Data Augmentation

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: `a6_ex1.py` 70 Points

1.1 Write a function (40 points)

```
augment_image(
    img_np: np.ndarray,
    index: int
) -> torch.Tensor, str
```

that transforms an image and returns a transformed image as a PyTorch tensor and the corresponding transformation name.

The function's parameters are as follows:

- `img_np`: An input image, after resizing and grey-scaling, retrieved from `ImagesDataset` of Assignment 3.
- `index`: Depending on this value, an option to transform randomly the input image is selected from a predefined list of transformations. This index value is also used for the reproducibility of the resulting output image.

The transformation list consists of the following 5 transformations with their corresponding numbers:

1. `torchvision.transforms.GaussianBlur`
2. `torchvision.transforms.RandomRotation`
3. `torchvision.transforms.RandomVerticalFlip`
4. `torchvision.transforms.RandomHorizontalFlip`
5. `torchvision.transforms.ColorJitter`

There are 7 transformation options according to the result of the modulo division `v = index % 7`, in which `v` determines the applied transformation based on the corresponding transformation number.

1. For case `v = 0`, the function returns the input image (no transformation applied but converted to a `torch.Tensor`), use `"Original"` for the corresponding transformation name.
2. For cases `v = 1, ..., 5`, the transformation corresponding to the number is applied on the input image, and use the name of the transformation class for the transformation name.
3. For case `v = 6`, a chain of 3 transformations selected randomly are applied on the input image, use`"Compose"` for the corresponding transformation name.

There are no specific requirements on the arguments and the order of transformations in the chain as long as their data types are compatible between two transformations and the data type of the function's output is a PyTorch tensor (`torch.Tensor`).

1.2 Write the following class (30 points):

Class `TransformedImagesDataset` that extends `Dataset` to provide augmented images from an input dataset of images. The following methods must be implemented:

- `__init__(self, data_set: Dataset)`: `data_set` is an input dataset, e.g. an instance of class `ImagesDataset` defined in Assignment 3.
- `__getitem__(self, index: int)`: An image from the input dataset is retrieved through the input `index` (the corresponding index in the input dataset have to be derived from `index`). Afterwards, the image is transformed with the above function `augment_image`. The method must return a 6-tuple in which successive elements are as follows: the transformed PyTorch tensor image, the applied transformation name, the index, class ID, class name, image file path. Assume that an original image and its transformed versions have successive indices and share the same image file path.
- `__len__(self)`: The length of the augmented dataset.

Example program execution:

```python
if __name__ == "__main__":
    from matplotlib import pyplot as plt

    dataset = ImagesDataset("./validated_images", 100, 100, int)
    transformed_ds = TransformedImagesDataset(dataset)
    fig, axes = plt.subplots(2, 4)
    for i in range(0, 8):
        trans_img, trans_name, index, classid, classname, img_path =
        transformed_ds.__getitem__(i)
        _i = i // 4
        _j = i % 4
        axes[_i, _j].imshow(transforms.functional.to_pil_image(trans_img))
        axes[_i, _j].set_title(f'{trans_name}\n{classname}')

    fig.tight_layout()
    plt.show()
```

Example output:

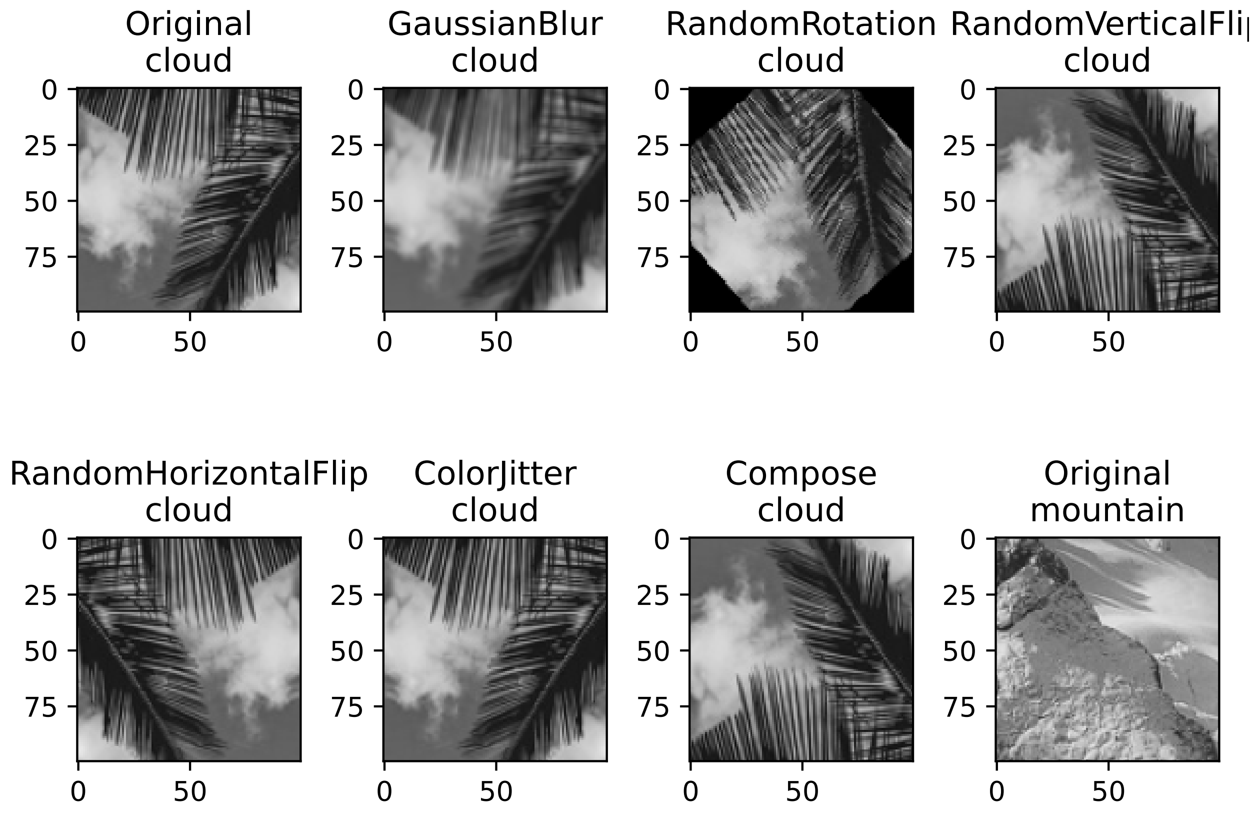

Figure 1: Visualization of the original and the transformed images retrieved from `TransformedImagesDataset`

Exercise 2 – Submission: `a6_ex2.py` 30 Points

Write a function `stacking(batch_as_list: list)` that can be used as a `collate_fn` function of a `torch.utils.data.DataLoader`. It must work on samples provided by `TransformedImagesDataset` (see exercise above), i.e., 6-tuples of `(trans_img, trans_name, index, class_id, class_name, img_path)`, as follows:

- Each `trans_img` 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` indicates the number of image channels, `H` and `W` are the height and the width of the batch images respectively. The stacked result must be a PyTorch tensor. The data type of the stacked result must match the data type of the images.
- Each `index` or 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` indicates the index or class id. The stacked result must be converted to a PyTorch tensor.
- Each `trans_name`, each `class_name`, and each `img_path` must be stored in three separate lists (no conversion is done here).

The function must then return the following 6-tuple: `(stacked_images, trans_names, stacked_indices, stacked_class_ids, class_names, img_paths)`, where the individual entries are as explained above.

Example program execution:

```python
if __name__ == "__main__":
    dataset = ImagesDataset("./validated_images", 100, 100, int)
    transformed_ds = TransformedImagesDataset(dataset)
    dl = DataLoader(transformed_ds, batch_size=7, shuffle=False, collate_fn=stacking)
    for i, (images, trans_names, indices, classids, classnames, img_paths) in enumerate(dl):
        print(f'mini batch: {i}')
        print(f'images shape: {images.shape}')
        print(f'trans_names: {trans_names}')
        print(f'indices: {indices}')
        print(f'class ids: {classids}')
        print(f'class names: {classnames}\n')
```

Example output:

```
mini batch: 0
images shape: torch.Size([7, 1, 100, 100]), data-type: torch.int32
trans_names: ['Original', 'GaussianBlur', 'RandomRotation', 'RandomVerticalFlip',
'RandomHorizontalFlip', 'ColorJitter', 'Compose']
indices: tensor([[0], [1], [2], [3], [4], [5], [6]], dtype=torch.int32)
class ids: tensor([[0], [0], [0], [0], [0], [0], [0]], dtype=torch.int32)
class names: ['cloud', 'cloud', 'cloud', 'cloud', 'cloud', 'cloud', 'cloud']

mini batch: 1
images shape: torch.Size([7, 1, 100, 100]), data-type: torch.int32
trans_names: ['Original', 'GaussianBlur', 'RandomRotation', 'RandomVerticalFlip',
'RandomHorizontalFlip', 'ColorJitter', 'Compose']
indices: tensor([[ 7], [ 8], [ 9], [10], [11], [12], [13]], dtype=torch.int32)
class ids: tensor([[1], [1], [1], [1], [1], [1], [1]], dtype=torch.int32)
class names: ['mountain', 'mountain', 'mountain', 'mountain', 'mountain',
'mountain', 'mountain']

mini batch: 2
images shape: torch.Size([7, 1, 100, 100])
trans_names: ['Original', 'GaussianBlur', 'RandomRotation', 'RandomVerticalFlip',
'RandomHorizontalFlip', 'ColorJitter', 'Compose']
indices: tensor([[14], [15], [16], [17], [18], [19], [20]], dtype=torch.int32)
class ids: tensor([[2], [2], [2], [2], [2], [2], [2]], dtype=torch.int32)
class names: ['snow', 'snow', 'snow', 'snow', 'snow', 'snow', 'snow']
```