38e45f8333403a
 Path('/content/drive/MyDrive/fastai/unpackai/cats/prettycats/0ac1ddbdfee143e5
```

Print number of Images

```
# Size of Datasets
print("Train:\t", dls.train.n)
print("Val:\t", dls.valid.n)
print("Total:\t", dls.train.n+dls.valid.n)
```

```
Train:   559
Val:     139
Total:   698
```

6️⃣ Train Model

The following cell will train a basic image classifier based on the so called `resnet18` model. As a I am not focussing in this notebook on model building I only show this for the solely purpose to show how to use `ImageClassifierCleaner()` - a useful class in *fastai* to delete/reclassify images in you *DataLoader*.

> 🧐 The execution time of the following code might take longer on your computer depending wether you can have GPU available or not. You can run the next cell to find out which processor is available for you. If you don't have an appropriate GPU (cuda) the execution time might be too long and you should run this notebook on `GoogleColab` instead.

```
if torch.cuda.is_available():
    print('cuda available 🕺💃')
else:
    print('could not find cuda 🙁')
```

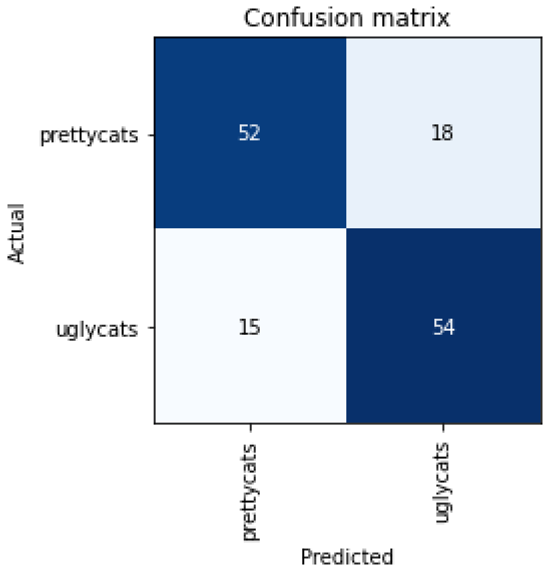
```
cuda available 🕺💃
```

```
%%time
learn = cnn_learner(dls=dls, arch=resnet18, metrics=accuracy) #build a Deep Lear
learn.fit_one_cycle(2) #start model over 2 epochs
```

epoch	train_loss	valid_loss	accuracy	time
0	1.240832	0.620930	0.762590	00:11
1	1.005913	0.501392	0.762590	00:11

```
CPU times: user 1.68 s, sys: 383 ms, total: 2.06 s
Wall time: 24.1 s
```

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



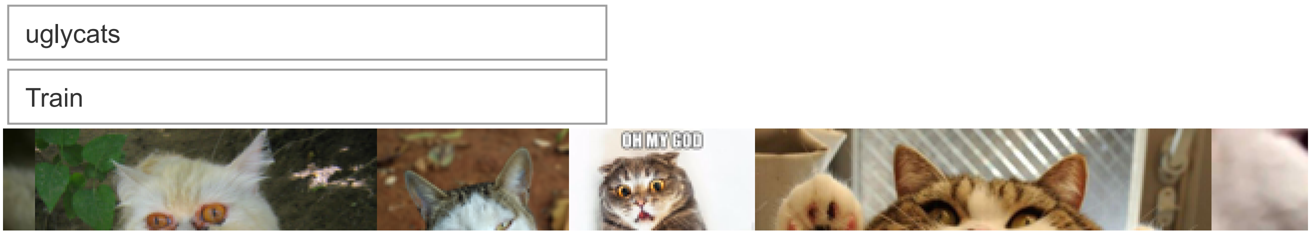
A lot of cats are wrongly classified. But this might be due to the bad quality of our downloaded images. The following code helps with that ...

7️⃣ Data Munging II: Delete/Reclassify Images with `DataLoader`

Manually delete/reclassify images

Running the following cell gives you for every misclassified image a dropdown menu where you can decide wether to keep that image, delete it, or put in into another category.

```
cleaner = ImageClassifierCleaner(learn)
cleaner
```



uglycats

Train

After marking all images you need to run the next to let the change happen. But before lets print again our stats about the image files:

```
print("pretty cat files: ", len(os.listdir(PATH_TO_DATA+"/prettycats")))
print("ugly cat files: ", len(os.listdir(PATH_TO_DATA+"/uglycats")))
```

```
pretty cat files:  333
ugly cat files:  364
```

```
#this code unlinks and then DELETE all selected images in the cleaner

for category in dls.vocab:
  n=0
  for idx in cleaner.delete():
      n+=1
      try:
        cleaner.fns[idx].unlink()
      except:
        pass
      cleaner.delete
  print("Deleted {} in {}".format(n,category))
```

```
Deleted 1 in prettycats
Deleted 1 in uglycats
```

```
print("pretty cat files: ", len(os.listdir(PATH_TO_DATA+"/prettycats")))
print("ugly cat files: ", len(os.listdir(PATH_TO_DATA+"/uglycats")))
```

```
pretty cat files:  331
ugly cat files:  363
```

```
#this code UNCATEGORIZE all selected images in the cleaner
for category in dls.vocab:
  n=0
  for idx,category in cleaner.change():
    try:
      shutil.move(str(cleaner.fns[idx]), PATH_TO_DATA+"/"+category)
    except:
      pass
    n+=1
  print("Recategorized {} in {}".format(n,category))
```

```
Recategorized 1 in prettycats
Recategorized 1 in prettycats
```

```
print("pretty cat files: ", len(os.listdir(PATH_TO_DATA+"/prettycats")))
print("ugly cat files: ", len(os.listdir(PATH_TO_DATA+"/uglycats")))
```

```
pretty cat files:  329
ugly cat files:  361
```

📚 References

- [Fast.Ai **DataLoader** Documentation]()
- [Fast.Ai **DataBlock** Documentation]()
- [Practical Deep Learning for Coders]()

⛹️Coding Backyard

> 🧑‍🏫 I use this section for testing code, finding errors, trying out new code, etc as I usually like to keep code snippets for future use.

```
# Show sample b&w images in training set

# Create DataBlock
cats_bw = DataBlock(
    blocks=(ImageBlock(cls=PILImageBW), CategoryBlock), #cls=PILImageBW
    get_items=get_image_files,
    splitter=RandomSplitter(valid_pct=0.2, seed=42),
    get_y=parent_label,
    item_tfms=Resize(128))

# Create DataLoader
dls_bw = cats_bw.dataloaders(source = PATH_TO_DATA)

dls_bw.train.show_batch(max_n=6, nrows=1)
```

```

```

✓ 0 s Abgeschlossen um 16:25 ● ×