PYTHON

import numpy as np

import matplotlib.pyplot as plt

import statistics as st

import pandas as pd

from skimage.morphology import square

from skimage.measure import label, regionprops

from matplotlib.collections import PatchCollection

import matplotlib.patches as mpatches

from matplotlib.axes import Axes

from skimage.filters import threshold\_otsu

import skimage.io

import skimage.filters

import skimage.measure

import os

#Show the actual image from an image matrix

def show\_image(image, title='Image', cmap\_type='gray'):

plt.imshow(image, cmap=cmap\_type)

plt.title(title)

plt.axis('off')

#To read an image:

#go to the files icon on the left toolbar, press "Upload to session storage" and select the image so it is in the session

def read\_image(filename):

image = plt.imread(filename)

base\_dir = os.path.normpath("/Users/celestecohen/Downloads")

#This would be the Code folder on google drive, where all the other required files are

#Make crop function for scans (only for plates scanned as plate 1 on the JB scanner)

def crop(image):

image=np.delete(image,range(950,1017),0)

image=np.delete(image,range(0,75),0)

image=np.delete(image,range(1400,1432),1)

image=np.delete(image,range(0,90),1)

return image

**1- Creating a labelled binary for 7x7 squares**

#Define each colony as a distinct "object"

def connected\_components(filename, connectivity=2):

# load the image

image = skimage.io.imread(filename)

#Crop the image

image=crop(image)

#blur the image

image = skimage.filters.gaussian(image, sigma=7, truncate=8, channel\_axis=True)

# make an inverted binary

thresh = threshold\_otsu(image)

binary = image > thresh

binary\_mask=np.invert(binary)

#Set the minimum area of an object to 1000 pixels (so it doesn’t pick up noise)

mask = skimage.morphology.remove\_small\_objects(binary\_mask,1000)

# perform connected component analysis (count=number of objects)

labeled\_image, count = skimage.measure.label(mask,connectivity=connectivity, return\_num=True)

regions=regionprops(labeled\_image)

return labeled\_image, count, regions, mask

#you don’t need to return the mask but it can be helpful to visualise it

#labeled\_image is a matrix where each object is numbered 1-96 (the background is 0)

#count is the number of objects there are (numbered 1-96 if there are 96 objects)

#regions group the individual object properties

#mask is the binary mask from which objects were found (a matrix of 1s and 0s)

#Find and label objects in the original JB50 7x7 plate

baseline\_fname = base\_dir + os.sep + "20220627\_1\_biofilm\_JB50\_7x7.jpg"

labeled\_image,count,regions,mask=connected\_components(baseline\_fname)

#Name each object after its position (A1-H12)

def number\_plate(regions):

x\_array=[]

y\_array=[]

for r in regions:

x,y=r["centroid"]

x\_array.append(x)

y\_array.append(y)

rows=pd.cut(x\_array,8, labels=["A","B","C","D","E","F","G","H"])

cols=pd.cut(y\_array,12, labels=["1","2","3","4","5","6","7","8","9","10","11","12"])

index=[]

for i in range(0,len(rows)):

index.append(rows[i]+cols[i])

d = {'x': x\_array, 'y': y\_array, "row": rows, "col":cols, "index": index}

df = pd.DataFrame(data=d)

return df

#Show the dataframe of x (vertical) and y (horizontal) coordinates of the centroid of each object

df=number\_plate(regions)

df

#Make a new binary where the centroid of each square is the average of the centroids on the side of the plate

#Eg centroid of D4 is: x=average x of D1 and D12, y=average y of A4 and H4

#Make each square in the binary 90x90 pixels

#Make a new blank mask m of the same size as the image to add squares to

m=np.zeros((len(mask),len(mask[0])))

for ind in df["index"]:

i=df[df['index']==ind].index[0]

r=df[df["row"]==df.iat[i,2]]

c=df[df["col"]==df.iat[i,3]]

x=st.mean([df.iat[r[r["col"]=="12"].index[0],0],df.iat[r[r["col"]=="1"].index[0],0]])

y=st.mean([df.iat[c[c["row"]=="A"].index[0],1],df.iat[c[c["row"]=="H"].index[0],1]])

minr=int(round(x-45))

minc=int(round(y-45))

maxr=int(round(x+45))

maxc=int(round(y+45))

m[minr:maxr,minc:maxc]=square(90)

show\_image(m)

#Find and label objects in the new binary

labeled\_image, count = skimage.measure.label(m,connectivity=2, return\_num=True)

regions=regionprops(labeled\_image)

#Return a dataframe of x and y coordinates of object centroids

df=number\_plate(regions)

#Make a dictionary of each plate number

plate={}

n=0

for r in range(0,len(regions)):

n=df.iat[r,2]+df.iat[r,3]

plate[n]=regions[r]

**2- Using the binary for image processing**

#Import images

image\_fname = base\_dir + os.sep + "biofilm\_washing\_scans" + os.sep + "20220704\_biofilm\_lib1" + os.sep + "20220704\_1\_biofilm\_lib1.jpg"

image\_1=np.invert(crop(skimage.io.imread(image\_fname)))

image\_fname = base\_dir + os.sep + "biofilm\_washing\_scans" + os.sep + "20220704\_biofilm\_lib1\_w" + os.sep + "20220704\_1\_biofilm\_lib1\_w.jpg"

image\_1\_w=np.invert(crop(skimage.io.imread(image\_fname)))

image\_fname = base\_dir + os.sep +"biofilm\_washing\_scans" + os.sep + "20220704\_biofilm\_lib2" + os.sep + "20220704\_1\_biofilm\_lib2.jpg"

image\_2=np.invert(crop(skimage.io.imread(image\_fname)))

image\_fname = base\_dir + os.sep +"biofilm\_washing\_scans" + os.sep + "20220704\_biofilm\_lib2\_w" + os.sep + "20220704\_1\_biofilm\_lib2\_w.jpg"

image\_2\_w=np.invert(crop(skimage.io.imread(image\_fname)))

image\_fname = base\_dir + os.sep + "biofilm\_washing\_scans" + os.sep + "20220704\_biofilm\_lib3" + os.sep + "20220704\_1\_biofilm\_lib3.jpg"

image\_3=np.invert(crop(skimage.io.imread(image\_fname)))

image\_fname = base\_dir + os.sep +"biofilm\_washing\_scans" + os.sep + "20220704\_biofilm\_lib3\_w" + os.sep + "20220704\_1\_biofilm\_lib3\_w.jpg"

image\_3\_w=np.invert(crop(skimage.io.imread(image\_fname)))

image\_fname = base\_dir + os.sep + "biofilm\_washing\_scans" + os.sep + "20220704\_biofilm\_lib4" + os.sep + "20220704\_1\_biofilm\_lib4.jpg"

image\_4=np.invert(crop(skimage.io.imread(image\_fname)))

image\_fname = base\_dir + os.sep + "biofilm\_washing\_scans" + os.sep + "20220704\_biofilm\_lib4\_w" + os.sep + "20220704\_1\_biofilm\_lib4\_w.jpg"

image\_4\_w=np.invert(crop(skimage.io.imread(image\_fname)))

#Get the layout of the strains, make a vector of strain names in order (by row so A1, A2, etc.)

st\_filename = base\_dir + os.sep + "strains.csv"

strains=pd.read\_csv(st\_filename,header=None)

strains=pd.DataFrame.transpose(strains)

strain\_layout=[]

for i in range(0,8): #get strains in each of 8 columns

strain\_layout.extend(strains[i])

#Create an array of segregant names

segregants=['R' + str(ind) for ind in list(range(1,55))]

#Create an array of mean intensity for each square for "image" from "labeled\_image"

def mean\_int(image):

mi=[]

for i in dict.keys(plate):

m=np.mean(image[np.where(labeled\_image==plate[i]["label"])])

mi.append(m)

return mi

#Make a dataframe with all info for each strain and only keep segregants and B2 (759) and C2 (JB50)

def processing(image,image\_w,rep):

mean\_intensities=mean\_int(image)

mean\_intensities\_w=mean\_int(image\_w)

#Substract the minimum mean intensity (aka just agar no yeast) from all intensities to normalise them

m=min(min(mean\_intensities),min(mean\_intensities\_w))

d = {"rep":rep,

'position': dict.keys(plate),

"x\_coord":[plate[i]["centroid"][1] for i in dict.keys(plate)],

"y\_coord":[plate[i]["centroid"][0] for i in dict.keys(plate)],

"strain":strain\_layout,

'before\_wash': mean\_intensities-m,

"after\_wash": mean\_intensities\_w-m

}

df = pd.DataFrame(data=d)

df["ratio"]=df["after\_wash"]/df["before\_wash"]

df\_seg=df.loc[df['strain'].isin(segregants)]

df\_seg=pd.concat([df.loc[df['position'] == "B2"],df.loc[df['position'] == "C2"],df\_seg])

return df\_seg

#Image processing for each rep

df1=processing(image\_1, image\_1\_w, rep=1)

df2=processing(image\_2, image\_2\_w, rep=2)

df3=processing(image\_3, image\_3\_w, rep=3)

df4=processing(image\_4, image\_4\_w, rep=4)

DF=pd.concat([df1,df2,df3,df4])

import re

#Preparing phenotype data for QTL

DF\_grouped=DF.groupby("strain",as\_index=False).mean()

phenotype=DF\_grouped[["strain","ratio"]]

#Make all segregant names R1\_x (like they are in the genotype data)

for p in range(0,len(phenotype["strain"])):

if phenotype["strain"][p][0]=="R":

phenotype["strain"][p]=str("R1\_"+re.findall("[0-9]+",phenotype["strain"][p])[0])

#Make parental values X968 (JB50) and Y0036 (JB759)

phenotype["strain"][0]="X968"

phenotype["strain"][1]="Y0036"

#phenotype.to\_csv("washing\_phenotypes.csv")

#when you want to actually create the phenotypes csv file, remove the #

**3- Plot**

#Barplot of phenotype values for all strains

plt.rcParams["figure.figsize"] = (30,5.5)

fig, ax = plt.subplots()

strain\_order=np.concatenate((["JB50","JB759"],segregants))

x=list(range(0,len(set(DF['strain']))))

y=[DF.loc[DF["strain"]==name] for name in strain\_order]

colours=np.concatenate((["red","orange"],np.repeat("royalblue",54)))

strain\_order\_official=np.concatenate((["968","Y0036"],segregants))

ax.bar(x,

height=[np.mean(yi["ratio"]) for yi in y],

yerr=[np.std(yi["ratio"]) for yi in y], # error bars

capsize=6, # error bar cap width in points

width=0.8, # bar width,

color=colours,

zorder = 0,

tick\_label=strain\_order\_official

)

ax.set\_xlim(-1,56)

ax.set\_ylabel('Cell–Agar adhesion',fontsize=30,fontname="Arial")

ax.set\_xlabel("Parental strains and segregants",fontsize=30,fontname="Arial")

ax.set\_title("Washing assay",fontsize=40,fontweight="bold",fontname="Arial")

plt.axhline(y=1, color='grey', linestyle='dashed', linewidth=1)

# Scatter plot of Raw Instensity Before and After Wash

plt.rcParams["figure.figsize"] = (7,6)

fig, ax = plt.subplots()

[ax.scatter(y[i]["before\_wash"], y[i]["after\_wash"], color=colours[i], zorder = 1,s=20) for i in range(2,56)]

[ax.scatter(y[i]["before\_wash"], y[i]["after\_wash"], color=colours[i], zorder = 1,s=20) for i in range(0,2)]

ax.set\_ylabel("After Wash")

ax.set\_xlabel("Before Wash")

ax.set\_xlim(0,125)

ax.set\_ylim(0,100)

#Scatter plot of rep1 vs rep2

plt.rcParams["figure.figsize"] = (6,6)

fig, ax = plt.subplots()

x=list(DF.loc[DF["rep"]==2]["ratio"])

y=list(DF.loc[DF["rep"]==4]["ratio"])

[ax.scatter(x[i], y[i], color=colours[i], zorder = 1,s=20) for i in range(2,56)]

ax.scatter(x[0],y[0], color=colours[1], zorder = 1,s=20)

ax.scatter(x[1],y[1], color=colours[0], zorder = 1,s=20)

#I plotted the parentals last so their coloured dots were plotted last therefore weren’t hidden behind the segregant dots

ax.set\_ylabel('Ratio values from Rep 4',fontsize=11)

ax.set\_xlabel("Ratio values from Rep 2",fontsize=11)

ax.set\_xlim(0.15,1.05)

ax.set\_ylim(0.15,1.05)

#Plate scan with coloured squares to show which regions are being studied

fig, ax = plt.subplots()

show\_image(np.invert(image\_1\_w))

#can be done with any of the plate scans

rect=[]

for p in list(df1.loc[df1["strain"]=="JB50"]["position"]):

i=df[df['index']==p].index[0]

r=df[df["row"]==df.iat[i,2]]

c=df[df["col"]==df.iat[i,3]]

x=st.mean([df.iat[r[r["col"]=="12"].index[0],0],df.iat[r[r["col"]=="1"].index[0],0]])

y=st.mean([df.iat[c[c["row"]=="A"].index[0],1],df.iat[c[c["row"]=="H"].index[0],1]])

minr=x-45

minc=y-45

maxr=x+45

maxc=y+45

rect.append(mpatches.Rectangle((minc, minr), maxc - minc, maxr - minr,

fill=False, edgecolor='red', linewidth=3))

for p in list(df1.loc[df1["strain"]=="JB759"]["position"]):

i=df[df['index']==p].index[0]

r=df[df["row"]==df.iat[i,2]]

c=df[df["col"]==df.iat[i,3]]

x=st.mean([df.iat[r[r["col"]=="12"].index[0],0],df.iat[r[r["col"]=="1"].index[0],0]])

y=st.mean([df.iat[c[c["row"]=="A"].index[0],1],df.iat[c[c["row"]=="H"].index[0],1]])

minr=x-45

minc=y-45

maxr=x+45

maxc=y+45

rect.append(mpatches.Rectangle((minc, minr), maxc - minc, maxr - minr,

fill=False, edgecolor='orange', linewidth=3))

for s in segregants:

for p in list(df1.loc[df1["strain"]==s]["position"]):

i=df[df['index']==p].index[0]

r=df[df["row"]==df.iat[i,2]]

c=df[df["col"]==df.iat[i,3]]

x=st.mean([df.iat[r[r["col"]=="12"].index[0],0],df.iat[r[r["col"]=="1"].index[0],0]])

y=st.mean([df.iat[c[c["row"]=="A"].index[0],1],df.iat[c[c["row"]=="H"].index[0],1]])

minr=x-45

minc=y-45

maxr=x+45

maxc=y+45

rect.append(mpatches.Rectangle((minc, minr), maxc - minc, maxr - minr,

fill=False, edgecolor='royalblue', linewidth=3))

for r in rect:

ax.add\_patch(r)

ax.set\_axis\_off()

plt.tight\_layout()

plt.show()