Report

Data Validation

Overall, my approach to data validation was to adopt solutions which would allow me to validate inputs to the highest degree I possibly could using as little separate submodules as possible.

This philosophy allowed me to eliminate potential error points as soon as I possibly could, segregating the matter of *major* validations to only a couple of submodules at most. For example, concerning importing an image and kernel, I designed and implemented FileIO in such a way that it would ensure a file existed, not throw an exception while reading said file nor if any element is not an integer, merely outputting an error message to the user.

Effectively, the majority of data validation was completed upon running readFromCSV or readFromPNG within FileIO–just a single submodule. This solution was the most optimal due to the fact that the usage of FileIO and, in particular, designed and implemented as I detailed above, virtually validated images and kernels in their entirety (and as previously stated) without the use of multiple submodules in addition.

Design Decisions and Program Separation

My design decisions were largely dictated by the approach I had to data validation. Specifically, I wanted to ensure my classes all had *a single, well defined-responsibility*.

This is objectively the best manner in which to design and implement code. Most notably, this design philosophy:

- Ensures potential error-points are minimised
- Promotes long-term maintainability
- Promotes long-term useability
- Allows for easier integration of new features
- Allows for easier removal of old features

Furthermore, the aforementioned points (in addition to many more), and the promotion of loosely coupled and highly cohesive classes is the reason why programs are separated into different classes and model objects. Without making these kinds of considerations, we are destined to design and implement poorly conceived code.

Practical Implications

Edge detection forms the very founding of many higher-level image processing functionalities. Specifically, with the combination of Artificial Intelligence and Machine Learning, edge-detection operations can be used to interpret, “see”, or “visualise”, objects within images (Adapted from Worksheet 5: Modularity, pg. 11).

In turn, forming the basis of functionalities such as facial recognition scans, much like those used to secure mobile phones and identify people in public settings.

Challenges

I faced two particularly trying tasks during the design and implementation of this program. In particular:

1. Determining the overall design of the program
2. Determining how submodules would be influenced by their parameters

Firstly, I wanted DetectEdges to function as the hub of my program, initiating all higher-level processes on behalf of the user. However, attempting to design and implement such a feature was not easy considering the sheer amount of functionalities this program would need to support.

Secondly, figuring out how to reliably control the execution of submodules using their imports was quite challenging. For example, displaying error messages in my convolution, smoothen and export submodules until an image was imported into the program. (Of course, this increases coupling between various submodules but this feature was necessary to prevent potential errors associated with operations being conducted on non-initialised or non-existent object and variables).

Fortunately enough, I was able to remedy these issues over the course of the assignment with (hopefully, well-received design and implementation).