Deep Learning Frameworks: PyTorch

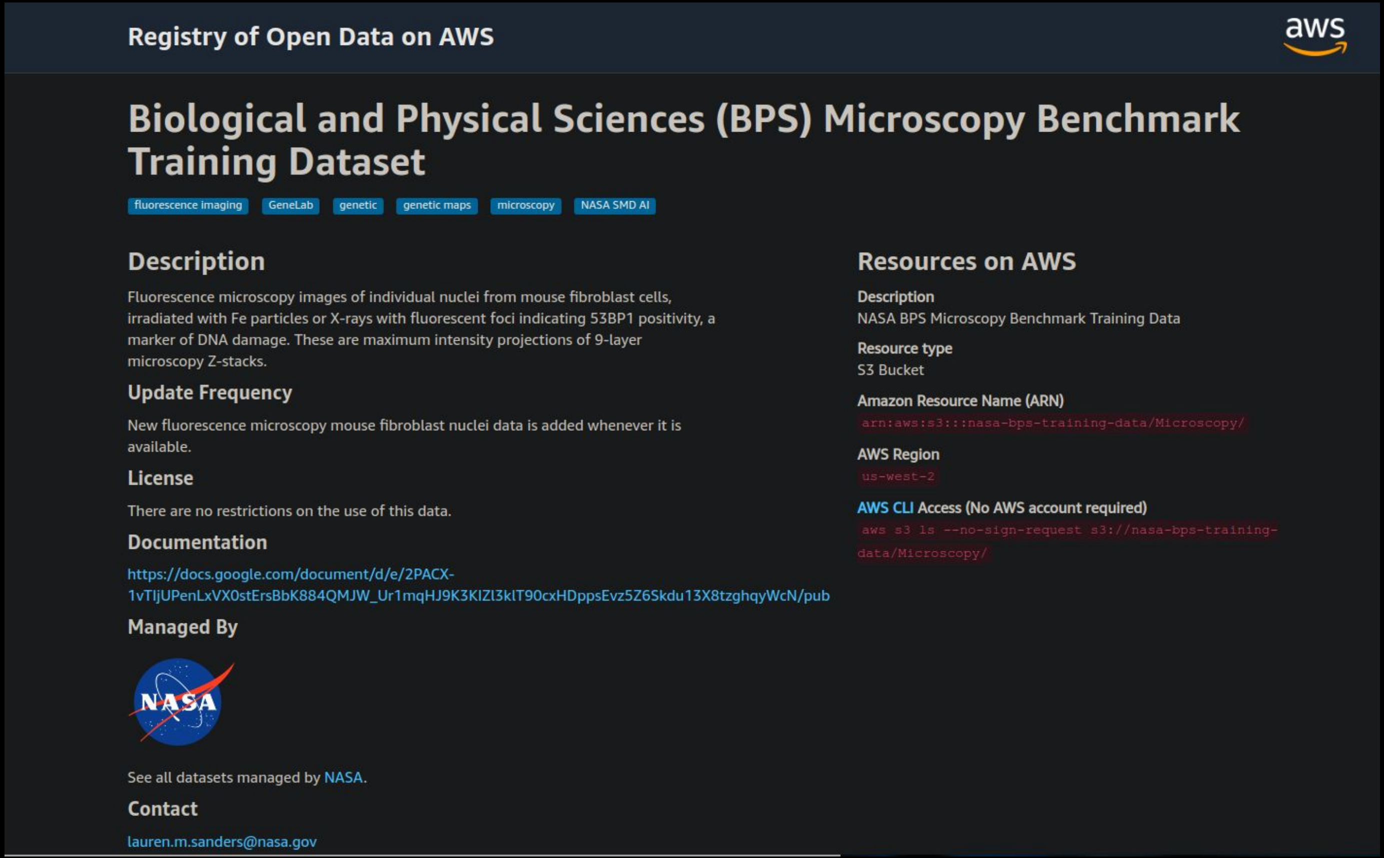
Nadia Ahmed

Overview

In this lecture we will cover:

- The BPS Mouse Data Format
- Deep learning frameworks
  - PyTorch
  - Tensorflow
  - Jax
- Custom datasets
- Augmenting datasets

The BPS Mouse Data



Registry of Open Data on AWS

Biological and Physical Sciences (BPS) Microscopy Benchmark Training Dataset

fluorescence imaging · GeneLab · genetic · genetic maps · microscopy · NASA SMD AI

Description

Fluorescence microscopy images of individual nuclei from mouse fibroblast cells, irradiated with Fe particles or X-rays with fluorescent foci indicating 53BP1 positivity, a marker of DNA damage. These are maximum intensity projections of 9-layer microscopy Z-stacks.

Update Frequency

New fluorescence microscopy mouse fibroblast nuclei data is added whenever it is available.

License

There are no restrictions on the use of this data.

Documentation

https://docs.google.com/document/d/e/2PACX-1vTIjUPenLxVX0stErsBbK884QMJW_Ur1mqHJ9K3KIZl3klT90cxHDppsEvz5Z6Skdu13X8tzghqyWcN/pub

Managed By

See all datasets managed by NASA.

Contact

lauren.m.sanders@nasa.gov

Resources on AWS

Description
NASA BPS Microscopy Benchmark Training Data

Resource type
S3 Bucket

Amazon Resource Name (ARN)
`arn:aws:s3:::nasa-bps-training-data/Microscopy/`

AWS Region
`us-west-2`

AWS CLI Access (No AWS account required)
```
aws s3 ls --no-sign-request s3://nasa-bps-training-data/Microscopy/
```

The BPS Mouse Data

- meta.csv format:
  - filename,dose_Gy,particle_type,hr_post_exposure
  - P242_73665006707-A6_003_013_proj.tif,0.82,Fe,4
  - P242_73665006707-A6_008_034_proj.tif,0.82,Fe,4
  - P242_73665006707-A6_009_007_proj.tif,0.82,Fe,4
- Microscopy and Medical Imaging
  - Files are usually in .FITC format
  - Our files have been compressed to **TIFF**
  - **uint16**: each pixel is represented by 16 bits or 2 bytes
    - allows for wider pixel range for fine details
    - requires scaling following download to float32 and values between 0 and 1 for model
  - Microscopy images have been cropped to different sizes
    - Resizing images will be necessary to utilize batch feature of deep learning frameworks

Deep Learning Frameworks

PyTorch

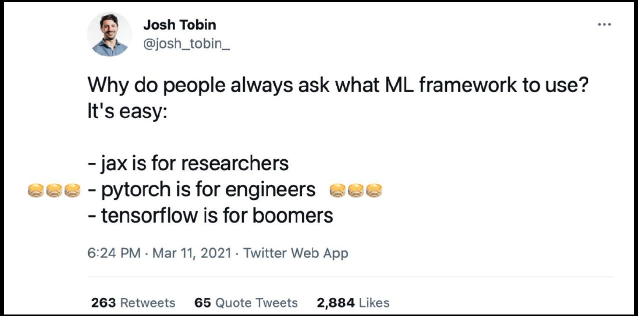
mxnet

PaddlePaddle

- Provide a high-level interface to build and train neural networks
- Optimized to run computations on GPUs or TPUs which can speed up training
- Faster training = faster experimentation
  - model architecture
  - hyperparameters
- Reproducibility

In this class we will use:

PyTorch


Josh Tobin
@josh_tobin_

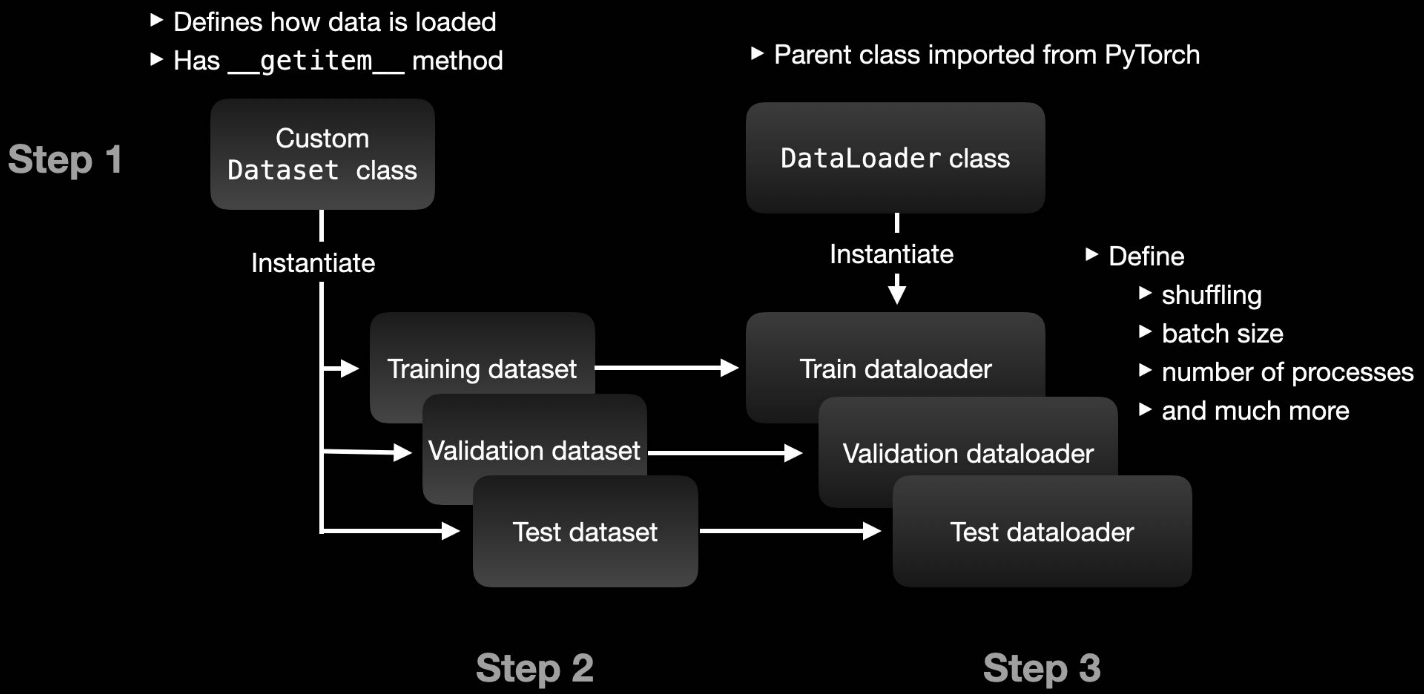
Why do people always ask what ML framework to use? It's easy:

- jax is for researchers
🥞🥞🥞 - pytorch is for engineers 🥞🥞🥞
- tensorflow is for boomers

6:24 PM · Mar 11, 2021 · Twitter Web App

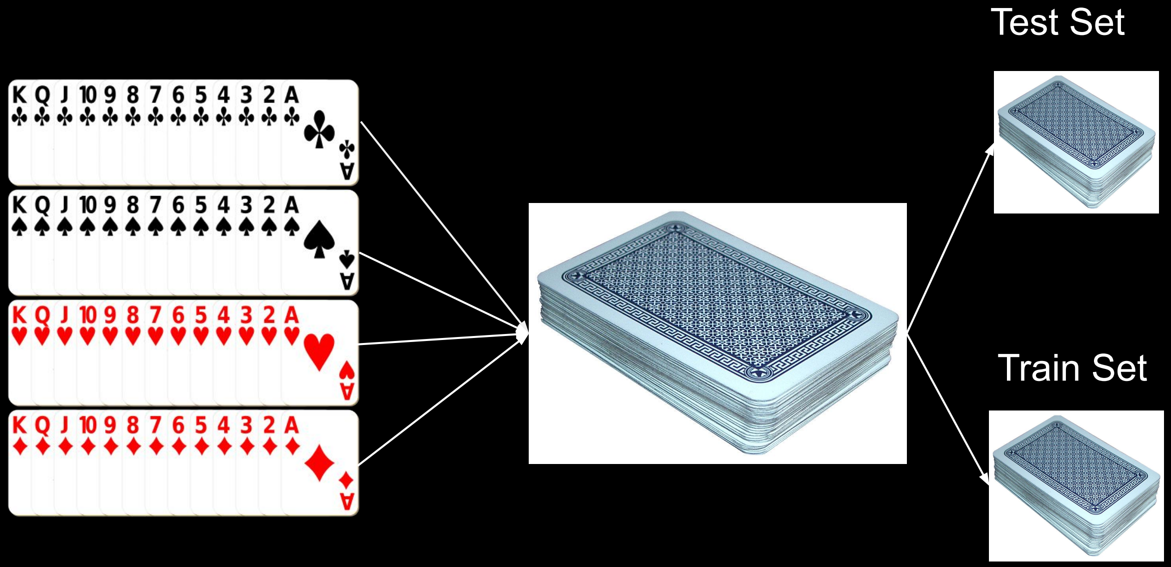
263 Retweets 65 Quote Tweets 2,884 Likes

Making the Data ML-Ready in the Context of our DL Framework



*source: S. Raschka

We may think of our data as cards…



The Data

The Prepared Decks: DataSet

The Dealer: DataLoader

- shuffles
- cuts every "worker" a hand
- serves the data to the model

Identify the PyTorch Domain Library To Use:

PyTorch offers domain libraries specific to your project's needs and the type of data you are interested in learning from:

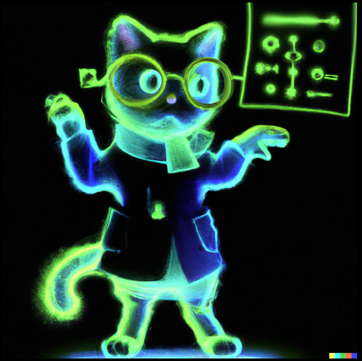
- torchAudio - audio data
- TorchData - new: data pipe functionality
- TorchRec - recommender system
- torchtext - text data
- torchvision - image data
- PyTorch on XLA Devices - specific accelerated linear algebra compiler optimized for specific hardware

Custom Datasets: Why Do We Need to Format the Data for Use with Deep Learning Frameworks?

Deep learning frameworks require specific data formats for a few reasons:

- Efficiency: Deep learning models often require a large amount of data to train, so it's important that the data is stored in an efficient way that allows for fast processing. By storing data in a format that can be easily loaded into a GPU's memory, deep learning frameworks can process data faster and more efficiently.
- Standardization: Different deep learning frameworks have different requirements for data format, so by standardizing on a specific format, data can be easily shared between different frameworks and researchers.
- Preprocessing: Deep learning models often require input data to be preprocessed in specific ways. By having a standard format for data, it's easier to apply preprocessing steps such as normalization, data augmentation, and resizing.

In the case of image data, deep learning frameworks often require images to be stored in a specific format such as a NumPy array or a tensor, with specific dimensions and channel order. This allows the frameworks to efficiently load the data into a GPU's memory and process it using operations such as convolution and pooling. Additionally, by standardizing on a specific image format, it's easier to apply common preprocessing techniques such as normalization and data augmentations



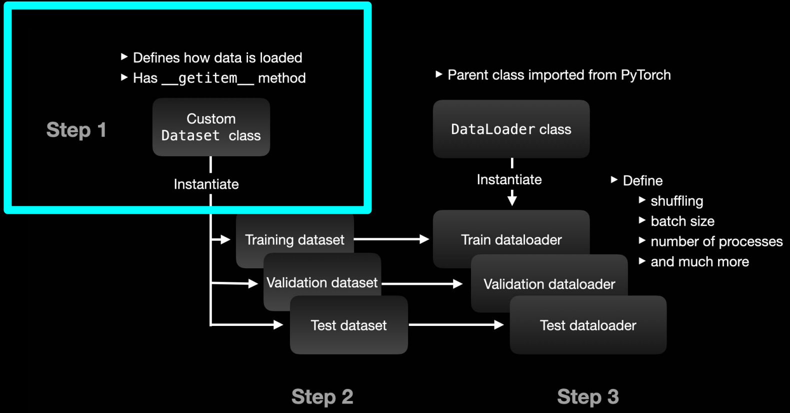
*source: Le Chat GPT and DALL-E

Why Custom Datasets?

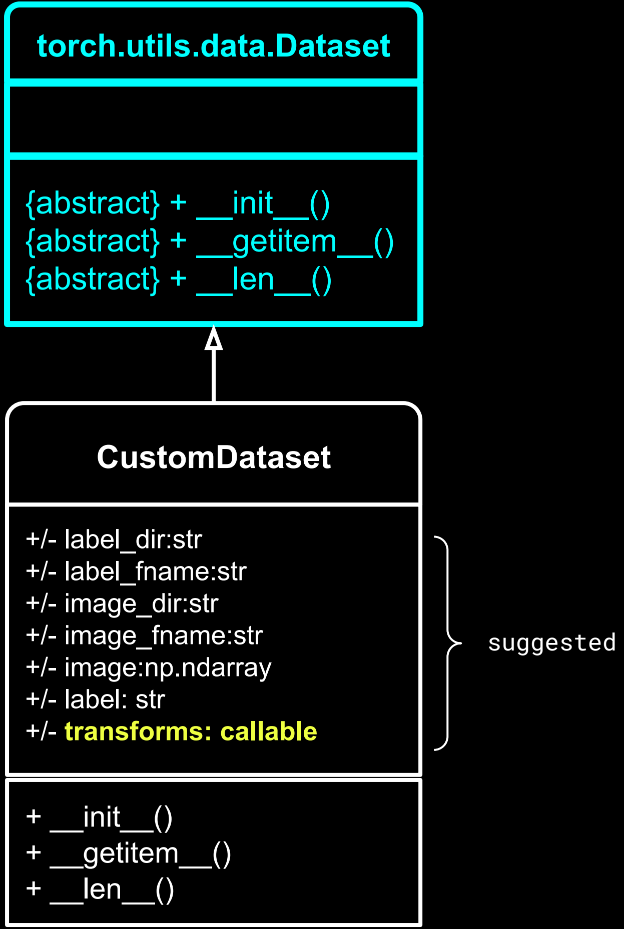
TLDR Answer: We would like to efficiently load large amounts of data and distribute operations to take advantage of GPU/TPU acceleration.

The PyTorch `torch.utils.data.DataLoader` class

- Customizes data loading order
- Performs automatic batching
- Can configure single and multi-process data loading
- automatic memory pinning
- The most important argument of the DataLoader constructor is the **dataset object**
  - Data Format:
    - map ←
    - iterable



The PyTorch torch.utils.data.Dataset class



Map-style Dataset Edition

We create a derived dataset class from the abstract parent class and implement the following methods:

- **__init__():**
  - the constructor and necessary data attributes associated with the data and label (or data alone if using unsupervised techniques)
  - advice: select data attributes to make it easier to find the data, open it, save it as an attribute in constructor
- **__getitem__():** allows for accessing data via an idx and bracket operator
  - dataset[idx] will give you the idx-th image and its label
  - Return a sample as a dictionary or return two items as a tuple (eg. {'image':img, 'label':label} or (img, label))
- **__len__():** allows you to get the length of the dataset

** Note the attribute: **transforms**

…more on this next slide

Why Transforms?

TLDR Answer: We would like to give the model more examples of our data to increase its robustness to variations in the data.

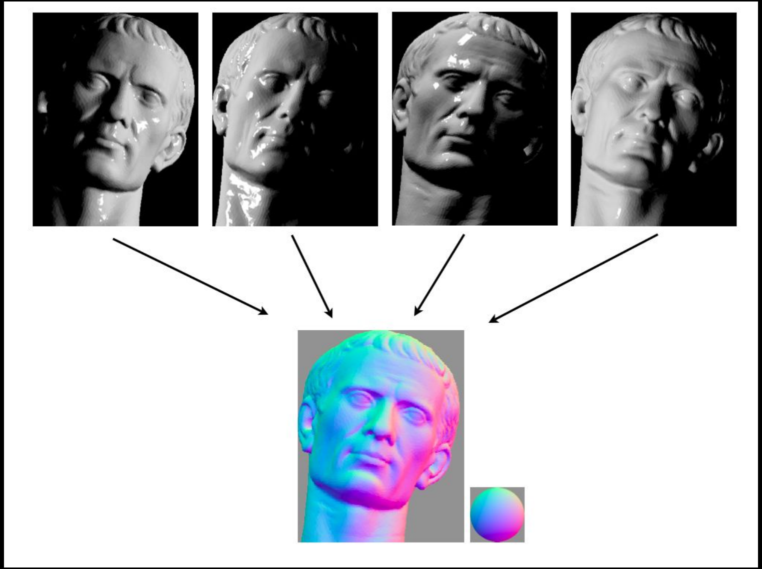
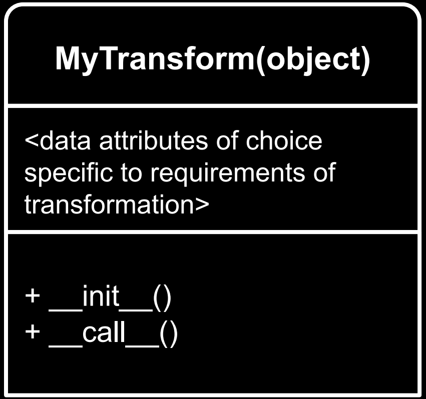


*image source: Wikipedia

The PyTorch transforms


MyTransform(object)
<data attributes of choice specific to requirements of transformation>
+ __init__()
+ __call__()

There are 2 ways to utilize transforms:

- built-in torchvision transforms
- custom transforms

Since this class shows you how to do AI projects from scratch we will do the latter…

To adhere to PyTorch best practice, we specify our image augmentations with **transforms** implemented as a **callable** class so that parameters of the transform will not be passed every time a new transformation is required.

This is because we would like to take advantage of the **torchvision.transforms.Compose** which will allow us to string our transformations in a sequence using a list of Transform objects:

- `transforms.Compose([Transf_1(), Transf_2(), …])`

Transforms and BPS Considerations

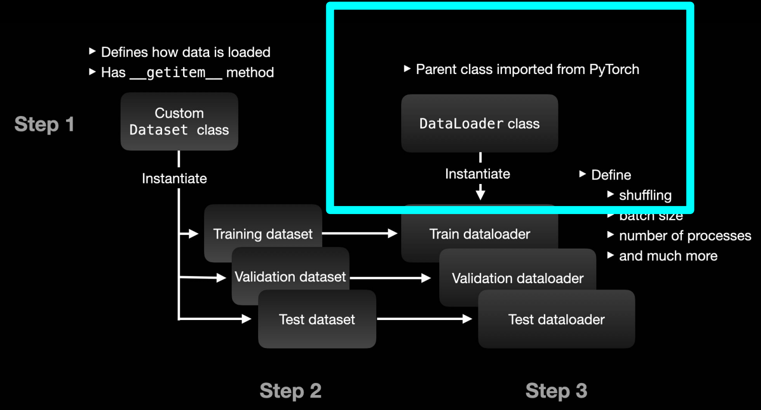
When composing transformations together it is important to consider whether a transformation is invariant.

It is also important to understand the order of transformations.

BPS Mouse Dataset Considerations and Recommendations:

- We will keep our data in the form of numpy arrays and utilize a transform called ToTensor to do the **final** transformation to PyTorch's desired tensor format.
    - numpy arrays for images: (height, width, channels)
    - tensor for images: (batch_size, channel, height, width)
- BPS images are uint16, the pixels have large dynamic range. The first transformation to perform should normalize the pixel into the float32 range with values from 0-1.
- Neural networks expect images of fixed size, because the BPS data contains samples of different sizes, the Resize transformation must be performed to scale the images such that they are the same dimensions

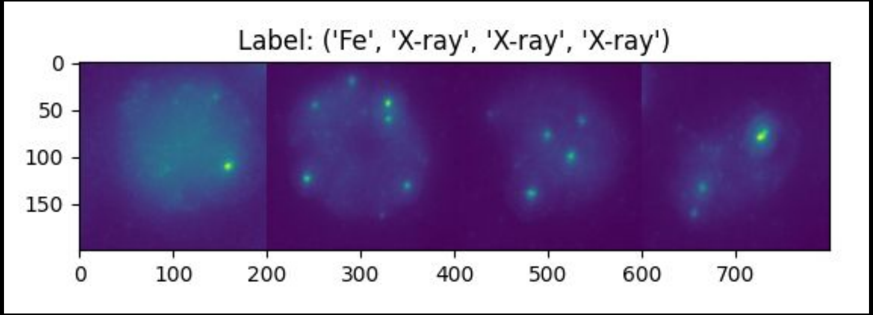
Iterating Through the Dataset: The PyTorch DataLoader



```
dataloader = DataLoader(transformed_dataset,
batch_size=4, shuffle=True, num_workers=0)
```

There are two ways to sample the data:

- Iterating with a loop manually
- Utilizing the **DataLoader**
  - batch data
  - shuffle data
  - load in parallel using multiprocessing



Now We are ML Ready!