import os

import zipfile

import time

import numpy as np

import tifffile as tiff

import matplotlib.pyplot as plt

from skimage import measure

import imagej

from skimage.draw import polygon

def load\_image(image\_path):

"""Charge une image TIFF à partir d'un chemin donné"""

try:

# Vérifier si le fichier est bien un fichier

if not os.path.isfile(image\_path):

raise ValueError(f"Le chemin spécifié {image\_path} n'est pas un fichier valide.")

# Afficher le chemin fourni

print(f"Chargement de l'image depuis {image\_path}...")

# Si l'image est au format TIFF, utiliser tifffile

if image\_path.lower().endswith(('.tif', '.tiff')):

image = tiff.imread(image\_path) # Charger l'image TIFF

else:

img = Image.open(image\_path) # Charger les autres formats avec PIL

img = img.convert("RGB") # Convertir l'image en RGB si nécessaire

image = np.array(img) # Convertir en tableau NumPy

print(f"Image {image\_path} chargée avec succès.")

return image

except Exception as e:

print(f"Erreur lors du chargement de l'image {image\_path}: {e}")

return None

def extract\_zip(zip\_path, output\_dir):

"""Extrait un fichier zip dans un dossier spécifié"""

if not zip\_path.endswith('.zip'):

raise ValueError("Le fichier spécifié n'est pas un fichier .zip valide.")

if not os.path.isfile(zip\_path):

raise ValueError(f"Le chemin spécifié {zip\_path} n'est pas un fichier.")

print(f"Extraction du fichier zip : {zip\_path}")

start\_time = time.time() # Début du chronométrage de l'extraction

with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

# Affichage du contenu du zip avant extraction

print("Contenu du fichier zip :")

zip\_ref.printdir()

zip\_ref.extractall(output\_dir)

end\_time = time.time() # Fin du chronométrage

print(f"Extraction terminée dans le dossier {output\_dir}. Temps écoulé: {end\_time - start\_time:.2f} secondes")

def load\_roi(roi\_path, image\_shape):

"""Charge un fichier ROI et le convertit en masque binaire"""

ij = imagej.init('sc.fiji:fiji') # Initialiser ImageJ via pyimagej

# Charger le fichier ROI

roi = ij.io().open(roi\_path)

rois = roi.get(0) # On récupère la première ROI

# Créer un masque vide

mask = np.zeros(image\_shape, dtype=np.uint8)

# Extraire les coordonnées des ROI et remplir le masque

for r in rois:

if r is not None:

x, y = r.getBounds()

# Remplir le masque avec 1 pour les pixels dans la ROI

rr, cc = polygon([p[1] for p in r], [p[0] for p in r], shape=image\_shape)

mask[rr, cc] = 1

return mask

def preprocess\_data(image, mask):

"""Redimensionner les images et les masques pour correspondre aux dimensions"""

return image, mask

def show\_image\_with\_cells(image, mask):

"""Affiche l'image avec les contours des cellules extraite du masque"""

print("Affichage de l'image avec contours des cellules...")

contours = measure.find\_contours(mask, 0.5)

# Affichage de l'image

plt.imshow(image)

plt.axis('off') # Ne pas afficher les axes

# Tracer les contours des cellules

for contour in contours:

plt.plot(contour[:, 1], contour[:, 0], color='r', linewidth=2) # Tracer les contours en rouge

plt.show()

def interactive\_image\_processing():

"""Interface interactive pour traiter les images"""

while True:

# Demander le chemin de l'image de référence

image\_path = input("Entrez le chemin de l'image de référence (format .tif, ou tapez 'exit' pour quitter): ")

if image\_path.lower() == 'exit':

break

# Afficher le chemin fourni pour vérifier s'il est correct

print(f"Chemin de l'image fourni : {image\_path}")

# Charger l'image

image = load\_image(image\_path)

if image is None:

print(f"Erreur lors du chargement de l'image : {image\_path}")

continue # Si l'image n'a pas pu être chargée, passer à l'itération suivante

# Image chargée avec succès, procéder à l'étape suivante

print(f"Image chargée avec succès!")

# Demander le chemin du fichier zip contenant les masques

zip\_path = input("Entrez le chemin du fichier .zip contenant les masques de référence (ou tapez 'exit' pour quitter): ")

if zip\_path.lower() == 'exit':

break

# Vérifier si le fichier zip est valide et mesurer le temps d'extraction

try:

extract\_zip(zip\_path, "extracted\_masks")

except ValueError as e:

print(e)

continue # Si le fichier n'est pas valide, on continue à demander l'entrée

print("Masques extraits. Traitement des masques...")

# Extraire les masques du fichier zip

output\_dir = "extracted\_masks"

mask\_files = [f for f in os.listdir(output\_dir) if f.endswith('.roi')]

print(f"Masques extraits : {mask\_files}")

if not mask\_files:

print("Aucun masque ROI trouvé dans le fichier ZIP.")

continue

# Boucle sur chaque masque

for mask\_file in mask\_files:

mask\_path = os.path.join(output\_dir, mask\_file)

# Charger le masque ROI et le convertir en masque binaire

print(f"Chargement du masque ROI : {mask\_file}")

mask\_resized = load\_roi(mask\_path, image.shape)

# Affichage de l'image avec les contours des cellules

show\_image\_with\_cells(image, mask\_resized)

print(f"Affichage de l'image avec contours de {mask\_file}")

# Demander à l'utilisateur si la détection est correcte

user\_input = input("Est-ce que la détection est correcte pour ce masque ? (yes/no): ").lower()

if user\_input == 'yes':

print("Très bien, procédons à l'image suivante !")

continue

elif user\_input == 'no':

print("Veuillez ajuster les paramètres de détection et réessayer.")

continue

else:

print("Entrée non valide. Veuillez répondre par 'yes' ou 'no'.")

# Une fois le traitement de tous les masques, demander si l'on continue

continue\_processing = input("Souhaitez-vous traiter une autre image et son fichier zip ? (yes/no): ").lower()

if continue\_processing != 'yes':

print("Fin du traitement.")

break

# Lancer l'interface interactive

interactive\_image\_processing()