Assignment no. 2

Scenario: Participating in a ML challenge

Now you will implement your NN using PyTorch and train/tune it to compete in the project challenge. First you will write a function that creates the input and target arrays used in the challenge. Then you will implement/train/tune your NN to process an input image and predict a target array for the project challenge.

Exercise 4 [20 points]

As mentioned in the lecture, we can create our inputs and targets for training directly from our images.

Therefore, in this exercise you should create a function `ex4(image_array, border_x, border_y)` that creates two input arrays and one target array from one input image. For this, your function should remove (=set values to zero) the specified borders in the image. These removed values will later become the target, i.e. our network has to learn to recreate the values that we set to zero.

Since it could be valuable information for our network to know which part is known and which part was removed/never known, we will also prepare an additional input channel that includes information about which pixel values are unknown.

In detail, your function should take the following keyword arguments:

- `image_array`: A numpy array of shape `(X, Y)` and arbitrary datatype, which contains the image data.
- `border_x`: A tuple containing 2 `int` values. These two values specify the border thickness at the start and end of the x axis of the `image_array`, respectively. Please see the figure below for more information.
- `border_y`: A tuple containing 2 `int` values. These two values specify the border thickness at the start and end of the y axis of the `image_array`, respectively. Please see the figure below for more information.

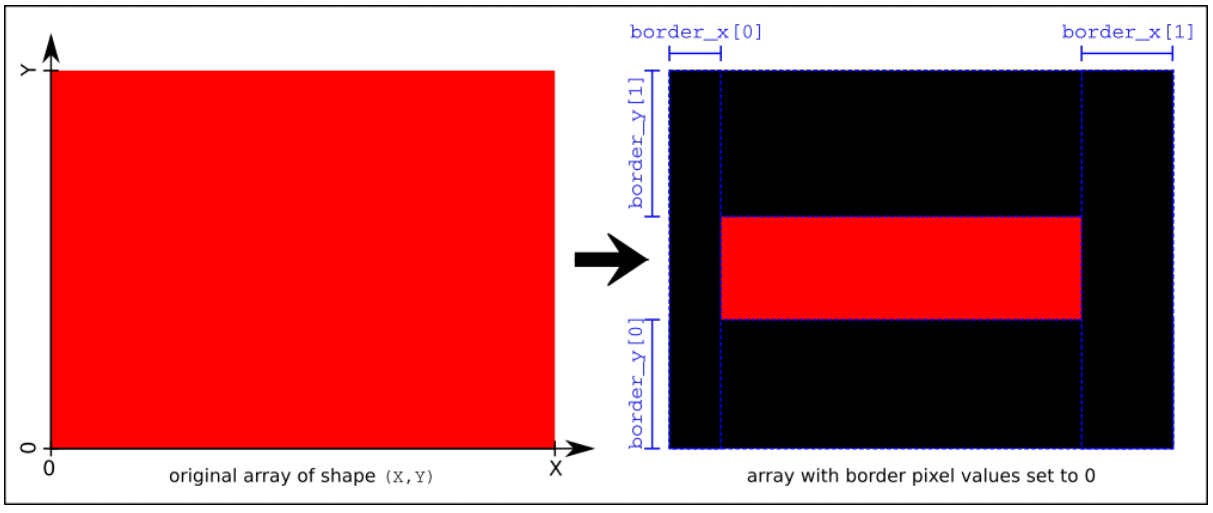

Fig.1: `image_array` with border specifications.

Your function should return a tuple `(input_array, known_array, target_array)`.

`input_array` should be a 2D numpy array of same shape and datatype as `image_array`. It should have the same pixel values as `image_array`, with the exception that the to-be-removed pixel values in the specified borders are set to 0. You may edit the original `image_array` in-place or create a copy.

`known_array` should be a 2D numpy array of same shape and datatype as `image_array`, where pixels in the specified borders should have value 0 and other, known, pixels have value 1.

`target_array` should be a 1D numpy array of the same datatype as `image_array`. It should hold the pixel values of the specified borders (the pixels that were set to 0 in `input_array`). The order of the pixels in `target_array` should be the same as if one would use `known_array` as boolean mask on `image_array` (like `image[boolean_mask]`). The length of `target_array` should therefore be the number of pixels in the specified borders.

Your function should raise a `NotImplementedError` exception if:

- `image_array` is not a numpy array (see hints on how to check if an object is a numpy array instance).
- `image_array` is not a 2D array.

Your function should raise a `ValueError` exception if:

- The values in `border_x` and `border_y` are not convertable to `int` objects.
- The values in `border_x` and `border_y` are smaller than 1.
- The shape of the remaining known image pixels would be smaller than `(16, 16)`.

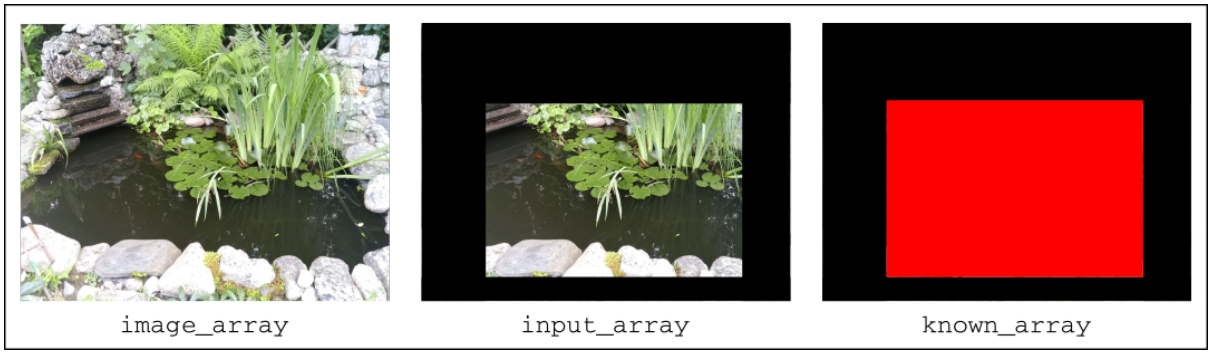

Fig.2: Example for `image_array`, `input_array`, and `known_array`. `target_array` contains the pixel values in `image_array` that are located in the borders and set to 0 in `input_array`.

Hint: To check whether an object is an instance of a certain type, you can use the `isinstance()` function. Numpy arrays are instances of `np.ndarray`.

Hint: To create an array with same shape and datatype as another array, you can use the `np.zeros_like()`, `np.ones_like()`, `np.empty_like()`, or `np.full_like()` functions.

Hint: Slicing and other indexing operations will typically not create copies of a numpy array, which allows for in-place modification of array elements. `np.copy(my_array)` will create an actual copy of an array and its values. In-place operations are more memory efficient and faster than copying the array elements to a new array but might have adverse effects if you overwrite values that you would need later.

Exercise 5 [40 points + 10 bonus points]

Exercise 5 is the ML challenge, where points are determined by the performance of your model. You will have to create and train a neural network to extrapolate unknown border pixels of an image. The challenge is accessible as *Image extrapolation challenge 2021* at `https://apps.ml.jku.at/challenge`. User logins for submission will be provided on 19.05.2020.

Challenge specifications:

Train a neural network to extrapolate unknown parts of an image.

- Samples considered in the challenge are grayscale images for which a certain amount of pixels at the borders of the images is unknown (=was set to zero).
- Your model should predict (=extrapolate) the unknown border pixel values (see `target_array` in exercise 4).
- The images collected in exercise 1 will be the training set, however, you are free to include more images of your choosing into your training set. Since we already collected a lot of images, training on the training set itself will be sufficient.
- The download-link for the training set is given in Moodle.

Predict the unknown parts of the test set images.

- You will be provided with test set images, where a certain amount of pixels at the borders of the images is unknown (=was set to zero).
- The test set images will have a shape of `(90, 90)` pixels. They have been downscaled from various shapes using `transforms.Resize(size=im_shape)` with `BILINEAR` interpolation (`https://pytorch.org/vision/stable/transforms.html#torchvision.transforms.Resize`).
- The area of known pixels in the test set images will be at least `(75, 75)` pixels large.
- The borders containing unknown pixels in the test set images will be at least `5` pixels wide on each side.
- The test set images and the specifications of their unknown borders will be provided as pickle file. The pickle file will contain a dictionary with the following entries:
  - `"input_arrays"` (list of `"input_array"` as created via exercise 4 for each sample),
  - `"known_arrays"` (list of `"known_array"` as created via exercise 4 for each sample),
  - `"borders_x"` (`"border_x"` kwarg as used for exercise 4 for each sample),
  - `"borders_y"` (`"border_y"` kwarg as used for exercise 4 for each sample),
  - `"sample_ids"` (sample ID as string for each sample).
- The download-link to the true test set will be given in Moodle starting on 19.05.2020.

- The download-link to the supplements which contain an example test set and example predictions is given in Moodle.

Upload your predictions for the unknown pixel values.

- You will need to upload your predictions to the challenge server.
  - Your upload should be a pickle file that contains a list of `np.uint8` Numpy arrays.
  - Each array should hold the predictions for the unknown pixels of a sample (same format/layout as `target_array` in exercise 4).
  - The order of samples in this list should be the same as the order of samples in the test set pickle file.
- You only have 5 attempts to upload predictions.
- Only valid attempts will be counted. (E.g. it will not count as attempt if you upload a wrong format.)
- Your best attempt will be used as final attempt.
- You will also need to upload the Python code you used for your solution in Moodle as `.zip` or `.py` file.
  - The `.py` or `.zip` file must include the Python code you used to train your network and create the predictions.
  - This Python code has to be submitted as `.py` Python file or multiple `.py` Python files.
  - You may optionally include other files and file types (e.g. .json config files or .pdf files for documentation purposes).
  - This will be used to check for plagiarism and to verify that your model corresponds to the uploaded predictions.
  - Make sure to include all of your code files.
- The download-link to the supplements which contain an example test set and example predictions is given in Moodle.

The score of your model will be compared to 2 other models, ModelA and ModelB, for grading.

- The negative mean squared error between the true unknown pixel values and your predictions for the unknown pixel values will be used as score.
- After uploading a valid attempt to the challenge server, the score will be automatically computed and visible in the leader board.
- ModelA uses the mean value of the input image as prediction of the unknown pixel values.
- ModelB uses a CNN consisting of 3 hidden convolutional layers + 1 convolutional output layer, a kernel size of 7, and 32 kernels per CNN layer to predict the unknown pixel values.

- If your model has a lower or equal score than ModelA, you get 0 points.
- If your model has a higher or equal score than ModelB, you get 35 points.
- You will also receive bonus points, depending on how much higher your model score is compared to ModelB.
- Points for models with scores between ModelA and ModelB will be interpolated linearly.
- ModelA and ModelB scores will be visible starting with 2nd of June.
- The download-link to the supplements which contain a copy of the file that is used to compute the scores on the server is given in Moodle.

Important. You will only have 5 attempts to submit predictions, so it will be important for you to use some samples for a validation set and maybe another test set to get an estimate for the generalization of your model.

Deadline. Deadline for the submission is the 30th of August, 23:55. (An option for earlier grading will be provided.)

Hints:
Divide the project into sub-tasks and check your program after creation of each sub-task, e.g.

1. decide which samples you want to use in your training-, validation-, or test sets,
2. create the data reader (re-use your code from exercise 4),
3. create the data loader and stacking function for the minibatches (see code files for Unit 05),
4. implement a NN that computes an output given this input (see code files for Unit 06),
5. implement the computation of the loss between NN output and target (see code files for Unit 07),
6. implement the NN training loop (see code files for Unit 07),
7. implement the evaluation of the model performance on a validation set with early stopping (see code files for Unit 07).

You may want to use the project structure shown in the example project in the code files for Unit 07.

It makes sense to only work with inputs of sizes that can appear in the test set. For this you can use the torchvision transforms to downscale the input images using `BILINEAR` as interpolation method. For example:

```
from torchvision import transforms
from PIL import Image
im_shape = 90
resize_transforms = transforms.Compose([
    transforms.Resize(size=im_shape),
    transforms.CenterCrop(size=(im_shape, im_shape)),
])
image = Image.open(filename)
image = resize_transforms(image)
```

However, if you want to drastically increase your dataset size using data augmentation, you can also use random cropping followed by resizing to create more input images (e.g. via `torchvision.transforms.RandomResizedCrop`). More information on augmentation will follow in Unit 08.

You will need to write a stacking function for the DataLoader (`collate_fn`). For this you can take the maximum over the X and the maximum over the Y dimensions of the input array and create a zero-tensor of shape
`(n_samples, n_feature_channels, max_X, max_Y)` (so that it can hold the stacked input arrays). Then you can copy the input values into this zero-tensor. For a 1D example see "Task 01" in `05_solutions.py`.

It makes sense to feed additional input into the NN. You can concatenate the channels of `image_array` and `known_array` (see exercise 4) and feed the resulting tensor as input into the network.

Creating predictions and computing loss: To predict the unknown pixel values, you can implement a CNN that creates an output that has the same size as the input (see code files for Unit 07). Then you can use either a boolean mask like `known_array` or slicing as in exercise 4 to obtain the predicted pixel values.

If you normalize the NN input, de-normalize the NN output accordingly if you want to predict a not-normalized target array. The challenge inputs and targets will not be normalized. You do not have access to the targets to normalize them, so you will need to create not-normalized predictions. In practice this might be done by saving the mean and variance you used for normalizing the input and using these values to de-normalize the NN output.

Start with a small subset of dataset to quickly get suitable hyperparameters. Use a small subset (e.g. 30 samples) of your training set for debugging. You model should be able to overfit (=achieve almost perfect performance) on such a small dataset. You can guess the minimum number of necessary kernel sizes and numbers of layers from the maximum border width (see test set specifications). The parameters of ModelB might also be a good starting point.

Debug properly. Check the inputs and outputs of your network manually. Do not trust that everything works just because the loss decreases. Debug with `num_workers=0` for the DataLoader to be able to step into the data loading process.

You do not need to re-invent the wheel. Most of this project can be solved by re-using parts of the code materials from this semester. If you need help, ask in due time (see `https://moodle.jku.at/jku/mod/forum/discuss.php?d=62615` for help-options).

Exercise 6 [10 bonus points]

(tba)

Submission: electronically via Moodle:

`https://moodle.jku.at/`

Deadline: For deadlines see individual Moodle exercises.
Follow the **instructions for submitting homework** stated on the Moodle page!