# run\_workflow\_Dcp1a\_TTP

import subprocess

import os

def run\_imagej\_macro(imagej\_path, macro\_path, step\_description):

subprocess.run([imagej\_path, "-macro", macro\_path], check=True)

print(f"{step\_description} completed successfully.")

def run\_python\_script(script\_path, working\_directory=None, step\_description=None):

if working\_directory:

os.chdir(working\_directory)

subprocess.run(["python", script\_path], check=True)

print(f"{step\_description} completed successfully.")

**# Paths to ImageJ executable and macros**

imagej\_path = r"C:\Users\Victoria\fiji-win64\Fiji.app\ImageJ-win64.exe"

macros = [

(r"C:\Users\Victoria\fiji-win64\Fiji.app\ImageAnalysis\Dcp1a\convert\_czi\_to\_tiff\_Dcp1a\_TTP.ijm", "Convert to tif"),

(r"C:\Users\Victoria\fiji-win64\Fiji.app\ImageAnalysis\Dcp1a\set\_intensity\_split\_channel\_Dcp1a\_TTP.ijm", "Split channel and enhance intensity"),

(r"C:\Users\Victoria\fiji-win64\Fiji.app\ImageAnalysis\Dcp1a\identify\_Pbodies\_TTP.ijm", "Create Pbody Mask"),

(r"C:\Users\Victoria\fiji-win64\Fiji.app\ImageAnalysis\Dcp1a\CreatePbodyTTPcellMask.ijm", "Create adapted Masks for measurement"),

(r"C:\Users\Victoria\fiji-win64\Fiji.app\ImageAnalysis\Dcp1a\measure\_intensity\_Dcp1a\_cellTTP.ijm", "Measure Pbodies in CellTTP")

]

**# Step 1: Convert to tif**

run\_imagej\_macro(imagej\_path, macros[0][0], macros[0][1])

**# Step 2: Split channel and enhance intensity**

run\_imagej\_macro(imagej\_path, macros[1][0], macros[1][1])

**# Step 3: Create Mask in cell pose**

**# Activate the conda environment**

subprocess.run(["conda", "activate", "cellpose\_env"], shell=True, check=True)

print("Conda environment activated.")

**# Open GUI (if necessary)**

subprocess.run(["python", "-m", "cellpose"], shell=True)

print("Cellpose GUI opened.")

**# Run the cell pose script**

cellpose\_script\_path = r"C:\Users\Victoria\ImageAnalysis\CellPose\process\_images\_Dcp1a\_TTP.py"

cellpose\_working\_directory = r"C:\Users\Victoria\ImageAnalysis\CellPose"

run\_python\_script(cellpose\_script\_path, cellpose\_working\_directory, "Cellpose script")

**# Step 4: Create Pbody Mask**

run\_imagej\_macro(imagej\_path, macros[2][0], macros[2][1])

**# Step 5: Create adapted Masks for measurement**

run\_imagej\_macro(imagej\_path, macros[3][0], macros[3][1])

**# Step 6: Measure Pbodies in CellTTP**

run\_imagej\_macro(imagej\_path, macros[4][0], macros[4][1])

print("All steps completed successfully.")

## # Step 1: Convert to tif

// Macro to batch convert CZI files to TIFF using Bio-Formats

// Put this macro in a file named convert\_czi\_to\_tiff.ijm

// Set input and output folder paths

inputDir = "C:\\Users\\Victoria\\ImageAnalysis\\1\_Data\\Dcp1a\_TTP\\";

outputDir = "C:\\Users\\Victoria\\ImageAnalysis\\2\_TifConversion\\Dcp1a\_TTP\\";

// Get list of CZI files in the input directory

list = getFileList(inputDir);

// Check if the list is empty

if (list.length == 0) {

print("No CZI files found in the input directory.");

} else {

// Disable display updates for faster processing

setBatchMode(true);

// Iterate over each CZI file

for (i = 0; i < list.length; i++) {

// Print the name of the current file

print("Processing file: " + list[i]);

// Open CZI file using Bio-Formats importer

run("Bio-Formats Importer", "open=[" + inputDir + list[i] + "] autoscale color\_mode=Default rois\_import=[ROI manager] view=Hyperstack stack\_order=XYCZT");

// Wait for the image to open (adjust the delay if needed)

wait(1000);

// Extract filename without extension

fileName = File.getNameWithoutExtension(list[i]);

// Construct output file path

outputFile = outputDir + fileName + ".tif";

// Save the currently open image as TIFF

saveAs("Tiff", outputFile);

// Close the currently open image

close();

print("Converted " + list[i] + " to TIFF");

}

// Re-enable display updates

setBatchMode(false);

}

// Close ImageJ

run("Quit");

## # Step 2: Split channel and enhance intensity

// Set input and output folder paths

inputFolder = "C:\\Users\\Victoria\\ImageAnalysis\\2\_TifConversion\\Dcp1a\_TTP\\";

outputFolder = "C:\\Users\\Victoria\\ImageAnalysis\\3\_IntensityMask\\Dcp1a\_TTP\\";

// Function to process the image and adjust intensity for grayscale 16-bit images with different black and white values for each channel

function processImage(inputFolder, outputFolder, filename, minValues, maxValues) {

// Open the image from the input folder

open(inputFolder + filename);

// Retrieve information about the image

orgName = getTitle();

// Split channels

run("Split Channels");

// Adjust intensity for each channel and save them separately

for (i = 0; i < 3; i++) { // Assuming there are always 3 channels

// Select the current channel

selectWindow("C" + (i + 1) + "-" + orgName); // Channels are 1-indexed in ImageJ

// Get current minimum value

getStatistics(area, mean, min, max);

// Adjust intensity for the current channel using specific black and white values

setMinAndMax(min, max);

// Remove .tif extension from original name

baseName = replace(orgName, ".tif", "");

// Save the adjusted channel as TIFF

saveAs("Tiff", outputFolder + baseName + "\_Channel" + (i + 1) + ".tif");

// Close the channel window

close();

}

// Close the original split image windows if still open

for (i = 1; i <= 3; i++) {

if (isOpen("C" + i + "-" + orgName)) {

selectWindow("C" + i + "-" + orgName);

close();

}

}

}

// Set black and white values for each channel of 16-bit grayscale images

minValues = newArray(10000, 20000, 5000); // Example values for each channel (adjust as needed)

maxValues = newArray(20000, 30000, 8000); // Example values for each channel (adjust as needed)

// Get list of files in the input folder

list = getFileList(inputFolder);

// Process each image in the input folder

for (i = 0; i < list.length; i++) {

filename = list[i];

processImage(inputFolder, outputFolder, filename, minValues, maxValues);

}

// Close ImageJ

run("Quit");

## # Step 3: Create Mask in cell pose

import os

import subprocess

import shutil

# Function to rename output files

def rename\_output\_files(output\_dir, suffix):

for root, dirs, files in os.walk(output\_dir):

for file in files:

if file.endswith("\_cp\_masks.tif"):

base\_name = file.replace("\_cp\_masks.tif", "")

new\_name = f"{base\_name}\_{suffix}.tif"

os.rename(os.path.join(root, file), os.path.join(root, new\_name))

# Function to process images using Cellpose CLI with different settings for cytoplasm channels

def process\_images(input\_dir, output\_dir, cytoplasm\_channel1, cytoplasm\_channel2, cell\_diameter, flow\_threshold, cellprob\_threshold, use\_gpu):

# Create separate output directories for cytoplasm

cytoplasm\_output1\_dir = os.path.join(output\_dir, 'cell')

cytoplasm\_output2\_dir = os.path.join(output\_dir, 'cell\_TTP')

os.makedirs(cytoplasm\_output1\_dir, exist\_ok=True)

os.makedirs(cytoplasm\_output2\_dir, exist\_ok=True)

# Construct the command for cytoplasm segmentation for channel 1

cytoplasm\_command1 = [

'cellpose',

'--dir', input\_dir,

'--save\_tif',

'--savedir', cytoplasm\_output1\_dir,

'--chan', str(cytoplasm\_channel1),

'--chan2', str(cytoplasm\_channel2),

'--diameter', str(cell\_diameter),

'--flow\_threshold', str(flow\_threshold),

'--cellprob\_threshold', str(cellprob\_threshold),

'--model', 'cyto3' # Using 'cyto3' model for cytoplasm segmentation

]

# Construct the command for cytoplasm segmentation for channel 2

cytoplasm\_command2 = [

'cellpose',

'--dir', input\_dir,

'--save\_tif',

'--savedir', cytoplasm\_output2\_dir,

'--chan', str(cytoplasm\_channel2),

'--chan2', str(cytoplasm\_channel1),

'--diameter', str(cell\_diameter),

'--flow\_threshold', str(flow\_threshold),

'--cellprob\_threshold', str(cellprob\_threshold),

'--model', 'cyto3' # Using 'cyto3' model for cytoplasm segmentation

]

if use\_gpu:

cytoplasm\_command1.append('--use\_gpu')

cytoplasm\_command2.append('--use\_gpu')

# Run cytoplasm segmentation for channel 1

try:

subprocess.run(cytoplasm\_command1, check=True)

print("Cytoplasm images (channel 1) processed successfully!")

rename\_output\_files(cytoplasm\_output1\_dir, "cp\_masks\_cell")

except subprocess.CalledProcessError as e:

print("Error processing cytoplasm images (channel 1):", e)

# Run cytoplasm segmentation for channel 2

try:

subprocess.run(cytoplasm\_command2, check=True)

print("Cytoplasm images (channel 2) processed successfully!")

rename\_output\_files(cytoplasm\_output2\_dir, "cp\_masks\_cellTTP")

except subprocess.CalledProcessError as e:

print("Error processing cytoplasm images (channel 2):", e)

# Define input and output directories

input\_directory = 'C:\\Users\\Victoria\\ImageAnalysis\\2\_TifConversion\\Dcp1a\_TTP\\'

output\_directory = 'C:\\Users\\Victoria\\ImageAnalysis\\4\_CellposeTag\\Dcp1a\_TTP\\'

# Define channels for Cellpose processing

cytoplasm\_channel1 = 1 # First cytoplasm channel

cytoplasm\_channel2 = 3 # Second cytoplasm channel

# Define diameters for Cellpose processing

cell\_diameter = 60. # Diameter for cells

# Define flow and cell probability thresholds

flow\_threshold = 0.7 # Adjust based on your dataset

cellprob\_threshold = 0.0 # Adjust based on your dataset

# Flag to use GPU

use\_gpu = True # Set to True if you have a GPU and want to use it

# Process images

process\_images(input\_directory, output\_directory, cytoplasm\_channel1, cytoplasm\_channel2, cell\_diameter, flow\_threshold, cellprob\_threshold, use\_gpu)

## # Step 4: Create Pbody Mask

// Set input and output folder paths

inputFolder = "C:\\Users\\Victoria\\ImageAnalysis\\2\_TifConversion\\Dcp1a\_TTP\\";

MaskFolderCell = "C:\\Users\\Victoria\\ImageAnalysis\\4\_CellposeTag\\Dcp1a\_TTP\\cell\_TTP\\";

MaskFolderPbody = "C:\\Users\\Victoria\\ImageAnalysis\\5\_UpdateTag\\Dcp1a\_TTP\\Pbody\_mask\\";

outputFolder1 = "C:\\Users\\Victoria\\ImageAnalysis\\5\_UpdateTag\\Dcp1a\_TTP\\PbodiesTTPcell\\";

outputFolder2 = "C:\\Users\\Victoria\\ImageAnalysis\\5\_UpdateTag\\Dcp1a\_TTP\\TTPcellwoPbodies\\";

outputFolder3 = "C:\\Users\\Victoria\\ImageAnalysis\\5\_UpdateTag\\Dcp1a\_TTP\\PbodiesTTPcellNumber\\";

// Get a list of image files in the input folder

imageList = getFileList(inputFolder);

// Loop through each image file

for (i = 0; i < imageList.length; i++) {

// Check if the file is an image

if (endsWith(imageList[i], ".tif") || endsWith(imageList[i], ".jpg") || endsWith(imageList[i], ".png")) {

// Extract image name without extension

imageName = File.getNameWithoutExtension(imageList[i]);

// Open the Pbody mask (from MaskFolderPbody)

open(MaskFolderPbody + "Mask\_of\_" + imageName + "\_Channel2.tif");

selectWindow("Mask\_of\_" + imageName + "\_Channel2.tif");

run("Rename...", "title=Pbody");

// Open the cell mask (from MaskFolderCell)

open(MaskFolderCell + imageName + "\_cp\_masks\_cellTTP.tif");

selectWindow(imageName + "\_cp\_masks\_cellTTP.tif");

run("Rename...", "title=cell");

// Process Image calculator and use AND function on both images

run("Image Calculator...", "image1=cell operation=AND image2=Pbody create");

// Select newly generated image

selectWindow("Result of cell");

// Save newly generated image in output folder

saveAs("Tiff", outputFolder1 + "\\" + imageName + "\_masks\_PbodiesTTPcell.tif");

// Process Image calculator and use Subtract function on both images

run("Image Calculator...", "image1=cell operation=Subtract image2=Pbody create");

// Select newly generated image

selectWindow("Result of cell");

// Save newly generated image in output folder

saveAs("Tiff", outputFolder2 + "\\" + imageName + "\_masks\_TTPcellwoPbodies.tif");

// Select renamed image for thresholding

selectWindow(imageName + "\_masks\_PbodiesTTPcell.tif");

// Set threshold so all Pbodies have the same intensity 255/255

setThreshold(1, 65535);

run("Convert to Mask");

// Save the thresholded image in output folder

saveAs("Tiff", outputFolder3 + "\\" + imageName + "\_masks\_PbodiesTTPcellNumber.tif");

// Create ROI with analyze particles and save ROIs

run("Analyze Particles...", "size=0-Infinity display clear add");

roiManager("Deselect");

roiManager("Save", outputFolder3 + "\\" + imageName + "\_rois.zip");

// Close all images to start fresh for the next iteration

run("Close All");

}

}

// close ImageJ

run("Quit");

## # Step 5: Create adapted Masks for measurement

// Define the input and output folders

inputFolder = "C:\\Users\\Victoria\\ImageAnalysis\\3\_IntensityMask\\Dcp1a\_TTP\\";

outputFolder = "C:\\Users\\Victoria\\ImageAnalysis\\5\_UpdateTag\\Dcp1a\_TTP\\Pbody\_mask\\";

// Define the minimum size of P-bodies

var minSize = 1;

var maxSize = 10;

// Get list of files in input folder

list = getFileList(inputFolder);

// Function to isolate P-bodies within a mammalian cell image

function isolatePbodies() {

// Display the threshold box

run("Threshold...");

// Pause execution to allow user to adjust the threshold

waitForUser("Adjust the threshold then press OK");

// Retrieve the threshold values

getThreshold(lower, upper);

// Set proper options for 2D measurements

run("Set Measurements...", "area mean min center redirect=None decimal=3");

selectWindow(fileName);

// Run 2D object counter

run("Analyze Particles...", "size=" + minSize + "-100000 circularity=0.00-1.00 show=Masks display clear include summarize add");

// Save the mask image with the original image name

selectWindow("Mask of " + fileName);

saveAs("Tiff", outputFolder + "Mask\_of\_" + fileName);

}

// Process each file

for (i = 0; i < list.length; i++) {

fileName = list[i];

// Check if file ends with "Channel2"

if (endsWith(fileName, "Channel2.tif")) {

// Open the original image

open(inputFolder + fileName);

// Call the function to isolate P-bodies

isolatePbodies();

// Get the max particle number

max = getResult("Count", 0);

// Create ROIs for each particle

for (j = 1; j <= max; j++) {

roiManager("Select", j);

roiManager("Add");

}

// Save ROIs from mask

roiManager("Deselect");

roiManager("save", outputFolder + fileName + "\_rois.zip");

roiManager("reset");

// Close all windows

close("\*");

}

}

// Display completion message

print("P-body identification.");

// Close ImageJ

run("Quit");

## # Step 6: Measure Pbodies in CellTTP

// Set input and output folder paths

inputFolder = "C:\\Users\\Victoria\\ImageAnalysis\\2\_TifConversion\\Dcp1a\_TTP\\";

MaskFolder1 = "C:\\Users\\Victoria\\ImageAnalysis\\4\_CellposeTag\\Dcp1a\_TTP\\cell\_TTP\\";

MaskFolder2 = "C:\\Users\\Victoria\\ImageAnalysis\\5\_UpdateTag\\Dcp1a\_TTP\\Pbody\_mask\\";

MaskFolder3 = "C:\\Users\\Victoria\\ImageAnalysis\\5\_UpdateTag\\Dcp1a\_TTP\\PbodiesTTPcell\\";

MaskFolder4 = "C:\\Users\\Victoria\\ImageAnalysis\\5\_UpdateTag\\Dcp1a\_TTP\\TTPcellwoPbodies\\";

MaskFolder5 = "C:\\Users\\Victoria\\ImageAnalysis\\5\_UpdateTag\\Dcp1a\_TTP\\PbodiesTTPcellNumber\\";

outputFolder1 = "C:\\Users\\Victoria\\ImageAnalysis\\6\_Measurement\\Dcp1a\_TTP\\TTP\_cell\\";

outputFolder2 = "C:\\Users\\Victoria\\ImageAnalysis\\6\_Measurement\\Dcp1a\_TTP\\Pbodies\\";

outputFolder3 = "C:\\Users\\Victoria\\ImageAnalysis\\6\_Measurement\\Dcp1a\_TTP\\PbodiesTTPcell\\";

outputFolder4 = "C:\\Users\\Victoria\\ImageAnalysis\\6\_Measurement\\Dcp1a\_TTP\\TTPcellwoPbodies\\";

outputFolder5 = "C:\\Users\\Victoria\\ImageAnalysis\\6\_Measurement\\Dcp1a\_TTP\\PbodiesTTPcellNumber\\";

// Get a list of image files in the input folder

imageList = getFileList(inputFolder);

// Loop through each image file

for (i = 0; i < imageList.length; i++) {

// Check if the file is an image

if (endsWith(imageList[i], ".tif") || endsWith(imageList[i], ".jpg") || endsWith(imageList[i], ".png")) {

// Extract image name without extension

imageName = File.getNameWithoutExtension(imageList[i]);

// Open the mask (from MaskFolder)

open(MaskFolder1 + imageName + "\_cp\_masks\_cellTTP.tif");

selectImage(imageName + "\_cp\_masks\_cellTTP.tif");

// Run ROI generation for mask

getStatistics(area, mean, min, max, std, histogram);

for (j = 1; j < max; j++) {

setThreshold(j, j);

run("Create Selection");

roiManager("add");

}

// Save ROIs from mask

roiManager("Deselect");

roiManager("save", MaskFolder1 + imageName + "\_rois.zip");

roiManager("reset");

// Open the input image (multichannel)

open(inputFolder + imageList[i]);

selectImage(imageList[i]);

run("Set Slice...", "slice=3");

// Load and show ROIs for nucleus

roiManager("open", MaskFolder1 + imageName + "\_rois.zip");

roiManager("Show All");

// Set measurement options to include more parameters

run("Set Measurements...", "area mean min max integrated std perimeter shape display redirect=None decimal=3");

// Measure ROIs

selectImage(imageList[i]);

roiManager("Measure");

saveAs("Results", outputFolder1 + imageName + "\_Results01.csv");

roiManager("reset");

run("Clear Results");

// Open the mask (from MaskFolder)

open(MaskFolder2 + "Mask\_of\_" + imageName + "\_Channel2.tif");

selectImage("Mask\_of\_" + imageName + "\_Channel2.tif");

// Create ROI with analyze particles and save ROIs

run("Analyze Particles...", "size=0-Infinity display clear add");

// Save ROIs from mask

roiManager("Deselect");

roiManager("save", MaskFolder2 + imageName + "\_rois.zip");

roiManager("reset");

// Load and show ROIs for nucleus

roiManager("open", MaskFolder2 + imageName + "\_rois.zip");

roiManager("Show All");

// Measure ROIs

selectImage(imageList[i]);

roiManager("Measure");

saveAs("Results", outputFolder2 + imageName + "\_Results02.csv");

roiManager("reset");

run("Clear Results");

// Open the mask (from MaskFolder)

open(MaskFolder3 + imageName + "\_masks\_PbodiesTTPcell.tif");

selectImage(imageName + "\_masks\_PbodiesTTPcell.tif");

// Run ROI generation for nucleus mask

getStatistics(area, mean, min, max, std, histogram);

for (j = 1; j <= max; j++) { // Loop through threshold values

setThreshold(j, j);

run("Create Selection");

if (selectionType() != -1) { // Check if a selection is created

roiManager("add");

print("Added ROI for threshold: " + j);

} else {

print("No ROI created for threshold: " + j);

}

resetThreshold(); // Reset threshold after each iteration

}

// Save ROIs from mask

roiManager("Deselect");

roiManager("save", MaskFolder3 + imageName + "\_rois.zip");

roiManager("reset");

// Load and show ROIs for nucleus

roiManager("open", MaskFolder3 + imageName + "\_rois.zip");

roiManager("Show All");

// Measure ROIs

selectImage(imageList[i]);

roiManager("Measure");

saveAs("Results", outputFolder3 + imageName + "\_Results03.csv");

roiManager("reset");

run("Clear Results");

// Open the mask (from MaskFolder)

open(MaskFolder4 + imageName + "\_masks\_TTPcellwoPbodies.tif");

selectImage(imageName + "\_masks\_TTPcellwoPbodies.tif");

// Run ROI generation for nucleus mask

getStatistics(area, mean, min, max, std, histogram);

for (j = 1; j <= max; j++) { // Loop through threshold values

setThreshold(j, j);

run("Create Selection");

if (selectionType() != -1) { // Check if a selection is created

roiManager("add");

print("Added ROI for threshold: " + j);

} else {

print("No ROI created for threshold: " + j);

}

resetThreshold(); // Reset threshold after each iteration

}

// Save ROIs from mask

roiManager("Deselect");

roiManager("save", MaskFolder4 + imageName + "\_rois.zip");

roiManager("reset");

// Load and show ROIs for nucleus

roiManager("open", MaskFolder4 + imageName + "\_rois.zip");

roiManager("Show All");

// Measure ROIs

selectImage(imageList[i]);

roiManager("Measure");

saveAs("Results", outputFolder4 + imageName + "\_Results04.csv");

roiManager("reset");

run("Clear Results");

// Open the mask (from MaskFolder)

open(MaskFolder5 + imageName + "\_masks\_PbodiesTTPcellNumber.tif");

selectImage(imageName + "\_masks\_PbodiesTTPcellNumber.tif");

// Create ROI with analyze particles and save ROIs

run("Analyze Particles...", "size=0-Infinity display clear add");

// Save ROIs from mask

roiManager("Deselect");

roiManager("save", MaskFolder5 + imageName + "\_rois.zip");

roiManager("reset");

// Load and show ROIs for nucleus

roiManager("open", MaskFolder5 + imageName + "\_rois.zip");

roiManager("Show All");

// Measure ROIs

selectImage(imageList[i]);

roiManager("Measure");

saveAs("Results", outputFolder5 + imageName + "\_Results05.csv");

roiManager("reset");

run("Clear Results");

// Close all images to start fresh for the next iteration

close();

}

}

// Close ImageJ

run("Quit");