Assignment 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 (exercise 1). Afterwards you will perform a cleanup of the data (exercise 2). As last step, you will implement a first analysis and preprocessing of your dataset (exercise 3).

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

Exercise 1 [5 points]

In this exercise, you will collect images for the dataset. For this, you have to take 100 pictures, e.g., with your cellphone. You can choose which scenes/objects you take pictures of, as long as you adhere to the following rules: The pictures must

- not exceed 250kB (=250 000 Bytes) per picture (a script for decreasing the size of an image is provided in the supplements),
- 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/scene,
- be JPEG files (https://en.wikipedia.org/wiki/JPEG),
- have a minimum size of 96×96 pixels (other than that, there is no specific or consistent layout required),
- include color information (no grayscale images),
- be taken by you (do not use images from the Internet), and
- not include watermarks (make sure to deactivate watermarks if they are enabled).

Submission Information

Create a ZIP archive (https://en.wikipedia.org/wiki/ZIP_(file_format)) containing the 100 JPEG files. The name of the `.zip` file should be your student ID starting with `k` and followed by the 8-digit ID number (if necessary, include leading zeros, e.g., `k01234567.zip`).

- Due date: 06.04.2022 23:55
- File name: `k<8-digit-student-ID>.zip`
- Cloud server: https://cloud.ml.jku.at/s/9K4DWBqdwopnkka (you need the following password: `p2_data@22`). Please keep in mind that the server might be slow when a lot of students upload at the same time, so make sure to upload in time.

The JPEG files from all students will be pooled into one large dataset, which we will use to train our model. This dataset will be made available to all students in this course. Image metadata (GPS information, device type, etc.) will be removed before the dataset is made available.

Exercise 2 [15 points]

In this exercise, we will clean up the raw dataset. For this, you should write a Python function `validate_images` (in the file `ex2.py`) that takes three keyword arguments as input:

- `input_dir` (string): The input directory where your function should look for files.
- `output_dir` (string): The directory where your function should place the output files.
- `log_file` (string): The path of the log file.
- `formatter` (string, optional): Optional format string to use when writing the output files. It expects a single number that is used to apply the format string on. The result is the base name of the file, without any extension.

If `output_dir` or the directory containing `log_file` does not exist, your function should create them.

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. Create a manual copy based on raw pixel data to avoid copying image metadata. 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 does not exceed 250kB (=250 000 Bytes).
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 has a shape of `(H, W, 3)` with `H` (height) and `W` (width) larger than or equal to 96 pixels. The three channels must be in the order `RGB` (red, green, blue).
5. The image data has a variance larger than 0, i.e., there is not just one common RGB pixel in the image data.
6. The same image data has not been copied already.

The base name (without any extension) of the copied file should be as defined by the format string `formatter`, which expects a single number that is used to apply the format string on. If no format string is specified, then simply the number is used as base name. The number must be an integer starting at 0, and it is incremented by 1 for every file that has been copied. The extension of every copied file must be `.jpg`. Example:

- Input files:
  - `cat.jpeg` (valid)
  - `dog2.png` (invalid)
  - `tree1.jpg` (valid)
  - `tree2.JPG` (valid)

- Format string: `"06d"`
- Output files:
  - `000000.jpg` (from `cat.jpeg`)
  - `000001.jpg` (from `tree1.jpg`)
  - `000002.jpg` (from `tree2.JPG`)

File names of invalid files should be written to `log_file`. The format of `log_file` should be as follows: Each line should contain the file name of the invalid file, followed by a comma, 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 from above (i.e., there are a total of 6 error codes). Only one error code per file should be written, and the rules should be checked in the ascending order 1, 2, 3, 4, 5, 6. Each file name should only contain the relative file path starting from `input_dir` (this means `input_dir` is not part of the relative path anymore).

The function should return the number of valid files that were copied as integer. In the example above, 3 should be returned.

Hints

- You can store the hashes of all copied files to check if the current file has already been copied.
- `os.path.getsize()` will report the size of a file.
- Loading an image from a file can be done with `PIL.Image.open()`.
- You can use the attribute `my_image.mode` to check the mode of an image (e.g., `"RGB"`).
- You can use `im = PIL.Image.fromarray()` and `im.save()` to create a manual copy from a numpy array which holds raw image pixel data.
- `input_dir` might be an absolute or relative path. You can use `os.path.abspath()` to get the absolute path of `input_dir`.

Submission Information

- Due date: 27.04.2022 23:55
- File name: `ex2.py`
- Moodle: https://moodle.jku.at/jku/mod/assign/view.php?id=6559794

Exercise 3 [15 points]

In this exercise, we will load and standardize (zero mean, unit variance) each image by the global mean and standard deviation of each color channel. For this, you should write a class `ImageStandardizer` (in the file `ex3.py`). This class should provide three methods: `__init__`, `analyze_images` and `get_standardized_images`. You may add more methods/attributes to the class, but these three methods are required with their functionalities as described below.

The `__init__` method of this class should:

- Take one keyword argument `input_dir` (string), which is the path to an input directory. This can be an absolute or relative path.
- Scan this input directory recursively for files ending in `.jpg`.
- Raise a `ValueError` if there are no `.jpg` files.
- Transform all paths to absolute paths and sort them alphabetically in ascending order.
- Store the sorted absolute file paths in an attribute `self.files`.
- Create an attribute `self.mean` with value `None`.
- Create an attribute `self.std` with value `None`.

The `analyze_images` method of this class should:

- Take no additional arguments.
- Compute the means and standard deviations for each color channel of all images in the list `self.files`. Each mean and standard deviation will thus have three entries: one for the red (R), one for the green (G) and one for the blue channel (B).
- Store the average over these RGB means of all images in the attribute `self.mean` (global RGB mean). This value should be a 1D numpy array of datatype `np.float64` and with shape `(3,)`.
- Store the average over these RGB standard deviations of all images in the attribute `self.std` (global RGB standard deviation). This value should be a 1D numpy array of datatype `np.float64` and with shape `(3,)`.
- Return the tuple `(self.mean, self.std)`.

The `get_standardized_images` method of this class should:

- Take no additional arguments.
- Raise a `ValueError` if `self.mean` or `self.std` is `None`.
- Yield the pixel data of each image (generator function), i.e., the raw numpy data with shape `(H, W, 3)`, in the order that the image files appear in `self.files`. For this, in each yield-iteration, the method should:

  - Load the image and store the image data (pixels) in a 3D numpy array of datatype `np.float32`.
  - Standardize the image data using the global RGB mean and standard deviation in `self.mean` and `self.std`. To do this, subtract the corresponding `self.mean` from the pixel values and subsequently divide the pixel values by `self.std` for each color channel.
  - Yield the standardized image data as 3D numpy array of datatype `np.float32`.
- Note: Make sure to load the image data one by one in the yield iteration rather than loading the whole data once at the beginning and then yielding individual elements afterwards. (We are pretending that the complete dataset is too large to be loaded into the RAM at once.)

Hints

- You can assume that all files with file names ending in `.jpg` are valid RGB images.
- Small deviations due to float datatype are to be expected and allowed. Make sure to use the specified datatype for computations, or the deviations will be too large.
- Where possible, perform the array operations in-place for less RAM consumption and faster processing. Also try to use vectorized operations (e.g, do not split up the mean and standard deviation calculations for each color channel separately).
- How exactly you design the solution to this exercise is up to you, as long as it fulfills the requirements stated in the exercise text.

Submission Information

- Due date: 04.05.2022 23:55
- File name: `ex3.py`
- Moodle: https://moodle.jku.at/jku/mod/assign/view.php?id=6559795