Assignment 2

Programming in Python II

Scenario: Participating in the 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.

IMPORTANT: Make sure to adhere to the *Instructions for submitting homework* on Moodle!

Exercise 4 [20 points]

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

In this exercise, you have to create a function `ex4(image_array, offset, spacing)` that creates two input arrays and one target array from a given input image. For this, your function has to remove (=set values to zero) all pixels that do not lie on a grid specified by `offset` and `spacing`. These removed values will later become the target. That means our network has to learn to recreate the values that were 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 `(M, N, 3)` and numeric datatype, which contains the RGB image data.
- `offset`: A tuple containing 2 `int` values. These two values specify the offset of the first grid point in x and y direction. Please see Figure 1 below for more information.
- `spacing`: A tuple containing 2 `int` values. These two values specify the spacing between two successive grid points in x and y direction. Please see Figure 1 below for more information.

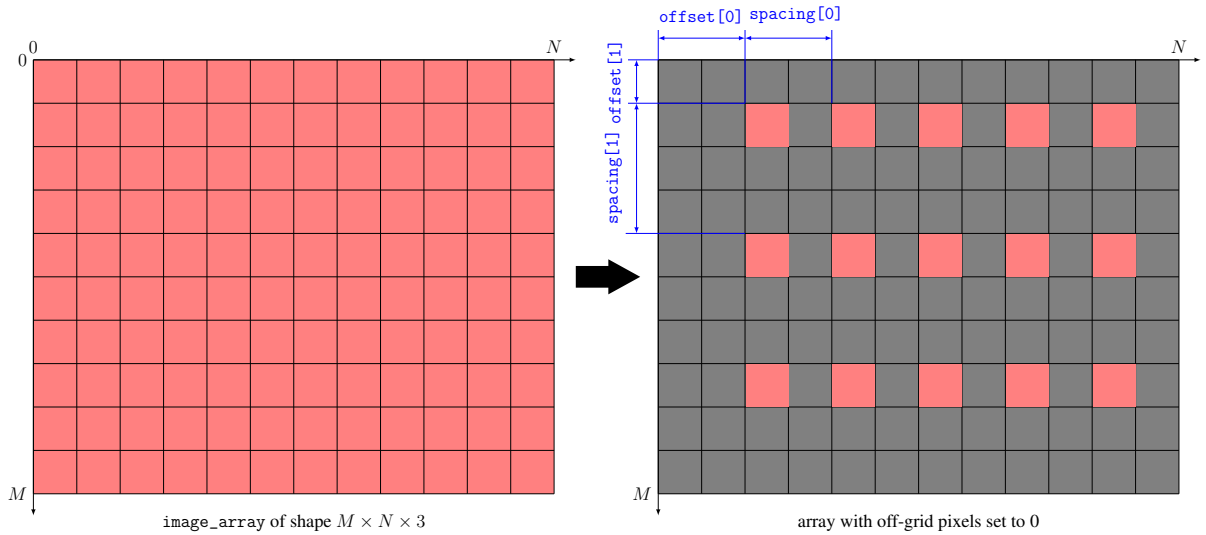
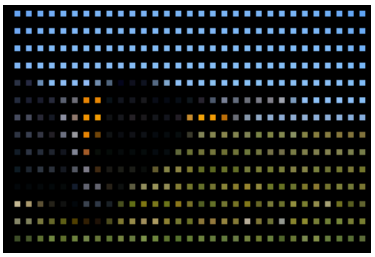


Figure 1: `image_array` with grid specifications.

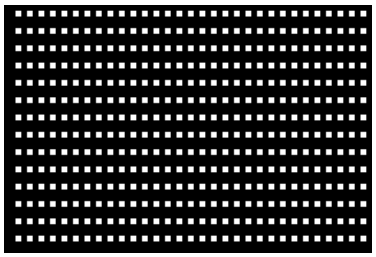
Your function should return a 3-tuple `(input_array, known_array, target_array)`, where an example is also shown in Figure 2:

- `input_array` should be a 3D numpy array of shape `(3, M, N)` and the same datatype as `image_array`. Note that the channel dimension moved to the front (use `np.transpose()` as described in the hints). `input_array` should have the same pixel values as `image_array`, with the exception that the to-be-removed pixel values off the specified grid are set to 0. You may edit the original `image_array` in-place or create a copy.

image_array



input_array



known_array

Figure 2: Example for `image_array`, `input_array`, and `known_array`. `target_array` contains the pixel values in `image_array` that are located outside the grid and set to 0 in `input_array`. Image taken by Ben Tubby – https://commons.wikimedia.org/wiki/File:Falkland_Islands_Penguins_49.jpg

- `known_array` should be a 3D numpy array of same shape and datatype as `input_array`, where pixels on the specified grid should have value 1 and all other unknown pixels have value 0.
- `target_array` should be a 1D numpy array of the same datatype as `image_array`. It should hold the R-, G-, and B-pixel values at the off-grid locations (the pixels that were set to 0 in `input_array`). The order of the R-, G-, and B-pixels in `target_array` should be the same as when using the inverted `known_array` as boolean mask on `image_array` (like `image[known_array < 1]`), which yields a flattened array of all matching R-pixels, followed by all matching G-pixels and all matching B-pixels (the length of the 1D numpy array `target_array` is thus the number of removed pixels times 3).

Your function should raise a `TypeError` exception if:

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

Your function should raise a `NotImplementedError` exception if:

- `image_array` is not a 3D array.
- `image_array` the size of the 3rd dimension is not equal to 3.

Your function should raise a `ValueError` exception if:

- The values in `offset` and `spacing` are not convertible to `int` objects.
- The values in `offset` are smaller than 0 or larger than 32.
- The values in `spacing` are smaller than 2 or larger than 8.
- The number of the remaining known image pixels would be smaller than 144.

Hints

- 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`.
- 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.
- To move dimensions (along with memory) of a numpy array, `np.transpose()` can be used. For instance, the channels of an array of shape `(M, N, C)` can be moved to the front using `np.transpose(array, (2, 0, 1))`, which yields an array of shape `(C, M, N)`.
- 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.

Submission Information

- Due date: 18.05.2022 23:55
- File name: `ex4.py`
- Moodle: https://moodle.jku.at/jku/mod/assign/view.php?id=6719732

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 predict unknown off-grid pixels of an image. The challenge is accessible as *Image Inpainting Challenge 2022* at https://apps.ml.jku.at/challenge. User logins for submissions will be provided on 11.05.2022.

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

- Samples considered in this challenge are RGB images for which a regular grid of pixels of the images is known (all other pixels were set to zero = unknown pixel values).
- Your model should predict by extra- and interpolating the unknown pixel values defined in exercise 4 (`target_array`).
- A subset of the images collected in exercise 1 will be the training set, however, you are free to include more images of your choosing into your own training set. Since we already collected a lot of images, training on the training set itself will be sufficient.
- Download link for training set: https://cloud.ml.jku.at/s/D62jmH8W3bTX7g6

Predict the unknown parts of the test images.

- You will be provided with test images where only the pixel values on a regular grid are available (all remaining pixels were set to zero).
- The test images will have a shape of 100×100 pixels. They have been downscaled from various shapes using `transforms.Resize(size=im_shape)` with `BILINEAR` interpolation (for more information, see the documentation https://pytorch.org/vision/stable/generated/torchvision.transforms.Resize.html).
- The spacing in the test set images will be in the set $\{2, 3, \ldots, 6\}$.
- The offset in the test set images will be at most 8 pixels in each direction.
- The test set images and the specifications of their unknown pixels will be provided as a `pickle` file. This 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),
  - `offsets` (`offset` kwarg as used for exercise 4 for each sample),
  - `spacings` (`spacing` kwarg as used for exercise 4 for each sample),
  - `sample_ids` (sample ID as string for each sample).
- Download link for test set: https://cloud.ml.jku.at/s/oiBb3ZqfWMkqSmB
- Download link for supplements which contain an example test set and example predictions: https://cloud.ml.jku.at/s/DTsJ7F9yrtQFMRj

Upload your predictions of 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 of 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 a ZIP archive.
  - The ZIP archive 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 score of your model will be compared to 2 other models, `ModelA` and `ModelC`, for grading.

- The root mean squared error (RMSE) between the true unknown pixel values and your predictions of 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 each channel of the input image as prediction of the unknown pixel values.
- `ModelB` predicts the unknown pixel values by linear interpolation to the closest neighbor(s). Its performance is better than `ModelA` but worse than `ModelC`.
- `ModelC` uses a CNN consisting of 5 hidden convolutional layers + 1 convolutional output layer, a kernel size of 3 and 64 kernels per CNN layer to predict the unknown pixel values from a preprocessed input.
- 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 `ModelC`, you get 40 points.
- You will also receive bonus points, depending on how much your model beats the score of `ModelC`.
- Points for models with scores between `ModelA` and `ModelC` will be interpolated linearly.

Hints

- **Divide the project into subtasks** and check your program regularly. Example:
  1. Decide which samples you want to use in your training-, validation- or test sets.
  2. Create the data reader (reuse your code from exercise 4).
  3. Create the data loader and stacking function for the minibatches (see code files of Unit 5).
  4. Implement a NN that computes an output given this input (see code files of Unit 6).
  5. Implement the computation of the loss between NN output and target (see code files of Unit 7).
  6. Implement the NN training loop (see code files of Unit 7),
  7. Implement the evaluation of the model performance on a validation set with early stopping (see code files of Unit 7).

  You may want to use the project structure shown in the example project in the code files of Unit 7.
- **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.
- **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 = 100
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 8.

- **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 of Unit 7). 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 non-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 non-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 data subset 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 spacing (see test set specifications). The parameters of `ModelC` 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 reinvent the wheel.** Most of this project can be solved by reusing parts of the code materials from this semester. If you need help, ask in due time (see https://moodle.jku.at/jku/mod/forum/view.php?id=6483985 for help options).

Submission Information

- Due date: 13.07.2022 23:55
- Part 1: Moodle submission:
  - File: A single ZIP archive (`.zip`) containing all code, config, documentation, etc. files of your machine learning project implementation.
  - Link: https://moodle.jku.at/jku/mod/assign/view.php?id=6719733
- Part 2: Challenge server submission:
  - File: A single `pickle` file containing your predictions (list of `np.uint8` numpy arrays) of the provided test set.
  - Link: https://apps.ml.jku.at/challenge (use provided username+password)

Exercise 6 [10 bonus points]

In this bonus exercise, you should create a function `ex6(logits, activation_function, threshold, targets)` that computes scores for the predictions of a NN model in a binary classification task using `PyTorch`.

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

- `logits`: A `torch.Tensor` of shape `(n_samples,)`, floating point datatype (as determined by `torch.is_floating_point`). `logits` contains the raw output of the NN before the application of `activation_function`.
- `activation_function`: The activation function to apply to `logits` using the syntax `nn_output = activation_function(logits)`.
- `threshold`: A `torch.Tensor` of arbitrary datatype, a threshold to decide if a sample is classified as `True` or `False`. If the NN output `nn_output` (see computation via `activation_function`) is greater than or equal to `threshold`, the sample is classified as `True`, otherwise, it is classified as `False`.
- `targets`: A `torch.Tensor` of shape `(n_samples,)`, `torch.bool` datatype. `targets` contains the true labels (=target classes) of the samples. The order of samples is the same as in `logits`.

Your function should apply the `activation_function` and `threshold` to the `logits` to generate the confusion matrix w.r.t. the `targets` and to calculate the following scores: F1-score, accuracy and balanced accuracy. For more details, see https://en.wikipedia.org/wiki/Confusion_matrix.

Your function is only allowed to use `PyTorch` to compute the scores. The computation of scores must be performed using `torch.float64` datatype. You are not allowed to import the `sklearn` module or other modules that provide functions for computing the scores directly.

Your function should raise a `TypeError` exception if

- `logits` is not a `torch.Tensor` of floating point datatype (as determined by `torch.is_floating_point`).
- `threshold` is not a `torch.Tensor`.
- `targets` is not a `torch.Tensor` of datatype `torch.bool`.

Your function should raise a `ValueError` exception if

- the shape of `logits` or `targets` is not `(n_samples,)`.
- `n_samples` is not equal for `logits` and `targets`.
- `targets` does not contain at least one entry with value `False` and at least one entry with value `True`.

Your function should return a tuple of length 4, sequentially containing:

1. the confusion matrix as a nested list `[[TP, FN], [FP, TN]]`
2. the F1-score as Python `float` object (set to 0 in case of a division by 0 error)
3. the accuracy as Python `float` object
4. the balanced accuracy as Python `float` object

Submission Information

- Due date: 13.07.2022 23:55
- File name: `ex6.py`
- Moodle: https://moodle.jku.at/jku/mod/assign/view.php?id=6719734