Installation and First UI Button

Installation Setup

1. Install Anaconda

Follow the instructions to install Anaconda: Installing Anaconda Distribution

2. Create the `kgml_workshop` Environment

Open your terminal or Anaconda Prompt and run the following:

```
conda create -n kgml_workshop python=3.9 -y
conda activate kgml_workshop
pip install numpy scipy pandas matplotlib scikit-image imageio tifffile magicgui napari[all] pyqt5
```

3. Install PyCharm Community Edition

Download and install from: Install PyCharm | PyCharm Documentation

---

A Working Napari Plugin That:

- Loads and displays an image
- Runs multiple segmentation methods (Otsu, Entropy, Gaussian, Stardist)
- Runs instance segmentation (label connected regions)
- Measures properties (area, solidity, etc.)
- Saves results to CSV
- Has a full interactive UI

Step-by-Step Implementation

1. Run the Skeleton Code

Make sure the base plugin structure is running with Napari.

2. Modify `ui_setup.py`

At the top of the file, add the following import:

```
from core.open_image_action import OpenImageAction
```

Then Inside the UI Block, Add This Code

```
#------------------ put the code here ---------------------------

buttons_info = [
    ("Open Dataset and Preprocess", lambda: OpenImageAction(plugin).execute()),
]

for text, func in buttons_info:
    button = QPushButton(text)
    button.clicked.connect(func)
    button.setMinimumSize(plugin.BUTTON_WIDTH, plugin.BUTTON_HEIGHT)
    layout.addWidget(button)

#----------------------------------------------------------------
```

Next Step

Create the file `open_image_action.py` inside the `core/` directory.

Create `open_image_action.py` in `core/`

```
 import matplotlib
matplotlib.use('Qt5Agg')

import numpy as np
from qtpy.QtWidgets import QMessageBox, QApplication, QFileDialog
from skimage.io import imread
import os

class OpenImageAction:
    """
        Action class to open and load an image into the Napari viewer.
        Supports both standard image formats and .npy arrays.

        Author: Dr. Yasmin Kassim
    """

    def __init__(self, plugin):
        self.plugin = plugin

    def execute(self):
        if self.plugin.analysis_stage is not None:
            QMessageBox.information(None, 'Analysis Details',
                'The stack is already read. If you want to read another stack, press "Reset" button.')
            return

        file_path, _ = QFileDialog.getOpenFileName(
            caption="Select Image",
            filter="Images (*.tif *.tiff *.png *.jpg *.jpeg *.npy)"
        )

        if not file_path:
            return

        self.plugin.pkl_Path = file_path
        self.plugin.loading_label.setText("<font color='red'>Processing... Please wait</font>")
        QApplication.processEvents()

        try:
            # Load image based on extension
            if file_path.endswith('.npy'):
                image = np.load(file_path)
            else:
                image = imread(file_path)

            # Add to Napari
            if hasattr(self.plugin, 'viewer'):
                self.plugin.original_image = image
                self.plugin.viewer.add_image(image, name=os.path.basename(file_path))
            else:
                QMessageBox.warning(None, 'Viewer Not Found', 'Napari viewer instance not available in plugin.')

            self.plugin.analysis_stage = 'image_loaded'
            self.plugin.loading_label.setText("<font color='green'>Image loaded successfully</font>")

        except Exception as e:
            QMessageBox.critical(None, 'Error', f'Failed to load image:\n{str(e)}')
            self.plugin.loading_label.setText("<font color='red'>Failed to load image</font>")
```

Segmentation Setup and `segmentation_action.py` Code

Add to `ui_setup.py` for Segmentation UI

```
 # --- Segmentation Method Selection Widget ---
segmentation_widget = QWidget()
segmentation_layout = QVBoxLayout(segmentation_widget)

# Label
segmentation_label = QLabel("Select Segmentation Method:")
segmentation_label.setAlignment(Qt.AlignLeft)
segmentation_layout.addWidget(segmentation_label)

# ComboBox with methods
plugin.segmentation_method_combo = QComboBox()
plugin.segmentation_method_combo.addItems(["Otsu", "Entropy", "Gaussian", "Stardist"])
segmentation_layout.addWidget(plugin.segmentation_method_combo)

# Add to main layout
layout.addWidget(segmentation_widget)

buttons_info = [
    ("Generate Segmentation", lambda: SegmentationAction(plugin).execute()),
]
for text, func in buttons_info:
    button = QPushButton(text)
    button.clicked.connect(func)
    button.setMinimumSize(plugin.BUTTON_WIDTH, plugin.BUTTON_HEIGHT)
    layout.addWidget(button)
```

Create `core/segmentation_action.py`

```
 import numpy as np
from qtpy.QtWidgets import QMessageBox
from skimage.filters import threshold_otsu, gaussian
from skimage.draw import disk
from skimage.filters.rank import entropy
from skimage.morphology import disk
from skimage.util import img_as_ubyte
# ------ to use stardist
#from csbdeep.utils import normalize
#from stardist.models import StarDist2D
#from stardist import random_label_cmap
import matplotlib.pyplot as plt
import os

class SegmentationAction:
    """
       A class to apply different segmentation methods to an image
       and update the Napari viewer with the result.
    """

    def __init__(self, plugin):
        self.plugin = plugin

    def execute(self):
        if self.plugin.analysis_stage != 'image_loaded':
            QMessageBox.warning(None, 'No Image Loaded', 'Please load an image before applying segmentation.')
            return

        try:
```

```
        try:
            image = self.plugin.original_image.copy()  # Always use original image

            method = self.plugin.segmentation_method_combo.currentText().lower()

            if method == "otsu":
                result = self.apply_otsu(image)
            elif method == "entropy":
                result = self.apply_entropy_segmentation(image)
            elif method == "gaussian":
                result = self.apply_gaussian_threshold(image)
            elif method == "stardist":
                self.apply_stardist(image)
                return
            else:
                QMessageBox.warning(None, 'Invalid Method', f'Segmentation method "{method}" is not supported.')
                return

            self.update_segmentation_layer(result)

        except Exception as e:
            QMessageBox.critical(None, 'Segmentation Error', str(e))

    def update_segmentation_layer(self, mask):
        layer_name = "segmentation_result"
        mask = mask.astype(np.uint8)

        if layer_name in self.plugin.viewer.layers:
            self.plugin.viewer.layers[layer_name].data = mask
        else:
            self.plugin.viewer.add_labels(mask, name=layer_name)

    def apply_otsu(self, image):
        if image.ndim > 2:
            image = image[0]
        thresh = threshold_otsu(image)
        return image > thresh

    def apply_entropy_segmentation(self, image, neighborhood_size=4, threshold=4.0):
        if image.ndim > 2:
            image = image[0]
        image = image.astype(np.float32)
        if image.max() > 1:
            image = image / image.max()  # normalize to [0,1]
        image = img_as_ubyte(image)  # safe to convert now
        ent = entropy(image, disk(neighborhood_size))
        return (ent > threshold).astype(np.uint8)

    def apply_gaussian_threshold(self, image, sigma=0.5):
        if image.ndim > 2:
            image = image[0]
        blurred = gaussian(image, sigma=sigma)
        return blurred > blurred.mean()

    def apply_stardist(self, image):
        QMessageBox.information(
            None,
            'Stardist Not Available',
            'Stardist segmentation is not installed or implemented yet.\n\n'
            'Please install stardist and its dependencies to use this method.'
        )
        return None
```

Adding Instance Segmentation and Delete Button

UI Setup: Update Buttons

Add this to your `ui_setup.py` button setup block:

```
buttons_info = [
    ("Generate Segmentation", lambda: SegmentationAction(plugin).execute()),
    ("Generate Instance Segmentation", lambda: InstanceSegmentationAction(plugin).execute())
]
```

InstanceSegmentationAction

Create a new file inside the `core` folder named `InstanceSegmentationAction.py`:

```
from skimage.measure import label
from qtpy.QtWidgets import QMessageBox

class InstanceSegmentationAction:
    """
        Converts a binary segmentation mask into an instance-labeled mask using connected components.
        The result is added as a Napari Labels layer and stored in the plugin's internal state.

        Author: Dr. Yasmin Kassim
    """

    def __init__(self, plugin):
        self.plugin = plugin

    def execute(self):
        # Check for segmentation_result layer
        if "segmentation_result" not in self.plugin.viewer.layers:
            QMessageBox.warning(None, 'Missing Layer', 'Segmentation result layer not found.')
            return

        # Get binary mask
        binary_mask = self.plugin.viewer.layers["segmentation_result"].data
        if binary_mask.max() == 0:
            QMessageBox.warning(None, 'Empty Mask', 'Segmentation result is empty.')
            return

        # Generate instance-labeled mask
        instance_mask = label(binary_mask)

        # Save in plugin for later use
        self.plugin.labels = instance_mask

        # Add to viewer with random colors
        layer_name = "instance_segmentation"
        if layer_name in self.plugin.viewer.layers:
            self.plugin.viewer.layers[layer_name].data = instance_mask
        else:
            self.plugin.viewer.add_labels(instance_mask, name=layer_name)

        QMessageBox.information(None, 'Instance Segmentation', f"Found {instance_mask.max()} instances.")
```

Add Delete Button to `ui_setup.py`

Also in `ui_setup.py`, add the following lines:

```
from core.delete_action import DeleteAction

delete_action = DeleteAction(plugin)
delete_widget = delete_action.create_filter_component_widget()
layout.addWidget(delete_widget.native)
```

DeleteAction

Create a new file inside the `core` folder named `delete_action.py`:

```
import numpy as np
from skimage.morphology import remove_small_objects
from qtpy.QtWidgets import QMessageBox, QApplication
from magicgui import magicgui

class DeleteAction:
    """
    This class deletes labels smaller than a certain size.
    Author: Dr. Yasmin Kassim
    """

    def __init__(self, plugin):
        self.plugin = plugin

    def create_filter_component_widget(self):
        @magicgui(call_button="Delete Labels")
        def _widget(size: int):
            self.delete_small_components(size)
        return _widget

    def delete_small_components(self, min_size: int):
        self.plugin.loading_label.setText("<font color='red'>Processing..., Wait</font>")
        QApplication.processEvents()

        if not hasattr(self.plugin, "labels") or self.plugin.labels is None:
            QMessageBox.warning(None, "No Labels", "No label mask found in plugin.")
            self.plugin.loading_label.setText("")
            return

        # Filter out small components
        instance_mask = self.plugin.labels.astype(np.int32)
        filtered_mask = remove_small_objects(instance_mask, min_size=min_size)

        self.plugin.labels = filtered_mask  # update stored labels

        # Update or add to viewer
        layer_name = "instance_segmentation"
        if layer_name in self.plugin.viewer.layers:
            self.plugin.viewer.layers[layer_name].data = filtered_mask
        else:
            self.plugin.viewer.add_labels(filtered_mask, name=layer_name)

        self.plugin.loading_label.setText("<font color='green'>Filtered small components ✔</font>")
```

Add Measurement Computation to UI

1. Update `ui_setup.py`

Add this import:

```
from core.measurments_action import LabelMeasurement
```

Add the following button inside your layout block:

```
buttons_info = [
    ("Calculate Measurements", lambda: LabelMeasurement(plugin).execute()),
]

for text, func in buttons_info:
    button = QPushButton(text)
    button.clicked.connect(func)
    button.setMinimumSize(plugin.BUTTON_WIDTH, plugin.BUTTON_HEIGHT)
    layout.addWidget(button)
```

---

2. Create `measurments_action.py` inside `core/`

```
import pandas as pd
from skimage.measure import regionprops_table
from qtpy.QtWidgets import QMessageBox, QApplication
import os

class LabelMeasurement:
    """
    Measures 2D region properties and overlays some features labels on each instance.
    Author: Dr. Yasmin Kassim
    """

    def __init__(self, plugin):
        self.plugin = plugin

    def execute(self):
        self.plugin.loading_label.setText("<font color='red'>Measuring..., please wait</font>")
        QApplication.processEvents()

        if not hasattr(self.plugin, "labels") or self.plugin.labels is None:
            QMessageBox.warning(None, "No Labels", "No label mask found in plugin.")
            self.plugin.loading_label.setText("")
            return

        label_mask = self.plugin.labels

        if label_mask.ndim != 2:
            QMessageBox.warning(None, "Invalid Shape", "Only 2D label masks are supported.")
            self.plugin.loading_label.setText("")
            return

        # Properties to extract
        props = [
            'label',
            'area',
            'bbox_area',
            'convex_area',
            'eccentricity',
            'equivalent_diameter',
            'extent',
            'feret_diameter_max',
            'major_axis_length',
            'minor_axis_length',
            'orientation',
            'perimeter',
            'solidity',
            'centroid',
```

```
            'bbox',
            'moments_central',
            'moments_hu',
            'moments_normalized',
        ]

        try:
            measurements = regionprops_table(label_mask, properties=props)
            df = pd.DataFrame(measurements)

            # Save automatically to self.plugin.rootfolder
            save_path = os.path.join(self.plugin.rootfolder, "label_measurements.csv")
            df.to_csv(save_path, index=False)
            QMessageBox.information(None, "Saved", f"Measurements saved to:\n{save_path}")

            # Overlay results using points + text
            text_coords = list(zip(df["centroid-0"], df["centroid-1"]))  # (row, col) = (y, x)
            text_strings = [
                f"Area: {a}\nSolidity: {s:.2f}\nEcc: {e:.2f}"
                for a, s, e in zip(df["area"], df["solidity"], df["eccentricity"])
            ]

            if "instance_annotations" in self.plugin.viewer.layers:
                self.plugin.viewer.layers["instance_annotations"].data = text_coords
                self.plugin.viewer.layers["instance_annotations"].text.values = text_strings
            else:
                self.plugin.viewer.add_points(
                    text_coords,
                    name="instance_annotations",
                    size=0,
                    text={
                        "string": text_strings,
                        "anchor": "upper_left",
                        "color": "white",
                        "size": 10,
                    }
                )

        except Exception as e:
            QMessageBox.critical(None, "Measurement Error", str(e))

        self.plugin.loading_label.setText("<font color='green'>Measurements Done ✔</font>")
```

Stardist Segmentation Setup

Step 1: Install Required Packages

First, activate your Anaconda environment and install the necessary packages:

```
 pip install tensorflow
pip install stardist
pip install "numpy<2.0"
```

Step 2: Add Stardist Segmentation Method

Add the following method to your `SegmentationAction` class:

```
def apply_stardist(self, image):
    try:
        if image.ndim > 2:
            image = image[0]
        # Normalize image
        image = normalize(image, 1, 99.8)
        # Load pretrained model
        base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), '..', 'models', 'StarDist2D'))
        model = StarDist2D(None, name='2D_versatile_fluo', basedir=base_dir)

        # Predict instances
        labels, _ = model.predict_instances(image, prob_thresh=0.6, nms_thresh=0.6)
        # Save to viewer and plugin
        self.plugin.labels = labels
        # Add result to viewer
        layer_name = "segmentation_result"
        if layer_name in self.plugin.viewer.layers:
            self.plugin.viewer.layers[layer_name].data = labels
        else:
            self.plugin.viewer.add_labels(labels, name=layer_name)
        QMessageBox.information(None, 'Stardist Segmentation', f'Successfully segmented {labels.max()} instances.')
    except Exception as e:
        QMessageBox.critical(None, 'Stardist Error', f'Error during StarDist segmentation:\n{str(e)}')

    return None
```

add those imports:

```
from csbdeep.utils import normalize
from stardist.models import StarDist2D
from stardist import random_label_cmap
```