Assignment 3: Python for the gram

Contents

updates

- Add links and references for Cython pure-Python mode and typed memoryviews
- Add hints on importing the Cython files directly. The skeleton has been updated to provide better errors here.
- Add more hints, details, and links on numba.

General information

Total points (IN3110): 30 (+ 3 bonus points)

Total points (IN4110): 40 (+ 3 bonus points)

Your solution to this mandatory assignment needs to be committed to the directory

Skip to main content

information on how to install and use the `in3110_instapy` package.

Furthermore, your code needs to be well commented and documented. Functions should have a docstring in e.g. Google style, explaining what the function does, how it should be used, the parameters and return value(s) (including types). We expect your code to be well formatted and readable. Coding style and documentation will be part of the point evaluation for **all tasks** in this assignment. Many provided functions will already have this information, but if you add any or modify specifications, make sure the docstrings reflect your code.

About the assignment

In this assignment, you will make a python package for turning your image of choice into a dramatic grayscale or nostalgic sepia image.

We provide a code skeleton for the package. At this point, you should familiarize yourself with the files and functions in the skeleton.

Download and extract the skeleton to your `assignment3` folder.

We have structured the assignment as follows: First, you will make a python package from the code skeleton. Next, you will implement image filters using pure python, numpy, and numba (and Cython for 4110). The runtimes of each implementation should differ; you will make a profiling report to compare their runtimes. Finally, you will create a command-line interface for your package.

You are free to perform the tasks in a different order if you so prefer. The code skeleton contains functions with comments added that suggest how to solve the task. You can implement a task differently if you prefer, but please do not change the function or file names (additional functions and files are always okay).

Python packages

> **Task 1 (1 point)** Make the `in3110_instapy` package pip-installable.

Hint: Complete the file `pyproject.toml`. Documentation on pyproject.toml can be found in the setuptools documentation.

You should at least fill out:

Skip to main content

- the package name (typically matches the `import` name of the package)
- description
- load the readme (to fill out at the end)
- specify dependencies used in your package. We are going to use at least `numpy`, `numba`, `pillow`, and `line-profiler`. You can add more, if you choose to use them.

Once the package is pip installable you can install it using

```
python3 -m pip install .
```

> **Note:** if you encounter weird output like `Successfully built UNKNOWN`, this could be due to out-of-date setuptools and/or a bug in pip's build isolation. First, make sure to update pip:

> ```
> python3 -m pip install --upgrade setuptools pip # you may want to add --user
> ```

> Then, if that still doesn't work you may need to add `--no-build-isolation` to your pip install command:
>
> ```
> python3 -m pip install --no-build-isolation .
> ```

If this succeeded, you should be able to pass the `package` tests (pytest can be installed with `pip install pytest`):

```
python3 -m pytest -v test/test_package.py
```

Installing the package ensures that its functions and modules are added to the python path, allowing you to run:

```
from in3110_instapy import get_filter
```

Skip to main content

proceeding, you should fix this issue. **Hint:** We want you to pass a specific flag to make the package install **editable**.

Image filtering

An image can be represented in Python as a 3-dimensional array `(H, W, C)`, with `H, W` being the height and width of the image, respectively, and `C` referring to the number of color channels. We will represent an image as an array.

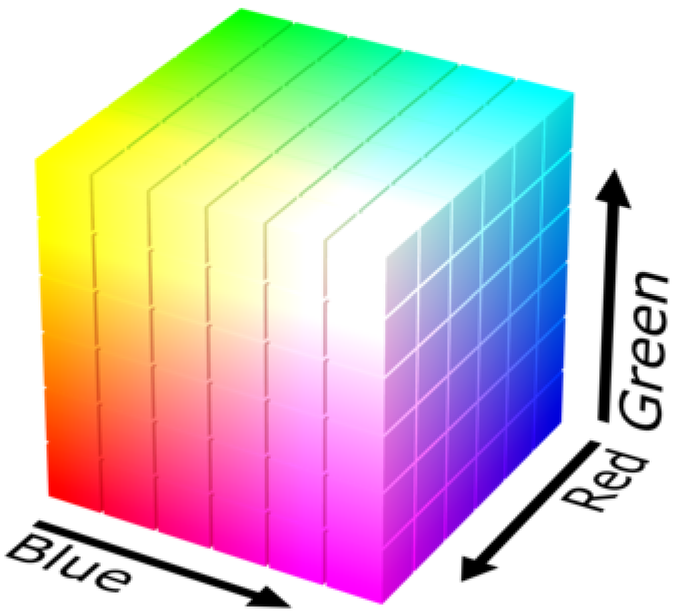
To load images into `numpy` arrays we will use Pillow. Both are common packages that can be installed by conda or pip, and part of the default Anaconda installation. They should be installed automatically as dependencies of your package.

An image stored in `filename` can be loaded to a `numpy` array as

```
import numpy as np
from PIL import Image

filename = "rain.jpg"
pixels = np.asarray(Image.open(filename))
```

`pixels` is then a 3-dimensional `numpy` array with channels $C = 3$, in the order red, green, blue (RGB)



Skip to main content

```
pixels = np.asarray(image)
```

and you can convert back to a PIL image at any time with `Image.fromarray`:

```
image = Image.fromarray(pixels)
```

A handy utility for viewing your images is `Image.display`, so you can preview the results of your filters.

We have provided utilities for these read/write/display actions in `in3110_instapy.io`.

Grayscale Filter

We will be implementing filters in the files `in3110_instapy/{implementation}_filter.py`. The code skeleton has all the functions already defined, you only need to provide the implementation.

In the most simplistic approach the conversion to gray-scale takes the weighted sum of the red, green, and blue values, which determines the gray value. Our approach will take into account that human perception is most sensitive to green and least sensitive to blue. Therefore we apply 0.21, 0.72, and 0.07 as weights for the red, green, and blue channels, respectively. The weighting has to be applied to each pixel.

Images are loaded via `in3110_instapy.io.read_image` as (unsigned 8-bit) integers (`np.uint8`). However, after applying the grayscale function your image likely contains floating point numbers. For PIL to understand your converted image you should convert the values back to integers. This can be done as follows

```
grayscale_array = grayscale_array.astype("uint8")
```

Once the grayscale image is converted to the correct type, you can save it to a file using `Image.save`:

```
image = Image.fromarray(grayscale_array)
image.save("rain_grayscale")
```

For convenience, we have provided functions in `in3110_instapy.io` to do these two steps,

Skip to main content

```
from in3110_instapy.io

io.write_image(grayscale_array, "rain_grayscale.jpg")
```

Pure python implementation

> **Task 2 (2 points)** Implement the function `python_color2gray` in `in3110_instapy/python_filters.py` which returns the grayscale version of an image.

Make sure to write your own *pure Python* implementation using for-loops (no usage of `numpy` and vectorization). You should use `numpy` only for storing the image. All computations should be done with *pure Python*.

The computation time could grow very large in this task when using images with higher resolution, size or width. Consider using a small image for testing during development, or resize the provided image using Pillow. The following code will decrease the resolution of the image by a factor of two:

```
im = Image.open("rain.jpg")
resized = im.resize((im.width // 2, im.height // 2))
pixels = np.asarray(resized)
```

`numpy` Implementation

> **Task 3 (2 points)** Implement the function `numpy_color2gray` in `in3110_instapy/numpy_filters.py` that returns the grayscale version of an image, now using numpy to vectorize the computationally expensive operations.

To be more specific, try to replace your for-loops with `numpy` slicing. Compare the runtime on some input of your choosing. Does this implementation seem faster?

For the sake of clarity, you should not use for example `skimage.color.rgb2gray()` or `PIL.Image.convert()`. Instead, convert the function you wrote in `python_color2gray` to a vectorized version.

Numba Implementation

> **Task 4 (2 points)** Implement `numba_color2gray` in `in3110_instapy/numba_filters.py`

Skip to main content

Note

Your numba implementation may be deceptively simple. That's okay! The performance may be surprising.

Cython Implementation (IN4110 only)

> **Task 5 (2 points, IN4110 only)** Implement `cython_color2gray` in `in3110_instapy/cython_filters.py` similar to that in Task 2, now using Cython to speed things up.

Once you are working on Cython, you will need to set `use_cython = True` in `setup.py`, and (if you haven't already) uncomment the lines adding cython to your build requirements in pyproject.toml. We will be using Cython's pure Python mode, and it will be relevant to read about typed memory views.

Note

The Cython .py file should never be imported as-is by Python. It is a source file that Cython compiles (to a `.so` file, usually), which is what actually gets imported. If you see errors related to importing from `cython.cimports`, this means your source .py file was imported instead of the compiled module. Likely because:

1. the module was not compiled, or
2. the module was compiled, but the location being imported is the original source (e.g. `pip install .` instead of `pip install --editable .`).

Make sure that you have compiled the Cython extension and that the compiled version is what is being imported (check the imported `in3110_instapy.__file__`).

If you see *warnings* from e.g. a linter in your editor, this is because the .py file is not meant to be imported as-is, and can be safely ignored. You can put `# noqa` on the end of the `cimports` line to tell the linter to ignore the line.

Note

To accept read-only views, a Cython typed memoryview argument must be annotated with `const`. The code skeleton has declared a `const uint8_t` type as `const_uint8_t` which you may find useful for this

Skip to main content

Because Cython files require compilation, your changes won't be reflected automatically. To rebuild your Cython files after you change them:

```
python3 setup.py build_ext --inplace
```

or re-running

```
python3 -m pip install .
```

after each change.

Hint: Adding `annotate=True` to `cythonize` will produce `cython_filters.html`, showing you more information about the code Cython is generating. This will help you figure out how to optimize your implementation.

Profiling reports

Hopefully, you will have experienced that each implementation of the grayscale filter is faster than the previous one. We now ask you to formalize this finding by producing a time profiling report.

> **Task 6 (3 points)** Complete the function `make_reports` in `in3110_instapy/timing.py` so that it makes a profiling report of the runtimes of the pure python, numpy and numba implementations of the grayscale filter (**IN4110:** include cython **(1 point, IN4110 only)**).

The report should:

- Show the image being used, including its dimensions
- give the runtimes of each implementation and compare them against each other.

An example output:

```
Timing performed using test/rain.jpg: 600x400

Reference (pure Python) filter time color2gray: 0.855s (calls=3)
Timing: numpy color2gray: 0.00189s (speedup=451.95x)
Timing: cython color2gray: 0.000818s (speedup=1044.92x)
Timing: numba color2gray: 0.0432s (speedup=19.78x)
```

Skip to main content

```
Timing: numpy color2sepia: 0.00693s (speedup=333.31x)
Timing: cython color2sepia: 0.00128s (speedup=1807.72x)
Timing: numba color2sepia: 0.00206s (speedup=1120.90x)
```

Save this report in `timing-report.txt`.

Are any implementations not as fast as you expect? Try improving their performance and running again.

Note

Some questions to think about on numba performance. Is your numba implementation surprisingly slow? Think about *how* numba makes things fast (see also: numba docs on measuring performance). Do the results change if you increase the number of samples? What about running the filter function once before measuring? Is excluding the first run a fair comparison for our command-line tool or not?

Sepia Filter - Add Vintage Style to your Images

As `in3110_instapy` is a python package, it makes sense that it should implement at least two filters. A sepia filter will add a nice touch to your images. To display a source image in sepia we need to average the value of all color channels and replace the resulting value with sepia color.

Here is the sepia filter matrix in RGB order you will be using in this task. You can multiply each color value in the corresponding channel of a pixel with the RGB ordered matrix given here:

```
sepia_matrix = [
    [ 0.393, 0.769, 0.189],
    [ 0.349, 0.686, 0.168],
    [ 0.272, 0.534, 0.131],
]
```

Each of the rows represents the weights for the colors RGB in their respective channels.

Because the input image is read as an array of unsigned 8-bit integers (uint8), adding such values will cause an overflow when the sum exceeds 255. To combat such overflows, one can

Skip to main content

Hint: Setting pixels to `min(255, pixel_value)` results in a truncation of the pixel values at 255 instead of keeping the ratios between the pixels in the image at the correct value relative to each other. The alternative would be to scale all pixel values in the image down so that the maximum value is 255. If you want to implement it with the scaling, you are free to do so. However, you do not have to.

> **Task 7 (9 points)** Implement a sepia filter using pure python (3 points), numpy (3 points) and numba (3 points).

> **Task 8: (3 points, IN4110 only)** Implement the sepia filter using cython.

Again, you are not supposed to use any library function that applies the sepia filter for you.

Unit tests

> **Task 9 (5 points)**: Write unit tests in `test_python.py`, `test_numpy.py` and `test_numba.py` checking the grayscale and sepia filters provided in the package.

> **Task 10 (1 point, IN4110 only)**: Write unit tests in `test_cython.py` checking the cython implementation of the grayscale and sepia filters.

The tests should perform a sanity check on the return values, checking that the filter functions return numpy arrays with the expected shape and dtype. Further, they should check that the filters have been applied correctly to the pixels; we leave the details up to you. You do not need to check that all the pixels have been filtered correctly. Instead, you can check a few selected pixels.

You should verify that your different filter functions produce the same results as each other.

Hint: numpy has utilities allclose and assert_allclose that can help comparing arrays, accepting small numerical variations.

You can check that your tests pass by using the testing framework `pytest`.

Skip to main content

Command-line interface

> **Task 11 (5 points)** Complete the file `in3110_instapy/cli.py` so that it implements a command-line interface where the user can pass arguments to the `in3110_instapy` filter functions. Modify the `pyproject.toml` file to register an entrypoint script.

The command-line interface should use `ArgumentParser` from the standard library module `argparse`. The design of this is up to you, but it should provide instructions by calling it with a `--help` flag, and it should be possible to specify the input and output image filename (for example as passed arguments). It should also be possible to switch between your 3 implementations with a command-line argument.

Modify your `pyproject.toml` file to add a `script`, called `instapy`. It should call the function `main` in the module `in3110_instapy.cli`. Refer again to pyproject.toml docs.

You should have a user-friendly implementation that can be called as either

```
# option 1
python3 -m in3110_instapy <arguments>
# or option 2
instapy <arguments>
```

We prepared a list of **expected flags** shown below. You are free to create additional flags that add useful functionality.

These are flags that we expect to be implemented:

```
positional arguments:
  file                  The filename to apply filter to

options:
  -h, --help            show this help message and exit
  -o OUT, --out OUT     The output filename
  -g, --gray            Select gray filter
  -se, --sepia          Select sepia filter
  -sc SCALE, --scale SCALE
                        Scale factor to resize image
  -i {python,numba,numpy,cython}, --implementation {python,numba,numpy,cython}
                        The implementation
```

Skip to main content

Complete the README

Excellent work, by now you have made a python package!

> **Task 12 (1 point)** Write a `README.md` file containing
>
> - the package name as title (`h1`)
> - summary of what your package does
> - full instructions on how to install it
> - instructions on how to run your package

The readme should be loaded into your pyproject.toml.

Tunable Sepia Filter (BONUS)

Bring your sepia filter to the next level by making it tunable! Allow the user to adapt the level of sepia they want to apply.

> **Task 13 (1 bonus point)** Extend `numpy_color2sepia`, where the user can pass in `k` from 0-1 in order to define 0-100% Sepia effect.

When `k=0`, the result should be the original image. When `k=1`, the result should be the same as the first implementation.

Hint The `k=0` matrix is the identity matrix:

```
[
    [1, 0, 0],
    [0, 1, 0],
    [0, 0, 1],
]
```

Skip to main content

Runtime Tracking Functionality in `in3110_instapy`

> **Task 14 (2 bonus points)** Implement an additional flag `-r, --runtime` that allows the user to track the average runtime spent on the task with the chosen implementation.

The user should receive the average time over 3 runs printed to the terminal, e.g

```
print(f"Average time over 3 runs: {runtime}s")
```

More profiling (IN4110 only)

> **Task 15 (3 points, IN4110 only)** Use profiling to understand the difference in performance between the different implementations.

Here we will use two profilers:

- The built-in cProfile, and
- line-profiler, installable with: `pip install line-profiler`

Hint: you will likely want to use profiler.runcall or profiler.runctx (both cProfile and line-profiler provide these methods).

We will be working in `in3110_instapy/profiling.py`.

1. implement the function `profile_with_cprofile` to profile a given filter function with cProfile, and print the top 10 calls, by cumulative time
2. implement the function `profile_with_line_profiler` to profile a given filter function with line_profiler and print the line profile
3. implement the function `run_profiles` to profile all of our implementations using one of the two above `profile_with` functions for comparison.
4. In the file `profile-report.md`, paste the output of the two different runs of `run_profiles` (now executable as `python -m in3110_instapy.profiling`), and answer the questions based on the profile reports

Skip to main content

Turning it in

Remember: turn the assignment in by pushing to your GitHub repo (`IN3110/in3110-yourname`) by **23:59 CEST, Friday 6. October**. **Committing is not pushing!**, and **Do not push to your fork**.

Previous
Assignment 2: Automating the boring stuff

Next
Assignment 4: Web scraping