Cancer Cell Detection with Mask R-CNN

By
Ryan Wong

A THESIS

submitted to
Oregon State University
Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Physics
(Honors Scholar)

Presented 6/3/21
Commencement June 2021
AN ABSTRACT OF THE THESIS OF

Ryan Wong for the degree of Honors Baccalaureate of Science in Physics presented on 6/3/21. Title: Cancer Cell Detection with Mask R-CNN

Abstract approved:

__________________________
Bo Sun

Instance segmentation, the classification and localization of objects in an image, is a problem in cellular biophysics due to the physiological relevance of cell morphology. Particularly, cancer cells migrate in tissue space by changing their body shape similarly to how humans extend their limbs to swim through water. Monitoring such shape changes in a dense cell population requires high throughput methods to process and analyze microscopy timelapse video frames. This thesis explores the use of a deep learning algorithm in the instance segmentation of MDA-MB-231 cancer cells at varying densities. We employ Mask R-CNN and characterize the model’s performance with varying hyperparameters, including learning rate, gradient clip norm, learning momentum, non-max suppression, and anchor ratios. We apply performance metrics, namely the precision-recall curve and mean average precision, to validate the model’s detections. We then explore how well Mask R-CNN generalizes from a low cell density training set to high density images. Our results demonstrate Mask R-CNN as a reliable instance segmentation model to segment dense cell populations, and serves as a starting point to develop an automated pipeline for cell experiments.

Key Words: Deep Learning, Mask R-CNN, Cancer Cell, Cellular Biophysics

Corresponding e-mail address: wongryan@oregonstate.edu
Cancer Cell Detection with Mask R-CNN

By
Ryan Wong

A THESIS

submitted to
Oregon State University
Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Physics
(Honors Scholar)

Presented 6/3/21
Commencement June 2021
Honors Baccalaureate of Science in Physics project of Ryan Wong presented on 6/3/21.

APPROVED:

______________________________
Bo Sun, Mentor, representing Physics

______________________________
Janet Tate, Committee Member, representing Physics Department

______________________________
Chris Eddy, Committee Member, representing Physics Department

______________________________
Toni Doolen, Dean, Oregon State University Honors College

I understand that my project will become part of the permanent collection of Oregon State University Honors College. My signature below authorizes release of my project to any reader upon request.

______________________________
Ryan Wong, Author
Chapter 1

Introduction

1.1 Biological Cell Morphology

Cell shape is essential. As the basic unit of life, cells must assume the most suitable morphology to accomplish each biological task. Motile animal cells, such as neutrophils, extend protrusive structures known as filopodia to achieve motility on a substrate [1]. Being heavily coupled with external stimuli, cell morphology also provides information about its environment. For example, migrating breast cancer cells have been found to be more filopodial in anisotropic extracellular matrix (ECM), as opposed to being more rounded in a homogeneous ECM (Figure 1.1) [2]. In medicine, a cell’s shape, such as loss of polarity during breast cancer progression, is associated with its health and pathogenicity [3]. Furthermore, cell morphology offers a window into the cell’s internal properties. For instance, cell curvature has been shown to influence activated receptor concentrations at the plasma membrane, thus regulating a cell’s ability to sense and communicate [4], and protrusions of filopodia to initiate a crawling locomotion indicates coordinated biochemical interplay between actin filaments and the cell membrane [1]. Morphology is therefore a highly physiologically relevant knowledge base of cell state.

1.2 Current Techniques in Cell Analysis

Currently, cell flow cytometry (FCM) is a standard high throughput method to collect morphology data. By forcing water or a saline solution through a chamber under pressure, a flow system can hydrodynamically deliver single particles past a point at which they are imaged (Figure 1.2). Autofluorescent or fluorescence-labeled cells are then detected with a focused laser beam [5]. Combined with the spatial resolution of digital microscopy, FCM is convenient and practical for clinical applications, where cells are imaged directly in bodily fluids [5].
However, though FCM is convenient for single cell analysis, forcing cells through a narrow point means tissue architecture and cell-cell interactions are nonexistent. This prevents the quantification of cell morphology in the context of its natural microenvironment and collective cellular behaviors.

To record in vitro experiments closer to the cell’s biological context, image cytometry must be used. This technique uses an optical system, such as confocal microscopy, to statically image a large number of cells. Recent developments in automation, such as programmable platforms and digital cameras, have allowed confocal microscopes to generate time-lapse videos in three dimensions. The major bottleneck in analyzing
morphology is segmenting, or tracing out, single cells. Manually segmenting cells were an industry standard and depending on how confluent an image is, processing a single image may take hours. The image analysis community has taken advantage of new software technologies to automate this process.

In the last 10 years artificial intelligence has blossomed in biophysics, especially for cell analysis. To establish tools for eliminating the segmentation bottleneck, open source, user friendly software has been developed: machine learning networks such as Ilastik [7] and CellProfiler [8] allow the user to input image with preset features – structured data – and classify objects based on those features. On the other hand, deep learning allows users to input only one image – unstructured data – and let the network learn features of the data by itself, and create a model.

Already well established in cell biology are open-source frameworks that use semantic segmentation, such as U-Net [9], to separate cells from background but does not separate cells from each other. It has thus become easy to recognize multiple cells or perform detailed analysis of single cells.

More desirable but extremely limited in high density tumor experiments is the analysis of individual behavior in a many-celled environment, where cells are not only differentiated from the background but also from each other. In computer vision, this problem is known as instance segmentation. In the following chapter, the details of instance segmentation will be elaborated and a framework will be introduced.

1.3 Objectives

In this thesis, we study the application of instance segmentation framework Mask RCNN to migrating MDA-MB-231 breast cancer cells. The objective is to find suitable hyperparameters, test Mask RCNN’s efficacy in cell segmentation, and probe its ability to generalize to high cell densities given a low density training set.
Chapter 2

Computer Vision and Deep Learning Basics

With to high level deep learning libraries, it is possible to build and evaluate deep learning models without learning their back end. However, to fully appreciate neural networks, we start with a low-level, mathematical understanding of deep learning before applying the prepackaged software.

2.1 Instance Segmentation

With how much detail can we infer the identity and location of objects in an image? Such is the question posed by object recognition. As a research area, it seeks to continuously improve from coarse-grained inference towards fine-grained inference [10]. Currently, instance segmentation is the finest-grained object recognition problem tackled by the deep learning community. Since we want to proceed with the end goal in mind, we’ll first provide a high level overview of instance segmentation and three closely related problems: classification, semantic segmentation, and localization.

In an image containing objects, each object belongs to a class as defined by the user. For example, an image containing a bottle, a cup, and a cube may have the classes “bottle”, “cup”, and “cube”. Classification refers to the process of predicting the classes of objects in an input image. Semantic segmentation seeks to perform per-pixel classification on the image such that every pixel is labeled with the class of an object [11]. In other words, a classification algorithm returns the list of classes [bottle, cup, cube] while a semantic segmentation algorithm classifies each pixel as part of a bottle, cup, cube or background (Figure 2.1a,c).

One can also perform inference on spatial location – localization. While localization is considered to be more coarse-grained because it doesn’t consider individual pixels, it provides extra information which semantic segmentation does not, which is the location of individual objects. It is therefore advantageous to solve both the localization and semantic segmentation problem simultaneously to extract the richest
possible information from the input image. Such is the goal of instance segmentation. Taking our previous example, while localization returns the spatial coordinates of each bottle, cup and cube or the region around them, instance segmentation classifies each pixel as bottle, cup, cube, or background and returns the spatial coordinates of each object. (Figure 2.1b,d).

For the cell biologist, instance segmentation is the ideal tool during the data processing step. From localization and per-pixel classification, one can obtain a time series from which to extract spatial and morphological (morpho-) dynamics. Deep learning has been of interest in the last decade because of its ability to automate instance segmentation. In subsequent sections in this chapter, we will introduce the mathematical background of deep learning and successful modern frameworks.

![Figure 2.1: Object recognition problems ranked from coarse-grained to fine-grained inference: classification, localization, semantic segmentation, instance segmentation. Image taken from [11].](image)

### 2.2 What Learning Algorithms Do

According to Mitchell, a learning algorithm such as a neural network learns from an experience $E$ with respect to a task $T$ and performance measure $P$. Its performances at $T$, as measured by $P$, should improve with $E$ [12].
2.3 Artificial Neurons

Neural networks, inspired by neuroscience [13], utilise interconnected artificial neurons to store and evolve their knowledge [14]. In this section we outline the mathematics of this process, using feedforward neural networks as a starting point.

The basic unit of a neural network is the artificial neuron, which receive inputs from other units to compute an output. Mathematically, each neuron \( n_i \) calculates an output signal as a function of the weighted sum of its inputs and a bias value

\[
y_i = f_i(\sum_{j=1}^{n} w_{ij}x_j + b_i),
\]

(2.1)

One can write Equation 2.1 as

\[
y_i = f_i(a_i),
\]

(2.2)

where \( a_i \) is known as the activation of \( n_i \), and \( f_i \) is the nonlinear activation function. Although not commonly used, one can imagine that if \( f_i \) is a step function, the neuron would 'activate' to a constant value if \( \sum_{j=1}^{n} w_{ij}x_j > b_i \). Common functions for \( f_i \) include the rectified linear unit (ReLU) (Equation 2.3), which scales linearly after zero, and sigmoid (Equation 2.4), which provides an uncertainty to the activation by allowing nonzero output when \( \sum_{j=1}^{n} w_{ij}x_j < b_i \) [14]:

\[
f(a) = \max\{0, a\},
\]

(2.3)

\[
f(a) = \frac{1}{1 + e^{-ca}},
\]

(2.4)

where \( c \) is the degree of uncertainty of activation by the sigmoid function.

The artificial neuron is inspired by its biological counterpart, which accumulates biological signals and non-linearly computes their own activation [13]. As one can imagine, the smallest neural network contains just one artificial neuron (Figure 2.2), coined as the perceptron by Frank Rosenblatt in 1958 [15].

![Figure 2.2: Qualitative representation of a perceptron, a one-neuron neural network. It contains an arbitrary number of numerical inputs and one numerical output.](image)

A natural next step is to introduce the behavior of a network of many neurons. A neural network contains an input layer that takes input data \( \mathbf{X} \), an output layer
which yields the final output, and zero or more hidden layers in between. The output of each layer is defined by the vector $Y$, where each entry $y_i$ is the function described in Equation 2.1. The input and output vectors can be written as

$$X = [x_1, x_2, \ldots]^T,$$

(2.5)

and

$$Y = [y_1, y_2, \ldots]^T. $$

(2.6)

As the neural network is a sequential series of layers, each layer only takes input from the one immediate preceding it. The value of $n$ in $y_i$ of the current layer is therefore the length of $Y$ in the previous layer.

### 2.4 Optimization

#### 2.4.1 Cost and Loss Functions

To optimize parameters, we need some measure of error. Since Equation 2.1 contains unknown parameters, a loss function $L(\theta)$ ($\theta$ is some parameter) can measure the error incurred by the network during one training example. We can obtain the expectation value of $L$ across the entire training set to obtain the cost function $J(\theta)$, with a goal of minimizing $J$ with respect to $\theta$. Though we’ve made this distinction between loss and cost functions, both terms are generally used interchangeably. Gradient descent (GD) is a popular technique which takes $J$ at a particular $\theta$ value and proposes a new value $\theta'$ [13]. The change between $\theta$ and $\theta'$ is determined by the direction in which $J$ decreases the fastest, or the gradient, with respect to the entire training set:

$$\theta' = \theta - \epsilon \nabla_{\theta} J(\theta),$$

(2.7)

where $\epsilon$, the learning rate, is the (positive) size of each GD step [13].

#### 2.4.2 Stochastic Gradient Descent

With large amounts of data, GD becomes too computationally expensive. Stochastic gradient descent is an extension of GD that updates $\theta$ once per training example on each step. This introduces stochasticity, where the gradient can fluctuate with respect each training example. Here we formally define $J$ as the expectation value of $L$

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(x^{(i)}, y^{(i)}, \theta),$$

(2.8)
where $\boldsymbol{\theta}$ is a parameter vector and $m$ is the number of training examples. $L$ is calculated per-example and is chosen according to the model and task. As the training set size $m$ grows, so does the computational cost [13]. A single GD step can thus become unfeasible. Stochastic gradient descent (SGD) solves this problem by making the gradient an approximation [13]. Particularly, we can sample a minibatch of the training set $B = [x^{(1)}, ..., x^{(m')}]$, where $m' \leq m$ is held constant as $m$ increases. The estimated gradient thus becomes

$$g = \frac{1}{m'} \nabla \sum_{i=1}^{m'} L(x^{(i)}, y^{(i)}, \boldsymbol{\theta}),$$  \hspace{1cm} (2.9)$$

and the parameter $\theta$ is updated per SGD step as

$$\theta' = \theta - \epsilon g.$$  \hspace{1cm} (2.10)$$

We have thus introduced a way to find parameters $w$ and $b$ for each neuron based on the gradient of a given cost function. In subsequent chapters, and mention of training refers to performing SGD or a similar optimization method to a neural network. Variations of SGD and specific parameters are discussed in Chapter 5.
Chapter 3
Mask R-CNN

3.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) are specialized neural networks for processing grid-like data [13]. They have been especially practical in processing images, which are a two dimensional grid of pixel values [16]. A general CNN consists of four components: an image sampling convolution layer, a dimension-reducing pooling layer, a feature-extracting activation function, and a classifying fully connected layer. Here we introduce convolutional layers, which are often employed in more complex architectures.

![General components of a CNN](image)

Figure 3.1: General components of a CNN [17].

3.1.1 Convolution Layer

As its name suggests, the convolution layer employs the convolution operation

\[
S(i, j) = (I \ast K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n),
\]  

(3.1)

where \( I \) is the image matrix, \( K \) is a two-dimensional square matrix of chosen parameters known as a kernel [13]. The convolution is a linear operation and is therefore commutative, meaning that an equivalent expression is

\[
S(i, j) = (K \ast I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n),
\]  

(3.2)
such that we can preset the range of valid $m$ and $n$ values, also known as the kernel size, based on the image dimensions. A similar function, often also referred to as convolution, is

$$S(i, j) = (K * I)(i, j) = \sum_{m} \sum_{n} I(i + m, j + n)K(m, n), \quad (3.3)$$

where the kernel is flipped [13]. $S(i, j)$ is known as the feature map because it captures the relevant characteristics of the convoluted region, or the receptive field [16]. One can then iterate, or slide, the kernel across all points of the image in which the kernel fully fits and obtain a feature map for the entire image. Figure 3.2 shows one such operation with a 2x2 kernel on a 3x4 image, which creates a 2x3 feature map.

![Figure 3.2: Convolution operation with a 2x2 kernel calculates a 2x3 feature map from a 3x4 image. [13].](image)

So what does the convolution do in the context of image data? They decrease computational cost. Instead of a fully connected network, where every output neuron interacts with every input neuron, convolutions allow for CNNs to be sparsely connected [13]. By making the kernel $K$ smaller than the input image $I$, we produce a feature map that feeds fewer pixels into the next layer.
3.2 Mask R-CNN

Mask R-CNN [18] is an instance segmentation framework that detects objects while generating segmentation masks for each instance. It builds on its predecessors Faster R-CNN [19] and Fast R-CNN [20]: Fast R-CNN performs classification and bounding-box regression on features extracted from candidate bounding boxes, Faster R-CNN adds a preceding stage known as a region proposal network, and Mask R-CNN additionally generates a object masks.

3.2.1 Backbone

Mask RCNN uses a convolutional backbone architecture to compute feature maps from an input image [18]. Particularly, He et al. used ResNet [21] with a feature pyramid network (FPN) [22]. FPN is not to be confused with a featured image pyramid, which employs multi-scale images and computes feature maps at each scale, and instead takes a single-scale image as input and outputs feature maps of different scales. FPN consists of a bottom-up pathway, a top-down pathway, and lateral connections. The bottom-up pathway is the feedforward computation of the convolutional backbone, which portions the CNN into stages based on the output size of each layer and samples the last output of each stage as the feature map of that particular scale to create a hierarchical feature pyramid. The qualitative architecture and pathway directions of the FPN are shown in Figure 3.3. In ResNet, conv2 through conv5 are used with respective strides \(\{4, 8, 16, 32\}\) pixels, corresponding to a scaling step of 2, to create feature maps \(\{C_2, C_3, C_4, C_5\}\).

![Figure 3.3: FPN architecture. Blue line thickness indicates semantic strength [22].](image)

In the top-down pathway, the semantically stronger high level feature maps are interpolated to produce higher resolution low level features, again with a scaling step of 2. They are enhanced by lateral connections, 1x1 convolutional layers, to corresponding levels in the bottom-up feature pyramid. The pathway begins by applying the 1x1 convolution to the coarsest feature map in the bottom-up pathway, and the interpolation is iterated until a feature pyramid \(\{P_2, P_3, P_4, P_5\}\) is generated.
3.2.2 Region Proposal Network

Regions of interest (RoIs), or candidate bounding boxes, can then be identified on each level of the pyramid. In Fast R-CNN, an RoI of width $w$ and height $h$ is assigned to a level $P_k$ using

$$k = [k_0 + \log_2(\sqrt{wh}/224)],$$

(3.4)

where 224 is the canonical ImageNet pre-training size and, as is the case in Mask R-CNN, $k_0 = 4$ is the target level on which an RoI is $wxh = 224^2$ is mapped to, corresponding to $P_4$ being set as the single-scale feature map in Fast R-CNN. In Equation 3.4, smaller RoI’s are mapped to a finer resolution level (lower on the pyramid).

To generate RoIs, Mask R-CNN uses the region proposal network (RPN) introduced in Faster R-CNN [19]. The RPN is a CNN that slides a mini-network over the feature map, where an $n \times n$ convolutional layer takes a spatial window of the same size, followed by two sibling $1 \times 1$ convolutional layers for box-regression ($\text{reg}$) and classification ($\text{cls}$).

Adhering to the notation of Ren et al., a preset maximum of $k$ region proposals are made in each sliding window location. The $\text{reg}$ layer therefore has $4k$ outputs corresponding to the four corner coordinates, and the $\text{cls}$ layer has $2k$ outputs corresponding to the probabilities of object or not object. The value of $k$ indicates $k$ reference boxes of preset scales and aspect ratios (Figure 3.4), referred to as anchors. Therefore, in a feature map of size $W \times H$, there are $WHk$ anchors to compute.

Figure 3.4: A maximum of $k$ region proposals are made at each sliding window location according to the number of preset anchors. This generates $2k$ probability scores and $4k$ corner coordinates [19]
3.2.3 ROIAlign

The ROIAlign layer extracts a $7 \times 7$ feature map from each RoI to unify the output size of the RPN, and is the improved iteration of Fast R-CNN’s RoIPool [18][20]. RoIPool quantizes via convolution a continuous RoI by first computing $[x/16]$ where $x$ is effectively continuous, 16 is the feature map stride and $[\cdot]$ is rounding, further portioning the quantized RoI into $7 \times 7$ bins, then aggregating the feature values using max pooling. The quantizations introduce misalignments between the original RoI and the features, which is acceptable for classification but can lower the accuracy of pixel-resolution masks. Instead of quantization and rounding to bins, RoIAlign bilinearly interpolates the values of 4 sampling points in each RoI bin before feeding it into a pooling layer (Figure 3.5).

![Figure 3.5: ROIAlign bilinearly interpolates 4 sampling points in each RoI bin using the feature map instead of rounding RoI boundaries to feature map bins, [18].](image)

3.2.4 Network Head

The network head applies classification, bounding-box regression, and mask predictions to each RoI [18]. Similar to the backbone, it has a Resnet and FPN architecture but additionally adds a mask-predicting CNN branch (Figure 3.6).

3.2.5 Loss Function

Mask R-CNN employs a multi-task loss function which combines the loss of classification, localization (bounding box), and segmentation masks.
Figure 3.6: The network head extends the Resnet and FPN architecture with a convolutional branch. Included are conv (3x3), deconv (2x2 with stride 2), and fully connected (fc) layers. Conv layers preserve 2D resolution through padding, deconv layers increase it, and fc layers preserve 3D resolution [18].

\[
L = L_{cls} + L_{box} + L_{mask},
\]  \hspace{1cm} (3.5)

\[
L_{cls}, \text{ the classification loss, is a log loss function defined as}
\]

\[
L_{cls} = \frac{1}{N_{cls}} \sum_i -p_i^* \log p_i - (1 - p_i^*) \log(1 - p_i),
\]  \hspace{1cm} (3.6)

where the sum is over the number of anchors, \(N_{cls}\) is the mini-batch size (256), \(p_i^*\) is the ground truth label of whether anchor \(i\) is an object, and \(p_i\) is the predicted probability of anchor \(i\) being an object.

\(L_{box}\), the bounding box loss, sums over the coordinates of all bounding boxes

\[
L_{box} = \sum_i L_{smooth}^1 (t_i^u - \nu_i)
\]  \hspace{1cm} (3.7)

where \(u\) is the class label, \(t_i^u\) is the predicted bounding box, and \(\nu\) is the true bounding box. \(t\) and \(\nu\) both have the format \((x, y, w, h)\). \(L_{smooth}^1\), shown in Figure 3.7 is the smooth L1 loss function

\[
L_{smooth}^1 = \begin{cases}
0.5x^2 & \text{if } |x| < 1 \\
|x| - 0.5 & \text{otherwise}
\end{cases}
\]  \hspace{1cm} (3.8)

\(L_{mask}\), the mask loss, is the average binary cross-entropy loss

\[
L_{mask} = -\frac{1}{m^2} \sum_{1 \leq i, j \leq m} [y_{ij} \log \hat{y}_{ij}^k + (1 - y_{ij}) \log 1 - \hat{y}_{ij}^k]
\]  \hspace{1cm} (3.9)

where \(y_{ij}\) is the label of a pixel \((i, j)\) in the ground truth mask for the region of size \(m \times m\). \(\hat{y}_{ij}^k\) is the predicted label of the pixel for the class \(k\).
Figure 3.7: $L_1^{smooth}$ function [23].
Chapter 4

Methodology

4.1 Instance Segmentation on Cell Migration Data

Our data set contains 105 10x magnified confocal images of MDA-MB-231 breast cancer cells during migration. The majority of these images are of low cell density with a small fraction of high density organoids. The images were maximum-projected onto two dimensions in the z-direction. In each image, every cell was polygon-outlined using the VGG Image Annotator (VIA) to generate ground truth masks, which were stored in a .json file [24]. Mask R-CNN, the instance segmentation model, is trained on 90 images for 20 epochs at 2 training steps per epoch. We observe the progression of the training and validation loss as a function of training step to find SGD parameters, namely learning rate, gradient clip norm and learning momentum, and use inference to find other hyperparameters, namely non-max suppression and anchor ratios (introduced in Chapter 5). We then apply metrics to evaluate the model’s performance on a validation set of 10 images. Additionally, we perform inference on a test set of 5 high cell density foreign images.

4.2 Performance Metrics

4.2.1 Precision Recall Curve

The precision-recall (PR) curve describes the trade-off between the relevance of the prediction (i.e. true positives) and how many predictions are returned. It is commonly used for evaluating classifiers across multiple thresholds [25]. Precision is defined as

\[
\text{Precision} = \frac{TP}{TP + FP},
\]  

where \(TP\) is the number of true positives and \(FP\) is the number of false positives. Recall is defined as

\[
\text{Recall} = \frac{TP}{TP + FN},
\]  

where \(FN\) is the number of false negatives.
Recall = \frac{TP}{TP + FN}, \quad (4.2)

where $FN$ is the number of false negatives. The two terms represent the number of accurate positive predictions in the context of all positive predictions and all positive ground truths, respectively. For example, a classifier that processes 10 photos of 5 dogs and 5 cats will have a precision of 0.5 and recall of 0.4 if it identifies 2 dogs and 2 cats as dogs. One can then plot precision against recall to obtain a PR plot and take the area under the curve ($AUC - PR$).

The typical classifier lies between the following thresholds (Figure 4.2: employing a “baseline” classifier that labels all examples positive to a balanced data set means that precision is exactly 0.5, hence $AUC - PR = 0.5$. On the other hand, the perfect classifier has $AUC - PR = 1$ because the precision is, by definition, exactly 1 across all recall values. We use the PR curve to identify the per-pixel accuracy of Mask-RCNN’s classification branch. That is, we calculate precision based on how well each pixel is classified on an image, either as cell or background. $AUC - PR$ is also referred to as average precision (AP) because it averages precision values across all recall values. Mean average precision (mAP) is the average AP value across all classes. Since our system only contains the MDA-MB-231 cancer cell, we make no distinction between mAP and AP. We evaluate mAP at different intersection over union (IoU) thresholds, defined as

Recall = \frac{IntersectionArea}{UnionArea}, \quad (4.3)
where intersection area is the number of pixels in which the ground truth and prediction occupy the same area, and union is the total number of unique pixels occupied by both the ground truth and the prediction.

Figure 4.2: The PR curve and its two theoretical thresholds [25]
Chapter 5

Results and Discussion

5.1 Hyperparameter Optimization

Mask-RCNN comes with hyperparameters that control its behavior. Herein, we explore several hyperparameters and their effects on model performance.

5.1.1 Gradient Descent Parameters

The learning rate (LR) $\epsilon$ is the size of each SGD step and can be tuned between 0.0 and 1.0. It is known to be the most important hyperparameter because it defines the effective capacity of a model [13]. An overly large learning rate may inadvertently increase the training error, while an overly small learning rate may permanently stall SGD at a high training error. The former can be imagined as “skipping” over the cost function minima, while the latter is a less understood effect. This manifests as a U-shaped relationship between training error and the logarithmic learning rate (Figure 5.1), where error steadily decreases as the learning rate reaches its optimal value and sharply increases when it goes above it [13].

Figure 5.1: The U-shaped relationship between training error and logarithmic learning rate, where the error steadily decreases and shoots up above the optimal value [13]
Recurrent networks such as Mask R-CNN, which are highly nonlinear functions, tend to have very large or small gradients during SGD. Since the gradient indicates the steepest descent within an infinitesimal region of the parameter space, one SGD step in a large gradient may cause the parameter to “overstep” into a region where the cost function curves back up [13]. In this case, rather than tuning the learning rate, one can perform a clipping of the gradient norm

\begin{align}
    \text{if} & \quad ||g|| > \nu \\
    g & \leftarrow \frac{g\nu}{||g||},
\end{align}

where \( \nu \) is the gradient clip norm (GCN) and \( g \) is the parameter gradient. The GCN can therefore prevent exploding gradients while guaranteeing that the SGD step is still in the direction of the gradient.

Figure 5.2 shows the training and validation loss at various LR and GCN values. We aim for values that minimize the loss functions.
Figure 5.2: Training and validation losses for combinations of learning rate (LR) and gradient clipping norm (GCN) values. The backbone is resnet101.

The learning rate $\epsilon = 0.001$ consistently outperforms $\epsilon = 0.0001, 0.01$ while $\nu = 10$ generally outperforms $\nu = 5$. The validation loss experiences local peaks and valleys in $\epsilon = 0.01$, suggesting that the SGD step overshoots parameter values; as expected, this effect is less apparent with $\nu = 10$ than with $\nu = 5$.

A problem in SGD is choosing between undesirable optimums. For example, take the ellipsoid objective function with parameters $x_1$ and $x_2$

$$f(x) = 0.1x_1^2 + 2x_2^2.$$ (5.3)
The gradient in the $x_2$ direction is much higher by design. By construction, SGD will cause $x_2$ to oscillate around its minimum while $x_1$ progresses slowly towards it. During SGD, a small learning rate ensures the solution does not diverge in the $x_2$ but causes a slow convergence in the $x_1$ direction. A large learning rate converges rapidly in the $x_1$ but overshoots and diverges in $x_2$. See Figure 5.3 for an illustration of how both small and large learning rates compromise one parameter [26].

Figure 5.3: Trade off between small and large learning rates for parameters in Equation 5.6 [26].

To solve this problem, we introduce velocity and momentum terms in the update equation

$$v_t \leftarrow \beta v_{t-1} + g_{t,t-1}, \quad (5.4)$$

where $g_{t,t-1}$ is the parameter gradient during the time step $t - 1$ to $t$, $\beta$ is the learning momentum, and $v_{t-1}$ and $v_t$ are velocities. The parameter update equation then becomes

$$\theta_t \leftarrow \theta_{t-1} - \epsilon v_t. \quad (5.5)$$

$v_t$ is essentially an average of all previous gradients weighted by $\beta$:

$$v_t = g_{t,t-1} + \beta g_{t-1,t-2} + \beta^2 g_{t-2,t-3} + \ldots = \sum_{\tau} \beta^{t-\tau} g_{t-\tau,t-\tau-1}, \quad (5.6)$$

such that large $\beta$ is a ”long” memory of past gradients, a small $\beta$ is a ”short” memory. For Equation 5.6, an average of past gradients can increase the step size in the $x_1$ direction by considering early, large steps during later, smaller steps as $x_2$ reaches its minimum. It also decreases the step size in the $x_2$ because oscillations along this axis cancel each other out, thus preventing divergence.
We train Mask-RCNN for learning momentum (LM) values between $\beta = 0.7$ and $\beta = 1.0$. Figure 5.4 shows the training and validation loss at each value.

![Training and Validation Loss](image)

(a) Training Loss  
(b) Validation Loss

Figure 5.4: Training and validation losses for learning momentum values at $\epsilon = 0.001$ and $\nu = 10$. The backbone is resnet101.

We can see that the loss decays the fastest at $\beta = 0.9$. Finding $\epsilon = 0.001$, $\nu = 10$ and $\beta = 0.9$ provides a basis to further explore other hyperparameters outside of SGD.

### 5.1.2 Non-Max Suppression

Non-max suppression (NMS) is a threshold value for the confidence score of region proposals, below which the proposal is removed. We consider NMS values of 0.5 and
0.7 and find that NMS = 0.7 detects both elongated and rounded cells with higher confidence than NMS = 0.5 (Figures 5.5, 5.6). Though the full range of NMS values can be tested qualitatively to find the optimal value, we consider the general trend that higher NMS is better for our dataset, given common false negatives during inference (Figure 5.7).

(a) NMS = 0.5 with confidence interval of 0.909.
(b) NMS = 0.7 with confidence interval of 0.998.

Figure 5.5: Elongated cell detection at different NMS values.

(a) NMS = 0.5 with confidence intervals of 0.992, 0.911, 0.988 (top to bottom).
(b) NMS = 0.7 with confidence intervals of 1.000, 0.979, 0.999.

Figure 5.6: Rounded cell detection at different NMS values.
5.1.3 Anchor Ratios

Anchor ratios are the aspect ratios of anchor boxes in the RPN and should be tuned according to the generic aspect ratio of the object. We consider “long” anchor ratios of [0.25,1,4] and “short” anchor ratios of [0.5,1,2]. We find that long anchor ratios detect cells with marginally higher accuracy (Figure 5.7), agreeing with the observation that most cells in the data set are either rounded (aspect ratio $\approx 1$) or very elongated (aspect ratio $\approx 4$). Figure 5.8 additionally shows that the model with long anchors properly segments an elongated cell, while the model with short anchors captures it as two cells.

![Figure 5.7: Cell detection at different anchor ratios.](image1)

(a) Anchor Ratios = (0.25,1,4).  (b) Anchor Ratios = (0.5, 1,2).

Figure 5.7: Cell detection at different anchor ratios.

![Figure 5.8: Cell detection at different anchor ratios.](image2)

(a) Anchor Ratios = (0.25,1,4).  (b) Anchor Ratios = (0.5, 1,2).

Figure 5.8: Cell detection at different anchor ratios. Elongated cell is properly segmented with long anchors and improperly segmented with short anchors.
5.2 Model Performance

Figure 5.9 shows Mask R-CNN’s PR curve calculated with the validation set at an IoU threshold of 0.5. The validation set contains images at low cell density.

![Mask R-CNN’s PR curve](image)

Figure 5.9: Mask R-CNN’s PR curve.

For the particular dataset, Mask R-CNN maintains a precision of 1.0 up to a recall of 0.6, where it then drops steadily to 0. Compared to a baseline classifier (characterized by a horizontal line in PR space at \( \text{precision} = 0.5 \)), Mask R-CNN does remarkably well at returning relevant positive results. Since recall is a measure of the fraction of relevant predictions retrieved, Mask R-CNN is shown, at IoU = 0.5, to retrieve near-perfect predictions for up to 60% of all cells in low density images.

Figure 5.10 shows the mAP at different IoU thresholds calculated by taking the area under the PR curve.
For this particular dataset, Mask R-CNN maintains mAP$_{0.8}$ up to $IoU = 0.5$. A steady decrease is observed afterwards. This shows that though many predictions are considered relevant given a baseline threshold, they do not strictly adhere to ground truth and become irrelevant as that threshold increases. For example, Figure 5.5b shows a prediction that misses part of the cell’s protrusion. Furthermore, Figure 5.11 shows the difference between ground truth and prediction masks in the validation set. It’s clear that Mask R-CNN simplifies the geometry of the cell and biases its prediction towards brighter regions, thus skewing the shape of the mask.

![Figure 5.10: Mask R-CNN’s mAP at different IoU thresholds.](image)

5.3 Generalization to Higher Densities

The model is trained on a dataset of mostly low density, sparse cell images with a small fraction of high density organoid images. We use the trained model to per-
form inference on foreign (not involved in training) images of high density organoids. Figure 5.12 shows two examples where Mask R-CNN does not perform well.

Figure 5.12: Blurred cells are missed and cell-cell borders are misinterpreted at high cell densities.

With about 60%-70% of cells segmented, the vast majority of missed cells are exceedingly dim or blurred, which might be avoided with more training data of dim or blurred cells. Cell geometry is also oversimplified, and a lot of thinner protrusions were not captured. cell-to-cell borders are also an error source. First, many of the missed cells are aggregated into shapes that the model was not trained on, since low density images have a more even spatial distribution. When the separation becomes unclear, the model fails to distinguish them. Figure 5.13 shows better segmentations when tumors metastasize, prompting cells to leave the dense organoid into a more even distribution.
Figure 5.13: Cell segmentation improves when spatial distribution is more even during metastasis.

To continue this pattern, Figure 5.14 shows a very well segmented image that has very little cells left in the organoid.

As seen here, the Mask R-CNN model fails to pick up dim and blurred cells, but successfully detects bright, rounded cells to a reasonable degree.
Chapter 6

Conclusion

Deep learning has blossomed in the recent decade and has benefited cellular biophysics tremendously. Instance segmentation frameworks enable single cell detection in dense cell populations at a high resolution. In this thesis we have explored Mask R-CNN’s capabilities to detect cancer cells. To this end, we used a training set of 90 images with a small fraction of high density organoid images and high fraction of sparse cell images. We first explored and optimized an array of parameters available in the model. Three of these are related to stochastic gradient descent, which we optimized through the training and validation loss through the training cycle, namely learning rate, gradient clip norm, and learning momentum. We also optimized two region proposal network parameters, namely non-max suppression and anchor ratios, using inference. We then applied performance metrics to the model and found that Mask R-CNN is reasonably good at maintaining high mAP (>80%) at baseline IoU values and, in general, do not return false positives. However, it oversimplifies the cell geometry compared to the ground truth and fails to detect some dim and blurred cells, both of which were reflected in higher density organoid images. Additionally, because Mask R-CNN was not trained with images of cell-cell contact, metastasizing tumors causing cells to leave the center of the organoid gave a more even spatial distribution and resulted in less false negatives. To conclude, given a small dataset, Mask R-CNN performs decent instance segmentation at both low and high cell densities. It’s pretty clear that if a more robust model is desired, more training data of elongated and blurred cells, as well as physically touching cells, are needed. Our results suggest a much more robust detector can be achieved with improvements such as image augmentations and a larger dataset. In the future, statistical inference or CNN architectures can be used to track cancer cells in microscopy videos as the next step to building an experimental pipeline.
Bibliography


