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 [15 points]

In this exercise you should create a function `ex4(image_array, crop_size, crop_center)` that creates two input arrays and one target array from one input image. For this, your function should crop out (=set values to zero) a part of the image, which will then become the target. Since it could be valuable information for our network to know which part was cropped out and should be restored, we will also prepare an additional input channel that includes information about which pixels are in or outside the cropped-out rectangle.

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.
- `crop_size`: A tuple containing 2 odd `int` values. These two values specify the lengths of the rectangle that should be cropped-out in pixels for the two spatial dimensions `X` and `Y` in that order.
- `crop_center`: A tuple containing 2 `int` values. These two values are the position of the center of the to-be cropped-out rectangle in pixels for the two spatial dimensions `X` and `Y` in that order.

Your function should return a tuple `(image_array, crop_array, target_array)`, where the returned `image_array` is a modified version of the original `image_array` that the function gets as argument.

`image_array` should be modified such that the pixels in the cropped-out rectangle are set to 0, while the rest of the pixels remains unchanged. You may edit the original `image_array` in-place or create a copy.

`crop_array` should be a numpy array of same shape and datatype as `image_array`, containing value 0 for pixels located outside the cropped-out rectangle and 1 for pixels located in the cropped-out rectangle.

`target_array` should be a 2D numpy array of the same datatype as `image_array`, containing the values of the original `image_array` in the cropped-out rectangle.

The to-be cropped-out rectangle is specified via the center of the rectangle `crop_center` and the size of the rectangle in pixels `crop_size`. Theoretically, we could rotate the rectangle or choose other forms to crop out but we will not consider these cases here.

Your function should raise a `ValueError` exception if

- `image_array` is not a 2D numpy array (see hints on how to check if an object is a numpy array instance).
- `crop_size` or `crop_center` do not contain exactly 2 objects. (You do not need to check the datatype of the objects, you can assume them to be integers.)
- The values in `crop_size` are even numbers.
- The minimal distance between the to-be cropped-out rectangle and the border of `image_array` is less than 20 pixels.

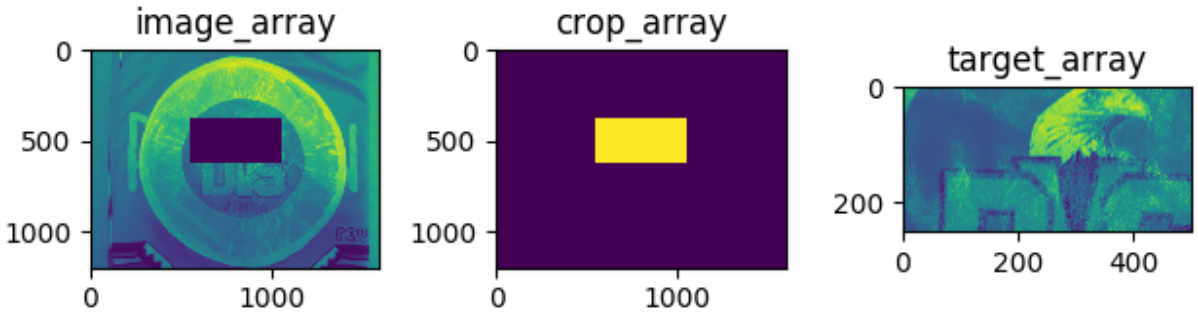

Fig.1: Example for arrays returned by function `ex4()` (plotted using matplotlib) with `crop_size=(251, 501)` and `crop_center=(500, 800)`.

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

Hint: To create `crop_array` with same shape and datatype as `image_array`, you can use the `np.zeros_like()` function. To create the `target_array` you can use slicing to obtain the values from `image_array` (but don't forget to use `np.copy(target_array)` if you modify the `image_array` values in-place afterwards).

Hint: For feeding the input into a NN in exercise 5, you could concatenate the channels of `image_array` and `crop_array`, resulting in an input array of shape `(2, X, Y)` or `(X, Y, 2)`.

Exercise 5 [35 points + 10 bonus points]

Exercise 5 is the ML challenge, where points are determined by the performance of your model. You will have to train a neural network to restore missing parts of an image. The challenge is accessible as *Data imputation challenge 2020* at `https://apps.ml.jku.at/challenge`. User logins for submission will be provided at 15.06.2020.

Challenge specifications:

- Samples considered in the challenge are grayscale images where a part of the image was cropped out, as done in exercise 4.
- Your model should predict (=restore) the cropped-out 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.

Scoring will be performed as follows:

- You will be provided with test set images, where a part of the image was cropped out.
- The images and the center and size of the cropped-out part will be provided as pickle file (format see below).
- The images will have a width and height of at least 70 and at most 100 pixels.
- The width and height of the cropped-out rectangle will be odd integers of at least 5 and at most 21 pixels.
- The border between the cropped-out rectangle and the borders of the image will be at least 20 pixels, as in exercise 4.
- Predictive performance will be measured by the mean squared error. The error per sample will be the mean squared error of target vs. predicted pixels. The final loss of your model will be the mean over the sample errors in the test set.
- You will need to predict the missing pixel values for each sample and submit the predicted values to the challenge server as pickle files (format see below). The loss of your model will be computed on the server.
- The test set will be made available on 24th of June. You will only have 5 attempts to upload submissions to the server. The submission with the best loss will be taken as final. Submissions with wrong format will not be considered in the count of the 5 attempts.

Test set format. The test set will be provided as pickle file containing a dictionary with entries `"images"`, `"crop_sizes"`, and `"crop_centers"`.

- `"images"` will be a list of numpy arrays of shape (X, Y) and datatype `numpy.uint8`, where each numpy array represents one image and X and Y are the spatial dimensions. The images were read from grayscale `.jpg` images into numpy arrays using `PIL` and resized using the interpolation method `PIL.Image.LANCZOS`.

- `"crop_sizes"` will be a list of tuples with 2 elements per tuple of datatype `int`, where each tuple represents the `crop_size` as used in exercise 4.
- `"crop_centers"` will be a list of tuples with 2 elements per tuple of datatype `int`, where each tuple represents the `crop_center` as used in exercise 4.
- As in exercise 4, `crop_size` and `crop_center` are tuples specifying the length and position for dimension X and Y in the respective numpy array.
- All elements in the lists (images, `crop_size`, and `crop_center`) have the same order. That is, element at index 0 in the list in `"images"` is the numpy array containing the pixel values for the first image, element at index 0 in the list in `"crop_sizes"` is the `crop_size` for the first image, and element at index 0 in the list in `"crop_centers"` is the `crop_center` for the first image. Elements at index 1 in the lists would represent the data for the second image, elements at index 2 the third image, and so on.

Upload format. The predictions have to be uploaded to the server using a pickle file containing a list of numpy arrays. The ordering of elements in this list MUST be the same as in the 3 lists `"images"`, `"crop_sizes"`, and `"crop_centers"`. Each numpy array must have the shape of the respective `crop_size`, contain the predicted values of the cropped-out pixels, and be of datatype `numpy.uint8`. The loss of your model will be computed from the uploaded pickle file.

Additionally, you have to upload your code as `.py` or `.zip` file to moodle for plagiarism checks. 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).

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. Once you have found a good hyperparameter setting, you may use the full data for re-training the model in that setting (this is commonly done in ML challenges but requires intuition).

Grading. The loss of your model will be compared to 2 baseline models for grading.

- Model A uses the mean value of the input image as prediction of the cropped-out pixel values.
- Model B uses a CNN consisting of 3 layers (=2 layers + 1 output layer), a kernel size of 7, and 32 kernels per CNN layer to predict the cropped-out pixel values.
- If your model has a higher loss than model A, you get 0 points.
- If your model has a lower loss than model B, you get 35 points. You will also receive bonus points, depending on how much better than model B your model is.
- Points for models with losses between model A and model B will be interpolated linearly.
- Model A and Model B losses will be visible in the leader-board starting with 24th of June.

Deadline. Deadline for the submission is the 1st of September, 23:55. (An option for earlier grading will be provided.)

Hints:

Divide the project into sub-tasks, e.g. (1) creation of data reader, (2) loading a sample and stacking the minibatch, (3) implementing a NN that computes an output given this input, (4) computing the loss between NN output and target, (5) implementing the NN training loop, (6) implementing the evaluation of the model performance on a validation set with early stopping.

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 `PIL.Image.LANCZOS` as interpolation method. 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`).

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 images passed to `collate_fn` and create a zero-tensor of shape `(n_samples, n_feature_channels, max_X, max_Y)` (so that it can hold all input images). 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 `crop_array` (see exercise 4) and feed the resulting tensor as input into the network.

Creating predictions and computing loss: To predict the missing pixel values, you can implement a CNN that creates an output that has the same size as the input (see example project). Then you can use either a boolean mask like `crop_array` or slicing via `crop_size` and `crop_center` to obtain the predicted pixel values in the cropped out rectangle. See file `ex5_example_cropping.py` for an example that uses a boolean mask.

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 semsester.

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.

Exercise 6 [10 bonus points]

In this 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 without using `sklearn`.

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

- `logits`: A `torch.Tensor` of shape `(n_samples,)` and floating point datatype (as decided by `torch.is_floating_point`). `logits` contains the output of the NN before the application of `activation_function`.
- `activation_function`: The activation function to apply to `logits`.
- `threshold`: The threshold to decide if a sample is classified as `True` or `False` as `torch.Tensor` of arbitrary datatype. If the model prediction (the `logits` value after the application of the `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,)` and datatype `torch.bool`. `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` and compute the following scores w.r.t. the `targets`: true positive rate, true negative rate, false positive rate, false negative rate, accuracy, balanced accuracy.

Your function is only allowed to use PyTorch to compute the scores (do not import the `sklearn` module!). Computation of scores must be done using `torch.float64` datatype.

Your function should raise a `TypeError` exception if

- `logits` is not a `torch.Tensor` of floating point datatype (as decided 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 6 containing the values for the true positive rate, true negative rate, false positive rate, false negative rate, accuracy, and balanced accuracy as Python `float` objects in that order. Your function must not print any output to the console.

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!