Assignment 01 - Welcome and getting started with image processing

Please refer to MyCourses/Gradescope for assignment deadlines.

The goal of this assignment is to get everything set up and to get you familiar with the tools we will be using in this course. Topics covered include:

- Setting up your development environment
- Some basic image loading and manipulation in numpy and OpenCV
- How to check your code and how to submit to Gradescope

Part 0: One-time setup of your machine

Your instructor uses PyCharm, but you are free to use any IDE you like. These instructions will be specific to PyCharm, but if you use VS Code, things will be very similar. If you use a different IDE, you will need to figure out how to do these things on your own. Coding in a text editor without any IDE functionality is strongly discouraged.

Step 1: Make sure you have Python 3.10 or later installed on your machine. Follow instructions here if you need help. If you are on a Mac, I recommend using Homebrew, then running

```
brew install python@3.10 # or 3.11 or 3.12 or whatever
```

Step 2: Download and install your IDE of choice. If you choose PyCharm, I **strongly** encourage you to get the Professional version, which you can get for free as a student (note that their terms of service then prohibit you from using it for commercial purposes). You can do this via the JetBrains site, or via GitHub's education pack. The latter may make it easier to get Copilot access as well, but you will have to explore options on your own.

Step 3 (optional but encouraged): Enable GitHub copilot (follow instructions online for this).

Step 4 (optional but encouraged): If using PyCharm, install the OpenCV image viewer plugin. This will give you the very useful ability to inspect images while paused in a debugger.

Step 5 (optional but encouraged): Open the preferences of your IDE and configure things (fonts, highlighting, code formatting, keyboard shortcuts, etc.) to your liking. For example, I used to use Sublime Text before switching to PyCharm, so I set up my PyCharm to use Sublime Text keybindings.

Part 1: Project setup and workflow

This part of the assignment will walk you through the setup and steps you'll need to follow for each assignment. The instructions here are specific to PyCharm and Unix/Linux systems. If you are using a different IDE or operating system, you will need to figure out how to do these things on your own.

Step 1: Download and unzip the assignment. If you're currently reading the `README.pdf` file, you've already done this!

Step 2: Create a PyCharm project for this assignment and set up a new virtual environment for this assignment. We've provided some command line tools if you prefer to do the environment setup manually.

1. Option 1: set up the environment in the terminal:

   ```
   cd /path/to/assignment
   make setup
   ```

   This will create a virtual environment in the `venv` directory and install all the dependencies listed in `requirements.txt`. You can then activate the virtual environment in a terminal with

   ```
   source venv/bin/activate
   ```

   This will be a required step for running any of the other command-line checks or tools.

   Once you've manually created the virtualenv this way, you can open a new pycharm project and tell it to use an "existing virtualenv" as the interpreter. You'll then need to point it to the `venv/bin/python` executable in the project directory.

   After creating the project in this way, you should double-check that the `venv` directory is marked as "excluded" in the project settings (right-click on the 'venv' directory and choose "mark as > excluded"). This will prevent PyCharm from indexing the virtual environment, which can cause problems / slow things down.

2. Option 2: set up the project/environment from inside PyCharm:

   First, open the project directory in PyCharm. When it asks to set up the project interpreter, select the option to create a new virtual environment. It will suggest something like `venv` or `.venv` as the place where it will store things. It doesn't matter what directory name you choose. Make sure you set the "base interpreter" for the virtualenv to be the Python 3.10 (or higher) executable on your system that you installed in Part 0 above.

   After opening the project, there will be a notification that there are dependencies named in requirements.txt that are not installed. Click the "install requirements" button to install them.

Step 3: Workflow

We've provided a `Makefile` with some useful commands for running checks and tests. We haven't yet figured out how to get all of it to work from inside PyCharm (let us know if you do!), so for now you'll need to run most of these commands from a terminal. To do so, you'll need to do:

```
cd /path/to/assignment
source venv/bin/activate
```

This will activate the virtual environment. From here, you can run any of the following commands:

- `make lint`: runs the `flake8` linter on your code. Any errors or warnings that show up here will directly affect your grade. You can also configure PyCharm to run `flake8` automatically in the settings to give you live feedback on style and probable errors. **Note:** we are allowing you to adapt your own style guide. If there are any style "rules" that flake8 enforces but you want to do differently, you may make a change to the `tox.ini` file to disable certain kinds of checks. We are allowing this flexibility so that you must choose *a* code style not necessarily *our* code style. However, excessive deviations from the default style will be penalized (i.e. you don't get to just tell flake8 to ignore ALL errors. Be thoughtful and deliberate about which kinds of errors you care about and want to ignore).
- `make format`: runs the `black` code formatter on your code. This will automatically fix most style issues, but it will not fix all of them. You should still run `make lint` to check for anything that `black` missed. Black can also be configured to run inside of PyCharm either on save or with a particular keyboard shortcut. Black does not allow for much customization, but if you like, you can put some configuration options in the `pyproject.toml` file.
- `make submission.zip`: this is the command you should run to generate the zip file that you will upload to Gradescope. Note that this includes a built-in check that `make lint` has no errors and that you've filled out the `collaboration-disclosure.txt` file.

In terms of workflow, we recommend that you upload a `submission.zip` file to gradescope as early as possible. This will run a set of tests on your code and give you feedback on how you're doing. You can then make changes to your code and re-upload the `submission.zip` file as many times as you like. You can also run the `make lint` and `make format` commands as many times as you like to check your code and fix any issues.

Part 2: Image handling

This is the actual coding assignment. Open the `src/image_handling.py` file. Everywhere you see `raise NotImplementedError("your code here")` is a block of code that you need to write. The docstring of each function tells you what you need to do. When you upload your submission, each of the functions you write will be subjected to a battery of tests on Gradescope and you'll get some indirect feedback about what is working and what isn't. Beyond that, you'll need to figure out how to fix things yourself.

To help you visualize the expected outputs, here are some examples of what the functions should do, using the provided `bouquet.png` image as a reference:

Figure 1: Expected output of `crop(image, 373, 1424, 200, 100)`

Figure 2: Expected output of `crop(image, -50, 500, 200, 200)`

Figure 3: Expected output of `scale_by_half_using_numpy(image)`

Figure 4: Expected output of `horizontal_mirror_image(image)`

Figure 5: Expected output of `rotate_counterclockwise_90(image)`

Figure 6: Expected output of `swap_b_r(image)`

Figure 7: Expected output of `scale_saturation(image, 2.0)`

Figure 8: Expected output of `scale_saturation(image, 0.0)`

Figure 9: Expected output of `grayscale(image)`

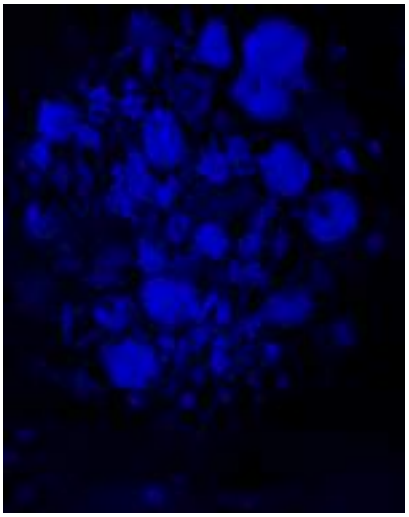

Figure 10: Expected output of `blues(image)`

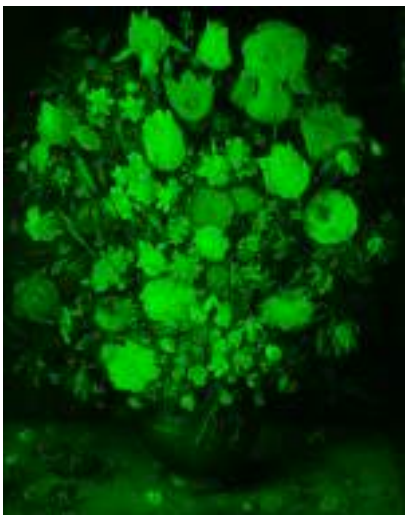

Figure 11: Expected output of `greens(image)`

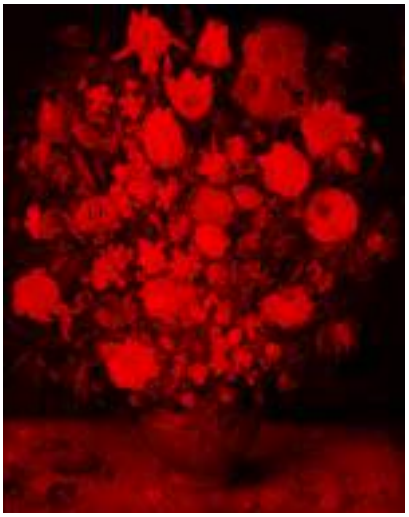

Figure 12: Expected output of `reds(image)`

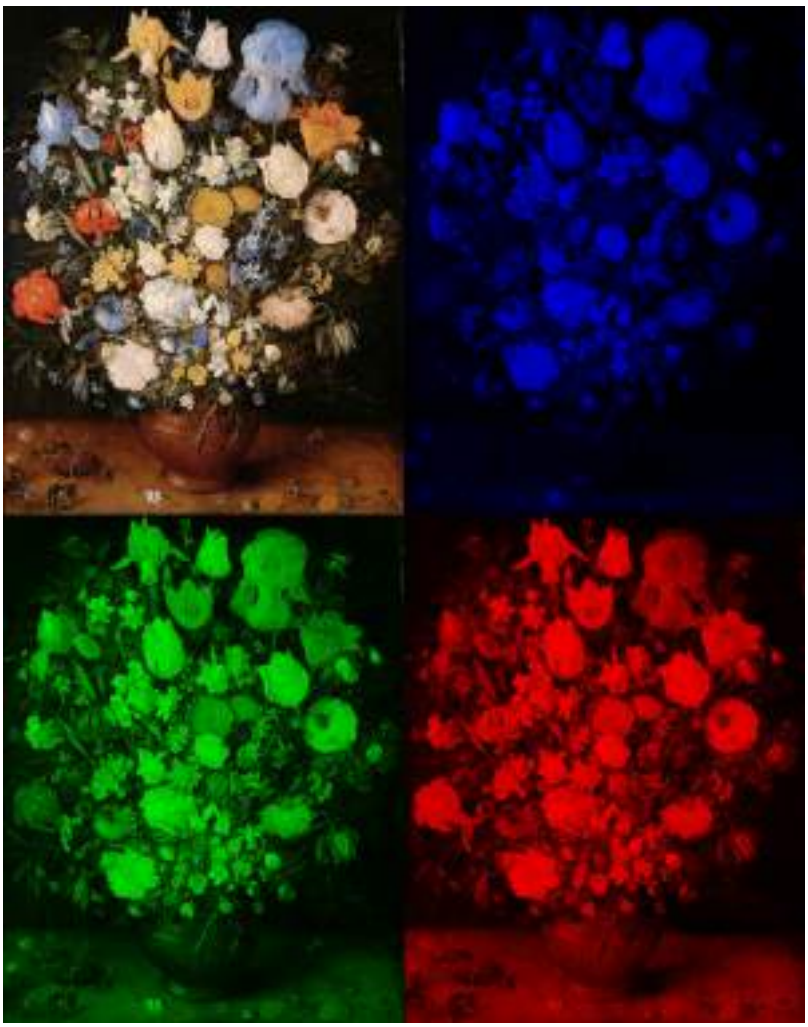

Figure 13: Expected output of `tile_bgr(image)`

Collaboration and Generative AI disclosure

Did you collaborate with anyone? Did you use any Generative AI tools? Briefly explain what you did in the `collaboration-disclosure.txt` file.

Reflection on learning objectives

This is optional to disclose, but it helps us improve the course if you can give feedback.

- what did you take away from this assignment?
- what did you spend the most time on?

- about how much time did you spend total?
- what was easier or harder than expected?
- how could class time have better prepared you for this assignment?

Give your reflections in the `reflection.txt` file if you choose to give us feedback.

Submitting

As described above, use `make submission.zip` to generate a zip file that you can upload to Gradescope. You can upload as many times as you like before the deadline. To account for late penalties, contact the instructor or grader directly if you plan on uploading any further revisions after the deadline.