**Data File S4. Codes for U-Net deep-learning and ImageJ analysis**

**Script for U-Net deep learning**

from fastai.vision import \*

from fastai.callbacks import \*

path = untar\_data(URLs.CAMVID)

print(path.ls())

path\_img = path/'images

path\_lbl = path/'labels

fnames = get\_image\_files(path\_img)

lbl\_names = get\_image\_files(path\_lbl)

img\_f = fnames[0]

img = open\_image(img\_f)

img.show(figsize=(5, 5))

get\_y\_fn = lambda x: path\_lbl/f'{x.stem}\_P{x.suffix}'

print(get\_y\_fn(img\_f))

mask = open\_mask(get\_y\_fn(img\_f), div=True)

mask.show(figsize=(5, 5), alpha=1)

src\_size = np.array(mask.shape[1:])

src\_size,img.data

src\_size,mask.data

class BinaryLabelList(SegmentationLabelList):

def open(self, fn): return open\_mask(fn, div=True)

class BinaryItemList(SegmentationItemList):

\_label\_cls = BinaryLabelList

codes = np.array(["Background","Droplet"])

size = src\_size

bs = 2

src = (BinaryItemList.from\_folder(path\_img)

.split\_by\_fname\_file('../valid.txt')

.label\_from\_func(get\_y\_fn, classes=codes))

tfms = get\_transforms(flip\_vert=True)

data = (src.transform(tfms, size=size, tfm\_y=True)

.databunch(bs=bs, num\_workers=0)

.normalize(imagenet\_stats))

data.show\_batch(rows=2, figsize=(12, 9))

def acc\_camvid(input, target):

target = target.squeeze(1)

return (input.argmax(dim=1)==target).float().mean()

metrics = acc\_camvid

wd = 1e-2

learn = unet\_learner(data, models.resnet34, metrics=metrics, wd=wd, bottle=True, callback\_fns=[CSVLogger])

learn.lr\_find()

learn.recorder.plot()

lr = 1e-4

learn.fit\_one\_cycle(20, slice(lr), pct\_start=0.3)

learn.recorder.plot\_losses()

learn.save('Miyawaki-stage-1')

learn.show\_results(rows=2, figsize=(8, 9))

learn.unfreeze()

learn.lr\_find()

learn.recorder.plot()

lrs = slice(1e-4)

learn.fit\_one\_cycle(100, lrs, pct\_start=0.3)

learn.recorder.plot\_losses()

learn.save('Miyawaki-stage-2')

learn.show\_results(rows=2, figsize=(8, 9))

prediction = path/'prediction'

files = os.listdir(prediction)

count = len(files)

for i in range(count):

img = open\_image(f'{prediction}/{files[i]}')

mask\_pred = learn.predict(img);

mask\_pred[0].save(f'{prediction}/{files[i]}\_mask.png')

**Script for ImageJ**

* **Image extraction from CZI files**

dir=getDirectory("Choose a Directory");

print(dir);

splitDir=dir + "\Split\\";

print(splitDir);

File.makeDirectory(splitDir);

list = getFileList(dir);

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

if (endsWith(list[i], ".czi")){

print(i + ": " + dir+list[i]);

open(dir+list[i]);

imgName=getTitle();

baseNameEnd=indexOf(imgName, ".czi");

baseName=substring(imgName, 0, baseNameEnd);

run("Split Channels");

selectWindow("C1-" + imgName);

saveAs("Png", dir + "C1-" + baseName + ".PNG");

close();

selectWindow("C2-" + imgName);

saveAs("Png", dir + "C2-" + baseName + ".PNG");

close();

run("Close All");

}

}

* **Particle analysis**

dir=getDirectory("Choose a Directory");

print(dir);

list=getFileList(dir);

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

if (endsWith(list[i], "mask.png")){

print(i + ": " + dir+list[i]);

open(dir+list[i]);

imgName=getTitle();

baseNameEnd=indexOf(imgName, ".png\_mask.png");

baseName=substring(imgName, 3, baseNameEnd);

open(dir + "C1-" + baseName + ".png");

selectWindow(imgName);

setThreshold(1, 255);

redirectImageTitle = "C1-" + baseName + ".png";

run("Set Measurements...", "area standard deviation mean min & max integrated density centroid limit redirect=" + redirectImageTitle + " decimal=3");

run("Analyze Particles...", "size=0-Infinity circularity=0.00-1.00 show=Outlines display results");

selectWindow("Drawing of " + imgName);

saveAs("Png", dir + "Drawing of C2-" + baseName + ".png");

close();

selectWindow("Results");

saveAs("Results", dir + baseName + ".csv");

close();

run("Close All");

}

}