Video Dataset Loading PyTorch

Release 1.0

Raivo Koot

Dec 31, 2021

CONTENTS

CHAPTER

ONE

VIDEODATASET MODULE

class `video_dataset.`**`ImglistToTensor`**

Bases: `torch.nn.modules.module.Module`

Converts a list of PIL images in the range [0,255] to a torch.FloatTensor of shape (NUM_IMAGES x CHANNELS x HEIGHT x WIDTH) in the range [0,1]. Can be used as first transform for `VideoFrameDataset`.

static forward(*img_list: List[PIL.Image.Image]*) → torch.Tensor[NUM_IMAGES, CHANNELS, HEIGHT, WIDTH]

Converts each PIL image in a list to a torch Tensor and stacks them into a single tensor.

Parameters **`img_list`** – list of PIL images.

Returns tensor of size `NUM_IMAGES x CHANNELS x HEIGHT x WIDTH`

class `video_dataset.`**`VideoFrameDataset`**(*root_path: str*, *annotationfile_path: str*, *num_segments: int = 3*, *frames_per_segment: int = 1*, *imagefile_template: str = 'img_{:05d}.jpg'*, *transform=None*, *test_mode: bool = False*)

Bases: `torch.utils.data.dataset.Dataset`

A highly efficient and adaptable dataset class for videos. Instead of loading every frame of a video, loads x RGB frames of a video (sparse temporal sampling) and evenly chooses those frames from start to end of the video, returning a list of x PIL images or `FRAMES x CHANNELS x HEIGHT x WIDTH` tensors where FRAMES=x if the `ImglistToTensor()` transform is used.

More specifically, the frame range [START_FRAME, END_FRAME] is divided into NUM_SEGMENTS segments and FRAMES_PER_SEGMENT consecutive frames are taken from each segment.

Note: A demonstration of using this class can be seen in `demo.py` https://github.com/RaivoKoot/Video-Dataset-Loading-Pytorch

Note: This dataset broadly corresponds to the frame sampling technique introduced in `Temporal Segment Networks` at ECCV2016 https://arxiv.org/abs/1608.00859.

Note: This class relies on receiving video data in a structure where inside a `ROOT_DATA` folder, each video lies in its own folder, where each video folder contains the frames of the video as individual files with a naming convention such as img_001.jpg ... img_059.jpg. For enumeration and annotations, this class expects to receive the path to a .txt file where each video sample has a row with four (or more in the case of multi-label, see README on Github) space separated values: `VIDEO_FOLDER_PATH START_FRAME END_FRAME LABEL_INDEX`. `VIDEO_FOLDER_PATH` is expected to be the path of a video folder excluding the `ROOT_DATA` prefix. For example, `ROOT_DATA` might be `home\data\datasetxyz\videos\`, inside of which a `VIDEO_FOLDER_PATH` might be `jumping\0052\` or `sample1\` or `00053\`.

Parameters

- **`root_path`** – The root path in which video folders lie. this is ROOT_DATA from the description above.
- **`annotationfile_path`** – The .txt annotation file containing one row per video sample as described above.
- **`num_segments`** – The number of segments the video should be divided into to sample frames from.
- **`frames_per_segment`** – The number of frames that should be loaded per segment. For each segment's frame-range, a random start index or the center is chosen, from which frames_per_segment consecutive frames are loaded.
- **`imagefile_template`** – The image filename template that video frame files have inside of their video folders as described above.
- **`transform`** – Transform pipeline that receives a list of PIL images/frames.
- **`test_mode`** – If True, frames are taken from the center of each segment, instead of a random location in each segment.

CHAPTER

TWO

EFFICIENT VIDEO DATASET LOADING, PREPROCESSING, AND AUGMENTATION

To get the most up-to-date README, please visit Github: Video Dataset Loading Pytorch

Author: Raivo Koot

If you are completely unfamiliar with loading datasets in PyTorch using `torch.utils.data.Dataset` and `torch.utils.data.DataLoader`, I recommend getting familiar with these first through this or this.

2.1 Overview: This example demonstrates the use of `VideoFrameDataset`

The VideoFrameDataset class serves to `easily`, `efficiently` and `effectively` load video samples from video datasets in PyTorch.

1) Easily because this dataset class can be used with custom datasets with minimum effort and no modification. The class merely expects the video dataset to have a certain structure on disk and expects a .txt annotation file that enumerates each video sample. Details on this can be found below and at `https://video-dataset-loading-pytorch.readthedocs.io/`.

2) Efficiently because the video loading pipeline that this class implements is very fast. This minimizes GPU waiting time during training by eliminating input bottlenecks that can slow down training time by several folds.

3) Effectively because the implemented sampling strategy for video frames is very strong. Video training using the entire sequence of video frames (often several hundred) is too memory and compute intense. Therefore, this implementation samples frames evenly from the video (sparse temporal sampling) so that the loaded frames represent every part of the video, with support for arbitrary and differing video lengths within the same dataset. This approach has shown to be very effective and is taken from "Temporal Segment Networks (ECCV2016)" with modifications.

In conjunction with PyTorch's DataLoader, the VideoFrameDataset class returns video batch tensors of size `BATCH x FRAMES x CHANNELS x HEIGHT x WIDTH`.

For a demo, visit `https://github.com/RaivoKoot/Video-Dataset-Loading-Pytorch`.

2.2 QuickDemo (demo.py)

```
root = os.path.join(os.getcwd(), 'demo_dataset')  # Folder in which all videos lie in␣
↪a specific structure
annotation_file = os.path.join(root, 'annotations.txt')  # A row for each video␣
↪sample as: (VIDEO_PATH NUM_FRAMES CLASS_INDEX)

""" DEMO 1 WITHOUT IMAGE TRANSFORMS """
dataset = VideoFrameDataset(
    root_path=root,
    annotationfile_path=annotation_file,
    num_segments=5,
    frames_per_segment=1,
    image_template='img_{:05d}.jpg',
    transform=None,
    random_shift=True,
    test_mode=False
)

sample = dataset[0]  # take first sample of dataset
frames = sample[0]   # list of PIL images
label = sample[1]    # integer label

for image in frames:
    plt.imshow(image)
    plt.title(label)
    plt.show()
    plt.pause(1)
```

CHAPTER

THREE

TABLE OF CONTENTS

3.1 1. Requirements

```
# Without these three, VideoFrameDataset will not work.
torchvision >= 0.8.0
torch >= 1.7.0
python >= 3.6
```

3.2 2. Custom Dataset

To use any dataset, two conditions must be met. 1) The video data must be supplied as RGB frames, each frame saved as an image file. Each video must have its own folder, in which the frames of that video lie. The frames of a video inside its folder must be named uniformly as `img_00001.jpg` ... `img_00120.jpg`, if there are 120 frames. The filename template for frames is then "img_{:05d}.jpg" (python string formatting, specifying 5 digits after the underscore), and must be supplied to the constructor of VideoFrameDataset as a parameter. Each video folder lies inside a `root` folder of this dataset. 2) To enumerate all video samples in the dataset and their required metadata, a `.txt` annotation file must be manually created that contains a row for each video sample in the dataset. The training, validation, and testing datasets must have separate annotation files. Each row must be a space-separated list that contains `VIDEO_PATH NUM_FRAMES CLASS_INDEX`. The `VIDEO_PATH` of a video sample should be provided without the `root` prefix of this dataset.

This example project demonstrates this using a dummy dataset inside of `demo_dataset/`, which is the `root` dataset folder of this example. The folder structure looks as follows:

```
demo_dataset
│
├───annotations.txt
├───jumping # arbitrary class folder naming
│       ├───0001  # arbitrary video folder naming
```

(continues on next page)

(continued from previous page)

```
│         │      ├──img_00001.jpg
│         │      .
│         │      └──img_00017.jpg
│         └──0002
│                ├──img_00001.jpg
│                .
│                └──img_00018.jpg
│
└──running # arbitrary folder naming
          ├──0001  # arbitrary video folder naming
          │      ├──img_00001.jpg
          │      .
          │      └──img_00015.jpg
          └──0002
                 ├──img_00001.jpg
                 .
                 └──img_00015.jpg
```

The accompanying annotation `.txt` file contains the following rows

```
jumping/0001 17 0
jumping/0002 18 0
running/0001 15 1
running/0002 15 1
```

Instantiating a VideoFrameDataset with the `root_path` parameter pointing to `demo_dataset`, the `annotationsfile_path` parameter pointing to the annotation file, and the `imagefile_template` parameter as "img_{:05d}.jpg", is all that it takes to start using the VideoFrameDataset class.

3.3 3. Video Frame Sampling Method

When loading a video, only a number of its frames are loaded. They are chosen in the following way: 1. The frame indices [1,N] are divided into NUM_SEGMENTS even segments. From each segment, FRAMES_PER_SEGMENT consecutive indices are chosen at random. This results in NUM_SEGMENTS*FRAMES_PER_SEGMENT chosen indices, whose frames are loaded as PIL images and put into a list and returned when calling `dataset[i]`.

3.4 4. Using VideoFrameDataset for training

As demonstrated in `https://github.com/RaivoKoot/Video-Dataset-Loading-Pytorch/blob/main/demo.py`, we can use PyTorch's `torch.utils.data.DataLoader` class with VideoFrameDataset to take care of shuffling, batching, and more. To turn the lists of PIL images returned by VideoFrameDataset into tensors, the transform `video_dataset.imglist_totensor()` can be supplied as the `transform` parameter to VideoFrameDataset. This turns a list of N PIL images into a batch of images/frames of shape `N x CHANNELS x HEIGHT x WIDTH`. We can further chain preprocessing and augmentation functions that act on batches of images onto the end of `imglist_totensor()`.

As of `torchvision 0.8.0`, all torchvision transforms can now also operate on batches of images, and they apply deterministic or random transformations on the batch identically on all images of the batch. Therefore, any torchvision transform can be used here to apply video-uniform preprocessing and augmentation.

3.5 5. Conclusion

A proper code-based explanation on how to use VideoFrameDataset for training is provided in `https://github.com/RaivoKoot/Video-Dataset-Loading-Pytorch/blob/main/demo.py`

3.6 6. Acknowledgements

We thank the authors of TSN for their codebase, from which we took VideoFrameDataset and adapted it.

PYTHON MODULE INDEX

INDEX