Benzina Documentation

Olexa Bilaniuk

Oct 07, 2020

Contents

CHAPTER 1

Table of Contents

1.1 Examples

1.1.1 ImageNet loading in PyTorch

As long as your dataset is converted into Benzina's data format, you can load it to train a PyTorch model in a few lines of code. Here is an example demonstrating how this can be done with an ImageNet dataset. It is based on the ImageNet example from PyTorch

```
import torch
import benzina.torch as bz
import benzina.torch.operations as ops

seed = 1234
torch.manual_seed(seed)

# Dataset
train_dataset = bz.dataset.ImageNet("path/to/dataset", split="train")
val_dataset = bz.dataset.ImageNet("path/to/dataset", split="val")

# Dataloaders
bias = ops.ConstantBiasTransform(bias=(0.485 * 255, 0.456 * 255, 0.406 * 255))
std = ops.ConstantNormTransform(norm=(0.229 * 255, 0.224 * 255, 0.225 * 255))

train_loader = bz.DataLoader(
    train_dataset,
    shape=(224, 224),
    batch_size=256,
    shuffle=True,
    seed=seed,
    bias_transform=bias,
    norm_transform=std,
    warp_transform=ops.SimilarityTransform(scale=(0.08, 1.0),
```

(continues on next page)

(continued from previous page)

```
                                                ratio=(3./4., 4./3.),
                                                flip_h=0.5,
                                                random_crop=True))
val_loader = bz.DataLoader(
    val_dataset,
    shape=(224, 224),
    batch_size=256,
    shuffle=False,
    seed=seed,
    bias_transform=bias,
    norm_transform=std,
    warp_transform=ops.CenterResizedCrop(224/256)))

for epoch in range(1, 10):
    # train for one epoch
    train(train_dataloader, ...)

    # evaluate on validation set
    accuracy = validate(valid_dataloader, ...)
```

In the example above, two `benzina.torch.dataset.ImageNet` are first created with the location of the dataset and the desired split specified.

Note: To be able to quickly load your dataset with the hardware decoder of a GPU, Benzina needs the data to be converted in its own format embedding H.265 images.

```
train_dataset = bz.dataset.ImageNet("path/to/dataset", split="train")
val_dataset = bz.dataset.ImageNet("path/to/dataset", split="val")
```

Then the transformations to apply to the dataset are defined. It is usually a good idea to normalize the data based on its statistical bias and standard deviation which can be done with Benzina by using its `benzina.torch.operations.ConstantBiasTransform` and `benzina.torch.operations.ConstantNormTransform` respectively.

Note:

- `benzina.torch.operations.ConstantBiasTransform` will substract the bias from the images' RGB channels
- `benzina.torch.operations.ConstantNormTransform` will multiply the norm with the images' RGB channels

```
bias = ops.ConstantBiasTransform(bias=(123.675, 116.28 , 103.53))
std = ops.ConstantNormTransform(norm=(58.395, 57.12 , 57.375))
```

The dataloaders are now ready to be instantiated. In this example, the dataset's images are all of size 512 x 512 by the dataset specifications. A random crop resized to 224 x 224 and a random horizontal flip will be applied to the images prior feeding them to the model. In Benzina, this is done by defining the size of the output tensor with the dataloader's `shape` argument and using Benzina's similarity transform.

In the case of the validation transform, an alias to a specific similarity transform, which applies a center crop of edges scale 224 / 256, resize the cropped section to have its smaller edge matched to 224 then center a crop of 224 x 224. Another maybe more intuitive way to describe this transformation is to see it as a resize to have the smaller edge matched to 256 then center a crop of 224 x 224.

Note: It's useful to know that `benzina.torch.operations.SimilarityTransform` will automatically center the output frame on the center of the input image. This makes a vanilla `benzina.torch.operations.SimilarityTransform` equivalent a center crop of size of the output.

```
train_loader = bz.DataLoader(
    train_dataset,
    shape=(224, 224),
    batch_size=256,
    shuffle=True,
    seed=seed,
    bias_transform=bias,
    norm_transform=std,
    warp_transform=ops.SimilarityTransform(scale=(0.08, 1.0),
                                           ratio=(3./4., 4./3.),
                                           flip_h=0.5,
                                           random_crop=True))
val_loader = bz.DataLoader(
    val_dataset,
    shape=(224, 224),
    batch_size=256,
    shuffle=False,
    seed=seed,
    bias_transform=bias,
    norm_transform=std,
    warp_transform=ops.CenterResizedCrop(224/256))
```

As demonstrated in the full example loading ImageNet to feed a PyTorch model, code change between a pure PyTorch implementation and an implementation using Benzina holds in only a few lines.

```
$ diff -ty --suppress-common-lines examples/python/imagenet/main.py examples/python/
↪imagenet/imagenet_pytorch.py
```

```
                                                                  >  import torchvision.
↪transforms as transforms
                                                                  >  import torchvision.
↪datasets as datasets
### Benzina       ###                                             <
import benzina.torch as bz                                        <
import benzina.torch.operations as ops                            <
### Benzina - end ###                                             <
                                                                  <
                                                                  >  parser.add_
↪argument('-j', '--workers', default=4, type=int, met
                                                                  >                    ␣
↪ help='number of data loading workers (defau
    ### Benzina       ###                                       |      normalize =␣
↪transforms.Normalize(mean=[0.485, 0.456, 0.406]
    train_dataset = bz.dataset.ImageNet(args.data, split="train" |                    ␣
↪                  std=[0.229, 0.224, 0.225])
                                                                  <
    bias = ops.ConstantBiasTransform(bias=(0.485 * 255, 0.456 * <
    std = ops.ConstantNormTransform(norm=(0.229 * 255, 0.224 *  <
    train_loader = bz.DataLoader(                                 |      train_dataset␣
↪= datasets.ImageNet(
        train_dataset, shape=(224, 224), batch_size=args.batch_ |          args.data,
↪"train",
```

(continues on next page)

(continued from previous page)

```
        shuffle=True, seed=args.seed,                                 |           transforms.
↪Compose([
        bias_transform=bias,                                          |               ␣
↪transforms.RandomResizedCrop(224),
        norm_transform=std,                                           |               ␣
↪transforms.RandomHorizontalFlip(),
        warp_transform=ops.SimilarityTransform(                       |               ␣
↪transforms.ToTensor(),
            scale=(0.08, 1.0),                                        |               ␣
↪normalize,
            ratio=(3./4., 4./3.),                                     |           ]))
            flip_h=0.5,                                               |
            random_crop=True))                                        |       train_loader =␣
↪torch.utils.data.DataLoader(
                                                                      |           train_
↪dataset, batch_size=args.batch_size, shuffle=True
    val_loader = bz.DataLoader(                                       |           num_
↪workers=args.workers, pin_memory=True)
        bz.dataset.ImageNet(args.data, split="val"), shape=(224       |
        batch_size=args.batch_size, shuffle=args.batch_size, se       |       val_loader =␣
↪torch.utils.data.DataLoader(
        bias_transform=bias,                                          |           datasets.
↪ImageNet(args.data, "val", transforms.Compose(
        norm_transform=std,                                           |               ␣
↪transforms.Resize(256),
        warp_transform=ops.CenterResizedCrop(224/256))                |               ␣
↪transforms.CenterCrop(224),
    ### Benzina - end ###                                             |               ␣
↪transforms.ToTensor(),
                                                                      >               ␣
↪normalize,
                                                                      >           ])),
                                                                      >           batch_
↪size=args.batch_size, shuffle=False,
                                                                      >           num_
↪workers=args.workers, pin_memory=True)
```

1.2 Datasets List

1.2.1 General Description of a Dataset

Dataset Composition

A Benzina dataset is, in essence, an indexing over a concatenation of inputs, targets and possibly filenames with indexing

Dataset Structure

A Benzina dataset is structured using the mp4 format

ftyp Defines the compatibilities of the mp4 container

mdat Concatenation in 2-3 blocks of the inputs, targets and possibly filenames

moov Contains the metadata needed to load and present the raw data of *mdat*

mvhd Defines the *timescale* and the *duration* of the container

timescale How many units elapse in 1 second

duration Duration of the container in *timescale* units

next_track_id The id of the next track that could be appended to *moov*

trak

- ***Benzina input samples track*: This is the first track and it references** all the input samples
- *Benzina target track*
- *Benzina filename track*: This track is optional
- ***Video track*: This track is optional. If present it should be** positioned last

Each track can have a *train*, *validation* and *test* variants to reference the sets

tkhd Defines the resolution of the video and if the track should be displayed by an mp4 player

flags Defines if the track should be displayed

width Width of the video

height Height of the video

mdia Contains definitions related to the media type of the data

mdhd Redefines the *timescale* and the *duration* for the track

timescale How many units elapse in 1 second

duration Duration of the track in *timescale* units

hdlr Defines the media type of the track

handler_type Defines the type of handler that should be used to decode the data referenced by the track

name Human readable name for the track type (used for debugging)

minf Defines the characteristics of the media in the track

stbl Defines the data indexing of the media samples in the track along with coding information, if needed, to decode them

stsd Provides the information needed to decode the media samples

stts Defines the mapping from decoding time to sample number

sample_count The number of samples in the track

sample_delta The interval in `timescale` units for which a new sample should be decoded

stsz Defines the size of each samples

sample_count Number of samples in the track

entry_size Size of the sample. This field is repeated for each sample

stsc Defines the chunks splitting the data

stco Defines the chunks offset

entry_count Number of chunks

chunk_offset The chunk offset. This field is repeated for each chunk

Dataset's Input Sample Structure

A Benzina dataset's input sample can also be structured using the mp4 format. It is roughly the same as the dataset's structure with the differences that *mdat* will contains the raw concatenation of a single input, its target, possibly filename and possibly a 512 x 512 thumbnails stream.

1.2.2 ImageNet 2012

ImageNet 2012 classification dataset. It contains two size of the images along with their classification target and filename:

- Resized high resolution images each with a smaller edge of at most 512 while preserving the aspect ratio. This set is accessed by referencing the *bzna_input* track of the input samples.
- Resized images each with a longer edge of at most 512 while preserving the aspect ratio. This set is accessed by referencing the *bzna_thumb* track of the input samples.

The dataset is represented by `ImageNet` which simplifies the iteration of the data as a classification dataset.

Warning: 81 images are currently missing from the dataset and 111 had to be first transcoded to PNG prior to the final H.265 format. More details can be found in the dataset's README.

Warning: High resolution images stored in the the *bzna_input* track of the input samples are currently not available through the `DataLoader`. Their widely varying sizes prevent them from being decoded using a single hardware decoder configuration. The selected solution is to represent the images in the HEIF format which will be completed in future development.

Dataset Composition

The dataset is composed of a train set, followed by a validation set then a test set for a total of 1 431 167 entries. Targets and filenames are provided for each sets:

- **Train set**
  Entries 1 to 1281167 (1 281 167 entries)
- **Validation set**
  Entries 1281168 to 1331167 (50 000 entries)
- **Test set**
  Entries 1331168 to 1431167 (100 000 entries)

Dataset Structure

ilsvrc2012.bzna

ftyp Defines the compatibilities of the mp4 container

major_brand isom

minor_version 0

compatible_brands bzna, isom

mdat Raw concatenation in 3 blocks of the images, targets and filenames

- Concatenation of .mp4 files containing a single image, a thumbnail of a maximum size of 512 x 512 if the image does not already fit this resolution, the image's original filename and the target associated with the image
- Concatenation of images' targets as little-endian int64
- Concatenation of images' original filename

moov Contains the metadata needed to load and present the raw data of *mdat*

mvhd Defines the *timescale* and the *duration* of the container

timescale 20

duration 20 * 1 431 167

next_track_id The id of the next track that could be appended to *moov*

trak *Benzina input samples track*

This track references all the images of the dataset

tkhd Defines the resolution of the video and if the track should be displayed by an mp4 player

flags 000000 – This value informs that the track is not for display purpose

width 0.0 – This value reflects the variance in size of the frames

height 0.0 – This value reflects the variance in size of the frames

mdia Contains definitions related to the media type of the data

mdhd Redefines the *timescale* and the *duration* for the track

timescale 20

duration 20 * 1 431 167

hdlr Defines the media type of the track

handler_type `meta`

name `bzna_input`

minf Defines the characteristics of the media in the track

nmhd No specific media header is identified for the track

stbl Defines the data indexing of the media samples in the track along with coding information, if needed, to decode them

stsd Provides the information needed to decode the media samples

mett Defines the metadata as being text based

mime_format
`application/octet-stream`

stts Defines the mapping from decoding time to sample number

sample_count
1
431
167

sample_delta
20

stsz Defines the size of each samples

sample_count
1
431
167

entry_size
Size of the sample. This field is repeated for each sample

stsc Defines the chunks splitting the data

first_chunk
1

samples_per_chunk
1

sample_description_index
1

This definition means to consider that all samples are contained in their own chunk

stco Defines the chunks offset

entry_count
1
431
167

chunk_offset
The chunk offset. This field is

repeated for each chunk, i.e. for each sample

trak *Benzina target track*

This track is roughly the same as the *Benzina input track* with the following differences

mdia Contains definitions related to the media type of the data

hdlr Defines the media type of the track

handler_type `meta`

name `bzna_target`

trak *Benzina filename track*

This track is roughly the same as the *Benzina input track* with the following differences

tkhd Defines the resolution of the video and if the track should be displayed by an mp4 player

flags 000003 – This value informs that the track is enabled and can be used in the presentation

width 0.0 – This value informs that no width has be predefined for this track

height 0.0 – This value informs that no height has be predefined for this track

mdia Contains definitions related to the media type of the data

hdlr Defines the media type of the track

handler_type `meta`

name `bzna_fname`

minf Defines the characteristics of the media in the track

stbl Defines the data indexing of the media samples in the track along with coding information, if needed, to decode them

stsd Provides the

information needed to decode the media samples

mett Defines the metadata as being text based

mime_format `text/plain`

trak *Video track*

This track allows to play the thumbnails of the dataset's frames

tkhd Defines the resolution of the video and if the track should be displayed by an mp4 player

flags 000003 – This value informs that the track is enabled and can be used in the presentation

width 512.0

height 512.0

mdia Contains definitions related to the media type of the data

mdhd Redefines the *timescale* and the *duration* for the track

timescale 20

duration 1 431 167

hdlr Defines the media type of the track

handler_type `vide`

name `VideoHandler`

minf Defines the characteristics of the media in the track

vmhd Video media header is identified for the track

stbl Defines the data indexing of the media samples in the track along with coding information, if needed, to decode them

stsd
Provides the information needed to decode the media samples

avc1
Defines the AVC coding information

width
512

height
512

horizresolution
72

horizresolution
72

stts
Defines the mapping from decoding time to sam-

ple
num-
ber

sample_count
1
431
167

sample_delta
1

stsz
Defines
the
size
of
each
sam-
ples

sample_count
1
431
167

entry_size
Size
of
the
sam-
ple.
This
field
is
re-
peated
for
each
sam-
ple

stsc
Defines
the
chunks
split-
ting
the
data

first_chunk
1

samples_per_chunk
1

sample_description_index

1

This definition means to consider that all samples are contained in their own chunk

stco

Defines the chunks offset

entry_count

1

431

167

chunk_offset

The chunk offset. This field is repeated for each chunk, i.e. for each sample

Dataset's Input Samples Structure

A Benzina ImageNet dataset's input sample is structured using the mp4 format.

ftyp Defines the compatibilities of the mp4 container

major_brand isom

minor_version 0

compatible_brands bzna, isom

mdat Raw concatenation of the image, thumbnail, target and filename:

- A single image in H.265 format. The image is put in a frame with a size of a product of 512 in the 2 dimensions. The padding to make the image fit is filled with a smear of the image's borders
- A thumbnail in H.265 format. The image is put in a frame of size 512 x 512. The image is first resized to have its longest side be of 512. The padding to make the thumbnail fit the frame is filled with a smear of the image's borders. There will be no explicit thumbnail if the image already fit the thumbnail's frame
- The image's target in a little-endian int64
- The image's original filename

moov Contains the metadata needed to load and present the raw data of *mdat*

mvhd Defines the *timescale* and the *duration* of the container

timescale 20

duration 20

next_track_id The id of the next track that could be appended to *moov*

trak *Benzina input track*

This track references an image

tkhd Defines the resolution of the video and if the track should be displayed by an mp4 player

flags 000000 – This value informs that the track is not for display purpose

width Width of the image without padding

height Height of the image without padding

mdia Contains definitions related to the media type of the data

mdhd Redefines the *timescale* and the *duration* for the track

timescale 20

duration 20

hdlr Defines the media type of the track

handler_type `vide`

name `bzna_input`

minf Defines the characteristics of the media in the track

vmhd Video media header is identified for the track

stbl Defines the data indexing of the media samples in the track along with coding information, if needed, to decode them

stsd Provides the information needed to decode the media samples

avc1 Defines the AVC coding information

width Width of the image's frame. This is a product of 512

height Height of the image's frame. This

is a product of 512

horizresolution
72

horizresolution
72

clap
Defines the clean aperture of the image to remove the padding

clean_aperture_width_n
Width of the image without padding

clean_aperture_width_d
1

clean_aperture_height_n
Height of the image without padding

clean_aperture_height_d
1

horiz_off_n
The neg-

a-
tive
value
of
the
width's
padding

horiz_off_d
2

vert_off_n
The
neg-
a-
tive
value
of
the
height's
padding

vert_off_d
2

stts
Defines
the
map-
ping
from
de-
cod-
ing
time
to
sam-
ple
num-
ber

sample_count
1

sample_delta
20

stsz
Defines
the
size
of
each
sam-
ples

sample_count
1

entry_size
Size of the input

stsc
Defines the chunks splitting the data

first_chunk
1

samples_per_chunk
1

sample_description_index
1

stco
Defines the chunks offset

entry_count
1

chunk_offset
The chunk offset

trak *Benzina thumbnail track*

This track references an image's thumbnail. If the image already fits a thumbnail's frame, then this track will reference the same data as in the *Benzina input track*. In any case, it is roughly the same as the *Benzina input track* with the following differences

tkhd Defines the resolution of the video and if the track should be displayed by an mp4 player

flags 000003 – This value informs that the track is enabled and can be used in the presentation

width Width of the thumbnail without padding

height Height of the thumbnail without padding

mdia Contains definitions related to the media type of the data

hdlr Defines the media type of the track

handler_type `vide`

name `bzna_thumb`

trak *Benzina target track*

tkhd Defines the resolution of the video and if the track should be displayed by an mp4 player

flags 000000 – This value informs that the track is not for display purpose

width 0.0 – This value informs that the width has not been predefined for this track

height 0.0 – This value informs that no height has not been predefined for this track

mdia Contains definitions related to the media type of the data

mdhd Redefines the *timescale* and the *duration* for the track

timescale 20

duration 20

hdlr Defines the media type of the track

handler_type `meta`

name `bzna_target`

minf Defines the characteristics of the media in the track

nmhd No specific media header is identified for the track

stbl Defines the data indexing of the media samples in the track along with coding information, if needed, to decode them

stsd Provides the information needed to decode the media

samples

mett Defines the metadata as being text based

mime_format `application/octet-stream`

trak *Benzina filename track*

This track is roughly the same as the *Benzina target track* with the following differences

tkhd Defines the resolution of the video and if the track should be displayed by an mp4 player

flags 000003 – This value informs that the track is enabled and can be used in the presentation

width 0.0 – This value informs that no width has be predefined for this track

height 0.0 – This value informs that no height has be predefined for this track

mdia Contains definitions related to the media type of the data

hdlr Defines the media type of the track

handler_type `meta`

name `bzna_fname`

minf Defines the characteristics of the media in the track

stbl Defines the data indexing of the media samples in the track along with coding information, if needed, to decode them

stsd Provides the information

needed to decode the media samples

mett
Defines the metadata as being text based

mime_format
`text/plain`

1.3 Objectives

In much of the work in the field of machine learning and deep learning, a bottleneck exists in the dataloading phase itself. This is becoming increasingly recognised as an issue which needs to be solved.

Benzina aims to become a go-to tool for dataloading large datasets. Other tools exist, such as Dali. Yet Benzina concentrates itself on two aspects :

- Highest level of performance for dataloading using GPU as loading device
- Creation of a generalist storage format as a single file facilitating distribution of datasets and useful in the context of file system limits.

1.3.1 Further feature points

- Generalist DNN framework methods provided to integrate Benzina to PyTorch and TensorFlow
- Command line programs will be created to assist in Benzina - compatible datasets
- API interface to interact with Benzina

1.4 Known limitations and important notes

1.4.1 As of September 2020

- No TensorFlow integration
- Currently only supports ImageNet

- Unknown effect on model accuracy of transcoding from various JPEG formats to H.265
- Current transcoding filters failed on 81 images of the *ImageNet 2012* dataset forcing them to be excluded. More information can be found in the dataset's README.
- Current transcoding filters required 111 images of the *ImageNet 2012* dataset to first be transcoded to PNG prior to the final H.265 format. More information can be found in the dataset's README.
- High resolution images stored in the *bzna_input track of the input samples* are currently not available through the `Dataloader`. Their varying size prevent them from being decoded using a single hardware decoder configuration. The selected solution is to represent the images in the HEIF format which will be completed in future development.
- It is currently not possible to *compose* transformations like you can with `torchvision.transforms.Compose` but `SimilarityTransform` should cover most of the necessary images transformations.

1.5 Roadmap

1.5.1 Summer 2019

- Collaboration phase with researchers
- TensorFlow implementation
- **Normalized format**
  - Specification freeze
  - Dataset creation utils
  - More tests
  - Collaboration with researchers using new format

1.5.2 Autumn 2019

Conference Talk on Benzina

1.6 How to Contribute

This document is heavily based on Contributing to Open Source Projects

1.6.1 Submitting bugs

Due diligence

Before submitting a bug, please do the following:

- Perform **basic troubleshooting** steps:
  - **Make sure you're on the latest version.** If you're not on the most recent version, your problem may have been solved already! Upgrading is always the best first step.

  - **Try older versions.** If you're already *on* the latest release, try rolling back a few minor versions (e.g. if on 1.7, try 1.5 or 1.6) and see if the problem goes away. This will help the devs narrow down when the problem first arose in the commit log.
  - **Try switching up dependency versions.** If the software in question has dependencies (other libraries, etc) try upgrading/downgrading those as well.
- **Search the project's bug/issue tracker** to make sure it's not a known issue.
- If you don't find a pre-existing issue, consider **checking with the mailing list and/or IRC channel** in case the problem is non-bug-related.

What to put in your bug report

Make sure your report gets the attention it deserves: bug reports with missing information may be ignored or punted back to you, delaying a fix. The below constitutes a bare minimum; more info is almost always better:

- **What version of the core programming language interpreter are you using?** For example, are you using Python 3.5? Python 3.6?
- **Which version or versions of the software are you using?** Ideally, you followed the advice above and have ruled out (or verified that the problem exists in) a few different versions.
- **How can the developers recreate the bug on their end?** If possible, include a copy of your code, the command you used to invoke it, and the full output of your run (if applicable.)
  - A common tactic is to pare down your code until a simple (but still bug-causing) "base case" remains. Not only can this help you identify problems which aren't real bugs, but it means the developer can get to fixing the bug faster.

1.6.2 Contributing changes

Licensing of contributed material

Keep in mind as you contribute, that code, docs and other material submitted to open source projects are usually considered licensed under the same terms as the rest of the work.

The details vary from project to project, but from the perspective of this document's authors:

- Anything submitted to a project falls under the licensing terms in the repository's top level `LICENSE` file.
  - For example, if a project's `LICENSE` is BSD-based, contributors should be comfortable with their work potentially being distributed in binary form without the original source code.
- Per-file copyright/license headers are typically extraneous and undesirable. Please don't add your own copyright headers to new files unless the project's license actually requires them!
  - Not least because even a new file created by one individual (who often feels compelled to put their personal copyright notice at the top) will inherently end up contributed to by dozens of others over time, making a per-file header outdated/misleading.

Version control branching

- Always **make a new branch** for your work, no matter how small. This makes it easy for others to take just that one set of changes from your repository, in case you have multiple unrelated changes floating around.
  - A corollary: **don't submit unrelated changes in the same branch/pull request**! The maintainer shouldn't have to reject your awesome bugfix because the feature you put in with it needs more review.

- **Base your new branch off of the appropriate branch** on the main repository:
  - **Bug fixes** should be based on the branch named after the **oldest supported release line** the bug affects.
    - E.g. if a feature was introduced in 1.1, the latest release line is 1.3, and a bug is found in that feature - make your branch based on 1.1. The maintainer will then forward-port it to 1.3 and master.
    - Bug fixes requiring large changes to the code or which have a chance of being otherwise disruptive, may need to base off of **master** instead. This is a judgement call – ask the devs!
  - **New features** should branch off of **the 'master' branch**.
    - Note that depending on how long it takes for the dev team to merge your patch, the copy of `master` you worked off of may get out of date! If you find yourself 'bumping' a pull request that's been sidelined for a while, **make sure you rebase or merge to latest master** to ensure a speedier resolution.

Code formatting

- **Follow the style you see used in the primary repository**! Consistency with the rest of the project always trumps other considerations. It doesn't matter if you have your own style or if the rest of the code breaks with the greater community - just follow along.
- Python projects usually follow the PEP-8 guidelines (though many have minor deviations depending on the lead maintainers' preferences.)

Documentation isn't optional

It's not! Patches without documentation will be returned to sender. By "documentation" we mean:

- **Docstrings** (for Python; or API-doc-friendly comments for other languages) must be created or updated for public API functions/methods/etc. (This step is optional for some bugfixes.)
  - Don't forget to include versionadded/versionchanged ReST directives at the bottom of any new or changed Python docstrings!
    - Use `versionadded` for truly new API members – new methods, functions, classes or modules.
    - Use `versionchanged` when adding/removing new function/method arguments, or whenever behavior changes.
- New features should ideally include updates to **prose documentation**, including useful example code snippets.
- All submissions should have a **changelog entry** crediting the contributor and/or any individuals instrumental in identifying the problem.

Full example

Here's an example workflow for the project `Benzina`, which is currently in hypothetic version 1.0.x. Your username is `yourname` and you're submitting a basic bugfix.

Preparing your Fork

1. Click 'Fork' on Github, creating e.g. `yourname/Benzina`.
2. Clone your project: `git clone git@github.com:yourname/Benzina`.
3. `cd Benzina`

4. Create and activate a virtual environment.
5. Install the development requirements: `pip install -r dev-requirements.txt`.
6. Create a branch: `git checkout -b foo-the-bars 1.0`.

Making your Changes

1. Add changelog entry crediting yourself.
2. Hack, hack, hack.
3. Commit your changes: `git commit -m "Foo the bars"`

Creating Pull Requests

1. Push your commit to get it back up to your fork: `git push origin HEAD`
2. Visit Github, click handy "Pull request" button that it will make upon noticing your new branch.
3. In the description field, write down issue number (if submitting code fixing an existing issue) or describe the issue + your fix (if submitting a wholly new bugfix).
4. Hit 'submit'! And please be patient - the maintainers will get to you when they can.

1.7 API

1.7.1 benzina.torch.dataloader

class benzina.torch.dataloader.**DataLoader**(*dataset*, *shape*, *path=None*, *batch_size=1*, *shuffle=False*, *sampler=None*, *batch_sampler=None*, *collate_fn=<sphinx.ext.autodoc.importer._MockObject object>*, *drop_last=False*, *timeout=0*, *device=None*, *multibuffering=3*, *seed=None*, *bias_transform=None*, *norm_transform=None*, *warp_transform=None*)

Loads images from a `benzina.torch.dataset.Dataset`. Encapsulates a sampler and data processing transformations.

Parameters

- **dataset** (`benzina.torch.dataset.Dataset`) – dataset from which to load the data.
- **shape** (*`int or tuple of ints`*) – set the shape of the samples. Note that this does not imply a resize of the image but merely set the shape of the tensor in which the data will be copied.
- **path** (*`str, optional`*) – path to the archive from which samples will be decoded. If not specified, the dataloader will attempt to get it from `dataset`.
- **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** (*`torch.utils.data.Sampler, optional`*) – defines the strategy to draw samples from the dataset. If specified, `shuffle` must be `False`.

- **batch_sampler** (*torch.utils.data.Sampler, optional*) – like sampler, but returns a batch of indices at a time. Mutually exclusive with `batch_size`, `shuffle`, `sampler`, and `drop_last`.
- **collate_fn** (*callable, optional*) – merges a list of samples to form a mini-batch.
- **drop_last** (*bool, optional*) – set to `True` to drop the last incomplete batch, if the dataset size is not divisible by the batch size. If `False` and the size of dataset is not divisible by the batch size, then the last batch will be smaller. (default: `False`)
- **timeout** (*numeric, optional*) – if positive, the timeout value for collecting a batch. Should always be non-negative. (default: `0`)
- **device** (*torch.device, optional*) – set the device to use. Note that only CUDA devices are supported for the moment.
- **multibuffering** (*int, optional*) – set the size of the multibuffering buffer. (default: `3`)
- **seed** (*int, optional*) – set the seed for the random transformations.
- **bias_transform** (`benzina.torch.operations.BiasTransform` or float, optional) – set the bias transformation. Values to substract a pixel's channels with. Note that this transformation is applied before `norm_transform`.
- **norm_transform** (`benzina.torch.operations.NormTransform` or float or iterable of float, optional) – set the normalization transformation. Values to multiply a pixel's channels with. Note that this transformation is applied after `bias_transform`.
- **warp_transform** (`benzina.torch.operations.WarpTransform` or iterable of float, optional) – set the warp transformation or use as the arguments to initialize a WarpTransform.

1.7.2 benzina.torch.dataset

class benzina.torch.dataset.**Dataset**(*archive: Union[str, benzina.utils.file.Track] = None, track: Union[str, benzina.utils.file.Track] = 'bzna_input'*)

Parameters

- **archive** (str or `Track`) – path to the archive or a Track. If a Track, `track` will be ignored.
- **track** (str or `Track`, optional) – track label or a Track. If a Track, `archive` must not be specified. (default: `"bzna_input"`)

class benzina.torch.dataset.**ClassificationDataset**(*archive: Union[str, Tuple[Union[str, benzina.utils.file.Track], Union[str, benzina.utils.file.Track]]] = None, tracks: Tuple[Union[str, benzina.utils.file.Track], Union[str, benzina.utils.file.Track]] = ('bzna_input', 'bzna_target'), input_label: str = 'bzna_thumb'*)

Parameters

- **archive** (str or pair of `Track`) – path to the archive or a pair of Track. If a pair of Track, `tracks` will be ignored.
- **tracks** (pair of str or `Track`, optional) – pair of input and target tracks labels or a pair of input and target Track. If a pair of Track, `archive` must not be specified. (default: `("bzna_input", "bzna_target")`)
- **input_label** (*str, optional*) – label of the inputs to use in the input track. (default: `"bzna_thumb"`)

class benzina.torch.dataset.**ImageNet**(*root: Union[str, Tuple[Union[str, benzina.utils.file.Track], Union[str, benzina.utils.file.Track]]] = None, split: str = None, tracks: Tuple[Union[str, benzina.utils.file.Track], Union[str, benzina.utils.file.Track]] = ('bzna_input', 'bzna_target'), input_label: str = 'bzna_thumb'*)

Parameters

- **root** (str or pair of `Track`) – root of the ImageNet dataset or path to the archive or a pair of Track. If a pair of Track, `tracks` will be ignored.
- **split** (*`None or str, optional`*) – The dataset split, supports `test`, `train`, `val`. If not specified, samples will be drawn from all splits.
- **tracks** (pair of str or `Track`, optional) – pair of input and target tracks labels or a pair of input and target Track. If a pair of Track, `root` must not be specified. (default: `("bzna_input", "bzna_target")`)
- **input_label** (*`str, optional`*) – label of the inputs to use in the input track. (default: `"bzna_thumb"`)

1.7.3 benzina.torch.operations

class benzina.torch.operations.**WarpTransform**

Interface class that represents a warp transformation as a combined rotation, scale, skew and translation 3 x 3 matrix. The transformation is called for each sample of a batch.

class benzina.torch.operations.**NormTransform**

Interface class that represents a normalization transformation. The transformation is called for each sample of a batch.

class benzina.torch.operations.**BiasTransform**

Interface class that represents a bias transformation. The transformation is called for each sample of a batch.

class benzina.torch.operations.**ConstantWarpTransform**(*warp=(1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0)*)

Represents a constant warp transformation to be applied on each sample of a batch independently of its index.

Parameters **warp** (*`iterable of numerics, optional`*) – a flatten, row-major 3 x 3 warp matrix (default: flatten identity matrix).

class benzina.torch.operations.**ConstantNormTransform**(*norm=(1.0, 1.0, 1.0)*)

Represents a constant norm transformation to be applied on each sample of a batch independently of its index.

Parameters **norm** (*`numeric or iterable of numerics, optional`*) – an iterable in RGB order containing the normalization constant of a sample's RGB channels. Components will be multiplied to the respective channels of a sample (default: (1.0, 1.0, 1.0)).

class benzina.torch.operations.**ConstantBiasTransform**(*bias=(0.0, 0.0, 0.0)*)

Represents a constant bias transformation to be applied on each sample of a batch independently of its index.

Parameters **bias** (*`numeric or iterable of numerics, optional`*) – an iterable in RGB order containing the bias of a sample's RGB channels. Components will be substracted to the respective channels of a sample (default: (0.0, 0.0, 0.0)).

class benzina.torch.operations.**SimilarityTransform**(*scale=(1.0, 1.0), ratio=None, degrees=(-0.0, 0.0), translate=(0.0, 0.0), flip_h=0.0, flip_v=0.0, resize=False, keep_ratio=False, random_crop=False*)

Similarity warp transformation of the image keeping center invariant.

A crop of random size, aspect ratio and location is made. This crop can then be flipped and/or rotated to finally be resized to output size.

Parameters

- **scale** (*Sequence or float or int, optional*) – crop area scaling factor interval, e.g (a, b), then scale is randomly sampled from the range a <= scale <= b. If scale is a number instead of sequence, the range of scale will be (scale^-1, scale). (default: `(+1.0, +1.0)`)
- **ratio** (*Sequence or float or int, optional*) – range of crop aspect ratio. If ratio is a number instead of sequence like (min, max), the range of aspect ratio will be (ratio^-1, ratio). Will keep original aspect ratio by default.
- **degrees** (*Sequence or float or int, optional*) – range of degrees to select from. If degrees is a number instead of sequence like (min, max), the range of degrees will be (-degrees, +degrees). (default: `(-0.0, +0.0)`)
- **translate** (*Sequence or float or int, optional*) – sequence of maximum absolute fraction for horizontal and vertical translations. For example translate=(a, b), then horizontal shift is randomly sampled in the range -output_width * a < dx < output_width * a and vertical shift is randomly sampled in the range -output_height * b < dy < output_height * b. If translate is a number instead of sequence, translate will be (translate, translate). These translations are applied independently from `random_crop`. (default: `(0.0, 0.0)`)
- **flip_h** (*bool, optional*) – probability of the image being flipped horizontally. (default: `+0.0`)
- **flip_v** (*bool, optional*) – probability of the image being flipped vertically. (default: `+0.0`)
- **resize** (*bool, optional*) – resize the cropped image to fit the output size. It is forced to `True` if `scale` or `ratio` are specified. (default: `False`)
- **keep_ratio** (*bool, optional*) – match the smaller edge to the corresponding output edge size, keeping the aspect ratio after resize. Has no effect if `resize` is `False`. (default: `False`)
- **random_crop** (*bool, optional*) – randomly crop the image instead of a center crop. (default: `False`)

class benzina.torch.operations.**RandomResizedCrop**(*scale=(0.08, 1.0), ratio=(0.75, 1.3333333333333333)*)

Crop to random size, aspect ratio and location.

A crop of random size, aspect ratio and location is made. This crop is finally resized to output size.

This is popularly used to train the Inception networks.

Parameters

- **scale** (*Sequence or float or int, optional*) – crop area scaling factor interval, e.g (a, b), then scale is randomly sampled from the range a <= scale <= b. If scale is a number instead of sequence, the range of scale will be (scale^-1, scale). (default: `(+0.08, +1.0)`)
- **ratio** (*Sequence or float or int, optional*) – range of crop aspect ratio. If ratio is a number instead of sequence like (min, max), the range of aspect ratio will be (ratio^-1, ratio). Will keep original aspect ratio by default. (default: `(3./4., 4./3.)`)

class benzina.torch.operations.**CenterResizedCrop**(*scale=1.0, keep_ratio=True*)

Crops at the center and resize.

A crop at the center is made then resized to the output size.

Parameters

- **scale** (*float or int, optional*) – edges scaling factor. (default: `+1.0`)
- **keep_ratio** (*bool, optional*) – match the smaller edge to the corresponding output edge size, keeping the aspect ratio after resize. Has no effect if `resize` is

False. (default: False)

benzina.torch.operations.**compute_affine_matrix**(*in_shape*, *out_shape*, *crop=None*, *degrees=0.0*, *translate=(0.0, 0.0)*, *flip_h=False*, *flip_v=False*, *resize=False*, *keep_ratio=False*)

Similarity warp transformation of the image keeping center invariant.

Parameters

- **in_shape** (*Sequence*) – the shape of the input image
- **out_shape** (*Sequence*) – the shape of the output image
- **crop** (*Sequence, optional*) – crop center location, width and height. The center location is relative to the center of the image. If resize is not True, crop is simply a translation in the in_shape space.
- **degrees** (*float or int, optional*) – degrees to rotate the crop. (default: (0.0))
- **translate** (*Sequence, optional*) – horizontal and vertical translations. (default: (0.0, 0.0))
- **flip_h** (*bool, optional*) – flip the image horizontally. (default: False)
- **flip_v** (*bool, optional*) – flip the image vertically. (default: False)
- **resize** (*bool, optional*) – resize the cropped image to fit the output's size. (default: False)
- **keep_ratio** (*bool, optional*) – match the smaller edge to the corresponding output edge size, keeping the aspect ratio after resize. Has no effect if resize is False. (default: False)

CHAPTER 2

Description of the project

Benzina is an image loading library that accelerates image loading and preprocessing by making use of the hardware decoder in NVIDIA's GPUs.

Since it minimize the use of the CPU and of the GPU computing units, it's easier to reach saturation of GPU computing power / CPU. In our tests using ResNet18 models in PyTorch on the ImageNet 2012 dataset, we could observe an increase by 1.8x the amount of images loaded, preprocessed then processed by the model when using a single CPU and GPU:

Data Loader	CPU	CPU Workers	CPU Usage	GPU	Batch Size	Pipeline Speed
Benzina	Intel Xeon 2698*	1	33%	Tesla V100*	256	525 img/s
PyTorch ImageFolder	Intel Xeon 2698*	2	100%	Tesla V100*	256	290 img/s
PyTorch ImageFolder	Intel Xeon 2698*	4	100%	Tesla V100*	256	395 img/s
PyTorch ImageFolder	Intel Xeon 2698*	6	100%	Tesla V100*	256	425 img/s
DALI	Intel Xeon 2698*	1	100%	Tesla V100*	256	575 img/s

Note:

- Intel Xeon 2698 is the Intel Xeon E5-2698 v4 @ 2.20GHz version
- Tesla V100 is the Tesla V100 SXM2 16GB version

While DALI currently outperforms Benzina, the speedup can only be seen on JPEGs through the nvJPEG decoder. Benzina requires to transcode the input dataset to H.265 but then the gain can be seen on all type of images as well as providing the dataset in a format that is easier to distribute.

The name "Benzina" is a phonetic transliteration of the Ukrainian word "", meaning "gasoline" (or "petrol").

CHAPTER 3

Indices and tables

- genindex
- modindex
- search