Image Depixelation Project

This assignment is the machine learning project for depixelating images. There are two submissions that must both be uploaded until the specified due date: The test set predictions to the dedicated challenge server (username = password = `K<8-digit-matr-ID>`), and the project files to Moodle.

To get the test set predictions, you will have to train a machine learning model to predict the original values of pixelated areas within images. A subset of the images collected in Assignment 1 –Exercise 1 (34 635 images in 350 folders) is provided for training (download), the remaining images (6 635 images in 67 folders) serve as test set where only the pixelated versions are available (download).

Exercise 1 – Submission: Test Set Predictions (Challenge Server) 200 Points 50 or 100 Bonus Points

The test set (download) is a `pickle` file containing a dictionary with the following entries:

- `pixelated_images`: A tuple of 6 635 NumPy arrays representing the pixelated images as defined in Assignment 2 – Exercise 2.
- `known_arrays`: A tuple of 6 635 NumPy arrays representing the boolean masks that show the pixelated area within the images as defined in Assignment 2 – Exercise 2.

Each test set image was transformed to a shape of `(64, 64)` using `transforms.Resize(shape=64, interpolation=InterpolationMode.BILINEAR)` followed by `transforms.CenterCrop(size=(64, 64))` and converted to grayscale before the pixelation process (see Assignment 2 – Exercise 2) was applied. In this pixelation process, for each image, the x- and y-coordinates were randomly chosen from the ranges $[0, 64 - \texttt{width}]$ and $[0, 64 - \texttt{height}]$, where `width` and `height` were again randomly chosen from from the ranges $[4, 32]$ and $[4, 32]$. The pixelation block size was randomly chosen from the range $[4, 16]$ (see also Assignment 3 – Exercise 1).

Your task is to predict the true, original values for each of the `pixelated_images` and collect the predictions in a list as follows:

- Each prediction must be a flat 1D NumPy array of data type `np.uint8` containing all predicted pixel values in a flattened view (e.g., such as obtained via `some_image[~known_array]`).
- The order of this list must be the same as the order of the lists in the test set.

The list of predictions must then be serialized to a file using the `serialize` function of the provided `submission_serialization.py` script, and this serialized file must be uploaded to the challenge server. The filename of the serialized file can be arbitrary. The following restrictions apply:

- You only have 5 valid attempts to upload predictions.
- Invalid attempts (e.g., error parsing your predictions file) are not counted.
- Your best attempt will be used as final attempt.

The points are determined based on the model performance, the root-mean-squared error (RMSE), which is compared to two reference models: `IdentityModel` (uses input as prediction, i.e., it does not perform any actual computation) and `BasicCNNModel` (5 layers with 32 kernels of size 3). If your model's RMSE is equal to or higher than `IdentityModel`, you get 0 points. If your model's RMSE is equal to `BasicCNNModel`, you get 200 points. Everything in between the two models is linearly interpolated. If your model's RMSE is lower than `BasicCNNModel`, you get 50 bonus points. There is also a third model (only relevant for bonus points): `AdvancedModel`. If you manage to beat this as well, you instead get 100 bonus points. See the leaderboard for the individual RMSE scores.

Exercise 2 – Submission: ZIP Archive With Project Files (Moodle) 200 Points

In addition to the challenge server submission, you must also upload all of your project files to Moodle. This will be used to check for plagiarism and to verify that your model corresponds to the uploaded predictions. Create a ZIP archive that includes *all* project files, i.e., the entire source code, all configuration files and all documentation files. The filename of the ZIP archive can be arbitrary.

General Project 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 loader and stacking function for the minibatches (see code files of Unit 5).
  3. Implement a neural network (NN) that computes an output given this input (see code files of Unit 6).
  4. Implement the computation of the loss between NN output and target (see code files of Unit 7).
  5. Implement the NN training loop (see code files of Unit 7),
  6. Implement the evaluation of the model performance on a validation set (see code files of Unit 7).

  You can use the project structure shown in the example project in the code files of Unit 7.
- You 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, e.g., as follows:

```
from torchvision import transforms
from PIL import Image

im_shape = 64
resize_transforms = transforms.Compose([
    transforms.Resize(size=im_shape),
    transforms.CenterCrop(size=(im_shape, im_shape)),
])
with Image.open(filename) as image:
    image = resize_transforms(image)
```

  However, if you want to drastically increase your data set size using data augmentation, you can also use random cropping followed by resizing to create more input images (e.g., via `transforms.RandomResizedCrop`). For more details on data augmentation, see Unit 8.
- You will most likely 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. However, if you know that your input already only has a certain shape (see resizing transformation above), you can directly stack.

- It makes sense to feed additional input into the NN. You can concatenate the channels of `pixelated_image` and `known_array` (see Assignment 2 – Exercise 2) 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 to obtain the predicted pixel values.
- If you normalize the NN input, denormalize 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 denormalize the NN output.
- Start with a small data subset to quickly get suitable hyperparameters. Use a small subset (e.g., 30 samples or at the very beginning even just 2 samples) of your training set for debugging. Your model should be able to overfit (=achieve almost perfect performance) on such a small data set.
- 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.
- To show how the predictions file should look like, a debug predictions file is available (download). These predictions are just 0-value NumPy arrays, so do not use them for any other purpose than debugging. Use the `deserialize` function of the provided `submission_serialization.py` script to deserialize the file into its original list of NumPy arrays.
- You do not need to reinvent the wheel. Most of this project can be solved by reusing parts of the code materials and assignments from this semester.
- When uploading to the challenge server, you might not immediately see a result due to the server's task scheduling. Please be patient and check back later. This also means that you should try to upload your prediction in due time to avoid having no immediate feedback when the assignment deadline approaches.
- To put the hardware/compute requirements into perspective, the `BasicCNNModel` was trained on a notebook CPU Intel Core i5-1135G7. The RAM utilization was below 2GB, even when preloading the entire data set into RAM. The training including evaluations took about 2 hours and 30 minutes.