Data Loading

Solve the following exercises and upload your solutions to Moodle (unless specified otherwise) until the specified due date. Make sure to use the *exact filenames* that are specified for each individual exercise. Unless explicitly stated otherwise, you can assume correct user input and correct arguments. You are allowed to write additional functions, classes, etc. to improve readability and code quality.

Exercise 1 – Submission: `a3_ex1.py` 60 Points

Write a class `ImagesDataset` that extends `torch.utils.data.Dataset` and is responsible for providing the fixed-size gray-scale images and their additional data which was part of the previous assignment. The class has the following three instance methods:

-
  ```
  __init__(
      self,
      image_dir,
      width: int = 100,
      height: int = 100,
      dtype: Optional[type] = None
  )
  ```

  `image_dir` specifies the directory of validated images as the output directory by function `validate_images` from assignment 1. Assume that all image files with the extension `".jpg"` and the class file with the extension `".csv"` stored directly in the directory, not be in subdirectories. The found files must be collected using their absolute paths, and the list of these must be sorted afterwards in ascending order. The corresponding class names of images are loaded from the `".csv"` file. You can use `"numpy.genfromtxt()"` or `"pandas.read_csv()"` to read the class names file. A list of distinct class names must be sorted in ascending order, and their index is used as its respective class ID.

  `width` and `height` specify the fixed size of the resized copy of the images loaded from `image_dir`. If `width` or `height` are smaller than 100, a `ValueError` must be raised.

  `dtype` optionally specifies the data type of the loaded images (see `__getitem__` below).

- `__getitem__(self, index)`

  This method works as follow:

  - Given the specified integer `index`, the `index`-th image from the sorted list of image files (see `__init__` above) must be loaded with `PIL.Image.open`.
  - The image is then stored in a NumPy array using the optionally specified `dtype` (otherwise, the default data type is used).
  - This image array is then transformed into gray-scale using the `to_grayscale` method from the previous assignment.
  - Afterwards, again from the previous assignment, the method `prepare_image(image, width, height, x=0, y=0, size=32)` must be called where `width` and `height` are the fixed size of the resized image. The subarea of the resized image is not used, we therefore only pass fixed arguments for `x`, `y`, and `size`.

The method must then return the following 4-tuple: `(image, class_id, class_name, image_filepath)`, which are the fixed-size gray-scale copy, class ID (value), class name, and the absolute file path of the loaded image respectively.

- `__len__(self)`

  Returns the number of samples, i.e., the number of images that were found in `__init__`.

Example program execution:

```
dataset = ImagesDataset("./validated_images", 100, 100, int)
for resized_image, classid, classname, _ in dataset:
    print(f'image shape: {resized_image.shape}, dtype: {resized_image.dtype}, '
          f'classid: {classid}, classname: {classname}\n')
```

Example output (assuming some images in the provided directory):

```
image shape: (1, 100, 100), dtype: int32, classid: 0, classname: cloud

image shape: (1, 100, 100), dtype: int32, classid: 1, classname: mountain

image shape: (1, 100, 100), dtype: int32, classid: 2, classname: snow
```

Exercise 2 – Submission: a3_ex2.py 40 Points

Write a function stacking(batch_as_list: list) that can be used as collate_fn function of a torch.utils.data.DataLoader. It must work on samples provided by ImagesDataset (see exercise above), i.e., 4-tuples of (image, class_id, class_name, image_filepath), as follows:

- Each image must be stacked. The stacking dimension must be the first dimension, i.e., the stacked result has the shape (N, 1, H, W), where N is the batch size (the number of samples in the given batch), 1 the brightness channel size, and H is the height and W the width of the batch images. The data type of the stacked result must match the data type of the images. Ultimately, the stacked result must be converted to a PyTorch tensor.
- Each class_id must also be stacked in a similar way, i.e., the stacked result has the shape (N, 1), where N is the batch size (the number of samples in the given batch), 1 the class id. The stacked result must be converted to a PyTorch tensor.
- Each class_name and each image_filepath must be stored in two separate list (no conversion is done here).

The function must then return the following 4-tuple: (stacked_images, stacked_class_ids, class_names, image_filepaths), where the individual entries are as explained above.

Example program execution:

```
ds = ImagesDataset("./validated_images", 100, 100, int)
dl = DataLoader(ds, batch_size=2, shuffle=False, collate_fn=stacking)
for i, (images, classids, classnames, image_filepaths) in enumerate(dl):
    print(f'mini batch: {i}')
    print(f'images shape: {images.shape}')
    print(f'class ids: {classids}')
    print(f'class names: {classnames}\n')
```

Example output (assuming some images in the provided directory):

```
mini batch: 0
images shape: torch.Size([2, 1, 100, 100])
class ids: tensor([[0],
                   [1]], dtype=torch.int32)
class names: ['cloud', 'mountain']

mini batch: 1
images shape: torch.Size([1, 1, 100, 100])
class ids: tensor([[2]], dtype=torch.int32)
class names: ['snow']
```