Assignment no. 1

Programming in Python II

Scenario: Preparing data for your ML project

In this assignment you will prepare the dataset for your ML project using Python. This starts with collecting the data. Afterwards you will perform a first quick analysis of the data. As last step, you will apply preprocessing to your dataset.

Exercise 1 [6 points]

In this exercise you will collect data for the dataset. For this, you have to take 100 pictures, e.g. with your cellphone.

The pictures must

- not exceed 850 kB per picture,
- not show humans or parts of humans,
- not show personal/private information (license plates, door bell nameplate, private documents, etc.),
- be unique with a maximum of 5 pictures of the same object,
- be JPG files,
- have a minimum size of 64x64 pixels (otherwise there is no specific or consistent layout required),
- include color information (no grayscale images),
- be taken by you (don't steal images from the internet!), and
- not include watermarks (deactivate watermarks for Huawei phones).

Create a `.zip` archive containing the 100 JPG files. The name of the `.zip` file should be your student ID starting with the "k", as used for your KUSSS login (e.g. `k12345678.zip`). Upload this `.zip` file to our cloud server: `https://cloud.ml.jku.at/s/popqJBCFaNC56xq` (the password for the upload is `PyThOn.2`).

The JPG files from all students will be pooled into one large dataset, which we will use to train our model.

The upload will be open until March 25th, 23:55.

Exercise 2 [17 points]

In this exercise we will clean up the mess that a raw dataset usually is. For this, you should write a Python function that takes 3 keyword arguments as input:

- `input_dir`: The input directory as string in which your function should look for files.
- `output_dir`: The directory as string in which your function should place the output files.
- `logfile`: The file path of the logfile as string.

Your function should scan `input_dir` for files recursively and sort this list of file names. The files should then be processed in the order that they appear in the sorted list of file names. Files should be copied to `output_dir` if they are valid. The name of the copied filename should be `xxxxxx.jpg`, where `xxxxxx` is the serial number. The serial number is an integer starting at 1, zero-padded to 6 digits, and being increased with each file that has been copied. For example the first copied file should be placed in `output_dir` with the filename `000001.jpg`, the second copied file with `000002.jpg`, then `000003.jpg`, and so on.

Files are considered valid if the following rules are met:

1. The file name ends with `.jpg`, `.JPG`, `.JPEG`, or `.jpeg`.
2. The file size is larger than `10kB`.
3. The file can be read as image (i.e. the `PIL`/`pillow` module does not raise an exception when reading the file).
4. The image data does have variance> 0, i.e. there is not just 1 value in the image data.
5. The image data has shape (H, W), with H and W larger or equal to 100 pixels.
6. The same image data has not been copied already.

File names of invalid files should be written to `logfile`. The format of `logfile` should be as follows: Each line should contain the filename of the invalid file, followed by a semicolon, an error code, and a newline character. The error code is an integer with 1 digit, corresponding to the list of rules for file validity. Only one error code per file should be written and the rules should be checked in the order rule 1, 2, 3, 4, 5, 6. Each file name should only contain the relative file path starting from `input_dir`.

The function should return the number of valid files that were copied.

Hint: You can store the hashes of all copied files to check if the current file has already been copied.
Hint: `os.path.getsize()` will report the size of a file.
Hint: `input_dir` might be an absolute or relative path. In order to get the relative file path starting from `input_dir`, you can use `os.path.abspath(input_dir)` to get the absolute path of `input_dir`.

Exercise 3 [17 points]

In this exercise we will take a look at the mean values and standard deviations of our image files. We will also normalize each image file.

To encapsulate these two aspects into one tool, you should write a class `ImageNormalizer`.

The `__init__` method of this class should:

- Take one keyword argument, `input_dir`, which is the path to an input directory as string.
- Scan the input directory for files ending in `.jpg` and sort this list of file names.
- Store this list of file names in attribute `self.file_names`.

`ImageNormalizer` should have a method `get_stats()`, which:

- Takes no arguments.
- Computes the mean values and standard deviations of all images in the list `self.file_names` and stores them in two numpy arrays of datatype `np.float64`.
- Returns a tuple `(means, stds)`, where `means` is the numpy array containing the mean values for all files and `stds` is the numpy array containing the standard deviations for all files.

Example: If `self.file_names` has length 15, that means that the arrays `means` and `stds` should each be of shape `(15,)`.

`ImageNormalizer` should furthermore have a method `get_images()`, which

- Takes no arguments.
- Returns an iterator that yields the images in `self.file_names` one by one.

These images should be returned as 2D numpy arrays of type `np.float32` and should furthermore be normalized to mean$= 0$ and variance$= 1$.

You can assume that all files with file names ending in `.jpg` are valid image files.

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!