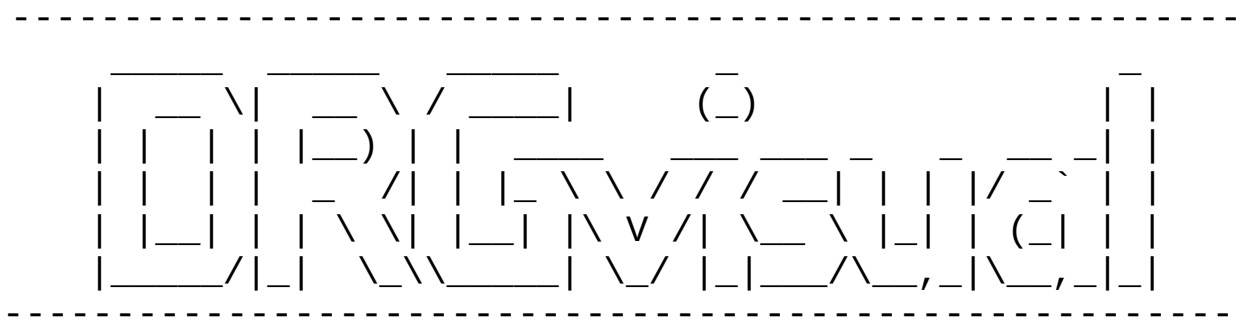

Overview

DRGvisual is a Python-based software designed to assist in the visualization and analysis of immunohistological images of Dorsal Root Ganglia (DRGs). It provides a suite of functions to process, display, and manipulate images, aiding in the exploration and understanding of visual data. The general workflow involves opening the software, loading images, segmenting cell interiors, and calculating corrected total cell fluorescence (CTCF) by subtracting background intensity from each pixel and summing the total fluorescence.

Installation

1. Clone the repository:

shCopy code

```
git clone https://github.com/cgonza1999/DRGvisual.git
```

2. Navigate to the repository directory:

shCopy code

```
cd DRGvisual
```

3. Install the required dependencies:

shCopy code

```
pip install -r requirements.txt
```

Usage

The main driver file for DRGvisual is **main.py**, which orchestrates the execution of various functions provided in the module. To run the software, execute:

shCopy code

```
python main.py
```

This command initiates the data processing and visualization tasks defined within the main function of **main.py**.

File Descriptions and Functions

main.py

The **main.py** file serves as the entry point for the application. It coordinates the overall workflow, calling various functions from **cv_functions.py** and **display_functions.py**. Key components include:

- **ImageLabelerApp**: The main application class handling the setup and initialization of the UI components and image processing variables.
    - **setup_canvases**: Initializes the canvases for image display.
    - **setup_image_variables**: Initializes variables related to image processing.
    - **setup_ui_components**: Initializes UI components such as buttons and labels.
    - **setup_event_handlers**: Configures event handlers for mouse actions and window interactions.
    - **finalize_ui_setup**: Performs final steps in setting up the UI, including drawing the initial image rectangle.
    - **on_mouse_press**: Handles mouse press events for starting pan operations.
    - **on_mouse_drag**: Handles mouse drag events for panning the image.
    - **on_mouse_wheel_scroll**: Handles mouse wheel scroll events for zooming in or out.

cv_functions.py

This file contains various utility functions for image processing, including reading, saving, and manipulating images.

- **display_image(image, title)**: Displays an image using Matplotlib.
- **read_image(file_path, flags=cv2.IMREAD_COLOR)**: Reads an image from a file.
- **save_image(file_path, image)**: Saves an image to a file.
- **convert_to_gray(image)**: Converts an image to grayscale.
- **apply_otsu_threshold(image)**: Applies Otsu's thresholding to an image.
- **find_edges(image)**: Finds edges in an image using Canny edge detection.

- **resize_image(image, width, height)**: Resizes an image to the specified width and height.
- **get_image_files_from_directory(directory)**: Retrieves a list of image files from a specified directory.
- **display_images_from_directory(directory)**: Displays all images from a specified directory.

display_functions.py

This file contains functions for displaying and manipulating images on a Tkinter canvas, including positioning and resizing images to fit within the application window.

- **convert_to_photoimage(image)**: Converts a CV2 image to a Tkinter PhotoImage object.
- **position_image_in_canvas(self, image)**: Positions and resizes an image to fit within the application's canvas.
- **display_current_image(self)**: Displays the current image on the application's canvas.
- **update_image_label_display(self)**: Updates the display of the current image's label.
- **rgb_split(self)**: Splits the current image into its RGB components and displays them.
- **next_image(self)**: Displays the next image in the series.
- **prev_image(self)**: Displays the previous image in the series.
- **prompt_for_labels(self)**: Prompts the user to label the loaded images.
- **save_labels()**: Saves the labels for the images.

Segmentation and Fluorescence Calculation

The core functionality of the software revolves around segmenting cells in immunohistological images and calculating fluorescence.

1. **Image Loading**: The software loads images using the **read_image** function.
2. **Grayscale Conversion**: Images are converted to grayscale using the **convert_to_gray** function to facilitate processing.
3. **Thresholding**: Otsu's thresholding is applied to segment the image into foreground (cells) and background.
4. **Edge Detection**: Canny edge detection is used to refine the segmentation.

5. **Segmentation**: A semi-automated method, potentially involving user input, is used to delineate cell interiors accurately.
6. **Fluorescence Calculation**: The corrected total cell fluorescence (CTCF) is calculated by subtracting the background intensity from each pixel's intensity and summing the total fluorescence across the cell.