Design Document for Node-Based Image Manipulation Editor

1. Overview

The Node-Based Image Manipulation Editor is a PyQt5-based GUI application that allows users to process images using a series of modular nodes. Each node performs a specific image manipulation task, and nodes can be connected to create complex image processing workflows. The application provides nodes for various image manipulation tasks like brightness/contrast adjustment, grayscale conversion, blur, edge detection, color channel splitting, and now, thresholding.

2. Architecture Overview

The architecture follows a **modular** design, with each image manipulation task represented by a **node**. These nodes are the building blocks of the editor and can be connected to each other to form processing pipelines. The editor provides a flexible, extensible interface that allows users to load, manipulate, preview, and save images.

Key Components:

- **Node Editor (`NodeEditor`)**: This is the central UI component. It manages the creation, interaction, and deletion of nodes. It connects nodes to build processing pipelines and provides an interface for loading and saving images.

- **Nodes (`ImageInputNode`, `BrightnessContrastNode`, `GrayscaleNode`, `ColorChannelSplitterNode`, `BlurNode`, `EdgeDetectionNode`, `ThresholdNode`)**: Each of these nodes represents a distinct image manipulation operation. The nodes have inputs and outputs that can be connected together.

- **Canvas (`QGraphicsView` and `QGraphicsScene`)**: The canvas displays nodes and their connections. Nodes are draggable and can be freely arranged on the canvas.

- **Image Display (`QLabel` and `QImage`)**: Displays the resulting image after processing.

3. Node Architecture

Each node has the following basic attributes:

- **Inputs/Outputs**: Each node accepts an image input (or a set of inputs) and generates an output image after performing its operation.

- **Position**: Nodes are movable on the canvas, allowing users to organize them for readability and ease of interaction.

- **State**: Nodes maintain their internal state, which could be image data (e.g., `image`), thresholds (e.g., `threshold_value` in `ThresholdNode`), or other configuration parameters.

Node Class Design:

- **Base Node Class (`Node`)**: This class defines common properties and methods for all nodes (e.g., setting input images, getting output images, and updating previews).

- **Specialized Nodes (`BrightnessContrastNode`, `GrayscaleNode`, etc.)**: These classes extend the base node class and implement specific image manipulation tasks. Each specialized node handles a particular image operation.

Example: ThresholdNode

The `ThresholdNode` applies a thresholding operation to an image, and it supports three different thresholding methods: Binary, Adaptive, and Otsu.

- **Threshold Value**: This parameter defines the threshold level for the Binary method. For Adaptive and Otsu methods, the threshold is automatically calculated.

- **Histogram Display**: The `ThresholdNode` generates and displays the histogram of the input image to assist in setting the threshold value.

- **Thresholding Methods**: The `ThresholdNode` supports three methods:

    - **Binary Thresholding**: Converts pixels to either black or white based on the threshold.

    - **Adaptive Thresholding**: Applies local thresholding based on the local pixel neighborhood.

    - **Otsu's Thresholding**: Automatically calculates an optimal global threshold.

4. Connection Handling

Connections between nodes are handled through the `ConnectionLine` class, which is responsible for drawing curved lines that represent the flow of data from one node to another.

- **Connection Class**: The connection is established between the output socket of one node and the input socket of another node. The `ConnectionLine` uses cubic Bezier curves to represent the link between nodes visually.
- **Multiple Connections**: A node can have multiple outputs and inputs, allowing for complex node graphs.

5. UI/UX Design

The editor follows a clean, minimalist design to ensure ease of use:

- **Left Panel**: This contains buttons for loading/saving images and adding/removing nodes. The layout also includes an image preview and metadata display.
- **Main Canvas**: This is the workspace where the nodes are added, connected, and manipulated. It is implemented using `QGraphicsView` and `QGraphicsScene`.
- **Node Preview**: After applying the operation from any node, the image preview is updated in real-time, allowing users to see the results of their image manipulations.

6. Image Loading and Saving

- **Image Input**: Users can load an image via the `QFileDialog`, and the image is displayed in the image preview area. Once loaded, the image is passed to the first node in the processing pipeline.
- **Image Saving**: After processing the image through one or more nodes, users can save the final result using another `QFileDialog` prompt.

7. ThresholdNode Integration

The `ThresholdNode` was integrated as a new node that performs thresholding operations on the image.

- **Node Layout**: The `ThresholdNode` is positioned on the canvas and added to the scene in a manner similar to other nodes. Users can drag and drop this node into the workflow.
- **Input/Output Handling**: Just like other nodes, the `ThresholdNode` has an input socket where it receives the image to process. It outputs the thresholded image after

processing.

- **Thresholding Control**: The user can adjust the `threshold_value` to control the thresholding effect. The method of thresholding can be selected based on the user's needs.

- **Histogram Display**: The `ThresholdNode` generates a histogram of the image to aid in choosing an appropriate threshold. This histogram is displayed within the node.

8. Extensibility

- **Adding New Nodes**: The modular design allows for easy integration of new nodes. New nodes can be added by extending the base `Node` class and implementing the specific image manipulation logic. For instance, adding new filters, transformations, or detection algorithms would be straightforward.

- **Future Nodes**: Nodes for additional tasks like edge detection, sharpening, or advanced color correction can be easily added to the system.

9. Performance Considerations

- **Image Processing**: The system uses OpenCV to handle image processing. The `cv2` library is efficient for tasks like thresholding, blurring, and edge detection, ensuring that the application can handle moderate-sized images without significant performance degradation.

- **Real-time Updates**: As the user adjusts node parameters (e.g., threshold value), the preview is updated in real-time, which may involve some computation overhead. However, since the processing is done on-demand, the application remains responsive even for larger images.

10. Conclusion

The Node-Based Image Manipulation Editor is a flexible and extensible system for creating complex image processing workflows. The architecture is designed to be modular, allowing for easy addition of new image manipulation nodes. The integration of the `ThresholdNode` follows the same principles and enables thresholding functionality with a clean and intuitive interface.

By utilizing PyQt5 and OpenCV, this editor provides a powerful tool for both novice and advanced users to manipulate images through a visual node-based interface.