Lightricks Students Home Assignment - Candidate Version

At Lightricks, we're at the forefront of developing cutting-edge solutions for image and video processing. We're excited to present you with a task that reflects our core activities. Your challenge is to build an Advanced Image Editing System which is capable of applying custom filters and adjustments to images. This exercise is your chance to demonstrate your ability to develop efficient, elegant, and extendable code.

Before diving into filter implementation, it's crucial to understand convolution, a mathematical operation used for applying filters to images. Convolution involves overlaying a kernel (or matrix) onto an image and computing the sum of the product of overlapping elements. This process allows us to apply various effects to an image, like blurring, sharpening, or edge detection. A kernel is essentially a small matrix of numbers that represents how we want to change the colors of the pixels. Different kernels produce different effects:

- **Box Blur Kernel**: This is a simple type of kernel where all values are equal. Imagine it as a small square that moves across every part of your image. For each part of the image it covers, it calculates the average color value of that area and applies it. This process results in the blurring of the image, making it look as if you're viewing it through a frosted glass. It's called "box" because the shape of the area it averages is square or rectangular.
- **Sobel Kernels**: The Sobel operator uses two matrices designed to detect horizontal and vertical changes in the image, essentially highlighting the edges. It's a bit like tracing the outlines of objects in an image with a pencil. The first matrix slides over the image to find vertical lines (up and down edges), and the second matrix looks for horizontal lines (side-to-side edges). When combined, they can show you where all the sharp edges are in an image, making it useful for edge detection and object segmentation.
- **Sharpen Kernel**: This matrix is cleverly designed to accentuate the details of an image, making it appear more “in focus”.

Understanding these concepts is foundational for creating effective image filters.

Languages:
This task can be completed in Python, Java, or C++. Select the language you're most comfortable with or interested in exploring further.

Modern software development has evolved and so did we, hence feel free to use ChatGPT or other LLMs while developing your solution, but we kindly request you to state in your code in docstrings where you used it and which prompts you used for the relevant parts of the code. This helps us understand your thought process better.

You should implement the image processing algorithms from scratch without relying on external libraries or modules that provide pre-implemented functions or utilities for image processing tasks; including color space conversions, kernels or filters. However, you are allowed to use NumPy (or its equivalent in other programming languages) for basic array operations, mathematical functions and loading and saving the image.

You can leverage NumPy's array operations to simplify your code and improve performance. However, the core implementation of the image processing algorithms should be your own original work.

Your Tasks:

1. Design and Build a Configurable Image Editing Tool
    a. Build a modular image editing tool that allows users to apply a sequence of filters and adjustments to an image. The tool should read its configuration from a structured input file (in JSON format), which specifies the image path, operations to apply, their parameters, and whether to display or save the result.
    b. You should be able to run the tool from the command line using the following structure `edit-image --config path_to_config.json`. This command loads the specified JSON file, parses the configuration, and applies the defined image editing operations in order.
2. Code Structure for Extensibility
    a. Design your code to easily accommodate new filters and adjustments. Your architecture should minimize the need for extensive modifications when adding new features.
3. Implement Essential Filters and Features
    a. Filters:
        i. **Box Blur**: Implement this filter by averaging the pixel values in a X*Y neighborhood around each pixel, where X and Y are user-defined as args. This filter should soften the image, creating a "blurry" effect.
        ii. **Edge Detection**: Use the Sobel operator to highlight the edges in the image. This involves applying two convolution kernels, one for horizontal changes and one for vertical, to capture edges in both directions.
        iii. **Sharpen**: Enhance image edges by subtracting the result of a blurred version (see unsharp mask). This filter should have a hardcoded radius of 2 pixels and get the amount (alpha) as a parameter.
    b. Adjustments: Brightness, Contrast, Saturation. Enable users to fine-tune the visual aspects of their images, enhancing their overall appearance.
4. Layering: Implement functionality that allows users to apply multiple filters and adjustments in sequence, with each action considering the results of the previous ones.
5. Display or Save Results. Equip your tool with the capability to either directly display the edited image or save it to a designated path, based on user preference.

Your program should accept a path to a JSON file with the following structure:

```
Unset
{
 "input": "string (required, path to input image)",
 "output": "string (optional, path to save output image)",
 "display": "boolean (optional, whether to display the final image)",
 "operations": [
  {
   "type": "string (required)",
   "<parameter_key>": "<parameter_value>"
  }
 ]
}
```

In case of an incorrect file, the validation process should signify the error with a clear, descriptive message. Here are some additional specifications:

- The configuration file must include at least one of the following: an `output` path to save the result, or `display` set to true to show the edited image, or both.
- Each feature supports specific parameters that control its behavior. Supported operations include `brightness`, `contrast`, `saturation`, `sharpen`, `sobel`, and `box`. You have the flexibility to define the required parameters and their accepted value ranges for each feature. For example, you might set brightness to 0.6, or configure the box filter with width: 5 and height: 3.
- The configuration file can include multiple feature operations, or none at all. The order in which operations appear in the operations list determines the sequence in which they are applied to the image
- Here's an example:

```
Unset
{
 "input": "image.png",
 "output": "output.png",
 "display": true,
 "operations": [
  {
   "type": "brightness",
   "value": 0.6
  },
  {
   "type": "box",
```

```
      "width": 5,
      "height": 3
    },
    {
      "type": "contrast",
      "value": -3
    }
  ]
}
```

The image editing operation will perform the following actions on the input image:

1. Adjust brightness to 0.6
2. Apply a box filter with the parameters: width=5, height=3
3. Adjust contrast to -3 (negative 3).

Good luck.