Phase 2: The Action (Callbacks)

Modules Covered:

- `gui/callbacks.py` (Event Handlers)

Goal: Explain what happens when the user clicks a button. This phase acts as the "Bridge" between the visual window (Phase 1) and the math engine (Phase 3).

---

1. Preparing the Image (Normalization)

Before we can do any math, we need to ensure the image data is in the correct format. The raw image might be integer data (0-255), but Fourier Transforms work best with decimal numbers (floats).

```
def normalize_to_float_for_processing(arr):
    # 1. Convert integers to floating point numbers
    arrf = arr.astype(np.float64)

    # 2. Handle "Percentile Stretching"
    # This ignores extreme outlier pixels (very bright/dark
       noise)
    # so they don't mess up the scaling.
    vmin = np.percentile(arrf, 1)
    vmax = np.percentile(arrf, 99)

    # 3. Scale the data to be strictly between 0.0 and 255.0
    scaled = (arrf - vmin) / (vmax - vmin)
    return scaled * 255.0
```

Why we do this: If we feed raw integers into the FFT math, we might lose precision. Converting to ‘float64‘ ensures all our calculations are accurate.

2. The Main Trigger (`run_highboost_callback`)

This is the function that runs when you click the green **"Run High-Boost"** button.

Step A: Gathering Inputs

First, it grabs the values the user typed into the GUI.

```
def run_highboost_callback(app):
    # 1. Get the Image
    if app.image_np is None:
        return  # Stop if no image is loaded

    # 2. Get Parameters from the GUI variables
    mask_type = app.filter_type.get().lower()  # e.g., "gaussian
       "
    D0 = float(app.D0_val.get())               # e.g., 50.0
    r = float(app.r_val.get())                 # e.g., 1.8
```

Step B: The Decision (Color vs. Gray)

It checks if the image has color channels and sends it to the correct processor.

```
    # 3. Decide which math engine to use
    if app.is_color:
        # Send to the Color Processor (Phase 3)
        out_img = process_color_rgb(
            arr_for_proc,
            mask_kwargs={"mask_type": mask_type, "D0": D0},
            r=r
        )
    else:
        # Send to the Grayscale Processor
        out_img = process_grayscale(...)
```

Step C: Updating the Display

Once the math returns a result (`out_img`), we put it back on the screen.

```
    # 4. Save the result to the app's memory
    app.output_np = out_img

    # 5. Show it in the Right-Hand Panel
    app.preview_boosted(out_img)
```

3. Saving the Result (`save_output_callback`)

This runs when you click the purple **"Save Output"** button.

```
def save_output_callback(app):
    # 1. Ask user where to save (Pop-up dialog)
    save_path = filedialog.asksaveasfilename(
        defaultextension=".png"
    )

    # 2. Save the file to disk
    if save_path:
        save_image(save_path, app.output_np)
```

Summary of Phase 2: This phase handles the logic of "orchestration." It doesn't do the heavy math itself; instead, it prepares the data, calls the math experts (Core modules), and handles the results.