- 03/10/2022

Complete hazard form & send to Somenath!

Useful papers:

-symbac: https://www.biorxiv.org/content/10.1101/2021.07.21.453284v4.abstract

-HTM: https://journals.biologists.com/jcs/article/120/21/3715/29844/High-throughput-microscopy-from-raw-images-to

-segmentation: https://portlandpress.com/essaysbiochem/article/65/1/67/228260/Challenges-of-analysing-stochastic-gene-expression

-distnet: https://link.springer.com/chapter/10.1007/978-3-030-59722-1_21

- 10/10/2022

Michaelmas term project roadmap

Aims to complete by end of week specified

Week 1

Send signed hazard form to safety office

Week 2

Meet other 4th year students on project + Georgeos, read through SyMBac paper and github repo

Get set up in INO-15, get access to analysis machine, play around with generating some data (sparse)

Read about abbe diffraction limit?

Week 3

Start generating data (sparse + MM, fluorescence + phase contrast)

Read about Otsu's method for segmentation, begin efforts to do some segmentation on data

Research any other classical methods of segmentation that could be used

Ask some friends to try identification + segmentation to check human performance

Week 4

Finish with Otsu + any other, begin trying omnipose/delta

Week 5

Attempt to finish characterising segmentation performance using omnipose/delta

Read into superresolution techniques/algorithms

Week 6

Week 7

Week 8

- 17/10/2022

SYMBAC GROUP MEETING

- Run rigid body simulation with some parameters eg cell width and length
- Generate PSF using microscope parameters eg NA, n
- Input real image, use napari to select media, cell, device pixels
- Interactive sliders to optimise synthetic image parameters, try to match visual properties of real images as much as possible

- 24/10/2022

PSF models/generator:

https://github.com/tlambert03/psfmodels

http://bigwww.epfl.ch/algorithms/psfgenerator/

FIJI TUTORIALS: https://www.youtube.com/c/haesleinhuepf

PPT Script

Background

High-throughput microscopy allows researchers to acquire images automatically from thousands of different treatments overnight or over several days. This makes it possible to conduct large-scale, image-based screens to discover novel genes and novel functions of familiar genes [1].

These experiments produce a lot of data in the form of images which then need to be analysed. Classification and segmentation are important steps in the analysis process; classification essentially puts a box around part of an image and says, “there is a cell here”, while segmentation creates a binary mask, separating background from foreground and foreground objects from each other. Segmentation and classification algorithms perform better on higher resolution images than lower resolution images, but this creates an issue for the throughput of the experiment. A screening experiment done in a mother machine might be tracking 10^5 trenches in parallel, but if each image must be taken at a high resolution to allow for good classification/segmentation performance, more images must be taken to cover every trench in the mother machine, and each individual trench will be imaged less frequently.

Spatial temporal & throughput tradeoff

Objectives

We see that higher spatial resolution leads to reduced temporal resolution for each individual trench of cells which causes lower data throughput (which makes it more difficult to track cells between concurrent images of an experiment), and vice versa. Therefore, a thorough investigation into the

performance of classification/segmentation algorithms with respect to image resolution should be carried out to get a good idea of how to navigate this trade-off effectively.

The ideal endpoint of this investigation will be to be able to produce some charts with some accuracy score for the algorithm on the y-axis against the image resolution/magnification on the x-axis. A chart of this kind could be produced to compare different algorithms, different experimental setups (sparse, MM, 2D, agar) and phase-contrast vs fluorescence.

Check performance on real data.

Once this relationship has been established, we wish to investigate to what extent using a deep-learning super-resolution algorithm can improve the relationship.

Plan

For the initial phase of the investigation, begin with one segmentation algorithm, using phase contrast and air objective. Use SyMBac to generate data at lots of different resolutions (e.g. starting at 5x and going up in multiples of 5 to 100x?). To start relatively simple, algorithm to use will be Otsu with watershed. Apply this to the generated data (hopefully a wide range of images to get a representative result) and compare with ground truth provided by SyMBac to produce the required chart. Repeat with the same data for different algorithms eg BACMMAN, DeLTA and generate charts for those algorithms. New data will only have to be generated when changing one of the experiment parameters eg PC vs fluorescence and air vs oil. The time taken to produce one of these graphs should decrease as more are produced, so the aim will be to finish this investigation sometime during the Christmas holidays at the latest, allowing for some delays and hiccups along the way.

Lent term can then be focused on investigating neural network architectures that can give good super-resolution performance for this specific application of images of bacteria. The current plan is to start off with a basic architecture known as SRCNN (super-resolution convolutional neural network) which implements bicubic interpolation up-sampling of the low-resolution image to the same size as the high-resolution image as the input, and then learns a mapping between the up-sampled image and the high-resolution image. Improvements in performance can then be characterised by generating new graphs of classification performance vs image resolution.

Using this performance as a benchmark, other architectures can be explored. One idea could be to go from the up-sampled image directly to the desired output of the classification/segmentation algorithm (i.e., some form of binary mask). Another idea could be to train a network that takes the low-resolution image as input without any up-sampling step, so it can try to learn a better up sampling process than bicubic interpolation.

This will likely take up the majority of Lent term, but hopefully good results will be able to be achieved by the end of term. This would leave the post-exam part of Easter term for any extension of the project that seems feasible to accomplish within that time frame, such as a preliminary investigation into improving temporal resolution, or trying more different neural net architectures.

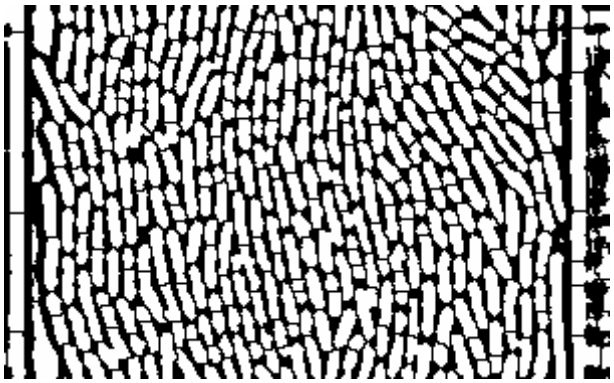
Work so far

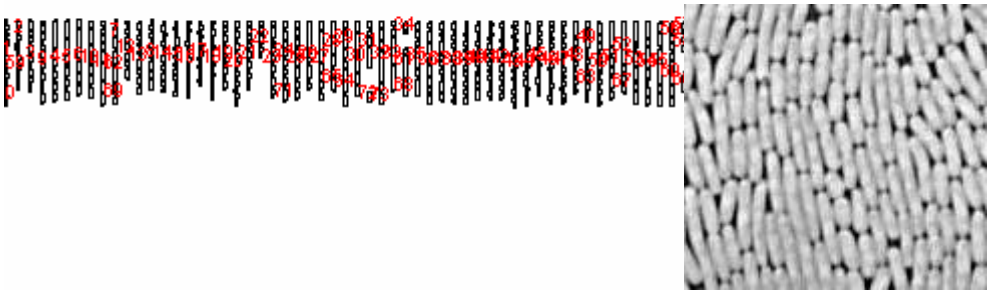
Used Fiji to do some segmentation tasks on some images with different resolutions

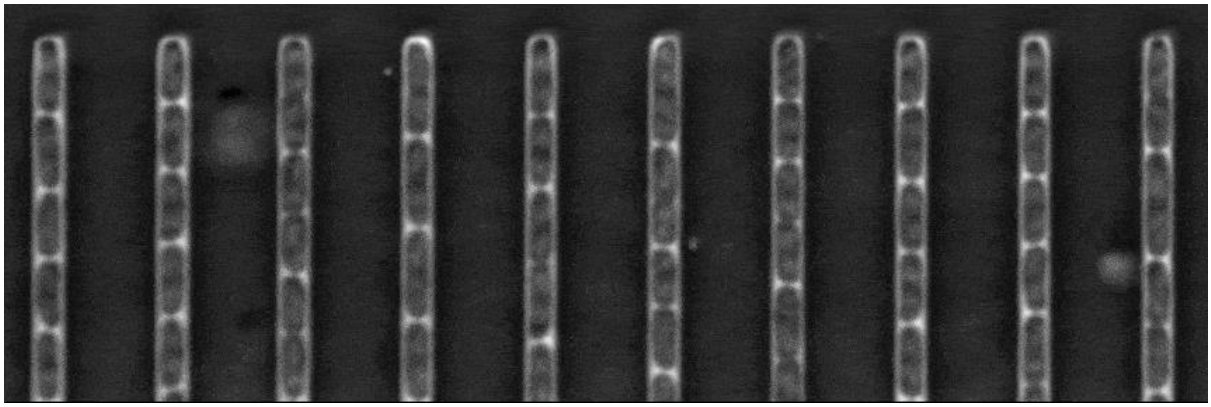
References

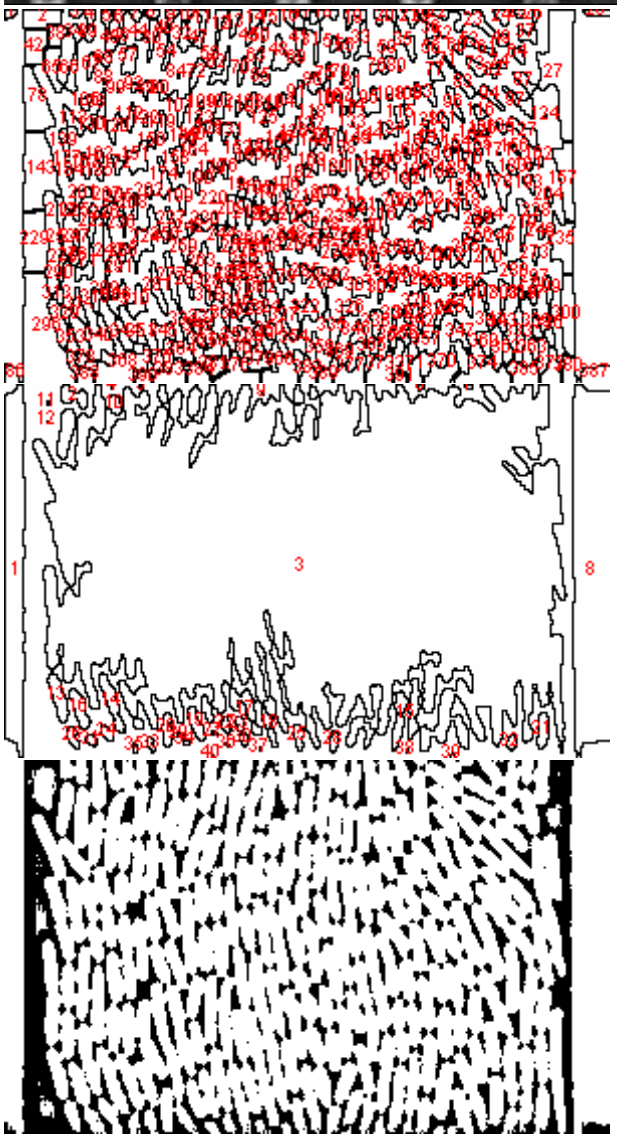
[1] (Wollman & Stuurman, 2007)

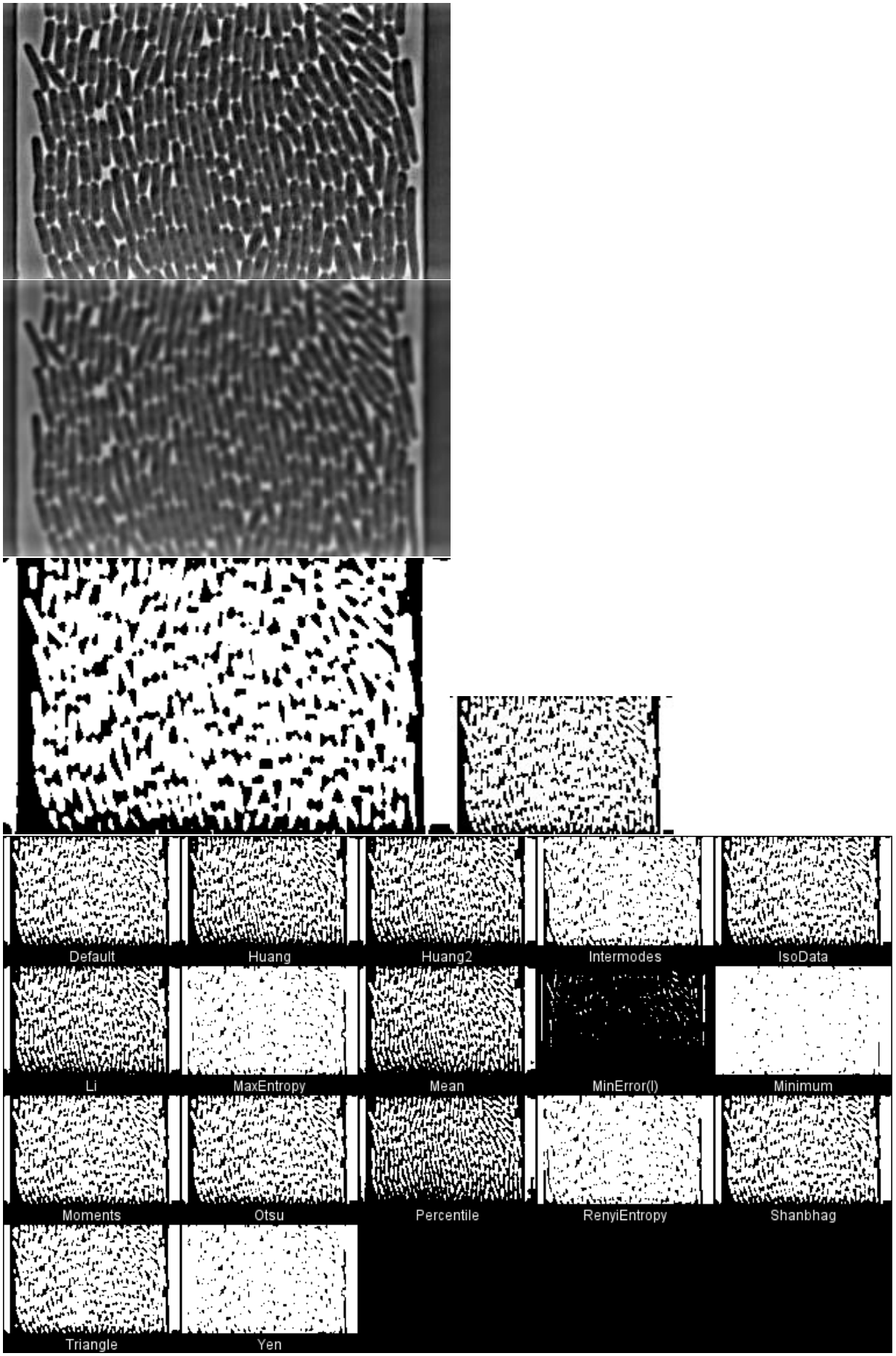
- 31/10/2022

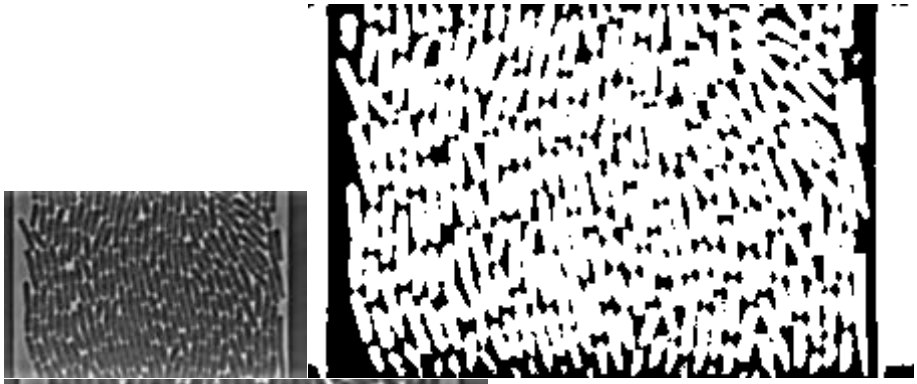


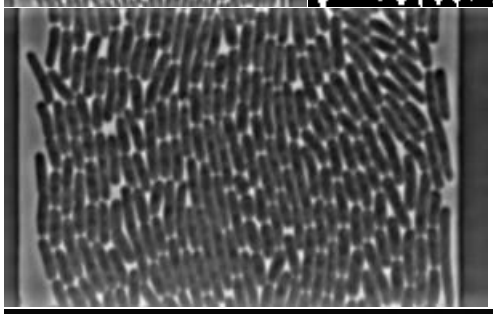


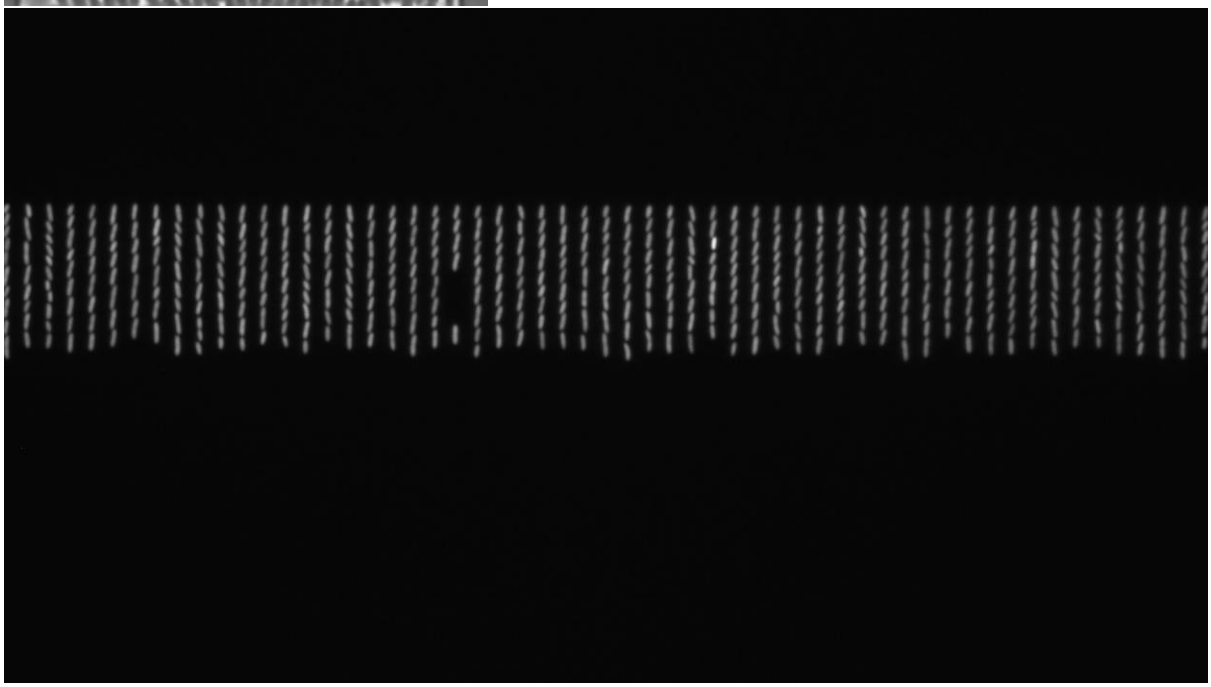



1
3
8


Default
Huang
Huang2
Intermodes
IsoData
Li
MaxEntropy
Mean
MinError(I)
Minimum
Moments
Otsu
Percentile
RenyiEntropy
Shanbhag
Triangle
Yen







- 07/11/2022

Meet with Georgeos to help with symbac setup – having a bunch of issues trying to get it working

Calculating id accuracy/iou for mother machine images??

Over-segmentation leads to more masks than ground truths, under-segmentation leads to less masks than ground truths

Counting number of masks can tell about id accuracy but some over seg and underseg errors can cancel making it look better when its actually bad

```
def id_accuracy(groundarray, imagearray, image_info=False):
    """return identification accuracy of image compared to ground truth image

    Arguments:
        groundarray -- ground truth image, as a numpy array
        imagearray -- image to find the error of, as a numpy array

    Returns:
        out[0] - num_identified/num_cells < 1
        out[1] - num_identified/num_cells = 1
        out[2] - num_cells/num_identified < 1
    """
    out = np.zeros(3)
    num_cells = np.max(groundarray)
    if image_info:
        num_identified = len(imagearray[0])
    else:
        num_identified = np.max(imagearray)
    if num_identified < num_cells:
        out[0] = num_identified/num_cells
    elif num_identified == num_cells:
        out[1] = 1
    elif num_identified > num_cells:
        out[2] = num_cells/num_identified
    return out
```

for now just keep track of it, can see later ways to improve id accuracy measure

first iou measure:

```
def pixel_accuracy(groundarray, imagearray):
    """find error rate of accurately segmented pixels in imagearray given the
ground truth groundarray

    Arguments:
        groundarray -- ground truth image, as a numpy array
        imagearray -- image to find the error of, as a numpy array

    Returns:
        segmentation pixel accuracy (false negatives) - how many pixels were
correct out of total number of pixels classified as a cell
    """
    # masks displayed by distinct non-zero integers
    max = np.max(groundarray) + 1
    # total number of mask pixels
    total_mask_pixels = groundarray[groundarray > 0].size
    tot_correct = 0
```

```
    previous_values = []
    for i in range(1, max):
        # array indices of cell masks in ground truth array
        coords = np.where(groundarray == i)
        # equivalent region in image being tested
        test_area = imagearray[coords]
        equivalent_value = stats.mode(test_area)[0][0]
        if equivalent_value not in previous_values:
            previous_values.append(equivalent_value)
        # get the count of the mode number in the region
        # (since masks don't necessarily have the same number)
            test_area = test_area[test_area > 0]
            if len(test_area) > 0:
                tot_correct += stats.mode(test_area)[1][0]
    return tot_correct/total_mask_pixels
```

gives precision of image, TP/(TP+FP)

- 14/11/2022

identification error: error in numbers detected

segmentation error: give threshold for accuracy (eg most of core pixels correct) - more important to see if cells were incorrectly merged or one cell was incorrectly split

can then classify using:

- global threshold

- local threshold

- + watershed

- interactive watershed

- stardist?

- trained agorithm

might be able to create a basic version of symbac (just create ground truth circles/cylinders then convolve with PSF (approximate with 2D Gaussian) etc.)

use fiji analyze>set measurements to see area, centroid locations, length and width, etc for use when doing analyze particles

- 21/11/2022

Presentations.

Initial feedback:

“the plan for the slides looks good to me. nicely introduces the background, current investigations, and plans to investigate further into different imaging modalities and platforms and also for

superresolution. I wonder if the superresolution bit should come at the end to improve the flow. Looking forward to the slides when you place the images in them."

Second feedback:

"looks very good. Try to reduce the text per slide. Now that have you have made the slides you can roughly remember what you need to say, so don't put them on the slides. just keep short phrases as pointers to what you are about to say and then let them hear it from you, not read it. Rehearse the talk to make sure the flow is good."

- 28/11/2022

- 05/12/2022

Symbac simulations started working but now optimisation is not working ://

- 12/12/2022

Phd apps + covid ://

- 19/12/2022

SYMBAC PARAMS:

The two main objectives we user are:

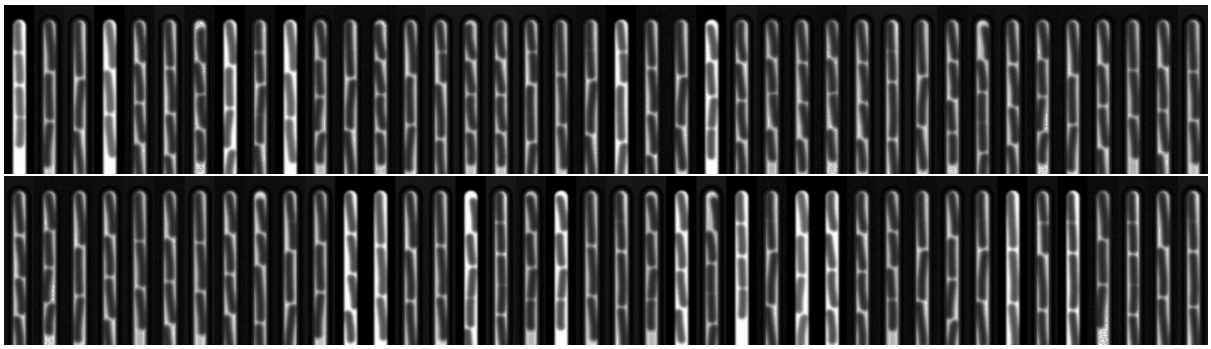
60x: 0.01 um/pix, Ph2 condenser ring, 0.95 NA, refractive index n = 1

100x: 0.0655 um/pix, Ph3 condenser ring, 1.45 NA, refractive index n = ~1.5

Use a resize amount of 3, wavelength of 0.6 um, The apo sigma argument is not well known, so you need to try various values (between 5-40) to get the best looking image.

- 26/12/2022

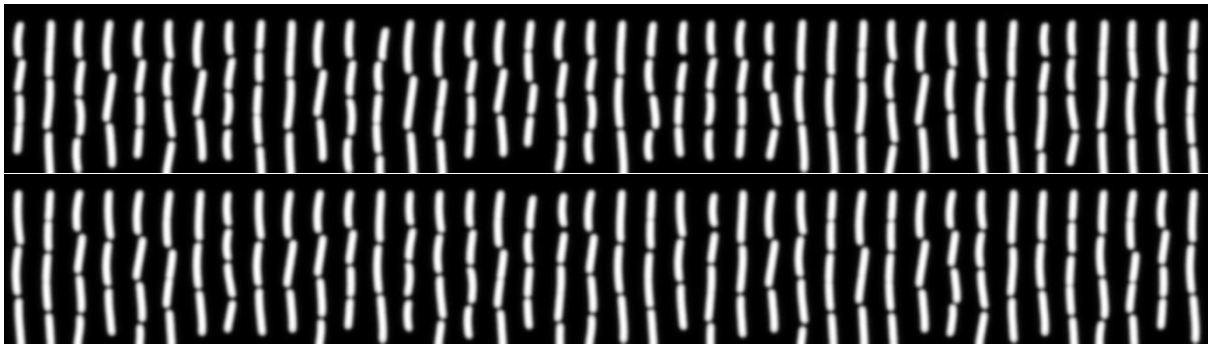
SYMBAC WORKING LETS GOOOOOOOO



- 02/01/2023

Training omnipose is proving a bit difficult, taking absolutely ages and not really segmenting properly

Might have to use stardist on fluorescence symbac for segmentation for TMR, then look into omnipose and try to fix it after



- 09/01/2023

TMR draft feedback

- Need a figure to explain the three-way trade-off in section 2.1
- Section on the mother machine seems disconnected from the rest. You need to motivate why you are inroducing/discussing mother machine.
- Give title to each figure caption to describe what the figure is about in a single sentence.
- Clearly distinguish two types of test data and their trade-offs (images from experiments and synthetic images from symbac pipeline)
- Regarding section 4.2, have a chat with Erez about the option to use SyMBac with pinching simulations included (if you have time)
- Try to use short sections with bullet points in the future direction. That way it's easier to see what's left to be done.

- 16/01/2023

TODO for tmr and start of term

- add the triangle next to figure 1

- remove some detail from fig 3 just long arrow left to right

- figure 4 orient pictures so mm is same way

- figure 5: get rid of top left panel and bottom right, mention in text

that iou and segmentation are similar tho not the same so other two

plots are bigger

- show some synthetic images, will collect real data at various objectives

to compare to symbac images to make them more realistic and refine camera

model etc say that images have room to be improved in this way.

- maybe merge the two bits of fig 5 and fig 6 into one

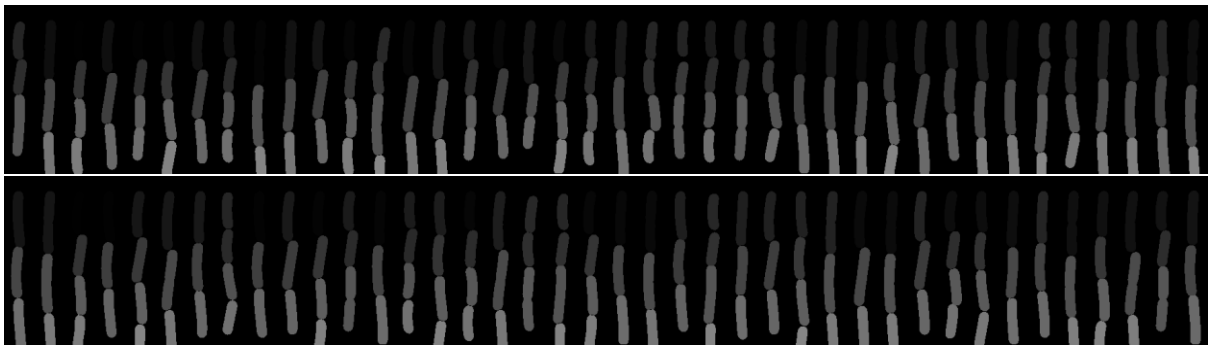
- check for mask sizes - are errors coming from not enough blurring or what

- 23/01/2023

OMNIPOSE WORKING

Issues related to how the gpu was being used, plus some random bugs when working with window/my laptop

Also training is massively sped up by using desktop fan to cool down laptop and keep gpu temps reasonable



- 30/01/2023

FIJI MACROS FOR SEGMENTING FL IMAGES

BERNSEN:

```
// Ask for the input folder

input = getDirectory("Choose the Input Folder");


//Ask for the output folder

output = getDirectory("Choose the Output Folder")


processFolder(input);


// function to scan folders/subfolders/files to find files with correct suffix

function processFolder(input) {
	list = getFileList(input);
	list = Array.sort(list);
	for (i = 0; i < list.length; i++) {


	// Do the processing here by adding your own code.
	// Leave the print statements until things work, then remove them.
```

```
	//@ File(style="directory") outputDirectory
open(input + list[i]);
//run("Properties...", "channels=1 slices=1 frames=1 pixel_width=0.6473791 pixel_height=0.6473791 voxel_depth=1.0000000 global");
myDir=getDirectory("image");
name1=getTitle();
name2=File.nameWithoutExtension;
run("Duplicate...", "title=[Raw Image]");

setOption("ScaleConversions", true);
run("8-bit");
run("Auto Local Threshold", "method=Bernsen radius=7 parameter_1=0 parameter_2=0 white");

rename(name1);

saveAs("Tiff", output+name2+"Bernsen"+".tif");

close();
close();
	print("Processing: " + input + list[i]);
	print("Saving to: " + output);
}
```

WATERSHED:

```
// Ask for the input folder
input = getDirectory("Choose the Input Folder");

//Ask for the output folder
output = getDirectory("Choose the Output Folder")

processFolder(input);
```

```
// function to scan folders/subfolders/files to find files with correct suffix
function processFolder(input) {
	list = getFileList(input);
	list = Array.sort(list);
	for (i = 0; i < list.length; i++) {

	// Do the processing here by adding your own code.
	// Leave the print statements until things work, then remove them.
	//@ File(style="directory") outputDirectory
open(input + list[i]);
//run("Properties...", "channels=1 slices=1 frames=1 pixel_width=0.6473791 pixel_height=0.6473791 voxel_depth=1.0000000 global");
myDir=getDirectory("image");
name1=getTitle();
name2=File.nameWithoutExtension;
run("Duplicate...", "title=[Raw Image]");

setOption("ScaleConversions", true);
run("8-bit");
run("Auto Local Threshold", "method=Bernsen radius=7 parameter_1=0 parameter_2=0 white");
run("Invert");
run("Watershed");
run("Invert");

rename(name1);

saveAs("Tiff", output+name2+"watershed"+".tif");

close();
close();
	print("Processing: " + input + list[i]);
```

```
	print("Saving to: " + output);
}
```

STARDIST

```
// Ask for the input folder
input = getDirectory("Choose the Input Folder");

//Ask for the output folder
output = getDirectory("Choose the Output Folder")

processFolder(input);

// function to scan folders/subfolders/files to find files with correct suffix
function processFolder(input) {
	list = getFileList(input);
	list = Array.sort(list);
	for (i = 0; i < list.length; i++) {

	// Do the processing here by adding your own code.
	// Leave the print statements until things work, then remove them.
	//@ File(style="directory") outputDirectory
open(input + list[i]);
//run("Properties...", "channels=1 slices=1 frames=1 pixel_width=0.6473791 pixel_height=0.6473791 voxel_depth=1.0000000 global");
myDir=getDirectory("image");
name1=getTitle();
name2=File.nameWithoutExtension;
run("Duplicate...", "title=[Raw Image]");
run("Command From Macro", "command=[de.csbdresden.stardist.StarDist2D], args=['input':'Raw Image', 'modelChoice':'Versatile (fluorescent nuclei)', 'normalizeInput':'true', 'percentileBottom':'1.0', 'percentileTop':'99.8', 'probThresh':'0.5', 'nmsThresh':'0.4', 'outputType':'Both', 'nTiles':'1', 'excludeBoundary':'2', 'roiPosition':'Automatic', 'verbose':'false', 'showCsbdeepProgress':'false', 'showProbAndDist':'false'], process=[false]");
selectWindow("Label Image");
```

```
rename(name1);
run("Set Measurements...", "area area_fraction limit display redirect=None decimal=3");
roiManager("Measure");
saveAs("Tiff", output+name2+"StarDist"+".tif");
roiManager("delete");
close("ROI Manager");
run("Clear Results");
close("Results");
close();
close();
close();
        print("Processing: " + input + list[i]);
        print("Saving to: " + output);
}
```

OTSU

```
// Ask for the input folder
input = getDirectory("Choose the Input Folder");

//Ask for the output folder
output = getDirectory("Choose the Output Folder")

processFolder(input);

// function to scan folders/subfolders/files to find files with correct suffix
function processFolder(input) {
        list = getFileList(input);
        list = Array.sort(list);
        for (i = 0; i < list.length; i++) {

        // Do the processing here by adding your own code.
        // Leave the print statements until things work, then remove them.
```

```
	//@ File(style="directory") outputDirectory
open(input + list[i]);
//run("Properties...", "channels=1 slices=1 frames=1 pixel_width=0.6473791 pixel_height=0.6473791 voxel_depth=1.0000000 global");
myDir=getDirectory("image");
name1=getTitle();
name2=File.nameWithoutExtension;
run("Duplicate...", "title=[Raw Image]");

setOption("ScaleConversions", true);
run("16-bit");

run("Auto Threshold", "method=Otsu white");

rename(name1);

saveAs("Tiff", output+name2+"otsu"+".tif");

close();
close();
	print("Processing: " + input + list[i]);
	print("Saving to: " + output);
}
```

- 06/02/2023

SIMULTANEOUS SYMBAC:

media_multiplier	generate_PC_OPL
cell_multiplier	generate_PC_OPL
device_multiplier	generate_PC_OPL
sigma	inside PSF loop
scene_no	generate_PC_OPL
match_fourier	inside PSF loop

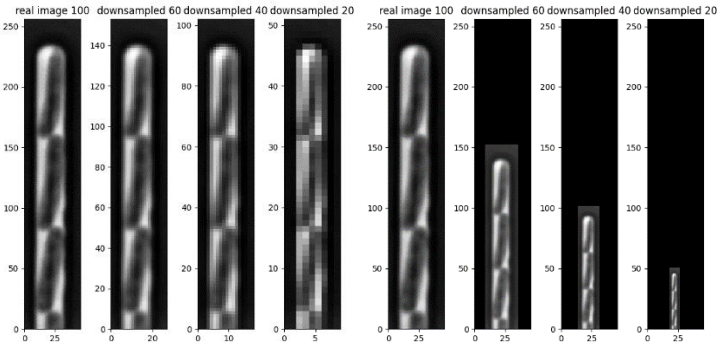
match_histogram	inside PSF loop
match_noise	inside PSF loop
noise_var	inside PSF loop
defocus	generate_PC_OPL

to newly compare seg methods on fluo gotta test it on individual trench images

as well as the concatenated tile images

GOTTA DO THIS ASAP PLEASE LET IT BE REASONABLE!!!^^^

SIMULTANEOUS IS WORKING



- 13/02/2023

Most likely will forego ID accuracy measurements, doesn't seem very useful and can get all information from iou plots

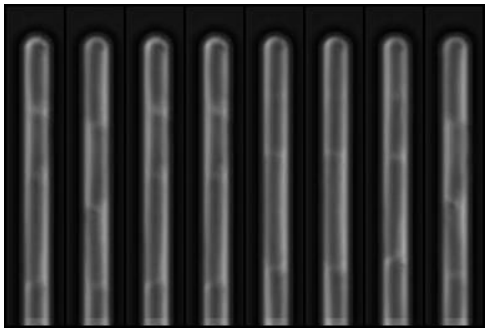
New iou code:

```
def IoU(groundarray, imagearray):
    """find error rate of accurately segmented pixels in imagearray given the
ground truth groundarray

    Arguments:
        groundarray -- ground truth image, as a numpy array
        imagearray -- image to find the error of, as a numpy array

    Returns:
        segmentation pixel accuracy (false negatives) - how many pixels were
correct out of total number of pixels classified as a cell
    """
```

```
    # masks displayed by distinct non-zero integers
    max = int(np.max(groundarray) + 1)
    # total number of mask pixels
    previous_values = []
    iou_vals = []
    sizes = []
    for i in range(1, max):
        if np.isin(i,groundarray):
            # array indices of cell masks in ground truth array
            coords = np.where(groundarray == i)
            sizes.append(len(coords[0]))
            # equivalent region in image being tested
            test_area = imagearray[coords]
            equivalent_value = stats.mode(test_area)[0][0]
            previous_values.append(equivalent_value)
            image_coords = np.where((imagearray == equivalent_value))
            if len(coords) > 0:
                iou_vals.append(andoveror(coords,image_coords))
    unq, counts = np.unique(previous_values,return_counts=True)
    iou_vals = np.array(iou_vals)
    pv = np.array(previous_values)
    out = np.array(iou_vals)
    for i, val in enumerate(unq):
        if counts[i] > 1:
            dodgy_vals = iou_vals[np.where(pv==val)]
            perc = dodgy_vals[:,0]/dodgy_vals[:,1]
            fak = dodgy_vals[np.argsort(perc)][:-1]
            for fake in fak:
                out[np.all(np.equal(out,fake),axis=-1)] = [0,sizes[i]]
    return out
```

SRCNN WORKING

Really good iou results for segmentation, but visually SR images look so weird, something strange here



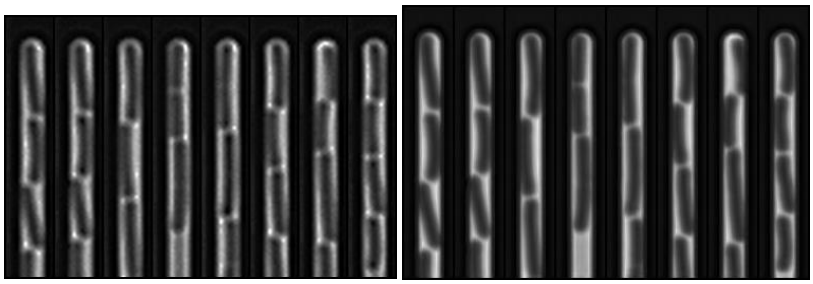
- 20/02/2023

Requirements for srcnn:

- python = 3.9.0
- conda install pytorch==1.12.1 torchvision==0.13.1 torchaudio==0.12.1 cudatoolkit=11.6 -c pytorch -c conda-forge
- pip install opencv-python = 4.7.0.68
- pip install matplotlib = 3.7.0
- pip install tqdm = 4.64.1

second batch working much better, bit better iou results but this time the SR output looks much more reasonable

also made the srcnn save the input and output images for validation to keep better track



- 27/02/2023

Updated Interpolation for images:

```
def linear_interpolate(original_array, target_shape,method='linear'):
    ts0, ts1 = target_shape
    ogarr = np.array(original_array)
    os0, os1 = ogarr.shape
    if target_shape == ogarr.shape:
        return ogarr
    methods = ['linear','nearest','slinear','cubic','quintic','pchip']
    if method not in methods:
        print(f"{method} is not a valid method. Valid methods are: {methods}.
\n Defaulting to linear interpolation.")
        method='linear'
    x, y = np.arange(os0), np.arange(os1)
    grid = RegularGridInterpolator((x,y), ogarr,method=method)
    a, b = np.meshgrid(np.linspace(0,int(os0-1),int(ts0)),
np.linspace(0,int(os1-1),int(ts1)),indexing='ij')
    points =
np.concatenate((a.reshape(int(ts0),int(ts1),1),b.reshape(int(ts0),int(ts1),1))
,axis=-1)
    newarr = grid(points)
    return newarr
```

- 06/03/2023

MADE FUNCTIONS TO DO EVERYTHING FROM TILING TRAINING SR MODEL TO MAKING SR IMAGES FOR TEST DATA AND CREATING TILED IMAGES FOR TRAINING OMNIPOSE AND THEN EVALUATING THE IOU life is now much easier

```
import tifffile
from PIL import Image
import numpy as np
import torch
import matplotlib.pyplot as plt
import os
import random
from glob import glob
from natsort import natsorted
from interpolate import linear_interpolate
from cellpose_omni import models, core
from cellpose_omni import plot
from error_algorithms import IoU
from error_algorithms import centroid_distances
from error_algorithms import ignore_duplicates
from piqa import SSIM, PSNR
from tqdm import tqdm

def open_image(filename):
    if "tif" in filename:
        img = np.array(tifffile.imread(filename),dtype=np.float32)
    elif "png" in filename:
        img = np.array(Image.open(filename),dtype=np.float32)
    else:
        raise TypeError(f"Images must be tif files or png files, not
{filename[-3:]}")
    return img

def do_everything_before(SR_TRAINING_IMAGES_DIR, HR_FOLDER_NAME,
LR_FOLDER_NAME,METHOD="linear"):
    methods = ['linear','nearest','slinear','cubic','quintic','pchip']
    if METHOD not in methods:
        print(f"{METHOD} is not a valid method. Valid methods are: {methods}")
        raise TypeError
    HR_TRAIN_DIR = SR_TRAINING_IMAGES_DIR + "/train/" + HR_FOLDER_NAME +
"/convolutions/"
    HR_TEST_DIR  = SR_TRAINING_IMAGES_DIR + "/test/"  + HR_FOLDER_NAME +
"/convolutions/"
    LR_TRAIN_DIR = SR_TRAINING_IMAGES_DIR + "/train/" + LR_FOLDER_NAME +
"/convolutions/"
    LR_TEST_DIR  = SR_TRAINING_IMAGES_DIR + "/test/"  + LR_FOLDER_NAME +
"/convolutions/"
    hr_image = open_image(HR_TRAIN_DIR+os.listdir(HR_TRAIN_DIR)[0])
```

```
    UPSAMPLED_TRAIN_DIR = SR_TRAINING_IMAGES_DIR + "/train/" + LR_FOLDER_NAME
+ "_" + METHOD + "/"
    UPSAMPLED_TEST_DIR  = SR_TRAINING_IMAGES_DIR + "/test/"  + LR_FOLDER_NAME
+ "_" + METHOD + "/"
    SAVE_OUTPUT_DIR     = SR_TRAINING_IMAGES_DIR + "/train/" + LR_FOLDER_NAME
+ "_" + METHOD + "_output/"
    try:
        os.mkdir(UPSAMPLED_TRAIN_DIR)
    except FileExistsError:
        print(f"folder already exists: {UPSAMPLED_TRAIN_DIR}")
    try:
        os.mkdir(UPSAMPLED_TEST_DIR)
    except FileExistsError:
        print(f"folder already exists: {UPSAMPLED_TEST_DIR}")
    for file in os.listdir(LR_TRAIN_DIR):
        lr_image = open_image(LR_TRAIN_DIR+file)
        upsampled =
np.rint(linear_interpolate(lr_image,target_shape=hr_image.shape,method=METHOD)
)//256
        upsampled = Image.fromarray(upsampled).convert("L")
        upsampled.save(UPSAMPLED_TRAIN_DIR+file[:-3]+"png")
    for file in os.listdir(LR_TEST_DIR):
        lr_image = open_image(LR_TEST_DIR+file)
        upsampled =
np.rint(linear_interpolate(lr_image,target_shape=hr_image.shape,method=METHOD)
)//256
        upsampled = Image.fromarray(upsampled).convert("L")
        upsampled.save(UPSAMPLED_TEST_DIR+file[:-3]+"png")
    return HR_TRAIN_DIR, HR_TEST_DIR, UPSAMPLED_TRAIN_DIR, UPSAMPLED_TEST_DIR,
SAVE_OUTPUT_DIR

def get_train_command(NUM_EPOCHS, HR_TRAIN_DIR, HR_TEST_DIR,
UPSAMPLED_TRAIN_DIR, UPSAMPLED_TEST_DIR, SAVE_OUTPUT_DIR):
    command = f'python train.py --epochs {NUM_EPOCHS} --hr-path
"{HR_TRAIN_DIR}" --lr-path "{UPSAMPLED_TRAIN_DIR}" --hr-validation-path
"{HR_TEST_DIR}" --lr-validation-path "{UPSAMPLED_TEST_DIR}" --save-path
"{SAVE_OUTPUT_DIR}"'
    return command

def do_everything_between(SEGMENTATION_IMAGES_DIR, HR_FOLDER_NAME,
LR_FOLDER_NAME,
SR_MODEL=None,METHOD="linear",HR_ONLY=False,TILE_LENGTH=40,TRAINING_SAMPLES=20
0,SR_PREFIX="SR",device=torch.device("cuda")):
    if  HR_ONLY:
        HR_TRAIN_DIR_MASKS = SEGMENTATION_IMAGES_DIR + "/train/" +
HR_FOLDER_NAME + "/masks/"
        HR_TEST_DIR_MASKS  = SEGMENTATION_IMAGES_DIR + "/test/"  +
HR_FOLDER_NAME + "/masks/"
```

```
        HR_TRAIN_DIR_CONVS = SEGMENTATION_IMAGES_DIR + "/train/" +
HR_FOLDER_NAME + "/convolutions/"
        HR_TEST_DIR_CONVS  = SEGMENTATION_IMAGES_DIR + "/test/"  +
HR_FOLDER_NAME + "/convolutions/"
        HR_TILED_TRAIN_DIR = SEGMENTATION_IMAGES_DIR + "/train/" +
HR_FOLDER_NAME + f"_tiled/"
        HR_TILED_TEST_DIR  = SEGMENTATION_IMAGES_DIR + "/test/"  +
HR_FOLDER_NAME + f"_tiled/"
        HR_TILED_TEST_SEG_DIR  = SEGMENTATION_IMAGES_DIR + "/test/"  +
HR_FOLDER_NAME + f"_tiled_segmentations/"

        try:
            os.mkdir(HR_TILED_TRAIN_DIR)
        except FileExistsError:
            print(f"folder already exists: {HR_TILED_TRAIN_DIR}")
        try:
            os.mkdir(HR_TILED_TEST_DIR)
        except FileExistsError:
            print(f"folder already exists: {HR_TILED_TEST_DIR}")

        MASKS_TRAIN = sorted(glob(HR_TRAIN_DIR_MASKS+"/*"))
        MASKS_TEST  = sorted(glob(HR_TEST_DIR_MASKS+"/*"))
        CONVS_TRAIN = sorted(glob(HR_TRAIN_DIR_CONVS+"/*"))
        CONVS_TEST  = sorted(glob(HR_TEST_DIR_CONVS+"/*"))

        TRAIN_INDICES = random.sample(range(len(MASKS_TRAIN)-TILE_LENGTH),
TRAINING_SAMPLES)
        TEST_SAMPLES = len(MASKS_TEST)//TILE_LENGTH
        TEST_INDICES = np.linspace(0,TILE_LENGTH*(TEST_SAMPLES-
1),TEST_SAMPLES).astype(int)

        try:
            os.mkdir(HR_TILED_TEST_SEG_DIR)
        except FileExistsError:
            print(f"folder already exists: {HR_TILED_TEST_SEG_DIR}")

        for i, x in enumerate(TRAIN_INDICES):
            x = TRAIN_INDICES[i]
            mask_tile = np.concatenate([np.array(Image.open(mask)) for mask in
MASKS_TRAIN[x:x+TILE_LENGTH]], axis=1)
            conv_tile = np.concatenate([np.array(Image.open(conv)) for conv in
CONVS_TRAIN[x:x+TILE_LENGTH]], axis=1)
            Image.fromarray(mask_tile).save(f"{HR_TILED_TRAIN_DIR}/train_{str(
i).zfill(5)}_masks.png")
            Image.fromarray(conv_tile).save(f"{HR_TILED_TRAIN_DIR}/train_{str(
i).zfill(5)}.png")
        for i, x in enumerate(TEST_INDICES):
            x = TEST_INDICES[i]
```

```
            mask_tile = np.concatenate([np.array(Image.open(mask)) for mask in
MASKS_TEST[x:x+TILE_LENGTH]], axis=1)
            conv_tile = np.concatenate([np.array(Image.open(conv)) for conv in
CONVS_TEST[x:x+TILE_LENGTH]], axis=1)
            Image.fromarray(mask_tile).save(f"{HR_TILED_TEST_DIR}/test_{str(i)
.zfill(5)}_masks.png")
            Image.fromarray(conv_tile).save(f"{HR_TILED_TEST_DIR}/test_{str(i)
.zfill(5)}.png")
        return HR_TILED_TRAIN_DIR, HR_TILED_TEST_DIR, HR_TILED_TEST_SEG_DIR
    elif SR_MODEL is None and not HR_ONLY:
        methods = ['linear','nearest','slinear','cubic','quintic','pchip']
        if METHOD not in methods:
            print(f"{METHOD} is not a valid method. Valid methods are:
{methods}")
            raise TypeError
        HR_TRAIN_DIR = SEGMENTATION_IMAGES_DIR + "/train/" + HR_FOLDER_NAME +
"/masks/"
        HR_TEST_DIR  = SEGMENTATION_IMAGES_DIR + "/test/"  + HR_FOLDER_NAME +
"/masks/"
        LR_TRAIN_DIR = SEGMENTATION_IMAGES_DIR + "/train/" + LR_FOLDER_NAME +
"/convolutions/"
        LR_TEST_DIR  = SEGMENTATION_IMAGES_DIR + "/test/"  + LR_FOLDER_NAME +
"/convolutions/"
        if len(os.listdir(HR_TRAIN_DIR)) != len(os.listdir(LR_TRAIN_DIR)):
            print(f"HR and LR training folders should be the same length, but
have lengths {len(os.listdir(HR_TRAIN_DIR))},
{len(os.listdir(LR_TRAIN_DIR))}")
            raise ValueError
        if len(os.listdir(HR_TEST_DIR)) != len(os.listdir(LR_TEST_DIR)):
            print(f"HR and LR training folders should be the same length, but
have lengths {len(os.listdir(HR_TRAIN_DIR))},
{len(os.listdir(LR_TRAIN_DIR))}")
            raise ValueError
        hr_image = open_image(HR_TRAIN_DIR+os.listdir(HR_TRAIN_DIR)[0])
        UPSAMPLED_TRAIN_DIR       = SEGMENTATION_IMAGES_DIR + "/train/" +
LR_FOLDER_NAME + f"_{METHOD}/"
        UPSAMPLED_TEST_DIR        = SEGMENTATION_IMAGES_DIR + "/test/"  +
LR_FOLDER_NAME + f"_{METHOD}/"
        UPSAMPLED_TILED_TRAIN_DIR = SEGMENTATION_IMAGES_DIR + "/train/" +
LR_FOLDER_NAME + f"_{METHOD}_tiled/"
        UPSAMPLED_TILED_TEST_DIR  = SEGMENTATION_IMAGES_DIR + "/test/"  +
LR_FOLDER_NAME + f"_{METHOD}_tiled/"
        UPSAMPLED_TILED_TEST_SEG_DIR  = SEGMENTATION_IMAGES_DIR + "/test/"  +
LR_FOLDER_NAME + f"_{METHOD}_tiled_segmentations/"

        try:
            os.mkdir(UPSAMPLED_TRAIN_DIR)
        except FileExistsError:
```

```
            print(f"folder already exists: {UPSAMPLED_TRAIN_DIR}")
        try:
            os.mkdir(UPSAMPLED_TEST_DIR)
        except FileExistsError:
            print(f"folder already exists: {UPSAMPLED_TEST_DIR}")
        try:
            os.mkdir(UPSAMPLED_TILED_TRAIN_DIR)
        except FileExistsError:
            print(f"folder already exists: {UPSAMPLED_TILED_TRAIN_DIR}")
        try:
            os.mkdir(UPSAMPLED_TILED_TEST_DIR)
        except FileExistsError:
            print(f"folder already exists: {UPSAMPLED_TILED_TEST_DIR}")

        utrd = os.listdir(UPSAMPLED_TRAIN_DIR)
        uted = os.listdir(UPSAMPLED_TEST_DIR)

        for file in os.listdir(LR_TRAIN_DIR):
            if file[:-3]+"png" in utrd:
                continue
            lr_image = open_image(LR_TRAIN_DIR+file)
            upsampled =
linear_interpolate(lr_image,target_shape=hr_image.shape,method=METHOD)
            upsampled = upsampled//256
            upsampled = Image.fromarray(upsampled).convert("L")
            upsampled.save(UPSAMPLED_TRAIN_DIR+file[:-3]+"png")
        for file in os.listdir(LR_TEST_DIR):
            if file[:-3]+"png" in uted:
                continue
            lr_image = open_image(LR_TEST_DIR+file)
            upsampled =
linear_interpolate(lr_image,target_shape=hr_image.shape,method=METHOD)
            upsampled = upsampled//256
            upsampled = Image.fromarray(upsampled).convert("L")
            upsampled.save(UPSAMPLED_TEST_DIR+file[:-3]+"png")

        MASKS_TRAIN = sorted(glob(HR_TRAIN_DIR+"/*"))
        MASKS_TEST  = sorted(glob(HR_TEST_DIR+"/*"))
        CONVS_TRAIN = sorted(glob(UPSAMPLED_TRAIN_DIR+"/*"))
        CONVS_TEST  = sorted(glob(UPSAMPLED_TEST_DIR+"/*"))

        TRAIN_INDICES = random.sample(range(len(MASKS_TRAIN)-TILE_LENGTH),
TRAINING_SAMPLES)
        TEST_SAMPLES = len(MASKS_TEST)//TILE_LENGTH
        TEST_INDICES = np.linspace(0,TILE_LENGTH*(TEST_SAMPLES-
1),TEST_SAMPLES).astype(int)

        try:
```

```
            os.mkdir(UPSAMPLED_TILED_TEST_SEG_DIR)
        except FileExistsError:
            print(f"folder already exists: {UPSAMPLED_TILED_TEST_SEG_DIR}")

        for i, x in enumerate(TRAIN_INDICES):
            x = TRAIN_INDICES[i]
            mask_tile = np.concatenate([np.array(Image.open(mask)) for mask in
MASKS_TRAIN[x:x+TILE_LENGTH]], axis=1)
            conv_tile = np.concatenate([np.array(Image.open(conv)) for conv in
CONVS_TRAIN[x:x+TILE_LENGTH]], axis=1)
            Image.fromarray(mask_tile).save(f"{UPSAMPLED_TILED_TRAIN_DIR}/trai
n_{str(i).zfill(5)}_masks.png")
            Image.fromarray(conv_tile).save(f"{UPSAMPLED_TILED_TRAIN_DIR}/trai
n_{str(i).zfill(5)}.png")
        for i, x in enumerate(TEST_INDICES):
            x = TEST_INDICES[i]
            mask_tile = np.concatenate([np.array(Image.open(mask)) for mask in
MASKS_TEST[x:x+TILE_LENGTH]], axis=1)
            conv_tile = np.concatenate([np.array(Image.open(conv)) for conv in
CONVS_TEST[x:x+TILE_LENGTH]], axis=1)
            Image.fromarray(mask_tile).save(f"{UPSAMPLED_TILED_TEST_DIR}/test_
{str(i).zfill(5)}_masks.png")
            Image.fromarray(conv_tile).save(f"{UPSAMPLED_TILED_TEST_DIR}/test_
{str(i).zfill(5)}.png")
        return UPSAMPLED_TILED_TRAIN_DIR, UPSAMPLED_TILED_TEST_DIR,
UPSAMPLED_TILED_TEST_SEG_DIR
    else:
        methods = ['linear','nearest','slinear','cubic','quintic','pchip']
        if METHOD not in methods:
            print(f"{METHOD} is not a valid method. Valid methods are:
{methods}")
            raise TypeError
        HR_TRAIN_DIR = SEGMENTATION_IMAGES_DIR + "/train/" + HR_FOLDER_NAME +
"/masks/"
        HR_TEST_DIR  = SEGMENTATION_IMAGES_DIR + "/test/"  + HR_FOLDER_NAME +
"/masks/"
        LR_TRAIN_DIR = SEGMENTATION_IMAGES_DIR + "/train/" + LR_FOLDER_NAME +
"/convolutions/"
        LR_TEST_DIR  = SEGMENTATION_IMAGES_DIR + "/test/"  + LR_FOLDER_NAME +
"/convolutions/"
        if len(os.listdir(HR_TRAIN_DIR)) != len(os.listdir(LR_TRAIN_DIR)):
            print(f"HR and LR training folders should be the same length, but
have lengths {len(os.listdir(HR_TRAIN_DIR))},
{len(os.listdir(LR_TRAIN_DIR))}")
            raise ValueError
        if len(os.listdir(HR_TEST_DIR)) != len(os.listdir(LR_TEST_DIR)):
```

```
            print(f"HR and LR training folders should be the same length, but
have lengths {len(os.listdir(HR_TRAIN_DIR))},
{len(os.listdir(LR_TRAIN_DIR))}")
            raise ValueError
        hr_image = open_image(HR_TRAIN_DIR+os.listdir(HR_TRAIN_DIR)[0])
        SR_TRAIN_DIR       = SEGMENTATION_IMAGES_DIR + "/train/" +
LR_FOLDER_NAME + f"_{METHOD}_{SR_PREFIX}/"
        SR_TEST_DIR        = SEGMENTATION_IMAGES_DIR + "/test/"  +
LR_FOLDER_NAME + f"_{METHOD}_{SR_PREFIX}/"
        SR_TILED_TRAIN_DIR = SEGMENTATION_IMAGES_DIR + "/train/" +
LR_FOLDER_NAME + f"_{METHOD}_{SR_PREFIX}_tiled/"
        SR_TILED_TEST_DIR  = SEGMENTATION_IMAGES_DIR + "/test/"  +
LR_FOLDER_NAME + f"_{METHOD}_{SR_PREFIX}_tiled/"
        SR_TILED_TEST_SEG_DIR  = SEGMENTATION_IMAGES_DIR + "/test/"  +
LR_FOLDER_NAME + f"_{METHOD}_{SR_PREFIX}_tiled_segmentations/"

        try:
            os.mkdir(SR_TRAIN_DIR)
        except FileExistsError:
            print(f"folder already exists: {SR_TRAIN_DIR}")
        try:
            os.mkdir(SR_TEST_DIR)
        except FileExistsError:
            print(f"folder already exists: {SR_TEST_DIR}")
        try:
            os.mkdir(SR_TILED_TRAIN_DIR)
        except FileExistsError:
            print(f"folder already exists: {SR_TILED_TRAIN_DIR}")
        try:
            os.mkdir(SR_TILED_TEST_DIR)
        except FileExistsError:
            print(f"folder already exists: {SR_TILED_TEST_DIR}")

        for file in tqdm(os.listdir(LR_TRAIN_DIR)):
            lr_image = open_image(LR_TRAIN_DIR+file)
            upsampled =
linear_interpolate(lr_image,target_shape=hr_image.shape,method=METHOD)
            upsampled =
upsampled.reshape(1,1,hr_image.shape[0],hr_image.shape[1])
            upsampled = (upsampled//256) #/255
            upsampled = torch.tensor(upsampled, dtype=torch.float)
            upsampled = upsampled.to(device)
            # print(f"the device here is {upsampled.device}")
            with torch.no_grad():
                output = SR_MODEL(upsampled).cpu()
                sr_image =
(np.array(output).reshape(hr_image.shape)).astype(int) # *255
            sr_image = Image.fromarray(sr_image).convert("L")
```

```
            sr_image.save(SR_TRAIN_DIR+file[:-3]+"png")
        for file in tqdm(os.listdir(LR_TEST_DIR)):
            lr_image = open_image(LR_TEST_DIR+file)
            upsampled =
linear_interpolate(lr_image,target_shape=hr_image.shape,method=METHOD)
            upsampled =
upsampled.reshape(1,1,hr_image.shape[0],hr_image.shape[1])
            upsampled = (upsampled//256) #/255
            upsampled = torch.tensor(upsampled, dtype=torch.float)
            upsampled = upsampled.to(device)
            # print(f"the device here is {upsampled.device}")
            with torch.no_grad():
                output = SR_MODEL(upsampled).cpu()
                sr_image =
(np.array(output).reshape(hr_image.shape)).astype(int) # *255
            sr_image = Image.fromarray(sr_image).convert("L")
            sr_image.save(SR_TEST_DIR+file[:-3]+"png")

        MASKS_TRAIN = sorted(glob(HR_TRAIN_DIR+"/*"))
        MASKS_TEST  = sorted(glob(HR_TEST_DIR+"/*"))
        CONVS_TRAIN = sorted(glob(SR_TRAIN_DIR+"/*"))
        CONVS_TEST  = sorted(glob(SR_TEST_DIR+"/*"))

        TRAIN_INDICES = random.sample(range(len(MASKS_TRAIN)-TILE_LENGTH),
TRAINING_SAMPLES)
        TEST_SAMPLES = len(MASKS_TEST)//TILE_LENGTH
        TEST_INDICES = np.linspace(0,TILE_LENGTH*(TEST_SAMPLES-
1),TEST_SAMPLES).astype(int)

        try:
            os.mkdir(SR_TILED_TEST_SEG_DIR)
        except FileExistsError:
            print(f"folder already exists: {SR_TILED_TEST_SEG_DIR}")

        for i, x in enumerate(TRAIN_INDICES):
            x = TRAIN_INDICES[i]
            mask_tile = np.concatenate([np.array(Image.open(mask)) for mask in
MASKS_TRAIN[x:x+TILE_LENGTH]], axis=1)
            conv_tile = np.concatenate([np.array(Image.open(conv)) for conv in
CONVS_TRAIN[x:x+TILE_LENGTH]], axis=1)
            Image.fromarray(mask_tile).save(f"{SR_TILED_TRAIN_DIR}/train_{str(
i).zfill(5)}_masks.png")
            Image.fromarray(conv_tile).save(f"{SR_TILED_TRAIN_DIR}/train_{str(
i).zfill(5)}.png")
        for i, x in enumerate(TEST_INDICES):
            x = TEST_INDICES[i]
            mask_tile = np.concatenate([np.array(Image.open(mask)) for mask in
MASKS_TEST[x:x+TILE_LENGTH]], axis=1)
```

```
            conv_tile = np.concatenate([np.array(Image.open(conv)) for conv in
CONVS_TEST[x:x+TILE_LENGTH]], axis=1)
            Image.fromarray(mask_tile).save(f"{SR_TILED_TEST_DIR}/test_{str(i)
.zfill(5)}_masks.png")
            Image.fromarray(conv_tile).save(f"{SR_TILED_TEST_DIR}/test_{str(i)
.zfill(5)}.png")
        return SR_TILED_TRAIN_DIR, SR_TILED_TEST_DIR, SR_TILED_TEST_SEG_DIR

def do_everything_after(SR_TILED_TRAIN_DIR, SR_TILED_TEST_DIR,
SR_TILED_TEST_SEG_DIR, TILE_LENGTH=40, model_number=-1):
    try:
        os.mkdir(SR_TILED_TEST_SEG_DIR)
    except FileExistsError:
        print(f"folder already exists: {SR_TILED_TEST_SEG_DIR}")
    all = sorted(glob(SR_TILED_TEST_DIR + "/*"))
    mask = [m for m in all if "mask" in m]
    conv = [c for c in all if "mask" not in c]
    print(f"length of masks: {len(mask)}, {len(conv)}")
    masks = [np.asarray(Image.open(file)) for file in mask]
    convs = [np.asarray(Image.open(file)) for file in conv]
    nimg = len(convs)
    model_list = natsorted(glob(SR_TILED_TRAIN_DIR+"models/*"))
    model_name = model_list[model_number]
    print(f"Using model: {model_name}")
    use_gpu = True
    model =
models.CellposeModel(gpu=use_gpu,pretrained_model=model_name,omni=True,concate
nation=True)
    chans = [0,0] #this means segment based on first channel, no second
channel

    n = [0] # make a list of integers to select which images you want to
segment
    n = range(nimg) # or just segment them all

    # define parameters
    mask_threshold = -1
    verbose = False # turn on if you want to see more output
    transparency = True # transparency in flow output
    rescale=None # give this a number if you need to upscale or downscale your
images
    omni = True # we can turn off Omnipose mask reconstruction, not advised
    flow_threshold = 0. # default is .4, but only needed if there are spurious
masks to clean up; slows down output
    resample = True #whether or not to run dynamics on rescaled grid or
original grid
```

```
    segmentations, flows, styles = model.eval([convs[i] for i in
n],channels=chans,rescale=rescale,mask_threshold=mask_threshold,transparency=t
ransparency,
                                      flow_threshold=flow_threshold,omni=omni,re
sample=resample,verbose=verbose)
    for idx,i in enumerate(n):
        maski = segmentations[idx]
        im = Image.fromarray(maski)
        im.save(f"{SR_TILED_TEST_SEG_DIR}/omni_{str(idx).zfill(5)}.png")
    outlist = []
    excesslist = []
    nearlist = []
    distlist = []
    for i in n:
        mask = masks[i]
        seg = segmentations[i]
        # plt.imshow(mask,"gray")
        # plt.title(f"mask {i}")
        # plt.show()
        # plt.imshow(seg,"gray")
        # plt.title(f"seg {i}")
        # plt.show()
        width = mask.shape[1]//TILE_LENGTH
        for j in range(TILE_LENGTH):
            maskj = mask[:,width*j:width*(j+1)]
            segj = seg[:,width*j:width*(j+1)]
            out, excess = IoU(maskj,segj)
            outlist.append(out)
            excesslist.append(excess)
            nearest, dist = centroid_distances(maskj,segj)
            nearlist.append(nearest)
            distlist.append(dist)
    cells, dists, duplicates, indices = ignore_duplicates(nearlist, distlist)
    dists = np.append(dists, [np.max(dists)]*duplicates)
    perclist = []
    for out in outlist:
        for a,b in out:
            perclist.append(a/b)
    percarr = np.array(perclist)
    excessout = []
    for exc in excesslist:
        excessout.extend(exc)
    excessarr = np.array(excessout)
    return percarr, cells, dists, duplicates, indices, excessarr

def get_test_ssim_psnr(HR_IMAGES_DIR,SR_IMAGES_DIR):
    ssim_list = []
    psnr_list = []
```

```
    ssim = SSIM(n_channels=1) #.cuda()
    psnr = PSNR()
    for hr_img, sr_img in
zip(os.listdir(HR_IMAGES_DIR),os.listdir(SR_IMAGES_DIR)):
        hr_img = open_image(HR_IMAGES_DIR + hr_img)
        sr_img = open_image(SR_IMAGES_DIR + sr_img)
        hr_img_torch =
torch.tensor(hr_img.reshape(1,1,hr_img.shape[0],hr_img.shape[1]))/np.max(hr_im
g)
        sr_img_torch =
torch.tensor(sr_img.reshape(1,1,sr_img.shape[0],sr_img.shape[1]))/np.max(sr_im
g)
        img_ssim = ssim(sr_img_torch,hr_img_torch)
        img_psnr = psnr(sr_img_torch,hr_img_torch)
        ssim_list.append(img_ssim)
        psnr_list.append(img_psnr)
    return ssim_list, psnr_list
```

- 13/03/2023

new results:

phase contrast omnipose data (is it valid to compare results from the actual LR segmentations to HR segmentations?

is the linear upsample a more valid comparison given the number of pixels can affect IoU indirectly? also since

for the upsampled images they are compared to the 100x actual masks instead of downsampled masks does this make

them more valid results?)

superres trial 1 (linear)

superres trial 2 (cubic) (did linear vs cubic actually make a difference? doubtful but should be tested with linear again just in case)

test on real data

next steps:

do again with realistic images (hopefully collected very soon) for PC (also with Fl)

think - do we actually expect SR to improve performance far beyond what just segmentation can do - both networks would be provided

with similar information via training data - very difficult inputs with clear outputs

SRCNN at very least is very quick to train compared to omnipose (due to simplicity) - maybe main benefit could be from the speedup of

training if it turns out that asymptotic results are similar (ie quickly train SR to 100x level then train omnipose for x epochs,

which might outperform omnipose at (5+)x epochs from the low res image) - should investigate this behaviour over epochs

trained (up to 4000+) (current results are sup 500 epochs mostly, 200 for trial 2)

SRCNN TRAIN COMMAND:

python train.py --epochs 100 --hr-path "C:/Users/robho/OneDrive/Desktop/Uni_Work/Year_4/project2/superres/first_trial/SR_training/pmc_0.0655/" --lr-path "C:/Users/robho/OneDrive/Desktop/Uni_Work/Year_4/project2/superres/first_trial/SR_training/pmc_0.1638/" --hr-validation-path "C:/Users/robho/OneDrive/Desktop/Uni_Work/Year_4/project2/superres/first_trial/main_batch/pmc_0.0655/test/convolutions" --lr-validation-path "C:/Users/robho/OneDrive/Desktop/Uni_Work/Year_4/project2/superres/first_trial/main_batch/pmc_0.1638/test/conv_linear"

- 20/03/2023

Results for srcnn on provisional data is really poor – need to make some more realistic images

Modify symbac to add on halo effect on trench images? And add more noise and stuff

- use ARMA model to generate effect on images based on signal from real image?

- actually just use the background lighting with a bit of smoothing to reduce noise effects LOL

- 27/03/2023

Modified symbac working, dataset generated for 20x 30x 40x 60x 90x – looks pretty good

- 03/04/2023

Exams

- 10/04/2023

Exams

- 17/04/2023

Exams

- 24/04/2023

Exams

- 01/05/2023

Exams

- 08/05/2023

Trying out RCAN – see notebooks in repo for more details