Week2 Introduction

Tutor:
Email:
Tutorial:
Code: https://github.com/Jinxu-Lin/COMP5329

Matrix Review

Vector

- Column Vector: $\mathbf{a} = [a_1, a_2, \ldots, a_m]^\top (m \times 1)$
- Row Vector: $\mathbf{b} = [b_1, b_2, \ldots, b_n]$ $(1 \times n)$
- Production:
- $ab = [[a_1b_1, a_1b_2, \ldots, a_1b_n], [a_2b_1, a_2b_2, \ldots, a_2b_n], \ldots, [a_mb_1, a_mb_2, \ldots, a_mb_n]]$ with shape of $(m \times n)$
- $$ba = \sum_{i=1}^{m} a_m b_m (m = n)$$

Matrix

- Define $A \in \mathbf{R}^{m \times n}, B \in \mathbf{R}^{m \times n}$
- Add: $C = A + B \in \mathbf{R}^{m \times n}$, which means $C_{i,j} = A_{i,j} + B_{i,j} \forall i, j$
- Subtract: $C = A - B \in \mathbf{R}^{m \times n}$, which means $C_{i,j} = A_{i,j} - B_{i,j} \forall i, j$
- Element-wise Multiplication: $C = A \odot B \in \mathbf{R}^{m \times n}$, $C_{i,j} = A_{i,j} * B_{i,j} \forall i, j$

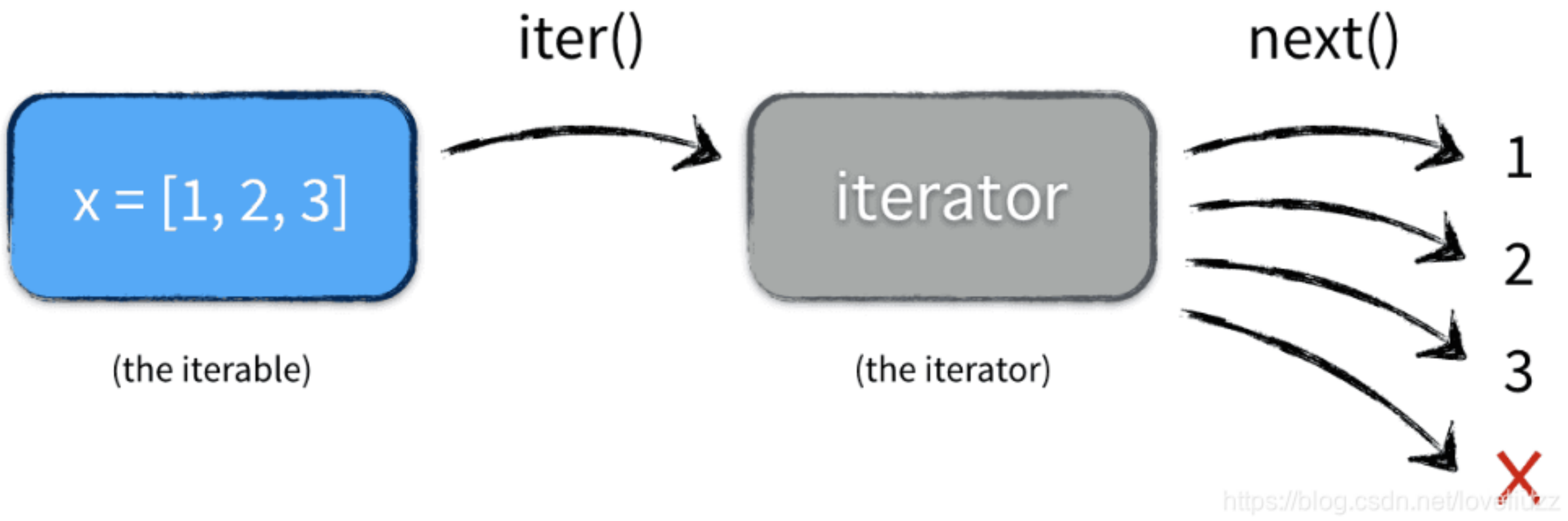
Matrix Production

- Define $A \in \mathbf{R}^{m \times n}, B \in \mathbf{R}^{n \times p}$
- $C = A \times B \in \mathbf{R}^{m \times p}$
- Such that $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} = a_{i1} b_{1j} + a_{i2} b_{2j} + \ldots + a_{in} b_{nj}$, which means c_{ij} is the production of the i-th row vector in A and the j-th column vector in B

Dataset and Dataloader

Dataset in PyTorch

- *# Step1: Construct datasets*
- dataset = MyDataset()
- *# Step2: Construct iterator (dataloader)*
- dataloader = DataLoader(dataset)
- num_epoches = 100
- for epoch in range(num_epoches):
-     for i, data in enumerate(dataloader):
-         *# Train !*
-



Dataset in PyTorch

- The *`Dataset`* class must override the *`__len__()`* and *`__getitem__()`* methods
- *`__init__(self)`*: Primarily used for data acquisition, such as loading data from a file.
- *`__len__(self)`:* Returns the total number of samples in the dataset.
- *`__getitem__(self, index)`: Implements dataset indexing, allowing retrieval of individual data samples.*

Dataset in PyTorch

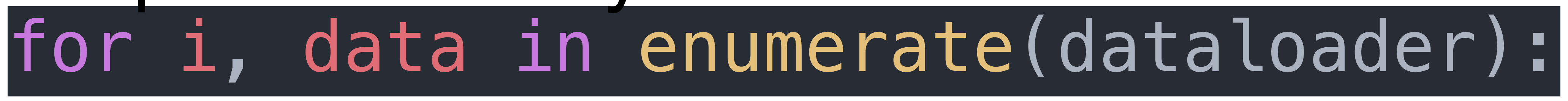
- Accessing `*dataset[i]*` returns the $(i+1)$-th data point. If the class defines the `*__getitem__()*` method, an instance (e.g., `*p*`) can use `*p[key]*` to retrieve values. When `*p[key]*` is accessed, Python automatically calls the `*__getitem__()*` method.

- By overriding `*__getitem__()*`, data can be accessed via an index, returning both the **data** and its corresponding **label**. Both should be returned as **Tensor** objects.

Dataloader in PyTorch

- The **DataLoader** in PyTorch is a tool used to handle input data for models. It combines a **dataset** and a **sampler**, providing an iterable object that supports both single-threaded and multi-threaded (`*num_workers*`) data loading.

- Ways to Access DataLoader:
    - Using `*iter()*` (Recommended):
        - The `*Dataloader*` is inherently an iterable object, meaning it should be accessed using `*iter()*`
    - It cannot be accessed directly with `*next()*`.
    - The preferred way to iterate over it is:

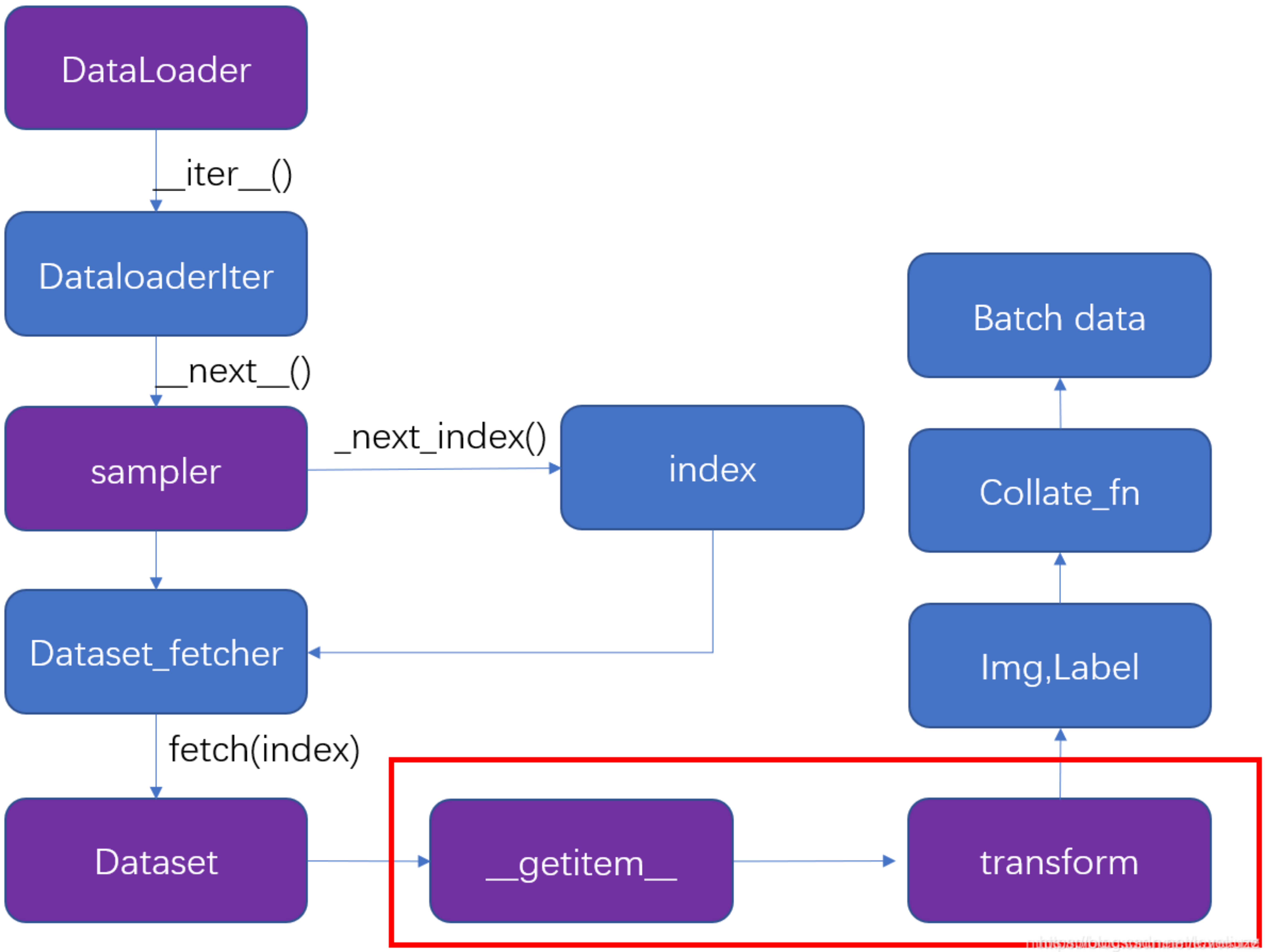


```
for i, data in enumerate(dataloader):
```

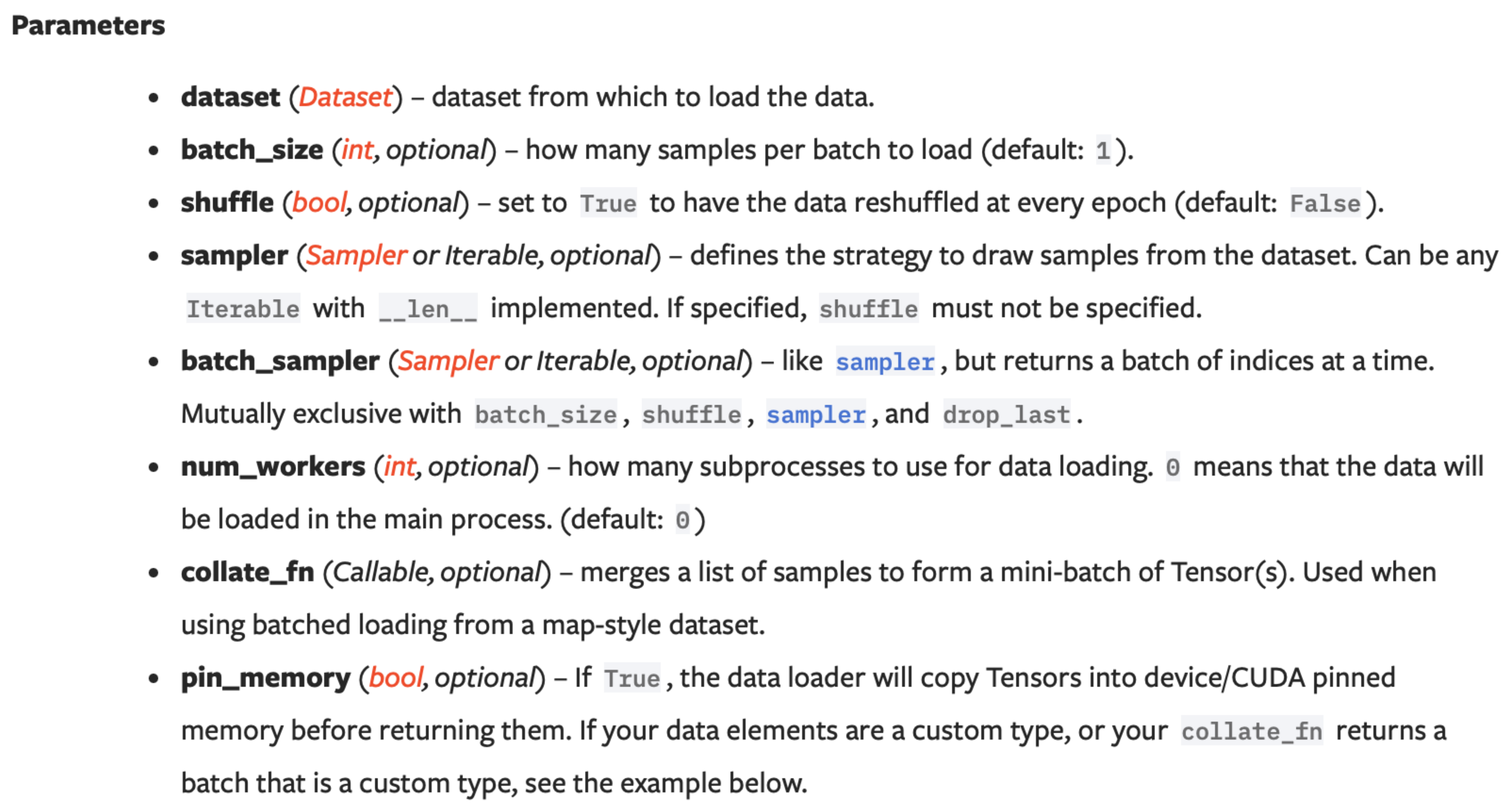
Dataloader in PyTorch

- The **DataLoader** in PyTorch is a tool used to handle input data for models. It combines a **dataset** and a **sampler**, providing an iterable object that supports both single-threaded and multi-threaded (`*num_workers*`) data loading.
- Ways to Access DataLoader:
  - Using `*iter(dataloader)*` and `*next()*`
    - First, wrap `*dataloader*` with `*iter()*`, which returns an iterator.
    - Then, `*next()*` can be used to retrieve batches:

```
data_iter = iter(dataloader)
batch = next(data_iter)
```


DataLoader
__iter__()
DataloaderIter
__next__()
sampler
_next_index()
index
Dataset_fetcher
fetch(index)
Dataset
__getitem__
transform
Img,Label
Collate_fn
Batch data

Parameters in Dataloader

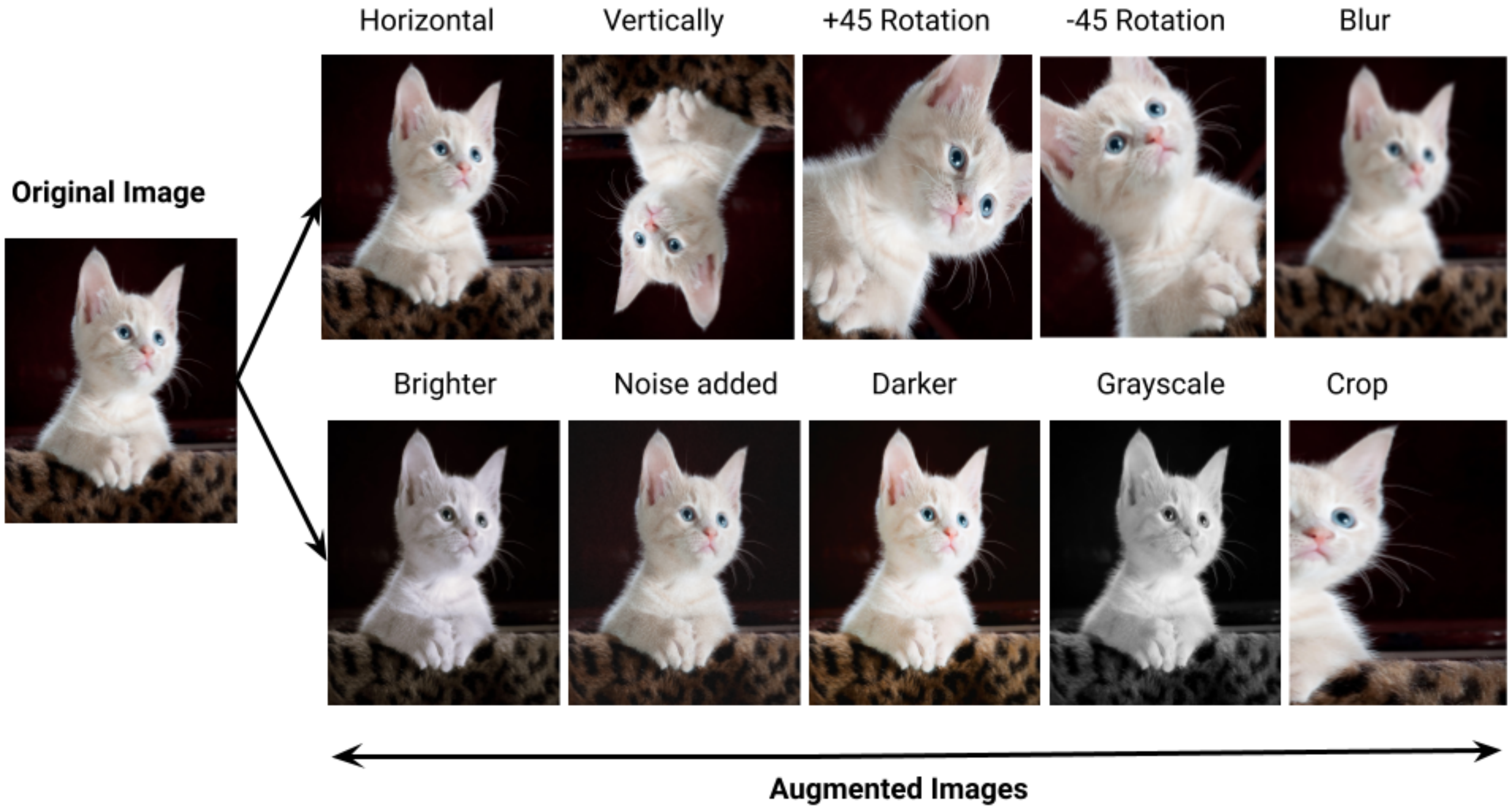


Parameters

- **dataset** (*Dataset*) – dataset from which to load the data.
- **batch_size** (*int*, *optional*) – how many samples per batch to load (default: `1`).
- **shuffle** (*bool*, *optional*) – set to `True` to have the data reshuffled at every epoch (default: `False`).
- **sampler** (*Sampler or Iterable*, *optional*) – defines the strategy to draw samples from the dataset. Can be any `Iterable` with `__len__` implemented. If specified, `shuffle` must not be specified.
- **batch_sampler** (*Sampler or Iterable*, *optional*) – like `sampler`, but returns a batch of indices at a time. Mutually exclusive with `batch_size`, `shuffle`, `sampler`, and `drop_last`.
- **num_workers** (*int*, *optional*) – how many subprocesses to use for data loading. `0` means that the data will be loaded in the main process. (default: `0`)
- **collate_fn** (*Callable*, *optional*) – merges a list of samples to form a mini-batch of Tensor(s). Used when using batched loading from a map-style dataset.
- **pin_memory** (*bool*, *optional*) – If `True`, the data loader will copy Tensors into device/CUDA pinned memory before returning them. If your data elements are a custom type, or your `collate_fn` returns a batch that is a custom type, see the example below.

Data Augmentation

- Flip
- Rotation
- Blur
- Crop
- Translation
- Noise
- …



Code

Code

- ./materials/Week2_Introduction/Week2_Introduction.ipynb
- ./ResNet/Data/cifar10.py
- ./ResNet/train.py: line190-197
- ./ResNet/train_utils.py: line22-25