8-2020

Applying deep learning for cell detection in time-lapse microscopic images

Jay Patel

University of Tennessee at Chattanooga, jxh869@mocs.utc.edu

Applying Deep Learning for Cell Detection in Time-Lapse Microscopic Images

By

Jay Patel

College of Engineering and Computer Science

University of Tennessee at Chattanooga

Undergraduate Thesis

Bachelor of Computer Science

Examination Date: July 9, 2020

Dr. Hong Qin

Associate Professor of Computer
Science and Engineering

Thesis Director

Dr. Yu Liang
Associate Professor of Computer
Science and Engineering
Department Examiner

Abstract

The budding yeast Saccharomyces cerevisiae is an effective model for studying cellular aging. We can measure the lifespan of yeast cells in two ways: replicative and chronological lifespans. Chronological focuses on the time that a cell can survive. The replicative lifespan (RLS) is the number of cell divisions that a single mother cell can go through before ceases to be dividing. RLS is a measurement of individual cells and is more informative on the aging process than in chronological lifespan. Many genes that influence yeast RLS have been shown to be highly conserved and have a similar effect on aging in humans. Hence, studies on cellular aging typically focus on RLS. RLS is traditionally measured by micro-dissection – a tedious and time-consuming process. Recently, a high-throughput yeast aging analysis (HYAA) based on microfluidics measurement of replicative aging has been developed. Each mother cell is captured by a trap on the microfluidic device. This device generates an enormous amount of dataset, but the process to manually track these objects is tedious and time consuming and would take years with how large a single dataset can be. This thesis is to address the challenges on how to efficiently and reliably infer the RLS from thousands of time-lapse microscopic images. We implemented two deep learning methods, Faster R-CNN and MASK R-CNN to detect cell the objects. Our results show that Mask R-CNN is a promising method to automate the HYAA image analysis compared to Faster R-CNN approach.

Table of Contents

Introduction

The huge traction in Computer Vision in recent years has led to great advancements in the field. We focus on the medical field, but it does not limit the advancements to just this field. Computer vision has allowed for faster and more accurate diagnoses. The manual tedious and time-consuming tasks over time can be passed on to computers to handle, which saves not only time but could provide faster diagnoses. A computer can do a far better job at recognition than the human eye can do in a short amount of time.

Microfluidics-based time lapse imaging has the potential to transform biomedical research [12]. One important application of microfluidics-based microscopic imaging is the research on cellular aging. One of the most important tasks in the microscopic image analysis is cell segmentation, as reviewed recently [4]. The primary challenges for microscopic cell segmentation include a poor contrast between cells, their background and irregular morphology [4]. Recently deep learning-based methods for microscopic cell image include a Mask Recurrent CNN, a U-net method that contains a convolutional layer and deconvolutional layer with skip connections, a pyramid-based multi-layered fully convolutional neural networks, a combined method with distance estimation and fully convolutional neural network approach [4].

Mask R-CNN deep learning-based approaches generally outperform traditional cell segmentation methods, such as water-shade algorithms. Mask R-CNN as written by the author is simple, flexible, and is a general framework for object instance segmentation [5]. It extends from Faster R-CNN and adds a branch for object mask which runs in parallel with the bounding box [5]. Justin Clark, an MS graduate student, in Dr. Qin's group compared the performance of several deep learning methods on yeast microfluidic trap images. Mehran Ghafari, a PhD student, in Dr. Qin's group, applied the Recurrent CNN approach to detect cell objects in rectangular boxes. These current and previous studies were things that laid the foundations for this thesis work. This thesis focuses on Faster R-CNN and Mask R-CNN, which are improvements made on Recurrent CNN and Fast R-CNN.

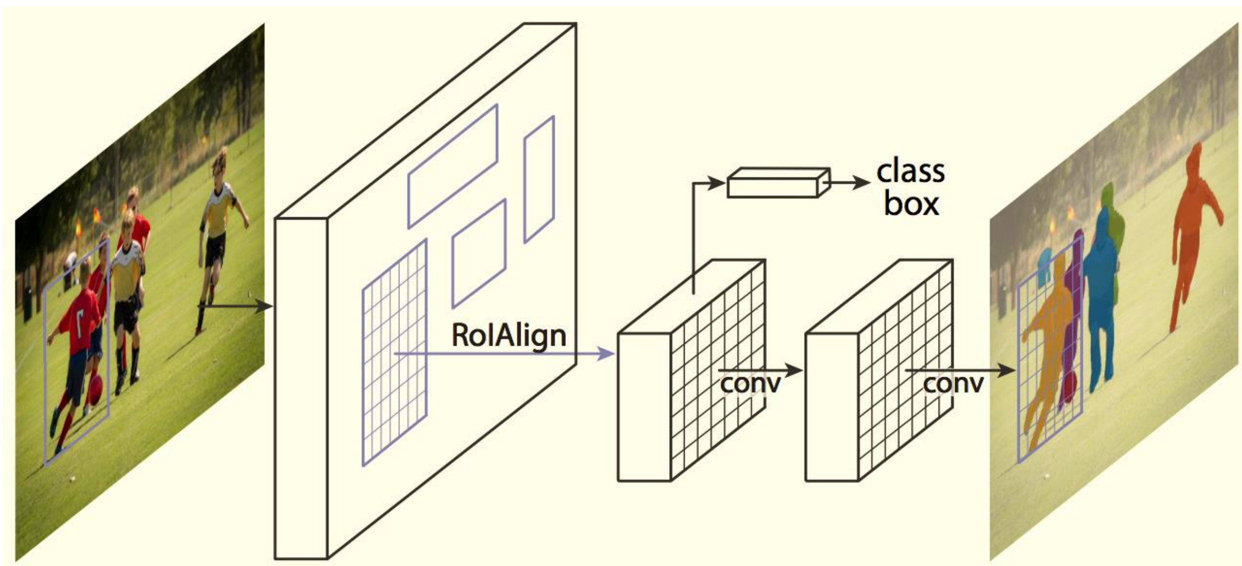

Figure 1 Mask R-CNN framework. [5]

The approach focused on this paper is Mask R-CNN, developed by the Facebook AI Research Team (FAIR) in 2017. Mask R-CNN uses a similar feature extraction model that is used within Faster R-CNN, but there are a few key differences. The first major difference between the two is that ROI-Pooling used in Faster R-CNN is replaced with ROI-Align, which according to the authors leads to a large improvement. Another difference between the two is that there is a network head in Mask R-CNN which generates the image segmentation see Figure 1 for more detail [5].

The HYAA instrument which is used automate lifespan tracking process. This instrument has 4 modules, and each module contains 4 channels per module, so in total there are 16 channels [14]. Each channel contains approximately 520 single traps structures and the device itself has a total of 8,320 single trap structure. Typically, HYAA images are taken every 10 minutes to record the division events of these mother cells. The picture that the instrument takes has about ~100 traps per image as shown in Figure 2. We end up taking the image with 100 traps and breaking them into individual trap structures. By breaking these large images into smaller ones, we generate an enormous amount of data. Taking that into consideration this process would be tedious and time consuming to do all manually. This thesis is to address the challenges on how to efficiently track the cell objects efficiently and reliably on a large dataset.

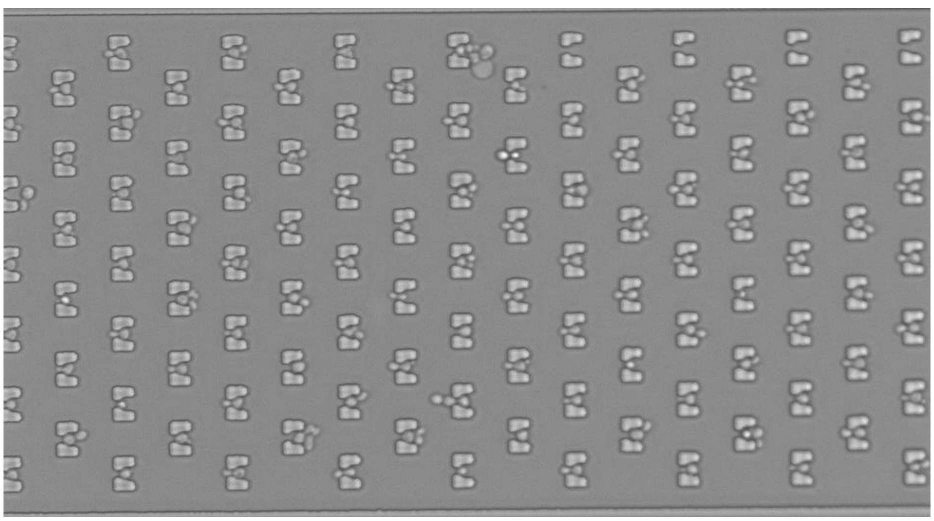

Figure 2. One of the many pictures taken showing mother cell and traps at one point in time lapse.

Methodology

There were two methods used in this thesis work: the first uses a faster R-CNN with Inception V2 [9], and the second uses the Mask R-CNN [5]. The dataset we used for this comprised 100 images, 80 for training, and 20 for validation. We randomly selected these images from a batch of images that contained anywhere between 1-3 cells per image. The training set had around 40 images that were much higher in contrast, which may have affected the performance of the training. The dataset contained a time-lapse image of the mother cell and daughter cells which are recorded every 10 minutes as mention previously. Both methods we approached required using a python script to resize the original images from 60x60 pixels to 512x512 pixels using cv2 and cubic interpolation on the resize [10].

The first attempt used to tackle this challenge was by using faster R-CNN with Inception v-2. This method generated an image with bounding boxes that outline the object and the confidence score. This approach required manual boxing of the cell object using LabelImg which generated an XML file which corresponding x min, y min, x max, y max corresponding to edges of the box where the object is located shown in Figure 3 [1]. For each of the images in the test and train directory, there’s a corresponding XML. After generating XML, we ended up generating TFRecords which was used to import data to the TensorFlow training model. The results for Faster R-CNN were very good however they generate a box and not a mask, so we shifted focus to Mask R-CNN which looked to be more promising.

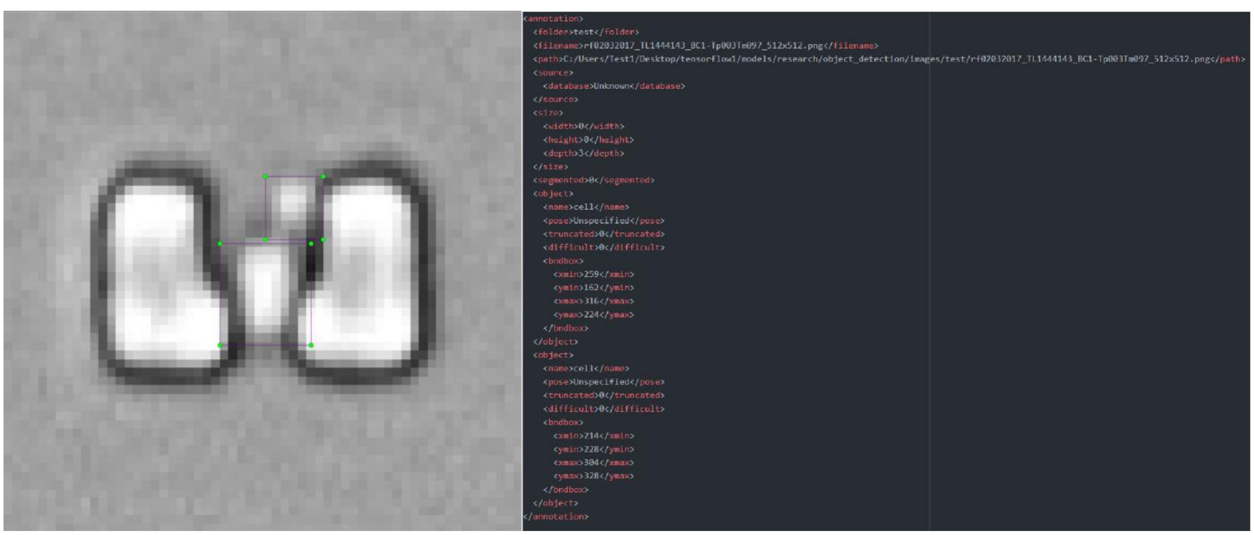

Figure 3. Faster R-CNN dataset format. On the left is an example trap with a pair of mother-daughther cells. On the right is the corresponding XML generated by LabelImg [1].

The second approach we used MatterPort's Mask R-CNN implementation to build a model that detects the generates the object mask. To prepare the dataset, we used VGG image annotator to outline the cell objects which gives us the output as JSON with an x and y coordinates as shown in Figure 4 [6].

Figure 4. Mask R-CNN dataset format. On the left is an example trap with a pair of mother-daughter cells. On the right is the JSON file generated based on manually outlining the object

After generating the dataset, we used Matterport's implementation for Mask R-CNN for our object detection. This was a very well designed and had great documentation for anyone to use on their project. After hours of trying to get a specified version to run with their implementation of Mask R-CNN, we were able to get files running. The final version that ended up working for us was python 3.7 with TensorFlow version 1.5.0 with Keras 2.0.8. For the backbone, we ended up using resnet50 to make sure we could run this on the virtual machine without any performance problems. Resnet101 or feature pyramid network (FPN) would have also been great choices since both are said to be faster and more accurate than resnet50 but require more resources [5]. We started by using weights from the MSCOCO dataset [8]. We trained the network for a total of 50 epochs with 50 steps per epoch. The learning rate was .001

and momentum of .9 with weight decay of .001. For the batch size, we kept it at 2 since we had a very small dataset on a virtual machine.

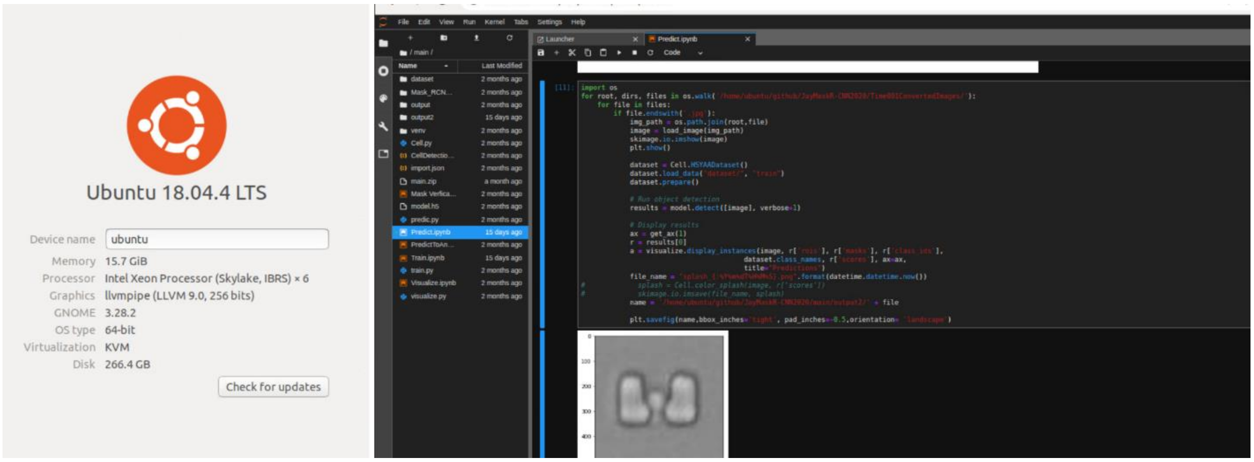

Figure 5. Ubuntu specification and Jupyter Lab running the code on a Linux virtual machine.

For this project, we used an Ubuntu 18.04.4 Virtual Machine (VM). The VM had 16gb of Ram and Intel Xeon Processor with 6 cores as shown in Figure 5. The virtual graphics for this is unknown but it is a Nvidia Graphics card. We used Jupyter Lab for running our code. Jupyter Notebook was having problems running the code, so we switched over to Jupyter Lab. For our purposes, the training and detection done were smooth, and the specifications were more than enough for this project. The training for Mask R-CNN was done with in few hours.

Results

Two approaches were taken, both of which yielded great results. The Faster R-CNN did a splendid job tracking the cells, but only provided the boxes, which were not very useful for our purpose, however, generated very accurate results. The second attempt was the Mask R-CNN, which was more difficult to set up and generate a dataset, but it yielded the best result by having an accurate outline of the objects. The first attempt at Mask R-CNN was a failure for some reason generated a model that wouldn't predict anything. The second attempt with Mask R-CNN with help from a collaborator was very accurate with a small dataset that had 80 images for training and 20 for the validation. After leaving the model to train on an OpenStack Virtual Machine for just a few hours, the results were accurate but still presented some problems. The model did a good job of tracking cell objects that were close together, but had a hard time finding objects that were very irregular in shapes. However, with a bigger dataset and more training, we believe this could be resolved. For smaller datasets, the predictions made by the model yielded very accurate results.

The result was a Mask R-CNN model that was accurately predicting the cell objects within the image. It did not distinguish by color which was mother cell, and which was the offspring's but that is something that we will be focused on in the future. Figure 6 shows the best results from running the model of a set of images. More results from the model can be found in the Appendix.

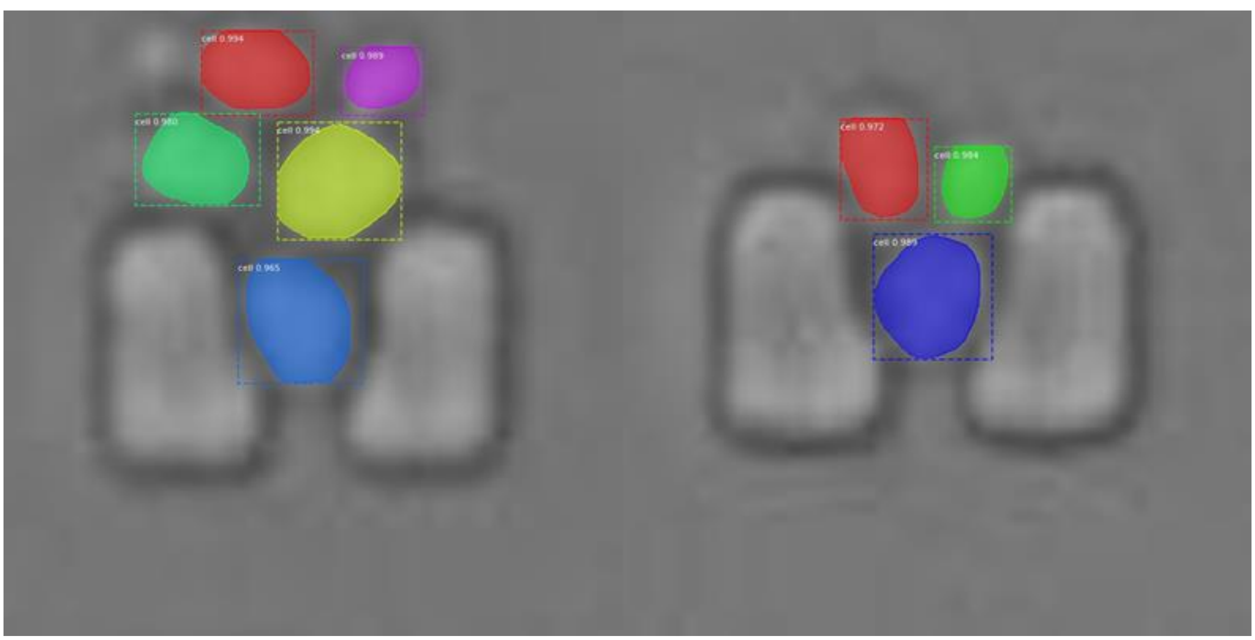

Figure 6. Results from the ResNet-50 Mask R-CNN model.

Conclusions

Computer vision has a lot of uses varying from facial recognition, self-driving cars, shopping, and many other things. With the huge traction in recent years, there are a lot of advancements being made in a brief span of time. While working on this project it's been interesting looking at how companies like Tesla have been using AI for their self-driving or how Apple has been using AI for face detection or how machine learning can find abnormalities to better treat patients. The possibilities are endless for this field and there are a lot of great things come from AI in the coming years. This research uses computer vision to automate the process of manually segmenting the cell objects in a large set of data which can save a lot of time. Mask R-CNN can generate instance segmentation and is more promising than Faster RCNN to improve the computational analysis of time-lapse images for microfluidics images for yeast lifespan inference.

Discussion

This work lays the groundwork to switch from manually segmenting to fully automated segmentation for this specific application. This will allow for hours or days' worth of work to be done within minutes. With more training and a bigger dataset which we plan to auto-generate based on previous results will lead to a much larger dataset in a short period. The results above are based on just the initial tests and will be improved in the future.

Lesson learned

Things learned through this is that it's difficult to set up environments to get programs to run. The TensorFlow version and Keras version must match with what MatterPort used. We also ran into issues with generating a model that detected anything. Not sure what exactly caused the issue but with the help from a collaborator we could generate a new model which detected objects. Another important thing learned is to keep worked constantly updated on GitHub [13]. Some things work for a few minutes and making minor changes that end up breaking the program with no way of backtracking. There are also many ways to approach a project like this because there are a couple of different frameworks out there to extend from, but some outperform others. There have been major improvements in this field, so we researched different options and compare results to select the best framework.

Future Work

Dr.Qin's group has multiple people working on a different aspect of this project, and this is just one of those pieces. This model is excellent at detecting the cell object, but still has problems distinguishing the hard edges around the object and some irregular objects shapes that are not being detected at all. The plan to improve this is by generating a larger dataset by using previously generated results. This will allow for the dataset getting larger without having to manually do all the outlines.

The predictions generated now are colored somewhat randomly, however, making sure the mother cell is the same color is something that will be worked on to improving object detections. To pinpoint the mother cell, we often look for the biggest one in the groups because the mother cell is often the largest one. After we have a successful model that generates a mask accurately, we will need to track the object to generate a how many offspring the mother has throughout the time lapse.

There are already studies out there that focus on this. The study that we investigated was Usiigaci's software, which takes input as a mask and original image in a time series and tracks the positions [11]. Mask R-CNN only generated a Mask so we will try to use what Usiigaci's has worked on seeing if results generated by their software yields accurate results for our purpose. We've not gotten an opportunity to try our output in the software yet, but we are interested in trying to see how they are presented.

Acknowledgements

Research on cellular aging in Dr. Hong Qin's research group is funded by NSF Career award 1453078 transferred to 1720215 and NSF BD Spoke award 1761839.

I would like to acknowledge Mehran Ghafari, Haobo Guo, Cristian Rudas, and Trevor Peyton for helping on different parts through this project.

References

[1] LabelImg https://github.com/tzutalin/labelImg. Accessed: 2019-10-15

[2] Longo, V. D., G. S. Shadel, M. Kaeberlein, and B. Kennedy. 2012. “Replicative and chronological aging in Saccharomyces cerevisiae”, *Cell Metab*, 16: 18-31.

[3] Xing, F., and L. Yang. 2016. “Robust Nucleus/Cell Detection and Segmentation in Digital Pathology and Microscopy Images: A Comprehensive Review”, *IEEE Rev Biomed Eng*.

[4] Zhao, T., and Z Yin. 2018. "Pyramid-Based Fully Convolutional Networks for Cell Segmentation." In Medical Image Computing and Computer Assisted Intervention – MICCAI 2018.

[5] He, K., Gkioxari, G., Dollar, P., & Girshick, R. 2020. “Mask R-CNN”, IEEE transactions on pattern analysis and machine intelligence, 42(2), 386–397.

[6] Dutta, Abhishek & Zisserman, Andrew. 2019. “The VGG Image Annotator (VIA)”. Proceedings of the 27th ACM International Conference on Multimedia.

[7] Johnson J. 2018. ‘Adapting Mask R-CNN for automatic Nucleus Segmentation’, Advances in Intelligent Systems and Computing.

[8] Lin T., et al. 2014. “Microsoft COCO: Common Objects in Context”. Computer Vision – ECCV 2014 Lecture Notes in Computer Science, 740-755.

[9] S. Ren, K. He, R. Girshick and J. Sun. 2017. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, 1 June 2017.

[10] Mahamkali, Naveenkumar & Vadivel, A. 2015. “OpenCV for Computer Vision Applications”.

[11] Tsai, H., Gajda, J., Sloan, T. F., Rares, A., & Shen, A. Q. (2019). Usiigaci: Instance-aware cell tracking in stain-free phase contrast microscopy enabled by machine learning. *SoftwareX*, *9*, 230-237

[12] Ghafari, M., et al. 2019. "Complementary Performances of Convolutional and Capsule Neural Networks on Classifying Microfluidic Images of Dividing Yeast Cells."

[13] 2020 Mask R-CNN: https://github.com/jaypatel98/2020-Mask-R-CNN-Project

[14] Jo, Myeong Chan, et al. “High-Throughput Analysis of Yeast Replicative Aging Using a Microfluidic System.” Proceedings of the National Academy of Sciences, vol. 112, no. 30, Proceedings of the National Academy of Sciences, July 2015, pp. 9364–9369.

Appendix

Cell.py

```
Impor
os
import sys
import itertools
import math
import logging
import json
import re
import random
from collections import OrderedDict
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.lines as lines
from matplotlib.patches import Polygon
import skimage.draw

ROOT_DIR = 'Mask_RCNN-master 3'
DATASET_DIR = os.path.abspath('dataset/')
assert os.path.exists(ROOT_DIR), 'ROOT_DIR does not exist. Did you forget to read
the instructions above?'

sys.path.append(ROOT_DIR)
from mrcnn.config import Config
import mrcnn.utils as utils
from mrcnn import visualize
from mrcnn.visualize import display_images
import mrcnn.model as modellib
from mrcnn.model import log

class CellConfig(Config):
    """
    Configuration for training on the cell dataset.
    """
    # Give the configuration a recognizable name
    NAME = "cell"

    # Train on 1 GPU and 1 image per GPU. Batch size is 1 (GPUs * images/GPU).
```

```
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1

    # Number of classes (including background)
    NUM_CLASSES = 1 + 1  # background + 2 (cell)

    # All of our training images are 512x512
    IMAGE_MIN_DIM = 512
    IMAGE_MAX_DIM = 512

    # You can experiment with this number to see if it improves training
    STEPS_PER_EPOCH = 50

    DETECTION_MIN_CONFIDENCE = 0.9

    # This is how often validation is run. If you are using too much hard drive
space
    # on saved models (in the MODEL_DIR), try making this value larger.
    VALIDATION_STEPS = 5

    # Matterport originally used resnet101, but I downsized to fit it on my
graphics card
    BACKBONE = 'resnet50'

    # To be honest, I haven't taken the time to figure out what these do
    RPN_ANCHOR_SCALES = (8, 16, 32, 64, 128)
    TRAIN_ROIS_PER_IMAGE = 32
    MAX_GT_INSTANCES = 50
    POST_NMS_ROIS_INFERENCE = 500
    POST_NMS_ROIS_TRAINING = 1000
config = CellConfig()

class HSYAADataset(utils.Dataset):
    def load_data(self, dataset_dir, subset):
        """Load a subset of the gns dataset.
        dataset_dir: Root directory of the dataset.
        subset: Subset to load: train or val
        """
        # Add classes. We have two classes to add.
        self.add_class("objects", 1, "cell")

        self.class_name_to_ids = {'cell':1}
        # Train or validation dataset?
```

```
        assert subset in ["train", "val"]
        dataset_dir = os.path.join(dataset_dir, subset)

        # Load annotations
        # VGG Image Annotator (up to version 1.6) saves each image in the form:
        # { 'filename': '28503151_5b5b7ec140_b.jpg',
        #   'regions': {
        #       '0': {
        #           'region_attributes': {},
        #           'shape_attributes': {
        #               'all_points_x': [...],
        #               'all_points_y': [...],
        #               'name': 'polygon'}},
        #       ... more regions ...
        #   },
        #   'size': 100202
        # }
        # We mostly care about the x and y coordinates of each region
        # Note: In VIA 2.0, regions was changed from a dict to a list.
        annotations = json.load(open(os.path.join(dataset_dir,
"via_region_data.json")))
        annotations = list(annotations.values())  # don't need the dict keys

        #if '_via_img_metadata' in annotations:
        #     annotations = list(annotations['_via_img_metadata'].values())  # don't
need the dict keys

        # The VIA tool saves images in the JSON even if they don't have any
        # annotations. Skip unannotated images.
        annotations = [a for a in annotations if a['regions']]

        # Add images
        for a in annotations:
            # Get the x, y coordinaets of points of the polygons that make up
            # the outline of each object instance. These are stores in the
            # shape_attributes (see json format above)
            # The if condition is needed to support VIA versions 1.x and 2.x.
            if type(a['regions']) is dict:
                polygons = [r['shape_attributes'] for r in a['regions'].values()]
                class_names = [list(r['region_attributes']['name']) for r in
a['regions'].values()]
            else:
                polygons = [r['shape_attributes'] for r in a['regions']]
```

```
                class_names = [r['region_attributes']['name'] for r in
a['regions']]

            # load_mask() needs the image size to convert polygons to masks.
            # Unfortunately, VIA doesn't include it in JSON, so we must read
            # the image. This is only managable since the dataset is tiny.
            image_path = os.path.join(dataset_dir, a['filename'])
            image = skimage.io.imread(image_path)
            height, width = image.shape[:2]

            self.add_image(
                "objects",
                image_id=a['filename'],  # use file name as a unique image id
                path=image_path,
                width=width, height=height,
                polygons = polygons,
                class_names = class_names
            )

    def load_mask(self, image_id):
        """Generate instance masks for an image.
       Returns:
        masks: A bool array of shape [height, width, instance count] with
            one mask per instance.
        class_ids: a 1D array of class IDs of the instance masks.
        """
        # If not a gns dataset image, delegate to parent class.
        image_info = self.image_info[image_id]
        if image_info["source"] != "objects":
            return super(self.__class__, self).load_mask(image_id)

        # Convert polygons to a bitmap mask of shape
        # [height, width, instance_count]
        info = self.image_info[image_id]
        mask = np.zeros([info["height"], info["width"], len(info["polygons"])],
                        dtype=np.uint8)
        class_ids = np.ones([mask.shape[-1]], dtype=int)

        for i, p in enumerate(info["polygons"]):
            # Get indexes of pixels inside the polygon and set them to 1
            rr, cc = skimage.draw.polygon(p['all_points_y'], p['all_points_x'])
            mask[rr, cc, i] = 1
```

```
        #  for i,cname in enumerate(info["class_names"]):
          #    class_ids[i] = self.class_name_to_ids[cname]

          # Return mask, and array of class IDs of each instance. Since we have
          # one class ID only, we return an array of 1s
          # Map class names to class IDs.
          return mask.astype(np.bool), class_ids

      def image_reference(self, image_id):
          """Return the path of the image."""
          info = self.image_info[image_id]
          if info["source"] == "objects":
              return info["path"]
          else:
              super(self.__class__, self).image_reference(image_id)

  def train(model, epochs, dataset_folder):
      """Train the model."""
      # Training dataset.
      dataset_train = HSYAADataset()
      dataset_train.load_data(dataset_folder, "train")
      dataset_train.prepare()

      # Validation dataset
      dataset_val = HSYAADataset()
      dataset_val.load_data(dataset_folder, "val")
      dataset_val.prepare()

      # *** This training schedule is an example. Update to your needs ***
      # Since we're using a very small dataset, and starting from
      # COCO trained weights, we don't need to train too long. Also,
      # no need to train all layers, just the heads should do it.
      print("Training network heads")
      model.train(dataset_train, dataset_val,
                  learning_rate=config.LEARNING_RATE,
                  epochs=epochs,
                  layers='heads')

  def color_splash(image, mask):
      """Apply color splash effect.
      image: RGB image [height, width, 3]
      mask: instance segmentation mask [height, width, instance count]
      Returns result image.
```

```
    """
    # Make a grayscale copy of the image. The grayscale copy still
    # has 3 RGB channels, though.
    gray = skimage.color.gray2rgb(skimage.color.rgb2gray(image)) * 255
    # Copy color pixels from the original color image where mask is set
    if mask.shape[-1] > 0:
        # We're treating all instances as one, so collapse the mask into one layer
        mask = (np.sum(mask, -1, keepdims=True) >= 1)
        splash = np.where(mask, image, gray).astype(np.uint8)
    else:
        splash = gray.astype(np.uint8)
    return splash


def detect_and_color_splash(model, image_path=None, video_path=None):
    assert image_path or video_path

    # Image or video?
    if image_path:
        # Run model detection and generate the color splash effect
        print("Running on {}".format(args.image))
        # Read image
        image = skimage.io.imread(args.image)
        # Detect objects
        r = model.detect([image], verbose=1)[0]
        # Color splash
        splash = color_splash(image, r['masks'])
        # Save output
        file_name = "splash_{:%Y%m%dT%H%M%S}.png".format(datetime.datetime.now())
        skimage.io.imsave(file_name, splash)
    elif video_path:
        import cv2
        # Video capture
        vcapture = cv2.VideoCapture(video_path)
        width = int(vcapture.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(vcapture.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = vcapture.get(cv2.CAP_PROP_FPS)

        # Define codec and create video writer
        file_name = "splash_{:%Y%m%dT%H%M%S}.avi".format(datetime.datetime.now())
        vwriter = cv2.VideoWriter(file_name,
                                  cv2.VideoWriter_fourcc(*'MJPG'),
                                  fps, (width, height))
```

```
    count = 0
    success = True
    while success:
        print("frame: ", count)
        # Read next image
        success, image = vcapture.read()
        if success:
            # OpenCV returns images as BGR, convert to RGB
            image = image[..., ::-1]
            # Detect objects
            r = model.detect([image], verbose=0)[0]
            # Color splash
            splash = color_splash(image, r['masks'])
            # RGB -> BGR to save image to video
            splash = splash[..., ::-1]
            # Add image to video writer
            vwriter.write(splash)
            count += 1
    vwriter.release()
print("Saved to ", file_name)
```

Train.ipynb

In [1]:

```
import os
import sys
import itertools
import math
import logging
import json
import re
import random
from collections import OrderedDict
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.lines as lines
from matplotlib.patches import Polygon

# Root directory of the project
ROOT_DIR = 'Mask_RCNN-master 3'

# Import Mask RCNN
sys.path.append(ROOT_DIR)  # To find local version of the library
from mrcnn import utils
from mrcnn import visualize
```

```
from mrcnn.visualize import display_images
import mrcnn.model as modellib
from mrcnn.model import log

import Cell
```

```
Using TensorFlow backend.
```

In [2]:

```
model_dir = "../logs/"
model_file = "coco.h5"
coco_path = os.path.abspath(model_dir + model_file)
```

In [3]:

```
model_dir = "../logs/"
model_file = "coco.h5"
coco_path = os.path.abspath(model_dir + model_file)
```

In [4]:

```
model = modellib.MaskRCNN(mode="training", config=Cell.config, model_dir=mode
l_dir)
```

```
WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/ke
ras/backend/tensorflow_backend.py:517: The name tf.placeholder is deprecated.
Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/ke
ras/backend/tensorflow_backend.py:74: The name tf.get_default_graph is deprec
ated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/ke
ras/backend/tensorflow_backend.py:4138: The name tf.random_uniform is depreca
ted. Please use tf.random.uniform instead.

WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/ke
ras/backend/tensorflow_backend.py:1919: The name tf.nn.fused_batch_norm is de
precated. Please use tf.compat.v1.nn.fused_batch_norm instead.

WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/ke
ras/backend/tensorflow_backend.py:3976: The name tf.nn.max_pool is deprecated
. Please use tf.nn.max_pool2d instead.

WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/ke
ras/backend/tensorflow_backend.py:2018: The name tf.image.resize_nearest_neig
hbor is deprecated. Please use tf.compat.v1.image.resize_nearest_neighbor ins
tead.
```

```
WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/te
nsorflow/python/ops/array_ops.py:1354: add_dispatch_support.<locals>.wrapper
(from tensorflow.python.ops.array_ops) is deprecated and will be removed in a
future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From Mask_RCNN-master 3/mrcnn/model.py:553: The name tf.ra
ndom_shuffle is deprecated. Please use tf.random.shuffle instead.

WARNING:tensorflow:From Mask_RCNN-master 3/mrcnn/utils.py:202: The name tf.lo
g is deprecated. Please use tf.math.log instead.

WARNING:tensorflow:From Mask_RCNN-master 3/mrcnn/model.py:600: calling crop_a
nd_resize_v1 (from tensorflow.python.ops.image_ops_impl) with box_ind is depr
ecated and will be removed in a future version.
Instructions for updating:
box_ind is deprecated, use box_indices instead
```

In [5]:

```python
if not os.path.exists(coco_path):
    utils.download_trained_weights(coco_path)
```

In [6]:

```python
model.load_weights(coco_path, by_name=True, exclude=[
            "mrcnn_class_logits", "mrcnn_bbox_fc",
            "mrcnn_bbox", "mrcnn_mask"])
```

In [7]:

```python
Cell.train(model, 50, "/home/ubuntu/github/2020MaskRCNN/main/dataset")
```

```
Training network heads

Starting at epoch 0. LR=0.001

Checkpoint Path: ../logs/cell20200529T1324/mask_rcnn_cell_{epoch:04d}.h5
Selecting layers to train
fpn_c5p5               (Conv2D)
fpn_c4p4               (Conv2D)
fpn_c3p3               (Conv2D)
fpn_c2p2               (Conv2D)
fpn_p5                 (Conv2D)
fpn_p2                 (Conv2D)
fpn_p3                 (Conv2D)
fpn_p4                 (Conv2D)
In model:  rpn_model
    rpn_conv_shared        (Conv2D)
    rpn_class_raw          (Conv2D)
```

```
    rpn_bbox_pred          (Conv2D)
mrcnn_mask_conv1       (TimeDistributed)
mrcnn_mask_bn1         (TimeDistributed)
mrcnn_mask_conv2       (TimeDistributed)
mrcnn_mask_bn2         (TimeDistributed)
mrcnn_class_conv1      (TimeDistributed)
mrcnn_class_bn1        (TimeDistributed)
mrcnn_mask_conv3       (TimeDistributed)
mrcnn_mask_bn3         (TimeDistributed)
mrcnn_class_conv2      (TimeDistributed)
mrcnn_class_bn2        (TimeDistributed)
mrcnn_mask_conv4       (TimeDistributed)
mrcnn_mask_bn4         (TimeDistributed)
mrcnn_bbox_fc          (TimeDistributed)
mrcnn_mask_deconv      (TimeDistributed)
mrcnn_class_logits     (TimeDistributed)
mrcnn_mask             (TimeDistributed)
WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/ke
ras/optimizers.py:790: The name tf.train.Optimizer is deprecated. Please use
tf.compat.v1.train.Optimizer instead.

/home/ubuntu/anaconda3/lib/python3.7/site-packages/tensorflow/python/ops/grad
ients_util.py:93: UserWarning: Converting sparse IndexedSlices to a dense Ten
sor of unknown shape. This may consume a large amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
/home/ubuntu/anaconda3/lib/python3.7/site-packages/tensorflow/python/ops/grad
ients_util.py:93: UserWarning: Converting sparse IndexedSlices to a dense Ten
sor of unknown shape. This may consume a large amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
/home/ubuntu/anaconda3/lib/python3.7/site-packages/tensorflow/python/ops/grad
ients_util.py:93: UserWarning: Converting sparse IndexedSlices to a dense Ten
sor of unknown shape. This may consume a large amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
/home/ubuntu/anaconda3/lib/python3.7/site-packages/keras/engine/training_gene
rator.py:47: UserWarning: Using a generator with `use_multiprocessing=True` a
nd multiple workers may duplicate your data. Please consider using the`keras.
utils.Sequence class.
  UserWarning('Using a generator with `use_multiprocessing=True`'
WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/ke
ras/callbacks.py:850: The name tf.summary.merge_all is deprecated. Please use
tf.compat.v1.summary.merge_all instead.
```

```
WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/ke
ras/callbacks.py:853: The name tf.summary.FileWriter is deprecated. Please us
e tf.compat.v1.summary.FileWriter instead.

Epoch 1/50
50/50 [==============================] - 51s 1s/step - loss: 1.8281 - rpn_cla
ss_loss: 0.0210 - rpn_bbox_loss: 0.7169 - mrcnn_class_loss: 0.0826 - mrcnn_bb
ox_loss: 0.4626 - mrcnn_mask_loss: 0.5450 - val_loss: 1.6183 - val_rpn_class_
loss: 0.0359 - val_rpn_bbox_loss: 0.6270 - val_mrcnn_class_loss: 0.0575 - val
_mrcnn_bbox_loss: 0.4147 - val_mrcnn_mask_loss: 0.4833
Epoch 2/50
50/50 [==============================] - 32s 645ms/step - loss: 1.2629 - rpn_
class_loss: 0.0202 - rpn_bbox_loss: 0.5408 - mrcnn_class_loss: 0.0668 - mrcnn
_bbox_loss: 0.2629 - mrcnn_mask_loss: 0.3722 - val_loss: 1.9032 - val_rpn_cla
ss_loss: 0.0131 - val_rpn_bbox_loss: 1.0978 - val_mrcnn_class_loss: 0.0318 -
val_mrcnn_bbox_loss: 0.4598 - val_mrcnn_mask_loss: 0.3007
Epoch 3/50
50/50 [==============================] - 29s 587ms/step - loss: 1.1343 - rpn_
class_loss: 0.0174 - rpn_bbox_loss: 0.4647 - mrcnn_class_loss: 0.0684 - mrcnn
_bbox_loss: 0.2176 - mrcnn_mask_loss: 0.3663 - val_loss: 1.2594 - val_rpn_cla
ss_loss: 0.0177 - val_rpn_bbox_loss: 0.5479 - val_mrcnn_class_loss: 0.0634 -
val_mrcnn_bbox_loss: 0.2834 - val_mrcnn_mask_loss: 0.3469
Epoch 4/50
50/50 [==============================] - 30s 607ms/step - loss: 1.2095 - rpn_
class_loss: 0.0225 - rpn_bbox_loss: 0.5814 - mrcnn_class_loss: 0.0426 - mrcnn
_bbox_loss: 0.2232 - mrcnn_mask_loss: 0.3398 - val_loss: 1.5153 - val_rpn_cla
ss_loss: 0.0152 - val_rpn_bbox_loss: 0.7047 - val_mrcnn_class_loss: 0.1124 -
val_mrcnn_bbox_loss: 0.3384 - val_mrcnn_mask_loss: 0.3446
Epoch 5/50
50/50 [==============================] - 31s 627ms/step - loss: 1.2041 - rpn_
class_loss: 0.0201 - rpn_bbox_loss: 0.5172 - mrcnn_class_loss: 0.0397 - mrcnn
_bbox_loss: 0.1880 - mrcnn_mask_loss: 0.4391 - val_loss: 1.5675 - val_rpn_cla
ss_loss: 0.0194 - val_rpn_bbox_loss: 0.8021 - val_mrcnn_class_loss: 0.1081 -
val_mrcnn_bbox_loss: 0.2705 - val_mrcnn_mask_loss: 0.3674
Epoch 6/50
50/50 [==============================] - 33s 656ms/step - loss: 1.1758 - rpn_
class_loss: 0.0159 - rpn_bbox_loss: 0.5649 - mrcnn_class_loss: 0.0388 - mrcnn
_bbox_loss: 0.2370 - mrcnn_mask_loss: 0.3191 - val_loss: 1.3723 - val_rpn_cla
ss_loss: 0.0263 - val_rpn_bbox_loss: 0.6892 - val_mrcnn_class_loss: 0.0730 -
val_mrcnn_bbox_loss: 0.2521 - val_mrcnn_mask_loss: 0.3317
Epoch 7/50
50/50 [==============================] - 32s 633ms/step - loss: 1.0836 - rpn_
class_loss: 0.0166 - rpn_bbox_loss: 0.5941 - mrcnn_class_loss: 0.0256 - mrcnn
_bbox_loss: 0.1817 - mrcnn_mask_loss: 0.2656 - val_loss: 1.4448 - val_rpn_cla
```

```
ss_loss: 0.0094 - val_rpn_bbox_loss: 0.7166 - val_mrcnn_class_loss: 0.0020 -
val_mrcnn_bbox_loss: 0.4638 - val_mrcnn_mask_loss: 0.2531
Epoch 8/50
50/50 [==============================] - 32s 637ms/step - loss: 0.9181 - rpn_
class_loss: 0.0120 - rpn_bbox_loss: 0.4566 - mrcnn_class_loss: 0.0285 - mrcnn
_bbox_loss: 0.2035 - mrcnn_mask_loss: 0.2175 - val_loss: 1.4244 - val_rpn_cla
ss_loss: 0.0173 - val_rpn_bbox_loss: 0.8481 - val_mrcnn_class_loss: 0.0192 -
val_mrcnn_bbox_loss: 0.3908 - val_mrcnn_mask_loss: 0.1491
Epoch 9/50
50/50 [==============================] - 30s 608ms/step - loss: 0.8509 - rpn_
class_loss: 0.0146 - rpn_bbox_loss: 0.4294 - mrcnn_class_loss: 0.0424 - mrcnn
_bbox_loss: 0.1723 - mrcnn_mask_loss: 0.1922 - val_loss: 1.1780 - val_rpn_cla
ss_loss: 0.0161 - val_rpn_bbox_loss: 0.5789 - val_mrcnn_class_loss: 0.0538 -
val_mrcnn_bbox_loss: 0.3316 - val_mrcnn_mask_loss: 0.1975
Epoch 10/50
50/50 [==============================] - 32s 643ms/step - loss: 1.0236 - rpn_
class_loss: 0.0087 - rpn_bbox_loss: 0.5480 - mrcnn_class_loss: 0.0728 - mrcnn
_bbox_loss: 0.1728 - mrcnn_mask_loss: 0.2213 - val_loss: 0.9564 - val_rpn_cla
ss_loss: 0.0112 - val_rpn_bbox_loss: 0.4097 - val_mrcnn_class_loss: 0.1045 -
val_mrcnn_bbox_loss: 0.2287 - val_mrcnn_mask_loss: 0.2022
Epoch 11/50
50/50 [==============================] - 33s 662ms/step - loss: 0.8041 - rpn_
class_loss: 0.0104 - rpn_bbox_loss: 0.3322 - mrcnn_class_loss: 0.0643 - mrcnn
_bbox_loss: 0.1748 - mrcnn_mask_loss: 0.2223 - val_loss: 3.5914 - val_rpn_cla
ss_loss: 0.0261 - val_rpn_bbox_loss: 2.5685 - val_mrcnn_class_loss: 0.0282 -
val_mrcnn_bbox_loss: 0.5553 - val_mrcnn_mask_loss: 0.4134
Epoch 12/50
50/50 [==============================] - 32s 649ms/step - loss: 1.0657 - rpn_
class_loss: 0.0162 - rpn_bbox_loss: 0.6846 - mrcnn_class_loss: 0.0261 - mrcnn
_bbox_loss: 0.1322 - mrcnn_mask_loss: 0.2067 - val_loss: 1.4014 - val_rpn_cla
ss_loss: 0.0257 - val_rpn_bbox_loss: 0.6293 - val_mrcnn_class_loss: 0.0392 -
val_mrcnn_bbox_loss: 0.4140 - val_mrcnn_mask_loss: 0.2930
Epoch 13/50
50/50 [==============================] - 33s 654ms/step - loss: 0.9671 - rpn_
class_loss: 0.0119 - rpn_bbox_loss: 0.5211 - mrcnn_class_loss: 0.0448 - mrcnn
_bbox_loss: 0.1760 - mrcnn_mask_loss: 0.2131 - val_loss: 0.9678 - val_rpn_cla
ss_loss: 0.0102 - val_rpn_bbox_loss: 0.6220 - val_mrcnn_class_loss: 0.0259 -
val_mrcnn_bbox_loss: 0.1628 - val_mrcnn_mask_loss: 0.1470
Epoch 14/50
50/50 [==============================] - 33s 655ms/step - loss: 0.9855 - rpn_
class_loss: 0.0119 - rpn_bbox_loss: 0.6009 - mrcnn_class_loss: 0.0460 - mrcnn
_bbox_loss: 0.1435 - mrcnn_mask_loss: 0.1830 - val_loss: 1.2739 - val_rpn_cla
ss_loss: 0.0163 - val_rpn_bbox_loss: 0.7229 - val_mrcnn_class_loss: 0.0553 -
val_mrcnn_bbox_loss: 0.1847 - val_mrcnn_mask_loss: 0.2947
```

```
Epoch 15/50
50/50 [==============================] - 33s 664ms/step - loss: 0.8248 - rpn_
class_loss: 0.0143 - rpn_bbox_loss: 0.4894 - mrcnn_class_loss: 0.0387 - mrcnn
_bbox_loss: 0.1233 - mrcnn_mask_loss: 0.1590 - val_loss: 1.1371 - val_rpn_cla
ss_loss: 0.0142 - val_rpn_bbox_loss: 0.6885 - val_mrcnn_class_loss: 0.0338 -
val_mrcnn_bbox_loss: 0.2085 - val_mrcnn_mask_loss: 0.1922
Epoch 16/50
50/50 [==============================] - 34s 683ms/step - loss: 0.9747 - rpn_
class_loss: 0.0174 - rpn_bbox_loss: 0.4858 - mrcnn_class_loss: 0.0323 - mrcnn
_bbox_loss: 0.2314 - mrcnn_mask_loss: 0.2077 - val_loss: 1.3913 - val_rpn_cla
ss_loss: 0.0143 - val_rpn_bbox_loss: 0.8123 - val_mrcnn_class_loss: 0.0811 -
val_mrcnn_bbox_loss: 0.2768 - val_mrcnn_mask_loss: 0.2069
Epoch 17/50
50/50 [==============================] - 33s 658ms/step - loss: 0.9311 - rpn_
class_loss: 0.0135 - rpn_bbox_loss: 0.5278 - mrcnn_class_loss: 0.0810 - mrcnn
_bbox_loss: 0.1328 - mrcnn_mask_loss: 0.1761 - val_loss: 1.1120 - val_rpn_cla
ss_loss: 0.0174 - val_rpn_bbox_loss: 0.4591 - val_mrcnn_class_loss: 0.1246 -
val_mrcnn_bbox_loss: 0.2597 - val_mrcnn_mask_loss: 0.2512
Epoch 18/50
50/50 [==============================] - 36s 719ms/step - loss: 0.9906 - rpn_
class_loss: 0.0130 - rpn_bbox_loss: 0.5632 - mrcnn_class_loss: 0.0592 - mrcnn
_bbox_loss: 0.1578 - mrcnn_mask_loss: 0.1973 - val_loss: 1.0112 - val_rpn_cla
ss_loss: 0.0089 - val_rpn_bbox_loss: 0.5209 - val_mrcnn_class_loss: 0.0250 -
val_mrcnn_bbox_loss: 0.1765 - val_mrcnn_mask_loss: 0.2799
Epoch 19/50
50/50 [==============================] - 34s 682ms/step - loss: 0.7387 - rpn_
class_loss: 0.0134 - rpn_bbox_loss: 0.4190 - mrcnn_class_loss: 0.0269 - mrcnn
_bbox_loss: 0.1145 - mrcnn_mask_loss: 0.1649 - val_loss: 0.9900 - val_rpn_cla
ss_loss: 0.0100 - val_rpn_bbox_loss: 0.5868 - val_mrcnn_class_loss: 0.0186 -
val_mrcnn_bbox_loss: 0.1387 - val_mrcnn_mask_loss: 0.2360
Epoch 20/50
50/50 [==============================] - 32s 639ms/step - loss: 0.7415 - rpn_
class_loss: 0.0100 - rpn_bbox_loss: 0.3805 - mrcnn_class_loss: 0.0607 - mrcnn
_bbox_loss: 0.1268 - mrcnn_mask_loss: 0.1634 - val_loss: 0.7809 - val_rpn_cla
ss_loss: 0.0144 - val_rpn_bbox_loss: 0.2906 - val_mrcnn_class_loss: 0.0310 -
val_mrcnn_bbox_loss: 0.1987 - val_mrcnn_mask_loss: 0.2462
Epoch 21/50
50/50 [==============================] - 33s 664ms/step - loss: 0.7526 - rpn_
class_loss: 0.0142 - rpn_bbox_loss: 0.3732 - mrcnn_class_loss: 0.0490 - mrcnn
_bbox_loss: 0.1504 - mrcnn_mask_loss: 0.1658 - val_loss: 0.8358 - val_rpn_cla
ss_loss: 0.0092 - val_rpn_bbox_loss: 0.4532 - val_mrcnn_class_loss: 0.0028 -
val_mrcnn_bbox_loss: 0.1626 - val_mrcnn_mask_loss: 0.2080
Epoch 22/50
```

```
50/50 [==============================] - 35s 701ms/step - loss: 0.8432 - rpn_
class_loss: 0.0169 - rpn_bbox_loss: 0.4374 - mrcnn_class_loss: 0.0324 - mrcnn
_bbox_loss: 0.1455 - mrcnn_mask_loss: 0.2110 - val_loss: 0.8577 - val_rpn_cla
ss_loss: 0.0079 - val_rpn_bbox_loss: 0.5787 - val_mrcnn_class_loss: 0.0073 -
val_mrcnn_bbox_loss: 0.1017 - val_mrcnn_mask_loss: 0.1620
Epoch 23/50
50/50 [==============================] - 33s 658ms/step - loss: 0.7613 - rpn_
class_loss: 0.0098 - rpn_bbox_loss: 0.3986 - mrcnn_class_loss: 0.0370 - mrcnn
_bbox_loss: 0.1496 - mrcnn_mask_loss: 0.1662 - val_loss: 0.7850 - val_rpn_cla
ss_loss: 0.0107 - val_rpn_bbox_loss: 0.3796 - val_mrcnn_class_loss: 0.0339 -
val_mrcnn_bbox_loss: 0.2059 - val_mrcnn_mask_loss: 0.1548
Epoch 24/50
50/50 [==============================] - 33s 661ms/step - loss: 0.6658 - rpn_
class_loss: 0.0129 - rpn_bbox_loss: 0.3692 - mrcnn_class_loss: 0.0207 - mrcnn
_bbox_loss: 0.1031 - mrcnn_mask_loss: 0.1599 - val_loss: 0.9166 - val_rpn_cla
ss_loss: 0.0100 - val_rpn_bbox_loss: 0.5662 - val_mrcnn_class_loss: 0.0063 -
val_mrcnn_bbox_loss: 0.0905 - val_mrcnn_mask_loss: 0.2437
Epoch 25/50
50/50 [==============================] - 34s 671ms/step - loss: 0.8095 - rpn_
class_loss: 0.0149 - rpn_bbox_loss: 0.5017 - mrcnn_class_loss: 0.0496 - mrcnn
_bbox_loss: 0.0813 - mrcnn_mask_loss: 0.1621 - val_loss: 1.2540 - val_rpn_cla
ss_loss: 0.0051 - val_rpn_bbox_loss: 0.8020 - val_mrcnn_class_loss: 0.0336 -
val_mrcnn_bbox_loss: 0.1647 - val_mrcnn_mask_loss: 0.2487
Epoch 26/50
50/50 [==============================] - 34s 674ms/step - loss: 0.6864 - rpn_
class_loss: 0.0087 - rpn_bbox_loss: 0.3404 - mrcnn_class_loss: 0.0384 - mrcnn
_bbox_loss: 0.1280 - mrcnn_mask_loss: 0.1709 - val_loss: 0.9576 - val_rpn_cla
ss_loss: 0.0111 - val_rpn_bbox_loss: 0.5143 - val_mrcnn_class_loss: 0.0386 -
val_mrcnn_bbox_loss: 0.1697 - val_mrcnn_mask_loss: 0.2240
Epoch 27/50
50/50 [==============================] - 34s 676ms/step - loss: 0.8523 - rpn_
class_loss: 0.0177 - rpn_bbox_loss: 0.4374 - mrcnn_class_loss: 0.0620 - mrcnn
_bbox_loss: 0.1258 - mrcnn_mask_loss: 0.2094 - val_loss: 0.9856 - val_rpn_cla
ss_loss: 0.0183 - val_rpn_bbox_loss: 0.5802 - val_mrcnn_class_loss: 0.0232 -
val_mrcnn_bbox_loss: 0.1684 - val_mrcnn_mask_loss: 0.1955
Epoch 28/50
50/50 [==============================] - 33s 665ms/step - loss: 0.7437 - rpn_
class_loss: 0.0139 - rpn_bbox_loss: 0.3869 - mrcnn_class_loss: 0.0421 - mrcnn
_bbox_loss: 0.1306 - mrcnn_mask_loss: 0.1703 - val_loss: 0.8788 - val_rpn_cla
ss_loss: 0.0055 - val_rpn_bbox_loss: 0.4578 - val_mrcnn_class_loss: 0.0759 -
val_mrcnn_bbox_loss: 0.1896 - val_mrcnn_mask_loss: 0.1500
Epoch 29/50
50/50 [==============================] - 33s 664ms/step - loss: 0.6641 - rpn_
class_loss: 0.0125 - rpn_bbox_loss: 0.3666 - mrcnn_class_loss: 0.0618 - mrcnn
```

```
_bbox_loss: 0.0717 - mrcnn_mask_loss: 0.1515 - val_loss: 1.2376 - val_rpn_cla
ss_loss: 0.0096 - val_rpn_bbox_loss: 0.7253 - val_mrcnn_class_loss: 0.0900 -
val_mrcnn_bbox_loss: 0.1768 - val_mrcnn_mask_loss: 0.2360
Epoch 30/50
50/50 [==============================] - 33s 655ms/step - loss: 0.5305 - rpn_
class_loss: 0.0103 - rpn_bbox_loss: 0.2717 - mrcnn_class_loss: 0.0250 - mrcnn
_bbox_loss: 0.0858 - mrcnn_mask_loss: 0.1376 - val_loss: 0.5782 - val_rpn_cla
ss_loss: 0.0146 - val_rpn_bbox_loss: 0.1060 - val_mrcnn_class_loss: 0.0561 -
val_mrcnn_bbox_loss: 0.2242 - val_mrcnn_mask_loss: 0.1773
Epoch 31/50
50/50 [==============================] - 33s 661ms/step - loss: 0.6107 - rpn_
class_loss: 0.0128 - rpn_bbox_loss: 0.3209 - mrcnn_class_loss: 0.0237 - mrcnn
_bbox_loss: 0.0913 - mrcnn_mask_loss: 0.1620 - val_loss: 1.0684 - val_rpn_cla
ss_loss: 0.0081 - val_rpn_bbox_loss: 0.6970 - val_mrcnn_class_loss: 0.0139 -
val_mrcnn_bbox_loss: 0.2216 - val_mrcnn_mask_loss: 0.1278
Epoch 32/50
50/50 [==============================] - 34s 677ms/step - loss: 0.6779 - rpn_
class_loss: 0.0122 - rpn_bbox_loss: 0.3383 - mrcnn_class_loss: 0.0427 - mrcnn
_bbox_loss: 0.1218 - mrcnn_mask_loss: 0.1629 - val_loss: 0.9730 - val_rpn_cla
ss_loss: 0.0141 - val_rpn_bbox_loss: 0.3881 - val_mrcnn_class_loss: 0.1079 -
val_mrcnn_bbox_loss: 0.1425 - val_mrcnn_mask_loss: 0.3204
Epoch 33/50
50/50 [==============================] - 34s 670ms/step - loss: 0.5099 - rpn_
class_loss: 0.0085 - rpn_bbox_loss: 0.2388 - mrcnn_class_loss: 0.0217 - mrcnn
_bbox_loss: 0.1046 - mrcnn_mask_loss: 0.1363 - val_loss: 0.8117 - val_rpn_cla
ss_loss: 0.0067 - val_rpn_bbox_loss: 0.4346 - val_mrcnn_class_loss: 0.0875 -
val_mrcnn_bbox_loss: 0.1013 - val_mrcnn_mask_loss: 0.1816
Epoch 34/50
50/50 [==============================] - 33s 665ms/step - loss: 0.5187 - rpn_
class_loss: 0.0080 - rpn_bbox_loss: 0.2056 - mrcnn_class_loss: 0.0261 - mrcnn
_bbox_loss: 0.1010 - mrcnn_mask_loss: 0.1780 - val_loss: 0.9894 - val_rpn_cla
ss_loss: 0.0072 - val_rpn_bbox_loss: 0.6238 - val_mrcnn_class_loss: 0.0108 -
val_mrcnn_bbox_loss: 0.1539 - val_mrcnn_mask_loss: 0.1935
Epoch 35/50
50/50 [==============================] - 33s 663ms/step - loss: 0.4663 - rpn_
class_loss: 0.0086 - rpn_bbox_loss: 0.1940 - mrcnn_class_loss: 0.0355 - mrcnn
_bbox_loss: 0.0913 - mrcnn_mask_loss: 0.1369 - val_loss: 0.7886 - val_rpn_cla
ss_loss: 0.0122 - val_rpn_bbox_loss: 0.5112 - val_mrcnn_class_loss: 0.0132 -
val_mrcnn_bbox_loss: 0.0893 - val_mrcnn_mask_loss: 0.1627
Epoch 36/50
50/50 [==============================] - 34s 674ms/step - loss: 0.5016 - rpn_
class_loss: 0.0084 - rpn_bbox_loss: 0.2044 - mrcnn_class_loss: 0.0654 - mrcnn
_bbox_loss: 0.0722 - mrcnn_mask_loss: 0.1512 - val_loss: 0.7997 - val_rpn_cla
```

```
ss_loss: 0.0114 - val_rpn_bbox_loss: 0.4152 - val_mrcnn_class_loss: 0.0363 -
val_mrcnn_bbox_loss: 0.1518 - val_mrcnn_mask_loss: 0.1850
Epoch 37/50
50/50 [==============================] - 34s 674ms/step - loss: 0.5879 - rpn_
class_loss: 0.0113 - rpn_bbox_loss: 0.2687 - mrcnn_class_loss: 0.0325 - mrcnn
_bbox_loss: 0.1050 - mrcnn_mask_loss: 0.1704 - val_loss: 0.4812 - val_rpn_cla
ss_loss: 0.0027 - val_rpn_bbox_loss: 0.0886 - val_mrcnn_class_loss: 0.0948 -
val_mrcnn_bbox_loss: 0.1227 - val_mrcnn_mask_loss: 0.1724
Epoch 38/50
50/50 [==============================] - 33s 652ms/step - loss: 0.5808 - rpn_
class_loss: 0.0093 - rpn_bbox_loss: 0.2617 - mrcnn_class_loss: 0.0421 - mrcnn
_bbox_loss: 0.1065 - mrcnn_mask_loss: 0.1612 - val_loss: 1.5786 - val_rpn_cla
ss_loss: 0.0082 - val_rpn_bbox_loss: 0.8140 - val_mrcnn_class_loss: 0.4038 -
val_mrcnn_bbox_loss: 0.2236 - val_mrcnn_mask_loss: 0.1289
Epoch 39/50
50/50 [==============================] - 31s 628ms/step - loss: 0.5086 - rpn_
class_loss: 0.0073 - rpn_bbox_loss: 0.1683 - mrcnn_class_loss: 0.0612 - mrcnn
_bbox_loss: 0.1226 - mrcnn_mask_loss: 0.1492 - val_loss: 1.0686 - val_rpn_cla
ss_loss: 0.0063 - val_rpn_bbox_loss: 0.5805 - val_mrcnn_class_loss: 0.1328 -
val_mrcnn_bbox_loss: 0.1675 - val_mrcnn_mask_loss: 0.1815
Epoch 40/50
50/50 [==============================] - 31s 627ms/step - loss: 0.5313 - rpn_
class_loss: 0.0110 - rpn_bbox_loss: 0.2644 - mrcnn_class_loss: 0.0385 - mrcnn
_bbox_loss: 0.0664 - mrcnn_mask_loss: 0.1510 - val_loss: 0.6062 - val_rpn_cla
ss_loss: 0.0037 - val_rpn_bbox_loss: 0.3833 - val_mrcnn_class_loss: 0.0155 -
val_mrcnn_bbox_loss: 0.0670 - val_mrcnn_mask_loss: 0.1367
Epoch 41/50
50/50 [==============================] - 33s 659ms/step - loss: 0.5273 - rpn_
class_loss: 0.0091 - rpn_bbox_loss: 0.2155 - mrcnn_class_loss: 0.0412 - mrcnn
_bbox_loss: 0.1131 - mrcnn_mask_loss: 0.1485 - val_loss: 0.9857 - val_rpn_cla
ss_loss: 0.0051 - val_rpn_bbox_loss: 0.6867 - val_mrcnn_class_loss: 0.0449 -
val_mrcnn_bbox_loss: 0.1062 - val_mrcnn_mask_loss: 0.1427
Epoch 42/50
50/50 [==============================] - 33s 655ms/step - loss: 0.5280 - rpn_
class_loss: 0.0103 - rpn_bbox_loss: 0.2633 - mrcnn_class_loss: 0.0369 - mrcnn
_bbox_loss: 0.0670 - mrcnn_mask_loss: 0.1506 - val_loss: 1.3616 - val_rpn_cla
ss_loss: 0.0327 - val_rpn_bbox_loss: 1.0616 - val_mrcnn_class_loss: 0.0269 -
val_mrcnn_bbox_loss: 0.1111 - val_mrcnn_mask_loss: 0.1294
Epoch 43/50
50/50 [==============================] - 34s 676ms/step - loss: 0.5386 - rpn_
class_loss: 0.0114 - rpn_bbox_loss: 0.2349 - mrcnn_class_loss: 0.0466 - mrcnn
_bbox_loss: 0.1105 - mrcnn_mask_loss: 0.1352 - val_loss: 1.0406 - val_rpn_cla
ss_loss: 0.0115 - val_rpn_bbox_loss: 0.2693 - val_mrcnn_class_loss: 0.0879 -
val_mrcnn_bbox_loss: 0.3130 - val_mrcnn_mask_loss: 0.3589
```

```
Epoch 44/50
50/50 [==============================] - 35s 706ms/step - loss: 0.5267 - rpn_
class_loss: 0.0083 - rpn_bbox_loss: 0.2056 - mrcnn_class_loss: 0.0424 - mrcnn
_bbox_loss: 0.0949 - mrcnn_mask_loss: 0.1755 - val_loss: 0.6450 - val_rpn_cla
ss_loss: 0.0091 - val_rpn_bbox_loss: 0.2131 - val_mrcnn_class_loss: 0.0176 -
val_mrcnn_bbox_loss: 0.2218 - val_mrcnn_mask_loss: 0.1834
Epoch 45/50
50/50 [==============================] - 39s 782ms/step - loss: 0.5541 - rpn_
class_loss: 0.0111 - rpn_bbox_loss: 0.2744 - mrcnn_class_loss: 0.0386 - mrcnn
_bbox_loss: 0.0812 - mrcnn_mask_loss: 0.1488 - val_loss: 1.1043 - val_rpn_cla
ss_loss: 0.0104 - val_rpn_bbox_loss: 0.3476 - val_mrcnn_class_loss: 0.0897 -
val_mrcnn_bbox_loss: 0.3340 - val_mrcnn_mask_loss: 0.3226
Epoch 46/50
50/50 [==============================] - 32s 649ms/step - loss: 0.4966 - rpn_
class_loss: 0.0164 - rpn_bbox_loss: 0.2024 - mrcnn_class_loss: 0.0317 - mrcnn
_bbox_loss: 0.0925 - mrcnn_mask_loss: 0.1536 - val_loss: 0.7521 - val_rpn_cla
ss_loss: 0.0175 - val_rpn_bbox_loss: 0.3457 - val_mrcnn_class_loss: 0.0351 -
val_mrcnn_bbox_loss: 0.2000 - val_mrcnn_mask_loss: 0.1537
Epoch 47/50
50/50 [==============================] - 35s 708ms/step - loss: 0.4430 - rpn_
class_loss: 0.0063 - rpn_bbox_loss: 0.1752 - mrcnn_class_loss: 0.0350 - mrcnn
_bbox_loss: 0.0845 - mrcnn_mask_loss: 0.1420 - val_loss: 1.0256 - val_rpn_cla
ss_loss: 0.0182 - val_rpn_bbox_loss: 0.6128 - val_mrcnn_class_loss: 0.1406 -
val_mrcnn_bbox_loss: 0.1192 - val_mrcnn_mask_loss: 0.1348
Epoch 48/50
50/50 [==============================] - 34s 690ms/step - loss: 0.4131 - rpn_
class_loss: 0.0045 - rpn_bbox_loss: 0.1494 - mrcnn_class_loss: 0.0340 - mrcnn
_bbox_loss: 0.0790 - mrcnn_mask_loss: 0.1461 - val_loss: 1.6007 - val_rpn_cla
ss_loss: 0.0163 - val_rpn_bbox_loss: 1.1491 - val_mrcnn_class_loss: 0.0577 -
val_mrcnn_bbox_loss: 0.1596 - val_mrcnn_mask_loss: 0.2181
Epoch 49/50
50/50 [==============================] - 32s 647ms/step - loss: 0.4665 - rpn_
class_loss: 0.0114 - rpn_bbox_loss: 0.1986 - mrcnn_class_loss: 0.0242 - mrcnn
_bbox_loss: 0.0777 - mrcnn_mask_loss: 0.1547 - val_loss: 0.6317 - val_rpn_cla
ss_loss: 0.0061 - val_rpn_bbox_loss: 0.3042 - val_mrcnn_class_loss: 0.0493 -
val_mrcnn_bbox_loss: 0.0960 - val_mrcnn_mask_loss: 0.1761
Epoch 50/50
50/50 [==============================] - 33s 658ms/step - loss: 0.4245 - rpn_
class_loss: 0.0081 - rpn_bbox_loss: 0.1827 - mrcnn_class_loss: 0.0303 - mrcnn
_bbox_loss: 0.0683 - mrcnn_mask_loss: 0.1352 - val_loss: 0.6966 - val_rpn_cla
ss_loss: 0.0122 - val_rpn_bbox_loss: 0.3828 - val_mrcnn_class_loss: 0.0093 -
val_mrcnn_bbox_loss: 0.1142 - val_mrcnn_mask_loss: 0.1782
```

Python predictions.ipynb

In [1]:

```
import os
import sys
import itertools
import math
import logging
import json
import re
import random
from collections import OrderedDict
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.lines as lines
from matplotlib.patches import Polygon
import skimage

# Root directory of the project
ROOT_DIR = 'Mask_RCNN-master 3'

# Import Mask RCNN
sys.path.append(ROOT_DIR)  # To find local version of the library
from mrcnn import utils
from mrcnn import visualize
from mrcnn.visualize import display_images
```

```
import mrcnn.model as modellib
from mrcnn.model import log

import Cell
import cv2
```

```
Using TensorFlow backend.
```

In [2]:

```
model_dir = "../logs/cell20200529T1324/"             # 60 epochs
model_file = "mask_rcnn_cell_0050.h5"
coco_path = os.path.abspath(model_dir + model_file)
```

In [3]:

```
model_dir = "../logs/cell20200529T1324/"             # 200 epochs
model_file = "mask_rcnn_cell_0050.h5"
coco_path = os.path.abspath(model_dir + model_file)
```

In [4]:

```
model = modellib.MaskRCNN(mode="inference", config=Cell.config, model_dir=mod
el_dir)
```

```
WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/ke
ras/backend/tensorflow_backend.py:517: The name tf.placeholder is deprecated.
Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/ke
ras/backend/tensorflow_backend.py:74: The name tf.get_default_graph is deprec
ated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/ke
ras/backend/tensorflow_backend.py:4138: The name tf.random_uniform is depreca
ted. Please use tf.random.uniform instead.

WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/ke
ras/backend/tensorflow_backend.py:1919: The name tf.nn.fused_batch_norm is de
precated. Please use tf.compat.v1.nn.fused_batch_norm instead.

WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/ke
ras/backend/tensorflow_backend.py:3976: The name tf.nn.max_pool is deprecated
. Please use tf.nn.max_pool2d instead.

WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/ke
ras/backend/tensorflow_backend.py:2018: The name tf.image.resize_nearest_neig
hbor is deprecated. Please use tf.compat.v1.image.resize_nearest_neighbor ins
tead.
```

```
WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/mr
cnn/model.py:341: The name tf.log is deprecated. Please use tf.math.log inste
ad.

WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/mr
cnn/model.py:399: add_dispatch_support.<locals>.wrapper (from tensorflow.pyth
on.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/mr
cnn/model.py:423: calling crop_and_resize_v1 (from tensorflow.python.ops.imag
e_ops_impl) with box_ind is deprecated and will be removed in a future versio
n.
Instructions for updating:
box_ind is deprecated, use box_indices instead
WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/mr
cnn/model.py:723: The name tf.sets.set_intersection is deprecated. Please use
tf.sets.intersection instead.

WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/mr
cnn/model.py:725: The name tf.sparse_tensor_to_dense is deprecated. Please us
e tf.sparse.to_dense instead.

WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.7/site-packages/mr
cnn/model.py:775: to_float (from tensorflow.python.ops.math_ops) is deprecate
d and will be removed in a future version.
Instructions for updating:
Use `tf.cast` instead.
```

In [5]:

```
model.load_weights(coco_path, by_name=True)
```

```
Re-starting from epoch 50
```

In [6]:

```
# Function taken from utils.dataset
def load_image(image_path):
    """Load the specified image and return a [H,W,3] Numpy array.
    """
    # Load image
    image = skimage.io.imread(image_path)
    # If grayscale. Convert to RGB for consistency.
    if image.ndim != 3:
        image = skimage.color.gray2rgb(image)
    # If has an alpha channel, remove it for consistency
    if image.shape[-1] == 4:
        image = image[..., :3]
```

```
    return image
```

In [7]:

```
def get_ax(rows=1, cols=1, size=16):
    """Return a Matplotlib Axes array to be used in
    all visualizations in the notebook. Provide a
    central point to control graph sizes.

    Adjust the size attribute to control how big to render images
    """
    _, ax = plt.subplots(rows, cols, figsize=(size*cols, size*rows))
    return ax
```

In [8]:

```
import os
for root, dirs, files in os.walk('/home/ubuntu/github/2020MaskRCNN/inputImage
s/'):
    for file in files:
        if file.endswith('.jpg'):
            img_path = os.path.join(root,file)
            image = load_image(img_path)
            skimage.io.imshow(image)
            plt.show()

            dataset = Cell.HSYAADataset()
            dataset.load_data("dataset/", "train")
            dataset.prepare()

            # Run object detection
            results = model.detect([image], verbose=1)

            # Display results
            ax = get_ax(1)
            r = results[0]
            a = visualize.display_instances(image, r['rois'], r['masks'], r['
class_ids'],
                                            dataset.class_names, r['scores'], ax=
ax,
                                            title="Predictions")
            file_name = "splash_{:%Y%m%dT%H%M%S}.png".format(datetime.datetim
e.now())
#             splash = Cell.color_splash(image, r['scores'])
#             skimage.io.imsave(file_name, splash)
            name = '/home/ubuntu/github/2020MaskRCNN/main/output/' + file
```

```
            plt.savefig(name,bbox_inches='tight', pad_inches=-0.5,orientation
= 'landscape')
```

Results from mass predictions